Localization, motion planning and control of mobile robots in intelligent spaces

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING
SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Mišel Brezak

LOCALIZATION, MOTION PLANNING AND CONTROL OF MOBILE ROBOTS IN INTELLIGENT SPACES

LOKALIZACIJA, PLANIRANJE GIBANJA I UPRAVLJANJE MOBILNIM ROBOTIMA U INTELIGENTNIM PROSTORIMA

DOCTORAL THESIS

DOKTORSKA DISERTACIJA

Zagreb, 2010

DOCTORAL THESIS is written at the Department of Control and Computer Engineering, Faculty of Electrical Engineering and Computing, University of Zagreb

Supervisor: Professor Ivan Petrović

DOCTORAL THESIS has 229 pages. Dissertation No.

The dissertation evaluation committee:

1. Professor Sven Lončarić, University of Zagreb, Faculty of Electrical Engineering and Computing
2. Professor Ivan Petrović, University of Zagreb, Faculty of Electrical Engineering and Computing
3. Professor Bojan Jerbić, University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture

The dissertation defense committee:

1. Professor Sven Lončarić, University of Zagreb, Faculty of Electrical Engineering and Computing
2. Professor Ivan Petrović, University of Zagreb, Faculty of Electrical Engineering and Computing
3. Professor Joško Deur, University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture
4. Professor Nedjeljko Perić, University of Zagreb, Faculty of Electrical Engineering and Computing
5. Professor Vesna Županović, University of Zagreb, Faculty of Electrical Engineering and Computing

Date of dissertation defense: 26th March, 2010

Predgovor i zahvala

Kako bih došao do cilja, a to je izrada ove disertacije, moji roboti i ja morali smo zaobići mnoge prepreke, kako one geometrijske u radnom prostoru robota, tako i one u stvarnom životu, mnogo teže savladive. Ponekad su se prepreke činile toliko visoke da je cilj izgledao nedostižan. Strah od nepoznatog davao mi je osjećaj da ako i uspijem zaobići brdo ispred sebe, možda me iza čeka još veća planina.

Iako sam se u disertaciji odlučio za determinističku metodu planiranja gibanja do cilja, pisanje ove disertacije bilo je mnogo manje determinističko i u mnogočemu sličnije planiranju gibanja jednom mnogo manje determinističkom metodom – metodom polja potencijala. Pritom sam upadao u "klopke" lokalnog minimuma svojstvene upravo toj metodi, često se iz njih izvlačivši upravo kao što je spomenuto u odlomku o poljima potencijala: slučajnim hodom – pristupom koji ne garantira mnogo više nego da ćete se u nekom konačnom vremenu *možda* izvući iz klopke. Pritom zahvaljujem svim svojim "atraktivnim potencijalima" koji su me vukli ka cilju, a ponajviše svojem mentoru prof. Ivanu Petroviću koji mi je dao mnogo korisnih savjeta. No, po svemu sudeći, najteži lokalni minimum kojeg je trebalo svladati bio je upravo u meni samom.

Posebno zahvaljujem svojoj obitelji na potpori. Također zahvaljujem svojim kolegama na mnogo lijepih trenutaka provedenih zajedno. Zahvaljujem studentima koji su implementirali neke od algoritama u disertaciji i svima ostalima za koje nemam mjesta da ih pojedinačno navedem, a koji su na bilo koji način doprinijeli nastanku ove disertacije.

Potrebno je još spomenuti da se čitajući ovu disertaciju ne stječe pravi dojam kako je sve to nastajalo. A sve je inspirirano robotskim nogometom te je gotovo svaki algoritam opisan u disertaciji testiran na platformi za robotski nogomet. Pritom mi cilj nije bio izgraditi sustav koji će raditi samo za robotski nogomet, već za čim širi spektar aplikacija. Tako se došlo na ideju da bi razvijeni algoritmi bili idealni za primjenu u inteligentnim prostorima, koji svojom mrežom raspodijeljenih senzora pružaju infrastrukturu sličnu onoj koja se koristi u robotskom nogometu. Kako se u praksi uvijek mora načiniti kompromis između općenitosti i usmjerenosti ka specifičnoj primjeni, nastojanje da se algoritme učini fleksibilnim i univerzalnijim, kao i čitav niz različitih znanstvenih područja u koja je trebalo savladati kako bi se dobio funkcionalan sustav, značajno je produljilo razvoj cijelog sustava.

Mišel Brezak,

Pregrada, 9. lipnja 2010.

Contents

Chapter 1

Introduction

1.1 Motivation and Scope

Recent advances in field of sensor networks, mobile robotics and artificial intelligence enable us to realize today what previously could be seen only in science fiction (SF) movies: Spaces do not interact with the humans only passively, but can also support humans in a true physical way. The possibilities are numerous, e.g. load delivery, visitor guidance etc. Such a space is called an *Intelligent Space* (*iSpace*). To enable advanced services, iSpace must be capable of monitoring events in it, communicating with objects in it, making decisions in order to provide services, and act based on these decisions.

In general case an iSpace consists of sensors, processors (computers), actuators and communication devices. The sensors (e.g. cameras, microphones, ultrasound or laser beacons) have the role of identification and tracking the objects in the space and taking orders from space users. The iSpace interacts with objects in it using actuators (e.g. mobile robots) that provide various advanced services to the space users. The processors act as the brain of the system, controlling the actuators with the purpose of performing various tasks, using information obtained from sensors as a feedback. Actuators, processors and sensors, connected with communication devices, together make a distributed system, which is the core of the intelligent space.

As the whole space is intelligent, it is able to provide various services to its clients (i.e. users), where clients are primarily humans. However, any other device can be a client of the intelligent space. For example, robots are utilized to provide physical services to humans as physical agents, but the robot as well as the human is supported by Intelligent Space if necessary. E.g. if a robot is lacking necessary sensors to be able to navigate in the space, it is treated as a client, and the required information is provided to the robot by the intelligent space.

Making a space intelligent is especially useful when multiple mobile robots navigate within the same structured space (e.g. a flat, warehouse, airport, supermarket, etc.). In this case it may be reasonable to install sensors and computers, that are commonly installed in the robot, to the space instead. In this way the system price can be significantly reduced, because robots are enabled to contain only cheap sensors for basic collision avoidance (e.g. bumpers and sonars). The system performances can be increased, too, because more powerful computer systems can be used resulting in possible higher sampling rate and thus lower delay and reaction time and higher speeds of the robots.

Moreover, this approach can have multiple advantages regarding three most common problems that arise in mobile robots navigation: robot localization, map building and robot motion planning. First, robot localization gets simpler because sensors can determine robot pose directly in global coordinate frame. Consequently, the environment map is no more required for robot localization. Second, the map building process gets also simpler because the robot location is always known and there is no problem of simultaneous localization and map building. Besides, distributed sensors, as well as robot onboard sensors, enable easy and continuous inclusion of new objects in the environment map, so the map is always up-to-date. Third, robot motion planning is also much easier and more reliable, since the actual picture of the whole space and all obstacles in it is always available. Moreover, it is possible to predict the changes in the space and crate a predictive map, which can then be used for robot optimal path planning in spite of moving obstacles. The most appropriate sensors to be installed in iSpace and used for navigation of multiple mobile robots in it are cameras. Such systems are known as *global vision systems*.

The research of intelligent spaces is prevalently multidisciplinary—it borrows many ideas from other research fields, e.g. sensor fields, computer vision, mobile robotics. Under so broad scale of possible research directions, the focus of this thesis is set to developing the capability of the space to fully utilize mobile robots, with the emphasis on development of low-level algorithms that can benefit by using intelligent space advantages, such as precise localization and mapping and increased computational power.

To actually provide useful services using mobile robots as agents, apart from other tasks, intelligent space must be able to locate the robots, plan their motion, and control them in order to execute the planned motion. Consequently, a research is conducted in two directions. First, a method for fast and precise mobile robot localization using distributed cameras is developed. Second, a fast and flexible robot motion planning method appropriate for intelligent space application is developed. Here the idea was to maximally utilize intelligent space's potential of accurate robot localization in order to achieve precise and deterministic planning or robot motion. Besides, the developed algorithms are organized and designed with maximum modularity in mind, which is the reason why the developed motion planning algorithms do not explicitly rely on distributed architecture of the iSpace, but can be used generally for navigation of autonomous mobile robots.

1.2 Thesis Overview

The rest of the thesis is organized into 7 chapters and a conclusion as follows.

Chapter 2

An overview of the Intelligent Space concept is given. The most important examples of the similar researches are given, followed by description of how to utilize mobile robots as agents of the intelligent space. A description of sensors appropriate for the intended application is given, as well as some applications in the intelligent spaces.

Chapter 3

A novel algorithm that enables precise and accurate robot localization within intelligent spaces using global vision is described. An overview of the physical system design used to build global vision system is given. Vision algorithms for robot detection and tracking are presented. Finally, a detailed experimental analysis of the developed algorithms is conducted.

Chapter 4

In this chapter a general motion planning problem is defined, so that it serves as an introduction to later chapters which provide more details about the topic. The direct and decoupled approaches to motion planning are introduced. Finally, selection of the methods used in this thesis is discussed.

Chapter 5

Path planning while avoiding obstacles has long time been the main goal within motion planning research community, while in this thesis it is part of the first stage of the overall motion planning algorithm. Therefore, this chapter is concerned with methods used to plan obstacle-free paths between two robot configurations. A novel path-planning algorithm is introduced that enables fast path-replanning.

Chapter 6

Common path-planning algorithms usually give obstacles-free path, but with no or very little concern about path feasibility or optimality. Thus, it is described how to transform a path so that the robot can track it faster. Smoothing algorithm is given that can transform a path that consists of straight line segments to continuous curvature path, which is essential for fast robot motion. The algorithm is intended for differential drive robots and uses clothoid curves as primitives for path smoothing, which have inherent property that their curvature changes proportionally with distance traveled along the curve.

Chapter 7

This chapter is concerned with problem of finding an optimal velocity profile along the planned path so that the path is traversed in shortest time. A dynamic model of the differential drive robot is developed and used to derive actuator limits model that is required by time-optimal trajectory planning.

Chapter 8

Control algorithms for generating commands that force the robot to track the planned trajectory are described. Three types of trajectory tracking controllers are described and experimentally compared.

Chapter 9

This chapter brings conclusions and summary of the scientific contributions. Some ideas for future work are given as well.

Chapter 2

Mobile Robots in Intelligent Spaces

This chapter brings an overview of the Intelligent Space concept. The most important examples of the similar researches are given, followed by description of how to utilize mobile robots as agents of the intelligent space. A description of sensors appropriate for the intended application is given, as well as description of some applications in the intelligent spaces.

2.1 Intelligent Spaces

Although the idea of making spaces intelligent in order to better suit human users is very old, systematic researches are relatively new and can be tracked back to 1990's. The concept was initially called *ubiquitous computing* described by Weiser [155], where people should be able to use computational resources everywhere in the environment. This work introduced new infrastructure and paradigms of interaction inspired by need of widespread access to information and computational capabilities. Wang and Garlan [153] introduced later *task-driven computing* which enables users to interact with computer systems in terms of high-level tasks. In this way an user does not need to learn low-level configuration details, which results with reduced attention and knowledge requirements of mobile users in a pervasive computing environment.

With the recognition and awareness of people in environments, the work on ubiquitous computing later evolved to research of *ambient intelligence*. Here the focus was moved toward intelligent environment research in order to provide human-oriented services. Several famous corporation laboratories conducted their own research devoted to various related fields. So the FX Palo Alto laboratory of Xerox started Smart Media Space project [39], where in network-based environments media interacts with the environment to promote knowledge sharing. In the project called EasyLiving Technologies [28] at Microsoft Research human tracking technology was employed in order to guess the intent of users in the space

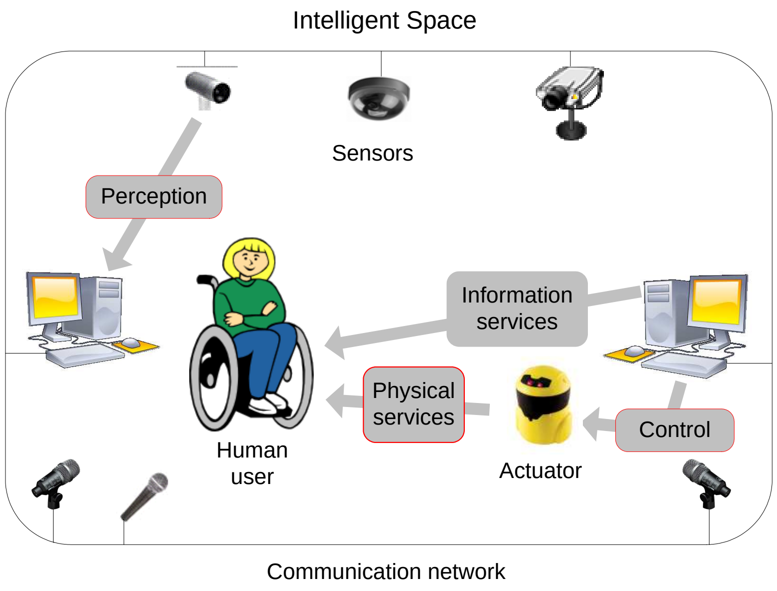


Figure 2.1. *The Intelligent Space*

and automate everyday tasks. In the Interactive Workspace Project [66] of HP the applications of ubiquitous embedded sensors and information displays were investigated. In yet another work, the Visualization Space research [94] at IBM, a visual computing system was introduced and used as a testbed for a deviceless multimodal user interface.

An important research on ambient intelligence was conducted at MIT, where the concepts such as Smart Room, Smart Space, and Intelligent Room were introduced. The research is now active under name Oxygen which aims at the development of intelligent environments based on human-centered computation. The goal is to make the computation "freely available everywhere, like batteries and power sockets, or oxygen in the air we breathe" [49].

With similar motivation, Hashimoto et al. introduced in 1996 a concept called *Intelligent Space* (*iSpace*) (see e.g. [6, 56]). The novelty of the Intelligent Spaces over existing concepts was incorporation of physical services for humans, as opposed to only informational services. To achieve this, the space could be equipped with a range of actuated devices such as mobile robots. The overall scheme of the intelligent space is shown in Figure 2.1. To accomplish its task, the intelligent space must contain the following:

- distributed sensors;
- computers;
- actuators;
- communication network.

The main role of sensors is to enable perception of what is happening in the space. Thus sensors provide various information to the control logic of the intelligent space, and implicitly also to the user (through information services). The obtained information can give answers to questions such as who the current user is, what his intentions and habits are, what commands does he give to the space, etc. The sensors can also measure various environmental parameters if necessary (temperature, humidity, light). Typically, multiple sensors are distributed so that the whole space can be perceived. Cameras, microphone arrays and ultrasound beacons are mostly used nowadays.

The computers build a brain of the space and run various software components with the main goal of providing services. There are typically three types of tasks performed by the software [91].

1. *Sensor and actuator servers.* Those are specialized modules for the data preprocessing in order to derive relevant information from the sensors and offer this information on the network.

2. *Intermediate processing.* Those modules connect as clients to one or more sensor servers and process their data in order to perform tasks such as sensor fusion, temporal integration and model building. The intermediate results are again offered on the network.

3. *Application processes.* Here the actual applications of the space are implemented. Those can include generation of information understandable to the user or control of mobile robots using information obtained from sensors as a feedback.

The software components can be assigned to multiple distributed computers connected by the network, or onboard computers embedded in robots, and an actual distribution is application-dependent. Typically, the computers that run sensor servers are incorporated together with sensor devices and communication circuitry. In this case they are referred as *Distributed Intelligent Network Devices.*

Specific tasks are accomplished by utilizing actuators (sometimes called agents). In order to be able to provide informational services, the space uses passive devices such as monitors, pointing devices, projectors, speakers and other interfaces. However, to be able to provide physical support active devices must be utilized, such as embedded motors, active switches, robot arms, and especially mobile robots.

Finally, the communication network is used to connect sensors, computers and actuators. Multiple types of networking technologies can be employed and combined depending on type of communication required. e.g. LAN networks can be used for connecting computer workstations, wireless LAN for communication with mobile robots, and CAN networks for real-time sensor communication. In general, the intelligent space will benefit greatly with the faster network.

2.2 Introducing Mobile Robots

Due to their mobility, the mobile robots are the most appropriate physical agents in the intelligent spaces. So mobile robots can be used for various services such as carrying, delivering, cleaning, guiding. Another possible application of mobile robots is the interaction with humans (see e.g. [137]).

There are mutual benefits that the intelligent space and mobile robots can provide each other. Through interaction with the intelligent space, the space can now be employed as an powerful interpreter between human and robot, in this way avoiding the necessity of implementing the human interface on each robot. A human operator can in this way control many systems with only a single human interface. The space also plans high-level actions to be executed by robots in order to provide useful services.

Regarding robot control, the system performances can be increased because more powerful computer systems installed in space can be used resulting in possible higher sampling rate and thus lower delay and reaction time and higher speeds of the robots. Moreover, intelligent space integration can have multiple benefits regarding three most common problems that arise in mobile robots navigation: robot localization, map building and robot motion planning.

1. Robot localization gets simpler because distributed sensors can determine robot pose directly in global coordinate frame, and consequently the environment map is no more required for robot localization purpose.

2. The map building process (map is required for robot motion planning) gets also simpler because the robot location is always known and consequently there is no problem of simultaneous localization and map building.

3. Robot motion planning becomes also easier and more reliable, since the actual picture of the whole space and all obstacles in it is always available. For optimal planning of robot motion in spite of moving obstacles, it is beneficial to predict the changes in the space and crate a predictive map, which again is being made possible by the intelligent space.

2.3 Sensing in Intelligent Spaces

Sensors in the intelligent space can be divided to distributed (at fixed locations in the space) and onboard (embedded in the mobile robots). Both have advantages that make them particularly appropriate for certain applications. So distributed sensors are better for mobile robot localization, and onboard sensors enable reliable mapping of the environment. Therefore it is beneficial to use both types of sensors if possible. Systems of distributed sensors are usually referred as tracking systems and localization is in this case referred as tracking (e.g. human tracking,

Table 2.1. *Comparison of different indoor sensors.*

METHOD	ADVANTAGE	LIMITATION	MOUNT
Cameras	Cheap; Easy to install	Hard to achieve robustness	Distrib. or onboard
Range cameras	Expensive	Low resolution	Distrib. or onboard
Sonars	Cheap	Occlusions; Low angular resolution	Distrib. or onboard
Laser range finders	Accurate; Relatively low price	Occlusions	Distrib. or onboard
Ultrasound, indoor GPS, etc.	Accurate	Rather expensive	Distrib.; Objects must wear tags
RFID	Cheap	Not accurate	Distrib.; Objects must wear tags
Inertial / Encoders	Direct measurement of speed, orientation	Accumulating error	Onboard
Microphone arrays	Cheap	Not accurate	Distrib. or onboard

robot tracking). The other special case is when only onboard sensors are used to estimate robot's position—this is often called self-localization.

In this thesis distributed cameras are used for robot tracking, but a whole range of other sensors can be combined for other purposes, such as mapping or human tracking. Detailed discussion about sensing in intelligent spaces and a fusion of distributed and onboard sensors can be found in [27], and here only a short overview is given. Commonly used sensors for applications in intelligent spaces and their characteristics are given in Table 2.1, and further described in the sequel.

Cameras

Of all sensors, cameras provide the greatest amount of data. With appropriate computer vision techniques many useful information can be extracted, such as color or shape. However, this could easily turn into a disadvantage: it is difficult to robustly process so many information and high processing power is required. Problems come from the complexity of real environments, effects of changing illumination (the color looks different under different illumination), small resolution if cameras are too far away, the need for accurate calibration, etc.

Nevertheless, specialized algorithms for tracking particular objects (e.g. robots) using artificial landmarks can circumference this problem. In this way precise and robust robot tracking can be obtained as described in Chapter 3. Cameras are also appropriate for human tracking in intelligent spaces [6]. Special camera systems, such as infrared cameras, can be beneficial in some circumstances. e.g. with infrared cameras human tracking can become a rather easy job. Different aspects of visual tracking of objects are described by Hu et al. [60].

Although cameras are relatively cheap today, additional costs come from requirements for appropriate image processing hardware and adequate lighting, and low-cost cameras may not be the best choice if one wants to achieve robust and precise tracking. Special types of camera, such as infrared cameras, are very expensive compared to common cameras.

Range Cameras

As the image obtained from camera has only two dimensions, to obtain a full 3D observation distance information is required (so called range imaging). This may be achieved by combining information from two (stereo cameras) or more cameras. However this technique is very computationally intensive as complex correlation algorithms have to be implemented and it is hard to achieve acceptable robustness. Another possibility is to combine camera and an additional range sensor, such as laser range finder.

Recently, with advance of semiconductor technology time-of-flight (TOF) cameras [158] have become available. Their principle is similar to that of laser range finders but with the advantage that whole scene is captured at once. Those cameras have illumination unit based on laser or LED array, that emits pulsed light which is normally in infrared spectrum range to make the illumination unobtrusive. The light is reflected from objects and picked up by the camera to calculate the distances to objects by measuring time-of-flight.

The advantage of TOF cameras are their compactness (as opposed to stereo cameras) and they have no mechanical parts (in contrast to laser range finders). Moreover, it is very easy to detect objects based on provided distance information. Currently, these cameras are expensive and give rather noisy measurements. However, this is expected to improve in the future, so that many researchers agree that time-of-flight cameras will become the dominant sensors in the field of robotics, just as laser range finders are nowadays.

Sonars

Sonar (“SOund NAvigation and Ranging”) measures propagation time of ultrasound wave from sensor to some object or surface in the space, and back. A sensor contains both ultrasound transmitter and receiver. Knowing the speed of sound, the distance to an object can be computed.

It can be used both as onboard or distributed sensor. However, sonars are rarely used as distributed sensors, but ultrasound based trackers are used instead (they use different arrangement of transmitter and receiver, as described later). Sonars are cheap, relatively accurate and capable of detecting most materials typically present in indoor spaces. Problems arise when specular, weak or multiple reflections occur. Also, sonars have bad angular resolution and short range (usually 3 m) and limitations connected with low speed of sound. e.g. to measure a distance of 3 m, a time of 17.6 ms is required. Multiple circularly-placed onboard sonars are usually mounted on the robot. With this arrangement a 2-D distance-scan of the 360° space around the robot is obtained.

Laser range finders

Laser range finders (LRF; sometimes also called LIDAR – Laser Imaging Detection and Ranging) are devices that determine the distance to an object by measuring time-of-flight of the reflected laser rays. LRF-s achieve significantly finer angular resolution and shorter measurement time compared to sonars, but have problem detecting opaque surfaces and mirrors. Most common types contain mechanical system that rotates the sensor resulting with a 2-D distance cross-scan of the space.

Due to their relatively low price, LRF-s are today frequently used as onboard sensors in mobile robotics, especially for applications such as self-localization, simultaneous localization and mapping (SLAM), and sometimes for detection and tracking of humans in the vicinity of robots. Distributed LRF-s are seldom found in literature, but Brščić [27] has found them useful in intelligent spaces for human and robot tracking. Distributed LRF-s can also be used for mapping, although onboard LRF-s are much better for this purpose.

Ultrasound and Indoor GPS

Ultrasound trackers work in different way than sonars. Namely, multiple transmitters (or receivers) are mounted into space at known locations, while tracked object wears a receiver (or transmitter). Some examples of tracking systems based on ultrasound are Active Bat, Cricket [59] and the Zone Positioning System [114]. Apart from ultrasound, other signal types can be employed, such as radio waves or light. These systems are sometimes called "indoor GPS" inspired by their similarity to the GPS system. A typical example are the so-called "pseudolites" [74], which use a signal similar to the GPS signal. In this way the same GPS receiver can be used for both outside and inside localization.

The advantages of these systems are relatively high accuracy, especially for systems based on electromagnetic waves, and low computational requirements for sensor data processing. The drawbacks are rather high price (usually proportional with accuracy), need for installation of multiple devices in the space and cumbersome calibration process. Furthermore, in human-tracking applications users have to carry tags in order to be tracked which makes this method impractical.

Radio-Frequency Identification (RFID) Systems

Recently, radio frequency identification (RFID) devices are being extensively used for various identification purposes. Such identification is based on RFID tag device applied to or incorporated into an object or person for the purpose of identification and tracking using radio waves. RFID tags are very cheap and small, so they are becoming increasingly prevalent in our everyday life [157], e.g. for product tracking, transportation and logistics, human identification, but they are also utilized for localization purposes [76].

It must be emphasized that RFID tags are actually not sensors, but more like landmarks for RFID readers that achieve localization only within a certain area, whereas accurate position cannot be determined. This is what makes RFID tags inappropriate for applications such as mobile robot control. Nevertheless, easy installation and small price makes them attractive, so that their increased application combined with other tracking methods is expected in the future.

Encoders, Inertial Measurement Units

Wheel encoders and inertial measurement units, such as accelerometers and gyroscopes, are commonly used as auxiliary sensors in robot localization. The position estimation based on these sensors is called dead reckoning [17]. Usually some kind of integration of sensor data must be used to estimate position, or more precisely, relative change of the position. Because of that dead reckoning suffers of error accumulation over time, caused e.g. by measurement noise, wheel slippage, inaccurate robot model etc. Therefore only a short-term estimation can be achieved reliably, so that these sensors are appropriate for interpolation between measurements from other, usually slower sensors. Of course, encoders and inertial sensors can only be used in onboard option.

Microphone Arrays

Using microphone array a position of the sound source can be estimated by recording the sound with spatially separated microphones. Most common methods are based on measuring the phase shift between signals acquired by the microphones, which in turn is mostly used for localization of a human speaker. Microphone array can be mounted both distributed and onboard. Distributed microphone array can also be used for localization of the robot or other sound sources.

The problem with sound-based localization is that it is not easy to determine location when multiple sources of sound are present, and obtained location may not be very accurate due to sound reflection, noise, or other disturbances. However, the advantages are low price and possibility to use microphones for voice communication with the robot or the intelligent space.

2.3.1 Mobile Robot Localization

The most appropriate sensors to be installed in the space and used for robot tracking are cameras, resulting with so called *global vision systems* [129, 24]. With use of advanced image processing techniques robot pose can be determined with much higher precision than by relying solely on robot's onboard sensors. As general visual-based pattern recognition suffers from weak robustness, usually some kind of color-based markers placed on the robot body are utilized. Such an algorithm that enables fast and precise localization of multiple mobile robots is developed in this work and is described in details in Chapter 3.

Besides visual tracking, in our lab several other localization solutions have been implemented. Regarding distributed sensors, a pose tracking algorithm has been developed that ensures accurate mobile robot localization based on active radiofrequency and ultrasound beacons and passive listeners (Cricket system), to support pervasive indoor location determination [80]. Pose estimation is done by Extended Kalman Filter with variable sampling time because of stochastic nature of measurement arrival times. Experimental verification yielded poses error approx. 1 ± 2.5 cm and 1 ± 2 °.

On the other side, self-localization and SLAM algorithms are needed when certain areas are not covered by the distributed sensors. Here we take the classical approach to model-based localization, which consists of matching the local representation of the environment built from sensor information with the global model map. Our algorithm for robot pose tracking is based on the Unscented Kalman Filter (UKF), which is computationally simple and provides low uncertainty of the pose [64]. Since UKF cannot solve the problem of global localization and the problem of kidnapped robots, a multicriteria localization algorithm based on particle filter has also been developed. Furthermore, our SLAM solution is based on a view-based representation of the environment and the current and key past robot poses estimation by using the Extended Information Filter (EIF) [79].

If multiple sensors are utilized, there appears a problem of combining multiple sensor measurements in order to obtain the best estimate. Onboard sensor fusion is well covered in literature, but fusion of both onboard and distributed sensors is yet an open area of research. An approach to this problem is given in [27], where use of Covariance Intersection method is proposed and several fusion architectures are introduced.

2.3.2 Human Tracking

To support humans in the space, the iSpace tracks humans. Various methods exist for human tracking, and most are based on background subtraction [65]. Afterwards, features of a human such as the head, hands, feet, eyes, etc., can be located. Using the images of several cameras, the 3D position of the human can be obtained.

In our lab a human tracking algorithm was implemented which is based on background substraction, motion detection and region and shape tracking [109]. It is still under development as the problems with false detections caused by shadows still exist.

Besides that, a general probabilistic algorithm for multiple moving objects tracking using onboard laser range finder is developed [67], that can also be used to specifically track humans in combination with other sensors.

Finally, we have been working on the auditory system for mobile robots, which detects and tracks the sound source [100], but also can recognize who is speaking and establish a dialog with him/her.

2.4 Mapping in Intelligent Spaces

A map of the environment is usually required for fast and safe navigation of a mobile robot. In this way a map serves as an abstraction level between sensors and motion planning modules. In intelligent spaces, distributed sensors enable easy and continuous inclusion of new objects in the environment map, so the map is always up-to-date. The space can then provide information which the robot can lack because it is missing adequate sensors. This includes obstacles not visible to robot, either because of occlusion or limited sensor range. Humans can also be tracked by the space and included in the map.

However, using only distributed sensors for mapping may not be sufficient. e.g. in case of global vision some objects may not be detected due to occlusion or limited sensor range and complex image processing is required in order to differentiate objects in the space from image background. Besides that, it is hard to calibrate the distributed sensors completely accurately so some residual error always exists and sensors usually have a systematic error in the distance measurement which cannot be corrected by taking further measurements since the measurements of static sensors are not informative (the background of the sensor scan does not change no matter how many scans are taken and further measurements give no additional information).

Nevertheless, intelligent spaces enable significant simplification of the mapping process when both distributed and onboard sensors are used [27]. Robot onboard sensors can provide information about distance to nearest obstacles which is required for map building, and is not easy to obtain by using distributed sensors only. e.g. cameras installed into space and robots equipped with laser range sensors is a very powerful combination that enables both, efficient robot localization and mapping of the space. In this way robot onboard sensors bring benefits not only to the robot, but also to the whole space. Robot itself becomes a mobile sensor of the intelligent space and informations of both distributed and onboard sensors can be combined together.

Currently in our lab a complete solution for exploration and mapping of unknown arbitrary polygonal environments is developed, which assumes an onboard range sensor of finite angular resolution and thus provides sampled version of the visibility polygon instead of imposing further restrictions on the environment [4].

2.5 Motion Planning in Intelligent Spaces

The tasks of a mobile robot motion planning system are finding the path and guiding the robot to the goal position given by the user or by superimposed task planning and scheduling controller. The majority of existing planning algorithms produce a graph of possible paths to the goal and then a global path is found by a graph search algorithm. The path produced by such an algorithm is a straight line path with sharp turns which cannot be used directly due to robot's kinematic and dynamic constraints. Therefore a local planner is used that locally modifies the planned path at the same time ensuring obstacle avoidance by incorporating reactive behavior using some kind of sensor feedback. Such solution is also developed in our lab, where D* algorithm is used for path planning and dynamic window algorithm is used as local planner [132].

Due to use of local short-term planning, this approach cannot ensure long-term prediction of robot movement which may be too restrictive in some applications (e.g. coordinated multi-robot planning). However, in intelligent spaces a more deterministic behavior can be achieved, which is the goal of the motion-planning algorithm developed in this thesis.

2.6 High-level Applications

The ultimate goal of the intelligent space project is to accomplish an environment that comprehends human intentions and satisfies them. Such a system is hard to achieve, since a huge number of functions would need to be prepared, and human-like intelligence is required. Even though such a complete system cannot be achieved immediately, it is believed that a useful system can be achieved with current technology by proper system integration [56]. In this way, the intelligent spaces are expected to spread from laboratories, where they currently reside, to our homes and offices.

Some of the so far developed higher level applications in intelligent spaces include map building by looking at people [6], which uses the fact that mobile robots can navigate robustly without a precise geometrical model if some other way of localization is given and a topological map is supplied. This topological map can be build simply by looking at the movements of people in the room.

Next example is compressed human motion video and identification, where the intelligent space recognizes what a human is doing and separates the real video data into index images and actions with a time stamp. In this way a compressed human motion is obtained. Moreover, a human recognition function can be achieved by using visual information.

Another examples are mobile robot control for following a human [65] and internal state inference by motion tracking, where the space can detect if the human is e.g. confused with user interface by analyzing movements of her hands and head.

Although intelligent spaces are primarily oriented toward human servicing, it must be emphasized that not only human can be a client of the intelligent space—robot can be a client, too. In this way the space can be configured to provide services solely to robots. This concept is useful in applications such as delivery in factories or entertainment platforms such as robot soccer.

Chapter 3

Robot Localization Using Global Vision

This chapter presents a new global vision system for tracking of multiple mobile robots. To the best knowledge of the authors it outperforms all existing global vision systems with respect to measurement precision and accuracy, high speed and real time operation and reliable tracking of large (theoretically unlimited) number of robots under light intensity changes. The originality of the proposed system lies mainly in specially designed robot marks and robots' poses measuring directly in Bayer format image delivered by the camera. These two measures enable robust pose estimation of the robots with subpixel precision, while the significant simplification of the image processing algorithms ensures tracking of many robots with very high framerate. With algorithms running on a 3 GHz Athlon 64 processor 65 robots can be tracked at 80 fps. Moreover, in order to perform a thorough analysis of the system performances related to defined requirements, we propose a new experimental procedure that can serve as a benchmark for evaluation of other systems for the same purpose.

This chapter has been previously published as: M. Brezak, I. Petrović and E. Ivanjko. Robust and accurate global vision system for real time tracking of multiple mobile robots. *Robotics and Autonomous Systems*, 56:213–230, 2008 [24]. The early versions of the work have appeared as [20, 84, 26], while the extended abstract is published in [21]. The application of the algorithm for mobile robot odometry calibration is described in [63].

3.1 Introduction

A problem of mobile robot navigation is commonly solved using autonomous concept where all necessary sensors and computers are on-board. However, when multiple mobile robots navigate within the same structured space (e.g. a flat, warehouse, airport, supermarket, etc.) it may be reasonable to install sensors and computers in the space instead. In this way the system price can be significantly reduced, because robots are enabled to contain only cheap sensors for basic collision avoidance (e.g. bumpers and sonars), and also the system performances can be increased. The most appropriate sensors to be installed in a space and used for navigation of multiple mobile robots in it are cameras. Such systems are known as *global vision systems*.

The main task of a global vision system is first to detect robots in the image, and then to track their poses in the image. In order to be applicable such a system has to meet the following requirements:

(i) it must ensure high tracking accuracy and precision;

(ii) it must operate with high framerate in order to enable real time tracking of fast-moving robots, so computational complexity of the image processing algorithms must be as low as possible;

(iii) it must have ability to track large number of robots using images from one or more cameras, and finally

(iv) it must be highly robust to light intensity changes.

Therefore, one must carefully design the vision system and address a number of problems, concerning both physical system design, such as camera and hardware elements selection, artificial landmarks design etc., and software design, such as which visual cues to use for robot detection and tracking, which image processing algorithms to employ, etc. There are a number of existing approaches tackling those problems. For example, a global vision system with active cameras is used in [148] and with omnidirectional cameras in [108]. These two approaches are not suited for practical use due to high computational complexity and low tracking accuracy. Other attempts are based on fixed cameras and they have been mainly concerned with global vision intended for robot soccer. For example, in [85] an example of global vision system designed for soccer robots is described, but with limitation to small number of robots that algorithm can track, and with considerable computational complexity since the algorithm always makes global search of the image. In [139] local image search is used, but robot pose measuring is based only on color marks which cannot guarantee high measuring accuracy without using expensive cameras that do not require color interpolation. None of mentioned works specifically addresses the problems of precision and accuracy. In [31] a system is presented that can accurately track a broad class of robot patterns, but it still relies on color information only so that there remain possibilities for further improvements. Some more recent papers are especially concerned with problems of camera distortion and non-uniform illumination, for example [86], [45] and [54]. However, the assumptions that are made in these works make them applicable only in robot soccer applications.

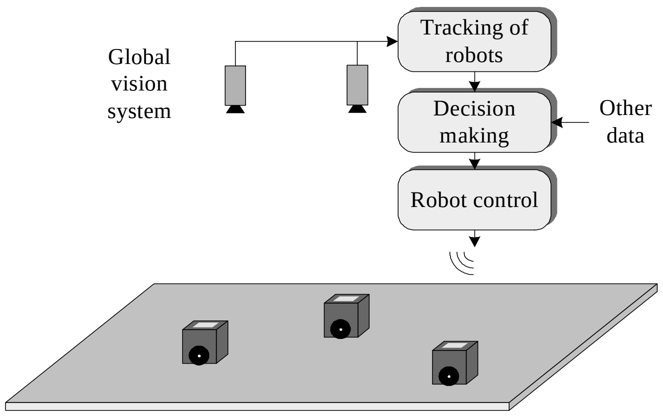


Figure 3.1. *System overview scheme*

In this work a new global vision system that fulfills all of the above enlisted requirements is proposed. To the best knowledge of the authors the proposed system outperforms all existing global vision systems with respect to at least first three out of four enlisted requirements. The originality of the proposed system lies mainly in specially designed robot marks and robots' poses measuring directly in Bayer format image. These two measures enable robust pose estimation of the robots with subpixel precision and significant simplification of the image processing algorithms ensuring many robots tracking with very high framerate. Moreover, in order to perform a thorough analysis of the system performances related to defined requirements, a new experimental procedure that can serve as a benchmark for evaluation of other systems of the same purpose is proposed.

3.2 Physical System Design

The system consists of multiple mobile robots moving in the area supervised by one or multiple distributed cameras fixed above the robots, which track robots' poses, and one or multiple distributed computers connected via a high speed communication bus, which execute image processing algorithms as well as decision making and robot control algorithms (Figure 3.1). Behavior of the whole system is highly dependent on the performances of the subsystem for robots' poses tracking, which are determined not only by the vision algorithm performances but also by the characteristics of used cameras and marks placed on the robots for their detection and tracking. While vision algorithm is described in the next section, camera selection and calibration as well as robot marks selection are described hereafter.

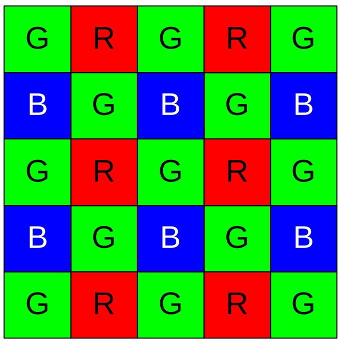


Figure 3.2. *Bayer color filter array*

3.2.1 Camera Selection and Calibration

Considering requirements imposed on the vision system, particularly requirement for high framerate, cameras with Bayer color filter arrays (Figure 3.2) [12] are selected. These cameras can deliver color image in standard VGA image resolution (640×480 pixels) with high framerate (80 fps) via standard IEEE 1394a bus and have acceptable price. Today are already available cameras that offer even higher framerates via IEEE 1394b or gigabit ethernet interfaces. Intelligent cameras would also be of great benefit since robot poses could be computed directly in the camera and therefore the transfer of the image to the host computer would become unnecessary reducing the need for high speed communication bus. Although high prices of intelligent cameras currently limits their application, in the future these cameras could become the prime choice for described system implementation.

To capture the entire region of interest with as few cameras as possible wide angle lens are commonly used, but at the cost of severe image distortions such as radial and tangential distortion. It is very important to correct those distortion effects in order to reach good measurement accuracy. In order to correct distortion effects, camera calibration procedure has to be performed for extracting distortion parameters. Many calibration techniques have been suggested, e.g. [19], [146], [159]. Currently Camera Calibration Toolbox for Matlab [19] is used for determining both intrinsic and extrinsic parameters of the camera using calibration grid. As the camera is fixed, calibration has to be done only once, and then calibration parameters are stored and further used. In the future, possibilities of calibration procedure automation will also be considered, so that system setup would be further simplified.

3.2.2 Design of Robot Marks

The main tasks of a vision algorithm are reliable detection of the robots in the image and measuring their poses. This means that vision system has to be capable of distinguishing image elements that represent robots from elements that represent other objects or background. Here different approaches can be applied, but the most commonly used one is the background subtraction [147]. It performs subtraction of acquired image and previously stored background image to differentiate background pixels, which should be ignored, from foreground pixels, which should be processed for identification or tracking. Although this method can be very flexible since it can detect not only robots, but also humans or other moving objects, it suffers from several drawbacks such as if a background object moves, it can be recognized as a foreground object or if a foreground object has the similar color as a background one, it is very difficult to detect it. Other segmentation methods, like gray scale or color based thresholding methods, edge-based methods, or region-based methods could also be employed, but generally, regardless of applied method it is very difficult to design an universal system that can reliably detect any kind of robot no matter of its size, color or shape, as it would require very complex algorithms and would result with poor tracking accuracy.

To overcome the problem of detection generality, customized patterns specifically designed for detection are used. Although this prevents such vision system from being applied in every environment, in many situations this is not a major limitation, and can save substantial amount of computation time and increase the reliability and accuracy significantly. This customized pattern is implemented as a robot mark placed on the top of the robot and tracked by the camera. The robot mark is a critical component of the whole system, because it directly impacts the system reliability and precision. Although both tasks, robot detection and robot pose measuring, can be implemented using single robot mark, in order to reach maximum detection reliability and maximum measuring precision and accuracy a combination of two robot marks is used: one for robot detection and one for robot pose measuring.

Robot Detection Mark

Design of the detection mark can employ different kinds of visual cues, like texture, known geometry, color etc. It is common for most cues that they depend on relatively large portions of the image at once or require extensive hypothesis-driven detection. In case when distinct geometric patterns are used high spatial resolution of the image and high computational power are required to detect every possible object position and orientation. On the contrary, using color as a cue offers several advantages such as robustness under rotation, scale and resolution changes, and the main advantage is processing speed. Namely, decisions can be initially made at pixel level, and then inductively at whole regions of similar color, and in case of careful robot mark design and efficient color recognition algorithm, high detection reliability can be obtained. This is the reason why color patches for design of the robot detection mark are chosen.

The first question that arises in detection mark design is whether or not robots should have different marks in order to enable the vision system to differentiate and uniquely identify each of them. In case of different marks it is possible to assign different color or combination of colors to each particular robot, so that robots can be uniquely distinguished from each other, but this approach has multiple drawbacks. Firstly, limited number of robots can be tracked since limited number of colors can be reliably distinguished, which is opposite to our goal that unlimited number of robots has to be tracked. Secondly, the system has to be calibrated for each color, and this would result with reduced robustness to light intensity changes and would also require long setup time. Thus, it can be more appropriate that all robots use equal color marks. Unfortunately, this has the drawback that it is impossible to distinguish one robot from another based solely on robot detection marks. To cope with this problem, identification method which can identify robots independently of robot detection marks is used, as discussed later in Section 3.3.3.

Other important questions concerning robot detection mark design are: (1) how many color patches to use; (2) where to place them on the mark; (3) which colors to select and (4) which color patch shape to choose. Of course there are no general answers to these questions and they depend on many factors, such as available robot mark area, camera spatial resolution (pixels per meter), robot environment etc.

Generally, number of color patches is limited with available area on the robot where the mark can be placed and size of a particular color patch must be sufficiently large in order for vision system to be able to recognize its color reliably. A higher number of color patches results with increased detection robustness because of lower probability that the selected combination of colors is accidentally found in image background thus resulting with false robot detection. On the other side, this also results with increased computation time required to detect and track higher number of colors, and the final decision about the number of color patches is application dependent.

Once the decision is made about the number and the size of color patches, it is necessary to place them optimally on the robot detection mark, where criterion can be maximization of detection reliability. In order to reach unambiguity of robot mark detection, it is convenient to place a “key patch” [31] in the middle of the color mark, and other color patches around it so that their distances to the key patch are identical. In this way, the key patch is first located in the image and after that other belonging color patches are located on a circle around the key patch with radius equal to the given distance, where it is guaranteed that no color patches of other robots can appear, but only patches that belong to the related robot, which significantly simplifies the color mark detection process. Another criterion is that color patches must be distributed so that the robot orientation can be unambiguously determined once locations of all individual patches are known.

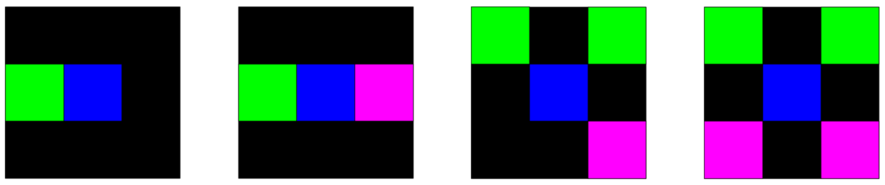


Figure 3.3. *Possible designs of the robot detection mark*

The patch colors can be selected in a manner that detection reliability is maximized, so that only saturated colors are used, and distance between particular colors on the hue scale is maximized, or alternatively colors that are less likely to appear in the image background can be used.

Concerning the shape of the color patch, in order to maximize color recognition reliability, the square color patch shape which provides maximum space utilization can be selected. Figure 3.3 shows some examples of possible detection mark designs, where black color is used for the background and blue color for the key patch. Of course many other designs are possible.

Robot Pose Measuring Mark

Once the problem of robot detection is solved, it is necessary to accurately and precisely measure its position and orientation. Since the camera with Bayer color filter array is used (Figure 3.2), only one color per pixel is transferred via bus to the host computer so that the remaining color information not detected by that pixel must be estimated in host computer using some kind of interpolation algorithm [125], [77]. Even in case that camera delivers full color images, the color interpolation is also performed, but it is hidden from the user and is implemented internally in camera (unless an expensive 3-sensor camera is used). Because color interpolation around the edges in the image can result with colors that do not appear in the real scene (zipper effect) [125], using color as visual cue for measuring position and orientation of the robot would cause low measuring accuracy and precision. Therefore, it would be of great benefit if a way of measuring directly in original raw Bayer image acquired by the camera could be found, thus avoiding measuring on estimated RGB image.

In order to solve the aforementioned problem, introduction of a new mark dedicated only for robot pose measuring is proposed. In this design, robot pose measuring mark is a white square surrounded by black edge, so there exists clear, maximum contrast and length black-white edge (Figure 3.4). The key point here is that if using black-white edge for robot pose measuring, color information is not important and measuring can be performed directly in Bayer image without need for any color interpolation, which results in significantly improved precision and accuracy over existing systems.

It is desirable that the robot pose measuring mark is as large as possible, so that maximum possible edge length is used for pose measurement. The size of the robot detection mark is less critical, since it is not used for measurement. Therefore, it is proposed that the robot pose measuring mark covers the whole available area on the robot and that the detection mark is placed on it, but in such a way that it doesn't cover edges of the robot pose measuring mark. The final recommended design of the robot mark, which consists of both robot detection and robot pose measuring marks, is shown in Figure 3.4.

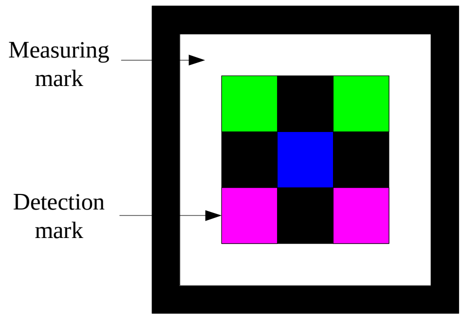


Figure 3.4. *Proposed robot mark consisting of robot detection and pose measuring marks*

3.3 Vision Algorithm

The task of the vision algorithm is detection and pose tracking of multiple mobile robots based on the analysis of the robot marks that appear in the image acquired by the camera. The vision algorithm consists of three main stages: robot detection, robot pose measuring and robot identification (Figure 3.5), which are described in detail bellow.

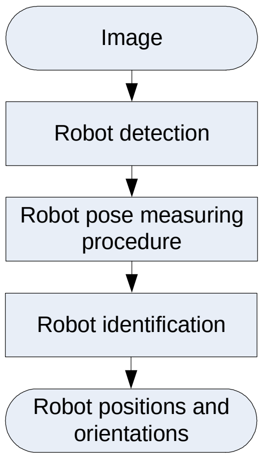


Image
Robot detection
Robot pose measuring procedure
Robot identification
Robot positions and orientations

Figure 3.5. *Block diagram of the vision algorithm*

3.3.1 Robot Detection

Robot detection is process of determining initial pose of the robot by means of locating its detection mark in the image. The flowchart of the robot detection algorithm is shown in Figure 3.6. As can be seen, the input to the robot detection algorithm is raw Bayer image from the camera, and output is approximate robot pose obtained using determined pose of color patches on the robot detection mark. Robot detection algorithm consists of two main stages: color classification and extraction of color patches.

Color Classification

As the color patches are used for robot detection, it is necessary to scan the image and classify pixels according to a color classification function. If the robot pose in previous frame is known it is enough to perform local image scan around expected robot pose (left branch of the flowchart in the Figure 3.6), with benefit of significant computation time reduction. But, in order to detect new robots entering the supervised area or to detect again temporarily occluded robots it is necessary to perform global scan of the image (right branch of the flowchart in Figure 3.6). Therefore, both local and global search are implemented and combined in such a way that ensures reliable vision system operation.

Local search procedure starts with prediction of the robot pose taking into consideration robot model and robot pose and velocity measured in previous sampling instant. Because of high measurement precision and accuracy (which are experimentally verified later in this chapter), and high framerate, the special filtering schemes such as Kalman filtering are not necessary, and only simple prediction based on kinematic model of the unicycle robot is used:

$$\begin{aligned}
x(k+1) &= x(k) + v(k)T\cos(\theta(k) + \omega(k)T),\\
y(k+1) &= y(k) + v(k)T\sin(\theta(k) + \omega(k)T),\\
\theta(k+1) &= \theta(k) + \omega(k)T,
\end{aligned} \tag{3.1}$$

where x, y, and θ are coordinates of measured robot position and orientation relative to some global coordinate system, v and ω are linear and angular robot velocities, respectively and T is the sampling time. Linear velocity of the robot is estimated with the following equations:

$$\begin{aligned}
&v(k) = \operatorname{sgn}(\vec{o}\cdot\vec{d})\frac{\left|\vec{d}\right|}{T},\\
&\vec{o} = \left[\cos(\theta(k)),\ \sin(\theta(k))\right]^T,\\
&\vec{d} = \left[x(k)-x(k-1),\ y(k)-y(k-1)\right]^T,
\end{aligned} \tag{3.2}$$

where $\vec{o}$ is robot orientation vector and $\vec{d}$ is robot motion vector. Angular velocity estimate is given by:

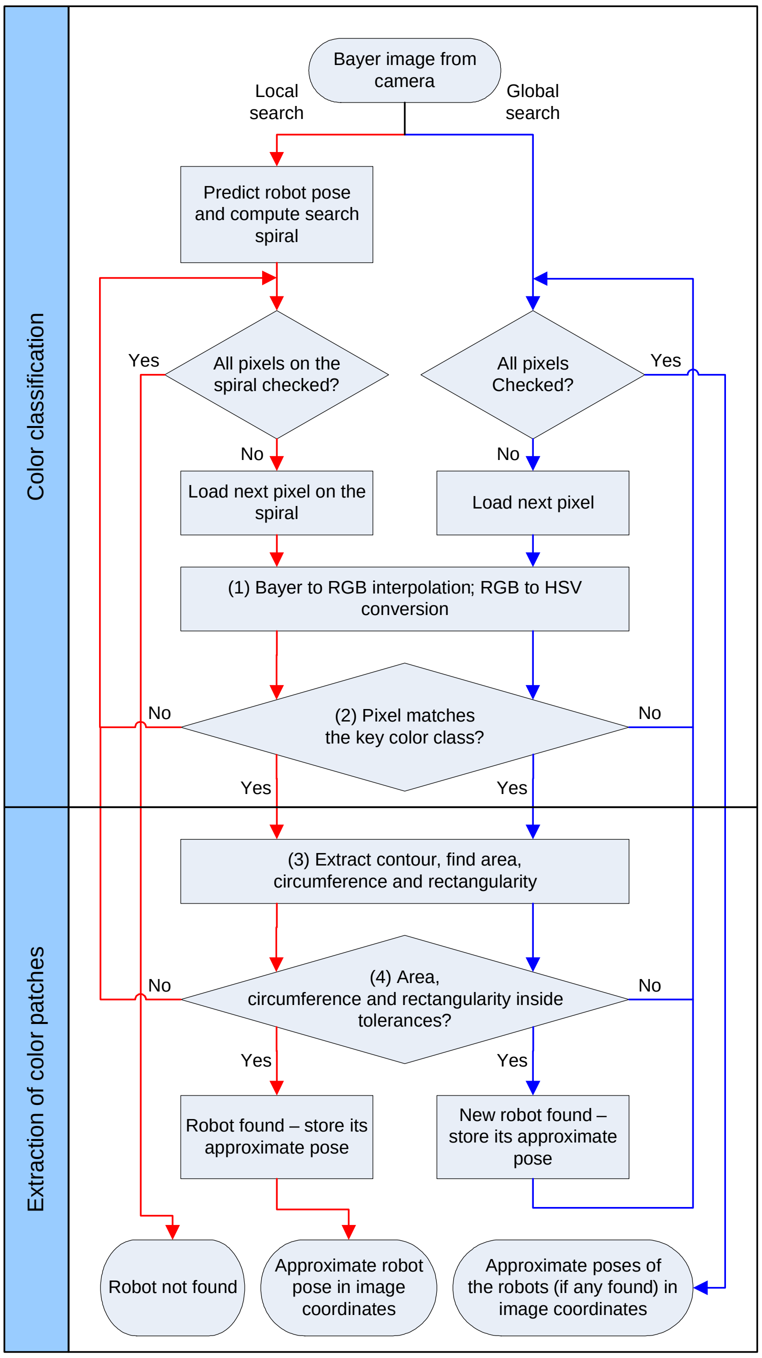

Color classification
Extraction of color patches
Bayer image from camera
Local search
Global search
Predict robot pose and compute search spiral
All pixels on the spiral checked?
All pixels Checked?
Yes
No
Yes
No
Load next pixel on the spiral
Load next pixel
(1) Bayer to RGB interpolation; RGB to HSV conversion
(2) Pixel matches the key color class?
No
No
Yes
Yes
(3) Extract contour, find area, circumference and rectangularity
(4) Area, circumference and rectangularity inside tolerances?
No
No
Yes
Yes
Robot found – store its approximate pose
New robot found – store its approximate pose
Robot not found
Approximate robot pose in image coordinates
Approximate poses of the robots (if any found) in image coordinates

Figure 3.6. *Flowchart of the robot detection stage*

$$\omega(k) = \begin{cases} \dfrac{\Delta\theta}{T}, & |\Delta\theta| < \dfrac{2v_{\max}T}{L} \\ \dfrac{\Delta\theta - 2\pi}{T}, & \Delta\theta > \dfrac{2v_{\max}T}{L} \\ \dfrac{\Delta\theta + 2\pi}{T}, & \Delta\theta < \dfrac{-2v_{\max}T}{L} \end{cases}, \tag{3.3}$$

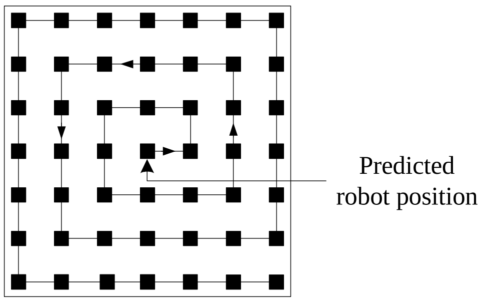


Figure 3.7. *Spiral search of the region of interest in local search*

where $\Delta\theta = \theta(k) - \theta(k-1)$, v_{max} is maximum robot velocity and L is the distance between robot wheels. The predicted robot pose obtained by (3.1) is taken as the center of the rectangular region of interest that is searched in order to detect the robot. The size of the region of interest is chosen to grow proportionally with the robot velocity. The next step is scanning of determined region of interest in order to localize the key color patch on the robot detection mark. To minimize the probability that any other object of similar color, or any other robot is found instead of robot that is being searched (this tracking failure would result with swapping of robot identifications), the searching begins right from the middle of the region of interest, i.e. from the predicted robot pose. Then the searching is continued following the rectangular spiral toward the border of the region of interest (Figure 3.7). Actually, regarding the dimensions of the color patch, not every pixel must be checked. For example, if color patch has size of 9 pixels, as was the case in the experiments, it is safe enough to have a scan interval of one third of searched patch size in pixels, i.e. every third pixel is scanned. In this way, a total number of checked pixels is reduced nine times. Using this approach in combination with high vision system framerate, swappings of robot identifications are almost completely avoided.

Global search procedure is much more computationally complex than local search as explained afore. Therefore it must be performed only in situations when previous pose of a robot in the space is not known. However, it is not always possible to unambiguously detect such situations, e.g. vision system may not be aware that a temporary occluded robot is visible again. Because of that a more advanced approach is used, where global search is performed continuously in every image frame. In order not to interrupt other algorithms and not to increase system latency, it is conducted in computer idle time, after local search and all other algorithms are executed, and commands are already sent to robots, but before a new image is acquired. Of course, this approach requires that some processor idle time is left available after all other algorithms are executed, but it is not necessary to assure that this time is sufficient for complete global search execution. Namely, if hardware is not capable to process entire image in remaining processor idle time, the image is scanned only partially in single image frame, so that complete global search procedure is spanned over multiple frames. This trade-off enables maintaining a high frame rate, while continuously executing global search procedure. Of course, for computational efficiency reasons here also every third pixel is scanned.

In order to enable color-based robot detection the color interpolation from Bayer format to the RGB color space is computed for every pixel that is being scanned, no matter global or local search is performed (block (1) in the Figure 3.6). Advanced interpolation methods have been developed [125] that can deal with most common interpolation artifacts such as zipper effect, but they have high computational complexity. Since color patches are used only for robot detection and not for robot pose measuring, it is not necessary to apply an advanced interpolation method. Therefore the bilinear interpolation algorithm is used that has advantages of relatively low complexity and acceptable interpolation quality for given application. However, it is well known that the RGB color space shows too high scattering of all three color components when illumination changes so that reliable color detection is very difficult, and that the HSV (hue-saturation-value) color space shows high scattering of V component only, while H and S components have relatively low scattering no matter of location in the image or light intensity [51]. Therefore the HSV color system is far better choice for color detection as it enables reliable color recognition based on H and S components. However, many real-time color tracking systems use RGB color space due to computationally expensive conversion from RGB to HSV space. In order to enable the usage of the HSV color system, an efficient way of RGB to HSV conversion is used. In computer vision applications lookup tables are used commonly as they are the most efficient way to avoid complex calculations. The lookup table (LUT) used for RGB to HSV conversion is of the following form:

$$(H, S, V) = LUT(R, G, B). \tag{3.4}$$

This is a 3D lookup table whose size in computer memory for 8 bits per color component is $3{\cdot}2^{3\cdot 8} = 48$ MB. As this is not acceptable, the color resolution of input components (R, G, B) is reduced to 6 bit per color component so that table size is decreased to $3{\cdot}2^{3\cdot 6} = 768$ kB which is acceptable for today's computers. In this way color conversion resolution is reduced, especially for colors with low saturation and intensity values, but this is not an issue in our application because acquired image contains high amount of noise anyway, and only a few colors have to be recognized that are well separated on a hue scale and have high saturation and intensity values. Note that the approach with the reduced lookup table also allows conversion to any color space other than HSV, and in the same time, various corrections to pixel values can be made if necessary, such as gamma correction etc.

The final operation of the color classification stage is examination if the color of a pixel matches any of defined color classes (color class is a subset of all possible pixel color values) assigned to color patches (block (2) in the Figure 3.6). The matching criterion is implemented as a thresholding operation, where each color class is specified as a rectangular volume in the color space and is described with set of six thresholds: low and high threshold are defined for each color space dimension. Now the test is performed to check if pixel's HSV components fall between thresholds of a particular color class, where the thresholds are defined a priori using a GUI tool that we have developed. A naive implementation of this thresholding operation is rather inefficient because in the worst case it could require six comparisons per each color class, which can result with bad performance especially on modern pipelined processors with speculative instruction execution. Therefore a color classification algorithm is used that is capable of testing membership for all defined color classes at once with only two AND operations, as described in [30]. Since the HSV color representation is used, this classification method gave good classification results in the experiments with moderate light intensity changes. In case that light intensity change would be so high that classification results would no longer be acceptable despite of using HSV, such as in outdoor environments, it is possible to apply schemes that have been specially developed to deal with this problem in real time, for example see [29], [143], [53]. Since lookup table for RGB to HSV color conversion is already used in previous step, one may wonder why the color–color class mappings are not stored in this lookup table and in this way color recognition additionally accelerated, as was done for example in [11]. The reason is that this scheme would not allow to dynamically adjust thresholds of color classes because it would took too long time to update 3D lookup table. Instead, in algorithm that is used it is only required to update three arrays of 256 members to adjust thresholds, which enables real time thresholds adjustment and consequently increases flexibility.

Extraction of Color Patches

If the color classification procedure identifies a pixel that meets the matching criterion for color patch class, the procedure is executed that labels the whole region of the given color class starting from that pixel. As the robot detection mark is used with the key patch in its center, only color class of the key patch is being searched initially. The common approach to region labelling problem, in case starting pixel is known, is application of region growing methods. Under the assumption that we are not interested in region interior properties, such as number of holes in it, it is sufficient to extract only edge contour of the region. Thus a contour following algorithm is used [141], which has the advantage of low computational cost because not every pixel of the region is being checked, but only pixels around the edge of the region, and pixels inside this contour are assumed to belong to the region. The algorithm begins from some starting pixel on the bottom edge of the region and its outputs are integer coordinates of edge contour pixels. Effect of the algorithm on real robot image is illustrated in Figure

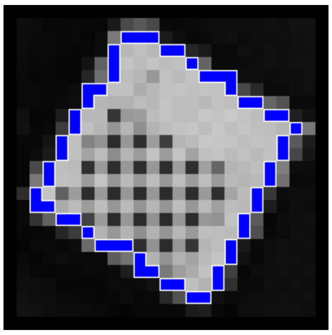


Figure 3.8. *An example of contour following on real robot image in Bayer format (extracted contour pixels are labelled with blue color)*

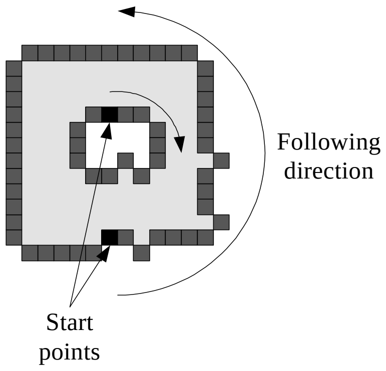


Figure 3.9. *Contour following procedure with different starting points*

3.8.

Here a brief overview of the procedure for determining the starting pixel for contour following is also given. Contour following algorithms locate this pixel beginning from some pixel inside the region (in our case this is the pixel found by the global or local search procedure), and scan the image in e.g. bottom direction. The starting pixel is then the first pixel found, which has bottom neighbor that does not belong to the region. However in this application this simple approach does not suffice, since it does not ensure this pixel to be located on the edge of the region, as it could also be on the edge of a hole contained inside the region (such a hole may be result of the noise) as shown in Figure 3.9. This problem can be solved by examining the contour following direction, which can be obtained using the chain code representation [141] of the extracted contour. In case that the found starting pixel is really on the region edge, the contour following algorithm will run in counterclockwise direction starting from that pixel (see Figure 3.9). But if starting pixel is on a hole edge, the following direction will be clockwise. In that case the starting pixel searching procedure is continued by searching again in bottom direction until new edge point is found that will result in correct following direction.

Finally, object recognition is done by computing area, circumference and rectangularity of the region surrounded by extracted contour (block (3) in the Figure 3.6) and their comparison with the expected values for given pose in the image (these values vary along the image because of lens distortion). If these descriptors are inside tolerances (block (4) in the Figure 3.6), there is a high probability that the found region really represents the key color patch on the detection mark of the robot. In case that apart from the key color patch additional color patches are used, their locations have also to be determined. As they all have equal, exactly defined distance to the key patch (see Figure 3.3), it is sufficient to search along the circle centered in the key patch position, with radius equal to the given distance. When a pixel is found that belongs to some of predefined color classes of other color patches, the same contour following procedure is repeated as for the key color patch. If all color patches are located successfully, robot detection procedure is completed and robot pose is now approximately known. While the existing color-based tracking algorithms usually finish at this point and use computed robot pose as the exact one, e.g. [85], [31], [45], [104], here it is used only as the input for the robot pose measuring procedure, which significantly increases accuracy and precision of the robot pose estimation.

3.3.2 Robot Pose Measuring Procedure

The flowchart of the robot pose measuring procedure is given in Figure 3.10. Measuring of robot position and orientation is based on subpixel detection of the black-white edge of the robot pose measuring mark (Figure 3.4). Here advantage can be taken that this edge can be detected directly in raw Bayer image because theoretically gray tones should have equal values of all three color components of RGB image so that raw Bayer image can locally be treated as a monochrome gray scale image. However in praxis these three color components may differ slightly depending on the illumination type and imaging sensor type (example can be seen in Figure 3.11 (a) where a raw Bayer image of the robot mark is given). This can be corrected by applying appropriate gains to red and blue color components so that for gray tones these components are equal to the green component. This correction is usually called white balance adjustment [3]. An example of white balance corrected image is given in Figure 3.11 (b), and for illustration in Figure 3.11 (c) a RGB image interpolated from corrected Bayer image using bilinear interpolation is given.

Robot pose measuring procedure starts similarly to the one used for extracting the region of the color mark. Namely, the edge contour of the robot pose measuring mark is extracted (block (2) in Figure 3.10) from the white balance corrected source image in raw Bayer format (block (1) in Figure 3.10), using the known approximate robot pose in image coordinates as a starting point. To save computational time, white balance correction is done only for pixels that are tested by contour following algorithm. As a criterion for examining if a pixel belongs to the

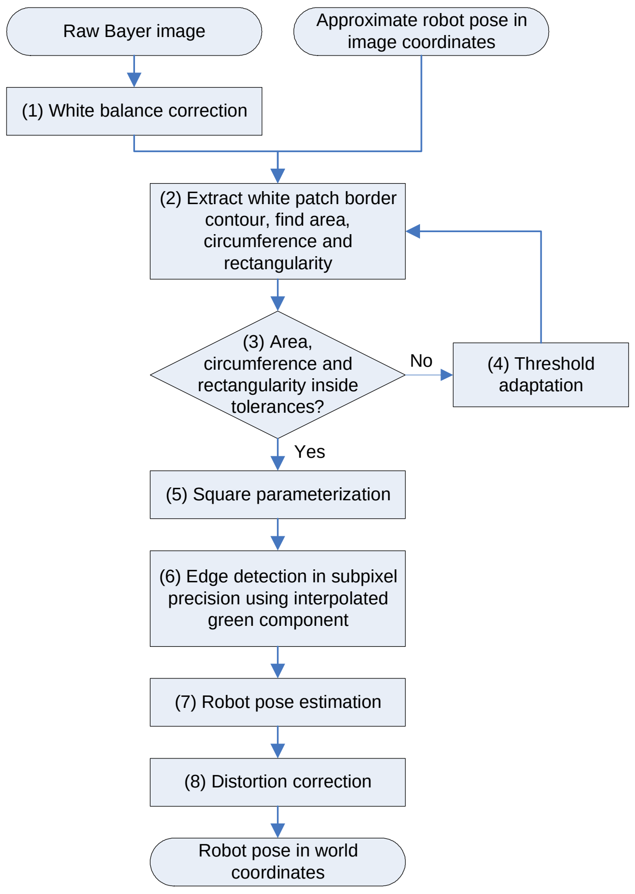

Raw Bayer image
Approximate robot pose in image coordinates
(1) White balance correction
(2) Extract white patch border contour, find area, circumference and rectangularity
(3) Area, circumference and rectangularity inside tolerances?
No
(4) Threshold adaptation
Yes
(5) Square parameterization
(6) Edge detection in subpixel precision using interpolated green component
(7) Robot pose estimation
(8) Distortion correction
Robot pose in world coordinates

Figure 3.10. *Flow chart of the robot pose measuring procedure*

white region, a threshold is used. If the gray tone intensity of a pixel is greater than the threshold value, the pixel is labelled as white. When the contour of the white measuring mark region is extracted using some initial threshold value, region area, circumference and rectangularity are computed. If these descriptors are inside given tolerances (block (3) in Figure 3.10), we declare that the white measuring mark is located. Since light intensity is not constant, but depends on the location in the image, and can also be time variant, the failure can occur in the contour edge localization. This implies that adaptive threshold should be used in order to get system robust to light intensity variations. Threshold adaptation algorithm (block (4) in Figure 3.10) computes new threshold value using previous threshold value, approximate robot pose in image coordinates which is used as starting point for contour extraction, and white balance corrected raw Bayer image. The threshold is adapted using the fact that the circumference and area of extracted white mark region are proportional to the light intensity, so that threshold is iteratively adjusted and the edge contour extraction process is repeated with this adjusted threshold value until the area of white mark region

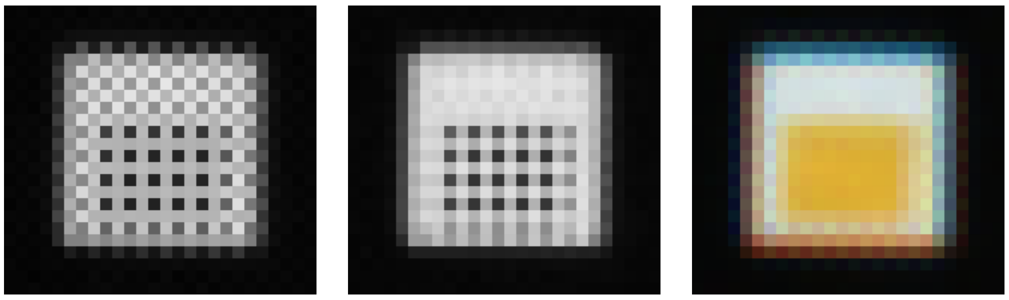

(a) (b) (c)

Figure 3.11. *Bayer image of a robot pose measuring robot mark:* (a) *raw image;* (b) *white balance corrected image;* (c) *interpolated RGB image*

gets as close to real as possible (real area is known because of known white mark size and known location in the image). When the appropriate threshold value is found, it is stored in virtual grid where each grid cell is assigned to respective image fragment. This threshold value is then used until the illumination changes again. In case that illumination variation is so high that even the threshold adaptation does not give acceptable results, camera gain or shutter can be adapted as well.

Once the edge contour of the white mark is extracted, it is parameterized with the best fit square (block (5) in Figure 3.10). Inputs to this block are coordinates of white patch edge contour pixels and outputs are parameters of the best fit square: coordinates of square center (i.e. intersection of diagonals), side length, and rotation angle, which is defined as orientation of the bottom side of square which can be in range $(-45°, 45°]$. Center of the square is computed as an average value of x and y coordinates of all edge contour pixels, side length is taken as known size of the white mark in pixels, and the only parameter that left to be found is square orientation. It is found by an iterative method, where the specified square is rotated, beginning from angle 0°, in finer and finer angle steps, until the mean square distance of contour points to the square sides is minimized [26]. The rotation angle step is halved in each iteration, and the direction of rotation is taken so that the distance criterion gets lower. This method achieves square orientation resolution of less than 1° with only six iterations, and computational cost is not high because the number of contour pixels that are fitted with the square is small. Although the method can determine orientation in range $(-45°, 45°]$ only, the range can be extended to $(-180°, 180°]$ based on the previously detected color mark orientation. In this way, approximate location of the white region edge is found with relatively low computational effort, what is of great benefit for acceleration of later algorithm stages.

The contour following algorithm uses threshold criterion to find the contour so that the determined edge position is not very precise and is sensitive to noise.

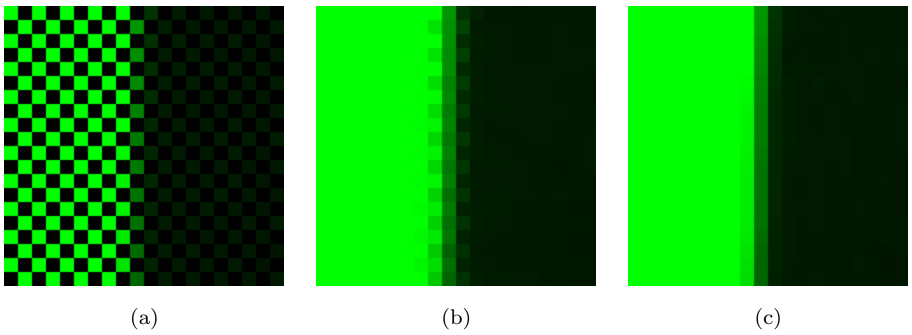


Figure 3.12. *Green component interpolation:* (a) *raw green pixels;* (b) *bilinear interpolation;* (c) *adaptive interpolation*

To further improve measurement results, refinement of extracted white area edge contour in subpixel precision is made. A problem that arises here, which is common for color cameras with wide-angle lens, is chromatic aberration. There are two types of chromatic aberration: (i) longitudinal aberration, which is inability of lens to focus different colors in the same focal plane, so that only one color component is sharply focused, and (ii) transverse aberration, which refers to sidewards displaced foci for different colors and results with displaced edges of objects in different color planes. If imaged object is near the image border, chromatic aberration can result with object edge displaced even for several pixels in different color planes which would make measurement in subpixel precision senseless. Since software compensation of this distortion would require too high computation time, our solution for this problem is to use only one of the three color components for measurement so that chromatic aberration is avoided. The green component is chosen because in Bayer pattern there are twice as many green pixels as red or blue ones (Figure 3.2), so that only 50 % of pixels have to be interpolated. In order not to deteriorate measurement precision it is very important to perform interpolation in an adaptive way so that edge structure is preserved. Here the known approximate edge direction can be utilized for adaptation, by interpolating missing values in the direction of the edge. In order not to significantly increase computational complexity, only three cases are considered: (i) if the edge is horizontal a missing green pixel is interpolated as mean value of left and right neighbor pixels; (ii) if the edge is vertical it is interpolated as mean value of upper and lower pixels; and (iii) if the edge is slopewise it is interpolated as mean of four neighboring pixels. This is illustrated in Figure 3.12 with images of nearly vertical black-white edge, where Figure 3.12 (a) shows raw green pixels acquired by the camera, Figure 3.12 (b) the edge structure obtained with bilinear interpolation (zipper effect is visible), and Fig 3.12 (c) the edge structure with described adaptive interpolation.

The next step is edge detection of robot pose measuring mark in subpixel

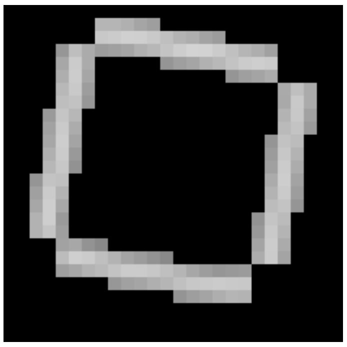


Figure 3.13. *An image with gradient magnitudes of the edge of the robot pose measuring mark*

precision using interpolated green component (block (6) in Figure 3.10). Inputs to this block are square parameters from the square parameterization procedure (block (5) in Figure 3.10) and white balance corrected raw Bayer image (block (1) in Figure 3.10). Outputs are coordinates of white patch edge contour pixels refined in subpixel precision.

The main problem with subpixel edge detection methods is their high computational cost, because it is commonly required to compute gradient magnitude and direction for each image pixel. This represents high computational burden, as it is commonly conducted by applying a derivative filter in both horizontal and vertical image direction, and then the gradient magnitude and direction are computed in each pixel from horizontal and vertical gradient components. Fortunately, in our case a great reduction of computational time can be obtained by utilizing the fact that approximate edge positions and orientations are known from the square parameterization procedure. This results with the following three benefits: (i) it is not necessary to interpolate the green component, nor to compute gradient magnitude and direction for each pixel, but only for pixels in neighborhood of best fit square border, (ii) gradient directions do not need to be computed since they are already accurately approximated by directions perpendicular to the sides of the best fit square, and (iii) using the fact that gradient directions are known, computing of gradient magnitudes is considerably simplified, since it is possible to reduce complicated problem of edge detection in two-dimensional space to the problem of detecting edges in one dimension only. Namely, it is sufficient to apply the derivative filter in one direction only, i.e. in the previously computed gradient direction. To further reduce computational burden, gradient directions are discretized, so that there are eight possible gradient directions allowed.

Thus, gradient magnitude is computed by applying the derivative filter in horizontal, vertical or slopewise image direction, depending on gradient direction. The example of gradient magnitude image of the edge of the robot pose measuring

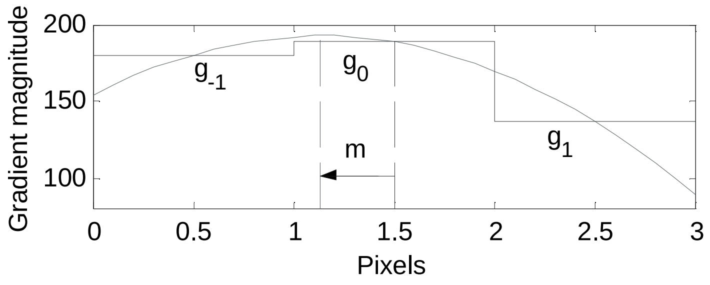


Figure 3.14. *Example of computing the edge position in subpixel precision*

mark obtained using this method is shown in Figure 3.13 (size of applied derivative filter convolution mask was 4), where it can be seen that for each edge pixel, three gradient magnitude values are computed. Local maximums of the gradient image can be observed in the image as the most bright pixels in the middle of the edge, and for each local maximum two neighbor gradient values are computed so that subpixel edge position can be estimated. The subpixel edge position is computed as follows: if the local maximum of gradient image is in pixel with coordinates (x, y), g_0 is gradient magnitude of that pixel, and g_1 and g_{-1} are gradient magnitudes of neighbor pixels in gradient direction, and in direction opposite to the gradient direction, respectively, (see Figure 3.14) then the bias of edge position m is computed using quadratic interpolation [43]:

$$m = \frac{g_{-1} - g_1}{2\left(g_{-1} - 2g_0 + g_1\right)}. \tag{3.5}$$

The subpixel edge position is now estimated as:

$$\begin{aligned} x_s &= x + md_x \\ y_s &= y + md_y, \end{aligned} \tag{3.6}$$

where (x_s, y_s) is subpixel edge position and (d_x, d_y) are projections of gradient direction vector to x and y axes with possible values of $\{-1, 0, 1\}$ depending on edge direction.

Based on the subpixel coordinates of the white Bayer edge contour, robot pose is estimated (block (7) in Figure 3.10) relative to the image coordinate frame. First, edge points of each square side are approximated with lines using least squares method, and then robot position is computed as centroid of quadrangle formed by those four lines, and robot orientation angle is computed as average value of all four line angles. Using this method, measurement precision is greatly improved with only slightly increased total computational time.

In order to calculate robot pose in the world coordinate frame from its pose in the image coordinate frame it is necessary to correct lens distortion (block (8) in Figure 3.10). The correction could be applied directly to the image pixels, or to the measured position and orientation of the robots. Approach that works directly on the image data is not applicable in this case, because of its high computational load and loss of precision. Thus the correction is performed on measured coordinates using camera calibration parameters determined in setup phase (see Subsection 3.2.1). Intrinsic parameters are used to obtain undistorted robot pose relative to the image coordinate frame, and then these corrected coordinates are converted to 2D world coordinate frame using extrinsic parameters of the camera.

3.3.3 Robot Identification

Robot identification is a procedure of assigning unique identification number to each individual robot localized in the image. As the equal marks for each robot enable only localization of the robots, but not their identification, reliable robot identification method is essential for practical use of the proposed vision algorithm, otherwise the measured positions and orientations are useless for robot navigation because we do not know which data belongs to which robot.

The simplest identification method is manual assignment of identification numbers to individual robots in the image. Unfortunately, this method is suitable only for small number of robots and can be used only in initialization phase, so that application of one of two different methods is suggested. The first method is based on special movement commands that are transmitted to robots and then this movement is recognized by the software. For example in case of mobile robots with differential drive that are capable of rotating in place, the first robot can be commanded to rotate full circle, the second can rotate also full circle but in opposite direction, the third robot rotates two circles and so on. The second method is usage of controllable identification light (LED) mounted on top of the robot. Then if necessary the software can turn on and detect this light in the image, and distinguish the robot based on the color of light, number of light on-off sequences or duration of the light. These procedures are used whenever robot identifications are unknown, that is in initialization phase, in case when new robots enter the field of view, when robots identification marks are temporary occluded and when two or more robots swap identifications because of some unpredictable situations like very fast collisions.

Another problem that arises here is how to detect tracking failure, i.e. false robot identification (e.g. when swapping occur). One possible solution, which is also able to correct this mistake in a few samples only, is described in [20]. The approach is based on continuous online comparison of command velocities sent to robots and measured robots velocities using residuals. If $v_i(k)$ and $\omega_i(k)$ are commanded linear and angular velocities and $\hat{v}_i(k)$ and $\hat{\omega}_i(k)$ are measured linear and angular velocities of the i-th robot at time sample k, residual Rv_{ij} for the linear velocities is calculated as

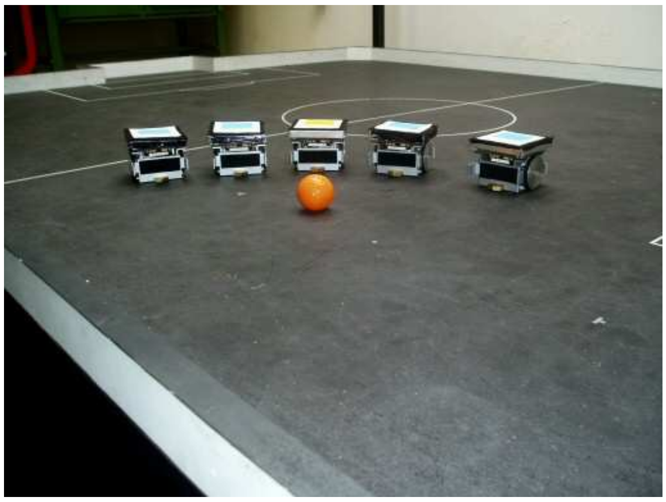


Figure 3.15. *Robot soccer platform*

$$Rv_{ij}(k) = e^{-\frac{T}{T_f}} Rv_{ij}(k-1) + (1 - e^{-\frac{T}{T_f}}) \left| v_i(k) - \hat{v}_j(k) \right| , \tag{3.7}$$

and residual for angular velocities $R\omega_{ij}$ as

$$R\omega_{ij}(k) = e^{-\frac{T}{T_f}} R\omega_{ij}(k-1) + (1 - e^{-\frac{T}{T_f}}) \left| \omega_i(k) - \hat{\omega}_j(k) \right| , \tag{3.8}$$

where T_f is the filter time constant. If there are no wrong identifications, then all Rv_{ij} and $R\omega_{ij}$ with $i = j$ are lower than some predefined thresholds. However in case that commanded and measured velocities are not correlated, i.e. identification failure has occurred, the process of reassignment of identification numbers is started so that proposed criterion is minimized. This approach is successfully used in this work.

The main drawback of described methods is that none can identify the robot based on single image. This may be limitation for some time critical applications because some initial time is required for identification. However for many applications it is acceptable, because this time is very short since high sampling rate is used.

3.4 Experimental Results

The main goal of experiments was to verify whether the proposed vision system meets the given requirements. Experiments were performed using robot soccer platform (Figure 3.15) that is ideal for testing various mobile robot navigation algorithms [78]. It consists of a team of five radio-controlled microrobots of size 7.5 cm cubed with differential drive and maximum velocity 4 m/s. The playground is of size 2.2×1.8 m. Above the center of the playground, Basler a301fc

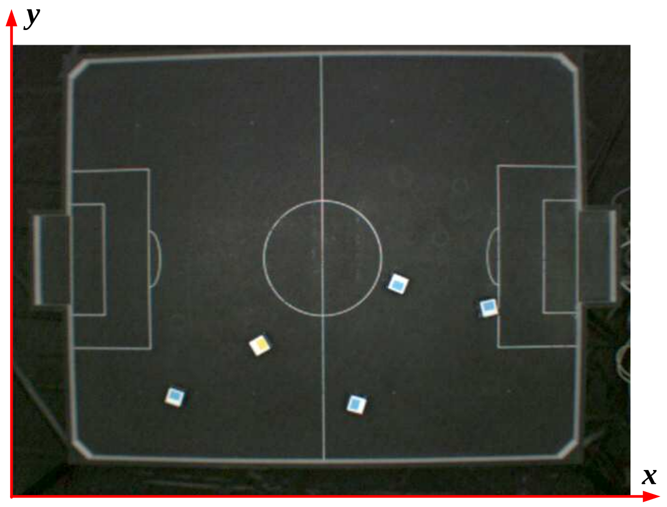


Figure 3.16. *Typical image acquired by the camera*

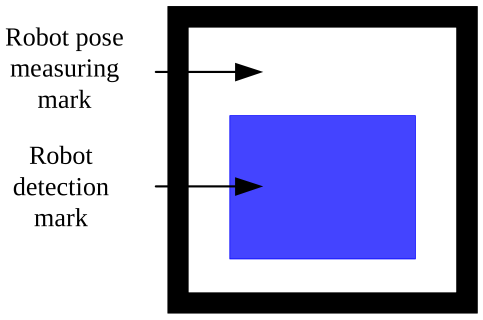


Figure 3.17. *Robot marks used in experiments*

IEEE-1394 Bayer digital color camera with resolution of 656×494 pixels and with maximal framerate of 80 fps is mounted perpendicular to the playground. The height of the camera to the playground is 2.40 m. A wide angle 6 mm lens is used. For illustration, a typical image acquired by the camera is shown in Figure 3.16.

The robot mark shown in Figure 3.17 was used in experiments. The mark size is 7.5 × 7.5 cm, giving the image size of the white area of the robot pose measuring mark of about 16 × 16 pixels. As can be seen, robot detection mark consists of only one color patch, which was sufficient for reliable robot detection, because rules for robot soccer propose that blue or yellow color is reserved only for robot identification and must not be used for any other purposes. The benefit is also lower computational complexity of algorithms for robot detection. In order to determine robot orientation unambiguously the color patch is not placed in the middle of the white robot pose measuring mark, but shifted to the rear side of the robot.

In all experiments, an image coordinate frame (in pixels) that is used corresponds to the image borders as indicated in Figure 3.16, where x axis is in range $[0, 655]$ pixels, and y axis in range $[0, 493]$ pixels. Global coordinate frame (in meters) corresponds to playground borders, as indicated in Figure 3.18 (b), where x axis is in range $[0, 2.2]$ m, and y axis in range $[0, 1.8]$ m.

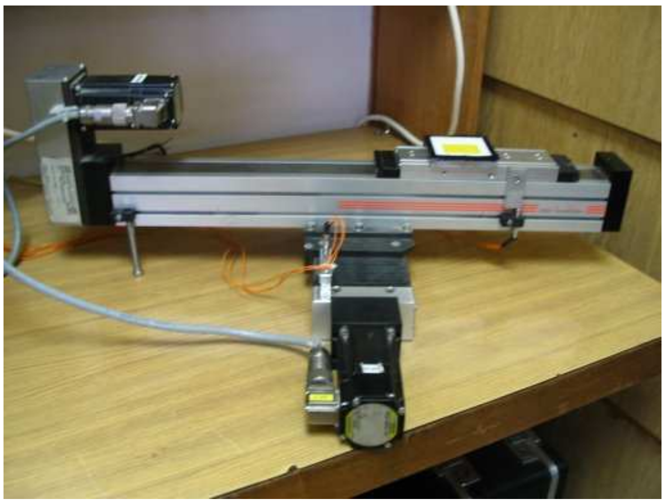
A number of experiments were performed and below obtained results and their analysis is presented. First, results of the experiments related to the precision and accuracy of measured robot position and orientation in real world conditions are described and analyzed. Then analysis of the vision system performances related to other requirements stated in the introductory section is conducted.

3.4.1 Analysis of Vision System Precision and Accuracy

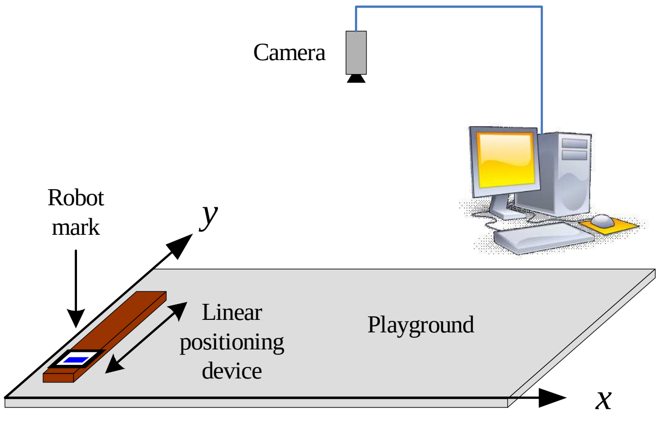
The analysis of precision and accuracy was performed using method described in [144], where it was used to evaluate subpixel line and edge detection precision and accuracy. Results of three experiments are presented illustrating high precision and accuracy of both robot position and orientation measurements.

In the first experiment, the linear positioning device Linear Technology, PE1.4, LM 6.2 (Figure 3.18 (a)) with robot mark on it were placed on playground (Figure 3.18 (b)). The precision of linear positioning device is 10 μm so that robot mark can be shifted in one direction with resolution of 10 μm. The robot mark was placed near the image border, so that influence of chromatic aberration was notable and comparison of various algorithms with respect to this effect could be performed. Using the linear positioning device, the robot mark was accurately shifted in y direction of global coordinate frame in increments of 100 μm. A total number of 100 increments were made so that total shift of the robot mark was 1 cm. In each position 20 images were taken, and for each of those images y coordinate of robot pose was calculated using proposed algorithm. Then the standard deviation and mean of those 20 obtained y position coordinates were calculated for each position. Figure 3.19 (a) shows obtained mean of measured y coordinates as a function of robot mark shift and Figure 3.19 (b) precision of the position measurement, i.e. its standard deviation. Each single value in those figures is calculated from 20 obtained y coordinates in each position. It can be seen that the standard deviation is almost everywhere less than 0.01 pixel (0.040 mm) and the average standard deviation is 0.007 pixel (0.027 mm).

To determine absolute position accuracy camera would have to be calibrated and results would be highly impaired by the quality of the calibration. To avoid this dependency in the experiment it was assumed that the linear shift of the object in the real world corresponds to the linear shift in the image which is true for small shifts. Therefore a straight line can be fitted through measured positions by means of least squares method. The equation of the line obtained from the experiment shown in Figure 3.19 (a) is $y = 0.2484x + 84.6973$, where y is in pixels and x in millimeters. Thus, 1 mm in the real world corresponds to 0.2484 pixels



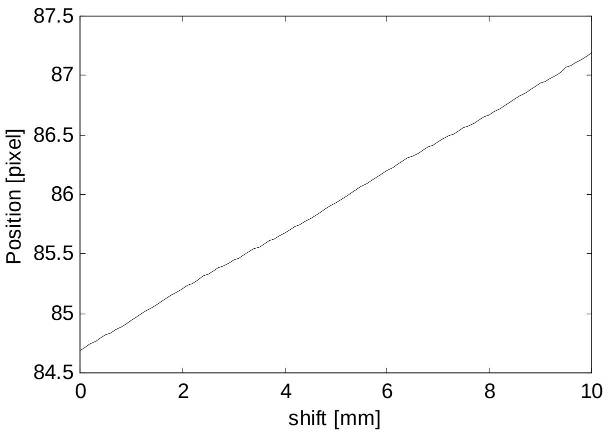
(a)



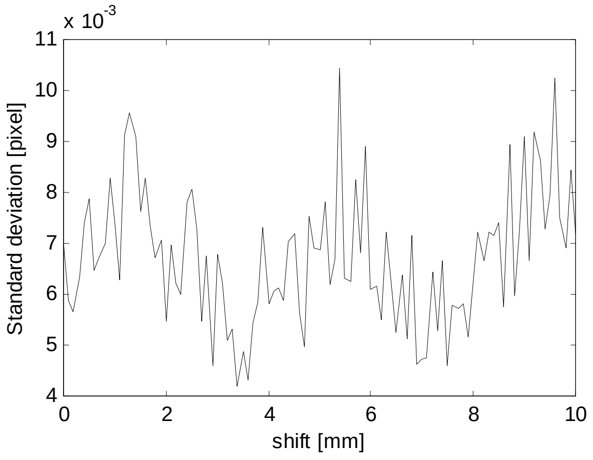
(b)

Figure 3.18. (a) *Linear positioning device;* (b) *Sketch of the experimental setup*

for this part of the image, i.e. one pixel corresponds to the length of 4.0258 mm. Using this equation, the absolute position error of the line is calculated as the difference of the measured position mean and obtained regression line. The results are shown in Figure 3.20. As can be seen, the absolute error shows a systematic sinusoidal component, which may be caused by the mapping of the scene intensity to the pixel values in the image, but it is always less than 0.002 pixels (0.008 mm). Of course, the absolute accuracy (but not the precision) will vary in different parts of the image, and in general can be worse than obtained in the experiment, because it depends not only on vision algorithm, but also on many other factors, such as quality of lens, quality of calibration, etc. However for many applications precision is more important than absolute accuracy, because it enables precise detection of robot relative movement and thus directly impacts the quality of robot control and motion.



(a)



(b)

Figure 3.19. *Mean* (a) *and standard deviation* (b) *of the measured robot position y coordinate as functions of the robot shift*

The next series of experiments were conducted in order to compare the proposed method with other similar methods. As the first method for comparison (method M1), the standard color based method is used (e.g. see [31]), which does not utilize any kind of subpixel edge detection algorithms and is commonly applied for robot pose tracking in robot soccer application. The second method (method M2) measures the black-white edge position directly in Bayer image with pixel precision. This method is based on square parametrization method described in Section 3.3.2, and it was our first attempt to enhance existing methods [26]. The third and the forth methods (methods M3 and M4) measures black-white

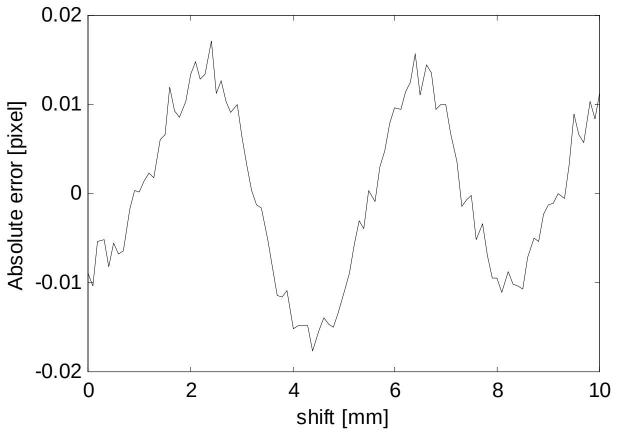


Figure 3.20. *Absolute error of the robot position y coordinate*

edge position in subpixel precision, but they differ in the fact that method M3 estimates edge position directly in Bayer image and is thus sensitive to chromatic aberration, and M4 is the proposed method that uses interpolated green component for edge detection. The comparison results are shown in Table 3.1. Measurements for all four compared methods were performed simultaneously, so that they operated in the same conditions. The table shows average standard deviations (σ_y) and average values of absolute error (e_y) in y coordinate of the position for all four methods as well as their decreases with respect to the method M1 ($\sigma_y^{(\mathrm{M1})}/\sigma_y^{(\mathrm{M}i)}, e_y^{(\mathrm{M1})}/e_y^{(\mathrm{M}i)}, i = 2, 3, 4$).

As can be seen from the table, improvements of both position precision and accuracy are noticeable even with method M2 (precision 2.6 times, accuracy 1.3 times), which are achieved by measuring directly in Bayer image although with pixel precision only. But, methods that deal with subpixel precision enhance the precision more than an order of magnitude (method M3 13.2 times and method M4 11.6 times). The slightly worse precision obtained by method M4 is most likely caused by green color interpolation. Methods M3 and M4 outperform the other two also in accuracy but method M3 not so significantly (improvement: 2.7 times wrt M1 and about 2 times wrt M2) as method M4 (improvement: 25.6 times wrt M1, about 20 times wrt M2 and 9.5 times wrt M3). So big difference in accuracy of methods M3 and M4 is most likely due to chromatic aberration effect, which is present in M3 and avoided in M4 by green color interpolation.

The precision and accuracy of measured orientation angle is evaluated in similar manner as for the position. The angle is measured while the robot mark is rotated in increments of 0.29°, which was achieved using described linear positioning device and a lever for converting linear shifts to rotation increments. In order to enable color based angle measurement, which was needed for methods comparison, apart from yellow color patch another color patch of smaller size was

Table 3.1. *Comparison of position precision and accuracy for different methods*

Method	σ_y [pix/mm]	$\sigma_y^{(\mathrm{M1})}/\sigma_y^{(\mathrm{M}i)}$	e_y [pix/mm]	$e_y^{(\mathrm{M1})}/e_y^{(\mathrm{M}i)}$
M1: Color tracking	0.0776/0.3124	1	0.2050/0.8253	1
M2: Bayer pixel	0.0297/0.1196	2.6	0.1535/0.6180	1.3
M3: Bayer subpixel	0.0059/0.0238	13.2	0.0746/0.3003	2.7
M4: Green comp. subp.	0.0067/0.0270	11.6	0.0080/0.0322	25.6

added next to it. For each angle increment 20 images were taken, and standard deviation and mean of 20 measured angle values were determined. Figure 3.21 (a) shows the mean of measured angle as a function of the robot mark rotation angle. The precision of the angle, i.e. its standard deviation is displayed in Figure 3.21 (b) wherefrom it can be seen that the standard deviation is everywhere less than 0.12°, and average standard deviation is 0.0797°.

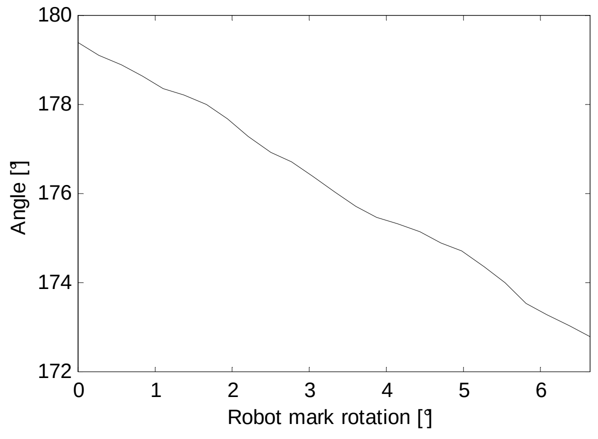
To determine angle accuracy a straight line is fitted through the measured angles. The equation of the line obtained from the experiment shown in Figure 3.21 (a) is -1.0001x+179.4794. Then, the absolute angle error is estimated as the difference of the measured angle mean and the regression line. The results are shown in Figure 3.22. As can be seen, the absolute angle error is almost everywhere less than 0.2°.

Comparison of angle precision and accuracy between different methods is made in similar manner as for position measurement and the results are shown in Table 3.2. As can be seen from the table, the similar observations can be made as for position measurement although the enhancements are slightly lower. Obviously, methods M3 and M4 give far better results than the other two methods, and the method M4 is better than M3, particularly in the accuracy enhancement. Therefore, the usage of method M4 is proposed.

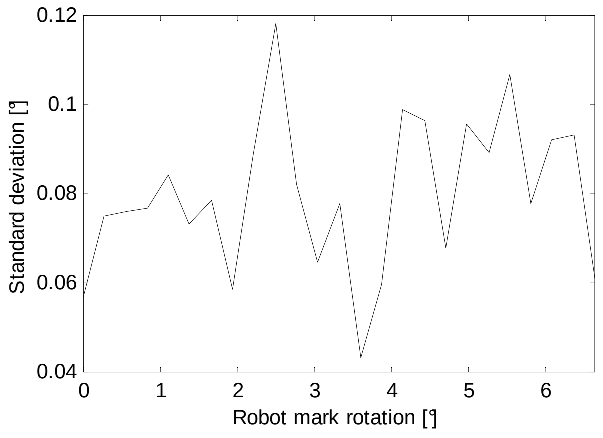
Table 3.2. *Comparison of angle precision and accuracy for different methods*

Method	$\sigma_\theta[^\circ]$	$\sigma_\theta^{(\mathrm{M1})}/\sigma_\theta^{(\mathrm{M}i)}$	$e_\theta[^\circ]$	$e_\theta^{(\mathrm{M1})}/e_\theta^{(\mathrm{M}i)}$
M1: Color tracking	0.6444	1	0.9762	1
M2: Bayer pixel	0.3036	2.1	0.7049	1.4
M3: Bayer subpixel	0.0911	7.1	0.3714	2.6
M4: Green comp. subpixel	0.0797	8.1	0.0831	11.8

Although it is not possible to directly compare the obtained results with results of others because of different camera types, different shapes and sizes of robot marks etc., here a comparison with results presented in referential paper [31] is given, where a very comprehensive analysis of the performance of the color based method is done. Only precision results could be compared, as results for accuracy are not presented there. Standard deviations reported there (σ_y = 0.3432 mm and σ_θ = 0.4011°) are comparable with those obtained with method M1, which is



(a)



(b)

Figure 3.21. *Measured angle mean* (a) *and standard deviation* (b) *as functions of the robot mark rotation*

our implementation of the color based method. Therefore, the proposed method M4 significantly outperforms the method from [31] with about 13 times smaller position standard deviation and about 5 times smaller angle standard deviation.

To investigate performance of the algorithm when robot is moving, a number of experiments were performed with different velocities ranging from 0.1 m/s to 1 m/s. It was observed that standard deviation of the measured velocity is always less than 2 % of actual robot velocity, which is rather small. For the sake of illustration, the result of an experiment is shown in Figure 3.23, where the robot was moving in horizontal direction from the left to the right border of the image with velocity of about 1 m/s with noncalibrated camera. The velocity of the robot is computed by dividing the distance between the current measured

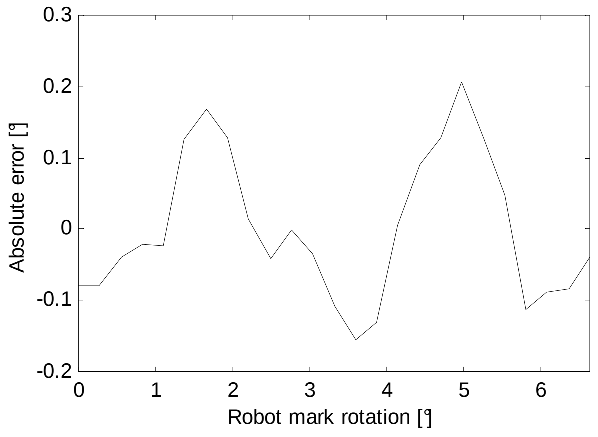


Figure 3.22. *Absolute error of the measured angle*

robot position and measured position in the previous sampling instant with the sampling time $T = 0.0125$ s, without any filtering. As can be seen, measured velocity is lower near the image borders than in the image center, and the main reason is lens distortion. Standard deviation of the velocity computed from 20 images taken near the image center is 0.01325 m/s, i.e. 1.3 % of the actual robot velocity. The variations of the measured velocity are mainly consequence of the image blurring caused by robot motion, lens distortion, and also fluctuations of actual robot velocity during motion.

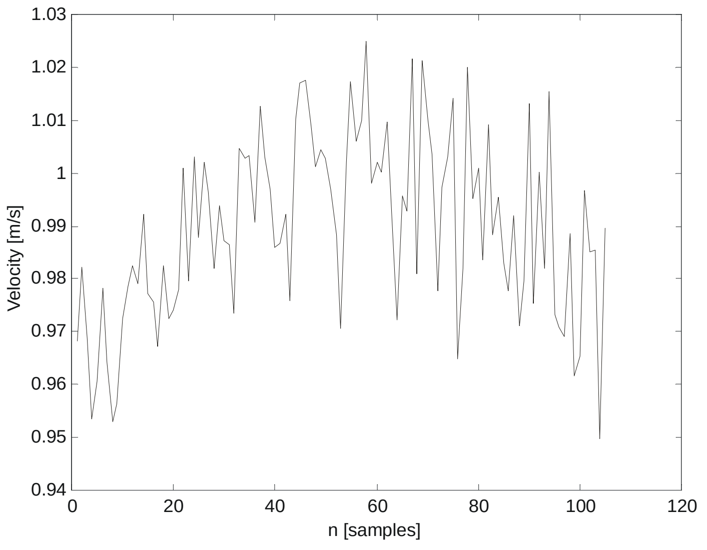


Figure 3.23. *Measured robot velocity*

3.4.2 Analysis of Robustness to Light Intensity Changes

There are two algorithm stages that are particularly sensitive to light intensity changes: thresholding of HSV image to detect color marks and thresholding of gray intensity image to detect white measuring marks. Regarding the first thresholding stage, using of HSV color system by itself guarantees some degree of robustness to light intensity changes, and by the second thresholding stage robustness of white measuring mark detection is achieved using algorithm for threshold adaptation described in 3.3.2.

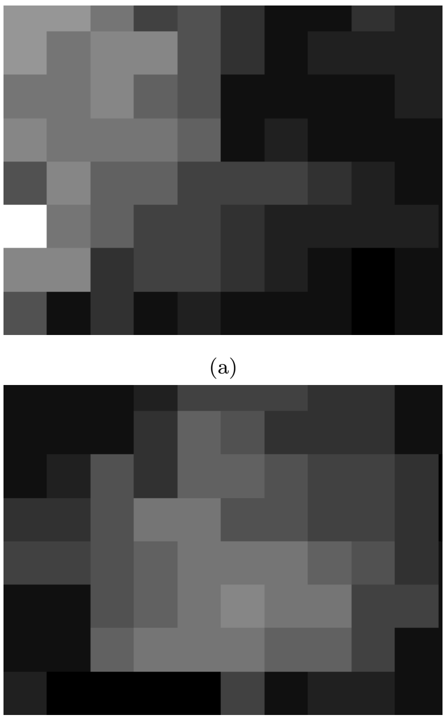
To test robustness of the system to light intensity changes in praxis, experiments with three different light sources were conducted: daylight, fluorescent lamps with daylight spectrum and fluorescent lamps with yellow spectrum. In the first experiment only daylight was used with measured illuminance ranging from approximately 150 to 500 lux. Under such light conditions HSV thresholds had been initially adjusted and were no more changed through later experiments. The algorithm was started and thresholds for white measuring mark detection were automatically adapted. The image obtained with adapted thresholds is shown in Figure 3.24 (a), where brighter areas correspond to image segments with higher thresholds. One can note that light distribution is not uniform and the brightest part is at the left image border. The system was not losing robots in any part of the image despite this nonuniform light distribution.

In the second experiment only fluorescent lamps with spectrum close to daylight were used as light source where the illuminance had its maximum value near the center of the image (140 lux), and the minimum illuminance was near the image border (100 lux). The system adapted successfully to light change and tracking of the robots was reliable in all parts of the image without change of any settings. The obtained distribution of white measuring mark detection threshold is shown in Figure 3.24 (b).

Above described two experiments verify tracking robustness to the wide range of illuminance change, i.e. from 100 to 500 lux. In cases when illuminance was outside of this range too low pixel intensity values or saturation occurred. If wider range of illuminance changes is necessary the camera parameters should be adapted (e.g. gain or shutter) according to illuminance change, and possibly a camera with higher dynamic range should be used. The adaptation of the camera parameters can be done by hardware (usually in the camera) or by software. In this case hardware adaptation is more convenient, because it does not require additional processing time. As the camera used in the experiments does not posses automatic gain feature, and software adaptation would consume too much processing time, camera parameters adaptation is not used so far.

The third experiment was performed with the fluorescent lamps with yellow light in order to test robustness to the light spectrum change. The spectrum change caused the change of the key patch hue component value from 140 to about 125, resulting occasionally with non detected robots in some parts of the image.

Therefore, the vision system is robust to light intensity changes but problems with robustness may arise in cases when the light spectrum changes (e.g. in outdoor environments). This problem is not addressed in this work but it was addressed by a number of researchers, e.g. possible solutions can be found in [29] or [143].


(a)

(b)

Figure 3.24. (a) *Thresholds obtained using daylight;* (b) *Thresholds obtained using fluorescent lamp with daylight spectrum*

3.4.3 Ability to Track Large Number of Robots

As stated in previous sections, the ability to track large (theoretically unlimited) number of robots is guaranteed by using equal robot marks for all robots while the only limitations are available computational power and sizes of the space and the robots. However, there remains potential problem of identification swapping. Therefore in the focus of the experiments was to find out how often identification swapping occurs and if algorithm is capable to recover from such situations. The experiment was performed through many robot soccer games, where 5 robots with equal robot marks were simultaneously on the playground together with 5 opponent robots. Because of the high frame rate used (80 fps) the tracking of robots was reliable and identification swapping did not occur. The only exception is when the vision system loses multiple robots for some reason (typically when robot marks are temporarily occluded by some object). But even in such situations, the identification algorithm is capable to recover correct identifications of robots in only few samples.

3.4.4 Analysis of Real Time Operation Requirement

In order to fulfill real time and high speed requirements, i.e. to enable the tracking of large number of fast moving robots, a special attention is given to the computational simplicity of the algorithms. The achieved execution time of the algorithm on a 3 GHz Athlon 64 processor, when tracking five robots, was only about 1.6 ms in local search mode and about 3.1 ms in global search mode. Therefore the proposed vision system is able to track five robots with framerate of at least 300 fps. In other words, there left enough time for execution of other algorithms (e.g. algorithms for robot mission planning and motion control) within the sampling time of 12.5 ms defined by the framerate of used camera (80 fps).

The find out which stage of the algorithm is most computationally expensive, execution times of various stages of the algorithm were measured. Results for the case where one robot is in the scene are shown in Table 3.3. As threshold adaptation time varies randomly depending on illumination changes, its maximum over multiple samples (about 500 samples) is taken and in this way a worst case is considered. Execution times of other stages are computed as average value over multiple samples, because these times are nearly constant over time. From the table it is evident that in the case of local search most significant part of total execution time refers to other necessary tasks that are not part of the algorithm, such as buffer copying, image flipping etc. Pose measuring task takes more time than robot detection, as it is expected because in local search robot position is known. In global search, the most significant part of the execution time refers to the color classification and robot detection time is significantly higher than the robot pose measuring time. This is expected because the whole image has to be scanned to detect the robot. The fact that in global search mode subpixel robot pose measuring time is minor compared to robot detection time also proves the claim that increased precision and accuracy are accomplished without significant performance loss.

The goal of the next experiment was to find out how the execution time depends on the number of robots, so that execution times for local and global search and for different number of the robots on the playground were measured. The results for local search mode are given in Figure 3.25. If line is fitted through obtained results by means of least squares method, we obtain the following line equation: $t = 0.1606n + 0.7652$, where t is execution time in milliseconds and n is the number of robots. From this equation it is evident that the majority of execution time is constant part, which refers to routines whose execution time is independent on the number of robots, like buffer copying, image flipping etc. It is also obtained that in local search mode all times except t_9 grow nearly linearly

Table 3.3. *Execution times of various stages of algorithm for one robot in milliseconds*

Stage	Local	Global
t_1: Color classification	0.0146	1.5290
t_2: Color patch extraction	0.0192	0.0181
$t_3 = t_1 + t_2$: Robot detection	0.0338	1.5471
t_4: White patch extraction	0.0161	0.0168
t_{5m}: Threshold adaption max.	0.0575	0.0556
t_6: Subpixel measuring	0.0347	0.0353
t_7: Square parameterization	0.0461	0.0439
$t_8 = t_4 + t_{5m} + t_6 + t_7$: Robot pose measuring	0.1544	0.1516
t_9: Other tasks	0.7636	0.7352
$t_{10} = t_3 + t_8 + t_9$: Total execution time	0.9518	2.4339

with the number of robots.

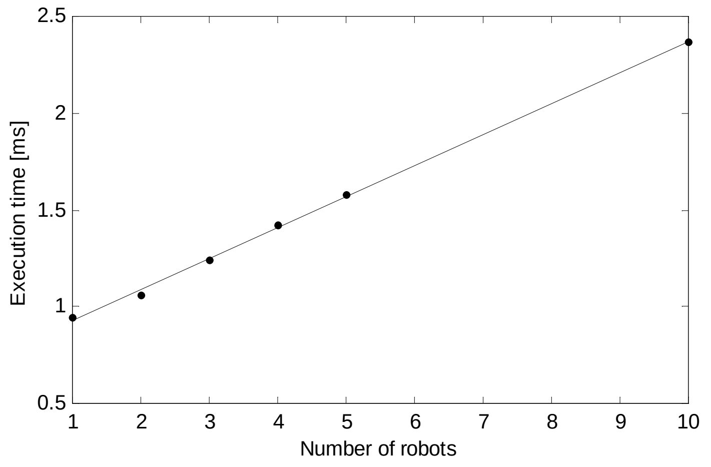


Figure 3.25. *Local search algorithm execution time as function of number of robots*

The results for global image search mode are given in Figure 3.26. If line is fitted through obtained results, we obtain the following equation: $t = 0.1569n + 2.2792$. From this equation, it can be observed that execution time grows with approximately same rate as for local search, but the constant part of the execution time is about 3 times bigger, because the whole image has to be scanned. Using this equation, it can also be estimated that within camera sample time of 12.5 ms, a maximum number of robots that can be tracked is 65. It is observed that in global search mode times t_1 and t_9 are independent on the number of robots, while other times are nearly proportional with the number of robots.

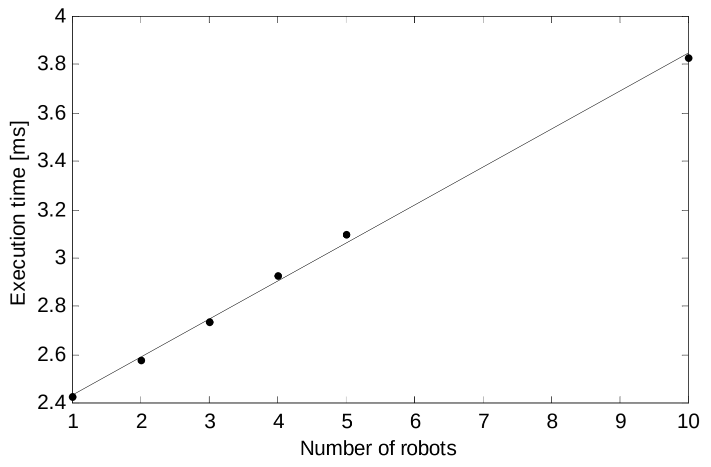


Figure 3.26. *Global search algorithm execution time as function of number of robots*

3.5 Summary

A new global vision system for real time tracking of two-dimensional poses of multiple mobile robots is presented. A novel algorithm is proposed that operates directly in Bayer format image thus enabling high framerates and at the same time high measurement precision and accuracy as it works in subpixel precision. Although the algorithm is highly optimized for Bayer format image, it is not restricted to it and can be easily adapted to work with any image format. High measurement precision and accuracy are verified by carefully performed experiments. Moreover, algorithm is robust to the light intensity variations, and number of robots that can be simultaneously tracked is not limited by the algorithm, but only by available computational power and sizes of the space and robots, so that it is very convenient for tracking robot teams with large number of robots. The system requires easy and short setup process.

It is very important that the above mentioned advantages are not gained by using high cost hardware or algorithms with high computational cost, so that the system price is reasonably low, and high processing speeds can be achieved (more than 80 fps is possible with adequate camera and hardware). The possible drawback of the proposed algorithm is the fact that all robots use equal detection marks, so that in some unpredictable situations (e.g. if robots mutually collide with high velocities), there is still a small possibility of swapping identifications of the robots. Implemented supervision algorithm requires some (but small) time to detect and correct such situations. However, because of high framerate and reliable tracking algorithm those situations occur very rarely.

Camera calibration procedure automation in order to further reduce the system setup time is also planned, although it is still very short comparing to other existing systems. Currently, every camera of the global vision system must be calibrated separately, which is a rather cumbersome task. However, the calibration can be automated by tracking the robots by the distributed cameras (and possibly other sensors) while at the same time estimating the calibration parameters, utilizing the fact that robot's state is continuously changing while moving from one to another camera's field of view. This is sometimes called simultaneous localization and tracking (SLAT) [99].

Besides tracking multiple objects in intelligent spaces, there are many other practical applications of developed vision algorithms, such as robot soccer, estimation of mobile robots odometry parameters using global camera as a reference sensor [5], tracing a marker on visually impaired person's finger to assist him reading tactile environment map [98] or for emulating an input device, like a switch, a joystick or a mouse in human-computer interaction for people with severe movement restrictions [101]. The developed vision algorithm is already successfully applied for calibration of mobile robot odometry parameters in our lab as described by Ivanjko in [62] and [63].

Chapter 4

Mobile Robot Motion Planning

This chapter defines a general motion planning problem and serves as an introduction to later chapters which provide more details about the particular topics. The direct and decoupled approaches to motion planning are introduced. Finally, selection of the motion-planning method used in this thesis is discussed. Previously published works on motion planning include [23] and [25].

4.1 Introduction

Once a robot is localized, i.e. its pose in global coordinate frame is determined, there remains a task of its automated driving to the desired location. This task is called the *mobile robot motion planning*. The directives of the robot are usually given in some kind of a high-level language, which the motion planning system automatically compiles into a set of low-level motion primitives to be accomplished by appropriate feedback controllers.

As a robot actually moves in a physical world, it is subject to physical laws, geometric constraints and uncertainty. Therefore, analysis and design of motion planning algorithms consists of a combination of problems in many scientific disciplines with contributions coming from fields such as robotics, control theory and artificial intelligence. Within field of robotics, the focus is on designing algorithms that generate useful motions by processing complicated geometric models. Here many algorithms from computational and differential geometry are used. Within control theory, the focus is on algorithms that compute feasible trajectories for systems, with some additional coverage of dynamics, feedback and optimality. Analytical techniques are typically used to derive appropriate feedback laws capable of executing desired robot motion. Within artificial intelligence, the focus is on designing systems that use decision-theoretic models to compute appropriate actions.

Pioneer researches in robot motion planning can be traced back to the late

60's (e.g. [113]). Nevertheless, most of the effort started during 80's and is still in progress. The motion planning problem was first extensively studied for the application in industrial manipulator robotics, and lately extended for use in mobile robotics.

It is interesting to compare motion planning algorithms specific to industrial manipulators to those in mobile robotics field. It can be concluded that motion planning in static environment for a manipulator with, for instance, six degrees of freedom is far more complex than that of a differential-drive robot operating in a flat environment [138]. This comes from the fact that the motion planning algorithms used by mobile robots tend to be simpler approximations owing to the greatly reduced degrees of freedom. Furthermore, industrial robots often operate at the fastest possible speed because of the economic reasons. So, the dynamics and not just the kinematics of their motions are significant, further complicating path planning and execution. In contrast, a number of mobile robots operate at such low speeds that dynamics are rarely considered during motion planning, further simplifying the mobile robot instantiation of the problem.

However, once we step out of static worlds, planning the motion of mobile robots becomes much more complex. The key difference is that in mobile robotics tasks are less repetitive and environments are less structured. Such dynamic environments are manifested by the unexpected obstacles that can show up anywhere and anytime in robot's workspace. Those obstacles can be static (e.g. a chair) or moving (e.g. people) which further complicates the problem. To adequately react to those changes in environment, a mobile robot must also incorporate some reactive behaviors based on readings of sensors. Static motion planning that is usually used for industrial manipulator motion planning cannot detect or predict, and therefore cannot react to changes in the environment.

Furthermore, possible motions of the most mobile robots are limited by nonholonomic constraint. It is the consequence of the fact that robot wheels can only roll without slipping so that many mobile robots have the property that they cannot instantaneously slide sideways. From the control point of view, nonholonomic systems are underactuated, i.e. they have less controls than configuration variables. For instance a car-like robot has two controls (longitudinal and angular velocity) while it usually operates in a 3-dimensional configuration space (2-D position and orientation).

The motion planning problem is usually solved by finding a path for a robot from initial pose (configuration) to desired pose (configuration) while avoiding obstacles. In many cases, e.g. when synchronized moving of multiple mobile robots is required, planning only the path (where we worry only about geometric or kinematic issues) is not sufficient, but also the velocity along the path must be planned. This is called a trajectory planning, where we also consider dynamics of the robot (e.g. its maximum velocity and acceleration). Today's research covers many more interesting topics, such as coverage, optimality, uncertainty constraints, etc.

The motion planning algorithms can be characterized according to the task to accomplish, properties of the used mobile robot and properties of the used algorithm. The most important motion planing tasks are *navigation*, *coverage* and *mapping*. Navigation is the problem of finding a collision-free motion for the robot system from initial configuration (state) to the desired configuration. Coverage is the problem of passing a robot (or some sensor or tool mounted on the robot) along all points in a space (mowing, cleaning). Mapping is the problem of exploring and sensing an unknown environment to construct a representation that is useful for navigation, coverage, or localization.

A selection of the most appropriate motion planning algorithm depends also on properties of the robot that will execute the given task. The main characterization of the robots is according to their mobility. In this way robots can be divided to *fixed-base robots* (industrial robot arms) and *mobile robots* (wheeled robots). Also, the robots are characterized by the number of degrees of freedom and the shape of the configuration space. Further, if a robot can move in any direction in its configuration space (when there are no obstacles), it is called *omnidirectional*. On the contrary, if the robot has some velocity constraints, such as a car that cannot move sideways, it is called *nonholonomic*.

Motion planning algorithms can be characterized according to many properties, such as *optimality*, *computational complexity*, *completeness* etc. Algorithm is *optimal* if it finds motions that have minimal length, execution time, energy consumption or any other criterion. Very important property is *computational complexity* of the algorithm. If it is expected that the inputs of the algorithm will vary in size (e.g. number of obstacles), then the algorithm is only considered practical if it runs in time polynomial or better in the inputs. The planner is *offline* if it constructs the motion plan in advance, based on known map or model of the environment, and it is *online* if it incrementally constructs the plan during robot motion. There is no definite distinction between offline and online algorithms, as offline planer can also be used as online if it has low computational time and it is used to re-plan the motion with each new sensor data. The motion planner algorithm can also be divided into *planning* and *reactive* algorithms. With *planning*, the robot uses models of the environment and itself to determine the motion plan to a specified goal in advance—this is the same as offline planning. With *reactive* (sensor-based) control the motion of the robot is conditioned by the current state of the environment based on sensor data, typically for small time period—this is kind of an online planning algorithm. The key advantage of planning is that it enables a robot to achieve complex goals. However, the planning capability has its limitations since environment may not be deterministic in a sense that it is not possible for the robot to predict all future states of both the robot and the environment. Therefore, it becomes apparent that it is best to combine advantages of both planning and reactive control to achieve robust execution of complex tasks.

We say that a planner is *complete* if it always finds a solution if one exists, and otherwise indicates a failure in finite time. In complex environments, as number of degrees of freedom increases, complete solution may be computationally infeasible. In this case the completeness requirements can be relaxed and we may be satisfied with *resolution completeness* (if a solution exists at a given resolution of discretization, the planner will find a solution) or with *probabilistic completeness* (probability of finding a solution, if one exists, converges to 1 as time goes to infinity). Demands such as completeness, optimality and computational complexity are of course contradictory. E.g. planners that are both complete and optimal will have increased computational complexity.

4.2 Common Approaches to Motion Planning

Motion planning is one of the fundamental challenges in robotics so that early studies in this area date from the 1960's. The motion planning problem was originally studied as path planning in field of robotics, but trough this research it has gained many applications in areas such as computer graphics, simulations, geographic information systems (GIS), very large scale integration (VLSI) design, and games. Due to its widespread application, there is a strong interest in industrial and research areas where a multitude of approaches have been proposed.

The Bug algorithm [95] is based on a simple idea of finding a path to the goal by avoiding obstacles by following their contours. In its first version Bug1, the robot circumvents the obstacle in its full contour, finds the point of departure that is closest to the goal position and then departs from the obstacle. In the worst case it can circumvent the object twice in its full size. Although this algorithm is inefficient, it is complete as it guarantees that the robot will reach a goal if it is possible.

The Bug2 algorithm bring some improvements compared to Bug1, as it departs immediately when it is able to move directly towards the goal. This may significantly speed up the traversal to the goal, but the drawback is that there exists situations where robot traversal is non optimal in the direction sense towards the goal. There are several extensions to the basic Bug algorithm, such as the Tangent Bug [68] which uses range sensing by constructing a local tangent graph towards the goal position direction.

The problem of motion planning considered in this thesis is actually trajectory planning problem where robot motion is planned as a function of time. Generally, the trajectory planning problem is to find control inputs (e.g. forces) yielding a trajectory that avoids obstacles, takes the robot to the desired goal state, and perhaps optimizes some objective function. This is a complete motion planning problem, as opposed to a path planning problem that only finds a feasible curve in the configuration space without reference to the velocity. Two approaches to this problem are usually divided into two categories: *direct motion planning* and *decoupled motion planning*, described in the sequel.

4.2.1 Direct Motion Planning

Direct motion planning is method for planning the trajectory directly in the state space. If additionally the trajectories are to be found that optimize some optimality criterion such as motion time or utilized energy, the problem of finding the trajectory is actually optimal control problem. Unfortunately, the optimal control problem is too complex for almost any robot system, and as such cannot be solved analytically. Therefore, a numerical approach must be used. In literature usually two numerical approaches are studied: *nonlinear optimization* and *grid based search*. When optimization is not crucial, *potential field* method and the *Rapidly exploring random tree method* (RRT) are used as well.

If motion planning is viewed as the optimal control problem, the problem can be transformed to a finite-dimensional parameter optimization problem, allowing nonlinear optimization to be used to numerically solve the optimality conditions [38]. If the problem is well formulated (e.g. the objective and constraint functions are sufficiently smooth), nonlinear optimization may result in rapid convergence to a locally optimal trajectory. The drawbacks of this approach are that the method requires an initial guess (possibly provided by another method), and the locally optimal solution reached generally depends heavily on this guess. Also, evaluation of constraint and objective functions, and their gradients, may be computationally demanded.

A grid-based search method enables the user to specify how close the solution should be to time-optimal while avoiding the obstacles. The planned motion is only approximate, but with property that user can determine how large the error in final state can be. The advantage of this approach is that it is global, i.e. it does not require initial guess like nonlinear optimization. The drawback is that the size of the grid grows exponentially in the dimension of the state space so that this approach is not appropriate for high dimensional systems. Unfortunately, both nonlinear optimization and grid based search approaches are usually computationally too complex to be used online.

Artificial potential fields method was first introduced for robotic manipulator arms and later suggested for mobile robot platforms. Drawing from the field theory concept in physics, this method models obstacles as emitting a repulsive force and the goal point as emitting an attractive force on the robot. The robot senses its current configuration, and applies the gradient forces at the actuators, i.e. the navigation is performed by moving the robot so as to minimize the potential energy. In this way a trajectory is implicitly defined by the potential field.

Finally, a Rapidly exploring random tree approach is a probabilistic method that trades off optimality for planner run time. It may be able to quickly find a feasible trajectory that is in no sense optimal.

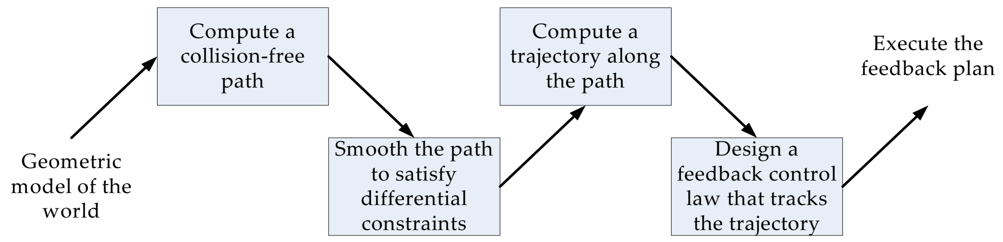

Geometric model of the world
Compute a collision-free path
Smooth the path to satisfy differential constraints
Compute a trajectory along the path
Design a feedback control law that tracks the trajectory
Execute the feedback plan

Figure 4.1. *A decoupled approach to motion planning.*

4.2.2 Decoupled Motion Planning

There are many situations in which direct planning is computationally too expensive due to numerical integration, collision detection etc. In those situations decoupled planning is appropriate because it divides the complicated problem of motion planning into subproblems (or modules) that are easier to solve [89]. This is the reason why this approach is sometimes called a *refinement approach*. A typical decoupled trajectory planning approach consists of four modules (Figure 4.1):

1. Using a path-planning algorithm to find collision-free path.
2. Transforming the obtained path so that velocity constraints (if any, e.g. nonholonomic) are satisfied. In case of a car-like robot this step ensures that the path is feasible for the robot by ensuring that the curvature of the path is never less than the minimum. Here at least a kind of smoothing algorithm is used in most circumstances.
3. Computing a time scaling function so that the path from previous module is time-parameterized while respecting robot actuator limitations.
4. Executing a feedback control law that will ensure tracking of the planned trajectory. The control law should minimize tracking error, i.e. difference between the desired state and measured state of the robot.

Decoupled approach is often used for planning in environments with moving obstacles or control of robot teams, because it decouples high-dimensional problems to multiple low-dimension problems. However, although decoupled approach performs well in many situations, sometimes there arise situations that is decoupled approach unable to solve. For example, typical decoupled schemes are unable to solve a multiple robot planning problem in Figure 4.2.

The main drawback of decoupled approach is therefore its incompleteness—in some situations it may not find a solution even if one exists. The incompleteness also occurs when a module finds a solution that causes the failure in some of the later modules. In other words it is often difficult for modules to take into account

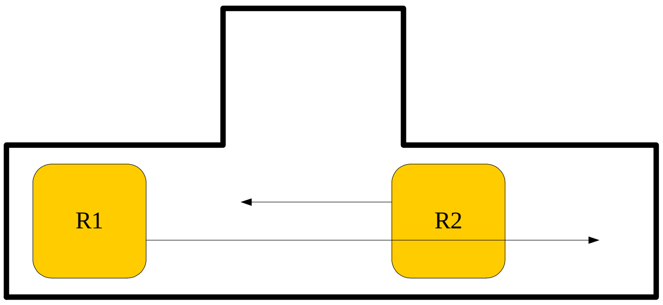


Figure 4.2. *An example of multi-robot planning problem where solution cannot be found by prioritized decoupled approach, although problem has a solution in general. This happens because in prioritized scheme priority is given to either robot robot R1 or R2, while the second robot is neglected.*

problems that may arise later. This is sometimes solved by merging the modules, e.g. there are many methods that solve the first two modules simultaneously.

4.3 Motivation

Regarding the motion planning task, this thesis covers the problem of robot navigation, i.e. finding a collision-free motion from the initial to the desired location. The solution should enable a wide spectrum of applications, ranging from simple load delivery to precise planning required for robot soccer. A robot that is used to accomplish those tasks is a differential drive robot. This is the robot that has two drive wheels and one or more castor wheels that ensure stability of the robot. The presumed operation space is flat environment, and it is assumed that the map of the environment is known (at least partially). The environment is assumed to be dynamic, where obstacles and zones can change shape or move concurrently with the robot so that ability of efficient recomputation of the path is required. It is also assumed that the robot is able to identify its position on the global scale so that the problem of finding a path to a desired goal is well defined.

The majority of approaches currently used in robot navigation are based on a combination of global path planner and local planner that avoids obstacles that were not known at the planning time. A typical example is combination of D* algorithm for global path planning and dynamic window algorithm which is a kind of reactive, direct grid-search algorithm that locally modifies the path in order to avoid obstacles (see e.g. [132]). Due to computational complexity, local planner is typically capable of planning only for a small time period ahead. This is similar to the way that human plans his motion, where he first plans the rooms, passages or doorways through which to pass, and then locally modifies the path according to perceived situation while traversing to the goal.

While this approach satisfies when it is simply required to drive the robot from point A to point B, it has two major drawbacks that limit its use in some applications:

- Even in static environments, it is impossible to precisely plan robot's path, as well as its velocity profile because reactive component is used. This also means that traveling time cannot be predicted and no precise scheduling of robot tasks can be made.
- Due to inability to precisely plan robot motion, it is also not possible to tell the robot how it should enter the goal configuration, e.g. at what velocity and heading direction. In other words, only goal position can be specified. This can be too restrictive in applications such as robot soccer. [1]

Motivation of this work is to overcome these difficulties, as well as to develop a flexible, modular and real-time motion planner. Here flexibility means that the method must be easily adaptable to environments of different complexity and structure, different tasks, multi-robot planning etc. This means that the planner, with little modification, must be applicable for various tasks in many possible variants of intelligent spaces e.g. cleaning, entertainment (robot soccer, robot dance), automated warehouses etc.

Particularly, the developed motion-planning algorithm must have the following capabilities:

- leading the robot to the desired state (e.g. position, orientation, velocity) in near-minimum time, while respecting actuator limits of the robot;
- reacting to changes in the environment;
- fast enough execution for online application.

4.4 Choosing an Adequate Planning Method

Although direct motion planning methods are superior over the decoupled approaches, they are still not not applicable for long-term online planning, mainly due to their high computational burden. Therefore the decoupled approach is selected as the method of choice because it decomposes complex motion planning problem into simpler subproblems allowing a real-time execution. Besides, a very important advantage of the decoupled approach is its flexibility, manifested by easy to achieve software modularity and reusability. Each module is a separate component that can easily be replaced or reused with other robot types or for other robot tasks. This is important because mobile robots are expected to spread

[1]If differential drive robot is used, it can attain the desired heading at the goal by simply reorientating itself after reaching the goal. However, this requires additional time.

all over our homes and offices—in that sense a motion planner that is hard to adapt to other tasks is of little practical use because robot destiny will be to accomplish very distinctive missions.

As mentioned, the main drawback of decoupled approach is its incompleteness. However, in this work this problem is at least alleviated by extending classical decoupled architecture in Figure 4.1. This is obtained by avoiding strict algorithm flow from the previous to the next module—in the developed method algorithm execution can flow back and forth as necessary because modules are capable of calling functions from other modules. In this way it became possible for a module to ask other modules for instructions for further planning or rate the current plan, so that it can be corrected as necessary.

In this way loss of completeness of the decoupled approach will manifest only in some special circumstances, e.g. in multi-robot example in Figure 4.2. Despite that fact, the predominant algorithms for coordinating teams of robots are still decoupled and prioritized. In recent time some efficient methods that can overcome this problem have been developed, such as by Bennewitz et al. [14], where a method for finding and optimizing priority schemes for multiple robots is described.

By a decision to use the decoupled approach, the selection job is in no way finished. There still left to select appropriate planning methods in each particular module. Those decisions will be guided by the goal of best utilization of intelligent space advantages, as will be described in the following chapters.

Chapter 5

Path Planning

Path planning while avoiding obstacles has long time been the main goal within motion planning research community, while in this thesis it is part of the first stage of the overall motion planning algorithm. Therefore, this chapter is concerned with methods used to plan obstacle-free paths between two robot configurations. A novel path-planning algorithm is introduced that enables fast path-replanning.

5.1 General Notions

The first stage of the decoupled approach is path planning. Path planning still remains one of the core problems in modern robotic applications. The basic path-planning problem is concerned with finding a good-quality path from a source point to a destination point that does not result in collision of the robot and obstacles.

Hereby, a robot is considered a rigid object capable of moving in a physical space called *workspace* $\mathcal{W}$, which is usually planar ($\mathbb{R}^2$) or three-dimensional $\mathbb{R}^3$. Although the robot always moves in the three-dimensional space, the third dimension (height) is often not considered in algorithms as the robot is constrained to move in the ground plane. This is also assumed throughout this work. The workspace of the robot often contains obstacles. Let $\mathcal{WO}_i$ be the closed set that represents the i-th obstacle in the robot workspace, where $\mathcal{WO}_i \in \mathcal{W}, \forall i \in [1, n]$. The *free workspace* is then defined as the set of points $\mathcal{W}_{free} = \mathcal{W} \setminus \bigcup_i \mathcal{WO}_i$.

Motion planning is usually not performed in the workspace but in the so called *configuration space* $\mathcal{C}$ of the robot (also called C-space). This is the set of all robot configurations, where the robot configuration q contains a complete specification of the position of every point of the robot system. E.g. for a mobile robot that can translate and rotate in a planar workspace, q contains position (x, y) and orientation θ. Therefore, its configurations space is an open unbounded environment given by $\mathbb{R}^2 \times S^1$ where the $\times$ represents Cartesian product and S^1

is the unit circle representing angles in range $[-\pi, \pi]$.

Let $R(q)$ be the set of points of the workspace occupied by the robot at configuration q. Then the i-th obstacle in the configuration space $\mathcal{CO}_i$ corresponds to the configurations of the robot that intersect an obstacle in the robot workspace $\mathcal{W}$, i.e. $\mathcal{CO}_i = \{q|R(q) \bigcap \mathcal{WO}_i \neq \emptyset\}$. Now the *free configuration space* can be defined as

$$\mathcal{C}_{free} = \mathcal{C} \backslash \bigcup_i \mathcal{CO}_i, \tag{5.1}$$

and is often referred simply as "free space". The configurations in the free space are called *free configurations* or *admissible configurations.*

Therefore, a path can be described as a continuous curve on the configuration space, i.e. it is a continuous function that maps some path parameter to a curve in $\mathcal{C}_{free}$. The path parameter can be chosen arbitrarily; in this work an interval $[0, s_g]$ is chosen, where s_g is parameter s at goal configuration. Therefore, a path can be written as a continuous function from the initial configuration q_{start} to the goal configuration q_{goal} such that

$$q(s) : [0, s_g] \to \mathcal{C}, \text{where } q(0) = q_{start}, q(s_g) = q_{goal} \text{ and } q(s) \in \mathcal{C}_{free} \quad \forall s \in [0, s_g]. \tag{5.2}$$

By definition, every configuration along the path is free, so that such a path is called *free path* or *admissible path.* If a path touches obstacles, but does not penetrate them, it is no more determined in free configuration space, but in its closure $\mathrm{cl}(\mathcal{C}_{free})$. Such a path is called *semi-free path.*

The main difference between path planning and trajectory planning is that the path planning only takes into account time-independent constraints such as geometric constraints (i.e. obstacles) and kinematic constraints (i.e. curvature constraint), but not time dependent (dynamic) constraints such as limits of robot velocity and acceleration. If moving obstacles have to be taken into account, obstacle avoidance is no more pure geometric constraint because obstacle positions must be treated as time-dependent variables. Therefore path-planning techniques deal only with static obstacles. Nevertheless, in some circumstances those methods can be adopted so that moving obstacles are directly taken into account. This is usually achieved by adding time as an additional configuration variable.

Numerous methods are proposed that solve the path-planning problems. Depending on the amount of information available about the environment, which can be completely or only partially known, the approaches vary considerably. In many path-planning algorithms computational geometry plays a special role. Many methodologies that rely on geometric representation of the space, have their roots in computational geometry. Path-planning problems that are solved using these methodologies usually have a well-defined and deterministic set of objectives, regular geometric space representation, and specific functions that describe robotic movements.

The common methods that are based on computational geometry are the cell decomposition methods, the roadmap methods (and the sampling-based methods as a special case), and the artificial potential field methods [38]. If robots are represented by polygonal objects, an approach based on the Minkowski sum is often used [73]. It must be emphasized that this classification is somewhat arbitrarily because many methods cannot be classified exactly to certain category as they combine algorithms and principles from multiple categories. In continuation it is shown how the free configuration space is geometrically constructed, which is a prerequisite for some methods. Thereafter, description of the most important path-planning methods and a literature review is given.

5.2 Free Configuration Space Construction

As motion planning is usually executed in the configuration space, many path-planning methods depend on explicit construction of the free configuration space $\mathcal{C}_{free}$ of the robot. The configuration of a robot system is a complete specification of the position of every point of that system, which is needed to ensure that no point on the robot collides with an obstacle. The dimension of the configuration space, i.e. minimum number of parameters needed to specify the configuration depends on the type of a robot, and is equal to the number of degrees of freedom of the robot.

This can be illustrated by considering a simple mobile robot that can translate without rotating in the plane. A common way to represent robot configuration is to specify the location of its center, (x, y) relative to some fixed coordinate frame. If radius r of the robot is known, the set of points occupied by the robot can be easily determined from the configuration $q = (x, y)$ and is denoted as $R(q)$. Therefore, the configuration space of this robot can be represented by $\mathbb{R}^2$. As this robot operates in a two-dimensional Euclidean ambient space (i.e. workspace) also represented by $\mathbb{R}^2$, one may assume that configuration space and workspace represent the same spaces. But those are different spaces, as will become evident in the next example.

Apart from translation, mobile robot are usually capable of rotating, so let's consider a more complex robot that can translate and rotate in a planar workspace. To specify configuration of this robot, it suffices that q contains robot position given by the coordinates (x, y) and the orientation given by the angle θ with respect to the workspace frame. Then the triple $q = (x, y, \theta)$ completely determines position of any point of the robot relative to the workspace coordinate frame because the robot is a rigid body. This configuration space is denoted by $\mathbb{R}^2 \times S^1$. The workspace of this robot is again $\mathbb{R}^2$, but it is now evident that it is different from its configuration space, $\mathbb{R}^2 \times S^1$. Properties of the different configuration spaces can be described using tools from mathematic field called topology, see e.g. [89].

The free configuration space $\mathcal{C}_{free}$ is in equation (5.1) defined as the set of

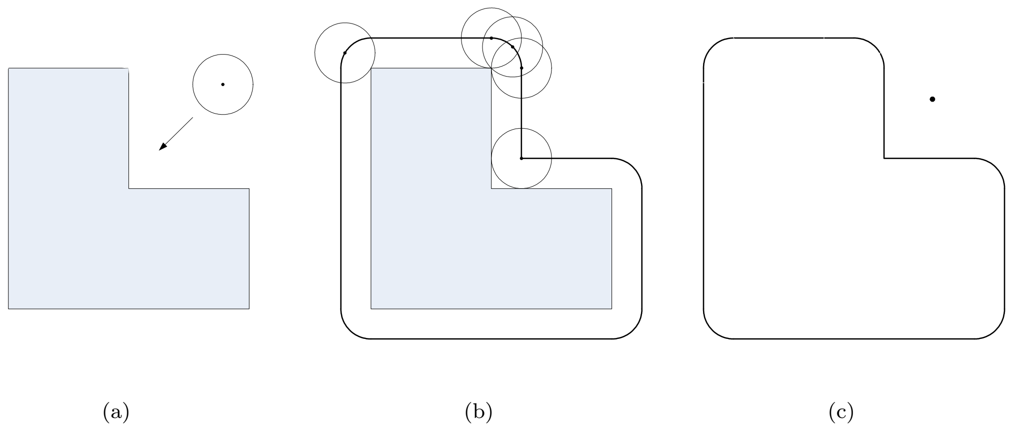


Figure 5.1. *Construction of the free configuration space.* (a) *The circular mobile robot approaches the workspace obstacle.* (b) *The robot slides around the obstacle touching it. Robot path coincides with the contour of the configuration space obstacle (the thick line). The configuration space obstacle therefore corresponds to the Minkowski sum of the workspace obstacle and the robot.* (c) *Problem has been transformed into motion planning for a point robot in the configuration space.*

configurations at which the robot does not intersect any obstacle. To plan a path, it is necessary to construct this set, i.e. map obstacles from robot workspace into its configuration space. A simple example of the free C-space construction is for the circular mobile robot in the planar workspace with a single polygonal obstacle (Figure 5.1). The configuration space obstacle is obtained by an isotropic growth of the workspace obstacle by the radius of the circular robot, which is actually the Minkowski sum of the obstacle and the disc. Even in this simple example it can be noticed that configuration space obstacles can have more complex geometric shape than corresponding workspace obstacles. In particular case the C-space obstacle contains also circular segments which were not present in the workspace obstacle. As the robot in this example is circular, its geometric representation is rotation invariant. Therefore, the shape of the free C-space does not change along the third dimension and planning can be performed in first two dimensions only.

Let's now consider a more general case: a mobile robot of arbitrary shape that can translate and rotate in the planar workspace. Whether or not is a particular point in the C-space free depends now also on robot orientation resulting in true 3-dimensional free configuration space. Because of this the motion planning can no more be reduced to two dimensional C-space, but has to be performed in 3 dimensions. This results in two main consequences. First, if motion planning is to be performed in environment with moving obstacles, the 3-D free configuration space has to be regularly updated, which is a complex operation. Second, motion planning in three dimensions is also much more complex than in two dimensions.

Both consequences contribute to significantly higher algorithm execution time.

This is the reason why in many applications the non-circular mobile robot is approximated by its bounding circle. Unfortunately, even if a complete path-planning algorithm is used, such an approximation makes it incomplete, meaning that the planner might not find a path even if one exists. However, this may not be considered as a practical limitation until e.g. precise manipulation tasks are demanded that require motion very close to the obstacles. Moreover, many commercial differential-drive mobile robots (e.g. Pioneer [2]) are approximately circular.

In applications where the approximation by a bounding circle is too restrictive, a multiple-planner strategy can be used. In this approach a high-level decision algorithm makes a selection of the the most appropriate planner in a particular situation. E.g. roadmap methods (Section 5.6) can be used for fast, near-optimal motion planning in open environments. When it comes to navigation through very narrow passages or precise manipulation tasks, sampling-based planners (Section 5.5) that do not depend on explicit configuration space construction are suitable. The sampling-based planner will produce a path that may not be optimal, however, optimality is not of primary concern in manipulation tasks, and non-optimal solution is better than no solution at all.

As it was shown, for the case of the circular mobile robot in a planar world populated with polygonal obstacles, it is easy to explicitly construct free C-space by dilating the workspace obstacles. When the robot is even slightly more complex, it becomes much more difficult to do so. This process is sometimes simplified by using grid-based representation of the configuration space. For the mentioned mobile robot of arbitrary shape that translates and rotates in the planar workspace, the configuration space $\mathcal{C} = \mathbb{R}^2 \times S^1$ can be partitioned to 3-dimensional grid. For each point in this grid we can perform a simple test to see if the corresponding configuration is in collision with any of the obstacles; in this case the corresponding grid cell can be labeled as "one", or as "zero" otherwise. Then the graph search algorithm can be used to find a path from start to goal configuration. However, this approach is not very practical for large robot workspaces as the 3-D grid can become very "memory-hungry". The memory requirements further depend on selected space and orientation resolution; finer the resolution—higher the memory requirements. Therefore, the grid partitioning approach is more commonly used with circular-shaped robots, where the resulting grid map is only two-dimensional.

5.3 Artificial Potential Fields

As it is not simple to explicitly represent the configurations space, especially for robots with many degrees of freedom, an alternative is to use methods that incrementally explore the free space while searching for a path. Example of this approach is the artificial potential field method, which was introduced by Khatib [75] for robotic manipulator arms and later suggested for mobile robot platforms (see e.g. [88, 18]).

The idea is to assign a function similar to the electrostatic potential to each obstacle and then derive the topological structure of the free space in the form of minimum potential valleys. The robot is pulled toward the goal configuration as it generates a strong attractive force. In contrast, the obstacles generate a repulsive force to keep the robot from colliding with them. The path from the start to the goal hopefully can be found by following the direction of the steepest descent of the potential toward the goal. This can be viewed as a landscape where the robot moves from a high-value state to a low-value state, i.e. the robot follows a path "downhill" by following the negated gradient of the potential function.

The environment for the original formulation of this idea is assumed to be static, however there have been adaptations for using this approach for dynamic environments. Potential can be associated with the objects in the environment as they are encountered. Various variants of the artificial potential field method have been developed to make the approach usable in dynamic or cluttered or partially known environments.

The potential field experienced by the robot at configuration q can be expressed as

$$U_{apf}(q) = U_{goal}(q) + U_{obs}(q). \tag{5.3}$$

where $U_{apf}(q)$, $U_{goal}(q)$ and $U_{obs}(q)$ denote the artificial potential field, the attractive potential from goal and the repulsive potential from obstacles, respectively. A related artificial force $F(q)$ is then obtained as negative gradient of the potential field $U_{apf}(q)$ as

$$F(q) = -\nabla U_{apf}(q). \tag{5.4}$$

Typically, obstacles are treated as exponentially repulsive bodies so that the repulsion experienced by the robot rises exponentially as it approaches the boundary of an obstacle at which point the force becomes practically infinite:

$$U_{obs}(q) = \log|\rho(q)| + \frac{1}{\rho(q)^2}, \tag{5.5}$$

where $\rho(q)$ is the distance between the robot and an object. The goal is typically chosen to have a parabolic well shaped attractive force such as

$$U_{goal}(q) = K|q - q_{goal}|^2, \tag{5.6}$$

where K is a positive constant.

The main advantage of the potential field method is that information on the locations of all obstacles is not required beforehand so that path planning can be done in real time by considering only the obstacles close to the robot. Besides, it is extremely easy to implement. However, as only local properties are used in planning, the robot may get stuck at local minima and never reach the goal. There are some adaptations of the original methods that use random walk sequences to escape from local minima traps. But the heavy amount of parameter tuning caused most people to abandon the method in recent times, in favor of newer methods.

5.4 Cell Decompositions

The cell decomposition method uses nonoverlapping cells to represent the free space connectivity. The decomposition can be exact or approximate. An exact decomposition divides the free space into regions called cells whose union is exactly the free space [10, 38]. The cells can be of various shape. The shared boundaries of cells often have a physical meaning such as a change in the closest obstacle, a change in line of sight to surrounding obstacles or some other change in the constraints applying to the motion of a robot. Two cells are adjacent if they have a common boundary. Therefore an adjacency (or connectivity) graph is constructed which stores the information about adjacency relationships of the cells, where a node corresponds to the particular cell, and edge denotes the adjacency between cells, i.e. ability to generate the path between corresponding cells (example: a straight line). Once the decomposition is computed, i.e. the space is divided up into cells, the adjacency graph that comes out gives all the regions that need to be traversed to get from initial to goal configuration. The path planning is then usually done in two steps:

1. The planner determines the cells that contain initial and goal configuration, respectively;
2. The planner searches for a path in the adjacency graph.

There exist many techniques for decomposition. E.g. for convex polygonal obstacles a typical sample algorithm is known as the sweep line algorithm that sweeps through the vertices of all obstacles and splits the cells according to local obstacle edge directions [88]. Other examples include triangulation, cylindrical decomposition and 3D vertical decomposition [89].

An approximate cell decomposition treats a set of cells which approximately covers free space. This scheme samples the free space with cells, where cells have regular boundaries and thus it is easier to compute traversals, but a lot of details about the environment may be lost. Specifically, a detail that is smaller than half the smallest dimension of the cell would not be captured at all. This method usually decomposes the free space recursively, stopping when a cell is entirely in free space or entirely inside an obstacle. Otherwise, the cell is further divided. Because of memory and time constraints, the recursive process stops when a certain degree of accuracy has been reached. An example is the quadtree algorithm [112]. The computational efficiency of this method depends on the fineness of the decomposition. The finer decomposition, the more paths are created, and the closer to the shortest path in the network.

An important decomposition that is frequently used in practical applications is the occupancy grid map which also can be classified in the category of approximate methods. The occupancy grid method covers the space with a regular grid and then determines whether a cell is free or occupied based on the presence or absence of an obstacle. Since in motion-planning problem one is basically concerned whether an object occupies a cell or not, the representation can be a simple histogram of sensor hits with respect to a certain cell. A threshold value is used to filter out false noisy obstacle detection. In order to discourage robot movements close to the obstacle contours, a certain mask operator is applied to the occupied cells which propagates the increased costs in the case when graph search algorithm are used to find the global path. The global path is usually found using some variant of D* graph search algorithm, which allows efficient path replanning. One such example is an integration of focused D* and Witkowski's algorithm which produces the shortest path in occupancy grid maps [133].

The cell decomposition method, although simple to implement, seldom yields high-quality paths. The exact cell decomposition technique is faster than the approximate one, but the path obtained is not optimal. The approximate cell decomposition can yield near-optimal paths by increasing the grid resolution, but the computation time will increase drastically. There is also the known problem of digitization bias associated with using a grid. This stems from the fact that while searching for the shortest path in a grid, the grid distance is measured and not the Euclidean distance.

However, cell decompositions distinguish themselves from other methods in that they can be used to achieve coverage. A coverage path planner ensures that an effector (e.g. a robot, a tool, etc.) passes over all points in a free space. Once the robot visits each cell, the coverage is achieved.

5.5 Sampling-Based Algorithms

As the dimension of the configuration space grows, path planners that depend on explicit representation of the configurations space become impractical. In those cases sampling-based methods [38] have proved as very promising as they can be used to solve very complex path-planning problems. Those planners do not attempt to explicitly construct the boundaries of the configuration space obstacles or represent cells of free configuration space. Instead, they conduct a search that probes the C-space with a sampling scheme using a procedure that can decide whether a given configuration of the robot is in collision with the obstacles or not. Therefore, efficient collision detection procedures ease the implementation of sampling-based planners and increase the range of their applicability. Furthermore, since collision detection is considered by a motion-planning algorithm as a "black box", collision detection module can be designed for specific robots and applications. The collision detection module handles concerns such as whether the models are semi-algebraic sets, 3D triangles, nonconvex polyhedra, and so on. Recent advances in collision detection algorithms have contributed heavily to the success of sampling-based planners. Sampling-based methods use different strategies to generate collision-free samples (i.e. configurations) and to connect the samples with paths to obtain feasible solutions to path-planning problems.

This general philosophy has been very successful in recent years for solving complex problems that involve thousands and even millions of geometric primitives. Such problems would be practically impossible to solve using techniques that explicitly represent C-space [89]. Sampling-based methods are often used to solve complex path-planning problems in industrial automation, bat are also used to solve problems beyond classic path planning. For example, sampling based planner can be used to determine if a part can be removed from an aircraft engine during construction phase using a CAD (computer-aided design) model of an engine. This information is extremely important to verify correct design of the engine, as some parts need to be removed and replaced for maintenance. Sampling-based planners can also be used in computer animation (e.g. to plan movements of a human model), planning with kinematic and dynamic constraints etc. Sampling-based algorithms and their applications are still under intensive research.

The Probabilistic RoadMap planner (PRM) [72] was one of the early approaches that demonstrated the tremendous potential of sampling-based methods. This planner fully exploits the fact that it is cheap to check if a certain configuration is in free configuration space or not. PRM uses coarse sampling to generate nodes of the path and very fine sampling to connect those nodes. In this way a roadmap is constructed (usually offline) which can be used later to find a path between user-defined initial and goal configurations (possibly online). Initially, node sampling in PRM was done by using a uniform random distribution (basic PRM). This planner is probabilistic complete and worked very well for a wide variety of problems. It was also observed that many other sampling schemes (quasirandom sampling, sampling on a grid) can be used to effectively solve many classes of problems.

PRM is intended for multiple-query problems (the constructed roadmap is used many times later). If PRM is used to answer a single query, some modifications can be made to optimize it: the initial and goal configurations are added to the roadmap nodes, and the construction of the roadmap is done incrementally and is stopped when the solution is found. However, the faster planners exist for single-query problems, e.g. Expansive-Spaces Tree planner (EST) and the Rapidly-exploring Random Tree planner (RRT).

In general, sampling-based methods, being probabilistic in nature, do not meet any optimality criteria. Further, execution time may vary considerably between different queries so that it is hard to achieve real-time performance. However, sampling-based planners have been successfully used for many complex problems where real-time issues are not of primary concern. Despite their simplicity, sampling-based planners are capable of dealing with robots with many degrees of freedom and with many different constraints. They can take into account kinematic and dynamic constraints, closed-loop kinematics, stability constraints, reconfigurable robots, energy constraints, contact constraints, visibility constraints and others. The PRM planner can also be used for dynamic path planning, as its complexity depends mostly on the difficulty of the path and to a much lesser extent on the global complexity of the environment or the dimension of the configuration space. For example, a recent PRM-based algorithm that can efficiently re-compute paths in dynamic environments can be found in [13].

5.6 Roadmap Methods

Data structure that models robot environment is called map, and mapping is the task of generating such models from sensor data. In the context of indoor systems, three map concepts prevail: topological, geometric, and grids [38].

Topological representations aim at representing environments with graph-like structures, where nodes correspond to "something distinct" and edges represent an adjacency relationship between nodes. For example, places may be locations with specific distinguishing features, such as intersections and T-junctions in an office building, and edges may correspond to specific behaviors or motion commands that enable the robot to move from one location to another, such as wall-following.

Geometric models use geometric primitives for representing the environment. Mapping then amounts to estimating the parameters of the primitives to best fit the sensor observations. In the past, different representations have been used with great success. Many researchers use line segments to represent parts of the environment, e.g. [7]. Popular approaches also represent three-dimensional structures of the environment with triangle meshes.

Occupancy grids are grid structures, where the value of each pixel corresponds to the likelihood that its corresponding portion of workspace or configuration space is occupied [46]. Occupancy grid maps were first introduced for mapping of robot environment using ultrasonic sensors.

A class of topological maps that attempt to capture the free-space connectivity with a graph are called roadmaps [34]. This roadmap graph is like network of 1D curves or lines (roads). Roadmap graph also has physical meaning because roadmap node corresponds to a specific location and an edge corresponds to a path between neighboring locations. Therefore it could be viewed as decomposition of the robot configuration space based on obstacle geometry. Planning with roadmaps is similar to the way people use highways. Instead of planning every possible side-street path to a destination, people usually plan their path to a network of highways, then along the highway system, and finally from the highway to the destination. Therefore, the bulk of motion occurs on the highway system. Similarly, the roadmap-based planner first constructs a collision-free path from initial configuration to the roadmap. Then it performs a graph search and finds the path to the vicinity of the goal, and then departs from roadmap and constructs the path to the goal configuration. The most of the motion occurs on the roadmap so that searching does not occur in a multidimensional configuration space or workspace. Here the challenge is to construct roadmap that enables the robot to reach any goal in the free space, while minimizing the length of roads or minimizing some other criterion.

Roadmaps are appropriate if numerous start-goal queries are given to the algorithm, while keeping the robot model and obstacles fixed. This leads to a multiple-query version of the motion planning problem. In this case, it makes sense to invest substantial time to preprocess the models so that future queries can be answered efficiently. Intuitively, the paths on the roadmap should be easy to reach from each initial and goal configuration, and the graph can be quickly searched for a solution. If the obstacles are not fixed (dynamic environment), a roadmap has to be efficiently updated, or a new roadmap must be constructed in each algorithm sample. In the later case, there is of course no sense to construct a whole roadmap, but only a part which is required to connect start and initial configuration. Then it makes sense to optimize roadmap construction algorithms so as to make them more appropriate for single query usage.

Formally the roadmap can be defined as [38]: *A union of one-dimensional curves is a* ***roadmap*** *RM if for all q_{start} and q_{goal} in $\mathcal{C}_{free}$ that can be connected by a path, the following properties hold:*

1. ***Accessibility****: there exists a path from $q_{start} \in \mathcal{C}_{free}$ to some $q'_{start} \in RM$;*
2. ***Departability****: there exists a path from $q'_{goal} \in RM$ to $q_{goal} \in \mathcal{C}_{free}$;*
3. ***Connectivity****: there exists a path in RM between q'_{start} and q'_{goal}.*

By satisfying those properties, a roadmap provides a discrete representation of the continuous motion planning problem without losing any of the original connectivity information needed to solve it. Those properties ensure completeness of the roadmap algorithms; the first two conditions ensure that any query can be connected to roadmap, and the third condition ensures that the search always succeeds if a solution exists.

Sampling methods, such as PRM, previously discussed in Section 5.5, can also be classified as roadmap methods as they represent the free space connectivity with a graph whose vertices are generated randomly in free space and connected to the neighboring vertices such that the connecting edges do not cross any obstacle, which is actually a roadmap. The probabilistic aspect, however, is not important to the method, but is relevant only to how the method determines roadmap nodes and edges.

Many types of the roadmaps can be found in literature, among them the most important are visibility maps, deformation retracts and silhouettes. Visibility maps are typically used for models with polygonal obstacles, where the edges of the corresponding graphs, as the name suggests, are defined between the vertices that are visible from each other, and the nodes of the graph are the vertices of the obstacle polygons. Deformation retractions can be viewed as first creating a cell decomposition of free space from which the roadmap is generated. A typical example is a Voronoi diagram based method. In continuation two examples of roadmaps will be described: visibility graphs and Voronoi diagrams that achieve roadmaps with dramatically different types of roads. In the case of the visibility graph, roads come as close as possible to obstacles and resulting paths are minimum-length solutions. In the case of the Voronoi diagram, roads stay as far away as possible from obstacles.

5.6.1 Visibility Graph

The visibility graph has the properties that its nodes share an edge if they are within line of sight of each other, and that all points in the free space of the robot are within line of sight of at least one node of the graph. The second property ensures accessibility and departability—the inherent properties of all roadmaps. The idea was first introduced by Nilsson [113], which is maybe the first work that concerns the path-planning problem. The visibility graph enables to find a shortest path. Such a path is only semi-free—the robot is allowed to "touch" the obstacles, but it is not allowed to penetrate them. To use such paths in praxis, where position uncertainty is present, they need to be transformed in some way so that they come close to obstacles but do not make a contact.

Problems of computational visibility found in the literature vary in form (see [52] for an extensive survey). Among the two-dimensional spaces, the problem is sometimes restricted to visibility in a simple polygon (no obstacles). More generally, there can be obstacles, sometimes called holes or islands. The obstacles can be restricted to special shapes, such as rectilinear, circular, line segments, or convex polygons; or they can be more general, such as simple polygons. The former case is often encountered in path planning, as in the real world obstacles are often nonconvex.

For path-planning purposes, the visibility graph is usually defined in a two-dimensional configuration space where obstacles are represented as simple polygons [38]. The nodes v_i of the visibility graph include the start location, the goal location, and all the vertices of the configuration space obstacles. The graph edges e_{ij} are straight-line segments that connect two line-of-sight nodes v_i and v_j. Nodes and edges are embedded in the free space and edges of the polygonal obstacles can also serve as edges in the visibility graph. The visibility graph can be searched for the shortest path using the Euclidean distance norm. The visibility graph can also be defined for a three dimensional configuration space with

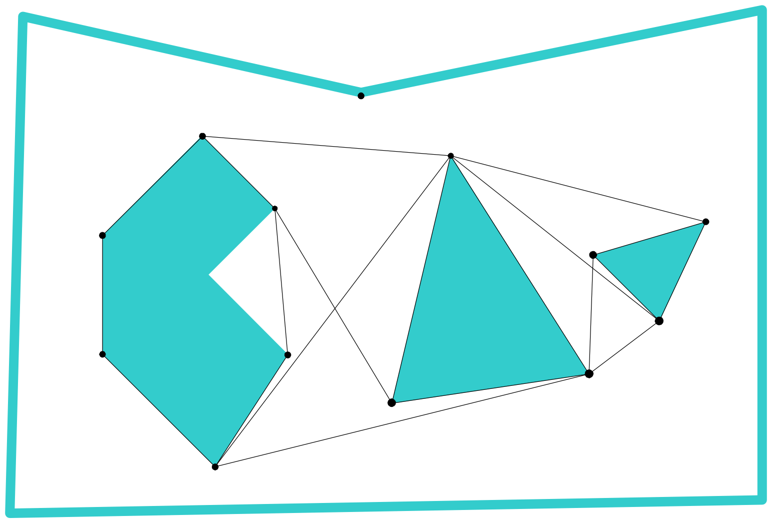


Figure 5.2. *The tangent visibility graph contains edges between reflex vertices and bitangent edges.*

polyhedral obstacles, but in this case it may not contain the shortest path.

In order to construct a shortest path it is not required to connect all mutually visible vertices within roadmap. Such visibility graph is called *tangent visibility graph*, *reduced visibility graph* or *shortest path roadmap*. The tangent visibility graph $\mathcal{G}$ is constructed as follows [89]. Let a reflex vertex be a polygon vertex for which the interior angle (in $\mathcal{C}_{free}$) is greater than π. All vertices of a convex polygon (assuming that no three consecutive vertices are collinear) are reflex vertices. The vertices of $\mathcal{G}$ are the reflex vertices. Edges of $\mathcal{G}$ are formed from two different sources:

- Consecutive reflex vertices: If two reflex vertices are the endpoints of an edge of $\mathcal{C}_{obs}$, then an edge between them is made in $\mathcal{G}$.
- Bitangent edges: If a bitangent line can be drawn through a pair of reflex vertices, then a corresponding edge is made in $\mathcal{G}$. A bitangent line is a line that is incident to two reflex vertices and does not poke into the interior of $\mathcal{C}_{obs}$ at any of these vertices. Furthermore, these vertices must be mutually visible from each other.

An example of the tangent visibility graph is given in Figure 5.2. It can be noticed that roadmap can have isolated vertices, like at the top of the figure. Furthermore, between two disjoint convex obstacles there are exactly four bitangent segments.

The path from initial configuration q_{start} and goal configuration g_{goal} is obtained by inserting q_{start} and q_{goal} as nodes to the graph and then connecting new nodes to all visible vertices; this is shown in Figure 5.3. This makes an extended

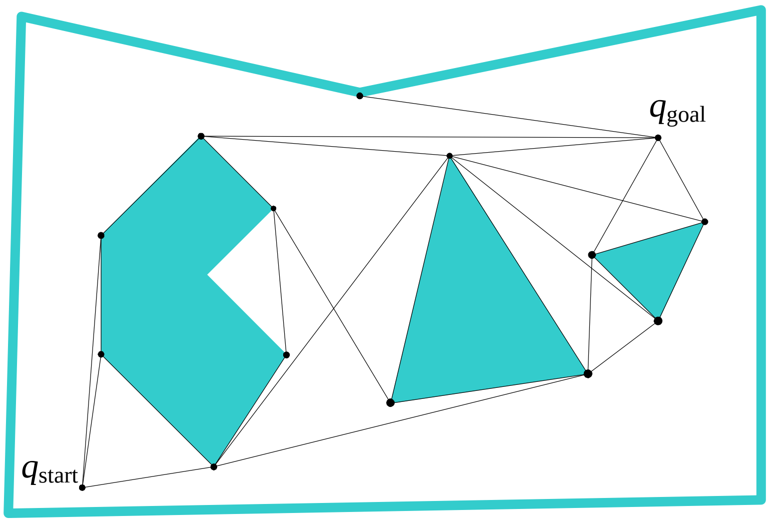


Figure 5.3. *The extended tangent visibility graph used to solve a query is obtained by connecting all visible roadmap vertices to* q_{start} *and n* q_{goal}.

roadmap that is searched for a shortest path. To each edge in graph a weight is assigned, which is the Euclidean length of the physical edge. Dijkstra algorithm can be used to find the shortest path within the visibility graph. Its running time is $O(n \log n + k)$, where n is the total number of obstacle edges, and k is the number of edges in the graph, which is in worst case $k = O(n^2)$ (to achieve this time bound, one has to use Fibonacci heaps in the implementation). An example of the shortest path is shown in Figure 5.4.

If the bitangent tests are performed naively, then the resulting algorithm requires $O(n^3)$ time. There are $O(n^2)$ pairs of reflex vertices that need to be checked, and each check requires $O(n)$ time to make certain that no other edges prevent their mutual visibility. The radial sweep algorithm can be used to obtain a better algorithm, which takes $O(n^2 \log n)$ time. The idea is to perform a radial sweep from each reflex vertex, v. A ray is rotated starting at $\theta = 0$, and check is performed when the ray touches vertices. A set of bitangents through v can be computed in this way in $O(n \log n)$ time. Since there are $O(n)$ reflex vertices, the total running time is $O(n^2 \log n)$ [40]. This algorithm is due to Lee [90].

More efficient algorithms for visibility graph construction based on arrangements have been proposed, which run in $O(n^2)$ time. An example is algorithm of Welzl [156], which works for a set of line segments, but can be adapted for sets of polygons. Ghosh and Mount developed an optimal, planar-scan technique using triangulation and funnel splits to achieve $O(k + n \log n)$ time bounds. This algorithm is important because it is output sensitive, meaning that its running time depends on the number of edges in the graph. Therefore it can be very efficient for sparse graphs. A practical comparison of those algorithms with real time measurements against a variety of testcases is performed by Kitzinger [81].

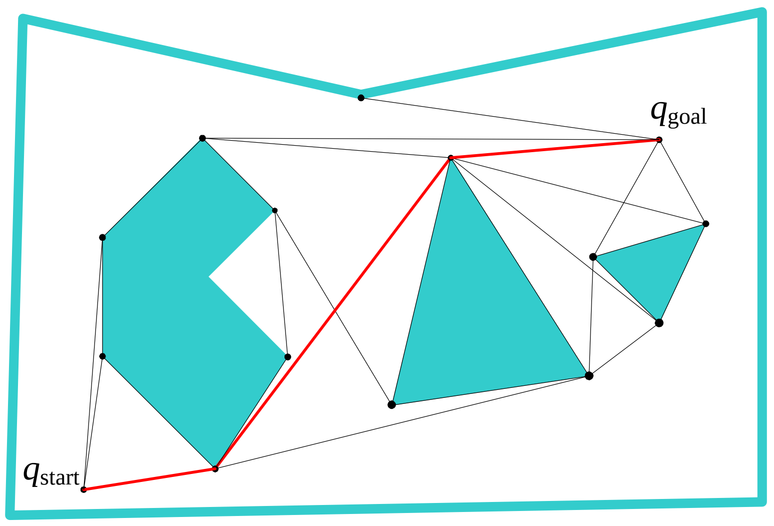


Figure 5.4. *The thick line represents the shortest path obtained by graph search in the extended tangent visibility graph.*

Any algorithm that computes a shortest path by first constructing the visibility graph is doomed to have at least quadratic running time in the worst case, because the visibility graph can have a quadratic number of edges. However, if a shortest path is constructed directly, it can be found in $O(n \log n)$ time, as demonstrated by Hershberger and Suri [58].

Sometimes it may be necessary to represent the obstacles with generalized polygons. Generalized polygons are regions bounded by straight segments and circular arcs. For example, generalized polygons show up when the polygonal obstacles are isotropically grown by a disc of radius c in a preprocessing phase. This ensures minimum clearance c between the path and the obstacles. The visibility graph method can be extended so that it handles generalized polygons [88].

5.6.2 Voronoi Diagram

Paths obtained using the Voronoi diagrams have the property that they keep as far as possible from obstacles, so that the corresponding roadmap is also called *maximum clearance roadmap*. Such paths may be preferred when the uncertainty of the robot and obstacles position is high, and when it is hard to precisely control mobile robot position. There are many variants of the Voronoi diagram (for a survey on Voronoi diagrams see [9]). Here we primarily address Voronoi diagram of polygons in the plane, which is a generalization of the Voronoi diagram of points in the plane (thus it is also called *generalized Voronoi diagram*, although the word "generalized" is often omitted).

A basic Voronoi diagram is defined for a set of points called sites [38]. A

Voronoi region is the set of points closest to a particular site. The Voronoi diagram is then the set of points equidistant to two sites; it sections off the free space into regions that are closest to a particular site. In the case of points in the plane, the Voronoi diagram contains only the line segments.

Within the path-planning, we can think of the point sites as obstacles. However, obstacles are usually represented by objects other than points. The distance from a point to an object is then measured to the closest point on the object. The definition of a Voronoi region is extended to the generalized Voronoi region, $\mathcal{F}_i$, which is the closure of the set of point closest to obstacle $\mathcal{CO}_i$, i.e.

$$\mathcal{F}_i = \{q \in \mathcal{C}_{free} \mid d_i(q) \leq d_h(q) \quad \forall h \neq i\}, \tag{5.7}$$

where $d_i(q)$ is the distance to an obstacle $\mathcal{CO}_i$ (its closest point) from q, i.e. $d_i(q) = \min_{c \in \mathcal{CO}_i} d(q, c)$.

The basic building block of the Voronoi diagram is now the set of points equidistant to two obstacles $\mathcal{CO}_i$ and $\mathcal{CO}_j$. This set is called a two-equidistant surface defined by $\mathcal{S}_{ij} = \{q \in \mathcal{C} \mid d_i(q) = d_j(q)\}$. This two-equidistant surface is further restricted to the set of points that are both equidistant to $\mathcal{CO}_i$ and $\mathcal{CO}_j$ and have $\mathcal{CO}_i$ and $\mathcal{CO}_j$ as their closest obstacles. This restricted structure is the two-equidistant face denoted by $\mathcal{F}_{ij} = \{q \in \mathcal{S}_{ij} \mid d_i(q) = d_h(q) \quad \forall h\}$. The union of the two-equidistant faces forms the Voronoi diagram VD, i.e.

$$VD = \bigcup_i \bigcup_j \mathcal{F}_{ij}. \tag{5.8}$$

This definition of the Voronoi diagram is valid in any dimensional space. For planar maps, the faces $\mathcal{F}_{ij}$ are called Voronoi diagram edges which terminate at meet points, i.e. graph nodes. Those points are equidistant to three or more closest obstacles.

If the obstacles are represented by polygonal objects, obstacles have two features, vertices and edges. The set of points equidistant to pair of vertices is a line, as well as set of points equidistant to pair of edges. But, the set of points equidistant to a vertex and an edge is a parabola, therefore the corresponding Voronoi diagram also contains parabolas. The Voronoi diagram can be built by breaking down the free space with the appropriate equidistant curves (Figure 5.5). The construction of the Voronoi diagram leads to a naive $O(n^4)$ time algorithm. Several algorithms exist that provide better asymptotic running times, but they are considerably more difficult to implement. The best-known algorithm uses sweep line method [40] and runs in $O(n \log n)$ time in which n is the number of roadmap curves.

Other useful generalizations of the Voronoi diagram concerning the shape of the sites is the Voronoi diagram of the edges of a simple polygon, interior to the polygon itself. Then Voronoi diagram is the subdivision of the interior of the

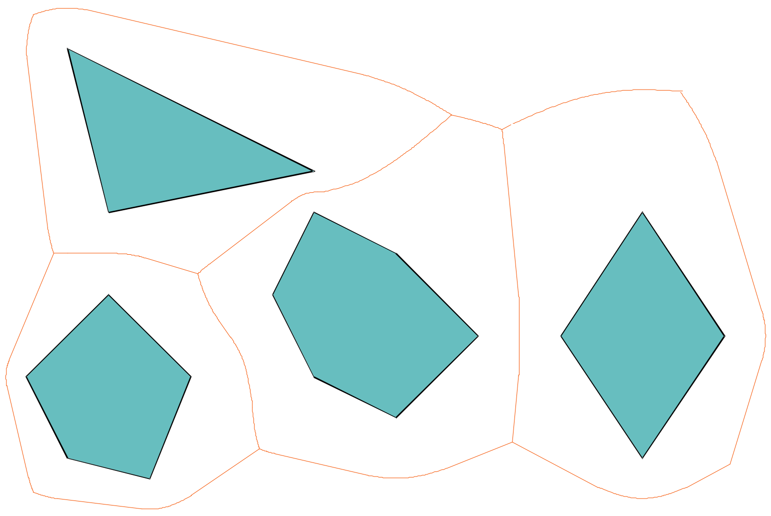


Figure 5.5. *The Voronoi diagram of polygonal obstacles.*

polygon into faces where one or two edges are the closest. This Voronoi diagram is also known as the medial axis or skeleton. The medial axis can be computed in $O(n)$ time, where n is the number of edges of the polygon, as demonstrated by Chin et al. [37].

In comparison to the Voronoi diagram for point sites, which is composed of straight edges, the occurrence of curved edges in the line segment Voronoi diagram can be a disadvantage in the computer representation and construction, and sometimes also in the application. There have been several attempts to linearize and simplify the Voronoi diagram, mainly for the sake of efficient point location and motion planning. For example McAllister et al. [102] developed an algorithm for building a compact piecewise-linear Voronoi diagram for convex sites in the plane.

In computational sense, the advantage of the Voronoi diagram over the visibility graph could be its usually better efficiency. The Voronoi diagram has $O(n)$ edges, so that querying for a path in the Voronoi diagram roadmap is faster than querying in a visibility graph. However, the quality of path obtained from the Voronoi diagram may be far from optimal. It usually has many unnecessary turns, and the length of the path may be undesirably long at regions where the obstacles are far apart. In fact, it is worth noting that minimizing the path length and maximizing the clearance seemingly contradict each other, as increasing the clearance results in a longer path whereas reducing the path length necessarily reduces the clearance from obstacles.

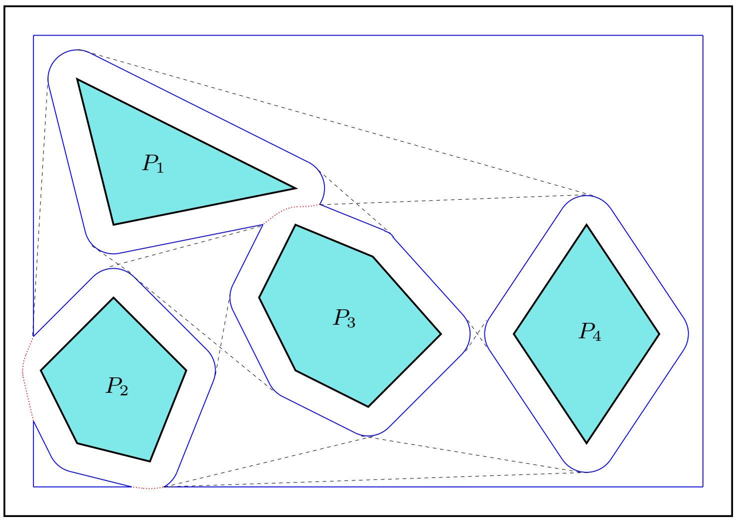


Figure 5.6. *The VV$^{(c)}$ diagram (image from [154]). The boundary of the union of the dilated obstacles is drawn in a solid blue line, the relevant portion of the Voronoi diagram is shown in dotted red. The visibility edges are drawn in a dashed line.*

5.6.3 Other Roadmap Methods

Other common roadmap-based path-planning algorithms include silhouette methods [38]. Those approaches use extrema of a function defined on a codimension one hyperplane called a slice. When the slice is one-dimensional, it can also be called a sweep line. As the slice is swept through the configuration space, the critical points of a function restricted to the slice are determined. The resulting network of extremal point forms the roadmap. The first silhouette method was roadmap algorithm of Canny [33], which indicated a begin of roadmap theory in motion planning.

Recently, Bhattacharya and Gavrilova [15] developed a roadmap-based algorithm that utilizes the Voronoi diagram to obtain a path that is a close approximation of the shortest path satisfying the required clearance to the obstacle value set by the user. If the required clearance is set to zero, the obtained path is the same as the one obtained with the visibility graph method. The obtained path is further refined in order to make it smoother. The advantages of their approach are claimed to be simplicity, versatility, and efficiency (it runs in $O(n \log n)$ time).

Another recent approach is a diagram called the VV$^{(c)}$ diagram (the Visibility-Voronoi diagram for clearance c) developed by Wein et al. [154]. An example of the VV$^{(c)}$ diagram is drawn in Figure 5.6. The motivation behind this work is also to obtain a shortest path for a specified clearance value. The diagram evolves from the visibility graph to the Voronoi diagram as the value of c increases. According to [154], the VV$^{(c)}$ diagram can be constructed in $O(n^2 \log n)$ time, however this result is further enhanced to $O(n \log n)$ by Bygi and Ghodsi [32].

5.7 Proposed Path-Planner

In the sequel a path-planning algorithm developed in this work is described. As a prerequisite, a map representation is discussed first, followed by path-planning algorithm description and some implementation aspects.

5.7.1 Map Representation

Distributed and onboard sensors of the intelligent space enable fast and precise sensing of the whole space. To utilize those advantages, it is desirable to use a deterministic environment model. Therefore a geometric map representation is selected as the most appropriate. The advantage of geometric maps over commonly used grid maps is their precision due to continuous representation of the robot workspace. Grid maps introduce discretization error typical for cell decomposition methods (see Section 5.4). Additionally, geometric maps are typically more compact and consume less memory, which is especially remarkable in sparse environments.

It is commonly agreed that most geometric scenarios can be modeled with sufficient accuracy by polygonal objects, especially in indoor environments where walls are usually straight, as well as other objects (furniture, etc.). Therefore it is decided to represent the obstacles by simple polygons. In the case that circular objects are present, they can be approximated with multiple lines where number of lines depends on desired quality of the approximation. The possible drawback of this approximation can be the map overfilled with lines which could lead to inefficiency. If this is a major problem, one can switch to generalized polygons representation which, apart from lines, can also contain circular arc segments (but algorithms are then more complex).

A possible difficulty with geometric maps could arise in large spaces with many and complicated obstacles (but this is common for most other map representations as well). In this case planning may become inefficient. A possible solution for this problem is decomposition of the complex map into smaller maps. For example, every room of a building could be a separate cell. Cell boundaries and connections could be automatically determined using the narrow passages as natural connectors between cells (see e.g. work by Seder et al. [131]). In this way path planning is decomposed into high-level and low-level planning. High-level planning first determines the cells with start and goal configurations. Then it performs graph search to determine which cells the robot should traverse in order to get from the start to the goal cell. Low-level planning then plans a path within each cell.

In this work it is assumed that environment is dynamic and that obstacles can move. Therefore, two kind of maps are utilized: static and dynamic map. The static map serves as a placeholder for non-moving obstacles, such as walls, furniture, etc. This map is created offline and can be generated using various methods, e.g. using building plan, using distributed sensors of the space, or using a mobile robot equipped with appropriate sensors to explore and map the environment. On the other side, dynamic map contains moving objects such as robots, and people, which are detected online using onboard or distributed sensors. Unlike the static map, the object positions in the dynamic map are updated in every processing cycle to enable handling of changes in the environment. Details about mapping algorithms are out of the scope of this work (see Section 2.4 for more details).

It should be mentioned that in this work a map is used as an abstraction layer between sensors on the one side, and planning modules on the other side. This has an advantage of avoiding an explicit dependency of algorithms on a certain type of sensors and increasing flexibility. Of course, use of the map abstraction level presumes that map update and replanning can be performed in real-time.

In this way a map abstraction layer substitutes reactive algorithms that use direct sensor feedback for obstacle avoidance commonly used in mobile robotics. However, reactive algorithms are still very useful, particularly for a supervision of higher-level planning algorithms. As sensor-feedback reactive algorithms can be implemented very efficiently, they can run in robot onboard hardware as a watchdog module. In this way a reactive algorithm would circumvent decisions of the high-level planner if a dangerous situation occurs, such as a potential collision. This can be caused by a number of reasons, such as hardware failure or an unexpected error in the high-level motion planner.

5.7.2 Path-Planning Algorithm

A path planner that is ideal in our application would at least have the following capabilities: (i) finding a shortest path that satisfies minimum clearance requirement wherever possible, (ii) reporting amount of available clearance otherwise[1], and (iii) acceptable execution time. Thus, a design of the algorithm was lead by the three enlisted guidelines.

Further, to utilize advantages of precise localization of the robot and obstacles in intelligent spaces, using of deterministic path-planning method is desirable. The potential field methods are not appropriate because of local minima problems and the reactive character of the method. The sampling-based methods are excluded because of their probabilistic nature. The cell decompositions are excluded as well because, due to decomposition, they produce approximative or low-quality paths.

Therefore the roadmap methods are picked as the most appropriate. One of the goals is to obtain fast robot motion, which in most cases corresponds to taking a shortest path. The visibility graph method is a roadmap method that produces

[1]This information is useful in later modules, e.g. a trajectory planner can reduce robot velocity if the required clearance is not available.

the shortest path in the geometric map with polygonal obstacles. Unfortunately, the visibility graph produces only a semi-free path (a path that touches obstacles), which is not acceptable in praxis due to uncertainty. Moreover, the next module of the decoupled approach (path-smoothing) will produce deviations from an original path, so that even if the original path is semi-free, smoothed path may not be. One possibility to overcome this problem is to insert additional module between path-planning and path-smoothing modules for path postprocessing. Its task would be ensuring a necessary clearance to the obstacles. However, besides the fact that it is very difficult to develop such algorithm, it would also bring increased computational cost.

To avoid path postprocessing, in this work the configuration space is transformed in order to account for required clearance from obstacles. This is achieved by dilating configuration space obstacles. Minkowski sum is typically used for this operation. This works well in most circumstances, but problems occur in narrow passages as passage may disappear due to dilating. Although in praxis we usually don't prefer our robot to go through narrow passages, there are two situations where this is still desirable: (i) if a pass through narrow passage considerably shortens the path, and (ii) if the narrow passage is the only way to the goal. Therefore, a way must be found to somehow allow passing through narrow passages even after dilating configuration space obstacles. Of course, such paths should preferably go through the middle of the narrow passage.

This is achieved by kind of a selective dilatation of the configuration space obstacles—obstacles are dilated for a required clearance only where possible. If the visibility graph is constructed among such dilated obstacles, the resultant roadmap will ensure required clearance from the obstacles. In narrow passages where the required clearance is not available, a path is constructed through the middle of the passage. This is obtained by using the Voronoi diagram. Therefore, in the final roadmap both the visibility graph and Voronoi diagram are combined.

For later modules, path smoothing and trajectory planning, it is beneficial to have information about the path clearance available at the particular path segment. This is exactly what this method enables. The path segments produced by the visibility graph are guaranteed to have required minimum clearance from the obstacles everywhere but in narrow passages, where in turn the clearance is reported based on Voronoi diagram.

The algorithm is decomposed to separately handle static and dynamic obstacles. This is motivated by the fact that static roadmap can be computed offline and efficiency is here not of big importance. On the contrary, it is critical that dynamic obstacles are processed fast enough as it must be done online. For this reason the algorithm that handles dynamic obstacles is designed to act more locally, i.e. it updates only those portions of the configuration space where the change occurred. Both algorithms are described in the sequel.

Construction of the Static Roadmap

In the first stage of the algorithm static obstacles are processed to construct the static roadmap. The algorithm is in many details similar to algorithm of Wein et al. [154] that produces the visibility–Voronoi diagram for clearance c. Input to the algorithm is the static workspace map containing a set of simple disjoint polygons $W_i,\ i \in [1, n_w]$ that represent the static workspace obstacles. The algorithm is called ConstructStaticRoadmap and is enlisted in Algorithm 5.7.1.

Algorithm 5.7.1: ConstructStaticRoadmap

Input: $W_i,\ i \in [1, n_w]$: static workspace
Output: SRM : static roadmap

1. Approximately construct the configuration space C, i.e. configuration space obstacles $C_i,\ i \in [1, n_c], n_c \leq n_w$ by approximately offsetting the workspace obstacles by the radius r of the robot (or its bounding circle) plus some preferred safety distance ε. Unify configuration space obstacles that intersect.

2. Construct the dilated configuration space obstacles $C_i^{(c)},\ i \in [1, n_c]$ by offsetting the configuration space obstacles by preferred clearance value c minus ε. Dilated configuration space obstacles $C_i^{(c)}$ may not be disjoint. A point of intersection of two dilated obstacle boundaries is called a chain point.

3. Find the dilated configuration space $C^{(c)}$ by computing union of all $C_i^{(c)}$.

4. Compute the extended visibility graph VG of $C^{(c)}$ by first computing the tangent visibility graph of $C^{(c)}$. Extend the graph by connecting chain points to the graph. In this way visibility edges between two chain points and tangent visibility edges emanating from chain points are added to the tangent visibility graph.

5. Construct the Voronoi diagram VD of the configuration space C. Compute the intersection $VD \bigcap C^{(c)}$, i.e. the part of the Voronoi diagram that is contained within the union of the dilated configuration space obstacles. Approximate Voronoi arcs by straight segments. Static roadmap SRM is obtained by combining extended visibility graph VG and corresponding Voronoi segments, i.e. $SRM = VG \bigcup (VD \bigcap C^{(c)})$. Also find and store the clearance of each Voronoi segment in the SRM.

The first step of the algorithm deserves a further explanation. In Figure 5.1 it is shown that the configuration space for the circular robot is exactly constructed by finding the Minkowski sum with the robot (i.e. disc). However, Minkowski sum, in addition to straight lines, produces also circular arcs. The obstacles dilated in this are generalized polygons whose boundary consists of straight lines and circular arcs and is therefore G^1 continuous. Consequently, the visibility graph constructed on such generalized polygons would also be G^1 continuous.

Although G^1 continuity may be sufficient in some applications, in this work G^2 continuity is required to achieve smooth velocity of the robot. As will be discussed in Chapter 6, the required G^2 continuity is ensured in the next, path-smoothing module. However, the path-smoothing module is designed so that its input can be only G^0 continuous (piecewise-linear) path, and the module does all the necessary job to transform it to G^2 continuos path.

Therefore, we do not insist on G^1 continuity in the path-planning module, so that it is designed to produce only G^0 continuous path. This also helps to save some computation time, as it is more complex to construct visibility graph of generalized polygons. This is the main difference between ConstructStatic-Roadmap algorithm and Wein's algorithm [154] that produces G^1 continuous paths. This is achieved by computing the Minkowski sum only approximately by approximating circular arcs by line segments. In general case polygons may not be convex, so that they are decomposed to convex polygons prior to computing Minkowski sum [36].

The obstacles are dilated for $r + \varepsilon$, where an extra amount ε is the safety distance that determines how close the robot may approach to an obstacle. This parameter is user-defined, and it must at least account for various uncertainties, such as uncertainty of the map, measurements, positioning, etc.

The most time-consuming task of the described algorithm is computing the visibility graph. If standard radial sweep algorithm is used $O(n^2 \log n)$ time is required, which is acceptable because computation is done offline.

Adding Dynamic Obstacles

Dynamic obstacles make robot life considerably harder compared to their relatives that reside in the static worlds. Consequently, the algorithm for incorporating dynamic obstacles into the roadmap is significantly harder to design—it is more time-critical because all computations have to be done online. For example, in the algorithm ConstructStaticRoadmap it is acceptable to construct the Voronoi diagram of the whole configuration space, although usually only a small portion of it is incorporated into the final roadmap. On the contrary, when handling moving obstacles the Voronoi diagram is computed only locally. The algorithm is called AddDynamicObstacles and is enlisted in Algorithm 5.7.2. Input to the algorithm is the dynamic workspace (i.e. map) containing a set of simple disjoint polygons $DW_i, i \in [1, n_{dw}]$ representing dynamic workspace obstacles.

Algorithm 5.7.2: ADDDYNAMICOBSTACLES

Input: $DW_i, i \in [1, n_{dw}]$: dynamic workspace
Output: RM : final roadmap

1. Initialize the final roadmap RM to be equal to the static roadmap, i.e. $RM = SRM$.

2. Approximately construct the dynamic configuration space DC, i.e. its obstacles $DC_i, i \in [1, n_{dc}], n_{dc} \leq n_{dw}$ by offsetting the dynamic workspace obstacles by the radius of the robot (or its bounding circle) plus the preferred safety distance ε. Use collision module to find collision(s) with the static configuration space obstacles or another dynamic configuration space obstacles. If some obstacles collide, there is no free pass between them so that they are joined by computing their union. As this could invalidate some of the static roadmap edges, these should be identified and marked as invalid (they are not deleted, because they may become valid again).

3. Construct the dilated dynamic configuration space obstacles $DC_i^{(c)}, i \in [1, n_{dc}]$ by offsetting the dynamic configuration space obstacles by clearance c minus ε. Again, dilated configuration space obstacles $DC_i^{(c)}$ may no longer be disjoint.

4. Use collision detection to find collision(s) with the static dilated configuration space obstacles. Compute intersection of colliding obstacles and construct the medial axis of this intersection. Insert valid edges of the medial axis into the final roadmap, where valid edges are all inner edges and those edges that are connected with chain points.

5. Update the visibility edges of the roadmap. This is done by updating the static extended visibility graph with any new chain points and dilated configuration space obstacles $DC_i^{(c)}$.

The last step of the algorithm requires use of the algorithm that can handle dynamic changes to maintain the visibility graph. For this the algorithm of Asano [8] is appropriate that can handle dynamic changes (inserts and deletes) each in $O(n)$ time. The algorithm of Vegter [150] can do the same using a structure called *visibility diagram*. This algorithms requires $O \log^2 n + k \log n$ time, where k is the number of visibility edges created or destroyed at the change.

It is also important to mention that, in case of multi-robot scenario, final roadmap RM is different for each particular robot because a robot must not be contained in its own map, but other robots may be, depending on the priority scheme. Only dynamic obstacles that are closer to the robot than some threshold distance are included into robot roadmap. The reason for this is, besides performance improvements, the position uncertainty of distant dynamic obstacles—their position could change significantly at the time the robot reaches them.

The algorithm AddDynamicObstacles can also be used to incrementally build a roadmap in case that map is initially unknown and new obstacles are detected online.

Once the final roadmap is constructed, it is used to plan the shortest path from the start to the goal configuration. This is performed by connecting start and goal configurations to the roadmap, which can be done in $O(n \log n)$ time using Lee's algorithm [90] that performs a radial sweep from each configuration. Then the shortest path is found using Dijkstra algorithm [44]. The execution of Dijkstra algorithm takes $O(n \log n + k)$, where k is the number of diagram edges encountered during the search. A* algorithm [55] can be used as well, as it increases the performance of the shortest path algorithm considerably in case one is only interested in a path between a single source and a single goal. Asymptotically however, the running time of Dijkstra algorithm and A* is equal. With this step the shortest path is found and the path-planning algorithm is terminated.

5.7.3 Collision Detection

In some circumstances the motion planner will require explicit collision check. In this work the collision checks are used for the following tasks:

1. To check whether the current robot configuration is in collision with an obstacle (which indicates robot position or map inconsistency) and whether the goal configuration is in collision (then the motion planning problem has no solution).

2. In Algorithm 5.7.2 (AddDynamicObstacles) to detect potential collisions of dynamic configuration space obstacles, i.e. to identify narrow passages. In this case, in addition to the logical predicate that denotes the collision, the output of the algorithm must also include the polygons that are result of intersection operation of underlaying polygons.

3. In some circumstances it is used by path-smoothing module to check path segment for the collision.

All collision checks are performed in configuration space, i.e. it must be determined whether a certain geometric primitive lies in free configuration space $\mathcal{C}_{free}$.

As the robot in configuration space is represented by a point, and obstacles are represented by polygons, the enlisted collision-check tasks impose the following possible geometric pairs that have to be checked for the collision: point–polygon, path segment–polygon, and polygon–polygon. Therefore, a path segment can be a line, circular arc or clothoid arc (see Chapter 6 for details about clothoids).

To check point–polygon collision an instance of point-in-polygon problem must be solved. If the polygon is convex the problem is straightforward to solve in $O(n)$ time, where n is the number of polygon edges. For simple polygons the problem is more complex but can still be solved in $O(n)$ time using ray crossings algorithm [119].

Having the algorithm for point–polygon collision, it is straightforward to extend it to handle path segment–polygon collision tests in linear time. Here the collision occurs if either of the following two cases is true: (i) the path segment is completely contained in the polygon interior, and (ii) the path segment intersects the polygon. To check the first case it is sufficient to detect whether either one of path-segment endpoints lies in the polygon. If the answer is positive, the collision is detected. To check the second case it is necessary to test every edge of the polygon for the intersection with the path segment. If any of the edge intersects the path segment, then the collision occurred. The second case requires algorithm for detecting intersection between line-line, circle-line and clothoid-line pairs. While the first two cases are straightforward to implement, the third case is more complex and is described in Section 6.7.

Finally, the most complex problem is to determine collision of polygon-polygon pairs. Here, if collision is detected, we also require computing of the corresponding polygon-polygon intersection. Many algorithms exist for this purpose and in this work a robust algorithm of Leonov and Nikitin [92] is used, whose running time is $O(n \log n)$.

To check whether a geometric primitive is in collision with any of configuration space obstacles, a naive implementation would require to perform collision check with every single polygon in the configuration space map, which is inefficient. To accelerate the collision check, the map is organized in rectangular grid. Commonly, every cell of the grid map stores the probability of the corresponding portion of the space being occupied by an obstacle, resulting with discrete representation of the free space. However, in order to retain continuous representation of the geometric map, here every cell stores a list of polygons that intersect that cell. If the map contains large and complex polygons, they are in this way decomposed into smaller pieces that can be checked more quickly for the collision. To perform a collision check, it is first determined which grid cells need to be tested for the collision. Then a collision algorithm is invoked only with the polygons that are stored in the identified grid cells.

To decide about an appropriate cell size a tradeoff must be made, which is also typical for any grid map application. So, too fine grid resolution will result with much time needed to update the grid and many cells will have to be checked in collision tests. On the contrary, too coarse resolution will result in poor performance gains. The final decision is application dependent and is usually conducted empirically.

5.7.4 Implementation Aspects

A whole motion planning algorithm is implemented using modular architecture imposed by the decoupled approach. Hereby, the path-planning algorithm is implemented as a separate module. The motion planning is executed with constant sampling time and the path-planning module is therefore executed in each processing cycle. The algorithm is divided into preprocessing component and online component, where online component is executed only as necessary (when goal point changes, an obstacle moves etc.).

The visibility graph algorithm implementation is based on Obermeyer's VisiLibity C++ library [116]. For some computational geometry operations in preprocessing stage CGAL (Computational Geometry Algorithms) library was used [35]. For online set operations with polygons (e.g. intersection), PolyBoolean C++ library was used [1].

5.8 Experimental Results

To verify effectiveness of the proposed path planner, the developed algorithms are applied to the exemplary workspace. First the construction of the dilated workspace is demonstrated, which actually represents first two stages of the algorithm CONSTRUCTSTATICROADMAP. The stages of the algorithms are illustrated in Figure 5.7. The test workspace is shown in Figure 5.7 a) and consists of a boundary, two convex obstacles and one nonconvex obstacle.

In Figure 5.7 b) the approximated configuration space of the robot is shown (a small robot with radius $r = 4$ cm is assumed and value $\varepsilon = 0$ was used), which is computed as the union of approximate Minkowski sums of each obstacle and the robot represented by a disc of radius r. By applying a similar algorithm again, the approximated dilated configuration space is constructed, which is shown in Figure 5.7 c). Here the clearance value of $c = 6$ cm was used. A test for intersection of the dilated configuration space obstacle boundaries was performed and in this way the chain points are identified.

Intersections of dilated C-space obstacles and chain points are shown in Figure 5.7 d). We see that dilatation of the configuration space results with a loss of space connectivity, as a passage between the left and the right part of the map disappeared. A care must be taken about the fact that a single dilated C-space obstacle can intersect even by itself, which typically happens with nonconvex obstacles. An example is the obstacle in lower right part of the map in Figure

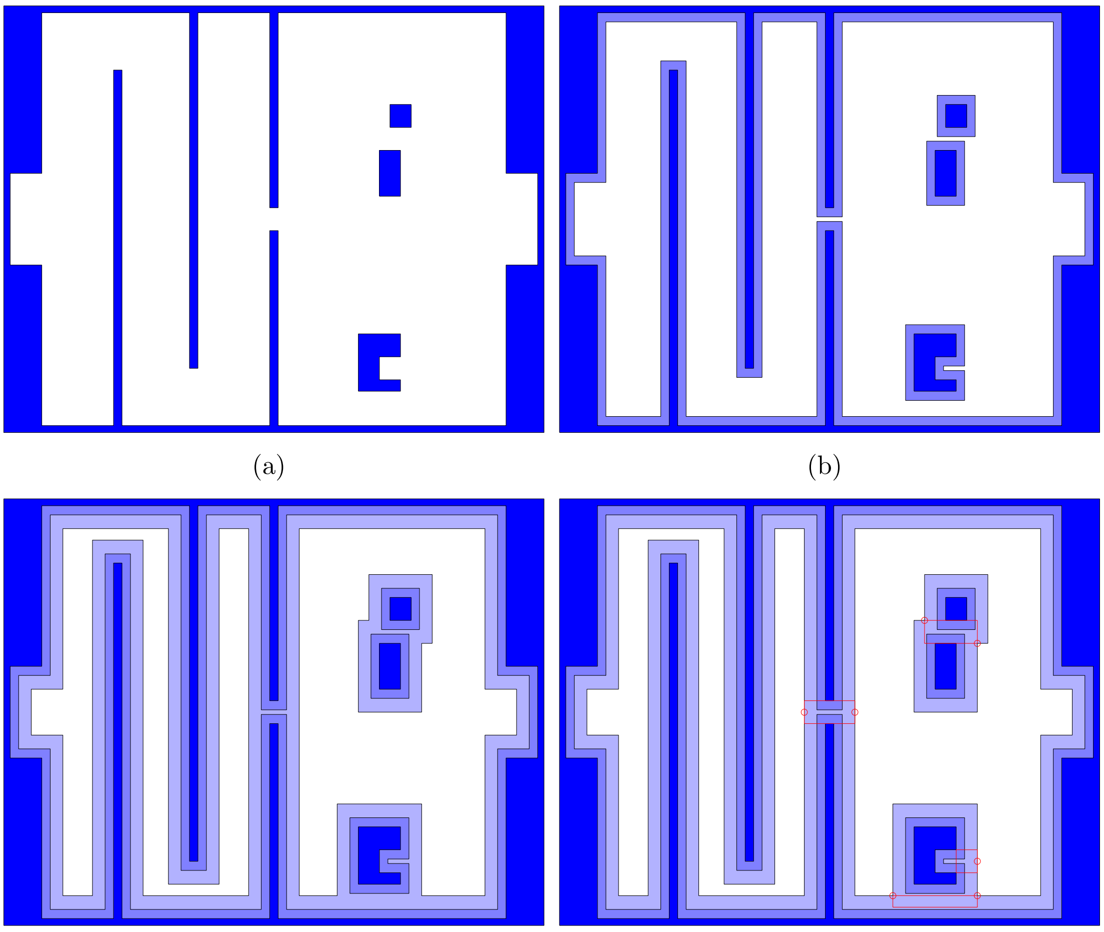

(a) (b) (c) (d)

Figure 5.7. *Construction of the dilated configuration space.* (a) *Workspace that consists of three obstacles and the boundary.* (b) *Configuration space (approximated).* (c) *Dilated configuration space (approximated).* (d) *Chain points (intersections of two dilated C-space obstacle boundaries).*

5.7 d). The same can happen by inward offsetting the boundary of the workspace, as can be seen in the narrow passage in the middle of the map.

Although this fact complicates the implementation of the algorithms, it can also be of a great benefit. Namely, in this way we can automatically detect narrow passages such as doors. At those passages the map can be decomposed into smaller parts (or graph nodes), and narrow passages can be used as natural bridges (or graph edges) between them. In this way we have obtained an algorithm for automatical building of the graph-like hierarchical map. This actually happened in the example shown, as we now have two maps, one of the left part and another of the right part of the workspace.

The remaining stages of the algorithm ConstructStaticRoadmap are demonstrated in Figure 5.8. First the tangent visibility graph of the dilated C-space is computed as shown in Figure 5.8 a). The resulting visibility graph is not connected, i.e. we actually have two visibility graphs. Also, the visibility graph is extended by connecting chain points to it, as can be seen in Figure 5.8 b).

The last stage of the algorithm ConstructStaticRoadmap begins with construction of the Voronoi diagram of the configuration space, which is shown in Figure 5.8 c). In current implementation the Voronoi diagram is found by computing a dual of the Delaunay graph of the configuration space. Here we don't need the whole Voronoi diagram, so that we extract only those parts that build up a skeleton of the configuration space, which is shown in Figure 5.8 d).

The configuration space skeleton enables now to recover from the lost connectivity caused by the C-space dilatation. This is done by computing intersection of skeleton arcs and dilated configuration space. In this way we obtain sub-skeletons that bridge narrow passages. If any non-straight segments are contained within sub-skeletons, they are approximated by line segments. Obtained sub-skeletons are shown in Figure 5.8 e). The final static roadmap is now obtained by connecting all extended visibility graphs and sub-skeletons into single graph, as shown in Figure 5.8 f).

Now let's suppose that a new obstacle is detected in the workspace. The constructed static roadmap becomes in this way invalid and needs to be updated, which is done by applying AddDynamicObstacles algorithm and is illustrated in Figure 5.9). A new obstacle drawn over the static roadmap is shown in Figure 5.9 a).

The steps 2 and 3 of AddDynamicObstacles algorithm, which give dynamic workspace and dilated dynamic configuration space, are illustrated in Figure 5.9 b). In step 4 we use collision detection module to detect possible collisions of the new (dilated) obstacle with other dilated C-Space obstacles. If collision exists, we compute one or more intersections and the corresponding chain points as illustrated in Figure 5.9 c). Next, the medial axis of the intersection is constructed as can be seen in Figure 5.9 d). In this way we bridge narrow passage between the new obstacle and static obstacles.

As can be seen in Figure 5.9 a), visibility edges of the static roadmap became invalid as they intersect with the dynamic obstacle. Thus we need to update the static roadmap and also connect chain points to it. For this an algorithm for dynamic update of the visibility graph is used, which correctly incorporates the new obstacle into the roadmap as shown in Figure 5.9 e). In Figure 5.9 f) the final dynamic roadmap is displayed over the original workspace.

Execution time of AddDynamicObstacles algorithm depends on many factors, such as complexity of the static roadmap, number and complexity of obstacles that are added into the roadmap and complexity of specific algorithms used to implement the algorithm. It is currently implemented using non-optimal algorithms (e.g. faster implementation of visibility graph construction could be achieved), but even so, it works sufficiently fast for smaller maps. E.g. it can be

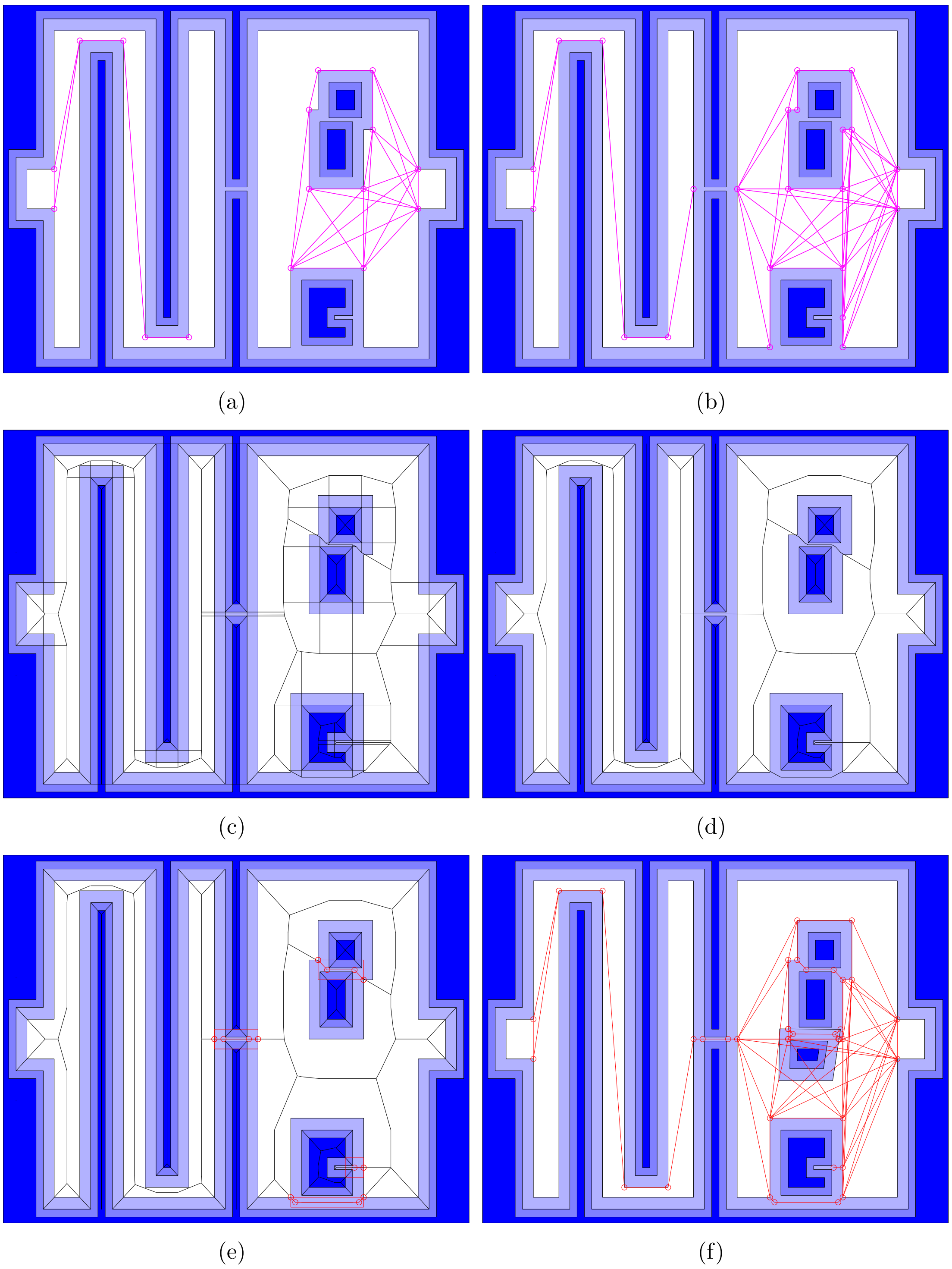


Figure 5.8. *Construction of the static roadmap.* (a) *Reduced visibility graph of the dilated configuration space.* (b) *Chain points connected to the graph.* (c) *Voronoi diagram of the configuration space.* (d) *Configuration space skeleton.* (e) *Intersections of skeleton arcs and dilated configuration space (sub-skeletons).* (f) *The final roadmap obtained by connecting all visibility graphs and sub-skeletons into single graph.*

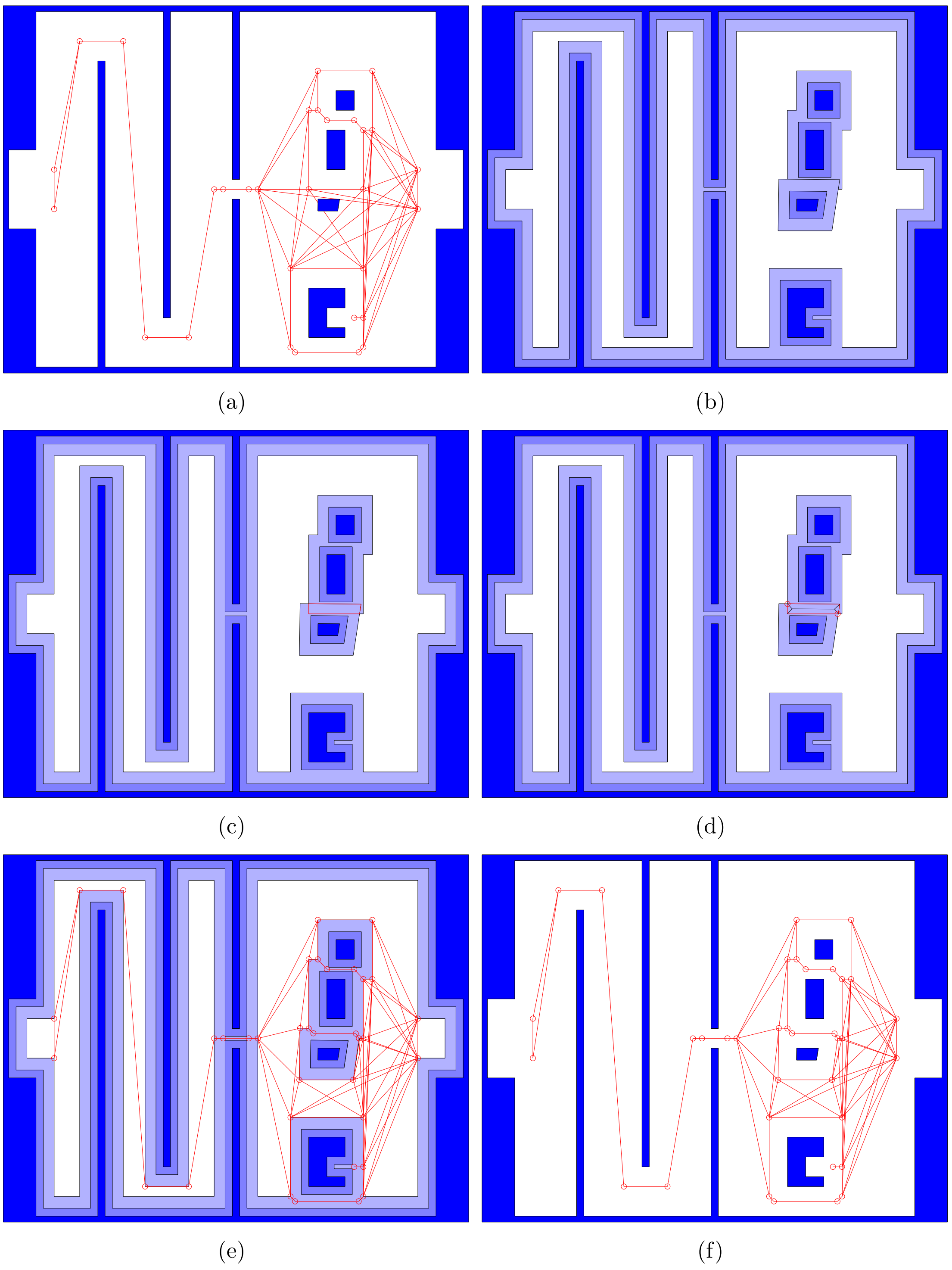


Figure 5.9. *Construction of the dynamic roadmap.* (a) *A new obstacle is detected so that static roadmap becomes invalid.* (b) *Dilated configuration space construction.* (c) *Intersection with other obstacles.* (d) *Medial axis and chain points.* (e) *Updated static roadmap.* (f) *Final dynamic roadmap which correctly incorporates the new obstacle.*

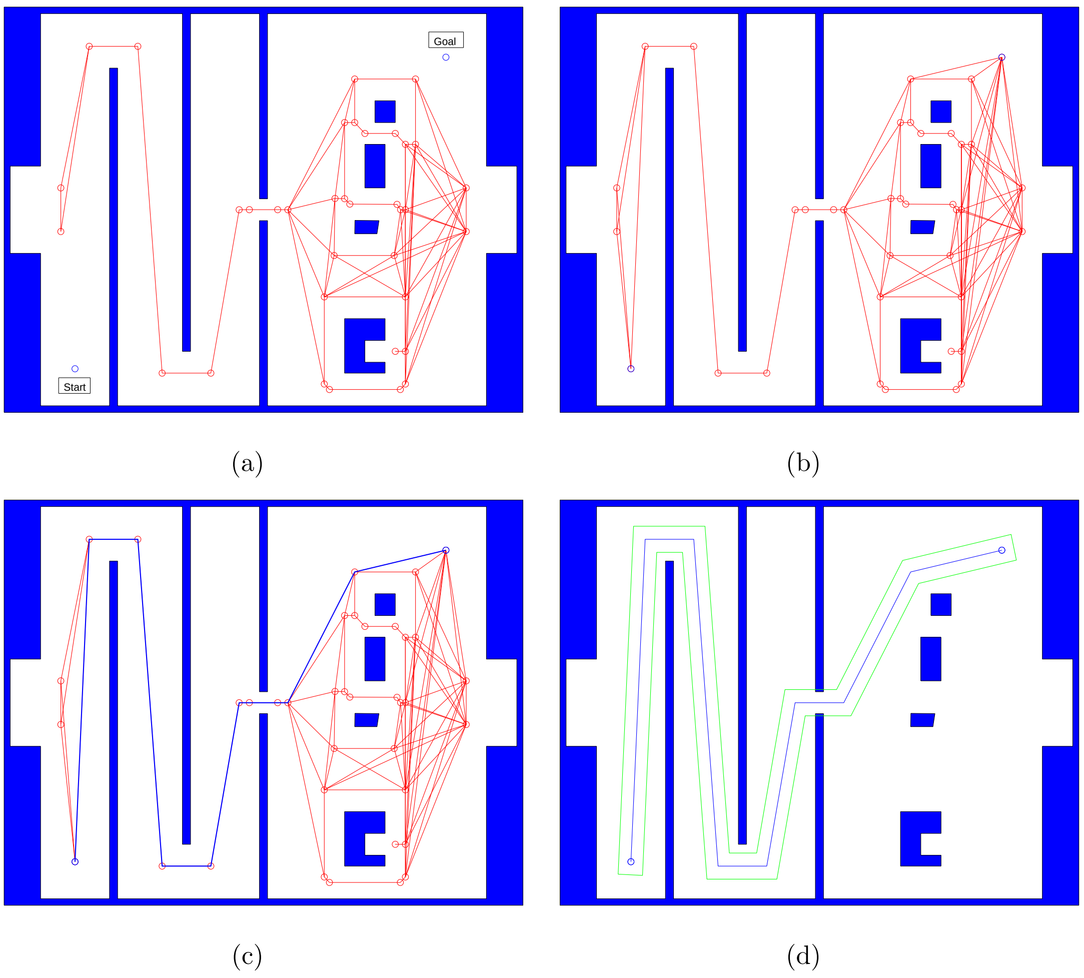
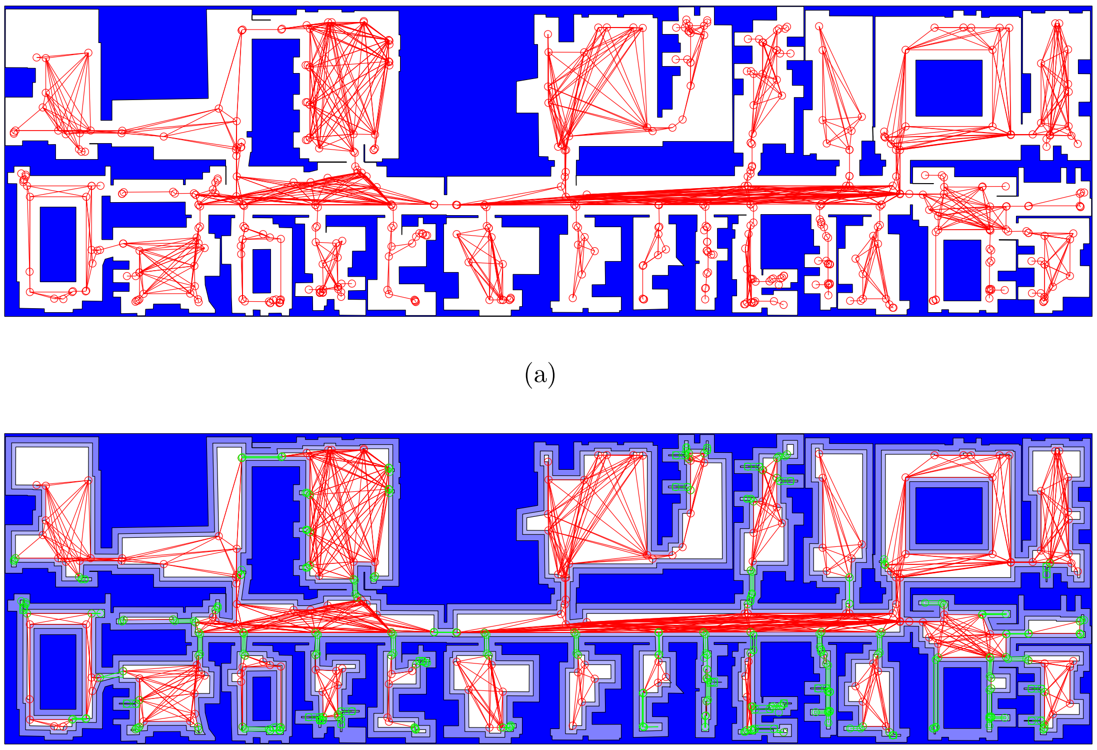


Figure 5.10. *Construction of the path.* (a) *Start and goal nodes.* (b) *Start and goal nodes connected to the dynamic roadmap.* (c) *Edges of the roadmap that are part of the shortest path from start to goal are shown with thick line.* (d) *Shortest path from the start to the goal that satisfies the minimum clearance c requirement wherever possible. A corridor of width* $2c$ *around the path is also shown.*

used in real time for robot soccer application, where a number of moving obstacles is typically ten (each robot acts as a moving obstacle). Optimization of the algorithm and extensive tests on larger maps are planned in the future.

Finally, a process of finding the shortest path in the constructed dynamic roadmap is conducted, as shown in Figure 5.9. The first step of the shortest path construction is inserting a start and goal nodes into the roadmap as displayed in Figure 5.9 a). Next we construct visibility polygons of the start and goal node and use them to connect new nodes to the roadmap, which is illustrated in Figure 5.9 b).

Finally, we search for the shortest map in the resulting roadmap. Dijkstra



(a)

(b)

Figure 5.11. (a) *The roadmap of our department obtained by the proposed algorithm.* (b) *The roadmap shown on the top of the dilated configuration space. The edges of the roadmap contained in the narrow passages are shown in green, while visibility edges are red. Here the decomposition of the space into small subspaces is clearly visible.*

algorithm is used for this purpose, and in the future it is planned to use the A* algorithm because it is faster than Dijkstra algorithm. Edges of the roadmap that are part of the shortest path from start to goal node are shown in Figure 5.9 c). Finally, we obtain the shortest path from the start to the goal that satisfies minimum clearance c requirement wherever possible, which is shown in Figure 5.9 d). We can see that we have actually obtained a corridor of width $2c$ around the path. The required clearance c is satisfied everywhere but in narrow passage between the left and the right part of the map, where the path passes right through the middle of the passage, which is just what we want.

The next path planning experiment was conducted using a real world map of our department (however, most of the furniture was not included in the map due to lack of the data). The map consists of a complex boundary polygon and six inner polygons and has totally 663 line segments. In this experiment robot radius was 0.3 m and required clearance was 0.2 m. The final roadmap obtained by the proposed algorithm is shown in Figure 5.11 a) and it consists of 547 nodes and

1384 edges.

In this example an ability of the algorithm to automatically decompose a large space into simpler subspaces is even more important. As the map is very complex, the preprocessing of the whole map consumes much time. However, automatic decomposition reduces required time significantly so that preprocessing time is about ten seconds, compared to more than a minute when the whole map is being processed. Furthermore, dynamic updates of the roadmap in a particular subspace is now possible in real time as well.

5.9 Summary

With growing computational capability of today's computers it has become feasible to perform real-time path planning using powerful computational geometry based roadmap methods, which prior were almost exclusively used for offline planning. A path-planning algorithm is proposed that is based on modified algorithm of Wein [154]. The method finds the shortest path amidst polygonal obstacles that satisfies minimum clearance requirement where possible. At those path segments where minimum clearance is not available the algorithm reports amount of clearance available, which is beneficial for other motion planning modules. Moreover, a novel algorithm that has a capability of dynamic path replanning is developed, which enables planning in dynamic environments. The path-replanning algorithm is efficient and allows realtime path replanning in presence of moderate number of moving obstacles.

In this way the robot can be controlled in closed loop, and a path can be replanned if necessary in each sample of the control algorithm in order to react to unexpected and moving obstacles. The complexity of the algorithm does not depend on size of the workspace, but only on its complexity. Under condition that robot workspace is relatively sparsely populated with obstacles, this opens possibilities such as path planning in huge workspaces, which is hard to achieve with grid-based planners.

Of course, the developed path-planner is not universal and other planners may be more appropriate in particular situations. However, a modular architecture of the overall motion planner allows using the developed method in combination with other path-planning methods as well. In such multiple-planner strategy for each subproblem the most appropriate planning method is picked by the high-level module. For example the developed path-planning method, although generally applicable for non-circular robots (e.g. car-like robots), will have difficulties with such robots when it comes to operation in narrow passages or manipulation tasks—the method may become incomplete because of circular-robot assumption it makes. The multiple-planner strategy may then switch to the sampling-based planner, which is more adequate in such situation as it is exact, although only probabilistically complete.

Another such example is when the environment becomes heavily populated by moving obstacles, where the methods based on computational geometry may become inefficient. In such conditions long-term planning cannot achieve good results anyway because long-term prediction of obstacle movements is generally unreliable. This may be a good point to switch to some more local-oriented planner, e.g. the D* algorithm in combination with the dynamic window method. Implementation of multiple-planner strategy is planned in future investigations.

Chapter 6

Path Smoothing

Common path-planning algorithms usually give obstacles-free path, but with no or very little concern about path feasibility or optimality. This chapter describes how to transform a path so that the robot can track it faster. Smoothing algorithm is given that can transform a path that consists of straight line segments to continuous curvature path, which is essential for fast robot motion. The algorithm is intended for differential drive robots and uses clothoid curves as primitives for path smoothing, which have inherent property that their curvature changes proportionally with distance traveled along the curve.

6.1 Introduction

Once the obstacle-free path is planned, it is usually necessary to apply some kind of transform algorithm to locally modify the path. There are various motivations to do this, e.g. the planned path may not be feasible for the particular robot at all (because the path planner does not consider kinematic or other constraints of the robot). The motivation could also be an optimization of the planned path, e.g. the initial path may contain sharp turns that the robot cannot follow fast enough. In the literature, path planning and transforming modules together are sometimes referred as *plan-and-transform* approach.

The path-transforming algorithm assumes that obstacle-free path is already planned by the path-planning module, that usually ignores nonholonomic constraints. This path is given as continuous function $q(s) : [0, s_g] \to \mathcal{C}_{free}$ and is defined by equation (5.2). Then the transforming algorithm can be formulated as follows [89]:

1. For the planned obstacle-free path $q(s)$, choose some $s_1, s_2 \in [0, s_g]$ such that $s_1 < s_2$ and use a local planning method to attempt to replace the portion of $q(s)$ from $q(s_1)$ to $q(s_2)$ by a path $\gamma(s)$ that satisfies the differential

(nonholonomic) constraints.

2. If $q(s)$ now satisfies the differential constraints over all $[0, s_g]$, then the algorithm terminates. Otherwise, return to Step 1.

The points s_1 and s_2 in step 1 of the algorithm may be chosen using random sequences or may be chosen deterministically.

A prerequisite for the further discussion is to introduce a path curvature—it is defined as reciprocal value of the path radius. For the path given parametrically in $\mathbb{R}^2$ the curvature $\kappa(s)$ is obtained as

$$\kappa(s) = \frac{\dot{x}(s)\ddot{y}(s) - \dot{y}(s)\ddot{x}(s)}{\left(\dot{x}(s)^2 + \dot{y}(s)^2\right)^{3/2}}, \tag{6.1}$$

where s denotes the path parameter. If traversing the path in direction of increasing s, a positive value of the curvature corresponds to steering to the left, and negative to the right, respectively. A zero curvature denotes a straight line path.

What significantly differs path planning for differential drive and car-like robots is path curvature limit that is characteristic for car-like robots, which is not present in differential drive robots. Thus, for differential drive robot any G^0 continuous path is feasible, even if it contains step changes of the curvature—this is enabled by robot's ability to rotate in place. However, the differential drive robot can pass such curvature steps only by first fully stopping, changing the orientation in place, and then accelerating again to traverse the rest of the path. The reason why the robot cannot pass curvature step with non-zero velocity is that an instantaneous change of its orientation is required, which of course is not possible in the real world.

If a robot has no curvature limits, as is the case with the robot used in this work, it may be sufficient to transform the path by using an appropriate smoothing algorithm that filters out the curvature steps. We refer to such algorithms as *path-smoothing* algorithms. The required properties of a path-smoothing algorithm in this work are at least the following:

i) G^2 continuity of the smoothed path;

ii) smoothing with non-zero initial curvature;

iii) low computational complexity.

The first property ensures that the robot will be able to traverse the smoothed path at non-zero velocity, allowing in this way fast robot motion without stopping. This also imposes that the path curvature must be continuous. Curvature continuity cannot be achieved by using solely lines and circular arcs as path-smoothing primitives (which are commonly used by many authors), as they can insure only G^1 continuity so that higher order-curves must be used.

The second property is important if the path is to be replanned while robot is moving. Namely, if current robot's path has non-zero curvature, which is often true in praxis, the smoothing method must be capable of taking this initial condition into account to obtain a feasible path.

6.2 Literature Review

Various path-smoothing methods are proposed in the literature. Often ideas originating from computer aided design applications are used, where cubic B-spline blending curves are popular because they allow inflection points and maintain curvature continuity at the joints. It is however difficult to control their curvature. Quadratic B-splines are simpler but do not maintain continuity of curvature.

Kanayama and Miyake [70] join initial and final configurations by a broken line that is used as an entry to a smoothing algorithm which produces sequences of clothoid pairs and straight lines. Initial and final configurations are supposed to have zero curvature.

Shin and Singh [135] address a similar problem with the additional constraint of a lower bounded turning radius. They produce a smooth path composed of clothoid arcs only. The developed method consists of two steps: first, a sequence of postures is obtained using given sequence of points, then each pair of neighboring postures is connected with three clothoid curve segments.

In [69], Kanayama and Hartman propose the use of cubic spirals for obtaining smooth trajectories linking a sequence of configurations two by two. They propose a criterion for smoothness (a quadratic function of the curvature or of its derivative). The use of cubic spirals produces trajectories of a larger curvature radius than in the case of clothoids, thus producing a "smoother" motion.

Delingette et al. [41] generalize the smoothing problem in order to take into account specified end conditions as well as a limited turning radius by adding control points to the trajectory. They introduced "intrinsic splines", curves whose curvature is a polynomial function of the arc length. These curves are generalizations of the clothoids and cubic spirals. In the work of Segovia et al. [134] Bezier's curves are used for this concern.

Fleury et. al. [48] use clothoids for smoothing the motion of a mobile robot moving along a trajectory but also address the problem of smoothing mobile robot motions when cusps are required, i.e., changes of motion direction along the trajectory. For this purpose, special curves called anticlothoids are used together with clothoids in order to smooth a predefined trajectory. The clothoids and anticlothoids are used because they represent the time-optimal trajectories of a two driving wheels mobile robot. Both types of curves are dual from the point of view of control. They are produced by applying, respectively, equal constant accelerations on both wheels (anticlothoids) or constant and opposite accelerations (clothoids).

Scheuer and Fraichard [130] use a set of paths, called bi-elementary paths, which are smooth and feasible for a car-like robot (curvature is upper-bounded). These paths are composed of two equal piecewise clothoids, and are used to define a simplified planner (i.e. not complete) that can locally connect two configurations. This planner can then be used in global planners, namely Ariadne's Clew algorithm and sampling-based planner, to obtain a complete planner.

Quinlan and Khatib [123] propose elastic bands to smooth a global path. Moreover, elastic bands can be adapted in real-time so that the path is deformed as changes in the environment are detected by sensors. This method enables the robot to accommodate uncertainties and react to unexpected and moving obstacles.

In [96, 97], Maček uses similar approach: D* method is used to find a global optimal path in a minimum distance sense within an environment. Thus generated geometrical path is further explored by free-space bubbles. Finally, the path is smoothed using by B-spline approximation. The method is designed for grid representation of the free space and has high computational requirements.

Montes et al. [110] also use clothoids to join two arbitrary poses in a plane by a shortest bounded-curvature path. They approximate clothoids offline with rational Bezier curve, which is later used to plan bi-elementary paths online as in [130]. This method is among the first methods that allow real-time application of clothoids.

While most of the described methods fulfil the requirement i) from Section 6.1, i.e. produce G^2 continuous path, neither method fulfils both requirement ii) and requirement iii). In other words, the existing methods are whether not specifically designed to work in real time, or do not allow specifying of non-zero initial curvature. For example the major problem with clothoid-based methods is that though they offer smoothness and linearity, it is non-trivial to generate clothoid segments for arbitrary starting and ending postures; the expressions for these curves are underconstrained and no closed form solution is available, which makes clothoid paths hard to compute efficiently.

6.3 Clothoid Steering Model

When dramatic steering changes are required to avoid the obstacles at high velocities, the navigation system operates almost entirely in the regime where curvature is continuously changing. Further, it is assumed that the robot mass is much greater than its moment of inertia, which is true for most differential drive mobile robots. This enables fast robot steering, i.e. high angular accelerations, while preventing high longitudinal accelerations. As a consequence, the longitudinal velocity can be assumed to be approximately constant, while the angular velocity will change linearly, i.e. with constant angular acceleration. Therefore, the robot mostly operates in linear curvature change regime. The question is now what is the geometric form of such a path?

For this analysis a kinematic model of the robot is used, which assumes that robot can ideally track referent longitudinal and angular velocity. There are several reasons to use kinematic model instead of the dynamic model, and the most important is that in the case of robots actuated with electrical motors, these motors are frequently equipped with low-level servo controllers that have a task to control angular velocity of the motor so that it tracks a referent angular velocity which is given by a high-level controller [111]. If the control loop is efficient, the difference between the referent and actual velocities remains small, even when the desired velocity and the motor load vary continuously, at least within a range imposed by the robot dynamic constraints (i.e. acceleration limits). Therefore, for a kinematic model to be valid, dynamic limitations of the robot should be taken into account by the motion planning algorithm. But the dynamic model of the robot is implicitly considered by the trajectory planning, which enables decoupling the kinematics from the dynamics of the vehicle. This gives a degree of robustness that allows to view the referent velocity as a free control variable, as opposed to dynamic model where the torque is viewed as control variable. In fact, for many commercially available robots the motor torques are even not available to the user-defined high-level control algorithm, but only the velocity.

Let $x(t)$ and $y(t)$ denote robot coordinates at time t in some global coordinate system in robot workspace, and let the robot orientation (heading direction) be denoted by $\theta(t)$. Let $v(t)$ denote robot linear velocity, and $\omega(t)$ robot angular velocity. Suppose that initial time is equal to zero, while $x_0 = x(0), y_0 = y(0)$, and $\theta_0 = \theta(0)$ denote initial robot position and orientation, respectively. Similarly, let $v_0 = v(0)$ and $\omega_0 = \omega(0)$ be initial robot linear and angular velocity, respectively. Then robot position at time $t \geq 0$ can be expressed as:

$$x(t) = x_0 + \int_0^t v(\tau)\cos\theta(\tau)d\tau, \tag{6.2a}$$

$$y(t) = y_0 + \int_0^t v(\tau)\sin\theta(\tau)d\tau. \tag{6.2b}$$

In linear curvature change regime angular velocity is changing linearly with time, i.e.

$$\omega(t) = \omega_0 + \alpha t, \tag{6.3}$$

where α is the angular acceleration. Now the orientation can be obtained by integrating angular velocity

$$\theta(t) = \theta_0 + \int_0^t \omega(\tau)d\tau = \theta_0 + \omega_0 t + \frac{1}{2}\alpha t^2. \tag{6.4}$$

By substituting equation (6.4) into (6.2), and assuming the constant linear velocity $v = v_0$, it is obtained

$$x(t) = x_0 + v_0 \int_0^t \cos(\theta_0 + \omega_0 \tau + \frac{1}{2}\alpha\tau^2) d\tau, \quad (6.5a)$$

$$y(t) = y_0 + v_0 \int_0^t \sin(\theta_0 + \omega_0 \tau + \frac{1}{2}\alpha\tau^2) d\tau. \quad (6.5b)$$

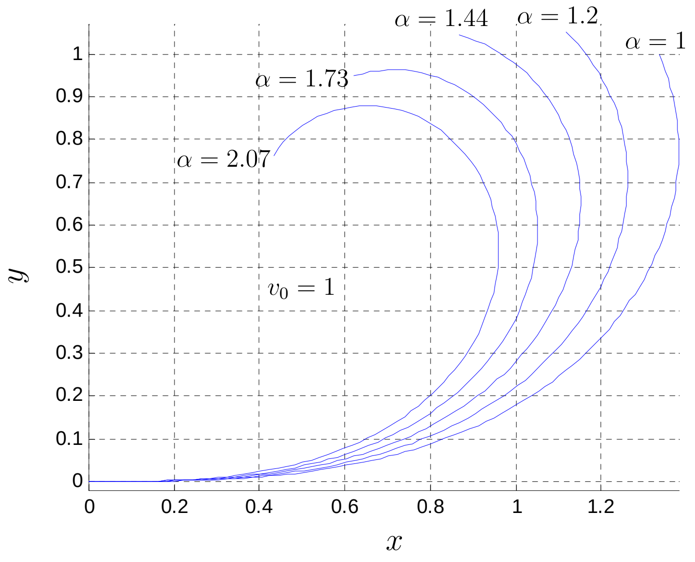


Figure 6.1. *The trajectories obtained by numerical integration of equations* (6.5a) *and* (6.5b) *using different values of angular acceleration* α. *The initial conditions are* $x_0 = y_0 = 0$, $\theta_0 = 0$, $v_0 = 1$, $\omega_0 = 0$. *The time interval is from* $t = 0$ *to* $t = 2$. *The obtained curves are called Euler spirals or clothoids and represent paths that robot traverses in the linear curvature change regime.*

The trajectories obtained by integrating (6.5a) and (6.5b) have the form of spiral, as can be seen in Figure 6.1. In the literature this curve can be found under name *Euler spiral* or *Cornu spiral*, but most often it is called *clothoid*. The corresponding steering model is therefore called *clothoid steering model*.

The clothoid curve had been originally characterized by Leonhard Euler in 1744, and rediscovered later by civil engineers in late-19th century. It is known as track transition curve in civil engineering that is fitted between a straight (also known as tangent) and a circular curve on a section of rail track or highway. A spiral easement is used to avoid abrupt changes (thus providing transition in the changes) in the centripetal acceleration experienced by a railroad vehicle and to prevent abrupt forces and discomfort. With a road vehicle the driver naturally applies the steering alteration in a gradual manner and the curve is designed to permit this. Besides in highway and railway route design, clothoids have also been used in computer graphics, e.g. McCrae and Singh [103] use clothoids to approximate sketch strokes in order to obtain continuous curvature curves.

By differentiating (6.5a) and (6.5b) and substituting into (6.1), the curvature of the clothoid is

$$\kappa(t) = \frac{\omega(t)}{v(t)} = \frac{\omega_0 + \alpha t}{v_0}, \tag{6.6}$$

from where it is evident that the curvature is changing linearly in time.

6.4 Clothoid Curve Properties

So far it is explained that the robot driving in linear curvature change regime traverses a clothoid-shaped path so that the clothoid spiral shows up as a natural choice for a geometrical primitive used to smooth robot paths. Therefore it will be useful to further explore properties of the clothoid to find out about its applicability in path-smoothing.

Let's first note that the clothoid curve has easy-to-compute analytical equations for both the tangent angle and the curvature—the tangent angle is given as quadratic polynomial in time (equation (6.4)) and curvature as linear polynomial in time (equation (6.6)). Therefore, both the orientation and curvature are smooth. This clearly encompasses advantages of clothoids over circular arcs, which is illustrated in Figure 6.2. When smoothing with circular arcs, although the path is somewhat shorter, it is not feasible unless the robot stops at curvature discontinuity point, visible in Figure 6.2 (c). Those curvature steps imply discontinuities in the angular velocity when robots travels at non-zero velocity. If the path is smoothed with clothoids, although the maximum curvature is greater, the curvature profile is continuous (Figure 6.2 (d)).

Unfortunately, equations (6.5a) and (6.5b) of the clothoid, that give path coordinates, contain integrals that cannot be solved analytically. As it will be shown in Section 6.6, this problem can be efficiently solved by using lookup-table approximation.

By applying the Pontryagin Maximum Principle (from optimal control theory [121]), one can prove that the time-optimal controls for the dynamic model of a differential drive mobile robot are [142]

$$|u_r| = |u_l| = a, \tag{6.7}$$

where u_r and u_l are accelerations of the right and left wheel, respectively, and a is maximum acceleration. The time-optimal trajectories are supported by two types of curves corresponding, respectively, to the cases $u_r = -u_l = \pm a$ and $u_r = u_l = \pm a$. In the first case the robot linear acceleration is zero, and as it was

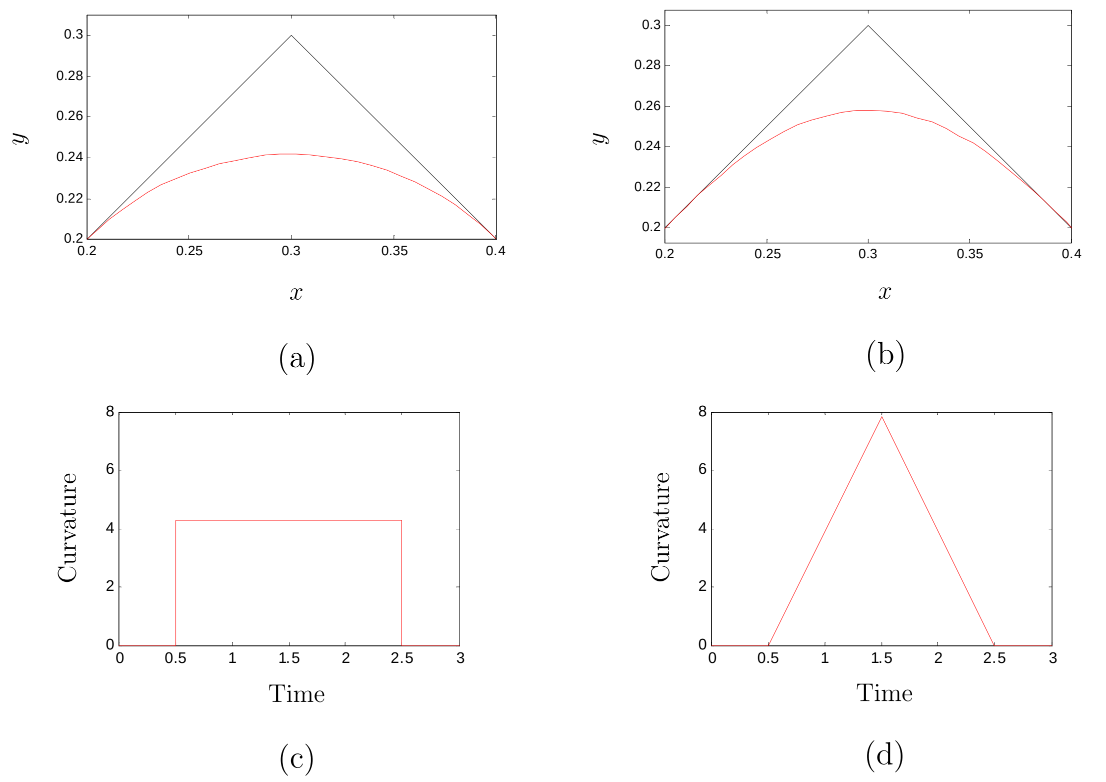


Figure 6.2. *Comparison of path smoothing with circular arcs and clothoids.* (a) *Path segment smoothed with a circular arc.* (b) *Path segment smoothed with a clothoid.* (c) *Curvature profile for path smoothed with a circular arc.* (d) *Curvature profile for path smoothed with a clothoid.*

shown, in the (x, y) plane the curve is a clothoid. In the second case the angular acceleration is null, and it can be shown that in the (x, y) plane the curve is an involute of a circle. If the initial angular velocity of the robot is null, then the involute of a circle is reduced to a straight line.

However, this result does not tell us what is the exact shape of the time-optimal trajectory from the start configuration to the goal configuration; it just gives necessary conditions for a trajectory to be time optimal. Nevertheless, it shows how the optimal trajectory looks like locally. Thus, it can be concluded that in the steering regime the optimal curve is a clothoid, which is one more reason to use clothoid as a smoothing primitive. In the longitudinal acceleration regime (zero angular acceleration), locally optimal curve is the involute of a circle. This tells us that the clothoid is not optimal in this regime.

However, at smoothing stage (where the trajectory-planning is not yet performed) it is not known where the robot will accelerate. All that can be done is to assume that robot mostly operates in steering (continuous curvature change) regime, as discussed in previous sections. Accordingly, the clothoid is chosen as the smoothing primitive that is optimal in most situations.

The validity of clothoid steering model is further experimentally confirmed by Meidenbauer [107] for Ackermann-steered vehicles. He obtained that the actual paths driven by the vehicle were generally a close match to the originally planned theoretical clothoid path.

The above considerations suggest that the clothoids are the best tradeoff regarding the required properties and they are therefore used in this work as geometric primitives for achieving smooth path-curvature transitions.

6.4.1 Traversing the Clothoid at Different Velocities

It is interesting to investigate what is the required steering control if we want to traverse the same clothoid path at different longitudinal velocities. Intuitively expected answer would be that double longitudinal velocity requires also double angular acceleration to retain the same path. But let's check it using equations. Let index "1" denote the case when longitudinal velocity is $v_1 = v_0$, and index "2" the case with different longitudinal velocity $v_2 = kv_0$, where k is a positive constant. For the simplicity, let the initial conditions be all zero, i.e. $x_0 = y_0 = 0$, $\theta_0 = 0$, and $\omega_0 = 0$. If the robot travels at different longitudinal velocity, both orientation and curvature at distance s of the clothoid must remain the same, i.e. $\theta_1(t_1) = \theta_2(t_2)$ and $\kappa_1(t_1) = \kappa_2(t_2)$, where t_1 and t_2 are times needed to travel the distance s at velocity v_1 and v_2, respectively. By using (6.4), the orientation equality condition can be written as

$$\frac{\alpha_1}{2}t_1^2 = \frac{\alpha_2}{2}t_2^2. \tag{6.8}$$

Times t_1 and t_2 are obtained as $t_1 = \frac{s}{v_0}$ and $t_2 = \frac{s}{kv_0}$. By substituting this into (6.8), we obtain

$$\alpha_2 = k^2 \cdot \alpha_1. \tag{6.9}$$

The same result would be obtained if the path curvature condition $\kappa_1(t_1) = \kappa_2(t_2)$ is examined instead the orientation condition. The result (6.9) tells us that by increasing the longitudinal velocity along the clothoid linearly in time, the angular acceleration required to follow the path must rise according to quadratic law. Another consequence of this is that all paths with the same value of ratio $\frac{\alpha}{v_0^2}$ will be clothoids that are geometrically identical.

It is also interesting to see what happens with the centrifugal acceleration. The centrifugal acceleration is an inertial acceleration that virtually tries to keep the vehicle going straight, and is obtained as the product of longitudinal and angular velocity, i.e. $a_{cf}(t) = v(t) \times \omega(t) = v(t)^2\kappa(t)$. By appropriate substitutions it is obtained that doubling the longitudinal velocity while traversing the clothoid results in eight times higher centrifugal acceleration. This explains why it could not be a very good idea to accelerate our car in the curve, especially while entering the clothoidal part of the curve.

6.5 Parametrization of a Clothoid

Until now, all clothoid equations were parameterized with the time variable t. As the path-planning does not deal with time variable but with geometric parameters only, it is necessary to rewrite clothoid equations without the time variable. Also, all other parameters that implicitly depend on time, such as velocities and accelerations, should be omitted.

As we move along the path, the parameter that uniquely determines location on the path is the traveled distance s. Therefore, it is a natural choice to parametrize the clothoid equations using the traveled distance instead of time. The required substitution is

$$t = \frac{s}{v_0}. \tag{6.10}$$

By substituting into (6.6), the equation for clothoid curvature becomes

$$\kappa(s) = \frac{\omega_0}{v_0} + \frac{\alpha}{{v_0}^2}s. \tag{6.11}$$

In this way the equation without time is obtained, but there left parameters v_0, ω_0 and α which implicitly consider time. From Section 6.4.1 it is evident that all clothoids with equal ratio $\frac{\alpha}{{v_0}^2}$ are geometrically identical. Therefore this ratio can be used as a parameter that uniquely describes geometrical shape of the clothoid curve. The parameter is denoted as

$$c = \frac{\alpha}{v_0^2}, \tag{6.12}$$

and it describes how sharp curvature changes with traveled distance, so that it is also called *sharpness* of the clothoid.

Using (6.12), the curvature of the clothoid becomes

$$\kappa(s) = \kappa_0 + cs, \tag{6.13}$$

where $\kappa_0 = \frac{\omega_0}{v_0}$ is initial path curvature.

Substituting equations (6.10) and (6.12) into equation (6.4), the tangent angle of the clothoid becomes

$$\theta(s) = \theta_0 + \kappa_0 s + \frac{1}{2}cs^2. \tag{6.14}$$

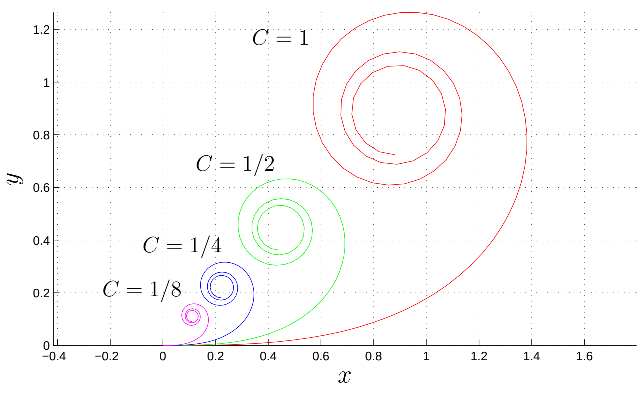


Figure 6.3. *Clothoid spirals with different values of scaling factor C.*

Similarly, equations (6.5a) and (6.5b) can be written as

$$x(s) = \mathrm{Clx}(x_0, y_0, \theta_0, \kappa_0, c, s) = x_0 + \int_0^s \cos(\theta_0 + \kappa_0\xi + \frac{1}{2}c\xi^2)d\xi, \tag{6.15a}$$

$$y(s) = \mathrm{Cly}(x_0, y_0, \theta_0, \kappa_0, c, s) = y_0 + \int_0^s \sin(\theta_0 + \kappa_0\xi + \frac{1}{2}c\xi^2)d\xi, \tag{6.15b}$$

where operators Clx and Cly denote clothoid coordinates and are introduced in order to obtain more compact equations in the sections that follow.

Instead of clothoid sharpness, *scaling factor* C is sometimes used. Relation between clothoid scaling factor and sharpness is given by

$$C^2 = \frac{1}{c}. \tag{6.16}$$

Using (6.16), it is obtained: $\int \cos(1/2c\xi^2)d\xi = C \int \cos(1/2\xi^2)d\xi$. Therefore the scaling factor directly determines the clothoid size, as it is also illustrated in Figure 6.3.

6.6 Approximation to a Clothoid

It is very easy to compute orientation and curvature of the clothoid path. However, the equations (6.15a) and (6.15b), that give point coordinates x and y of clothoid curve in Cartesian space, contain Fresnel integrals (see e.g. [87]), which in their standard form are

$$C_{Fresnel}(s) = \int_0^s \cos\left(\frac{\pi}{2}\xi^2\right) d\xi, \tag{6.17a}$$

$$S_{Fresnel}(s) = \int_0^s \sin\left(\frac{\pi}{2}\xi^2\right) d\xi. \tag{6.17b}$$

Unfortunately, the Fresnel integrals are transcendental functions—they cannot be expressed in terms of a finite sequence of the algebraic operations of addition, multiplication, and root extraction. The only way of computing clothoid coordinates is therefore the numerical approximation. Here we differentiate methods that compute coordinates in a specified point only and methods that approximate the clothoid at some interval.

Regarding methods that approximate clothoid coordinates in a single point, in [122] iterative method is used that utilizes power series. However, in this way error grows with parameter s so that power series are used only for small s. For large values of s continued complex fractions are used instead. Although power series can be computed quite quickly, continued fractions are numerically involved due to complex numbers calculations. Further, number of iterations depends on required precision.

Since clothoid coordinates are computed in terms of Fresnel integrals, many existing numerical integration algorithms can be utilized. Starting from some known initial point (usually (0,0)), such methods iteratively evaluate numerical integration algorithm with integration step that can be fixed or variable. Apart from the final point, numerical integration also outputs series of coordinates between initial and final point. Those points are unnecessary if only the final coordinates are desired, so that numerical integration is not suitable for single point approximation. However, numerical integration methods are good for offline evaluation of clothoid coordinates at some interval, as will be shown later in this chapter.

Regarding the methods that approximate a clothoid at some interval, in the literature a variety of approaches can be found that approximate clothoid curves by other analytical and easy to compute curves. For example Wang et al. [152] approximate a clothoid by Bézier curves or B-spline curves of low degrees, while Sanchez-Reyes and Chacon [128] use s-power series. Rational function approximations to the clothoid, which are very convenient in computer programs, are given by Heald [57]. Meek and Walton [105] use arc splines for this purpose. Montes et al. [110] use rational Bézier curves to approximate Fresnel integrals in real-time.

Nevertheless, if sufficient memory is available, the fastest solution and therefore the most appropriate for online computations, is to store clothoid coordinates in a lookup table. Of course, it is not possible to store all coordinates of all possible clothoids. So it must be investigated if it is possible to find appropriate numerical transformations that will compute coordinates of clothoid with any parameters based on a single clothoid stored in the memory. The clothoid whose coordinates are stored in the lookup table will be called the *basic clothoid*, and any other clothoid whose points we want to compute will be called the *general clothoid*.

Let the basic clothoid be denoted by $\mathcal{L}$. The basic clothoid has all initial conditions equal to zero and the sharpness that is some constant greater than zero, i.e.

$$x_{\mathcal{L}0} = y_{\mathcal{L}0} = 0, \quad \theta_{\mathcal{L}0} = 0, \quad \kappa_{\mathcal{L}0} = 0, \quad c_{\mathcal{L}} > 0. \tag{6.18}$$

By substituting (6.18) into (6.14) and (6.15), orientation and point coordinates of the basic clothoid become

$$\theta_{\mathcal{L}}(s_{\mathcal{L}}) = \frac{1}{2} c_{\mathcal{L}} s_{\mathcal{L}}^2, \tag{6.19a}$$

$$x_{\mathcal{L}}(s_{\mathcal{L}}) = \int_0^{s_{\mathcal{L}}} \cos\left(\frac{1}{2} c_{\mathcal{L}} \xi_{\mathcal{L}}^2\right) d\xi_{\mathcal{L}}, \tag{6.19b}$$

$$y_{\mathcal{L}}(s_{\mathcal{L}}) = \int_0^{s_{\mathcal{L}}} \sin\left(\frac{1}{2} c_{\mathcal{L}} \xi_{\mathcal{L}}^2\right) d\xi_{\mathcal{L}}. \tag{6.19c}$$

Now the question is whether it is possible to compute point coordinates of any clothoid (i.e. with any initial conditions x_0, y_0, θ_0, κ_0, and any sharpness c) using stored coordinates $(x_{\mathcal{L}}, y_{\mathcal{L}})$ of the basic clothoid. To answer this, let's first consider a slightly more general clothoid with zero initial conditions and different value of the sharpness, and denote it by $\mathcal{Z}$. Its initial conditions are thus

$$x_{\mathcal{Z}0} = y_{\mathcal{Z}0} = 0, \quad \theta_{\mathcal{Z}0} = 0, \quad \kappa_{\mathcal{Z}0} = 0, \tag{6.20}$$

and sharpness is some constant $c_{\mathcal{Z}} = c$ so that $c \neq c_{\mathcal{L}}$ and $c \neq 0$. Note first that pure scaling of the basic clothoid does not change tangent angle of the corresponding points. Therefore, for any point of the clothoid $\mathcal{Z}$ we need to find the corresponding point of the basic clothoid $\mathcal{L}$ so that there are equal tangent angles in both points, i.e. $\theta_{\mathcal{Z}}(s_{\mathcal{Z}}) = \theta_{\mathcal{L}}(s_{\mathcal{L}})$. Using (6.19a), and substituting initial conditions (6.20) into (6.14), we obtain

$$|c| s_{\mathcal{Z}}^2 = c_{\mathcal{L}} s_{\mathcal{L}}^2. \tag{6.21}$$

so that the corresponding traveled distance of the basic clothoid is

$$s_{\mathcal{L}}(s_{\mathcal{Z}}) = \sqrt{\frac{|c|}{c_{\mathcal{L}}}} s_{\mathcal{Z}}. \tag{6.22}$$

Substituting (6.22) and initial conditions (6.20) into equation for $x(s)$ (6.15a) yields

$$\begin{aligned} x(s_{\mathcal{Z}}) &= \int_0^{s_{\mathcal{L}}} \cos\left(\operatorname{sgn}(c)\frac{1}{2}c_{\mathcal{L}}\xi_{\mathcal{L}}^2\right) d\left(\sqrt{\frac{c_{\mathcal{L}}}{|c|}}\xi_{\mathcal{L}}\right) = \sqrt{\frac{c_{\mathcal{L}}}{|c|}} \int_0^{s_{\mathcal{L}}} \cos\left(\frac{1}{2}c_{\mathcal{L}}\xi_{\mathcal{L}}^2\right) d\xi_{\mathcal{L}} \\ &= \sqrt{\frac{c_{\mathcal{L}}}{|c|}} x_{\mathcal{L}}(s_{\mathcal{L}}) = \sqrt{\frac{c_{\mathcal{L}}}{|c|}} x_{\mathcal{L}}\left(\sqrt{\frac{|c|}{c_{\mathcal{L}}}} s_{\mathcal{Z}}\right). \end{aligned} \tag{6.23}$$

Analogously, for y coordinate it is obtained

$$y(s_{\mathcal{Z}}) = \operatorname{sgn}(c)\sqrt{\frac{c_{\mathcal{L}}}{|c|}} y_{\mathcal{L}}\left(\sqrt{\frac{|c|}{c_{\mathcal{L}}}} s_{\mathcal{Z}}\right). \tag{6.24}$$

The obtained result shows that coordinates of clothoid with any sharpness $c \neq 0$ can be obtained using stored coordinates of the basic clothoid with sharpness $c_{\mathcal{L}} > 0$.

The more complicated case is when $x_0 = y_0 = 0$, $\theta_0 = 0$, but initial curvature is $\kappa_0 \neq 0$. First it will be shown how this clothoid can be decomposed to combination of simpler clothoids $\mathcal{Z}$ with $\kappa_{\mathcal{Z}0} = 0$. The tangent angle for this case is

$$\theta(s) = \kappa_0 s + \frac{1}{2}cs^2. \tag{6.25}$$

By substituting this into (6.15a), x coordinate can be written as

$$\begin{aligned} x(s) &= \int_0^{s} \cos\left(\kappa_0\xi + \frac{1}{2}c\xi^2\right) d\xi = \int_0^{s} \cos\left(\kappa_0\xi + \frac{1}{2}\operatorname{sgn}(c)|c|\xi^2\right) d\xi = \\ &= \int_0^{s} \cos\left\{\operatorname{sgn}(c)\left[\left(\sqrt{\frac{|c|}{2}}\xi + \frac{\operatorname{sgn}(c)\kappa_0}{\sqrt{2|c|}}\right)^2 - \frac{\kappa_0^2}{2c}\right]\right\} d\xi. \end{aligned} \tag{6.26}$$

By using the cosine transform it can be rewritten as

$$\begin{aligned} x(s) =& \cos\left(-\frac{\kappa_0^2}{2c}\right)\int\limits_0^s \cos\left(\sqrt{\frac{|c|}{2}}\xi + \frac{\operatorname{sgn}(c)\kappa_0}{\sqrt{2|c|}}\right)^2 d\xi - \\ &\sin\left(-\frac{\kappa_0^2}{2\mathrm{c}}\right)\int\limits_0^{\mathrm{s}} \sin\left(\sqrt{\frac{|\mathrm{c}|}{2}}\xi + \frac{\operatorname{sgn}(\mathrm{c})\kappa_0}{\sqrt{2|\mathrm{c}|}}\right)^2 \mathrm{d}\xi \end{aligned} \tag{6.27}$$

To simplify the obtained integral, the following substitution can be used

$$\rho = \xi + \frac{\kappa_0}{c}, \tag{6.28}$$

which yields

$$\begin{aligned} x(s) =& \cos\left(-\frac{\kappa_0^2}{2c}\right)\int\limits_{\frac{\kappa_0}{c}}^{s+\frac{\kappa_0}{c}} \cos\left(\frac{c}{2}\rho^2\right) d\rho - \sin\left(-\frac{\kappa_0^2}{2c}\right)\int\limits_{\frac{\kappa_0}{c}}^{s+\frac{\kappa_0}{c}} \sin\left(\frac{c}{2}\rho^2\right) d\rho \\ =& \cos\left(-\frac{\kappa_0^2}{2c}\right)\left(\int\limits_0^{s+\frac{\kappa_0}{c}} \cos\left(\frac{c}{2}\rho^2\right) d\rho - \int\limits_0^{\frac{\kappa_0}{c}} \cos\left(\frac{c}{2}\rho^2\right) d\rho\right) - \\ &\sin\left(-\frac{\kappa_0^2}{2c}\right)\left(\int\limits_0^{s+\frac{\kappa_0}{c}} \sin\left(\frac{c}{2}\rho^2\right) d\rho - \int\limits_0^{\frac{\kappa_0}{c}} \sin\left(\frac{c}{2}\rho^2\right) d\rho\right). \end{aligned} \tag{6.29}$$

Now the following substitutions can be introduced into (6.29)

$$
\begin{aligned}
-\theta_{\mathcal{Z}}\left(\frac{\kappa_0}{c}\right) &= -\frac{\kappa_0^2}{2c}\\
x_{\mathcal{Z}}\left(\frac{\kappa_0}{c}\right) &= \int_0^{\frac{\kappa_0}{c}} \cos\left(\frac{c}{2}\rho^2\right) d\rho\\
y_{\mathcal{Z}}\left(\frac{\kappa_0}{c}\right) &= \int_0^{\frac{\kappa_0}{c}} \sin\left(\frac{c}{2}\rho^2\right) d\rho\\
x_{\mathcal{Z}}\left(s+\frac{\kappa_0}{c}\right) &= \int_0^{s+\frac{\kappa_0}{c}} \cos\left(\frac{c}{2}\rho^2\right) d\rho\\
y_{\mathcal{Z}}\left(s+\frac{\kappa_0}{2c}\right) &= \int_0^{s+\frac{\kappa_0}{2c}} \sin\left(\frac{c}{2}\rho^2\right) d\rho,
\end{aligned}
\tag{6.30}
$$

to obtain the following

$$
\begin{aligned}
x(s) = \cos\left(-\theta_{\mathcal{Z}}\left(\frac{\kappa_0}{c}\right)\right)&\left(x_{\mathcal{Z}}\left(s+\frac{\kappa_0}{c}\right) - x_{\mathcal{Z}}\left(\frac{\kappa_0}{c}\right)\right)\\
&- \sin\left(-\theta_{\mathcal{Z}}\left(\frac{\kappa_0}{c}\right)\right)\left(y_{\mathcal{Z}}\left(s+\frac{\kappa_0}{c}\right) - y_{\mathcal{Z}}\left(\frac{\kappa_0}{c}\right)\right).
\end{aligned}
\tag{6.31}
$$

Similarly, the y coordinate can be written as

$$
\begin{aligned}
y(s) &= \int_0^s \sin\left[\operatorname{sgn}(c)\left(\sqrt{\frac{|c|}{2}}\xi + \frac{\operatorname{sgn}(c)\kappa_0}{\sqrt{2|c|}}\right)^2 - \frac{\kappa_0^2}{2c}\right] d\xi\\
&= -\sin\left(\frac{\kappa_0^2}{2c}\right)\int_0^s \cos\left(\sqrt{\frac{|c|}{2}}\xi + \frac{\operatorname{sgn}(c)\kappa_0}{\sqrt{2|c|}}\right)^2 d\xi\\
&\quad + \mathrm{sgn(c)}\cos\left(\frac{\kappa_0^2}{2\mathrm{c}}\right)\int_0^{\mathrm{s}} \sin\left(\sqrt{\frac{|\mathrm{c}|}{2}}\xi + \frac{\mathrm{sgn(c)}\kappa_0}{\sqrt{2|\mathrm{c}|}}\right)^2 \mathrm{d}\xi.
\end{aligned}
\tag{6.32}
$$

Again, by using substitution (6.28) in (6.32) it is obtained

$$y(s) = -\sin\left(\frac{\kappa_0^2}{2c}\right)\left(\int_0^{s+\frac{\kappa_0}{c}} \cos\left(c\rho^2\right) d\rho - \int_0^{\frac{\kappa_0}{c}} \cos\left(c\rho^2\right) d\rho\right)$$
$$+ \operatorname{sgn}(c)\cos\left(\frac{\kappa_0^2}{2c}\right)\left(\int_0^{s+\frac{\kappa_0}{c}} \sin\left(c\rho^2\right) d\rho - \int_0^{\frac{\kappa_0}{c}} \sin\left(c\rho^2\right) d\rho\right). \quad (6.33)$$

And with substitution (6.30) we obtain

$$y(s) = -\sin\left(\theta_{\mathcal{Z}}\left(\frac{\kappa_0}{c}\right)\right)\left(x_{\mathcal{Z}}\left(s+\frac{\kappa_0}{c}\right) - x_{\mathcal{Z}}\left(\frac{\kappa_0}{c}\right)\right)$$
$$+ \operatorname{sgn}(c)\cos\left(\theta_{\mathcal{Z}}\left(\frac{\kappa_0}{c}\right)\right)\left(y_{\mathcal{Z}}\left(s+\frac{\kappa_0}{c}\right) - y_{\mathcal{Z}}\left(\frac{\kappa_0}{c}\right)\right). \quad (6.34)$$

By noticing that sine and cosine functions in (6.31) and (6.34) actually build up a rotation matrix, the following substitution can be made

$$\begin{bmatrix} \cos\left[-\theta_{\mathcal{Z}}\left(\frac{\kappa_0}{c}\right)\right] & -\sin\left[-\theta_{\mathcal{Z}}\left(\frac{\kappa_0}{c}\right)\right] \\ \sin\left[-\theta_{\mathcal{Z}}\left(\frac{\kappa_0}{c}\right)\right] & \cos\left[-\theta_{\mathcal{Z}}\left(\frac{\kappa_0}{c}\right)\right] \end{bmatrix} = R\left[-\theta_{\mathcal{Z}}\left(\frac{\kappa_0}{c}\right)\right], \quad (6.35)$$

where $R(\theta)$ is rotation matrix defined as

$$R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}. \quad (6.36)$$

The tangent angle of $\mathcal{Z}$ in $s = \kappa_0/c$ is

$$\theta_{\mathcal{Z}}\left(\frac{\kappa_0}{c}\right) = \frac{\kappa_0^2}{2c}, \quad (6.37)$$

so that

$$R\left[-\theta_{\mathcal{Z}}\left(\frac{\kappa_0}{c}\right)\right] = R\left(-\frac{\kappa_0^2}{2c}\right). \quad (6.38)$$

Using rotation matrix (6.38), (6.31) and (6.34) can be written in matrix form as

$$\begin{bmatrix} x(s) \\ y(s) \end{bmatrix} = R\left(-\frac{\kappa_0^2}{2c}\right)\begin{bmatrix} x_{\mathcal{Z}}\left(s+\frac{\kappa_0}{c}\right) - x_{\mathcal{Z}}\left(\frac{\kappa_0}{c}\right) \\ \operatorname{sgn}(c)\left(y_{\mathcal{Z}}\left(s+\frac{\kappa_0}{c}\right) - y_{\mathcal{Z}}\left(\frac{\kappa_0}{c}\right)\right) \end{bmatrix}, \quad (6.39)$$

which is telling us that to obtain Cartesian coordinates of clothoid with parameters $x_0 = y_0 = 0$, $\theta_0 = 0$, $\kappa_0 \neq 0$ and sharpness $c \neq 0$ from clothoid $\mathcal{Z}$ with zero initial conditions and sharpness $c_z = c$, the following transformations have to be performed:

- shift in s parameter;
- translation;
- rotation.

The final question is how to obtain coordinates of general clothoid with any initial conditions $x_0, y_0, \theta_0, \kappa_0$ and sharpness $c \neq 0$ using the clothoid $\mathcal{Z}$ with zero initial conditions and the same sharpness. Using previous result (6.39), this step is easy, since only additional rotation and translation must be introduced. Therefore

$$\begin{bmatrix} x(s) \\ y(s) \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + R\left(-\frac{\kappa_0^2}{2c} + \theta_0\right) \begin{bmatrix} x_{\mathcal{Z}}\left(s + \frac{\kappa_0}{c}\right) - x_{\mathcal{Z}}\left(\frac{\kappa_0}{c}\right) \\ y_{\mathcal{Z}}\left(s + \frac{\kappa_0}{c}\right) - y_{\mathcal{Z}}\left(\frac{\kappa_0}{c}\right) \end{bmatrix}. \tag{6.40}$$

Finally, using equations (6.23), (6.24) and (6.40), it is possible to obtain coordinates of the general clothoid using the coordinates $x_{\mathcal{L}}$ and $y_{\mathcal{L}}$ stored in the lookup table:

$$\begin{aligned} \begin{bmatrix} x(s) \\ y(s) \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} &+ R\left(-\frac{\kappa_0^2}{2c} + \theta_0\right) \\ &\cdot \sqrt{\frac{c_{\mathcal{L}}}{|c|}} \begin{bmatrix} x_{\mathcal{L}}\left(\sqrt{\frac{|c|}{c_{\mathcal{L}}}}\left(s + \frac{\kappa_0}{c}\right)\right) - x_{\mathcal{L}}\left(\sqrt{\frac{|c|}{c_{\mathcal{L}}}}\frac{\kappa_0}{c}\right) \\ \operatorname{sgn}(c)\left[y_{\mathcal{L}}\left(\sqrt{\frac{|c|}{c_{\mathcal{L}}}}\left(s + \frac{\kappa_0}{c}\right)\right) - y_{\mathcal{L}}\left(\sqrt{\frac{|c|}{c_{\mathcal{L}}}}\frac{\kappa_0}{c}\right)\right] \end{bmatrix}. \end{aligned} \tag{6.41}$$

The obtained result shows that it is possible to compute points of any clothoid by transforming points of the basic clothoid. Our plan is to store coordinates of the basic clothoid in the lookup table and reuse them later to accelerate calculations with clothoids. However, the number of points in the table is limited. This fact restricts us from computing *any* clothoid, but only a subset of all possible clothoids.

Therefore a careful analysis must be conducted in order to

1. determine a required set of clothoids that is sufficient for our application;
2. determine safe parameters of the lookup-table based on required set of clothoids, that guarantee bounded error of clothoid coordinates retrieved with equation (6.41).

The following lookup-table parameters have to be determined: (1) sharpness $c_{\mathcal{L}}$ of the basic clothoid stored in the lookup table; (2) interval of the basic clothoid that will be stored in the table, i.e. the length of the basic clothoid $s_{\mathcal{L}max}$; and

(3) how dense a selected interval will be sampled, i.e. sampling period $\Delta s_{\mathcal{L}}$—this is a distance between the successive points of the basic clothoid.

Note that although relation (6.41) is exact, the obtained point coordinates will only be approximate. Sources of the error and possible remedies are the following:

1. **Source:** Coordinates $x_{\mathcal{L}}$ and $y_{\mathcal{L}}$ of the basic clothoid still have to be computed numerically and therefore contain errors.

   Remedy: This error can be easily bounded because coordinates can be computed offline with any desired accuracy (at least up to machine precision), and could therefore be neglected.

2. **Source:** As clothoid curve needs to be discretized in order to store it in the lookup table, a sampling error is introduced.

   Remedy: To answer a query that falls between two successive points of the lookup table an interpolation can be used. Approximation error will then depend on the sampling interval $\Delta s_{\mathcal{L}}$ and the quality of the interpolation.

In the sequel it will be discussed how to best implement the interpolation. First the interpolation error will be analyzed. Then a possible procedure for determining the required set of clothoids for particular application will be given. Based on this set, safe lookup table parameters in order to retain required precision will be determined. Finally, an example of determining lookup table parameters for real robot will be given.

6.6.1 Interpolation Issues

To compute the clothoid coordinates that fall between two successive points of the lookup table an interpolation must be used. The fastest interpolation algorithm than can be applied is the first-order interpolation. Because of high error that it introduces it is no way applicable for this problem, unless very small sampling interval $\Delta s_{\mathcal{L}}$ is used, which again causes high memory consumption for table storage.

A better option is second-order interpolation where straight line is interpolated between two neighbor points in the lookup table. Unfortunately, if points are sampled with fixed $\Delta s_{\mathcal{L}}$, larger lengths of the clothoid will result with the higher interpolation error. This is because the curvature grows with parameter s, resulting with larger distance of the clothoid curve from the interpolation line. Therefore the straight-line interpolation is also a poor choice when dealing with clothoids.

Circular interpolation is a far better choice, as a clothoid can be viewed as infinite succession of circular arc segments with linearly growing curvature. Comparison of interpolation errors for straight line and circular interpolation is given

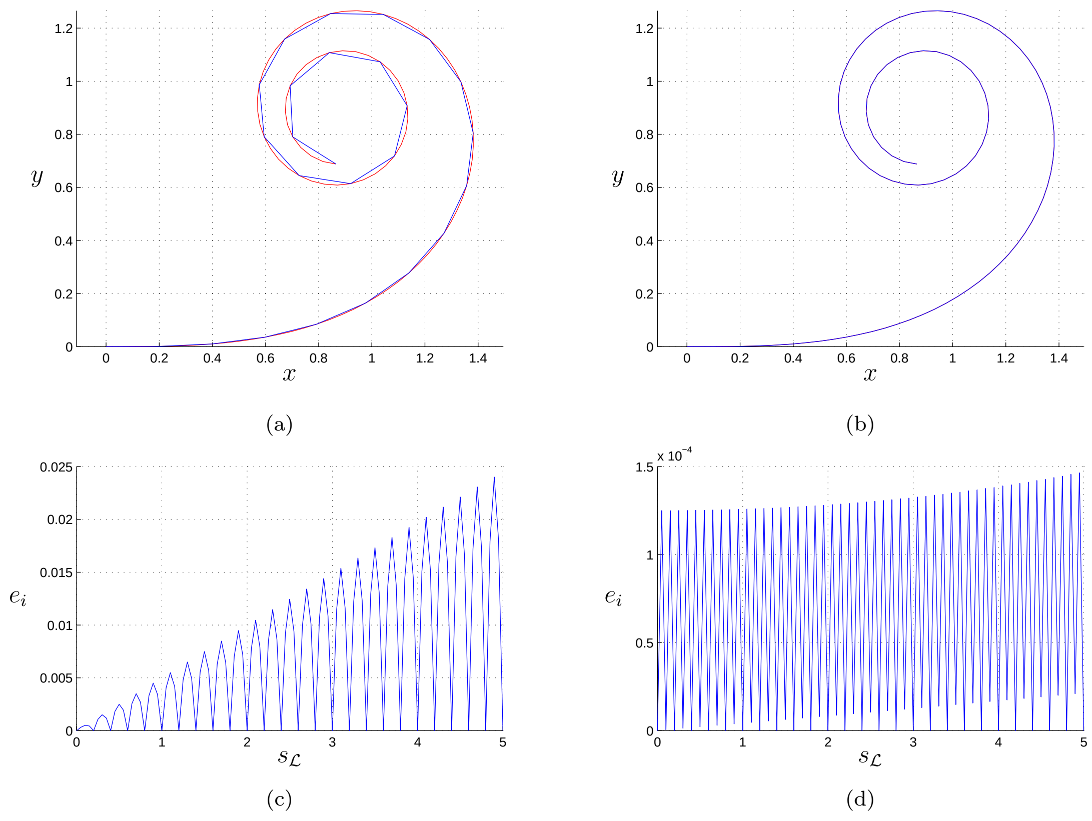


Figure 6.4. *Comparison of straight line and circular interpolation of a clothoid. The lookup table has the parameters $c_{\mathcal{L}} = 1$, $\Delta s_{\mathcal{L}} = 0.2$ and $s_{\mathcal{L}max} = 5$. To better illustrate effect of interpolation, lookup table is rather sparse and contains only 25 points. A reconstructed clothoid has the sharpness $c = 1$, too.* (a) *Clothoid interpolated by straight-line interpolation. Red curve is the exact clothoid, and blue is interpolated.* (b) *Clothoid interpolated by circular interpolation. A difference between exact and interpolated clothoid is hardly visible.* (c) *Interpolation error with straight-line interpolation. It is visible that error grows linearly. Thus it is proportional with the curvature.* (d) *Interpolation error with circular interpolation. This error is substantially lower than straight-line interpolation error. It grows approximately by quadratic law, but very slowly. If the change of orientation between neighbor points in the lookup table is kept low (e.g. lower than $10°$), this error can be considered as constant.*

in Figure 6.4. Although the circular interpolation somewhat increases computational burden, it is chosen in this application because of better accuracy. It is experimentally determined that approximation error of the circular interpolation grows approximately by quadratic law in parameter s, therefore it grows faster than straight-line interpolation, which grows linearly. Although the error is quadratic function, it grows very slowly with s and can be considered constant if the change of tangent angle between neighbor points in the lookup table is kept low (e.g. lower than 10°).

To find out how the values of parameters $\Delta s_{\mathcal{L}}$ and $c_{\mathcal{L}}$ are related with the interpolation error an experimental analysis is conducted. The interpolation error is defined as the Euclidean distance between the exact point and the interpolated point. Here the maximum interpolation error is considered, which is the maximum interpolation error that occurs between two neighbor points and is denoted by e_{im}.

First the sampling interval $\Delta s_{\mathcal{L}}$ is kept constant while the sharpness $c_{\mathcal{L}}$ is changed. It is obtained that interpolation error and sharpness are proportional. Thus, halving $c_{\mathcal{L}}$ also halves e_{im}, which is expected because curvature then grows half as fast. Next, the sharpness $c_{\mathcal{L}}$ is kept constant, and sampling interval $\Delta s_{\mathcal{L}}$ is changed. It is obtained that halving $\Delta s_{\mathcal{L}}$ results by lowering approximation error approximately by the factor $1/2^3$. Therefore, if the lookup table has n points, then the error in the approximation is of order $\mathrm{O}(1/n^3)$, which proves the effectiveness of circular interpolation. Finally, the following empirical relation is derived that estimates the maximum interpolation error using circular interpolation

$$e_{im} \approx 0.016 c_{\mathcal{L}} \Delta s_{\mathcal{L}}^3. \tag{6.42}$$

As already stated, care must be taken about the fact that relation (6.42) is valid only if maximum change of tangent angle between neighbor points in the lookup table is kept low. Maximum change of tangent angle will occur between last two points, so that the following condition must be satisfied

$$\frac{c_{\mathcal{L}} \Delta s_{\mathcal{L}}}{2} (2 s_{\mathcal{L}max} - \Delta s_{\mathcal{L}}) < \Delta \theta_{\mathcal{L}max}, \tag{6.43}$$

where $\Delta\theta_{\mathcal{L}max}$ is maximum allowed orientation change, whose safe value is empirically obtained to be 10°. Theoretical verification of these results is left for the future work.

Possible improvement in the interpolation algorithm could be the use of non-constant sampling interval to better interpolate parts with higher curvature. Another possible improvement could be the use of other interpolation curves instead of circular arcs, e.g. rational Bézier curves [110], resulting possibly with better approximation and smaller lookup table.

6.6.2 Determining a Required Set of Clothoids

To find a required set of clothoids an allowed range of clothoid parameters suitable for particular application has to be determined. First of all, we will restrict maximum clothoid length and orientation change. Second, we will avoid clothoids with extremely low or extremely high sharpness so that clothoid sharpness can be bounded to some closed interval. Without loss of generality, in the following sections only clothoids in the first quadrant will be assumed, unless stated otherwise. Thus all parameters of a clothoid are non-negative, i.e. $s \geq 0, c \geq 0, \kappa_0 \geq 0$. Clothoid initial conditions are assumed to be zero, i.e. $x_0 = y_0 = 0, \theta_0 = 0$.

Bounding the Clothoid Orientation Change and Length

In praxis the overall orientation change $\Delta\theta$ that occurs along the single clothoid can be upper bounded, e.g. an orientation change greater than π is rarely required. Nevertheless, if the planner needs higher orientation change than π, it is better then to combine both circular arcs and clothoids in order to avoid high curvature segments. Typically a pair of two clothoids is applied to plan curves, so that for a single clothoid maximum curvature change can be bounded to $\Delta\theta_{max} = \pi/2$.

After the upper-bound of clothoid orientation change $\Delta\theta_{max}$ is chosen, using the relation between clothoid orientation and distance (6.14), the length of the clothoid can also be upper bounded. Assuming $\kappa_0 \geq 0$ and using (6.14), we obtain

$$s \leq -C^2\kappa_0 + C\sqrt{C^2\kappa_0^2 + 2\Delta\theta_{max}}, \tag{6.44}$$

which yields upper bound of the clothoid length s.

The maximum length of the clothoid can be additionally limited by some fixed upper-bound s_{max}, which can be determined e.g. based on the size of robot navigation workspace. Then the planner must be instructed not to use the clothoid turns longer than this threshold. If properly chosen, this is no significant limitation for the planner, because it can avoid unnecessarily long clothoids by using lines and circular arcs. Using (6.44), the clothoid length is now upper bounded by

$$s \leq \min\left(-C^2\kappa_0 + C\sqrt{C^2\kappa_0^2 + 2\Delta\theta_{max}}, s_{max}\right). \tag{6.45}$$

In praxis we mostly have the case when $\kappa_0 = 0$, when (6.45) becomes

$$s \leq \min\left(C\sqrt{2\Delta\theta_{max}}, s_{max}\right). \tag{6.46}$$

Finding the Minimum Scaling Factor

To find the lower bound of clothoid scaling factor C_{min}[1], we can limit how sharp the clothoid path can steer, i.e. how fast orientation can change with the path length. The lower bound will be found by defining the shortest clothoid length s_{min} at which the maximum allowed orientation change of $\pi/2$ can occur. Then C_{min} can be determined using the orientation equation (6.14) and assuming the worst case ($\kappa_0 = 0$), obtaining

$$C_{min} = \frac{s_{min}}{\sqrt{2\Delta\theta}}. \tag{6.47}$$

[1]Minimum scaling factor corresponds to the maximum sharpness c (see equation (6.16)).

In other words, we have now determined how fast the curvature is allowed to grow with the traveled distance along the path.

There is an alternative way of finding the minimum scaling factor related with the maximum approximation error. Namely, it can be shown that every clothoid is a bounded spiral so that it can be fit inside its bounding circle. As scaling factor C decreases, this bounding circle becomes smaller, as can be observed in Figure 6.3.

In the limit case $C = 0$, a clothoid is reduced to a point. This means that for scaling factor smaller than some C_{min} clothoid can be practically approximated by a point without exceeding maximum allowed approximation error. Here approximation error e will be defined as the Euclidean distance between exact clothoid point (x_e, y_e) and its approximating point (x_a, y_a), i.e.

$$e = \sqrt{(x_e - x_a)^2 + (y_e - y_a)^2}. \tag{6.48}$$

For the case of approximating clothoid by a point, by considering the worst case when $\kappa_0 = 0$ and using (6.15) and (6.48), a corresponding approximation error e_{pt} will be the function of C and can be expressed as

$$e_{pt}(C) = \sqrt{x_d(C)^2 + y_d(C)^2}, \tag{6.49}$$

so that

$$x_d(C) = \int_0^{s_d} \cos \frac{1}{2C^2}\xi^2 d\xi, \quad y_d(C) = \int_0^{s_d} \sin \frac{1}{2C^2}\xi^2 d\xi,$$
$$s_d = \arg\max_s \left(\int_0^{s} \cos \frac{1}{2C^2}\xi^2 d\xi \right)^2 + \left(\int_0^{s} \sin \frac{1}{2C^2}\xi^2 d\xi \right)^2,$$

where (x_d, y_d) is a point of the clothoid that is most distant from its begin point, and s_d is length at which this happens.

Now we can find a minimum C_{min} by computing the scaling factor at which the approximation error e_{pt} is no greater than maximum allowed error e_{max}. This will happen at a scaling factor where the clothoid just touches, but not intersects a circle with center $(0, 0)$ and radius e_{max}. Then we lower-bound the scaling factor as $C \geq C_{min}$, where its minimum C_{min} is computed so that the following error condition is fulfilled

$$e_{pt}(C_{min}) = e_{max}. \tag{6.50}$$

The queries with $C < C_{min}$ can then be treated simply by returning $x = x_0,\ y = y_0$, without exceeding maximum allowed error e_{max}.

Finding the Maximum Scaling Factor

Another special case of a clothoid occurs when $\kappa_0 = 0$ and $C = \infty$. In this case clothoid does not change the tangent orientation and is reduced to a straight line. Of course we should not allow this case because a division by zero in equation (6.41) would occur otherwise. Therefore, scaling factor should be upper-bounded, too.

In case that clothoid has extremely high scaling factor it could be approximated by a straight line. Based on (6.48), and using (6.15), the error e_{ln} of approximating clothoid with zero initial curvature by a straight line at length s is

$$e_{ln}(C,s) = \sqrt{\left(s - \int_0^s \cos\left(\frac{1}{2C^2}\xi^2\right)d\xi\right)^2 + \left(\int_0^s \sin\left(\frac{1}{2C^2}\xi^2\right)d\xi\right)^2}. \tag{6.51}$$

Now we can find a maximum allowed scaling factor C_{max} by finding the scaling factor at which the approximation error at maximum allowed clothoid length s_{max} is no greater than predefined maximum allowed error e_{max}. Then we can upper-bound the scaling factor so that $C \le C_{max}$, where C_{max} is computed so that the following error condition is fulfilled

$$e_{ln}(C_{max}, s_{max}) = e_{max}. \tag{6.52}$$

If there, however, a query occurs with value $C > C_{max}$, we can safely treat it as $C = \infty$ case without violating the maximum allowed error e_{max}. Such a query is answered simply by solving the equation (6.15) for $C = \infty$ and $\kappa_0 = 0$, so that it reduces to the following straight-line equation

$$\begin{bmatrix} x(s) \\ y(s) \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \begin{bmatrix} \cos(\theta_0) \\ \sin(\theta_0) \end{bmatrix} s. \tag{6.53}$$

Finding the Maximum Initial Curvature

Yet another special case occurs when the clothoid has non-zero initial curvature $\kappa_0 \neq 0$ and scaling factor $C = \infty$. Then the clothoid is reduced to a circle with radius $1/\kappa_0$. We introduce a factor K defined as

$$K = \kappa_0 C \tag{6.54}$$

which will serve as a measure of similarity of a clothoid to a circle. By increasing K, a clothoid becomes more similar to a circle. So instead of bounding κ_0, we will bound factor K because by upper bounding K we can control how much a clothoid with initial curvature κ_0 and scaling factor C departs from a circular arc with radius $r = 1/|\kappa_0|$. If it departs less than predefined maximum error e_{max}, the clothoid can be approximated by a circle without exceeding maximum error.

To express the error, assume $\kappa_0 \neq 0$ and $C = \infty$. Then a clothoid reduces to a circular arc of radius $1/|\kappa_0|$, whose endpoint (x_a, y_a) at length s is

$$x_a(s) = \frac{1}{\kappa_0} \sin(\kappa_0 s), \tag{6.55a}$$

$$y_a(s) = \frac{1}{\kappa_0}(1 - \cos(\kappa_0 s)). \tag{6.55b}$$

According to error definition (6.48), and using (6.15) and (6.55), the error e_{arc} of approximating clothoid by a circular arc at length s can be expressed as

$$\begin{aligned} e^2_{arc}(\kappa_0, C, s) = &\left(\frac{1}{\kappa_0} \sin(\kappa_0 s) - \int\limits_0^s \cos(\kappa_0 \xi + \frac{1}{2C^2}\xi^2) d\xi \right)^2 \\ &+ \left(\frac{1}{\kappa_0}(1 - \cos(\kappa_0 s)) - \int\limits_0^s \sin(\kappa_0 \xi + \frac{1}{2C^2}\xi^2) d\xi \right)^2 . \end{aligned} \tag{6.56}$$

If this error is small, then it has approximately equal value no matter if the sharpness is positive or negative.

The approximation error (6.56) becomes higher as length of the clothoid (or tangent angle) is increased. To find the error bound, we must bound both clothoid length (by s_{max}) and its orientation change (by $\Delta\theta_{max}$), as we did in previous cases. Therefore

$$s \leq \min \left(\frac{\Delta\theta_{max}}{|\kappa_0|}, s_{max} \right). \tag{6.57}$$

To see how factor K is changing with the initial curvature κ_0, we will change the value κ_0 and search scaling factor C so that error (6.56) is exactly $e_{arc} = e_{max}$ for each particular value of κ_0. Hereby length of the clothoid is bounded by (6.57). The obtained graph is shown in Figure 6.5. As expected, the maximum approximation error occurs at the value of κ_0

$$\kappa_{0me} = \frac{\Delta\theta_{max}}{s_{max}}, \tag{6.58}$$

where both clothoid length and orientation change are at its maximum. Then K is maximal because the maximum scaling factor (relative to initial curvature) is required in order not to exceed the maximum allowed approximation error. If we denote this worst-case scaling factor as C_{me}, a lower-bound of factor K can be expressed as

$$K \leq \frac{\kappa_{0me}}{\sqrt{C_{me}}}, \tag{6.59}$$

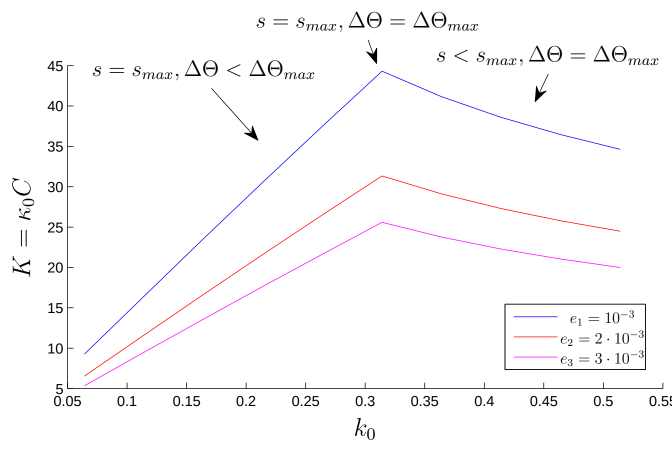


Figure 6.5. *Dependency of factor K on initial curvature of the clothoid κ_0. K is computed so that the maximum error between clothoid and circular arc is exactly e_{max}. The maximum length of circular arc was bounded by s_{max}, and its maximum orientation change by $\Delta\theta_{max}$. Three values of e_{max} were evaluated and it can be seen that tighter error tolerance results with higher K. Further, it can be noticed that K is at its maximum when both $s = s_{max}$ and $\Delta\theta = \Delta\theta_{max}$.*

where C_{me} is found so that it is fulfilled

$$e_{arc}(\kappa_{0me}, C_{me}, s_{max}) = e_{max}. \tag{6.60}$$

Like in previous cases, the queries where condition (6.59) is not fulfilled are answered by solving the equation (6.15) for $C = \infty$, which yields the following circular arc equation:

$$\begin{bmatrix} x(s) \\ y(s) \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \frac{1}{\kappa_0} \begin{bmatrix} -\sin\theta_0 + \sin(\theta_0 + \kappa_0 s) \\ \cos\theta_0 - \cos(\theta_0 + \kappa_0 s) \end{bmatrix}. \tag{6.61}$$

6.6.3 Determination of Lookup Table Parameters

Now that we have determined the set of required clothoid parameters, there left a task of finding a safe mapping between these parameters and parameters of the lookup table in order not to exceed the maximum approximation error.

First, to obtain required length of the basic clothoid $s_{\mathcal{L}max}$, maximum possible values of lookup table arguments, i.e. arguments of $x_{\mathcal{L}}$ and $y_{\mathcal{L}}$ in equation (6.41), must be found. To do this we will rewrite the first argument in equation (6.41) as

$$s_{\mathcal{L}} = C_{\mathcal{L}} \left(\frac{s}{C} + K \right), \tag{6.62}$$

where $C_{\mathcal{L}} = 1/\sqrt{c_{\mathcal{L}}}$ is scaling factor of a basic clothoid stored in the lookup table. The second argument in (6.41) need not to be examined because its absolute value is always less than or equal to absolute value of the first argument.

Now we will examine the maximum value of the first term in (6.62). Substituting the worst case bound on the clothoid length (6.46) it can be written as

$$\frac{s}{C} = \min\left(\sqrt{2\Delta\theta_{max}}, \frac{s_{max}}{C}\right), \tag{6.63}$$

whose maximum is

$$\left[\frac{s}{C}\right]_{max} = \min\left(\sqrt{2\Delta\theta_{max}}, \frac{s_{max}}{C_{min}}\right), \tag{6.64}$$

where C_{min} is minimum clothoid scaling factor defined by equation (6.50).

Regarding the second term in (6.62), we have already determined its maximum value by determining the maximum factor K in equation (6.59). Although both terms in (6.62) cannot be at their maximums instantaneously, for the simplicity we take the worst case and by using equation (6.64) write the maximum argument of the lookup table as

$$s_{\mathcal{L}max} = C_{\mathcal{L}}\left(\min\left(\sqrt{2\Delta\theta_{max}}, \frac{s_{max}}{C_{min}}\right) + K_{max}\right). \tag{6.65}$$

Another issue to consider is that values $x_{\mathcal{L}}$ and $y_{\mathcal{L}}$ returned by the lookup table in equation (6.41) will contain the interpolation error. Even worse, whatever error is introduced by the interpolation, it can be increased because of scaling by factor $C/C_{\mathcal{L}}$ in equation (6.41), which can be considerably larger than one.

Based on approximation error definition (6.48), the interpolation error introduced in equation (6.41) will in the worst case result with the following approximation error

$$e = 2\sqrt{2}\frac{C}{C_{\mathcal{L}}}e_{im}, \tag{6.66}$$

where e_{im} is maximum error that can be introduced by the interpolation. If we bound the maximum allowed error as $e \leq e_{max}$, consider the worst case $C = C_{min}$, and substitute maximum circular interpolation error (6.42) into (6.66), the following condition on the lookup-table maximum sampling interval is obtained

$$\Delta s_{\mathcal{L}max} = 2.8062 C_{\mathcal{L}}\sqrt[3]{\frac{e_{max}}{C_{min}}}. \tag{6.67}$$

By dividing the minimum required length of the lookup table (6.65) and maximum sampling interval, we obtain minimum number of points in the lookup table. We can notice that this minimum number of points is invariant of basic clothoid scaling factor $C_{\mathcal{L}}$.

The condition (6.43) on maximum allowed orientation change between neighbor points should also be considered. By introducing scaling factor $C_{\mathcal{L}}$ and noticing that $2s_{\mathcal{L}max} \gg \Delta s_{\mathcal{L}}$, condition (6.43) can be rewritten as

$$\frac{\Delta s_{\mathcal{L}} s_{\mathcal{L}max}}{C_{\mathcal{L}}^2} \leq \Delta\theta_{\mathcal{L}max}. \tag{6.68}$$

By substituting (6.65) and (6.67) into condition (6.68), we can notice that this condition is also invariant of $C_{\mathcal{L}}$. Therefore, basic clothoid scaling factor $C_{\mathcal{L}}$ is redundant so that any value of it can be chosen. To simplify calculations, the choice $C_{\mathcal{L}} = 1$ is used in the sequel.

The final procedure for determining the lookup table parameters can now be summarized as follows

1. Based on condition (6.65), choose the basic clothoid length $s_{\mathcal{L}}$ so that

$$s_{\mathcal{L}} \geq \min\left(\sqrt{2\Delta\theta_{max}}, \frac{s_{max}}{C_{min}}\right) + K_{max}. \tag{6.69}$$

2. Based on chosen $s_{\mathcal{L}}$ and conditions (6.67) and (6.68), choose basic clothoid sampling interval $\Delta s_{\mathcal{L}}$ so that

$$\Delta s_{\mathcal{L}} \leq \min\left(2.8062\sqrt[3]{\frac{e_{max}}{C_{min}}}, \frac{\Delta\theta_{\mathcal{L}max}}{s_{\mathcal{L}}}\right). \tag{6.70}$$

6.6.4 Example for the Pioneer 3DX Robot

An example of determining the lookup-table parameters for the Pioneer 3DX robot navigating in indoor spaces will be given. The radius of the Pioneer 3DX robot is approximately 20 cm. Regarding the size of the robot and precision of its position measurement, choosing the maximum allowed clothoid approximation error as $e_{max} = 1$ mm should be more than satisfactory.

Determination of a required set of clothoids. We choose maximum orientation change of a single clothoid $\Delta\theta_{max} = \pi/2$. Maximum allowed clothoid length that we allow is $s_{max} = 5$ m.

The minimum scaling factor is obtained by using condition (6.50) that bounds error of approximating a clothoid by a point. Using Matlab's Optimization and Symbolic toolboxes it is obtained $C_{min} = 5.9447 \cdot 10^{-4}$.

The maximum scaling factor is obtained using condition (6.52) that bounds error of approximating a clothoid by a line, so that it is obtained $C_{max} = 144.34$.

The maximum factor K is found by solving equation (6.60) in order to bound the error of approximating a clothoid by a circular arc, and it is obtained $K_{max} = 44.309$.

Determination of the lookup-table parameters. Using condition (6.69) a required length of the basic clothoid that will be stored in the lookup table is determined. It is obtained $s_{\mathcal{L}} \geq 46.081$, and we choose $s_{\mathcal{L}} = 46.1$.

Next, the sampling interval is obtained using (6.70). It is obtained $\Delta s_{\mathcal{L}} \leq 0.00378$, and we choose $\Delta s_{\mathcal{L}} = 0.0035$. The number of points in the lookup table is therefore $N_{\mathcal{L}} = \mathrm{ceil}(s_{\mathcal{L}}/\Delta s_{\mathcal{L}}) + 1 = 13173$, which is by no means acceptable for modern computers ($\approx$206 kB when stored in double precision).

6.6.5 Computation of Clothoid Coordinates

Now when the way is found how to compute points of any clothoid based on the clothoid in the lookup table, there still remains problem of computing the basic clothoid points in the lookup table. Therefore an appropriate numerical procedure must be utilized to solve equations (6.19b) and (6.19c). Here the computational efficiency is not of big importance because lookup table can be computed offline in the initialization stage.

As discussed in introduction part of Section 6.6, algorithms for numerical integration are appropriate for this task, since previously computed results are propagated to obtain new solution. In this way each successive call of the integration procedure benefits from the results obtained in the previous call, as opposed to methods that compute clothoid points in a single point.

For this task the Runge-Kutta numerical integration method [122] is chosen. The procedure is shown in Algorithm 6.6.1 called COMPUTECLOTHOIDCOORDINATES. In order to compute coordinates of clothoid points in $\mathbb{R}^2$, the algorithm numerically integrates equations (6.19b) and (6.19c).

The inputs of the algorithm are $s_{\mathcal{L}}$ and $\Delta s_{\mathcal{L}}$, which denote the basic clothoid length and sampling interval that is used as integration step, respectively. The output is the lookup table containing two arrays for x and y coordinates. The algorithm first determines number of points that the lookup table should contain and then allocates the table for coordinates storage. Numerical integration error becomes lower as internal step at which the distance parameter s increments becomes lower, therefore it is set to 10 times lower value than parameter Δs. In each step of the *for* loop it is first checked if the current values of x and y should be stored in the lookup table. For this check, an equality relation "==" is avoided because of possible numerical errors, and relation "<" is used instead (indexes for x and y arrays in the table start from zero). Finally, the coefficients $k_1, \ldots, k_4$ of Runge-Kutta method are separately computed for x and for y. Those coefficients are then used to obtain increments of x and y in current integration step.

Algorithm 6.6.1: COMPUTECLOTHOIDCOORDINATES

Input: $s_{\mathcal{L}}, \Delta s_{\mathcal{L}}$
Output: $\mathcal{L}$: lookup table with arrays of x and y coordinates

```
$nPoints = \text{CEIL}(s_{\mathcal{L}}/\Delta s_{\mathcal{L}})$;
$\mathcal{L} = \text{ALLOCATETABLE}(nPoints)$;
$sStep = \Delta s_{\mathcal{L}}/10$;
$iPoint = 0$;
$x = 0, y = 0$;
$s = 0$;
$c = 1$;
while $(iPoint < nPoints)$
  do
    if $(|s - iPoint \cdot \Delta s_{\mathcal{L}}| < sStep/10)$
      then
        $\mathcal{L}.x[iPoint] = x$;
        $\mathcal{L}.y[iPoint] = y$;
        $iPoint = iPoint + 1$;

    $k1 = \cos(c/2 \cdot s^2)$;
    $k2 = \cos(c/2 \cdot (s + sStep/2 \cdot k1)^2)$;
    $k3 = \cos(c/2 \cdot (s + sStep/2 \cdot k2)^2)$;
    $k4 = \cos(c/2 \cdot (s + sStep \cdot k3)^2)$;
    $x = x + sStep/6 \cdot (k1 + 2 * k2 + 2 * k3 + k4)$;

    $k1 = \sin(c/2 \cdot s^2)$;
    $k2 = \sin(c/2 \cdot (s + sStep/2 \cdot k1)^2)$;
    $k3 = \sin(c/2 \cdot (s + sStep/2 \cdot k2)^2)$;
    $k4 = \sin(c/2 \cdot (s + sStep \cdot k3)^2)$;
    $y = y + sStep/6 \cdot (k1 + 2 * k2 + 2 * k3 + k4)$;

    $s = s + sStep$;
```

In the future implementation it is planned to replace Runge-Kutta method by a more elaborated numerical integration method with variable integration step. This will result with better performance and accuracy.

In order to query coordinates of the basic clothoid from the lookup table, a procedure called GETBASICCLOTHOIDCOORDINATES is designed, whose pseudocode is enlisted in Algorithm 6.6.2. Input of the algorithm is distance s and outputs are basic clothoid coordinates $(x_{\mathcal{L}}, y_{\mathcal{L}})$ at distance s. Values $s_{\mathcal{L}}$ and $\Delta s_{\mathcal{L}}$

that are used in the algorithm are parameters of the lookup table.

Algorithm 6.6.2: GetBasicClothoidCoordinates

```
Input: $s$
Output: $x_{\mathcal{L}}, y_{\mathcal{L}}$

$i_1 = \text{Floor}(|s|/\Delta s_{\mathcal{L}})$;

if $(i_1 \geq nPoints - 1)$
  then $(x_{\mathcal{L}}, y_{\mathcal{L}}) = \text{Extrapolate}(s)$;
  else
    comment: Circular interpolation between points
    $i_2 = i_1 + 1$;
    if $(|s| > 10^{-5})$
      then $r = 1/|s|$;
      else $r = 2/(i_1 + i_2)/\Delta s_{\mathcal{L}}$;
    $x_1 = \mathcal{L}.x(i_1); y_1 = \mathcal{L}.y(i_1)$;
    $x_2 = \mathcal{L}.x(i_2); y_2 = \mathcal{L}.y(i_2)$;
    $A = y_1 - y_2; B = x_2 - x_1$;
    $x_c = (x_1 + x_2 + A\sqrt{4r^2/(A^2 + B^2) - 1})/2$;
    $y_c = (y_1 + y_2 + B\sqrt{4r^2/(A^2 + B^2) - 1})/2$;
    $\theta_1 = \text{atan2}(y_1 - y_c, x_1 - x_c)$;
    $\theta_2 = \text{atan2}(y_2 - y_c, x_2 - x_c)$;
    $p = (|s| - i_1 \cdot \Delta s_{\mathcal{L}})/\Delta s_{\mathcal{L}}$;
    $\theta = \theta_1 + p(\theta_2 - \theta_1)$;
    $x_{\mathcal{L}} = x_c + r\cos(\theta)$;
    $y_{\mathcal{L}} = y_c + r\sin(\theta)$;

if $(s < 0)$
  then
    comment: Return symmetrical point
    $x_{\mathcal{L}} = -x_{\mathcal{L}}$;
    $y_{\mathcal{L}} = -y_{\mathcal{L}}$;
```

6.6.6 Querying the Clothoid Coordinates

The algorithm first checks if the distance s is greater than the range stored in the table. In this case algorithm performs an extrapolation. Normally this shouldn't happen as the planner uses clothoids of limited length, however the extrapolation should be added for the sake of completeness. Otherwise, the algorithm performs a circular interpolation between coordinates stored in the table. As the basic clothoid is symmetrical with the origin of (x, y) plane as the center of symmetry, if distance s is negative a symmetrical point is returned in the last stage of the algorithm.

Having designed the algorithm GETBASICCLOTHOIDCOORDINATES for basic clothoid coordinates retrieval, it is now easy to design a higher level procedure that queries coordinates of a general clothoid. This algorithm should first check if the parameters of a general clothoid are in range of allowed parameters. If the answer is positive, it calls algorithm GETBASICCLOTHOIDCOORDINATES and uses equation (6.40) to compute the final coordinates. Otherwise, if the parameters of a general clothoid are out of range, it approximates a general clothoid by a point, a line or a circle, depending on value of parameters (see Subsection 6.6.2 for details).

6.7 Finding Intersections between a Clothoid and a Line

As discussed in Chapter 5, in this work obstacles are represented by polygons, which themself consist of straight lines. Therefore, to perform a collision check of clothoid path segments it is necessary to design algorithm that checks if a clothoid intersects a straight line and is also capable of computing the intersection points.

As we have bounded the maximum orientation change of a clothoid, for practical purposes it is sufficient to check only clothoids whose orientation change is less than π. Also, the analysis will be restricted only to clothoids with non-zero sharpness.[2] Further, only clothoids with the same signs of sharpness and initial curvature are considered. Such clothoids have tangent orientation that is monotonic function in s, i.e. the corresponding path steers only to the left, or only to the right, but not both. This ensures that there are no more than two intersection points with line, as illustrated in Figure 6.6. This restriction does not result with loss of generality since more complex clothoid can always be decomposed to simpler clothoids that fulfill this condition.

The algorithm should first determine how many intersection points between a clothoid and a line exist, if any. Let a line be given by an implicit equation

$$Ax + By + C = 0, \tag{6.71}$$

where A, B and C are real coefficients, and let a clothoid be given by a parametric equation (6.15). Let the range of parameter s of a clothoid be in interval $s \in [s_1, s_2]$, so that $s_1 = 0, s_2 > 0$ and $\theta(s_2) - \theta(s_1) < \pi$, where $\theta(s)$ is the clothoid tangent orientation given by equation (6.14).

Additionally, an auxiliary function $f(s)$ will be defined that will be used to determine a number of intersection points. It is obtained by substituting the

[2]Otherwise the clothoid becomes a line, which is easy to check for collision.

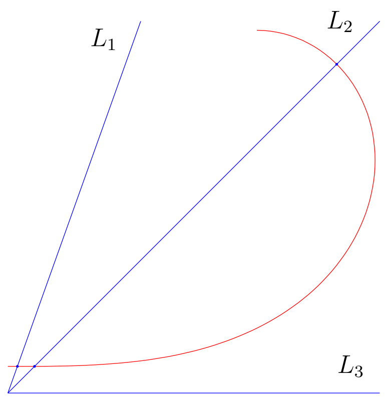


Figure 6.6. *Possible cases of intersection of a single-turn clothoid and a line. If orientation change of a clothoid is limited to π, there can be only one intersection point (line L_1), two intersection points (line L_2), or no intersections at all (line L_3).*

clothoid coordinates given by equation (6.15) into equation of the line (6.71) so that we have

$$f(s) = A \cdot \mathrm{Clx}(s) + B \cdot \mathrm{Cly}(s) + C. \tag{6.72}$$

Based on this function, a criterion for determining a number of intersection points is given in Table 6.1. The intervals of parameter s where the intersection(s) reside are given in the table, too. It can be shown that a necessary condition (but not sufficient) for the case of two intersection points is the existence of clothoid parameter $s_t \in (s_1, s_2)$ so that

$$\tan(\theta(s_t)) = -A/B, \tag{6.73}$$

i.e. the tangent direction of a clothoid in this point is equal to the direction of a line. If s_t exists, in general the first intersection will always be in interval $[s_1, s_t]$ of the clothoid, while the second intersection will reside in interval $[s_t, s_2]$.

To compute intersection points, the roots of function $f(s)$ given by (6.72) must be found, i.e. the equation $f(s) = 0$ must be solved. As the function (6.72) is transcendental, a numerical root-finding method must be used. The bisection method is the safe choice for this purpose because it is guaranteed to always converge if the solution exists [122]. However, the bisection method's rate of convergence is slow. On the other side, since a derivative of the function (6.72) can be efficiently evaluated, the Newton-Raphson method can be used that has substantially higher rate of convergence, but its convergence is not guaranteed even if solution exists. Therefore a fail-safe algorithm is used that utilizes a combination of both bisection and Newton-Raphson method [122].

$f(s_1)$	$f(s_2)$	$f(s_t)$	NI	$int1$	$int2$
> 0	> 0	n.a.	0		
< 0	< 0	n.a.	0		
> 0	> 0	> 0	0		
< 0	< 0	< 0	0		
> 0	< 0	n.a.	1	(s_1, s_2)	
< 0	> 0	n.a.	1	(s_1, s_2)	
$= 0$	$\neq 0$	n.a.	1	s_1	
$\neq 0$	$= 0$	n.a.	1	s_2	
$= 0$	> 0	> 0	1	s_1	
$= 0$	< 0	< 0	1	s_1	
> 0	$= 0$	> 0	1	s_2	
< 0	$= 0$	< 0	1	s_2	
$\neq 0$	$\neq 0$	$=0$	1	s_t	
> 0	< 0	< 0	1	(s_1, s_t)	
> 0	< 0	> 0	1	(s_t, s_2)	
< 0	> 0	< 0	1	(s_t, s_2)	
< 0	> 0	> 0	1	(s_1, s_t)	
> 0	> 0	< 0	2	(s_1, s_t)	(s_t, s_2)
< 0	< 0	> 0	2	(s_1, s_t)	(s_t, s_2)
$= 0$	$= 0$	$\neq 0$	2	s_1	s_2
$= 0$	> 0	< 0	2	s_1	(s_t, s_2)
$= 0$	< 0	> 0	2	s_1	(s_t, s_2)
> 0	$= 0$	< 0	2	(s_1, s_t)	s_2
< 0	$= 0$	> 0	2	(s_1, s_t)	s_2

Table 6.1. *Criterion for determining number of intersection points between a clothoid and a line based on the sign of function* (6.72). *Legend: NI – number of intersection points; $int1$, $int2$ – intervals of parameter s where the corresponding intersection reside; n.a. – point s_t does not exist. Knowing the intervals $int1$ and $int2$ is a prerequisite for numerical methods that compute intersection points.*

6.8 Smoothing Sharp Turn by a Clothoid Pair

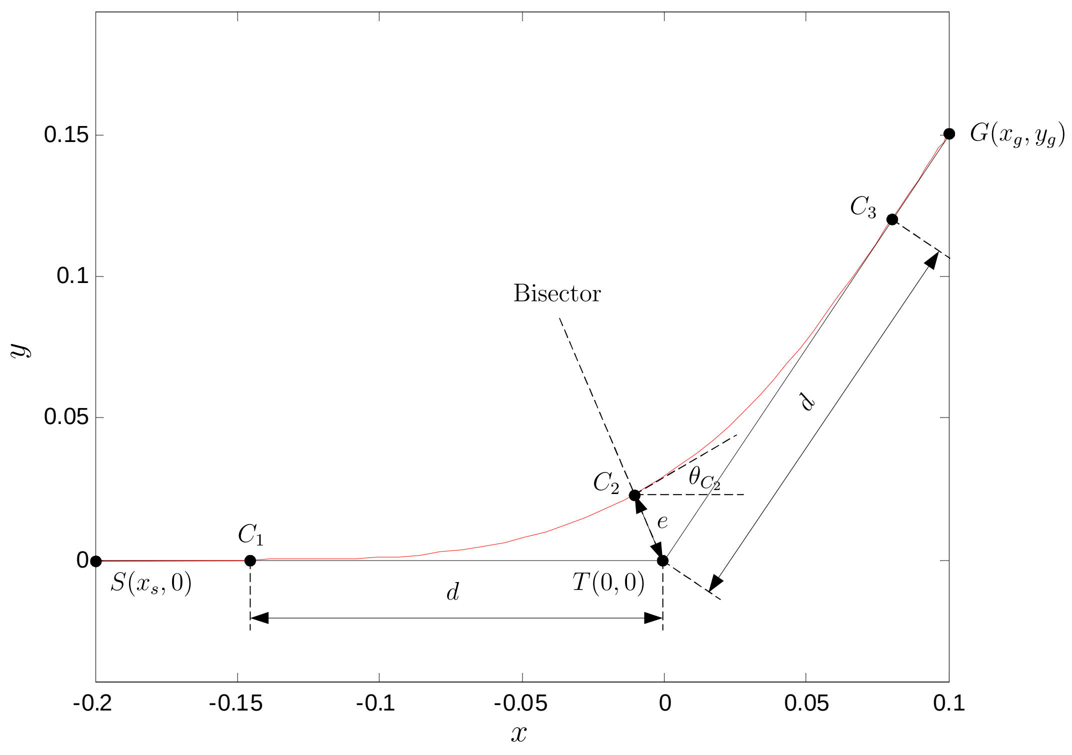
Now when basic algorithms for general clothoid calculations have been established, we have arrived to the point where the clothoid could be applied for path smooting. The simplest case is smoothing of a sharp turn that consists of two straight line segments, so that the problem is how to compute parameters of two clothoids that ensure smooth transition between two lines. This corresponds to the case where initial and final path curvatures are equal to zero. In the literature process of smoothing piecewise linear segments is commonly called blending (see e.g. [151]). With the use of clothoids a G^0 continuous curve is blended to G^2 continuous curve.

The input to a corresponding smoothing algorithm is a sharp turn given by two straight line segments. An example is shown in Figure 6.7, where the turn is given by points S, T and G in the (x, y) plane, which denote the start position, the location of the sharp turn, and the goal position, respectively. For simplicity, it is assumed that point T is in the origin of the (x, y) plane, point S is on the negative x-axis, while G can be anywhere, under condition that three point are not collinear. No generality is lost with this simplification, because any particular case can be transformed to the described one by an appropriate set of translations and rotations. The output of the algorithm are parameters of the two clothoids that build a final, smoothed path. The procedure is called SmoothSharpTurn and its pseudocode is enlisted in Algorithm 6.8.1.

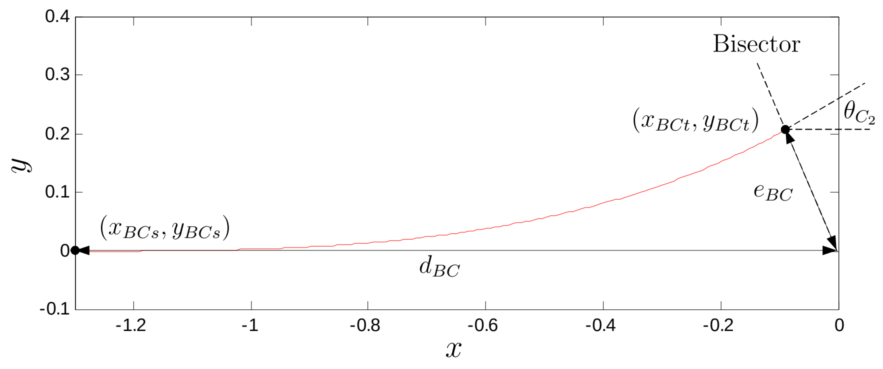
In step (1) of the algorithm it is being checked if the turn occurs in the negative direction (i.e. the goal point is in negative y half-plane). The case of negative turn can be viewed in the same way as the positive turn because of the symmetry with respect to x-axis. Thus, the goal point is mirrored to positive y half-plane and the appropriate flag is set in order to recover back to negative turn later.

To enable collision avoidance, the deviation of the smoothed path from the original path can be controlled by means of two input parameters, namely d_{max} and e_{max}. The parameter d_{max} denotes the maximum allowed distance between clothoid starting point and point T, denoted as d in Figure 6.7. This distance obviously cannot be longer than distances $|ST|$ and $|TG|$, which is ensured in step (2) of the algorithm. The parameter e_{max} denotes the maximum allowed distance of the closest point on the clothoid to point T, denoted as e in Figure 6.7. To robot will be able to reach higher velocity if the path curvature is lower. This is obtained by maximizing distances d and e, but without exceeding the bounds d_{max} and e_{max}.

Algorithm first solves the problem using the basic clothoid and resulting path is then scaled to match the original problem. The basic clothoid is depicted in Figure 6.7 (b). A path is smoothed using a pair of two symmetrical clothoids, the first from point C_1 to C_2, and the second from point C_2 to C_3. Because of symmetry, the amount of orientation change is the same for both clothoids. Therefore, the transition point C_2 from the first to the second clothoid will lie at the bisector of the angle spanned by lines ST and TG. The tangent angle at the transition point θ_{C_2} is determined in step (3) of the algorithm.



(a)



(b)

Figure 6.7. *Smoothing of sharp turn by clothoid pair.* (a) *A sharp turn is given by two straight lines: $|ST|$ from start to turn point and $|TG|$ from turn point to goal. A final smoothed path, shown as red curve, consists of two straight line segments $|SC_1|$ and $|C_3G|$, first clothoid between points C_1 and C_2, and second clothoid between C_2 and C_3.* (b) *Basic clothoid used as auxiliary curve for computing parameters of the clothoid pair.*

Algorithm 6.8.1: SMOOTHSHARPTURN

```
Input: $x_s, x_g, y_g, d_{max}, e_{max}$
Output:
    $x_{0C_1}, y_{0C_1}, \theta_{0C_1}, \kappa_{0C_1}, c_{C_1}, Length_{C_1}$ : first clothoid parameters
    $x_{0C_2}, y_{0C_2}, \theta_{0C_2}, \kappa_{0C_2}, c_{C_2}, Length_{C_2}$ : second clothoid parameters

(1) if $(y_g < 0)$
      then $\begin{cases} NegativeTurn = \textbf{ true };\\ y_g = -y_g; \end{cases}$
      else $NegativeTurn = \textbf{ false };$

(2) $d_{max} = \min(d_{max}, |\mathrm{ST}|, |\mathrm{TG}|);$
(3) $\theta_{C_2} = \arctan(y_g/x_g)/2;$
(4) $s_{BC} = \sqrt{2 \cdot \theta_{C_2}};$
(5) $(x_{BCs}, y_{BCs}, x_{BCt}, y_{BCt}, d_{BC}, e_{BC}) = \text{COMPUTEBASICCLOTHOID}(s_{BC}, \theta_{C_2});$

(6) $ScaleFactor = \max(d_{BC}/d_{max}, e_{BC}/e_{max});$
    $s = s_{BC}/ScaleFactor;$
    $x_{C_2} = x_{BCt}/ScaleFactor;$   $y_{C_2} = y_{BCt}/ScaleFactor;$
    $d = d_{BC}/ScaleFactor;$
    $e = e_{BC}/ScaleFactor;$

(7) $x_{0C_1} = -d; y_{0C_1} = 0;$
    $\theta_{0C_1} = 0;$
    $\kappa_{0C_1} = 0;$
    $c_{C_1} = ScaleRatio^2;$
    $Length_{C_1} = s;$
    if $(NegativeTurn)$
      then $c_{C_1} = -c_{C_1};$

(8) $x_{0C_2} = x_{C_2}; y_{0C_2} = y_{C_2};$
    $\theta_{0C_2} = \theta_{C_2};$
    $c_{C_2} = -ScaleRatio^2;$
    $Length_{C_2} = s;$
    if $(NegativeTurn)$
      then $\begin{cases} y_{0C_2} = -y_{0C_2};\\ \theta_{0C_2} = -\theta_{0C_2};\\ c_{C_2} = -c_{C_2}; \end{cases}$
```

In step (4) algorithm computes required length s_{BC} of the basic clothoid so that, at the point where it reaches the bisector, its tangent angle is equal to angle

θ_{C_2}, as can be seen in Figure 6.7 (b). In step (5), a procedure ComputeBasicClothoid is called, which computes start point (x_{BCs}, y_{BCs}), transition point (x_{BCt}, y_{BCt}) and distances d_{BC} and e_{BC} of the basic clothoid. This procedure is straightforward to implement using procedure GetBasicClothoidCoordinates given in Algorithm 6.6.2, so that its details are not given here. In general case the obtained sizes d_{BC} and e_{BC} will not fulfil constraints d_{max} and e_{max}.

This is solved by performing scaling of the basic clothoid in step (6) of the algorithm. Scaling factor is determined as a maximum of ratios d_{BC}/d_{max} and e_{BC}/e_{max}. Therefore the distance d will be always lower than or equal to distances $|ST|$ and $|TG|$, so that the final path, besides of two clothoids, can also consist of two straight line segments, SC_1 and C_3G.

Finally, parameters of both clothoids are computed in steps (7) and (8) of the algorithm, respectively. The second clothoid is obtained by reflecting the first one with respect to the bisector. Tangent angle and curvature continuity between the two clothoids (i.e. in the point C_2) follow by symmetry.

The described problem always has a solution. Even if one of the constraints d_{max} and e_{max} is equal to zero, the algorithm will produce a solution that consists of two clothoids with infinite maximum curvature, which actually represents a rotation in place.

The algorithm SmoothSharpTurn, as implemented in pseudocode given in Algorithm 6.8.1, results with maximum change of the orientation that is always lower than π. However, if required, algorithm can be extended to handle the cases where orientation change is in the range $[\pi, 2\pi)$. Further, the described implementation takes obstacles into account only implicitly, via parameter e_{max}. This can be limitation in some cases so that the algorithm should be extended to explicitly handle polygonal obstacles. This is obtained by using algorithm that finds an intersection of a clothoid and a line described in Section 6.7. In this case the clothoid path can be fine tuned so that it is placed in the middle of the passage bounded by inner and outer obstacles. An example of the algorithm output is illustrated in Figure 6.8.

This extension is especially useful if multiple successive short turns in the same direction are required. This can be the case when circular obstacles are approximated by polygons. When using the visibility graph with such polygonal representations, the resulting path may consist of many short line segments. If every pair of segments is smoothed separately, the curvature will change on every segment to some maximum value and back to zero, so that the robot will have a hard time following such a path. Therefore it makes sense to combine multiple short turns into the one turn if possible. In this way the safe distance to the obstacles is no more guaranteed implicitly by the path planner, so that the obstacles must be handled explicitly when smoothing a path. The extended algorithm illustrated in Figure 6.8 does exactly that.

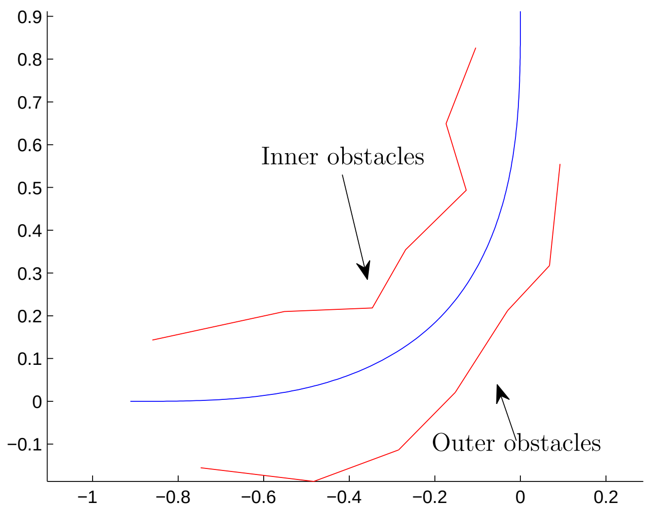


Figure 6.8. *The clothoid pair (blue) is tuned so that the minimum distance between the path and inner obstacles is approximately the same as the minimum distance between the path and outer obstacles.*

6.9 Connecting Circle and Line by Two Clothoids

Most of the motion planning approaches that are found in the literature assume that both initial and final robot velocities are equal to zero. This kind of smoothing corresponds to *static smoothing*. For static smoothing initial and final path segments are always straight line segments, so that the algorithm SmoothSharpTurn described in Section 6.8 does all the job. However, in dynamic environments we often encounter situations when current plan becomes invalid due to changes in the environment or change of the goal configuration. Then replanning must be performed. But now initial linear and angular velocities cannot be assumed to be zero because the robot was already moving while executing the previous plan. If initial angular velocity is non-zero, the initial path segment is no more line, but the circle.

This raises the question on how to extend the algorithm SmoothSharpTurn in order to account for non-zero initial curvature of a path—this is called a *dynamic smoothing* and the problem is considerably harder than static smoothing. The problem is equivalent to problem of connecting circle and line in order to obtain a G^2 continuous curve. When using clothoids, unlike static smoothing, this problem does not always have a solution. The solution path is not unique and may consist of various combinations of clothoids, circular arcs and straight lines. Here it will be solved by using two symmetric clothoids.

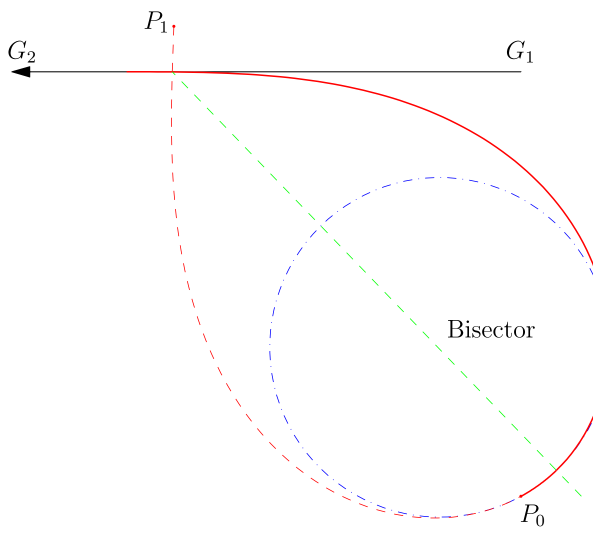


Figure 6.9. *An example of connecting circle and line G_1G_2 using two clothoids. The initial position of the robot is in point P_0 on the circle. The circle has radius that is inverse of initial curvature, i.e. $r = |1/\kappa_{P_0}|$.*

An example of the solved problem is shown in Figure 6.9. The algorithm developed for this problem is called ConnectCircleLine2C and the pseudocode is given in Algorithm 6.9.1. The input to the algorithm is an initial state denoted as P_0 with coordinates (x_{P_0}, y_{P_0}), tangent direction θ_{P_0} and curvature $\kappa_{P_0} \neq 0$. The goal line is given by points G_1 and G_2, where motion direction is from G_1 to G_2. The planned path must end on the line through points G_1 and $G2$, but not after point G_2. Outputs of the algorithm are parameters of two clothoids that are solution to the problem and the flag that reports success or failure.

In the function CheckNecessaryConditions the algorithm checks if necessary conditions are fulfilled in order to perform fast heuristics if it is worth to further trying to solve the problem. The solution may exist even if the necessary conditions are not fulfilled, but those are solutions that we are not interested, as they typically result in extremely long paths or extremely sharp turns. The first necessary condition is that the goal point is on the same side of the robot where the center of the circle of curvature is located. In the example in Figure 6.9 this means that at least one of the points G_1 and G_2 is to the left of the tangent vector in point P_0. The other necessary condition is that start point P_0 is to the left of vector G_1G_2 if the initial curvature is positive, and right otherwise.

To compute parameters of both clothoids, it is sufficient to find required sharpness of the first (entering) clothoid—it is then straightforward to determine remaining parameters of both clothoids. Actually, instead of sharpness, scaling factor C is chosen as unknown variable because it directly determines size of the clothoid and better numerical properties of the algorithms are achieved in this way. A single equation of the form $f(C) = 0$ has to be solved in order to obtain a solution. As it will be shown, function f is nonlinear so that numerical root-finding method must be utilized. Those methods require that a root is bracketed in some interval (C_1, C_2) so that $f(C_1)$ and $f(C_2)$ have opposite signs. This is done in function BracketSolution of the algorithm.

Algorithm 6.9.1: ConnectCircleLine2C

```
Input:
        $P_0$ : initial state with parameters $(x, y, \theta, \kappa)$
        $G_1, G_2$ : begin and end points of goal line with parameters $(x, y)$
Output:
        $CL1, CL2$ : two clothoids that connect circle and line with parameters
                $(x_0, y_0, \theta_0, \kappa_0, c, C, length)$
        Success flag ( true or false )

if fail CheckNecessaryConditions($P_0, G_1, G_2$)
  then return ( false );
if fail $(C_1, C_2)$ = BracketSolution($P_0, G_1, G_2$)
  then return ( false );
if fail $(C)$ = FindRoot($P_0, G_1, G_2, C_1, C_2$)
  then return ( false );
if fail CheckAdmissibility($P_0, G_1, G_2, C$)
  then return ( false );
$(CL1, CL2)$ = ComputeClothoidParameters($P_0, G_1, G_2, C$)
return ( true );
```

The lower bracket C_1 is determined so that for a while it is assumed that initial curvature κ_{P_0} is zero. Then the problem is reduced to the problem of smoothing the sharp turn with a pair of clothoids from Section 6.8. If problem is additionally constrained so that beginning clothoid must strictly begin in point P_0, then correspondent Algorithm 6.8.1 gives us the lower bracket C_1. This is enabled by the fact that if the initial curvature is zero, the required sharpness of the clothoid is always higher, i.e. scaling factor is lower, compared to the case of non-zero curvature.

Upper bracket C_2 is obtained by limiting total orientation change of the turn to be less than 2π. Let's suppose a turn in the positive direction. Total orientation change is then obtained as

$$\Delta\theta = \theta_{G_1G_2} - \theta_{P_0}, \tag{6.74}$$

where $\theta_{G_1G_2}$ is direction of vector G_1G_2. As this orientation change is divided to both entering and exiting clothoid, the maximum absolute orientation change of any of these clothoids is less than π. Let point P_1 in Figure 6.9 denote a virtual begin point of the entering clothoid obtained by following the entering clothoid in the direction of decreasing curvature until the curvature becomes zero. The maximum orientation change of the portion of entering clothoid from P_1 to P_0 is

$$\Delta\theta_{P_1P_0max} = \min(2\pi - \Delta\theta, \pi). \tag{6.75}$$

By using (6.13) and (6.16) we obtain the upper bracket as

$$C_2 = \frac{\sqrt{2\Delta\theta_{P_1P_0max}}}{\kappa_{P_0}}. \tag{6.76}$$

It remains to define an appropriate function $f(C)$ whose root gives a solution of our problem. As a preliminary, let's first define a *bisector line* as an axis of symmetry between vector opposite to tangent vector in point P_1 and vector G_1G_2 (see Figure 6.9). Now we want to find a value of C for which the entering clothoid intersects the bisector so that in the point of intersection clothoid and bisector are perpendicular to each other. If such value exists, then the problem has a solution and the function $f(C)$ has a root in this value.

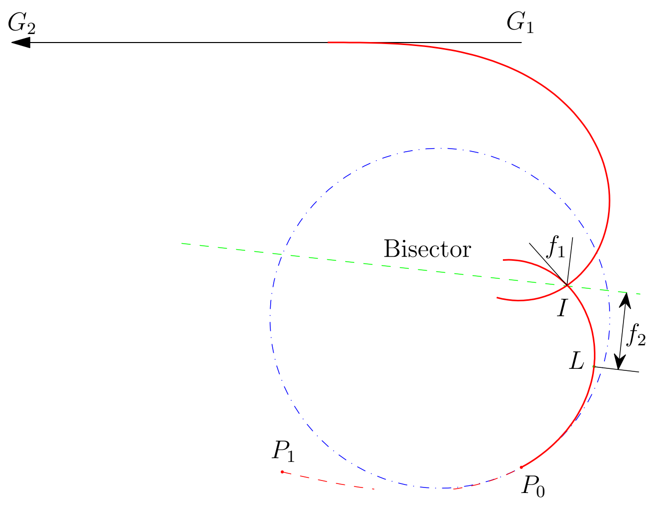


Figure 6.10. *An iteration of searching for the appropriate clothoid scaling, which is the root of function $f(C)$. For function $f(C)$, angular distance f_1 or Euclidean distance f_2 in the figure can be chosen. The scaling in the current iteration is lower than the root. At the root value, both f_1 and f_2 will be zero.*

The function $f(C)$ that has such a property can be defined in two ways. The first possibility is to first find an intersection point of clothoids with the bisector.

This is illustrated in Figure 6.10, where the same problem as in Figure 6.9 is used but lower scaling C was used. The mentioned intersection point is denoted as I in the figure. The output of the function $f(C)$ is then defined as a difference between clothoid orientation in intersection point I and orientation of the normal to the bisector, which is in the figure denoted as f_1. If the tangent of the clothoid and normal to the bisector are parallel, the solution is found and the output of the function is zero. However, this definition of $f(C)$ is problematic because for some scalings C there will be no intersections of clothoids and bisector so that function is undefined. The second problem is that to examine a function, an intersection between clothoid and a line must be found, which requires solving a nonlinear equation. This means that in every iteration of the root finding method, another root finding problem must be solved which can considerably increase the computation burden.

Algorithm 6.9.2: CriterionFunction

```
Input: $C, P0, G1, G2$
Output: $f$

$P1 = $ ComputeP1$(C, P0)$;
$P1TangentOpp = $ LineFromDirectionAndPoint$(P1.th + pi, P1.x, P1.y)$;
$G1G2 = $ LineFromPoints$(G1.x, G1.y, G2.x, G2.y)$;
if (IsParallelAndOppositeApprox$(P1TangentOpp, G1G2)$)
  then
    $distG1G2toP1 = $ GetDistanceToPoint$(G1G2, P1.x, P1.y)$;
    if (IsZeroApprox$(distG1G2toP1)$
      then $f = 0$;
      else if $(distG1G2toP1 * P0.k < 0)$
      then $out = inf$;
      else $out = -inf$;
  else
    $Bisector = $ GetSymmetricalLine$(P1TangentOpp, G1G2)$;
    $Cloth.x0 = P1.x$;    $Cloth.y0 = P1.y$;
    $Cloth.th0 = P1.th$;
    $Cloth.k0 = 0$;
    $Cloth.C = C$;
    $s = $ GetPointWithOrientation$(Cloth, Bisector.Direction - pi/2)$;
    $(xLat, yLat) = $ GetClothoidPoint$(Cloth, s)$;
    $distToBisector = $ GetDistanceToPoint$(Bisector, xLat, yLat)$;
    $f = -\operatorname{sgn}(C) * distToBisector$;
```

The second and better implementation of $f(C)$ is given in the Algorithm 6.9.2. Here a point on the clothoid is searched where the tangent is normal to the bisector line, which is denoted by point L in Figure 6.10. Then the value of the function is defined as the signed distance between point L and bisector (denoted as f_2 in Figure 6.10). This definition does not require searching of an intersection with the bisector line.

However, the problem with this definition is that distance f_2 can become infinite, as illustrated in Figure 6.11. This happens when vectors G_2G_1 and tangent vector in P_1 are parallel and have the same directions. Then the bisector line is in the infinity, and the output of the function can be $+\infty$, $-\infty$ or zero, depending on distance between vector G_1G_2 and point P_1. If at some value of scaling C this distance becomes approximately zero, then this value is a root so that value $f(C) = 0$ is returned in the function. This is because then the position of bisector line is undefined and any line normal to the vector G_1G_2 can be taken as the bisector. To obtain the correct solution, those cases are handled separately in Algorithm 6.9.2.

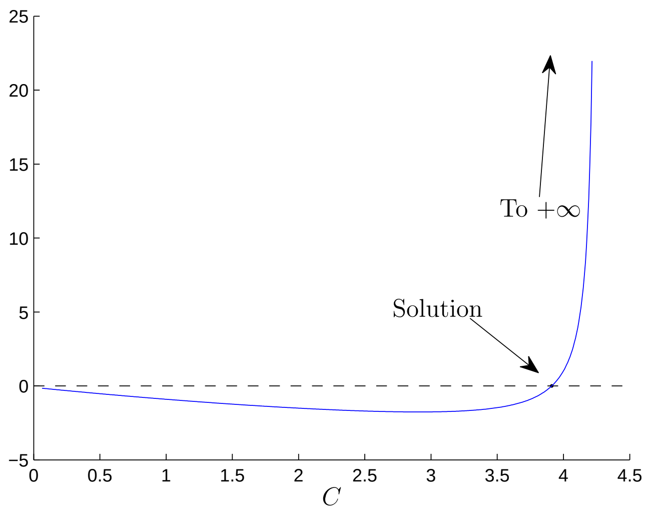


Figure 6.11. *Output of the function $f(C)$ obtained with Algorithm 6.9.2 for a query $P_0 = (0, 0, 0, 0.1)$, $G_1 = (6.5, -0.6)$ and $G_2 = (7.6, -0.7)$. Methods like false position or secant method are not adequate for this problem because the function goes to infinity. Newton-Raphson method would probably also have much difficulty because of the same reason. Although the bisection method rate of convergence is usually slower than other methods, in this particular case it is faster. It is also known that it is secure because it does not depend on output of the function, but only on its sign. In this example it needs 16 iterations to find a solution with precision of 10^{-4}.*

As in almost every computational geometry algorithm implementation, it is very important to take into consideration the limited precision of the computer floating point arithmetic, otherwise problems with the robustness may arise (see e.g. [145]). Therefore some functions in Algorithm 6.9.2, such as IsParalleLAndOppositeApprox, are evaluated approximately.

Now that function $f(C)$ is defined and its root is bracketed, it remains to find a solution to the equation $f(C) = 0$ which is done in function FindRoot in Algorithm 6.9.1. Not every root finding method is appropriate for this problem because the function $f(C)$ can go to infinity which is a serious problem for many root-finding methods. In such circumstances, surprisingly, the bisection method achieves faster rate of convergence than other, otherwise faster methods such as the false position method. The fact that function can go to infinity is not a problem for bisection method because it uses only a sign of the function value. Moreover, the bisection method is one that cannot fail; if there exists a solution, this method is guaranteed to find it.

The solution of $f(C)$ is always unique because the entering clothoid is constrained to pass exactly through the initial position P_0, as opposite to the case with zero initial curvature where initial position could be anywhere on the beginning line. It is left for future research to investigate if faster convergence can be obtained by combining the bisection method with the Newton-Raphson method.

If the function FindRoot reports success, the solution C to the equation $f(C) = 0$ is found. However, the solution may still not be admissible, so that it is additionally checked in function CheckAdmissibility in Algorithm 6.9.1 (ConnectCircleLine2C). Here the conditions are checked such as that point L in Figure 6.10 must lie after point P_0 on the path. Additionally, the produced path may not be collision-free so that it is tested using collision detection module. If the solution C is admissible, it is straightforward to compute remaining parameters of both required clothoids. This is done in function ComputeClothoidParameters in Algorithm 6.9.1.

6.10 Connecting Circle and Line by a Single Clothoid

In some circumstances it is possible to connect circle and line by a single clothoid. An example is shown in Figure 6.12. Assume that the point P_0 is initial position, so that the circle is determined with coordinates and tangent direction in point P_0, while its radius r is obtained as inverse of the initial curvature, i.e. $r = 1/|\kappa_{P_0}|$. Line G_1G_2 is the goal line. In Figure 6.12, the clothoid begins in point P_1 on the circle and ends on the line G_1G_2. As clothoid is not very flexible in number of parameters that can be tuned (only sharpness and length are available), additional circular arc must be used in order to obtain G^2 continuity, which is inserted between points P_0 and P_1.

To solve the problem, again a nonlinear equation $f(C) = 0$ has to be solved, whose root C represents, if it exists, the scaling of the required clothoid. In this case function $f(C)$ is defined as

$$f(C) = d(G_1G_2, M) - d(G_1G_2, M')$$

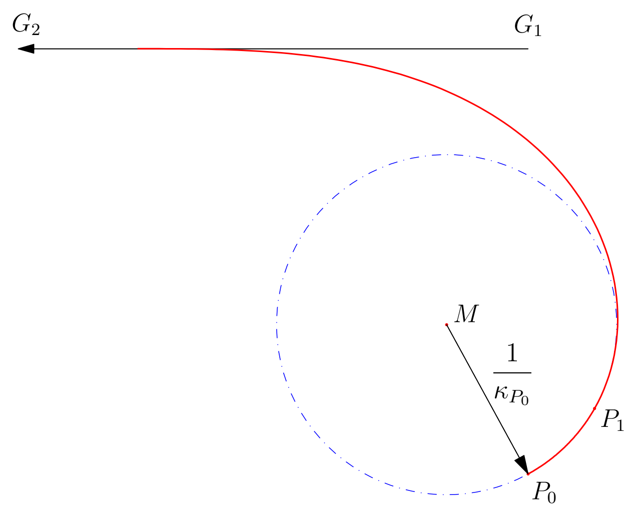


Figure 6.12. *An example of connecting circle with tangent in P_0 and the goal line G_1G_2 using a single clothoid. An additional circular arc is inserted between points P_0 and P_1.*

where $d(G_1G_2, M)$ and $d(G_1G_2, M')$ are signed Euclidean distances between vector G_1G_2 and points M and M', respectively (see Figure 6.13). The point M is the center of path curvature in the point P_0 and point M' is center of curvature of the clothoid in the point P_1 where it has the curvature equal to κ_{P_0}.

The coordinates of point M are

$$x_M = x_{P_0} - \frac{1}{\kappa_{P_0}} \sin\theta_{P_0}, \quad y_M = y_{P_0} + \frac{1}{\kappa_{P_0}} \cos\theta_{P_0}. \tag{6.77}$$

The point M' is center of curvature of the clothoid in the point P_1 where it has the curvature equal to κ_{P_0}. The parameter s of the clothoid in point P_1 is therefore

$$s_{P_1} = |\kappa_{P_0}|C^2. \tag{6.78}$$

The coordinates of point P_1 are now obtained as

$$x_{P_1} = \mathrm{Clx}(s_{P_1}), \quad y_{P_1} = \mathrm{Cly}(s_{P_1}), \tag{6.79}$$

where the coordinates of clothoid point $(\mathrm{Clx}(s), \mathrm{Cly}(s))$ are defined by equation (6.15), while its parameters are $(x_0, y_0, \theta_0, \kappa_0) = (x_{G_1}, y_{G_1}, \theta_{G_2G_1}, 0)$. Here the point G_1 was chosen as the begin point of the clothoid, but it can be any point on the line G_1G_2 as well, because only distance to G_1G_2 is important. The path tangent direction θ_{P_1} in point P_1 is opposite to clothoid tangent direction in P_1,

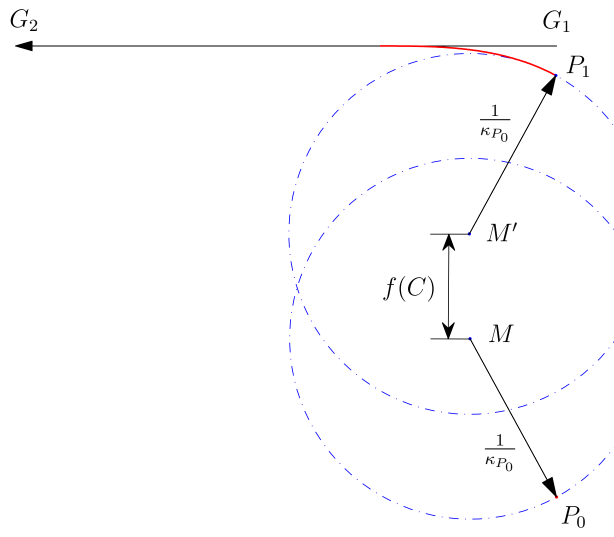


Figure 6.13. *An iteration of searching for the right clothoid scaling, which is the root of function $f(C)$ given by equation* (6.10). *The scaling in the displayed iteration is lower than the root. At the root value, function $f(C)$ becomes zero.*

so that

$$\theta_{P_1} = \theta_{G_2G_1} + \operatorname{sgn}(C)\frac{s_{P_1}^2}{2C^2} + \pi. \tag{6.80}$$

Finally, the coordinates of point M' are obtained as

$$x_{M'} = x_{P_1} - \frac{1}{\kappa_{P_0}}\sin\theta_{P_1}, \quad y_{M'} = y_{P_1} + \frac{1}{\kappa_{P_0}}\cos\theta_{P_1}. \tag{6.81}$$

The algorithm that solves the problem of connecting circle and line by a single clothoid is called ConnectCircleLine1C. Its overall structure is actually very similar to Algorithm 6.9.1 (ConnectCircleLine2C), except that the implementation details of the functions are different, so that pseudocode will not be given here.

Implementation of the function BracketSolution is also similar to one in algorithm ConnectCircleLine2C, and the absolute orientation change of the clothoid is again limited to π.

The problem of root finding in function FindRoot is here considerably easier compared to the case when two clothoids are used, since function $f(C)$ has no discontinuities, and its derivative can be easily found. Therefore, for root finding the Newton-Raphson method using derivative is used. For safety, its fail-safe combination with the bisection method is used. The function $f(C)$ as defined in equation (6.10) has the unique root.

Function CheckAdmissibility here checks if the point P_1 is before point P_0 on the path. Also, it is tested if the path is collision-free. If the path is not admissible, it is rejected.

An alternative approach to this problem is described by Meek and Walton [106], where instead of scaling factor, the tangent orientation of the clothoid is used as unknown variable.

6.11 Connecting Circle and Line by Three Clothoids

If replanning is required, algorithms ConnectCircleLine2C and ConnectCircleLine1C give a quick solution to the problem of connecting circle and line path segments. However there are configurations for which those algorithms may not have a solution (e.g. if robot is currently steering right, but the goal is to the left), solution may not be admissible (e.g. because of obstacles), or it may not be feasible (e.g. because of actuator limits). This is illustrated in Figure 6.14, where the robot initial configuration is in point P_0. Original shortest path connects initial configuration to the node G_1 of the roadmap. As current configuration in point P_0 has high curvature to the right, i.e. opposite of the goal line G_1G_2, both algorithms ConnectCircleLine1C and ConnectCircleLine2C fail to produce an acceptable path.[3]

In such situations three clothoids can be used in order to obtain more flexibility because in this way any circle and line can be connected. This may be obtained by first forcing the path to go straight. This transition from circle to line can be easily obtained by using a single clothoid called *transition clothoid*. The example is displayed in Figure 6.14, where the transition clothoid goes from initial point P_0 to point P_1.

After the path is made straight, the new shortest path is obtained by extending the straight line segment to point C_1 and dynamically inserting C_1 as a new node into the roadmap. In Figure 6.14 this results with the new shortest path (P_1-C_1-G_1). The new path segment (P_1-C_1-G_1) consist of straight lines so that it is easily smoothed using algorithm SmoothSharpTurn, resulting with line segment P_1P_2, and two clothoids from P_2 to P_3 and from P_3 to P_4.

Altogether tree clothoids were used to connect initial configuration in point P_0 with the roadmap point G_1, so that this is equivalent to connecting circle and line with three clothoids. This method gives more flexibility because in this way any circle and line can be connected.

However, in this case there is no more the unique solution as was the case

[3]Algorithm ConnectCircleLine2C, however, can find an admissible path, but it begins with very sharp turn which is not acceptable, except at very low speeds

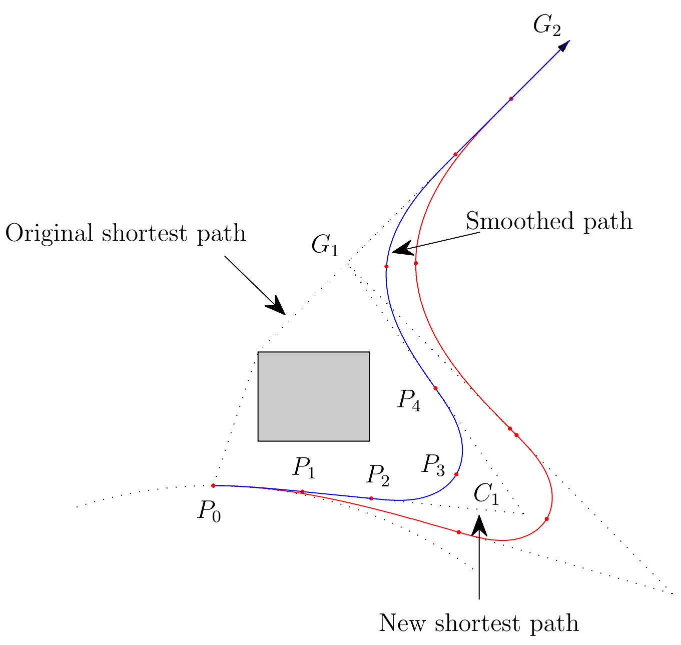


Figure 6.14. *Example of two paths constructed by* ConnectCircleLine3C *algorithm. Initial configuration is in point P_0, while the goal line goes from point G_1 to G_2 (G_1 and G_2 may e.g. be roadmap nodes). The blue path is a better choice as it is shorter. Red points in the figure denote transitions line-clothoid, clothoid-line and clothoid-clothoid. Original shortest path is also shown, which is not very useful in this case because current configuration cannot be connected to it. The new shortest path is constructed beginning from point P_1 where zero curvature is achieved.*

previously so that there is an infinite set of possible solutions. Usually some heuristics and trial and error method must be utilized to solve the problem in real time.

The smoothing algorithm alone in this case is simpler comparing to algorithms ConnectCircleLine2C and ConnectCircleLine1C because there is no need to solve any nonlinear equation. However, the whole problem is more complex and overall computation time is typically higher compared to previous algorithms, mainly due to the fact that a new node has to be inserted into the roadmap in every iteration. The correspondent algorithm is called ConnectCircleLine3C and pseudocode is given in Algorithm 6.11.1.

Algorithm first determines maximum allowed clothoid sharpness that can be used when transitioning to the straight line. As this requires knowledge about robot dynamics and actuator limits, this query is directed to function GetMaximumSharpness in the trajectory planning module.

In each iteration of the **for** loop in the algorithm, a new scaling factor from allowed range is picked based on heuristics. The allowed range of scalings is determined based on maximum sharpness and maximum allowed length of the transition clothoid.

Algorithm 6.11.1: ConnectCircleLine3C

```
Input: $ShortestPath, CurrentState$
Output: $SuccessFlag, SmoothedPath$

$cmax$ = TrajectoryPlanning->GetMaximumSharpness($CurrentState$);

for 1 to $MaxIterations$
  $C$ = PickScaling($cmax$);
  $ClothoidAndLine$ = StraightPath($CurrentState, C$);
  if ( fail CheckAdmissibility($ClothoidAndLine$))
    then continue ;
  PathPlanning->UpdatePath($ClothoidAndLine$);
  $ShortestPath$ = PathPlanning->GetShortestPath();
  $SmoothedPath$ = SmoothPath($ShortestPath$);
  if ( fail TrajectoryPlanning->CheckFeasibility($SmoothedPath$))
    then continue ;
    else return ( true );

return ( false );
```

Then for the current scaling the transition clothoid is computed in function StraigthPath. This function also returns a line where the clothoid lands, which is easily computed as tangent in point where curvature of the clothoid becomes zero. The obtained clothoid and line are then checked for admissibility, i.e. collision check is performed. If there is no collision, the endpoint of the line is inserted as a new node into the roadmap, and the new shortest path is computed. This path is then smoothed in function SmoothPath which internally calls SmoothSharpTurn algorithm.

Finally, the smoothed path is checked for feasibility, which again cannot be done by the path-smoothing module, but is directed to trajectory planning module. Here it is tested if the path can actually be tracked by the robot regarding its current velocity, actuator limits and other dynamic constraints. If the path is feasible, the algorithm terminates and reports success. Otherwise, the next iteration is tried, until maximum number of iterations is reached. If this happens, the failure is reported. Note that no optimality criterion is examined in the described algorithm, but this could easily be added so that the shortest, or even the fastest path can be searched for.

An example of two different paths produced by algorithm ConnectCircle-

Line3C are given in Figure 6.14. It can be seen that algorithm ConnectCircleLine3C successfully finds an admissible path. For illustration, another path produced by the same algorithm is shown with lower sharpness of the transition clothoid.

Another variant of this algorithm is possible that is less powerful but is potentially faster. In that variant the shortest path would not be replanned and therefore no roadmap update would be necessary. The idea is to directly construct three clothoids in order to connect to the current shortest path. In this case all three clothoids have to be explicitly checked for collision with obstacles. It is expected that in some circumstances that algorithm could give a solution that is more close to the optimum compared to described variant, so that it could be added to existing algorithms in the future.

It can be noticed that algorithm ConnectCircleLine3C works on higher level than any previously described algorithm—although it is located in the path-smoothing module, it calls some functions from path-planning and trajectory planning modules. This is because the solution is not unique, and path-smoothing module by itself cannot tell which path is feasible, nor it can replan the global path. This is one example where no strict algorithm flow of the decoupled approach is followed—control is directed back or forth to other modules as necessary.

6.12 Emergency Stop

In some situations changes in the environment happen to be so drastic that none of the described replanning algorithms can find acceptable solution—this is most likely to be caused by fast moving obstacles, change of goal configuration, or by algorithm incompleteness. Then the last resort is to brake as fast as possible and possibly to pick a steering direction that is clearest of obstacles—this is called *emergency stop* maneuver. Similar situations often occur in everyday life, such as in traffic, sport, or navigating trough spaces crowded with people. If some obstacle crosses our path unexpectedly, we typically stop and continue our motion when the path becomes clear again.

Once the robot has stopped, it can plan an alternative path to the goal if one exists, wait some time until path becomes clear again, or decide to pick alternative goal position. Path smoothing is always easier if the robot is in still state because the feasible trajectory along the smoothed path can always be found if one exists. This is enabled by the fact that no excessive space is needed for robot to continue its prior motion and robot can simply reorientate itself in path direction (remember that here we do not impose any constraints on maximum path curvature) and then normally execute the rest of the path.

In current implementation the EmergencyStop algorithm is implemented so that references of both linear and angular velocity are set to zero and no further planning is performed. This is because here we only want to stop as fast as possible, without predicting the motion to the goal. However, while doing so, it would be desirable to also pick a path that is free from obstacles if possible. For such purpose a pure reactive algorithm that also takes into account presence of obstacles is a good choice. The appropriate solution could be a dynamic window algorithm developed by Fox et al. [50], which is planned in future implementation. It is also desirable that this reactive algorithm is implemented in robot's hardware to obtain fastest possible reaction.

6.13 Smoothing at Goal Side

Once a path approaches to the goal, it is necessary to depart from the roadmap in order to reach the goal. The difference between smoothing at start and goal side is that at the goal we always want the path to have zero curvature, so that the goal is always a single point, or a line. There are three possibilities:

1. Only a desired goal position is specified. This is the simplest case where a path should only reach the goal point, no matter of tangent orientation at the goal.

2. Both desired goal position and orientation are specified. A path can then land somewhere at the tangent line that leads to the goal position and orientation. If enough free space is not available in front of the goal, the problem is still feasible because the robot can arrive to the goal, and then rotate in the place in order to attain desired orientation. Thus, the goal could be either a line or a point. The preferred case is the line, because then the goal configuration can be reached in shorter time.

   This case could be required e.g. if the robot should arrive at a passage between two rooms. Then it is desirable that the robot is properly oriented at the goal in order to quickly continue its motion.

3. Desired goal position, orientation and velocity are specified. In this case path should always land on the tangent line at the place before the goal position so that the goal is always a line. This means that the problem may not have a feasible solution if not enough free space is available before the goal.

   This case could be required for applications such as robot soccer, where the ball should be kicked with specified velocity and orientation.

In all three cases the problem of smoothing is actually problem of connecting two lines, or a circle and a line with G^2 continuous curve. This is straightforward to solve using so far described algorithms. If the goal is a line, then it is necessary to pull tangent from the goal point and use collision check algorithms to find out how long this tangent can be. In the case that a velocity at the goal is specified, it is also necessary to check the feasibility of the path, because it could be impossible to reach specified velocity if the path makes a sharp turn before the goal.

For some applications there could also arise the need for non-zero curvature at the goal. This imposes a problem of connecting two circles with G^2 continuous curve. Although the problem was not studied in this work, algorithms can be extended for this case as well.

6.14 Putting It All Together

The highest level algorithm of the path-smoothing module is called SmoothPath and its pseudocode is given in Algorithm 6.14.1. Algorithm inputs are current and goal state of the robot as well as the shortest path obtained from the path planning module, which consists of straight line segments. Outputs are success flag and final smoothed path. In the pseudocode some input arguments of the functions are omitted for the sake of brevity.

Algorithm 6.14.1: SmoothPath

```
Input: $CurrentState, GoalState, ShortestPath$
Output: $Success, SmoothedPath$

if DirectConnectionPossible()
  then $(Success, SmoothedPath)$ = DirectConnect();
  else
    $SmoothedShortestPath$ = SmoothPiecewiseLinearPath();
    $(Success1, ConnectSegment)$ = Connect();
    $(Success2, DepartSegment)$ = Depart();
    if $Success1$ and $Success2$
      then $SmoothedPath$ =
          $ConnectSegment + SmoothedShortestPath + DepartSegment$;
if not $Success$
  then EmergencyStop();
```

The algorithm first tests whether it is possible to directly connect start and goal configurations, i.e. if there are no obstacles between. If this is the case, there is no need for the roadmap, so that algorithm DirectConnect tries to directly connect start and goal configurations using either lines, circular arcs or clothoids.

If there is no direct connection from start to goal, the shortest path is first smoothed in algorithm SmoothPiecewiseLinearPath, which internally calls SmoothSharpTurn algorithm. This algorithm effectively smoothes majority of the path (except start and end parts). Note that this smoothing process is very lightweight because it can be conducted without explicitly considering obstacles as safe distance to the obstacles is ensured by the path-planning module. The algorithm SmoothPiecewiseLinearPath results with a smoothed shortest path using clothoids and lines.

Remember that roadmap methods require procedures to connect to and depart from the roadmap, and here we additionally require that connecting curves are G^2 continuous, which is done in Connect and Depart algorithms. If both operations succeeded, connecting path segments, smoothed shortest path and departing path segments are concatenated resulting with final smoothed path. In case of failure in either of algorithms, algorithm EmergencyStop is called.

The pseudocode of the algorithm Connect is given in Algorithm 6.14.2. Its main purpose is to find G^2 continuous connection from current configuration to the roadmap. It first checks if the curvature of robot's current motion is zero.[4] If this is the case, the algorithm ConnectLineLine is called. It internally calls algorithm SmoothSharpTurn, and also performs some collision checks and locally modifies the shortest path.

Algorithm 6.14.2: Connect

```
Input: $CurrentState, SmoothedShortestPath$
Output: $Success, ConnectSegment$

if $CurrentState.\kappa == 0$
  then $(Success, ConnectSegment) = $ ConnectLineLine();
  else
    $(Success, ConnectSegment) = $ ConnectCircleLine2C();
    if not $Success$
      then $(Success, ConnectSegment) = $ ConnectCircleLine1C();
    if not $Success$
      then $(Success, ConnectSegment) = $ ConnectCircleLine3C();
```

Algorithm 6.14.3: Depart

```
Input: $GoalState, SmoothedShortestPath$
Output: $Success, DepartSegment$

$(Success, DepartSegment) = $ ConnectLineLine();
```

The situation is more complicated if robot is moving along the path with non-zero curvature—this typically happens in replanning scenarios. Here algorithms that account for non-zero initial curvature have to be used. Start configuration has to be connected to the beginning segment of the shortest path, which is a line, therefore we are interested in algorithms that connect circle and

[4]if the robot is not moving, the curvature is also considered to be zero.

line, namely ConnectCircleLine2C, ConnectCircleLine1C, ConnectCircleLine3C.

If any algorithm fails to find an admissible path, the next algorithm is tried. In the current implementation algorithm ConnectCircleLine2C is tried first, as it gives the path that is most consistent with path obtained by SmoothSharpTurn so that some kind of repeatability is achieved and "path chattering" is avoided in case of replanning.

In connection process, the shortest path from path-planning module may be locally modified by some algorithms, so that connecting path segment has to be explicitly checked for collision. Feasibility check is also performed, which is done by calling functions from the trajectory planning module. For simplicity, these calls are not shown in the pseudocode.

Finally, algorithm Depart (Algorithm 6.14.3) is called that departs from the roadmap in order to connect it to the goal configuration. As it is assumed that the curvature at the goal is always zero, algorithm ConnectLineLine suffices for this purpose.

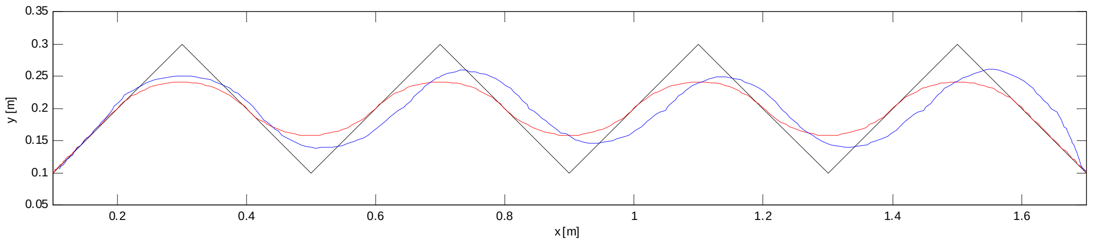
6.15 Experimental Results

To verify correctness of the clothoid steering model, some experiments on the real mobile robot were performed. For this the soccer robot (described in more details in Subsection 7.3) was used. To illustrate necessity of introducing G^2 continuous paths, the results were also compared to the case where only circular arcs are used for smoothing, which is only G^1 continuous. Circular arcs were used in former implementation of the algorithms, as described in [25].

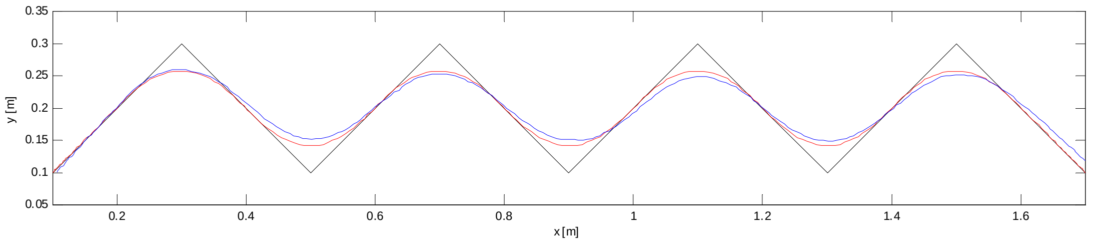
To best illustrate superiority of clothoids over circular arcs, a map with zig-zag shaped walls was used. This produced a path that is very demanded for the robot as it requires fast changes of steering direction. The trajectories were planned so that the robot tracks smoothed paths at a constant velocity of 0.6 [m/s], except at begin and end segments, where uniform acceleration and deceleration were applied. For trajectory tracking a nonlinear trajectory-tracking controller described in Section 8.3.2 was used.

The results obtained for path smoothing using circular arcs are shown in Figure 6.15 (a). It is visible that the robot has a hard time tracking the trajectory, as the tracking error is rather high. On the contrary, tracking of path smoothed by clothoids resulted with significantly lower tracking error (Figure 6.15 (b)). Particularly, the sum-squared-error (SSE) was 4 times lower.

Furthermore, with clothoid-smoothed path maximum trajectory-tracking velocity that could be achieved was up to 25 % higher compared to the case with circular arcs. Here the maximum trajectory-tracking velocity denotes the velocity at which the robot is still able to track the trajectory, i.e. the tracking error remains stable.



(a)



(b)

Figure 6.15. *Results of path smoothing and tracking experiment using different techniques of path smoothing. The paths are tracked by the real robot with velocity* 0.6 m/s. *Legend: black – original non-smoothed path; red – smoothed path; blue – robot actual path.* (a) *Path smoothed with circular arcs. It is visible that tracking error is high.* (b) *Path smoothed with clothoids results with much lower tracking error.*

To illustrate effectiveness of the proposed path smoothing algorithm in combination with the path planner described in Chapter 5, the planned path in Figure 5.10 is smoothed using clothoids. The resulting smoothed path is displayed in Figure 6.16. One can see that the smoothed path never touches edges of the corridor around the planned path, which confirms that the smoothed path is safe.

6.16 Summary

In this chapter a path-smoothing algorithm capable of smoothing piecewise linear paths is described. The produced path is G^2 continuous with linearly changing curvature which is achieved by using clothoid curves as smoothing primitives. Clothoids were used due to their very attractive properties, where the major one is linear relation between the curvature and the arc length. Moreover, efficient algorithms capable of smoothing a path with non-zero initial curvature are developed, which is essential when path replanning for moving robot is required. To the best author's knowledge, this is the first complete solution that enables application of clothoids for path replanning. This feature is useful for motion

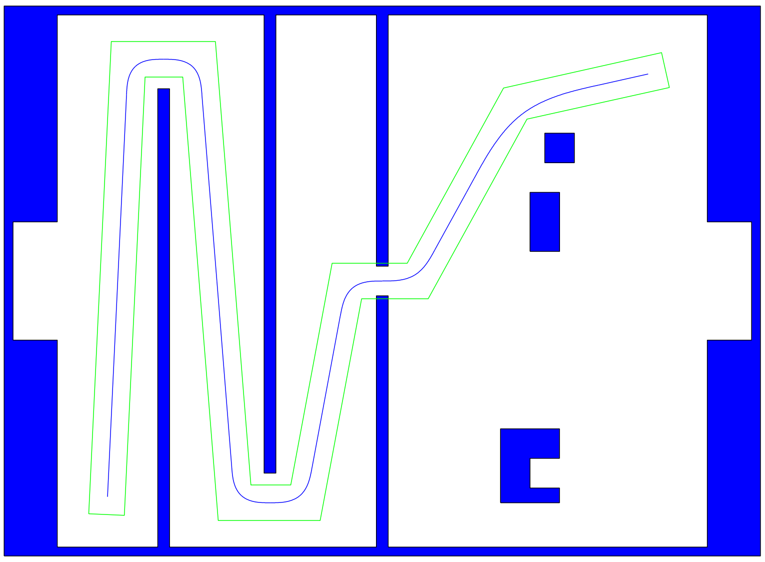


Figure 6.16. *The path from Figure 5.10 smoothed by using clothoids. A corridor around the path is also displayed.*

planning in changing environments.

In majority of the literature it is reported that clothoid-based path smoothing is computationally expensive operation and therefore hard to use in real time. On the contrary, in this work very fast smoothing using clothoid segments is achieved that can be used in real time. This is obtained by storing points of the clothoid in the lookup table. It is shown that points of any clothoid can be efficiently computed based on the stored clothoid by rescaling, rotating and translating. In this way the path smoothing is typically executed in time bellow 1 ms.

So far, we have obtained a smooth path, which may be used directly by applying the path-following controller [111], provided that we are not interested in questions such as how much time a robot will need to traverse the planned path or where a robot will be at some time instance. This is similar to driving on a road while trying to maintain the distance between the vehicle chassis and the side of the road constant, while the velocity is kept constant or adapted according to road conditions or other factors. However, our focus is in more deterministic planning, so that the next topic refers to trajectory planning along the planned path.

Chapter 7

Trajectory Planning

This chapter is concerned with the problem of finding an optimal velocity profile along the planned path in order to traverse the path in shortest time. The planned path and the velocity profile together build a trajectory. A dynamic model of the differential drive robot is developed that accounts for robot actuator constraints, as well as extrinsic constraints originating from the limited grip between the robot's wheels and the ground. Both types of constraints are very important for planning at high velocities. It is shown how to use the developed dynamic model to express acceleration limits and velocity limit curve required by the optimal time-scaling algorithm. The developed trajectory planning algorithm is demonstrated on two differential drive mobile robots: soccer robot and Pioneer 3DX robot.

7.1 Introduction

Trajectory planning can be considered as a complete motion-planning problem, as opposed to a path-planning problem described in Chapter 5, which only asks for a feasible curve $q(s)$ in the configuration space without reference to the speed of execution. As described in Chapter 4, trajectory planning problem can be solved using two approaches: (1) the direct approach, where the search is performed directly in the system's state space and (2) the decoupled approach, where first a path in the configuration space is found and then a time-optimal velocity profile subject to the actuator constraints is computed. In this chapter the decoupled approach is used, which can also be referred as trajectory planning problem constrained to precomputed path.

We assume that a path $q(s)$ is a twice-differentiable curve in the configuration space $q(s) : [0, s_g] \rightarrow \mathcal{C}$, as defined by equation (5.2). To specify a trajectory, we first introduce a *time scaling* function $s(t)$ as $s(t) : [0, t_g] \rightarrow [0, s_g]$, which assigns a value s to each time $t \in [0, t_g]$. The time scaling $s(t)$ is assumed to be twice-differentiable and monotonic, i.e. $\dot{s}(t) > 0,\ t \in (0, t_g)$, where $\dot{s}(t)$ denotes time derivative of $s(t)$. The twice-differentiability of $s(t)$ ensures that the acceleration is well defined and bounded. Using both path and time scaling, a trajectory can now be defined as a function

$$q(s(t)) : [0, t_g] \to \mathcal{C}, \tag{7.1}$$

which is short-written as $q(t)$.

We assume that a mobile robot is operating in a planar workspace, so that its configuration $q(t)$ is given by position coordinates $(x(t), y(t))$ and orientation $\theta(t)$, i.e. $q(t) = [x(t)\ y(t)\ \theta(t)]^T$. If both time derivatives $\dot{x}(t)$ and $\dot{y}(t)$ are non-zero, the orientation is not an independent variable and it can be calculated from position derivatives as

$$\theta(t) = \operatorname{atan2}(\dot{y}(t), \dot{x}(t)) + k\pi, \tag{7.2}$$

where $k = 0,\ 1$ defines the desired drive direction (0 for forward and 1 for backward) and the function atan2 is a four-quadrant inverse tangent function. The longitudinal and angular velocities are then obtained as

$$v(t) = \pm\sqrt{\dot{x}^2(t) + \dot{y}^2(t)}, \tag{7.3a}$$

$$\omega(t) = \frac{\dot{x}(t)\ddot{y}(t) - \dot{y}(t)\ddot{x}(t)}{\dot{x}^2(t) + \dot{y}^2(t)}. \tag{7.3b}$$

The chosen sign for $v(t)$ determines forward or backward motion of the robot.

However, if for some time t the longitudinal velocity $v(t)$ is zero, neither the orientation $\theta(t)$ nor the angular velocity $\omega(t)$ are defined by equations (7.2) and (7.3b), respectively, and must be given explicitly.

Until now, the path parameter s was not connected with any real-world variable. From now on, it will denote the distance travelled along the path in the case that robot's motion has translation component, i.e. its longitudinal velocity $v(t)$ is non-zero. Otherwise, in case of pure rotation, i.e. $v(t) = 0$ and $\omega(t) \neq 0$, it will be used to denote the travelled angle. Then in case $v(t) \neq 0$, using (7.3) and (6.1), the longitudinal and angular velocities are

$$v(t) = \dot{s}(t), \quad \omega(t) = \kappa(s(t))\dot{s}(t), \tag{7.4}$$

and accelerations are

$$\dot{v}(t) = \ddot{s}(t), \quad \dot{\omega}(t) = \frac{d\kappa(s)}{ds}\dot{s}(t)^2 + \kappa(s(t))\ddot{s}(t). \tag{7.5}$$

In case of pure rotation, the velocities are

$$v(t) = 0, \quad \omega(t) = \dot{s}(t), \tag{7.6}$$

and accelerations

$$\dot{v}(t) = 0, \quad \dot{\omega}(t) = \ddot{s}(t). \tag{7.7}$$

7.2 Decoupled Trajectory Planning

In this work the motion planning problem is solved using the decoupled approach. Therefore, the trajectory planning module takes a smoothed path $q(s)$ as an input, and tries to find the fastest feasible trajectory that follows this path. Here the term "fastest" denotes the time-optimal scaling $s(t)$, whereas the term "feasible" refers to the actuator limits. Besides in mobile robot motion planning, this problem is especially important for maximizing the productivity of static manipulators when a path has been given as a task specification or found by a path planner.

Let a general dynamic model of the robot be given by [38]

$$u = M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q), \tag{7.8}$$

where u is the vector of *generalized forces* acting on the generalized coordinates q. $C(q, \dot{q})\dot{q} \in \mathbb{R}^{n_{\mathcal{C}}}$ is a vector of velocity product terms with the Coriolis matrix $C(q, \dot{q})$ of dimension $n_{\mathcal{C}} \times n_{\mathcal{C}}$, linear in $\dot{q}$, $g(q) \in \mathbb{R}^{n_{\mathcal{C}}}$ is a vector of gravitational forces, and $M(q)$ is an $n_{\mathcal{C}} \times n_{\mathcal{C}}$ symmetric, positive definite mass or inertia matrix.

This model can also be expressed as

$$u = M(q)\ddot{q} + \dot{q}^T \Gamma(q)\dot{q} + g(q), \tag{7.9}$$

where $\Gamma(q) \in \mathbb{R}^{n_{\mathcal{C}} \times n_{\mathcal{C}} \times n_{\mathcal{C}}}$ can be viewed as an $n_{\mathcal{C}}$-dimensional column vector, where each element is a matrix whose elements are Christoffel symbols of the inertia matrix $M(q)$.

The developed forces are subject to the actuator limits

$$u_i^{\min}(q, \dot{q}) \leq u_i \leq u_i^{\max}(q, \dot{q}). \tag{7.10}$$

In the most general form the actuator limits are expressed as functions of the robot configuration and velocity. An example may be the torque available to accelerate a DC motor, which decreases as its angular velocity increases. The simplest form of actuator constraints is the limit independent of the robot configuration and velocity $|u_i| \leq u_i^{\max}$.

If path is given by $q(s)$, velocity and acceleration can be expressed as

$$\dot{q} = \frac{dq}{ds}\dot{s} \tag{7.11}$$

$$\ddot{q} = \frac{d^2q}{ds^2}\dot{s}^2 + \frac{dq}{ds}\ddot{s}. \tag{7.12}$$

By substituting this into dynamic model equation (7.9), we get

$$M(q(s))\left(\frac{d^2q}{ds^2}\dot{s}^2+\frac{dq}{ds}\ddot{s}\right)+\left(\frac{dq}{ds}\dot{s}\right)^T\Gamma(q(s))\left(\frac{dq}{ds}\dot{s}\right)+g(q(s))=u, \tag{7.13}$$

i.e.

$$\left(M(q(s))\frac{dq}{ds}\right)\ddot{s}+\left(M(q(s))\frac{d^2q}{ds^2}+\left(\frac{dq}{ds}\right)^T\Gamma(q(s))\frac{dq}{ds}\right)\dot{s}^2+g(q(s))=u. \tag{7.14}$$

This can be rewritten more compactly as the vector equation

$$a(s)\ddot{s}+b(s)\dot{s}^2+c(s)=u, \tag{7.15}$$

which defines robot dynamic model constrained to the path $q(s)$. The vector functions $a(s)$, $b(s)$, and $c(s)$ are inertial, velocity product, and gravitational terms in terms of s, respectively.

Because the robot motion is constrained to the path $q(s)$, its state at any time is determined by $(s,\dot{s})$. Now the actuator limits can be expressed as a function of $(s,\dot{s})$ by substituting equation (7.11) into equation (7.10), yielding lower limit $u^{\min}(s,\dot{s})$ and upper limit $u^{\max}(s,\dot{s})$. From equation (7.15), we conclude that the system must satisfy the constraints

$$u^{\min}(s,\dot{s})\le a(s)\ddot{s}+b(s)\dot{s}^2+c(s)\le u^{\max}(s,\dot{s}). \tag{7.16}$$

This equation enables us to express the minimum and maximum accelerations $\ddot{s}$ as functions of the current state $(s,\dot{s})$, which are required to obtain time-optimal scaling function. Let the minimum and maximum accelerations $\ddot{s}$ satisfying the i-th component of equation (7.16) be denoted by $L_i(s,\dot{s})$ and $U_i(s,\dot{s})$, respectively. We define

$$\alpha_i(s,\dot{s})=\frac{u_i^{\min}(s,\dot{s})-b_i(s)\dot{s}^2-c_i(s)}{a_i(s)},\quad \beta_i(s,\dot{s})=\frac{u_i^{\max}(s,\dot{s})-b_i(s)\dot{s}^2-c_i(s)}{a_i(s)}. \tag{7.17}$$

Now if $a_i(s)>0$, then

$$L_i(s,\dot{s})=\alpha_i(s,\dot{s}),\quad U_i(s,\dot{s})=\beta_i(s,\dot{s}).$$

If $a_i(s)<0$, then

$$L_i(s,\dot{s})=\beta_i(s,\dot{s}),\quad U_i(s,\dot{s})=\alpha_i(s,\dot{s}).$$

If $a_i(s)=0$, the system is at a zero inertia point (see e.g. [38] for details). If we

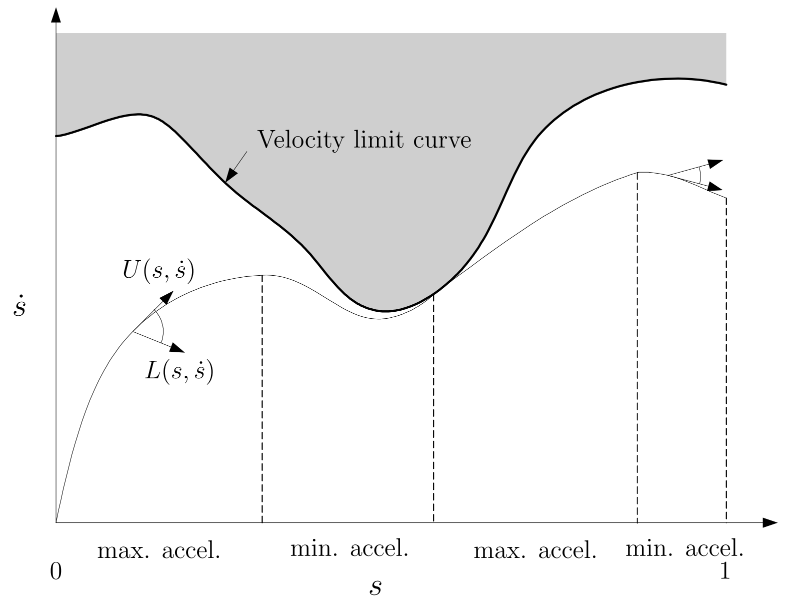


Figure 7.1. *At each state $(s,\dot{s})$ we can draw a cone defined by the minimum and maximum accelerations $\ddot{s}$ satisfying the actuator limits. The time-optimal trajectory is the curve that maximizes the area underneath it while remaining on the boundary of the motion cones. The velocity limit curve indicates the states where the cone collapses to a single tangent vector. Above the velocity limit curve are inadmissible states where no feasible actuation will keep the robot on the path.*

additionally define

$$L(s,\dot{s}) = \max_{i \in 1 \ldots n_{\mathcal{C}}} L_i(s,\dot{s}), \quad U(s,\dot{s}) = \min_{i \in 1 \ldots n_{\mathcal{C}}} U_i(s,\dot{s}), \tag{7.18}$$

the actuator limits (7.16) can now simply be expressed as

$$L(s,\dot{s}) \leq \ddot{s} \leq U(s,\dot{s}). \tag{7.19}$$

The problem of finding the time-optimal trajectory constrained to a path can now be stated [38]:

> Given a path $q(s) : [0, s_g] \rightarrow \mathcal{C}$, an initial state $(0, \dot{s}_0)$, and a final state $(s_g, \dot{s}_g)$, find a monotonically increasing twice-differentiable time scaling $s(t) : [0, t_g] \rightarrow [0, s_g]$ that (i) satisfies $s(0) = 0,\ \dot{s}(0) = \dot{s}_0,\ s(t_g) = s_g,\ \dot{s}(t_g) = \dot{s}_g$, and (ii) minimizes the total travel time t_g along the path while respecting the actuator constraints (7.19) for all time $t \in [0, t_g]$.

The problem is best visualized in the $(s,\dot{s})$ state space. The feasible acceleration constraint (7.19) can be illustrated as a cone of tangent vectors defined at any state $(s,\dot{s})$, as shown in Figure 7.1. Then the lower edge of the cone corresponds to the minimum acceleration $L(s, \dot{s})$, while upper edge corresponds to the maximum acceleration $U(s, \dot{s})$. The interior of the cone corresponds to a range of feasible accelerations $\ddot{s}$ at the state $(s, \dot{s})$ along the path so that $L(s, \dot{s}) \leq \ddot{s} \leq U(s, \dot{s})$.

Any of these tangent vectors denotes that if the corresponding acceleration is selected in the next time interval, then the system state transition will occur in the direction of this tangent. The problem is now to find a curve from $(0, \dot{s}_0)$ to $(s_g, \dot{s}_g)$ such that $\dot{s} \geq 0$ and the curve tangent is inside the cone at each state. Further, to minimize the travel time, the velocity $\dot{s}$ along the path should be maximized. This can be obtained by maximizing the area beneath the curve. A consequence of this is that the curve always follows the upper or lower bound of the cone at each state, i.e. the system always operates at minimum or maximum acceleration. In other words, at least one of the actuators is always saturated. The problem is now to find an alternating sequence, i.e. the switching points between maximum and minimum accelerations.

Because of the switching between extreme opposites this kind of trajectory is often called a “bang-bang” trajectory. The method was introduced by Dobrow et al. [16], and Shin and McKay [136]. One drawback of planning at maximum accelerations is that it does not leave any maneuver space to recover from eventual trajectory tracking error (the fourth module of the decoupled approach). Such an error could be caused by many factors, e.g. by the non-flat floor, which can be characterized as an unpredictable external disturbance. The solution could be to leave some acceleration reserves and work with lower accelerations than the actual maximums.

Another drawback of planning at maximum acceleration is high energy consumption. However, in case that low energy consumption is more important than to arrive at the goal in the shortest time, another time-scaling algorithms can be considered that result with lover energy consumption.

Actuation constraints (7.19) impose that there could be some states at which there is no feasible acceleration that is required for the system to continue to follow the path. This region is depicted in gray in Figure 7.1 and is called inadmissible region. If the robot state is in this region, it will leave the path immediately. But even if robot state is in admissible region, robot may still be doomed to leave the path, no matter of the selected command acceleration. This occurs if the range of admissible accelerations is directed to the inadmissible region, as illustrated in Figure 7.2. In this example the robot’s state, no matter if the robot is actuated with the minimum or maximum admissible acceleration, always ends up in the inadmissible region.

Here it is assumed that, for any s, the robot is strong enough to maintain its configuration statically, so all $\dot{s} = 0$ states are admissible and the path can be executed arbitrarily slowly. It is also assumed that as $\dot{s}$ increases from zero for a given s, there will be at most one switch from admissible to inadmissible, which

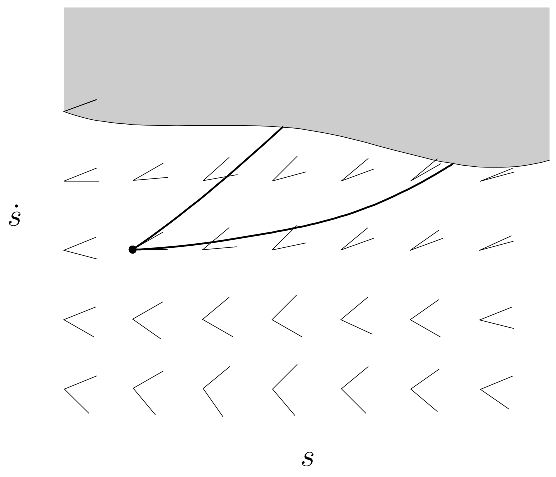


Figure 7.2. *Initial robot state (represented by the dot) is in admissible region, but the robot is doomed to the inadmissible region, because its motion is constrained by the admissible trajectories tangents represented by the cones, which are locally directed so that robot always penetrates the velocity limit curve. The set of all feasible trajectories is between the two integral curves shown, where the upper curve denotes the maximum acceleration, and the lower curve denotes the minimum acceleration.*

occurs at the *velocity limit curve* $V(s)$, consisting of states $(s, \dot{s})$ satisfying

$$L(s, \dot{s}) = U(s, \dot{s}). \tag{7.20}$$

Because of $\min(\cdot)$ and $\max(\cdot)$ function used in calculus of $L(s, \dot{s})$ and $U(s, \dot{s})$, these functions, and velocity limit curve are generally not smooth. However, in some special cases the equation (7.20) may have multiple solutions $\dot{s}$ for a single value of s. This may be due to friction in the system, weak actuators that cannot hold each configuration statically, or the form of the actuation limit functions. In this case, there may be inadmissible "islands" in the phase plane. This significantly complicates the problem of finding an optimal time scaling, and time-scaling algorithm for this case is given in [136].

The algorithm that gives optimal time scaling, i.e. the optimal sequence of s values where the switching between maximum and minimum acceleration should occur, is given by the following algorithm [38]:

Time-Scaling Algorithm

1. Initialize an empty list of switches $\mathcal{S} = \{\}$ and a switch counter $i = 0$. Set $(s_i, \dot{s}_i) = (0, \dot{s}_0)$.

2. Integrate the equation $\ddot{s} = L(s, \dot{s})$ backward in time from $(s_g, \dot{s}_g)$ until the velocity limit curve is penetrated (reached transversally, not tangentially), $s = 0$, or $\dot{s} = 0$ at $s < s_g$. There is no solution to the problem if $\dot{s} = 0$ is

reached. Otherwise, call this phase plane curve F and proceed to the next step.

3. Integrate the equation $\ddot{s} = U(s, \dot{s})$ forward in time from $(s_i, \dot{s}_i)$. Call this curve A_i. Continue integrating until either A_i crosses F or A_i penetrates the velocity limit curve. (If A_i crosses $s = s_g$ or $\dot{s} = 0$ before either of these two cases occurs, there is no solution to the problem.) If A_i crosses F, then increment i, let s_i be the s value at which the crossing occurs, and append s_i to the list of switches $\mathcal{S}$. The problem is solved and $\mathcal{S}$ is the solution. If instead the velocity limit curve is penetrated, let $(s_{lim}, \dot{s}_{lim})$ be the point of penetration and proceed to the next step.

4. Search the velocity limit curve $V(s)$ forward in s from $(s_{lim}, \dot{s}_{lim})$ until finding the first point where the feasible acceleration ($L = U$ on the velocity limit curve) is tangent to the velocity limit curve. If the velocity limit is $V(s)$, then a point $(s_0, v(s_0))$ satisfies the tangency condition if $\frac{dv}{ds}|_{s=s_0} = U(s_0, v(s_0))/v(s_0)$. Call the first tangent point reached $(s_{tan}, \dot{s}_{tan})$. From $(s_{tan}, \dot{s}_{tan})$, integrate the curve $\ddot{s} = L(s, \dot{s})$ backward in time until it intersects A_i. Increment i and call this new curve A_i. Let s_i be the s value of the intersection point. This is a switch point from maximum to minimum acceleration. Append s_i to the list $\mathcal{S}$.

5. Increment i and set $(s_i, \dot{s}_i) = (s_{tan}, \dot{s}_{tan})$. This is a switch point from minimum to maximum acceleration. Append s_i to the list $\mathcal{S}$. Go to step 3.

An illustration of the time-scaling algorithm is shown in Figure 7.3. Although the described optimal time-scaling algorithm is general, to use it we have to know actuator limits (7.19), which are specific for each particular robot type. Therefore in the sections that follow examples of modeling and trajectory planning for two types of robots possessed by our lab will be given: soccer robot and Pioneer 3DX robot.

7.3 Trajectory Planning for Soccer Robot

For lightweight robots with strong wheel-driving motors, longitudinal and angular accelerations can be controlled independently if the motors are powerful enough to attain both maximum longitudinal and angular acceleration simultaneously. An example of such robot is a soccer robot shown in Figure 7.4. This is a small robot with the differential drive that has two driving wheels and two castor (passive) wheels.

Motors of the soccer robot can provide forces that are significantly higher than available contact forces between driving wheels and the ground, so that accelerations are not constrained by actuator limits, but rather by available friction

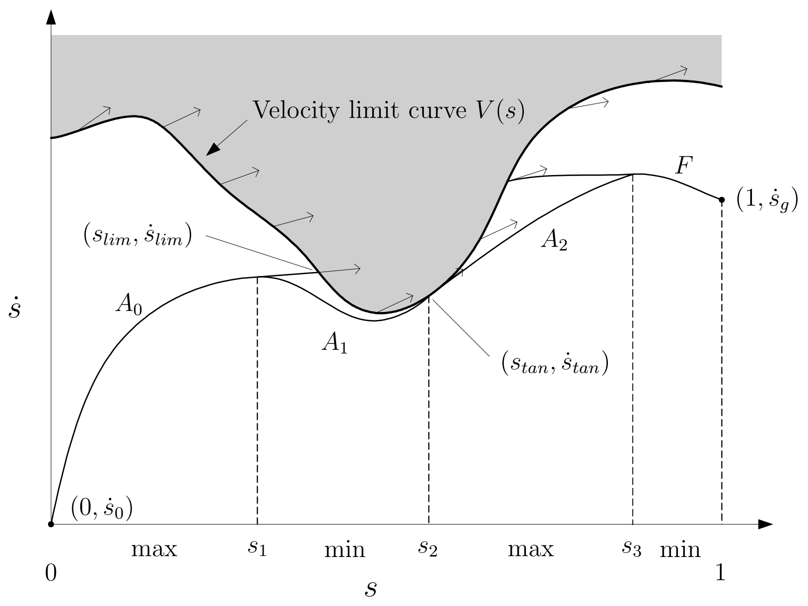


Figure 7.3. *An illustration of the optimal time-scaling algorithm. The feasible accelerations at points on the velocity limit curve are shown as arrows. The output of the algorithm is the switch list* $\mathcal{S} = \{s_1, s_2, s_3\}$.

force. Thus for the soccer robot, apart from intrinsic constraints imposed by motor torque limits, there are also extrinsic constraints imposed by available grip force of the wheels.

7.3.1 Intrinsic Constraints

The command inputs to the soccer robot are referent longitudinal and angular velocities that are sent to the robot via wireless communication. However, dynamic model given by equation (7.9) requires use of generalized forces as the command input. Therefore longitudinal acceleration a and angular acceleration α (which are proportional to the corresponding forces) will be used as components of generalized forces vector u, i.e. $u = [a\ \alpha]^T$. Final output of the trajectory planning algorithm will be the velocity profile as the function of time, from which the necessary acceleration commands are easily generated by differentiating the longitudinal and angular velocities.

Until now, the vector $[x\ y\ \theta]^T$ was used to describe robot state. However, with this choice it is not possible to directly use described time-scaling algorithm since we have three state components while there are only two components of the command input vector—this is a consequence of the nonholonomic constraint. Nevertheless, the velocity pair $[v\ \omega]^T$ can be used to describe robot state, so that

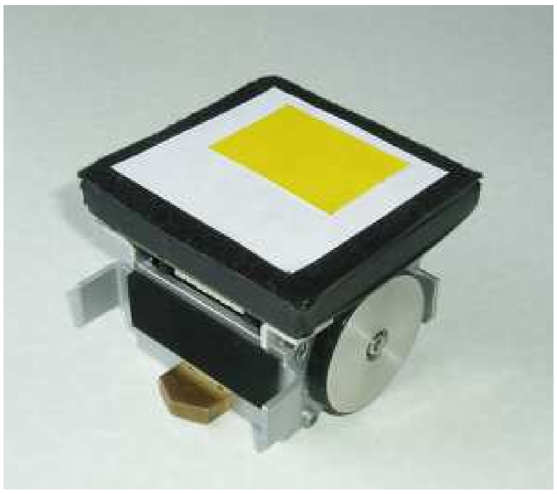


Figure 7.4. *Soccer robot.*

dynamic model of the robot can be written as

$$\begin{bmatrix} \dot{v} \\ \dot{\omega} \end{bmatrix} = u. \tag{7.21}$$

Although the state vector $[v\ \omega]^T$ does not answer us about the robot configuration directly, this information can be obtained implicitly through the time history of the velocity vector. In this way the configuration can be computed if an initial configuration $[x_0\ y_0\ \theta_0]^T$, time history of the velocity vector and kinematic model of the robot are known. Hereby, kinematic model of the differential drive robot is given by

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}. \tag{7.22}$$

Additionally, the nonholonomic constraint that comes from the fact that the driving wheels can only roll, but not slip, must be fulfilled:

$$-\dot{x}\sin\theta + \dot{y}\cos\theta = 0. \tag{7.23}$$

For $v \neq 0$ we can substitute (7.4) and (7.5) into dynamic model (7.21), obtaining

$$\begin{bmatrix} 1 \\ \kappa(s) \end{bmatrix} \ddot{s} + \begin{bmatrix} 0 \\ \frac{d\kappa(s)}{ds} \end{bmatrix} \dot{s}^2 = \begin{bmatrix} a \\ \alpha \end{bmatrix}. \tag{7.24}$$

From this model, and using analogy with (7.15) and (7.17), we have the following longitudinal acceleration limits imposed by the robot actuators

$$L_1(s,\dot{s}) = a_{min}(s,\dot{s}), \quad U_1(s,\dot{s}) = a_{max}(s,\dot{s}), \tag{7.25}$$

where the minimum longitudinal acceleration is nonlinear function given by

$$a_{min}(s,\dot{s}) = \begin{cases} \bar{a}_{min}, & \dot{s} \in (-\bar{v}_{max}, \bar{v}_{max}) \\ 0, & \text{otherwise} \end{cases}, \tag{7.26}$$

where $\bar{a}_{min}$ is a negative constant that denotes the minimum longitudinal acceleration, while $\bar{v}_{max}$ is the maximum longitudinal velocity of the robot. The maximum longitudinal acceleration is given similarly by

$$a_{max}(s,\dot{s}) = \begin{cases} \bar{a}_{max}, & \dot{s} \in (-\bar{v}_{max}, \bar{v}_{max}) \\ 0, & \text{otherwise} \end{cases}, \tag{7.27}$$

where $\bar{a}_{max}$ is a positive constant denoting the maximum longitudinal acceleration. It is usually $\bar{a}_{max} = |\bar{a}_{min}|$, although some electronic motor drivers achieve $|\bar{a}_{min}| > \bar{a}_{max}$, i.e. they enable faster braking than accelerating. It is important to mention that acceleration limits in the case of the soccer robot do not stem from the motor torque limits, but from software configuration of the robot control circuitry.

From robot dynamic model (7.24), the acceleration constraints due to angular acceleration limits are

$$\begin{aligned} L_2(s,\dot{s}) &= \begin{cases} \dfrac{\alpha_{min}(s,\dot{s}) - \frac{d\kappa(s)}{ds}\dot{s}^2}{\kappa(s)}, & \kappa(s) > 0 \\ \text{undefined}, & \kappa(s) = 0 \\ \dfrac{\alpha_{max}(s,\dot{s}) - \frac{d\kappa(s)}{ds}\dot{s}^2}{\kappa(s)}, & \kappa(s) < 0 \end{cases} \\ U_2(s,\dot{s}) &= \begin{cases} \dfrac{\alpha_{max}(s,\dot{s}) - \frac{d\kappa(s)}{ds}\dot{s}^2}{\kappa(s)}, & \kappa(s) > 0 \\ \text{undefined}, & \kappa(s) = 0 \\ \dfrac{\alpha_{min}(s,\dot{s}) - \frac{d\kappa(s)}{ds}\dot{s}^2}{\kappa(s)}, & \kappa(s) < 0 \end{cases}, \end{aligned} \tag{7.28}$$

where the minimum angular acceleration is nonlinear function given by

$$\alpha_{min}(s,\dot{s}) = \begin{cases} \bar{\alpha}_{min}, & \kappa(s) = 0 \vee \dot{s} \in [-\bar{\omega}_{max}/|\kappa(s)|, \bar{\omega}_{max}/|\kappa(s)|] \\ 0, & \text{otherwise} \end{cases}, \tag{7.29}$$

where $\bar{\alpha}_{min}$ is a negative constant denoting minimum angular acceleration, while $\bar{\omega}_{max}$ is the maximum angular velocity of the robot. The maximum angular acceleration is given similarly by

$$\alpha_{max}(s,\dot{s}) = \begin{cases} \bar{\alpha}_{max}, & \kappa(s) = 0 \vee \dot{s} \in [-\bar{\omega}_{max}/|\kappa(s)|, \bar{\omega}_{max}/|\kappa(s)|] \\ 0, & \text{otherwise} \end{cases}, \tag{7.30}$$

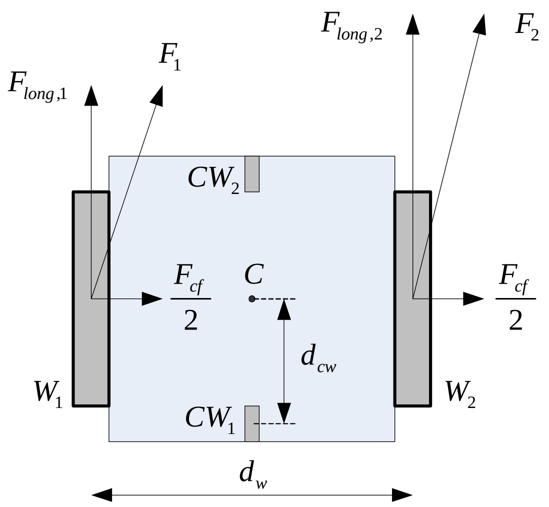


Figure 7.5. *Top view of the robot. The forces developed on robot driving wheels are illustrated.*

where $\bar{\alpha}_{max}$ is the a positive constant denoting maximum angular acceleration.

Experiments have shown that for the soccer robot from Figure 7.4, unlike for longitudinal acceleration, for angular acceleration it is $\bar{\alpha}_{max} < |\bar{\alpha}_{min}|$, i.e. maximum angular acceleration is lower than maximum angular deceleration. This is most likely caused by the bad implementation of the castor wheel, which results in high friction between the castor wheel and the ground in the case of robot rotation. Thus a part of the available torque is used to compensate this friction, which in effect decreases the angular acceleration and increases the angular deceleration.

In case when $v = 0$, we have a pure rotation. Then the dynamic model reduces to a single equation $\ddot{s} = \alpha$, from which we derive acceleration limits as

$$L_{1,rot}(s,\dot{s}) = \alpha_{min,rot}, \quad U_{1,rot}(s,\dot{s}) = \alpha_{max,rot}, \tag{7.31}$$

where

$$\begin{aligned} \alpha_{min,rot}(s,\dot{s}) &= \begin{cases} \bar{\alpha}_{min}, & \dot{s} \in [-\omega_{max}, \omega_{max}] \\ 0, & \text{otherwise} \end{cases} \\ \alpha_{max,rot}(s,\dot{s}) &= \begin{cases} \bar{\alpha}_{max}, & \dot{s} \in [-\omega_{max}, \omega_{max}] \\ 0, & \text{otherwise} \end{cases} . \end{aligned} \tag{7.32}$$

7.3.2 Extrinsic Constraints

Apart from intrinsic constraints originating from robot actuator limits, in case of the soccer robot we have also extrinsic constraints caused by limited grip force between the wheels and the floor. To find corresponding acceleration limits, first the forces developed between robot driving wheels and the ground will be expressed. A force on the each wheel consists of a longitudinal and a lateral component (Figure 7.5). The longitudinal component comes from the torque used to accelerate or decelerate wheels. The lateral component developes if robot translates and rotates at the same time and is caused by inertial centrifugal force. Each wheel takes the half of the total centrifugal force. Then for both wheels it can be written

$$F_i^2 = F_{long,i}^2 + \frac{F_{cf}^2}{4}, \quad i = 1, 2 \tag{7.33}$$

where index $i = 1$ stands for the left wheel, while $i = 2$ is for the right wheel, F_i is the overall force developed on the wheel, $F_{long,i}$ is the longitudinal force component, and F_{cf} is the overall centrifugal force acting on the robot.

The equations that describe robot motion dynamics are

$$m\dot{v} = F_{long,1} + F_{long,2}, \tag{7.34a}$$

$$J\dot{\omega} = (F_{long,2} - F_{long,1})\frac{d_w}{2}, \tag{7.34b}$$

where m is the mass of the robot, J is the inertia moment of the robot and d_w is the distance between robot wheels. The centrifugal forces does not influence robot motion as long as the developed force on the wheels is lower than the available friction force, i.e. we suppose that wheels can only roll. From (7.34) the longitudinal forces are obtained as

$$F_{long,1} = \frac{m}{2}\dot{v} - \frac{J}{d_w}\dot{\omega}, \tag{7.35a}$$

$$F_{long,2} = \frac{m}{2}\dot{v} + \frac{J}{d_w}\dot{\omega}. \tag{7.35b}$$

The centrifugal force is obtained as

$$F_{cf} = ma_{cf} = mv\omega, \tag{7.36}$$

where a_{cf} is the centrifugal acceleration.

We absolutely want to prevent the robot from entering the wheel slipping mode, as this would result in uncontrolled motion of the robot. Therefore we have to ensure that required overall forces F_i on the wheels do not exceed maximum available friction forces between the wheels and the ground. To express friction forces, the Coulomb's friction model model is used. A prerequisite for the Coulomb's model application is to compute weighting forces on the each wheel. Robot's center of the mass is located approximately in the middle point of the driving wheels axle (point C in Figure 7.5), so that in static conditions each wheel carries half the weight of the robot. However, the weight distribution changes as the robot accelerates or rotates since the mass center is at a certain height above the ground.

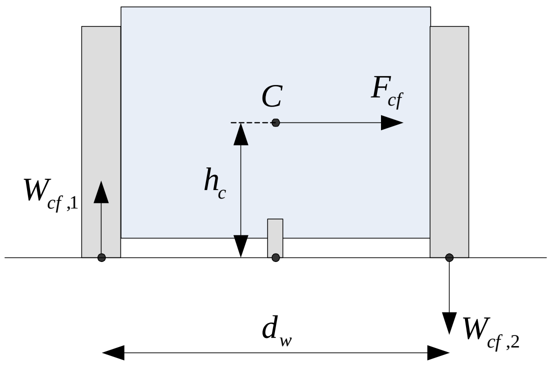


Figure 7.6. *Rear view of the robot. The centrifugal forces acting on robot driving wheels are illustrated.*

If the robot translates and rotates simultaneously, the centrifugal force is developed, which acts in the mass center (Figure 7.6). This has an effect of redistribution of the weighting force on the driving wheels. If we imagine that the robot is driving along a circle, the effective weighting force on the wheel that is closer to the center of the circle will decrease, and increase for the other wheel. From Figure 7.6 and using equation (7.36), the additional weighting forces due to centrifugal force can be expressed as

$$W_{cf,1} = -mv\omega\frac{h_c}{d_w}, \quad W_{cf,2} = mv\omega\frac{h_c}{d_w}, \tag{7.37}$$

where h_c is the height of the mass center relative to the ground. The positive sign of the force means that it increases the static weight on the wheel.

When the robot travels at constant velocity, its whole weight is distributed solely on driving wheels. However, as the robot accelerates or decelerates, a portion of the weight will also be carried by the castor wheel that is on the opposite side of the acceleration direction. This is illustrated in Figure 7.7, from where we derive the effective weighting force on the driving wheels due to longitudinal acceleration as

$$W_{acc,i} = -|F_{in}|\frac{h_c}{2d_{cw}} = -m|\dot{v}|\frac{h_c}{2d_{cw}}, \tag{7.38}$$

where F_{in} is the inertial force that resists to robot acceleration. The force $W_{acc,i}$ is always negative, no matter if the robot accelerates or decelerates—the only difference is that by acceleration the force acts on the rear castor wheel, while by deceleration it acts on the front castor wheel.

Now we can obtain the overall effective weighting forces acting on driving wheels as $W_i = W_{static} + W_{cf,i} + W_{acc,i}$, where $W_{static} = ma_g/2$ is the static weight on the wheel and a_g is gravitation acceleration. By using (7.37) and (7.38) the

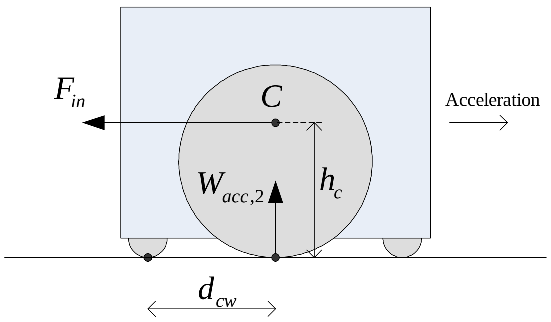


Figure 7.7. *Side view of the robot. The inertial force that resists to robot acceleration decreases the weight on driving wheels and increases the weight on the castor wheel.*

effective weighting forces on wheels are

$$W_1 = m\left(\frac{a_g}{2} - v\omega\frac{h_c}{d_w} - |\dot{v}|\frac{h_c}{2d_{cw}}\right), \tag{7.39a}$$

$$W_2 = m\left(\frac{a_g}{2} + v\omega\frac{h_c}{d_w} - |\dot{v}|\frac{h_c}{2d_{cw}}\right). \tag{7.39b}$$

Naturally, it must be $W_i > 0$, i.e. the weighting forces must remain positive at all times since otherwise the robot would tip over. Substituting (7.4) and (7.5) into (7.39) and considering the worst case, where the weighting force decreases due to centrifugal acceleration, yields the constraint

$$-\frac{h_c}{2d_{cw}}|\ddot{s}| - \frac{h_c}{d_w}|\kappa(s)|\dot{s}^2 + \frac{a_g}{2} > 0, \tag{7.40}$$

from where the following acceleration limits are obtained:

$$L_3(s,\dot{s}) = -\frac{2d_{cw}}{h_c}\left(\frac{a_g}{2} - \frac{h_c}{d_w}|\kappa(s)|\dot{s}^2\right), \tag{7.41a}$$

$$U_3(s,\dot{s}) = \frac{2d_{cw}}{h_c}\left(\frac{a_g}{2} - \frac{h_c}{d_w}|\kappa(s)|\dot{s}^2\right). \tag{7.41b}$$

Using the Coulomb's friction model, the maximum friction force that can be developed between the wheel and the ground is

$$F_{fr,i} = \mu W_i, \tag{7.42}$$

where μ is the friction coefficient. Now the condition that the force developed between the wheel and the ground must not exceed maximum friction force can be written as $F_i \leq F_{fr,i}$. By using (7.42), (7.39), (7.33), (7.35) and (7.36), this condition for $i = 1$ (left wheel) can be written as

$$\left(\frac{m}{2}\dot{v} - \frac{J}{d_w}\dot{\omega}\right)^2 + \frac{(mv\omega)^2}{4} \leqslant \left[\mu m\left(\frac{a_g}{2} - v\omega\frac{h_c}{d_w} - |\dot{v}|\frac{h_c}{2d_{cw}}\right)\right]^2, \tag{7.43}$$

and for $i = 2$ (right wheel)

$$\left(\frac{m}{2}\dot{v} + \frac{J}{d_w}\dot{\omega}\right)^2 + \frac{(mv\omega)^2}{4} \leqslant \left[\mu m\left(\frac{a_g}{2} + v\omega\frac{h_c}{d_w} - |\dot{v}|\frac{h_c}{2d_{cw}}\right)\right]^2. \tag{7.44}$$

To express the limits as a function of parameter s, for $v \neq 0$ (translational motion) we substitute (7.4) and (7.5) into (7.43), which yields the condition for the left wheel as

$$\left[\frac{m}{2}\ddot{s} - \frac{J}{d_w}\left(\frac{d\kappa(s)}{ds}\dot{s}^2 + \kappa(s)\ddot{s}\right)\right]^2 + \frac{\left(m\kappa(s)\dot{s}^2\right)^2}{4} \leqslant \left[\mu m\left(\frac{a_g}{2} - \frac{h_c}{d_w}\kappa(s)\dot{s}^2 - \frac{h_c}{2d_{cw}}|\ddot{s}|\right)\right]^2, \tag{7.45}$$

and for the right wheel as

$$\left[\frac{m}{2}\ddot{s} + \frac{J}{d_w}\left(\frac{d\kappa(s)}{ds}\dot{s}^2 + \kappa(s)\ddot{s}\right)\right]^2 + \frac{\left(m\kappa(s)\dot{s}^2\right)^2}{4} \leqslant \left[\mu m\left(\frac{a_g}{2} + \frac{h_c}{d_w}\kappa(s)\dot{s}^2 - \frac{h_c}{2d_{cw}}|\ddot{s}|\right)\right]^2. \tag{7.46}$$

Condition (7.45) can be written in form of a quadratic inequation as

$$A_{1,2}(s,\dot{s})\ddot{s}^2 + B_{1,2}(s,\dot{s})\ddot{s} + C_{1,2}(s,\dot{s}) \leqslant 0, \tag{7.47}$$

where coefficients A_1, B_1, C_1 are used in case $\ddot{s} \leqslant 0$, and coefficients A_2, B_2, C_2 in case $\ddot{s} \geqslant 0$. These coefficients are obtained as

$$A_{1,2}(s,\dot{s}) = \left(\frac{m}{2} - \frac{J\kappa(s)}{d_w}\right)^2 - \left(\frac{\mu m h_c}{2d_{cw}}\right)^2, \tag{7.48a}$$

$$B_{1,2}(s,\dot{s}) = \frac{-2J\frac{d\kappa(s)}{ds}\left(\frac{m}{2} - \frac{J\kappa(s)}{d_w}\right)}{d_w}\dot{s}^2 \mp \frac{m^2\mu^2 h_c\left(\frac{a_g}{2} - \frac{h_c\kappa(s)\dot{s}^2}{d_w}\right)}{d_{cw}}, \tag{7.48b}$$

$$C_{1,2}(s,\dot{s}) = \left(\frac{J^2}{d_w^2}\left(\frac{d\kappa(s)}{ds}\right)^2 + \frac{1}{4}m^2\kappa(s)^2\right)\dot{s}^4 - m^2\mu^2\left(\frac{a_g}{2} - \frac{h_c\kappa(s)\dot{s}^2}{d_w}\right)^2. \tag{7.48c}$$

From equation (7.47) we get lower and upper acceleration limits that suppress sliding of the left wheel for non-zero longitudinal velocity

$$L_4(s,\dot{s}) = \frac{-B_1(s,\dot{s}) - \sqrt{B_1(s,\dot{s})^2 - 4A_1(s,\dot{s})C_1(s,\dot{s})}}{2A_1(s,\dot{s})}, \tag{7.49a}$$

$$U_4(s,\dot{s}) = \frac{-B_2(s,\dot{s}) + \sqrt{B_2(s,\dot{s})^2 - 4A_2(s,\dot{s})C_2(s,\dot{s})}}{2A_2(s,\dot{s})}. \tag{7.49b}$$

If $\kappa(s) = 0$ and $d\kappa(s)/ds = 0$, (7.49) becomes

$$L_4(s,\dot{s})|_{\kappa=0,\frac{d\kappa}{ds}=0} = -\frac{\mu a_g}{1+\mu\frac{h_c}{d_{cw}}}, \quad U_4(s,\dot{s})|_{\kappa=0,\frac{d\kappa}{ds}=0} = \frac{\mu a_g}{1+\mu\frac{h_c}{d_{cw}}} \tag{7.50}$$

Analogously, right-wheel condition (7.46) can be written in form of a quadratic inequation as

$$A_{3,4}(s,\dot{s})\ddot{s}^2 + B_{3,4}(s,\dot{s})\ddot{s} + C_{3,4}(s,\dot{s}) \leqslant 0, \tag{7.51}$$

where coefficients A_3, B_3, C_3 are used in case $\ddot{s} \leqslant 0$, and coefficients A_4, B_4, C_4 in case $\ddot{s} \geqslant 0$. These coefficients are obtained as

$$A_{3,4}(s,\dot{s}) = \left(\frac{m}{2} + \frac{J\kappa(s)}{d_w}\right)^2 - \left(\frac{\mu m h_c}{2d_{cw}}\right)^2, \tag{7.52a}$$

$$B_{3,4}(s,\dot{s}) = \frac{2J\frac{d\kappa(s)}{ds}\left(\frac{m}{2} + \frac{J\kappa(s)}{d_w}\right)}{d_w}\dot{s}^2 \mp \frac{m^2\mu^2 h_c\left(\frac{a_g}{2} + \frac{h_c\kappa(s)\dot{s}^2}{d_w}\right)}{d_{cw}}, \tag{7.52b}$$

$$C_{3,4}(s,\dot{s}) = \left(\frac{J^2}{d_w^2}\left(\frac{d\kappa(s)}{ds}\right)^2 + \frac{1}{4}m^2\kappa(s)^2\right)\dot{s}^4 - m^2\mu^2\left(\frac{a_g}{2} + \frac{h_c\kappa(s)\dot{s}^2}{d_w}\right)^2. \tag{7.52c}$$

From (7.51) we get lower and upper acceleration limits that suppress sliding of the right wheel for non-zero longitudinal velocity

$$L_5(s,\dot{s}) = \frac{-B_3(s,\dot{s}) - \sqrt{B_3(s,\dot{s})^2 - 4A_3(s,\dot{s})C_3(s,\dot{s})}}{2A_3(s,\dot{s})}, \tag{7.53a}$$

$$U_5(s,\dot{s}) = \frac{-B_4(s,\dot{s}) + \sqrt{B_4(s,\dot{s})^2 - 4A_4(s,\dot{s})C_4(s,\dot{s})}}{2A_4(s,\dot{s})}. \tag{7.53b}$$

For $\kappa(s) = 0$ and $\dot{\kappa}(s) = 0$, (7.53) has the same form as (7.50).

To obtain the limits in terms of parameter s for $v = 0$ (pure rotation) we substitute (7.6) and (7.7) into (7.43), which yields the condition for both wheels as

$$\frac{J}{d_w}|\ddot{s}| \leqslant \frac{\mu m a_g}{2}, \tag{7.54}$$

from where the acceleration limits for pure rotation are obtained as

$$L_{2,rot}(s,\dot{s}) = -\frac{\mu m a_g d_w}{2J}, \quad U_{2,rot}(s,\dot{s}) = \frac{\mu m a_g d_w}{2J}. \tag{7.55}$$

Finally, acceleration limits are obtained using equation (7.18), i.e. for $v \neq 0$ we have

$$L(s,\dot{s}) = \max_{i \in 1\ldots5} L_i(s,\dot{s}), \quad U(s,\dot{s}) = \min_{i \in 1\ldots5} U_i(s,\dot{s}), \tag{7.56}$$

while for $v = 0$

$$L_{rot}(s,\dot{s}) = \max_{i \in 1,2} L_{i,rot}(s,\dot{s}), \quad U_{rot}(s,\dot{s}) = \min_{i \in 1,2} U_{i,rot}(s,\dot{s}). \tag{7.57}$$

The general rule for robot model development is that a more accurate model enables driving at higher velocities, but of course then the computational complexity is increased. The developed model already covers all the most important real-world effects so it can be used at high velocities. Nevertheless, it can be further improved by introducing a better friction model and modeling a friction introduced by the castor wheels.

7.3.3 Computing the Velocity Limit Curve

Computation of the velocity limit curve is complicated by the fact that min(·) and max(·) functions are used in equation (7.20). Thus we do not know in advance from which conditions to compute the curve. In other words, we must check all possible combinations of i and j in equation $L_i(s,\dot{s}) = U_j(s,\dot{s})$, which could be computationally expensive for large number of constraints.

However, if the values of robot parameters are known in advance, we can substitute them into corresponding equations to predict which i and j combinations are relevant. If we do that for the soccer robot, it becomes evident (as will be demonstrated by the experimental results) that for straight path segments ($\kappa(s) = 0$), only the condition $L_1(s,\dot{s}) = U_1(s,\dot{s})$ is relevant for the velocity limit curve. Similarly, for $\kappa(s) > 0$ (left curve) only the condition $L_4(s,\dot{s}) = U_4(s,\dot{s})$ for the left wheel is relevant, while for $\kappa(s) < 0$ (right curve) only the condition $L_5(s,\dot{s}) = U_5(s,\dot{s})$ of the right wheel should be evaluated. This significantly speeds up computation of the velocity limit curve. It must be emphasized that these results are not general—for different robots or parameter values the involved conditions might change.

Thus for $\kappa(s) = 0$, from $L_1(s,\dot{s}) = U_1(s,\dot{s})$ and using (7.25) the velocity limit curve is

$$V_{11}(s) = \bar{v}_{max}. \tag{7.58}$$

For $\kappa(s) > 0$, by using (7.49) the relevant condition $L_4(s, \dot{s}) = U_4(s, \dot{s})$ yields

$$\begin{aligned} &B_1(s,\dot{s}) + \sqrt{B_1(s,\dot{s})^2 - 4A_1(s,\dot{s})C_1(s,\dot{s})} \\ &\qquad = B_2(s,\dot{s}) - \sqrt{B_2(s,\dot{s})^2 - 4A_2(s,\dot{s})C_2(s,\dot{s})}, \end{aligned} \tag{7.59}$$

from which we obtain equation of the form $f(\dot{s}, \kappa(s), \dot{\kappa}(s)) = 0$. Solving this equation for $\dot{s}$ gives the velocity limit curve $V_{44}(s)$. A similar equation is obtained for $\kappa(s) < 0$ for the condition $L_5(s, \dot{s}) = U_5(s, \dot{s})$. Unfortunately, these equations do not have closed-form solution. The possible solution of this problem is discussed in the next section that discusses implementation aspects.

7.3.4 Implementation Aspects

As it was shown, the soccer robot has quite complicated expressions for acceleration limits and velocity limit curve. To enable trajectory planning in real time it is necessary to apply some optimizations to accelerate the calculations. The first possible optimization comes from the fact that for straight segments equations for acceleration limits and velocity limit curve become particularly simple; it is possible to explicitly find velocity limit curve, as well as the solution of integrals in the time-scaling algorithm. This makes possible to execute time-scaling algorithm even for long paths in real-time, as such paths will most likely contain many straight-line segments.

For segments with non-zero curvature such an optimization is not possible as the equations are too complicated to find an explicit solution. However, numerical integration is further optimized by making integration step variable, so that a greater step is used when acceleration limits change slowly, and a lower step otherwise.

As was already stated in Section 7.2, equation (7.59) for computing the velocity limit curve for the soccer robot does not have analytical solution and must be solved numerically. As it would be too slow to compute it online, the lookup-table approximation is used. The equation is solved numerically offline for different pairs of $(\kappa(s), \dot{\kappa}(s))$, and results are stored in the lookup table and reused later for online computations.

7.3.5 Experimental Results

The values of model parameters for the soccer robot are taken (with some corrections) from [47] and are shown in Table 7.1. To account for modeling uncertainties and disturbances, in praxis we will of course be more conservative and use lower acceleration limits than the maximums shown in the table. Also, the friction coefficient may vary depending on the floor type or other factors, so that it is also recommended to use somewhat lower value than one shown in the table.

Table 7.1. *Parameters of the soccer robot.*

Parameter	*Symbol*	*Value*	*Unit*
Mass	m	0.4924	kg
Inertia moment	J	$4 \cdot 10^{-4}$	kg m^2
Min. long. acceleration	$\bar{a}_{min}$	-2.5	m/s^2
Max. long. acceleration	$\bar{a}_{max}$	2.5	m/s^2
Min. ang. acceleration	$\bar{\alpha}_{min}$	-74.2	rad/s^2
Max. ang. acceleration	$\bar{\alpha}_{max}$	55.8	rad/s^2
Max. long. velocity	$\bar{v}_{max}$	4	m/s
Max. ang. velocity	$\bar{\omega}_{max}$	32	rad/s
Friction coefficient	μ	0.6	–
Mass center height	h_c	0.025	m
Distance between driving wheels	d_w	0.068	m
Distance to cast. wheels	d_{cw}	0.025	m
Gravitation	a_g	9.81	m/s^2

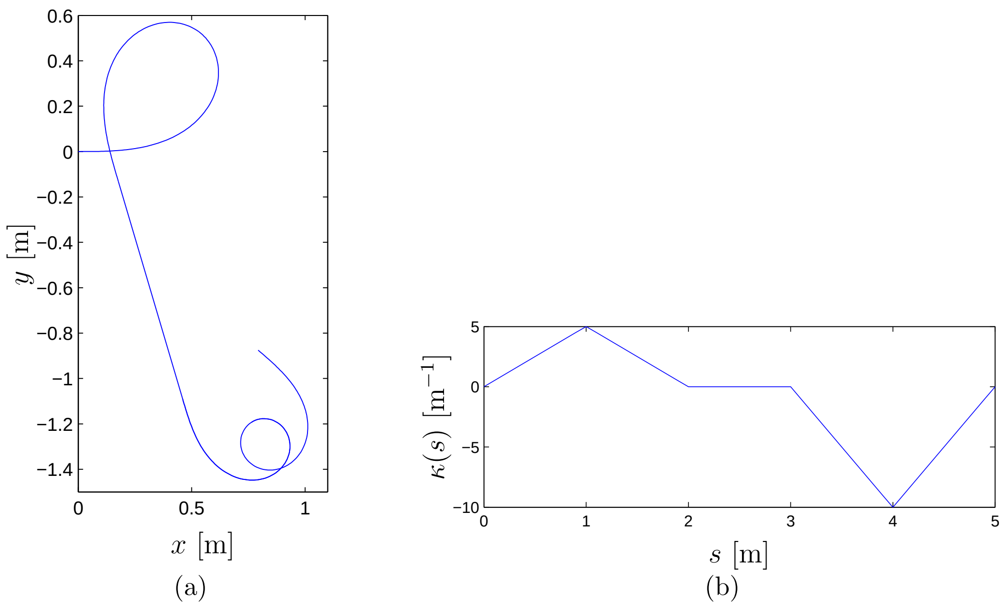


Figure 7.8. (a) *The experimental path.* (b) *Curvature profile of the path.*

A path that was used in all experiments, together with its curvature profile, is shown in Figure 7.8. The experimental path consists of two clothoids with positive curvature, a straight part and two clothoids with negative curvature.

For this path acceleration limits and velocity limit curves $V_{ii}(s)$ are determined separately for each constraint $L_i(s, \dot{s}) = U_i(s, \dot{s}),\ i \in 1 \ldots 5$ and are displayed in Figure 7.9 (a)–(e). In Figure 7.9 (f) all velocity limit curves are shown in different colors. The overall velocity limit curve $V(s)$ can now be obtained as the minimum of all velocity limit curves.

In Figure 7.9 (f) it can be noticed that for positive curvature only the velocity limit curve $V_{44}(s)$ is relevant (shown in blue), while for negative curvature only $V_{55}(s)$ matters (red). In zero-curvature segment, only the velocity limit curve $V_{11}(s)$ is relevant. It can be seen that the velocity limit $V_{44}(s)$ (ensures positive weight force on driving wheels) is never relevant, as non-slipping conditions are always stronger. The velocity limit $V_{22}(s)$ (angular acceleration limit) is also not relevant in any situation, however it could become important in pure rotation mode, which is not shown here.

The acceleration limits, which in Figure 7.9 are illustrated with cones, are dominantly determined by condition $L_1(s,\dot{s}) \leqslant \ddot{s} \leqslant U_1(s,\dot{s})$, while conditions $L_{3,4}(s,\dot{s}) \leqslant \ddot{s} \leqslant U_{3,4}(s,\dot{s})$ are relevant in curves and can be observed only near their corresponding velocity limit curves. The fact that the acceleration limits are dominantly determined by $L_1(s,\dot{s})$ and $U_1(s,\dot{s})$ (imposed by robot's internal longitudinal acceleration limit), and not by the available grip force, leads us to conclusion that robot's internal acceleration limit is too restrictive and much faster motion could be obtained on the straight path segments by increasing this limit. Unfortunately, available soccer robot has closed architecture and does not permit the modifications in its firmware.

It can also be noticed that tangents at the velocity limit curve are always horizontal, i.e. no accelerations other than $\ddot{s} = 0$ are feasible. This makes implementation of the time-optimal scaling algorithm easier, because at non-zero curvature path segments the minimum of the velocity limit curve is the only point where acceleration is tangent to the velocity limit curve. On the contrary, every point on the velocity limit curve corresponding to zero-curvature path segments has acceleration that is tangent to the velocity limit curve (this is zero inertia arc).

Finally, the optimal time-scaling algorithm is implemented and applied to the given path. The obtained velocity profile is shown in Figure 7.10. It can be seen that the time-optimal velocity profile contains three maximum acceleration and three maximum deceleration segments. The total travel time is 5.1 s, which gives an average velocity of 0.99 m/s.

The experiment has also been conducted with the goal to find out whether wheel sliding occurs during execution of the velocity profile in Figure 7.10. For this a high fidelity simulator [82] based on ODE (Open Dynamics Engine) [140] library was used. The reason of using simulator instead of the real robot is that the simulator enables detection of wheel sliding, while doing so on the real robot is hard to achieve. The robot was driven in open loop because the goal was not precise trajectory tracking but only checking feasibility of the generated commands. No sliding was detected during simulation, even on path segments with high curvature, which proves effectiveness of the developed algorithm. By scaling up the velocity profile for 5 % the robot was no more able to execute the

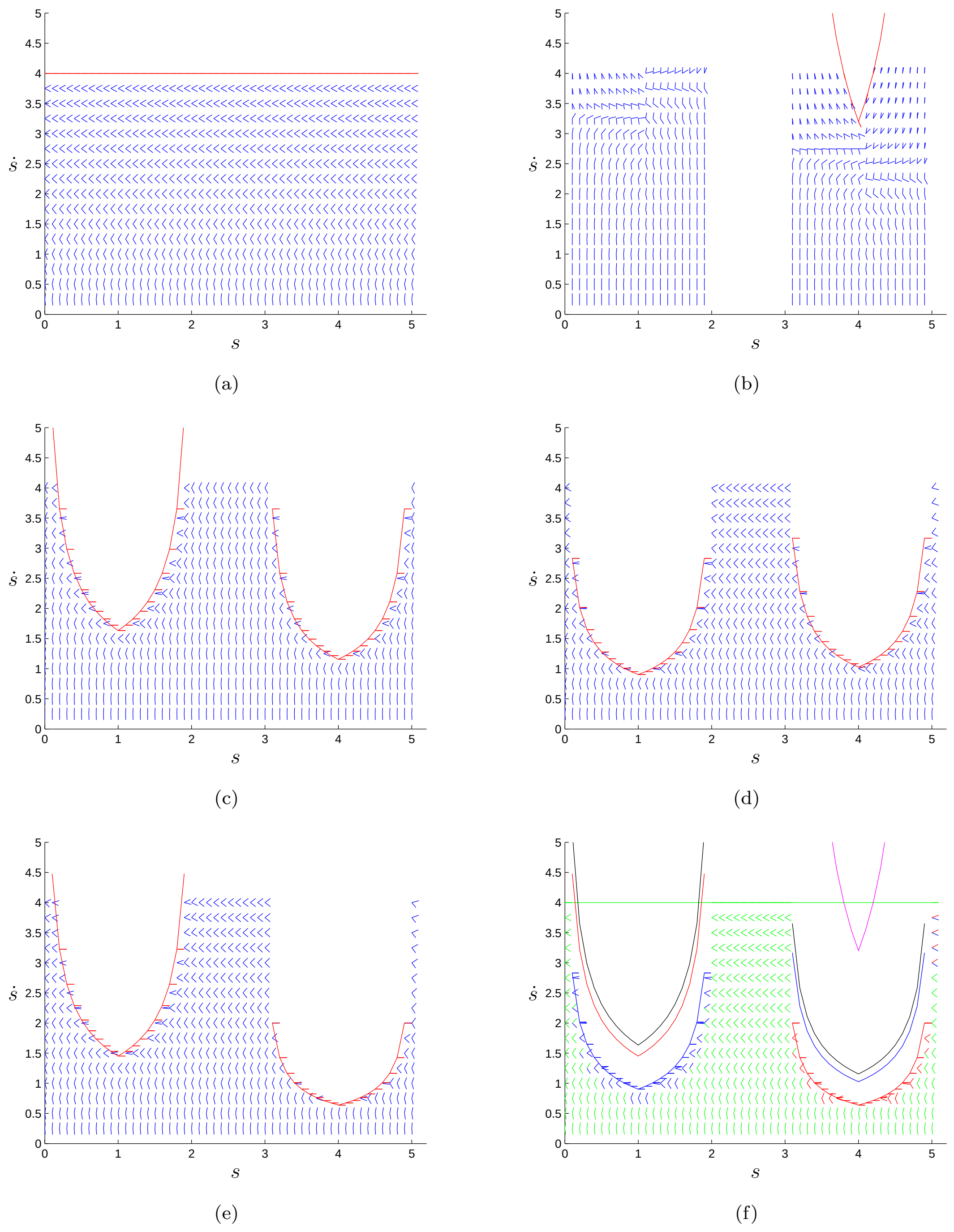


Figure 7.9. *Acceleration limits and velocity limit curves computed for the soccer robot using the path in Figure 7.8.* (a) *Velocity limit curve* $V_{11}(s)$. (b) *Velocity limit curve* $V_{22}(s)$. (c) *Velocity limit curve* $V_{33}(s)$. (d) *Velocity limit curve* $V_{44}(s)$. (e) *Velocity limit curve* $V_{55}(s)$. (f) *Overall velocity limit curve* $V(s)$ *is the minimum of all velocity limit curves. Acceleration limits* $L_i(s, \dot{s})$, $U_i(s, \dot{s})$, *and velocity limit curves* $V_{ii}(s)$ *originating from different constraints are plotted by the following colors:* $i = 1$ *(green),* $i = 2$ *(magenta),* $i = 3$ *(black),* $i = 4$ *(blue),* $i = 5$ *(red).*

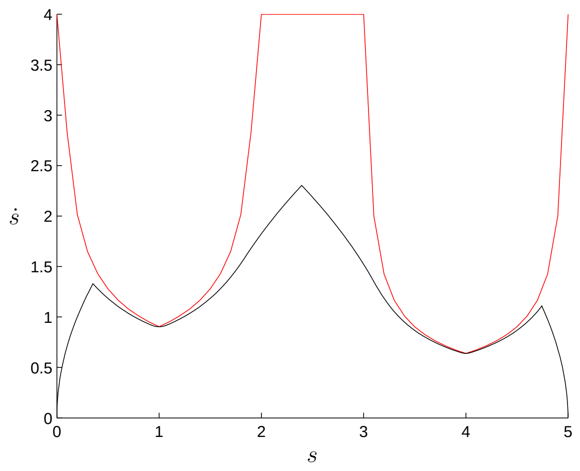


Figure 7.10. *Time-optimal velocity along the experimental path for the soccer robot is plotted in black color, while the overall velocity limit curve is red curve.*

trajectory, both because of acceleration limits and sliding, so that it confirms that the original trajectory is time optimal.

7.4 Trajectory Planning for Pioneer 3DX Robot

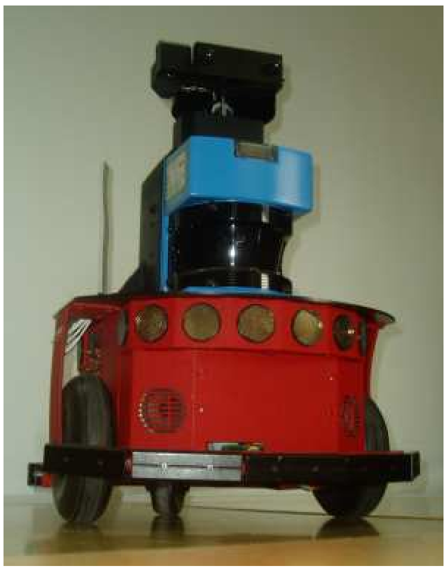


Figure 7.11. *Pioneer 3DX mobile robot.*

Pioneer 3DX robot (Figure 7.11) is a typical representative of middle-size differential drive mobile robots. It has two driving wheels and one castor wheel.

Due to its relatively heavy weight (more than 25 kg with sensors), wheel slipping is no more of big concern for this robot.

Some preliminary experiments [61] have shown that this robot, like soccer robot, also can attain maximum longitudinal and angular acceleration simultaneously. Under this assumption, the longitudinal and angular accelerations are independent and the model of robot's intrinsic constraints developed for the soccer robot, given by equations (7.25) and (7.28), is also valid for the Pioneer 3DX robot.

However, the assumption of independent longitudinal and angular accelerations yet has to be confirmed under the full load (all sensors and batteries) of the robot, as the relevant experiments in [61] are performed with no additional load. At the full weight of the robot and at the maximum longitudinal acceleration it is possible that motors approach to their torque limits, so that the maximum angular acceleration would no more be feasible at the same time. Then longitudinal and angular accelerations could no more be considered as independent, and two additional constraints on wheel torques should have been added.

Regarding extrinsic constraints, by substituting parameter values of the Pioneer 3DX robot in Table 7.2 into equations (7.43) and (7.44) that refer to the maximum grip force of the driving wheels[1], and considering the maximum accelerations and velocities (worst case), it is obvious that developed forces never exceed available grip force. This is the reason why for the Pioneer 3DX robot wheel slipping is not an issue, so that only intrinsic constraints can be considered.

7.4.1 Implementation Aspects

For Pioneer 3DX robot equations for the velocity limit curve and acceleration limits are particularly much simpler than for the soccer robot. Thus it is possible to express a closed forms of the velocity limit curve and find an explicit solution of integrals in time-scaling algorithm. This is planned for future implementation and it is expected to result with very fast trajectory planning.

7.4.2 Experimental Results

The values of model parameters for the Pioneer 3DX robot that are used for trajectory planning experiments are taken from [120] and are shown in Table 7.2. To account for modeling uncertainties and disturbances, in final implementation we need to be more conservative and use accelerations somewhat lower than maximums shown in the table.

The same experimental path as for the soccer robot is used (Figure 7.8). Acceleration limits L_i and U_i and velocity limit curves V_{ii} obtained from constraints

[1]Actually the modified equations (7.43) and (7.44) have been used that account for different position of robot's center of the mass.

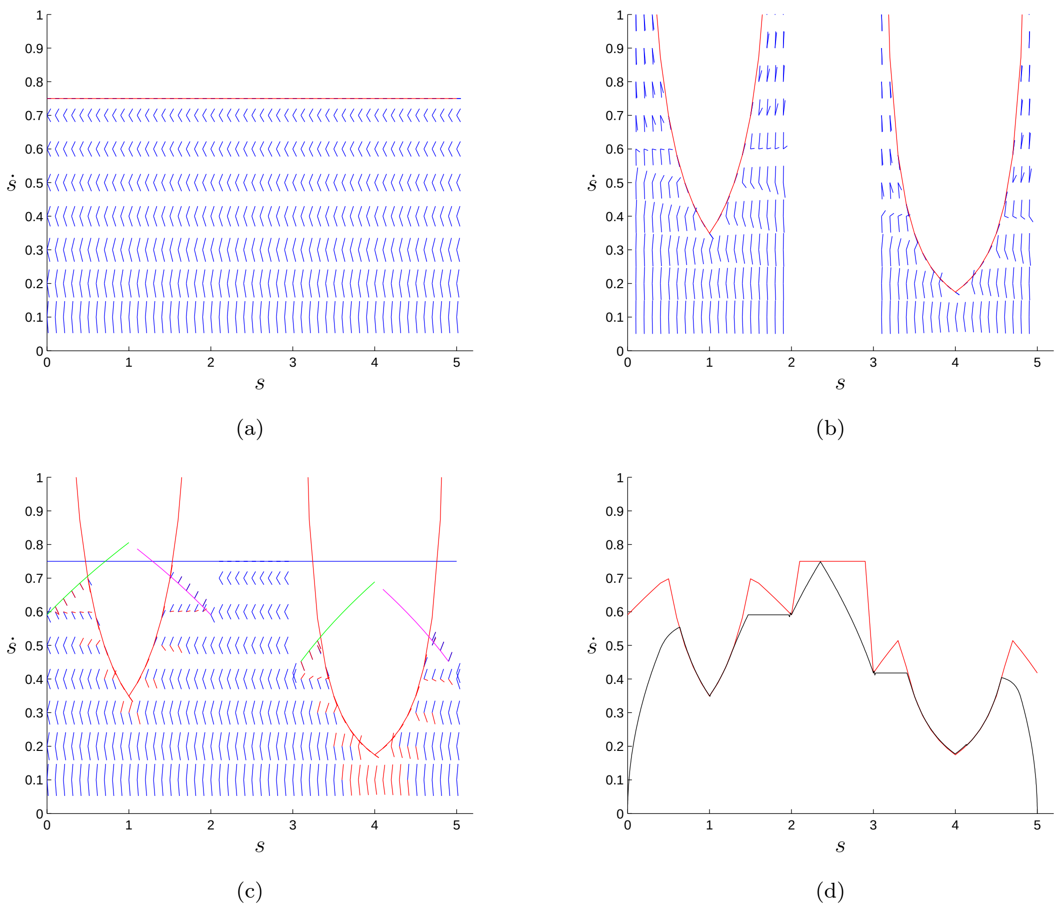


Figure 7.12. *Acceleration limits and velocity limit curves computed for the Pioneer 3DX robot with path in Figure 7.8.* (a) *Velocity limit curve* $V_{11}(s)$. (b) *Velocity limit curve* $V_{22}(s)$. (c) *Overall velocity limit curve* $V(s)$ *is the minimum of all velocity limit curves. Acceleration limits* $L_i(s, \dot{s})$, $U_j(s, \dot{s})$, *and velocity limit curves* $V_{ij}(s)$ *originating from different constraints are plotted with the following colors:* $i = 1, j = 1$ *(blue),* $i = 2, j = 2$ *(red),* $i = 1, j = 2$ *(green),* $i = 2, j = 1$ *(magenta).* (d) *Time-optimal velocity along the experimental path is plotted in black, while the overall velocity limit curve* $V(s)$ *is shown in red.*

Table 7.2. *Parameters of the Pioneer 3DX robot.*

Parameter	*Symbol*	*Value*	*Unit*
Mass	m	28.05	kg
Inertia moment	J	0.175	kg m^2
Min. long. acceleration	$\bar{a}_{min}$	-0.3	m/s^2
Max. long. acceleration	$\bar{a}_{max}$	0.3	m/s^2
Min. ang. acceleration	$\bar{\alpha}_{min}$	-1.745	rad/s^2
Max. ang. acceleration	$\bar{\alpha}_{max}$	1.745	rad/s^2
Max. long. velocity	$\bar{v}_{max}$	0.75	m/s
Max. ang. velocity	$\bar{\omega}_{max}$	1.745	rad/s
Friction coefficient	μ	0.6	–
Mass center height	h_c	0.18	m
Distance between driving wheels	d_w	0.27	m
Distance to cast. wheels	d_{cw}	0.25	m
Gravitation	a_g	9.81	m/s^2

$L_i(s, \dot{s}) = U_i(s, \dot{s})$ are shown in Figures 7.12 (a) and (b) for $i = 1$ and $i = 2$, respectively. The planning for Pioneer 3DX robot differs also also in the fact that the velocity limit curves V_{12} and V_{21} are relevant, which were not important for the soccer robot. These velocity limit curves are obtained from conditions $L_1(s, \dot{s}) = U_2(s, \dot{s})$ and $L_2(s, \dot{s}) = U_1(s, \dot{s})$, respectively. In Figure 7.12 (c) all velocity limit curves V_{ij} are shown in different colors, and the overall velocity limit curve $V(s)$ is their minimum. It can be noticed that in zero-curvature segment only the velocity limit curve $V_{11}(s)$ is relevant, again.

The final velocity profile is shown in Figure 7.12 (d). It can be seen that it consists of three maximum acceleration segments, three maximum deceleration segments and two constant velocity segments. The total travel time of the obtained trajectory is approximately 14 s, while the average longitudinal velocity is 0.36 m/s. Therefore, on the given path the Pioneer 3DX robot is about three times slower than the soccer, which is expected due to its large mass. In the future it is planned to test the trajectories with the real robot.

7.5 Moving Obstacles

Having covered basic stages of the decoupled planning, we can now consider some problems beyond basic planning, first of all the problem of moving obstacles. In current implementation the motion planner handles moving obstacles simply by replanning the path. However, as no prediction of future motion of the obstacles is performed, this approach is satisfactory only for slow-moving obstacles. If the obstacles are fast, the lack of prediction can result with non-optimal motion, or even with collision.

To better handle moving obstacles, we must assume that the future motion of obstacles is at least partially known, meaning that the future trajectory for the moving obstacles can be estimated based on acquired sensor data (e.g. by extrapolating current position, velocity and direction of motion). There are many researches on planning in dynamic environments (see e.g. [149]). The simplest approach is to extend the configuration space $\mathcal{C}$ with time dimension T, obtaining the state space X as the Cartesian product $X = \mathcal{C} \times T$. The complete planning is possible only if motions of the obstacles are perfectly known in advance.

The introduced state space X is in many ways similar to C-space seen so far, but there is one critical difference—time goes forward. Thus there must be a way of preventing the algorithm of generating paths that travel backward in time. If the roadmap method is used to solve the problem, the resulting roadmap must be directed with all edges being time-monotonic. Sampling-based methods can be applied as well.

An efficient alternative is again to decouple the problem into path planning and velocity planning parts. Here the path is planned as usual, avoiding the stationary obstacles. Moving obstacles are accounted in the second part, so that velocity of the robot along the pre-planned path is tuned to avoid collision with the moving obstacles. Therefore this approach is called the *velocity-tuning method* [89]. This results in a two-dimensional configuration-time space, as a configuration along a path can be described using one parameter. As a result, moving obstacles manifest as inadmissible “islands” in $(s, \dot{s})$ plane. In this case optimal time-scaling algorithm is more complicated and is described in [136]. As moving obstacles in $(s, \dot{s})$ phase plane do not have a fixed position with respect to $\dot{s}$ axis, it is better to perform the velocity tuning in (t, s) plane (configuration-time space) instead of $(s, \dot{s})$ plane.

Obviously, algorithms based on the decoupled approach are not complete, yet they provide a way to efficiently plan a collision-free trajectory through the configuration-time space. The incompleteness problem can be alleviated by using path adaptation in addition to velocity tuning. This can be obtained by treating a moving obstacle as the static one and adding it into the static roadmap. But instead of using its current position, position at the moment when the robot is estimated to first time hit the obstacle is used as it was done e.g. in [27]. In this process obstacles may also be dilated in order to account for uncertainty. This algorithm can be used in case that there is enough space to adapt the path. Otherwise, velocity tuning can be used. In this way a powerful combination is obtained that circumferences problem of incompleteness in almost all circumstances.

7.6 Multiple Robots

There are many problems that can be solved easier, more efficient or cheaper by using multiple robots. Examples include delivery in factories or warehouses, robot soccer etc. In multi-robot planning the problem is to plan motion for a set of robots in order to bring each robot from some start configuration to some goal configuration without mutual collisions and collisions with stationary obstacles. Finding paths for the individual robots only guarantees here that there are no collision with obstacles, but there is still possibility that two or more robots are in collision. Thus, motion plans have to be coordinated so that none of the robots are in collision. This makes the problem significantly harder than in the case of a single robot. The problem can be solved using centralized and decoupled planning.

In the centralized approach the set of robots are treated as one multi-body robot with one composite configuration space. The dimensionality of this configuration space is equal to the total number of degrees of freedom of all the robots. Therefore planning directly in this space is generally too slow for online planning. However, this formulation of the problem shows that we are dealing with a static motion planning problem in which time does not play an intrinsic role.

Decoupled approach works in two stages. Initially, a collision-free path is planned for each individual robot by ignoring all other robots. In the second stage velocity tuning along the previously planned paths is used too achieve coordination between robots. In this way dimensionality of the configuration space is not increased. Alternatively, a *prioritized planning* can be used. It is applied as follows: first, each robot is given a priority. Then, in order of priority a trajectory is planned for each individual robot, where robots with higher priority are treated as moving obstacles for the lower-priority robots. Thus, here multiple instances must be solved of single robot planning problem in a known dynamic environment. Decoupled multi-robot planning is very fast, but it is not complete, as the robots are not able to coordinate their motions, which is sometimes necessary. However, in most practical situations the prioritized approach is reported to work very well [149].

7.7 Summary

In this chapter the optimal time-scaling algorithm is presented for trajectory planning along the predefined path while respecting the actuator constraints. An overview of planning algorithms for moving obstacles and multiple robots is also given.

A contribution of this chapter is development of a dynamic model of two differential-drive robots: soccer robot and Pioneer 3DX robot. The model accounts for velocity and acceleration limits, as well as limited grip between the wheels and the ground, which is very important for reliable robot operation at high velocities. This model is then used to express acceleration limits and velocity limit curve for the optimal time-scaling algorithm, which are further used for algorithm implementation.

Developed algorithm can also be adapted for other types of limitations. E.g. it could be used to prevent tip over of a mobile robot used for human-robot interaction. For such interaction to be practical, those robots are usually high and therefore their motion can be unstable at high accelerations. This can be solved by adding an additional extrinsic constraint on the maximum value of the overall acceleration, which is a vector sum of the longitudinal and the lateral (i.e. centrifugal) acceleration. The developed model of intrinsic and extrinsic constraints can also be used in combination with reactive planners in order to check whether a particular command is admissible for the robot.

Besides optimization of the developed trajectory planning algorithms, future works will include extending of basic trajectory planning for the case of moving obstacles and implementation of the prioritized approach for multi-robot planning.

Chapter 8

Trajectory Tracking

This chapter presents an experimental overview of three common trajectory tracking methods for nonholonomic mobile robots: linear control, nonlinear control and model predictive control. All methods are compared experimentally on a two-wheel mobile robot with differential drive. The goal was to determine which control method is the best with respect to robustness and low trajectory tracking error. Thereby, a special emphasis is given to real-time trajectory tracking and behavior in conditions near the robot velocity and acceleration limits. This chapter was previously published as [22], while an early version was published in [23].

8.1 Introduction

Trajectory tracking is the last module in the chain of modules of the decoupled motion planning approach. The input of the trajectory tracking module is a feasible trajectory planned by the previous modules, where feasibility refers to the ability of a robot to actually track the planned trajectory. This means that the planned trajectory respects various robot physical and dynamical limitations such as its velocity and acceleration limits, as described in Chapter 7.

The task of a trajectory tracking algorithm is to ensure that the robot actually tracks the planned trajectory. One may wonder why we do not achieve this by simply letting the robot execute commands obtained from the planned trajectory, which theoretically should ensure perfect tracking. However, various real world issues would almost certainly result by a failure of such a simple plan—the tracking error would typically accumulate over time unboundedly. The sources of error are numerous and include unperfect robot model used for trajectory planning, external disturbances such as non-flat floor, delay of commands, unperfect measurement of robot initial state etc.

The problem of accumulating error can be addressed by frequent trajectory replanning, however it is better to replan the trajectory only if it is absolutely necessary (e.g. if an obstacle has moved), in this way saving computational time for other tasks. In case that tracking error is small, the problem is best solved by online reactive algorithm that uses sensor readings to generate appropriate robot commands in order to correct the tracking error. More precisely, such an algorithm should ensure that a selected reference point on the robot follows the planned trajectory. In other words, the distance between the reference point on the robot and current reference point of the trajectory must be kept as small as possible.

The stated problem is actually a control problem that can be formulated as follows: based on reference state from the trajectory and measured state of the robot used as a feedback, generate the robot commands such that the tracking error is stabilized. The corresponding algorithm is called *trajectory tracking controller*. Here we assume that the variables used for feedback in the control loop (typically the position and orientation of the mobile robot with respect to a fixed frame) can be reliably measured by either distributed sensors of the iSpace, onboard robot sensors, or both.

Various approaches to this problem that cover different mobile robot architectures regarding number and type of wheels, their location etc. can be found in the literature (see [111] for a general overview). In practical applications, the most common configurations are robots with differential drive or car-like drive and omnidirectional steering. In this work a problem of trajectory tracking for mobile robot with differential drive is addressed, which is a typical example of nonholonomic mechanism where the driving wheels can only roll without slipping. However, most algorithms can also be applied for more complex robot configurations. Nonholonomic constraints have motivated the development of various highly nonlinear control techniques.

A problem of trajectory tracking controller design using a nonlinear feedback action is addressed in [127]. In [42] the same problem is solved independently using dynamic feedback linearization. Trajectory tracking controllers based on linear control, nonlinear control and dynamic feedback linearization are described and compared in [93]. Advanced control methods, such as model predictive control, are also commonly used. Examples of using model predictive controller for trajectory tracking of nonholonomic systems can be found in [117] and [115], and a newer example is in [83]. As the trajectory tracking controller is in its nature a reactive algorithm, it can be implemented efficiently and thus can incorporate some real world issues that are hard to model at the planning time, such as friction in the actuators. One such controller that is based on fuzzy control is proposed in [124].

Unfortunately, an extensive experimental comparison of denoted control approaches in equal conditions is not well covered in the literature, especially not at high robot velocities. Therefore this chapter is concerned with an experimental comparison of several control architectures in conditions near the robot velocity and acceleration limits. The comparison also encompasses controller implementation issues and problem of determining controller parameters. In this work it is assumed that the most important limitations of the robot (acceleration limits etc.) are already well modeled in the trajectory planning module, so that the trajectory tracking controller need not to encompass these issues. Thus, simpler controller architectures are selected, with the additional demand that the controller must be simple enough to be implemented in the microcontroller of the robot. With this in mind, linear-design controller and nonlinear controller [93], which are simple and pure reactive algorithms, and model predictive controller (MPC) [83] as an representative of more advanced control methodologies are selected for comparison. All of the three selected controllers can be implemented in the microcontroller, although for MPC a more powerful microcontroller is required.

8.2 Robot Model

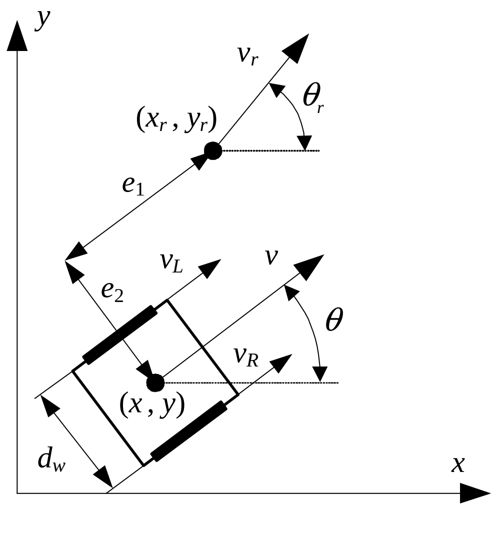


Figure 8.1. *Robot model*

A mobile robot with differential drive is kinematically equivalent to a unicycle and its kinematic model is already given by equation (7.22). This equation transforms longitudinal velocity v and angular velocity ω expressed in coordinates of mobile robot base to its velocities expressed in global Cartesian coordinates (Figure 8.1). Also, the nonholonomic constraint that the driving wheels purely roll and do not slip must be fulfilled, which is given by equation (7.23).

The mapping between longitudinal (driving) and angular (steering) velocities and circumferential velocities of the wheels, which are sometimes used as the commands to the system, is given by:

$$v_R = v + \omega d_w/2, \quad v_L = v - \omega d_w/2, \tag{8.1}$$

where d_w is the distance along the axle between the centers of the drive wheels, and v_R and v_L are circumferential velocities of the right and the left wheel, respectively.

It is important to mention that dynamics of a robot is not modeled by the kinematic model. Therefore, it is assumed that robot exactly realizes velocity commands v and ω. Of course, due to robot and actuator dynamics, and also non-idealities such as friction, gear backslash, wheel slippage, actuator deadzone and saturation, the robot cannot exactly realize velocity commands. Nevertheless, as we have assumed that the command trajectory is feasible for the robot, robot can approximately track the reference velocity commands.

As a safety measure, in case when trajectory tracking controller temporarily generates command velocities higher than robot limitations, a velocity saturation will occur. During this time it is important that velocity commands are saturated so that robot retains its steering direction, i.e. path curvature. This means that ratio ω/v (which is equal to path curvature) has to be preserved. Therefore, if $\bar{v}_{max}$ and $\bar{\omega}_{max}$ are maximum robot longitudinal and angular velocity commands, respectively, bounded command velocities v_c and ω_c that preserve path curvature are obtained by first defining

$$\sigma = \max\left\{|v|/\bar{v}_{\max},\ |\omega|/\bar{\omega}_{\max},\ 1\right\}.$$

Then the following algorithm is used to determine bounded velocities [118]:

$$\begin{array}{c} v_c = v, \quad \omega_c = \omega, \quad \text{if } \sigma = 1; \\ v_c = \text{sign}(v)\bar{v}_{max}, \quad \omega = \omega/\sigma, \quad \text{if } \sigma = |v|/\bar{v}_{max}; \\ v_c = v/\sigma, \quad \omega = \text{sign}(\omega)\bar{\omega}_{max}, \quad \text{else.} \end{array} \tag{8.2}$$

8.3 Trajectory Tracking Controllers

Trajectory tracking problem is a problem of computing appropriate robot commands so that robot (i.e. reference point on the robot) tracks a reference point on a trajectory. Hereby, a trajectory is given as a function that associates a robot configuration to each time instance t, i.e. $t \to q_r(t)$ (see equation (7.1)). In this chapter reference robot configuration imposed by the planned trajectory will be denoted by vector $q_r(t) = [x_r(t), y_r(t), \theta_r(t)]^T$. The problem of trajectory tracking may also be formulated as one of controlling the vehicle in order to track a reference vehicle whose trajectory is given by $t \to q_r(t)$.

Output of the trajectory tracking controller is control vector u, which consists of command longitudinal velocity v and command angular velocity ω, i.e. $u =$

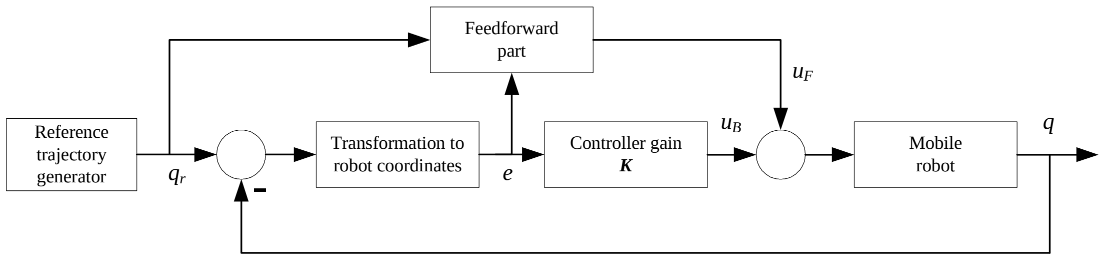

Feedforward part
u_F
Reference trajectory generator
q_r
Transformation to robot coordinates
e
Controller gain $\boldsymbol{K}$
u_B
Mobile robot
q
-

Figure 8.2. *Trajectory tracking controller structure*

$[v\ \ \omega]^T$. Commonly, control vector consists of feedforward and feedback part

$$u = u_F + u_B, \tag{8.3}$$

where u_F is feedforward control vector, and u_B is feedback control vector. The structure of general trajectory tracking controller, which consists of feedforward and feedback part, is shown in Figure 8.2.

Feedforward control vector is usually computed based on reference longitudinal velocity v_r and reference angular velocity ω_r, which are obtained from the trajectory using equations (7.3a) and (7.3b), respectively. In this work the feedforward control vector is obtained using the following nonlinear transformation of reference velocities

$$u_F = [v_r cos(e_3) \quad \omega_r]^T, \tag{8.4}$$

where $e_3 = \theta_r - \theta$ is angular error (see Figure 8.1). The term $\cos(e_3)$ in equation (8.4) has a task of avoiding high longitudinal velocities that can be counterproductive if the angular error is high. This feedforward generation algorithm is used by all presented control schemes.

The feedback control vector is computed based on current tracking error and is added to the generated feedforward commands. It consists of command longitudinal velocity v_B and command angular velocity ω_B, i.e. $u_B = [v_B\ \omega_B]^T$. Unlike feedforward part, the feedback part is different for each of the controllers presented in the sequel.

8.3.1 Linear-Design Controller

The simplest design of trajectory tracking controller is obtained using tangent linearization along the reference trajectory. The problem of minimizing difference between reference configuration and measured robot configuration is equivalent to problem of stabilization of tracking error dynamics. The tracking error $e(t)$ can be defined in the configuration space of the robot, however it is convenient to transform it to the robot local coordinate system. This transformation is performed as follows

$$e(t) = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r - x \\ y_r - y \\ \theta_r - \theta \end{bmatrix}, \tag{8.5}$$

where error component e_1 is the longitudinal error, e_2 is the orthogonal error, while e_3 is the orientation error as illustrated in Figure 8.1. By taking a derivative of equation (8.5) and taking into account robot kinematics (7.22) and (7.23), the error dynamics becomes

$$\begin{bmatrix} \dot{e}_1 \\ \dot{e}_2 \\ \dot{e}_3 \end{bmatrix} = \begin{bmatrix} \cos e_3 & 0 \\ \sin e_3 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_r \\ \omega_r \end{bmatrix} + \begin{bmatrix} -1 & e_2 \\ 0 & -e_1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}, \tag{8.6}$$

Using equations (8.3) and (8.4), equation (8.6) can be rewritten as

$$\dot{e} = \begin{bmatrix} 0 & \omega & 0 \\ -\omega & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} e + \begin{bmatrix} 0 \\ \sin e_3 \\ 0 \end{bmatrix} v_r + \begin{bmatrix} -1 & 0 \\ 0 & 0 \\ 0 & -1 \end{bmatrix} u_B. \tag{8.7}$$

Error dynamics model (8.7) is obviously nonlinear. Linearizing it around the reference trajectory ($e = 0$, $u_B = 0$, $v = v_r$, $\omega = \omega_r$) yields the linearized error dynamics as follows

$$\dot{e} = \begin{bmatrix} 0 & \omega_r & 0 \\ -\omega_r & 0 & v_r \\ 0 & 0 & 0 \end{bmatrix} e + \begin{bmatrix} -1 & 0 \\ 0 & 0 \\ 0 & -1 \end{bmatrix} u_B. \tag{8.8}$$

It can be shown that this system can be locally stabilized around reference trajectory which consists of linear or circular paths with constant velocity. Furthermore, it is shown that by using linear design methods it is possible to locally stabilize given system even for arbitrary feasible trajectories under condition that they do not come to a stop.

Now, following [93], the design procedure of the linear-design controller will be shortly described. First, the linear feedback law is defined as

$$u_B(t) = K_s e(t), \tag{8.9}$$

where the gain matrix K_s is defined as

$$K_s = \begin{bmatrix} k_1 & 0 & 0 \\ 0 & \operatorname{sign}(v_r)k_2 & k_3 \end{bmatrix}. \tag{8.10}$$

The structure of gain matrix (8.10) can be intuitively explained by looking at

Figure 8.1. The longitudinal error e_1 is reduced by manipulating the command longitudinal velocity v_B via gain k_1. The orientation error e_3 is reduced similarly by controlling command angular velocity ω_B of the robot via gain factor k_3. Finally, the orthogonal error e_2 is also reduced by changing the angular velocity, but here the driving direction (forward or backward) should also be taken into account.

Now the gains k_1, k_2, k_3 are determined by using desired closed-loop characteristic equation

$$(\lambda + 2\zeta\omega_n)(\lambda^2 + 2\zeta\omega_n\lambda + \omega_n^2), \tag{8.11}$$

with one negative eigenvalue at $-2\zeta\omega_n$ and a complex pair with natural angular frequency $\omega_n > 0$ and damping coefficient $\zeta \in (0, 1)$. The closed loop characteristic (8.11) is obtained by choosing the gains

$$k_1 = k_3 = 2\zeta\omega_n, \;\; k_2 = \frac{\omega_n^2 - \omega_r(t)^2}{|v_r(t)|}. \tag{8.12}$$

The problem with this choice of gains is that k_2 goes to infinity as reference longitudinal velocity v_r goes to zero. The possible gain scheduling scheme that solves the problem is obtained by letting $\omega_n(t) = \sqrt{\omega_r^2(t) + gv_r^2(t)}$, which gives:

$$k_1 = k_3 = 2\zeta\sqrt{\omega_r^2(t) + gv_r^2(t)}, \quad k_2 = g|v_r(t)|, \tag{8.13}$$

where parameter $g > 0$ gives an additional degree of freedom. It can be noted that with this choice of gains linear-design controller actually becomes nonlinear time-varying controller. It should be emphasized that, even if the closed-loop eigenvalues are constant and with negative real part, this control law does not guarantee the asymptotic stability of the tracking error e, as the system is time-varying. Also, this is a continuous controller, but as it has only the proportional gains, under assumption that sampling period is small, it can also be used directly as a discrete controller.

8.3.2 Nonlinear Controller

The main importance of nonlinear controller design is that its global asymptotic stability can be proven. Its feedback control is defined by the following gain matrix [126]

$$K_n = \begin{bmatrix} k_1(v_r(t), \omega_r(t)) & 0 & 0 \\ 0 & \overline{k_2} v_r(t) \frac{sin(e_3)}{e_3} & k_3(v_r(t), \omega_r(t)) \end{bmatrix}, \tag{8.14}$$

where $\overline{k_2} > 0$ is constant positive gain, and $k_1(\cdot, \cdot)$ and $k_3(\cdot, \cdot)$ are positive continuous gain functions. Assuming that v_r and ω_r are bounded and with bounded derivatives, and that $v_r(t) \nrightarrow 0$ and $\omega_r(t) \nrightarrow 0$ when $t \rightarrow \infty$, using Lyapunov analysis one can prove that the control law (8.14) globally asymptotically stabilizes the origin $e = 0$.

Following the design of previous linear-design controller, the gain functions k_1 and k_3, and the constant gain $\overline{k}_2$ can be chosen as

$$\begin{gathered} k_1(v_r(t), \omega_r(t)) = k_3(v_r(t), \omega_r(t)) = 2\zeta\sqrt{\omega_r^2(t) + g v_r^2(t)}, \\ \overline{k}_2 = g, \end{gathered} \tag{8.15}$$

where $g > 0$ and $\zeta \in (0, 1)$.

8.3.3 Model Predictive Controller

The main idea of the model predictive controller (MPC) is to minimize a difference between used-defined reference trajectory tracking error and predicted trajectory tracking error, as well as control effort. Accordingly, a quadratic cost function can be written as

$$J(u_B, k) = \sum_{i=1}^{h} \epsilon^T(k,i)\, Q\, \epsilon(k,i) + u_B^T(k,i)\, R\, u_B(k,i), \tag{8.16}$$

where $\epsilon(k,i) = e_r(k+i) - e(k+i|k)$ is difference between reference (desired) trajectory tracking error $e_r(k+i)$ and predicted trajectory tracking error $e(k+i|k)$, h is prediction horizon interval, and Q and R are weighting matrices, where $Q \geq 0,\ Q \in \mathbb{R}^n \times \mathbb{R}^n$ and $R \geq 0,\ R \in \mathbb{R}^m \times \mathbb{R}^m$, n is the number of state variables and m is the number of input variables.

For the design purpose, error dynamics (8.8) is discretized as

$$e(k+1) = Ae(k) + Bu_B(k), \tag{8.17}$$

where $A \in \mathbb{R}^n \times \mathbb{R}^n$ and $B \in \mathbb{R}^n \times \mathbb{R}^m$. Discrete model matrices A and B for short sampling time T_s can be well approximated as

$$\begin{gathered} A = I + A_c T_s \\ B = B_c T_s. \end{gathered} \tag{8.18}$$

Following [83], error prediction in h steps ahead can be written as

$$\begin{aligned} e(k+h|k) = \prod_{j=1}^{h-1} A(k+j|k)e(k) + \sum_{i=1}^{h} \left(\prod_{j=i}^{h-1} A(k+j|k) \right) \\ \times B(k+i-1|k)u_B(k+i-1) \\ + B(k+h-1|k)u_B(k+h-1). \end{aligned} \tag{8.19}$$

The vector of trajectory tracking error predictions is defined as

$$E^*(k) = \left[e(k+1|k)^T \; e(k+2|k)^T \; \dots \; e(k+h|k)^T\right]^T \tag{8.20}$$

where $E^* \in \mathbb{R}^{n \cdot h}$. If the control vector is defined as

$$U_B(k) = \left[u_B(k)^T \; u_B(k+1)^T \; \dots \; u_B(k+h-1)^T\right]^T, \tag{8.21}$$

where $U_B \in \mathbb{R}^{m \cdot h}$, and

$$\Lambda(k,i) = \prod_{j=i}^{h-1} A(k+j|k), \tag{8.22}$$

the vector of trajectory tracking error predictions can be written in matrix form as

$$E^*(k) = F(k)e(k) + G(k)U_B(k), \tag{8.23}$$

where

$$F(k) = \left[A(k|k) \quad A(k+1|k)A(k|k) \quad \dots \quad \Lambda(k,0)\right]^T, \tag{8.24}$$

and

$$G(k) = \\ \begin{bmatrix} B(k|k) & 0 & \cdots & 0 \\ A(k+1|k)B(k|k) & B(k+1|k) & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \Lambda(k,1)B(k|k) & \Lambda(k,2)B(k+1|k) & \cdots & B(k+h-1|k) \end{bmatrix}, \tag{8.25}$$

where $F(k) \in \mathbb{R}^{n \cdot h} \times \mathbb{R}^n, \quad G(k) \in \mathbb{R}^{n \cdot h} \times \mathbb{R}^{m \cdot h}$.

A dynamics of reference trajectory tracking error can be defined as

$$e_r(k+1) = A_r^i e(k), \quad i = 1, \dots, h, \tag{8.26}$$

whereas matrix A_r is selected so that reference trajectory tracking error decreases with desired dynamics over time. Now the vector of reference trajectory tracking errors is defined as

$$E_r^*(k) = \left[e_r(k+1|k)^T \; e_r(k+2|k)^T \; \dots \; e_r(k+h|k)^T\right]^T, \tag{8.27}$$

and it can be computed as

$$E_r^*(k) = F_r e(k), \tag{8.28}$$

where

$$F_r = \left[A_r \quad A_r^2 \quad \dots \quad A_r^h\right]^T, \tag{8.29}$$

and $F_r \in \mathbb{R}^{n \cdot h} \times \mathbb{R}^n$.

The cost function (8.16) can now be expressed as

$$J(U_B) = (E_r^* - E^*)^T \overline{Q}(E_r^* - E^*) + U_B^T \overline{R} U_B. \tag{8.30}$$

In order to obtain optimal control law, the cost function is minimized by setting its derivative to zero. In this way the control law becomes

$$U_B(k) = \left(G^T \overline{Q} G + \overline{R}\right)^{-1} G^T \overline{Q}(F_r - F)e(k), \tag{8.31}$$

where

$$\overline{Q} = \begin{bmatrix} Q & 0 & \cdots & 0 \\ 0 & Q & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & Q \end{bmatrix}, \overline{R} = \begin{bmatrix} R & 0 & \cdots & 0 \\ 0 & R & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & R \end{bmatrix}, \tag{8.32}$$

where $\overline{Q} \in \mathbb{R}^{n \cdot h} \times \mathbb{R}^{n \cdot h}$ and $\overline{R} \in \mathbb{R}^{m \cdot h} \times \mathbb{R}^{m \cdot h}$. The feedback control law of the MPC now becomes

$$u_B(k) = K_{mpc}\, e(k) \tag{8.33}$$

where K_{mpc} is defined as first m rows of the matrix $\left(G^T \overline{Q} G + \overline{R}\right)^{-1} G^T \overline{Q}(F_r - F)$, so that $K_{mpc} \in \mathbb{R}^m \times \mathbb{R}^n$.

The main improvement of MPC over pure reactive liner-design and nonlinear controllers is its prediction of future robot states, which should, at least theoretically, improve quality of the control and reduce tracking error.

8.4 Experimental Results

8.4.1 Trajectory Tracking Controllers Comparison

Three described trajectory-tracking algorithms were tested on robot soccer platform described in Section 3.4. Robot state was measured using global vision system described in Chapter 3. For experiments 8-shaped trajectory was used that is defined by

$$\begin{aligned} x_r(t) &= 1.1 + 0.7 sin\left(\frac{v_{max} t}{\sqrt{2.45}}\right) \\ y_r(t) &= 0.9 + 0.7 sin\left(\frac{2 \cdot v_{max} t}{\sqrt{2.45}}\right), \end{aligned} \tag{8.34}$$

where v_{max} is the maximum longitudinal velocity of the trajectory, which is achieved in the center point of shape "8", with coordinates (1.2, 0.9) m. This type of trajectory is appropriate for experiments, because it has sharp turns, as well as parts with high acceleration and velocity, but it's still feasible for the robot because there are no step changes of the longitudinal, nor angular velocity, and maximum acceleration of the robot is never exceeded. The maximum longitudinal

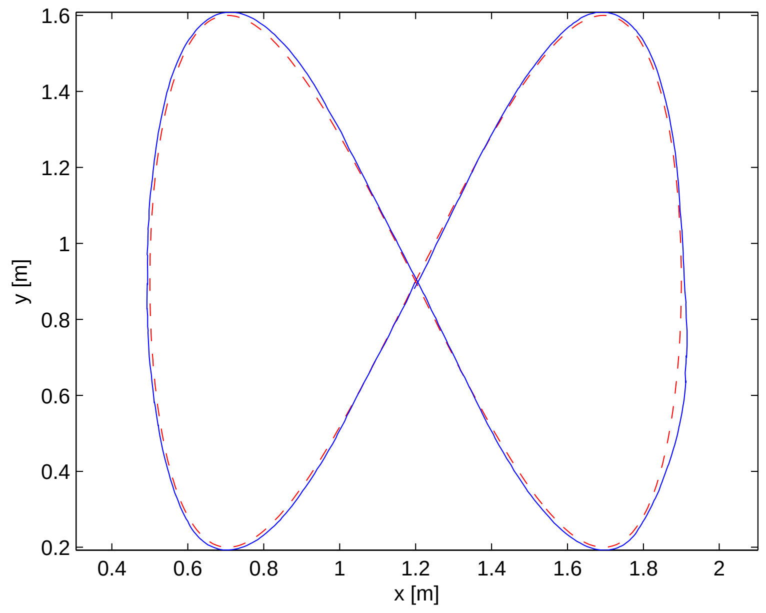


Figure 8.3. *Trajectory tracking experiment with linear-design controller: robot path (–), reference path (- -)*

velocity that was used in experiments is $v_{max} = 1.5$ m/s. With this setting, the maximum longitudinal acceleration of the trajectory is 1.9 m/s^2, which is close to robot acceleration limits (characteristics of the soccer robot can be seen in Table 7.1). In experiments, robot was first accelerated to achieve position, orientation and velocity that were approximately equal to trajectory initial condition.

The sampling period for all controllers was 12.5 ms. The parameters for linear-design controller were $\zeta = 0.7$ and $g = 60$, and the same parameters were used for nonlinear controller. For the MPC, after many trials the prediction horizon $h = 12$ was taken as the best choice. Also using trial and error, the reference matrix $A_r = I_{3\times3} \cdot 0.85$ and weighting matrices

$$Q = \begin{bmatrix} 4 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 0.1 \end{bmatrix}, \quad R = I_{2\times2} \times 10^{-3},$$

were selected.

The obtained robot paths for experiments with linear-design controller, nonlinear controller and MPC are given in Figures 8.3, 8.4 and 8.5, respectively. To compare tracking errors, for all experiments sum of squared errors (SSE) for each component of error vector e is computed and it is obtained:

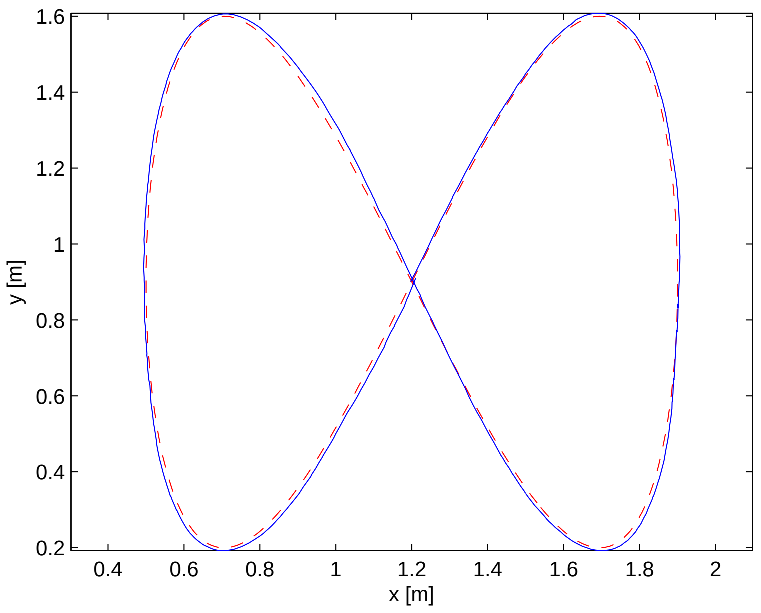


Figure 8.4. *Trajectory tracking experiment with nonlinear controller: robot path (–), reference path (- -)*

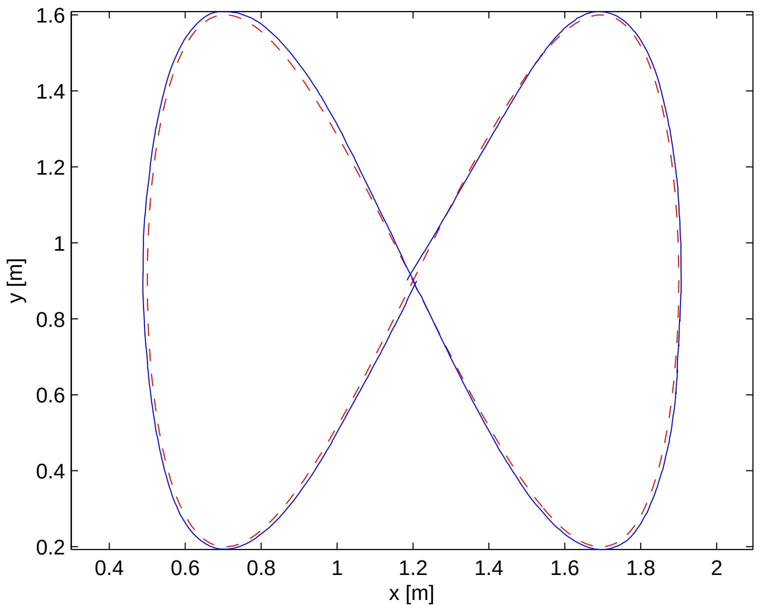


Figure 8.5. *Trajectory tracking experiment with MPC: robot path (–), reference path (- -)*

$$\begin{aligned}
\text{SSE}_{linear} &= (0.0177 \quad 0.0397 \quad 0.4395) \\
\text{SSE}_{nonlinear} &= (0.0144 \quad 0.0354 \quad 0.3555) \\
\text{SSE}_{MPC} &= (0.0442 \quad 0.0416 \quad 0.4010).
\end{aligned}$$

It can be seen that nonlinear controller has the lowest SSE for all three error components, although the difference is not significant. This can also be confirmed by visual comparison of obtained paths. The error of linear-design controller is only slightly higher compared to nonlinear. Although the simulation experiments performed prior to real-world experiments suggested superiority of MPC over other controllers, as its SSE was more than 10 times lower, surprisingly, in real world it could be no more confirmed—MPC resulted with about three times higher longitudinal error e_1 compared to nonlinear controller, while other two error components are comparable to other controllers.

The produced command velocities of linear-design, nonlinear and MPC controllers, together with ideal velocities generated from reference trajectory, are displayed in Figures 8.6, 8.7 and 8.8, respectively. The difference between generated command velocity and ideal velocity denotes how much feedback action a controller uses. It can be observed that all controllers produce similar command values, but MPC command values are smoothest and closest to ideal velocities. This is because it implicitly performs some filtering during prediction process. However, this makes the MPC more conservative—it reacts slower and is thus less robust and more prone to uncertainties. But this also suggests that MPC has the lowest control effort, i.e. it has the lowest energy consumption.

Regarding the choice of controller parameters, for linear-design and nonlinear controllers it was very easy to find controller parameters using trial and error. The control algorithm still performed well under moderate deviation of parameter values. On the contrary, MPC was very sensitive to choice of parameters and control sometimes became unstable under only slight modifications of parameter values.

The next important issue is sensitivity to initial tracking error. It is obtained that in presence of moderate initial position error (few centimeters) linear-design and nonlinear controllers show acceptable robustness, while MPC often demonstrates unstable behavior.

The experiment was also conducted to find out the maximum velocity at which the control remains stable. Here an additional problem appears, as measurements become more noisy at high velocities. The maximum longitudinal velocity was changed by tuning parameter v_{max} in equation (8.34). It was obtained that the maximum velocity that could be achieved is 1.5 m/s with MPC, 1.6 m/s with linear-design controller and 1.7 m/s with nonlinear controller. Thus, the nonlinear controller again demonstrated the best behavior.

All described controllers can be implemented in a microcontroller. Linear-design controller, being the simplest one of three controllers, is the easiest to

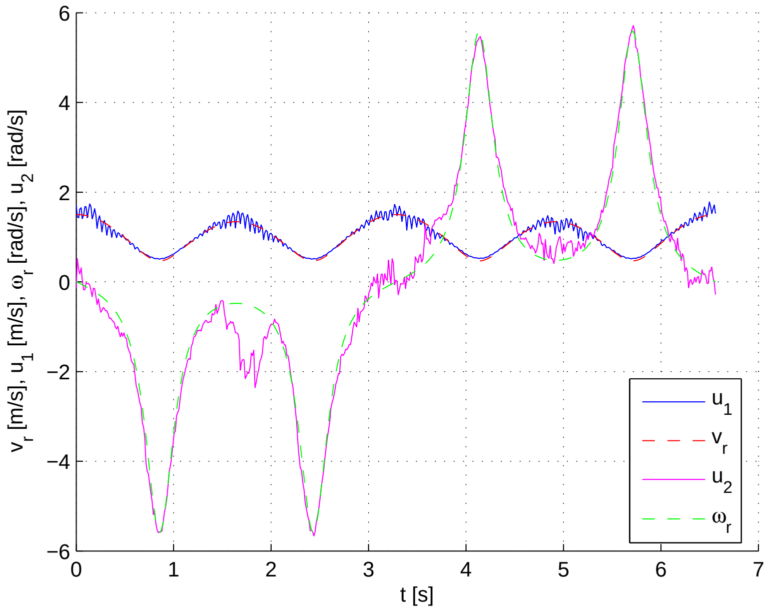


Figure 8.6. *Experiment with linear-design controller: ideal longitudinal (v_r) and angular (ω_r) velocity, generated command longitudinal (u_1) and angular (u_2) velocity*

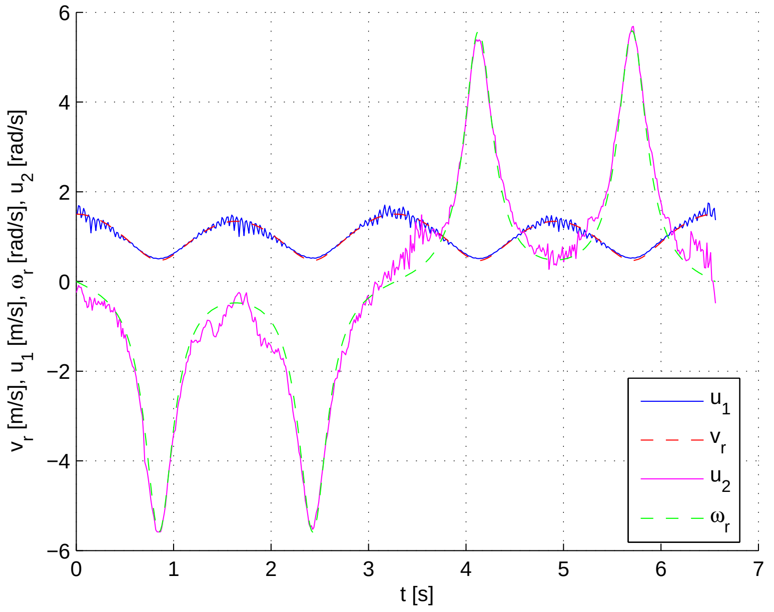


Figure 8.7. *Experiment with nonlinear controller: ideal longitudinal (v_r) and angular (ω_r) velocity, generated command longitudinal (u_1) and angular (u_2) velocity*

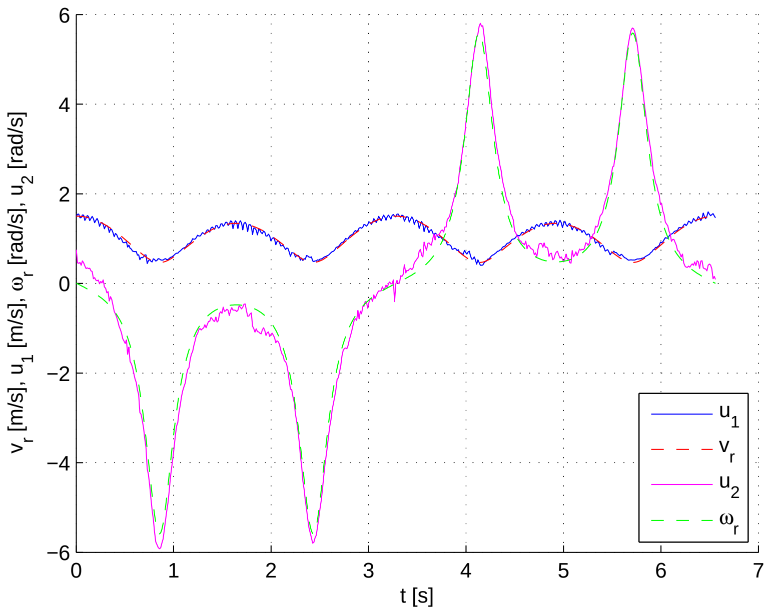
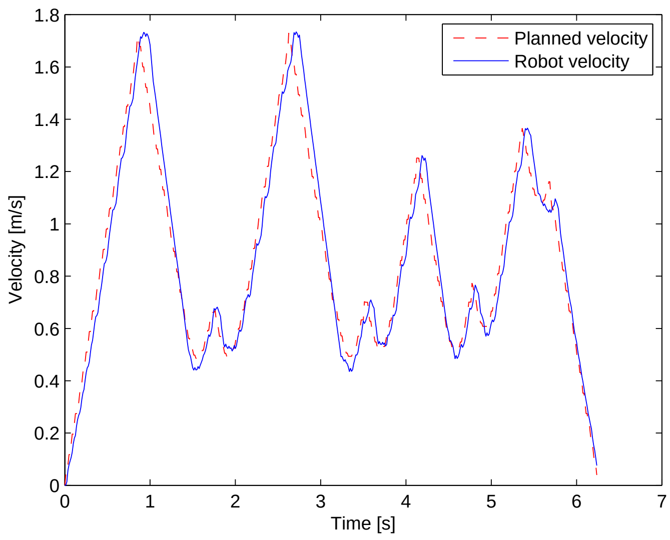


Figure 8.8. *Experiment with MPC: ideal longitudinal (v_r) and angular (ω_r) velocity, generated command longitudinal (u_1) and angular (u_2) velocity*

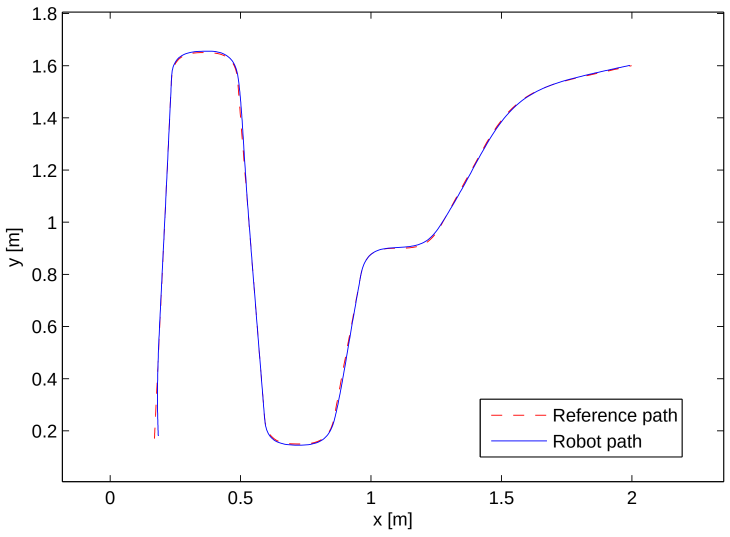
implement in a microcontroller. Nonlinear controller is only slightly more complicated, as its implementation involves computation of sinus function, which can be obtained through a lookup table. MPC is the most complicated, since its implementation requires matrix operations including matrix inversion. However, if the microcontroller is powerful enough, MPC can be still implemented, yet it is questionable whether it would execute fast enough for long prediction horizons.

Finally, it can be concluded that nonlinear controller achieves highest velocity and lowest tracking error, and it is easy to tune its parameters. For MPC it was noted that, although it is the best in simulations, in the real world it has the highest tracking error. The most serious problem of MPC is its weak robustness, as it is highly sensitive to initial position error and choice of parameters. This is most likely caused by the fact that the described MPC algorithm depends on linearized model for prediction of future errors. However, linearized model is inexact, especially for long prediction horizons, so that MPC makes wrong predictions. Moreover, it is difficult to determine its parameters. Its potential advantage is flexibility regarding tuning options and energy saving so that it could be used at low velocities in applications where energy saving is important. However, as in this work more emphasis is given to achieving fast robot motion, nonlinear controller is taken as a method of choice, since it demonstrated the best behavior so far.

8.4.2 Complete Motion Planner



(a)



(b)

Figure 8.9. *Motion planning experiment in simulator.* (a) *Robot velocity.* (b) *Robot path.*

Having tested all the particular modules, it is now necessary to check function of the complete motion planner. It consists of path planning module described in Chapter 5, path smoothing module described in Chapter 6, trajectory planning module described in Chapter 7 and trajectory tracking module based on nonlinear controller described in this chapter. In this experiment robot soccer simulation test bed is used, with some additional obstacles added to the robot soccer field, so that robot workspace was as shown in Figure 5.7. The planned path that was used in the experiment is shown in Figure 5.10 and is further smoothed by clothoids as shown in Figure 6.16.

The velocity profile obtained by applying the trajectory planning algorithm to the smoothed path is displayed in Figure 8.9 a), together with actual robot velocity obtained in high fidelity simulator by applying the nonlinear trajectory tracking controller. It can be observed that the actual robot velocity is very close to reference velocity although some delay is visible, which affects mainly longitudinal error. In Figure 8.9 b) robot path is displayed and it can be seen that robot tracks the planned path very closely, which implies that lateral error is low. It can be concluded that the developed motion planner shows good behavior. Experiments with real robots are planned in the future.

8.5 Summary

An experimental comparison of three trajectory tracking controllers for nonholonomic mobile robot is presented: linear-design controller, nonlinear controller and model predictive controller. The goal was to obtain tracking at highest possible trajectory velocity, and all controllers were tested on a real mobile robot. Experiments have shown that all three controllers produce comparable quality of trajectory tracking, but MPC is highly sensitive to initial robot state error, and it is very difficult to determine optimal parameters of the MPC algorithm. Moreover, the drawback of MPC is its complexity, especially if longer prediction intervals are used. Therefore, at high velocities the nonlinear controller is selected as it provides global asymptotic stability, it is easy to choose its parameters, is simple to implement and has low computational requirements, which makes it especially appropriate for fast, real-time trajectory tracking.

Finally, simulation experiment with the overall motion planner system is performed, which demonstrates that all modules work well when combined together.

Future work will include developing a nonlinear model predictive controller (e.g. [71]) that is expected to provide better prediction of future robot states, and therefore better robustness at high velocities. Also, the procedure of determining optimal controller parameters will be developed.

It should also be mentioned that in some applications velocity planning is not of particular importance for the motion planning, so that trajectory planning algorithm could be skipped and path-following controller can be applied directly to the planned path. A number of different path-following algorithms is described in the literature, see e.g. [111]. Investigation of path-following algorithms could be also an interesting topic of ongoing research.

Chapter 9

Conclusion

The thesis investigates application of mobile robots in environments that are provided with ambient intelligence to form the so-called Intelligent Spaces (iSpaces). The purpose of the iSpace is providing various services to its users, where mobile robots provide numerous possibilities, such as load delivery, visitor guidance etc. Thus the focus of the thesis is set to developing the capability of the space to fully utilize mobile robots, with the emphasis on development of low-level algorithms that can benefit by using intelligent space advantages. The problem of introducing mobile robots can be considered from two main aspects—sensing and planning. Consequently, a research is conducted in two directions. First, a method for fast and precise mobile robot localization using distributed cameras is developed. Second, a fast and flexible robot motion planning method appropriate for intelligent space application is developed.

The iSpace infrastructure presumes installation of distributed sensors into the space. One of the most versatile sensors to be installed is camera, where such systems are known as global vision systems. Important issues when using global vision for robot localization are measurement errors due to noise contained in the image and sensitivity to illumination changes. In this work a method for mobile robot localization using distributed cameras is developed that solves both issues. The method uses specially designed markers mounted on the top of the robot. The developed vision algorithm achieves high measurement precision and accuracy as it works in subpixel precision and is robust to light intensity changes. Moreover, the algorithm is efficient, as it operates directly in Bayer image, so that high framerates (> 100 fps) can be achieved with typical hardware. High measurement precision and robustness are verified by carefully performed experiments.

Regarding robot motion planning, benefits of iSpace can manifest in more accurate robot's state estimation, known environment and more computational power available. Therefore it is reasonable to expect that more deterministic motion planning can be achieved compared to typical reactive planners. One such approach that uses decoupled planning to obtain deterministic long-term planning is developed in this thesis. The method consists of four stages: path planning, path smoothing, trajectory planning and trajectory tracking. As the basis of path planning a modified visibility-Voronoi diagram for clearance is used that finds the shortest path amidst polygonal obstacles satisfying minimum clearance requirement where possible. The method is extended to handle path replanning in real time, which enables planning in dynamic environments. A path-smoothing algorithm capable of smoothing piecewise linear paths is developed, that produces G^2 continuous paths. Moreover, the curvature of the produced path changes linearly with distance, which is obtained by using clothoids as path-smoothing primitives. Efficient algorithms capable of smoothing a path with non-zero initial curvature are also developed, which represents an important contribution, essential for path replanning when robot initial velocity is non-zero. An optimal trajectory planning algorithm along the predefined path is developed that accounts for velocity and acceleration limits, as well as limited grip between the wheels and the ground, which is very important for reliable robot operation at high velocities. Finally, three trajectory tracking controllers for nonholonomic mobile robot are implemented and experimentally compared, namely linear design controller, nonlinear controller and model predictive controller. The overall system is successfully tested on high-fidelity robot soccer simulator.

9.1 Contributions

The contributions of the presented work can be summarized as follows:

1. A new global vision system for real-time tracking of two-dimensional poses of multiple mobile robots is presented. A novel algorithm is proposed that operates directly in Bayer format image thus enabling high framerates and at the same time high measurements precision and accuracy as it works in subpixel precision. High measurement precision and accuracy are verified by carefully performed experiments.

2. A path-planning algorithm is proposed that finds a shortest path amidst polygonal obstacles that satisfies minimum clearance requirement where possible. Hereby, a novel algorithm that has a capability of dynamic path replanning is developed, which enables planning in dynamic environments. The developed path-replanning algorithm is efficient and allows real-time path replanning in presence of moderate number of moving obstacles. The complexity of the algorithm does not depend on size of the workspace, but only on its complexity.

3. A path-smoothing algorithm capable of smoothing piecewise linear paths is proposed, which produces G^2 continuous path with linearly changing curvature. This is obtained by using clothoid curves as smoothing primitives.

Efficient algorithms capable of smoothing a path with non-zero initial curvature are developed, which is essential when path replanning for moving robots is required. To the best of author's knowledge, this is the first complete solution that enables application of clothoid for path replanning.

It is known that expressions for clothoids coordinates have no closed form solution, which makes calculations with clothoids hard to do in real time. This problem is solved by storing points of the clothoid in the lookup table. It is shown that points of any clothoid can be efficiently computed based on the stored clothoid by rescaling, rotating and translating.

4. A dynamic model of differential-drive robot is developed. The model accounts for acceleration limits, as well as limited grip between the wheels and the ground, which is very important for reliable robot operation at high velocities. The developed model is then used to express acceleration limits and velocity limit curve required by the optimal time-scaling algorithm. The developed algorithms are used for time-optimal trajectory planning along the predefined path for the soccer robot and Pioneer 3DX robot.

5. An experimental comparison of three trajectory tracking controllers for nonholonomic mobile robot is presented: linear-design controller, nonlinear controller and model predictive controller.

9.2 Future Work

A number of algorithms presented in this thesis could be further improved or extended. Some of the possible improvements are already mentioned in the main text, and here we bring a summary:

- Currently, every camera of the global vision system must be calibrated separately, which is a rather cumbersome task. However, the calibration can be automated by tracking the robots by the distributed cameras (and possibly other sensors) while at the same time estimating the calibration parameters, utilizing the fact that robot's state is continuously changing while moving from one to another camera's field of view.

  Furthermore, fusion of global vision with other sensors, both distributed and onboard, should be implemented to increase measurement reliability and avoid e.g. occlusion problems.

- The developed path planner is not appropriate in all situations (e.g. manipulation), so that it could be combined with other methods, such as sampling-based planner. In such multiple-planner strategy for each subproblem the most appropriate planning method would be picked by the higher-level module.

- Several improvements can be made in path-smoothing module, such as use of Bezier curves for clothoid interpolation.

- The current motion planner handles moving obstacles by performing path replanning, without any prediction of future obstacles' positions. As such, it is acceptable only for slow-moving obstacles. For faster obstacles velocity tuning, path adaptation, or both can be implemented.

- Currently the nonlinear trajectory tracking controller is used for trajectory tracking. It should be investigated whether other algorithms, such as the model predictive controller based on nonlinear model, could achieve better performance and robustness at high velocities.

This thesis covers an application of mobile robots in iSpaces, which is only a small segment of an overall iSpace problematic. Thus, there is yet a long way to achieving a fully operational system, where to the author's opinion at least the following issues must be addressed:

- Robust human tracking;
- User friendly human-iSpace interface (speech recognition, etc.);
- To allow manipulation tasks, mobile robot should be equipped with a grasping manipulator.

Bibliography

[1] The PolyBoolean library. `http://www.complex-a5.ru/polyboolean/`.

[2] ActivMedia Robotics. *Pioneer 2 Mobile Robots with Pioneer 2 Operating System Servers – Operations Manual*, 2000.

[3] J. E. Adams, K. Parluski, and K. Spaulding. Color processing in digital cameras. *IEEE Micro*, 18(6):20–29, 1998.

[4] Šandor Ileš, M. Seder, and I. Petrović. Improvement of map building during the exploration of polygonal environments using the range data. In *Proc. of the 15th International Conference on Electrical Drives and Power Electronics (EDPE2009)*, Dubrovnik, Croatia, 2009.

[5] G. Antonelli, S. Chiaverini, and G. Fusco. A calibration method for odometry of mobile robots based on the least-squares technique: Theory and experimental validation. *IEEE Transactions on Robotics*, 21(5):994–1004, October 2005.

[6] G. Appenzeller, J.-H. Lee, and H. Hashimoto. Building topological maps by looking at people: An example of cooperation between intelligent spaces and robots. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS'97)*, Grenoble, France, 1997.

[7] K. O. Arras, N. Tomatis, B. T. Jensen, and R. Siegwart. Multisensor on-the-fly localization: Precision and reliability for applications. *Robotics and Autonomous Systems*, 34(2-3):131–143, 2001.

[8] T. Asano, T. Asano, L. Guibas, J. Hershberger, and H. Imai. Visibility of disjoint polygons. *Algorithmica*, 1(1):49–63, 1986.

[9] F. Aurenhammer and R. Klein. Voronoi diagrams. Technical report, FernUniversität Hagen, Department of Computer Science, Germany, 1996. Technical Report 198.

[10] F. Avnaim, J.-D. Boissonnat, and B. Faverjon. A practical exact motion planning algorithm for polygonal objects amidst polygonal obstacles. In *Proceedings of IEEE Int. Conf. on Robotics and Automation*, volume 3, 1988.

[11] D. Ball, G. Wyeth, and S. Nuske. A global vision system for a robot soccer team. In *Proceedings of the Australasian Conference on Robotics and Automation (ACRA'04)*, Canberra, Australia, 2004.

[12] B. E. Bayer. Color imaging array. U.S. Patent No. 3,971,065, 1976.

[13] K. Belghith, F. Kabanza, L. Hartman, and R. Nkambou. Anytime dynamic path-planning with flexible probabilistic roadmaps. In *Proceedings of IEEE Int. Conference on Robotics and Automation*, pages 2372–2377, 2006.

[14] M. Bennewitz, W. Burgard, and S. Thrun. Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots. *Robotics and Autonomous Systems*, 41:89–99, 2002.

[15] P. Bhattacharya and M. L. Gavrilova. Roadmap-based path planning - using the Voronoi diagram for a clearance-based shortest path. *Robotics and Automation Magazine, IEEE*, 15(2):58–66, June 2008.

[16] J. E. Bobrow, S. Dubowsky, and J. S. Gibson. Time-optimal control of robotic manipulators along specified paths. *International Journal of Robotics Research*, 4(4):3–17, 1985.

[17] J. Borenstein, H. R. Everett, and L. Feng. Where am I? sensors and methods for mobile robot positioning. Technical Report 1996.MI 48109, University of Michigan, 1996.

[18] J. Borenstein and Y. Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Trans. on Systems, Man, and Cybernetics*, 19(5):1179—1187, 1998.

[19] J.-Y. Bouguet. *Camera Calibration Toolbox for Matlab.*

[20] M. Brezak, G. Klančar, I. Petrović, and D. Matko. Supervised tracking of a mobile robot team. In *Proceedings of International Electrotechnical and Computer Science Conference (ERK'05)*, Portorož, Slovenia, 2005.

[21] M. Brezak and I. Petrović. Global vision based tracking of multiple mobile robots in subpixel precision. In *Proc. of International Conference on Industrial Informatics (INDIN 2007)*, 2007.

[22] M. Brezak, I. Petrović, and N. Perić. Experimental comparison of trajectory tracking algorithms for nonholonomic mobile robots. In *Proc. of 35th Annual Conference of the IEEE Industrial Electronics Society*, Porto, Portugal, 2009.

[23] M. Brezak, I. Petrović, and K. Vrdoljak. A trajectory tracking controller for mobile robot with differential drive. In *Proc. of International Conference on Electrical Drives and Power Electronics*, Slovakia, 2003.

[24] M. Brezak, I. Petrović, and E. Ivanjko. Robust and accurate global vision system for real time tracking of multiple mobile robots. *Robotics and Autonomous Systems*, 56:213–230, March 2008.

[25] M. Brezak, I. Petrović, and A. Kitanov. An approach to motion planning of mobile robots considering dynamic constraints. In *Proceedings of International Conference on Power Electronics and Motion Control (EPE-PEMC 2004)*, Riga, Latvia, 2004.

[26] M. Brezak, I. Petrović, and D. Rožman. Global vision based tracking of multiple mobile robots. In *Proceedings of International Symposium on Industrial Electronics (ISIE'06)*, Montreal, Canada, 2006.

[27] D. Brščić. *Mobile robot control scheme based on distributed and onboard sensors*. PhD thesis, University of Tokyo, Intelligent Control System Laboratory, 2008.

[28] B. B. Brian, B. Meyers, J. Krumm, A. Kern, and S. Shafer. Easyliving: technologies for intelligent environments. In *Proceedings of the 2nd international symposium on Handheld and Ubiquitous Computing*, pages 12–29, Bristol, UK, 2000. Springer-Verlag.

[29] B. Browning and M. Veloso. Real-time, adaptive color-based robot vision. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS'05)*, Edmonton, Canada, 2005.

[30] J. Bruce, T. Balch, and M. Veloso. Fast and inexpensive color image segmentation for interactive robots. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS'00)*, Takamatsu, Japan, 2000.

[31] J. Bruce and M. Veloso. Fast and accurate vision-based pattern detection and identification. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation (ICRA'03)*, Taipei, Taiwan, 2003.

[32] M. N. Bygi and M. Ghodsi. Improving the construction of the visibility–Voronoi diagram. In *Proc. of European Workshop on Computational Geometry*, Graz, Austria, 2007.

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

[34] J. F. Canny. Constructing roadmaps of semi-algebraic sets I: Completeness. *Artificial Intelligence*, 37:203–222, 1988.

[35] CGAL*, Computational Geometry Algorithms Library.* http://www.cgal.org.

[36] CGAL Editorial Board. CGAL *User and Reference Manual*, 3.5 edition, 2009.

[37] F. Chin, J. Snoeyink, and C. A. Wang. Finding the medial axis of a simple polygon in linear time. *Discrete Computational Geometry*, pages 382–391, 1995.

[38] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion.* MIT Press, 2005.

[39] F. Daoud and T. Nomura. Preface to 'smart spaces'. *Journal of Network and Computer Applications*, 25(4):239–242, 2002.

[40] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry, Algorithms and Applications.* Springer, third edition, 2008.

[41] H. Delingette, M. Herbert, and K. Ikeuchi. Trajectory generation with curvature constraint based on energy minimization. In *Proceedings of Int. Conference on Intelligent Robots and Systems (IROS '91)*, Osaka, Japan, November 1991.

[42] A. DeLuca and M. DiBenedetto. Control of nonholonomic systems via dynamic compensation. *Kybernetika*, 29(6):593–608, 1993.

[43] F. Devernay. A non-maxima suppression method for edge detection with sub-pixel accuracy. Technical Report RR-2724, INRIA Institute, 1995.

[44] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[45] A. Egorova, M. Simon, F. Wiesel, A. Gloye, and R. Rojas. Plug and play: Fast automatic geometry and color calibration for cameras tracking robots. In *Proceedings of the 8th International RoboCup Symposium (RoboCup'04)*, Lisboa, Portugal, 2004.

[46] A. Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, 3:249–265, June 1987.

[47] Željko Srbljinović. Dinamički modeli mobilnih robota s diferencijalnim pogonom i njihovo parametriranje. University of Zagreb, Faculty of Electrical Engineering and Computing, October 2008. Diploma thesis.

[48] S. Fleury, P. Soueres, J.-P. Laumond, and R. Chatila. Primitives for smoothing mobile robot trajectories. *IEEE Transactions on Robotics and Automation,*, 11(3):441–448, June 1995.

[49] M. L. for Computer Science and M. A. Intelligence. MIT Project Oxygen. `http://www.oxygen.lcs.mit.edu/`.

[50] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1), 1997.

[51] T. Gevers and A. W. M. Smeulders. Color-based object recognition. *Pattern Recognition*, 32(3):453–464, April 1999.

[52] S. K. Ghosh. *Visibility Algorithms in the Plane*. Cambridge, 2007.

[53] C. Gönner, M. Rous, and K.-F. Kraiss. Real-time adaptive colour segmentation for the robocup middle size league. In *Proceedings of RoboCup International Symposium 2004*, Lisbon, Portugal, 2004.

[54] K. Gunnarsson, F. Wiesel, and R. Rojas. The color and the shape: Automatic on-line color calibration for autonomous robots. In *Proceedings of The 9th International RoboCup Symposium (RoboCup'05)*, Osaka, Japan, 2005.

[55] P. Hart, N. Nilsson, and B. Rafael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4:100–107, 1968.

[56] H. Hashimoto. Present state and future of intelligent space—discussion on the implementation of RT in our environment. *Artificial Life and Robotics*, 11(1):1–7, 2007.

[57] M. A. Heald. Rational approximations for the fresnel integrals. *Math. Comp*, 44:459–461, 1985.

[58] J. Hershberger and S. Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM Journal on Computing*, 28:2215–2256, 1999.

[59] J. Hightower and G. Borriella. Location systems for ubiquitous computing. *IEEE Computer*, 34(8):57–66, 2001.

[60] W. Hu, T. Tan, L. Wang, and S. Maybank. A survey on visual surveillance of object motion and behaviors. *IEEE Transactions on Systems, Man and Cybernetics*, 34:334–352, 2004.

[61] E. Ivanjko. Umjeravanje dinamike mobilnog robota. Technical report, University of Zagreb, Faculty of Electrical Engineering and Computing, Croatia, 2008.

[62] E. Ivanjko. *Autonomna navigacija mobilnih robota zasnovana na ultrazvučnim senzorima udaljenosti*. PhD thesis, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, Croatia, January 2009. (In Croatian).

[63] E. Ivanjko, M. Brezak, and I. Petrović. Eksperimentalni postav za automatsko umjeravanje odometrijskog sustava mobilnih robota. In *Proceedings of MIPRO*, Opatija, Croatia, 2008. (In Croatian).

[64] E. Ivanjko, A. Kitanov, and I. Petrović. *Robot Localization*, chapter Model based Kalman Filter Mobile Robot Self-Localization. I-Tech, Vienna, 2009.

[65] T. Jin, K. Morioka, and H. Hashimoto. Human-following robot using the particle filter in ispace with distributed vision sensors. *Artificial Life and Robotics*, 10(2):96–101, 2006.

[66] B. Johanson, A. Fox, , and T. Winograd. The interactive workspaces project: Experiences with ubiquitous computing rooms. *IEEE Pervasive Computing Magazine*, 1(2):67–74, 2002.

[67] S. Jurić-Kavelj, M. Seder, and I. Petrović. Tracking multiple moving objects using adaptive sample-based joint probabilistic data association filter. In *Proc. of the Fifth International Conference on Computational Intelligence, Robotics and Autonomous Systems (CIRAS 2008)*, pages 93–98, Linz, Austria, 2008.

[68] I. Kamon, E. Rivlin, and E. Rimon. Range-sensor based navigation in three dimensions. In *Proceedings of IEEE International Conference on Robotics and Automation*, 1999.

[69] Y. Kanayama and B. Hartman. Smooth local planning for autonomous vehicles. In *Proceedings of IEEE Int. Con. on Robotics and Automation (ICRA '89)*, 1989.

[70] Y. Kanayama and N. Miyake. Trajectory generation for mobile robots. In M. Press, editor, *Proceedings of the International Symposium on Robotics Research*, pages 16–23, 1985.

[71] K. Kanjanawanishkul, M. Hofmeister, and A. Zell. Smooth reference tracking of a mobile robot using nonlinear model predictive control. In *Proceedings of European Conference on Mobile Robots*, Mlini/Dubrovnik, Croatia, 2009.

[72] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transaction on Robotics and Automation*, 12(4):556–580, June 1996.

[73] K. Kedem and M. Sharir. An efficient motion planning algorithm for a convex rigid polygonal object in 2-dimensional polygonal space. *Discrete and Computational Geometry*, 5(1):43–75, 1990.

[74] C. Kee, H. Jun, and D. Yun. Indoor navigation system using asynchronous pseudolites. *The Journal of Navigation*, 56(3):443–455, 2003.

[75] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1):90–98, 1986.

[76] B. Kim, N. Tomokuni, K. Ohara, T. Tanikawa, K. Ohba, and S. Hirai. Ubiquitous localization and mapping for robots with ambient intelligence. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '01)*, Beijing, China, 2006.

[77] R. Kimmel. Demosaicking: Image reconstruction from color CCD samples. *IEEE Transaction on Image Processing*, 7(3):1221–1228, 1999.

[78] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. RoboCup: The robot world cup initiative. In *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, Marina del Rey, California, USA, 1997.

[79] A. Kitanov and I. Petrović. Exactly sparse delayed state filter based robust slam with stereo vision. Submitted to the 41st International Symposium on Robotics – ISR2010, Munich, 2010.

[80] A. Kitanov, V. Tubin, and I. Petrović. Extending functionality of RF ultrasound positioning system with dead-reckoning to accurately determine mobile robot's orientation. In *Proc. of the 3rd IEEE Multi-conference on Systems and Control*, pages 1152–1157, Saint Petersburg, Russia, 2009.

[81] J. Kitzinger. The visibility graph among polygonal obstacles: a comparison of algorithms. Master's thesis, University of New Mexico, 2003.

[82] G. Klančar, M. Brezak, I. Petrović, and D. Matko. Two approaches to mobile robots simulator design. In *Proceedings of the 6th EUROSIM congress on Modelling and Simulation*, 2007.

[83] G. Klančar and I. Škrjanc. Tracking-error model-based predictive control for mobile robots in real time. *Robotics and Autonomous Systems*, 55:460–469, 2007.

[84] G. Klančar, M. Brezak, D. Matko, and I. Petrović. Mobile robot tracking using computer vision. *Automatika*, 46(3-4):155–163, 2005.

[85] G. Klančar, M. Kristan, S. Kovačič, and O. Orqueda. Robust and efficient vision system for group of cooperating mobile robots with application to soccer robots. *ISA Transactions*, 43:329–342, 2004.

[86] G. Klančar, M. Kristijan, and R. Karba. Wide-angle camera distortions and non-uniform illumination in mobile robot tracking. *Robotics and Autonomous Systems*, 46:125–133, 2004.

[87] L. Korkut, D. Žubrinić, and V. Županović. Box dimension and Minkowski content of the clothoid. *Fractals*, 17:485–492, 2009.

[88] J.-C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Boston, MA, USA, 1991.

[89] S. M. LaValle. *Planning algorithms.* Cambridge University Press, 2006.

[90] D. T. Lee. Proximity and reachability in the plane. Technical report, Dept. Elect. Engineering, Univ. Illinois, Urbana, IL, 1978.

[91] J.-H. Lee and H. Hashimoto. Intelligent space — concept and contents. *Advanced Robotics*, 16(3):265–280, 2002.

[92] M. V. Leonov and A. G. Nikitin. A closed set of algorithms for performing set operations on polygonal regions in the plane. Technical report, A. P. Ershov Institute of Informatics Systems, 1997.

[93] A. D. Luca, G. Oriolo, and M. Vendittelli. *RAMSETE. Articulated and Mobile Robotics for Services and Technologies*, volume 270, chapter Control of wheeled mobile robots: An experimental overview, pages 181–226. Springer-Verlag, London, third edition, 2001. Lecture Notes in Control and Information Sciences.

[94] M. Lucente, G.-J. Zwart, and A. George. Visualization space: a testbed for deviceless multimodal user interface. In *Proceedings of AAAI Intelligent Environments Symposium*, pages 87–92, California, USA, 1998.

[95] V. J. Lumelsky and A. A. Stepanov. Dynamic path planning for a mobile automaton with limited information on the environment. *IEEE Trans. on Automatic Control*, 31(11):1058–1063, November 1986.

[96] K. Maček. Motion planning of mobile robots in indoor environemnts. Master's thesis, Faculty of Electrical Engineering and Computing, Zagreb, October 2004.

[97] K. Maček, I. Petrović, and R. Siegwart. A control method for stable and smooth path following of mobile robots. In *Proc. of the 2nd European Conference on Mobile Robots*, pages 128–133., 2005.

[98] R. Mahkovic. Improved use of the tactile maps for visually impaired people. In *Proceedings of 16th International Workshop on Robotics in Alpe-Adria-Danube Region (RAAD 2007)*, Ljubljana, Slovenia, 2007.

[99] D. Marinakis, D. Meger, I. Rekleitis, and G. Dudek. Hybrid inference for sensor network localization using a mobile robot. In *Proc. of the 22nd AAAI Conference on Artificial Intelligence (AAAI '07)*, pages 1089–1094, Vancouver, Canada, July 2007.

[100] I. Marković and I. Petrović. Speaker localization and tracking in mobile robot environment using a microphone array. In *Proc. of of 40th International Symposium on Robotics*, pages 283–288, Barcelona, Spain, 2009.

[101] C. Mauri, T. Granollers, J. Lorés, and M. García. Computer vision interaction for people with severe movement restrictions. *Human Technology*, 2(1), April 2006.

[102] M. McAllister, D. Kirkpatrick, and J. Snoeyink. A compact piecewise-linear Voronoi diagram for convex sites in the plane. *Discrete Computational Geometry*, 1996.

[103] J. McCrae and K. Singh. Sketching piecewise clothoid curves. *Computers & Graphics*, 33:452–461, August 2009.

[104] M. McNaughton and H. Zhang. Color vision for robocup with fast lookup tables. In *Proceedings of IEEE International Conference on Robotics, Intelligent Systems and Signal Processing (RISSP 2003)*, Changsha, Hunan, China, 2003.

[105] D. S. Meek and D. J. Walton. An arc spline approximation to a clothoid. *Journal of Computational and Applied Mathematics*, 170(1):59–77, September 2004.

[106] D. S. Meek and D. J. Walton. A note on finding clothoids. *Journal of Computational and Applied Mathematics*, 170(2):433–453, 2004.

[107] K. R. Meidenbauer. An investigation of the clothoid steering model for autonomous vehicles. Master's thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, July 2007.

[108] E. Menegatti, G. Gatto, E. Pagello, T. Minato, and H. Ishiguro. Distributed vision system for robot localisation in indoor environment. In *Proceedings*

of the 2nd European Conference on Mobile Robots (ECMR'05), Ancona, Italy, 2005.

[109] I. T. Miletić. Sustav za praćenje ljudi u stvarnome vremenu pomoću digitalne kamere. University of Zagreb, Faculty of Electrical Engineering and Computing, 2008. Diploma thesis.

[110] N. Montés, A. Herraez, L. Armesto, and J. Tornero. Real-time clothoid approximation by rational bezier curves. In *Proceedings of International Conference on Robotics and Automation*, pages 2246–2251, 2008.

[111] P. Morin and C. Samson. *Motion control of wheeled mobile robots*, chapter 34, pages 799–826. Springer Berlin Heidelberg, 2008.

[112] H. N. T. Naniwa and S. Arimoto. A quadtree-based path-planning algorithm for a mobile robot. *Robotic Systems*, 1990.

[113] N. J. Nilsson. A mobile automaton: An application of artificial intelligence techniques. In *Proceedings of 1st International Conference on Artificial Intelligence*, Washington D.C., USA, 1969.

[114] A. Nishitani, Y. Nishida, T. Hori, , and H. Mizoguchi. Portable 3D ultrasonic tag system based on a quick calibration method. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pages 1561—1568, The Hague, Netherlands, October 2004.

[115] J. E. Normey-Rico, J. Gomez-Ortega, and E. F. Camacho. A smith-predictor-based generalised predictive controller for mobile robot path-tracking. *Control Engineering Practice*, 7(6):729–740, 1999.

[116] K. J. Obermeyer. The VisiLibity library. `http://www.VisiLibity.org`, 2008. R-1.

[117] A. Ollero and O. Amidi. Predictive path tracking of mobile robots. In *Proceedings of 5th International Conference on Advanced Robotics, Robots in Unstructured Environments (ICAR '91)*, volume 2, June 1991.

[118] G. Oriolo, A. D. Luca, and M. Vendittelli. WMR control via dynamic feedback linearization: Design, implementation, and experimental validation. *IEEE Tranasactions on Control Systems Technology*, 10(6):835–852, 2002.

[119] J. O'Rourke. *Computational Geometry In C*. Cambridge University Press, second edition, 1998.

[120] T. Petrinić. Dinamički model – Pioneer 3DX. University of Zagreb, Faculty of Electrical Engineering and Computing, May 2009. Seminar work.

[121] L. Pontryagin, V. Boltyansky, R. Gamkrelidze, and E. Mischenko. *The Mathematical Theory of Optimal Processes*. Wiley, New York, 1962.

[122] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C — The Art of Scientific Computing*. Cambridge University Press, second edition, 1992.

[123] S. Quinlan and O. Khatib. Elastic bands: Connecting path planning and control. In *Proceedings of int. Conference on Robotics and Automation*, pages 802–807, Atlanta, Georgia, USA, 1993.

[124] F. M. Raimondi and M. Melluso. A new fuzzy robust dynamic controller for autonomous vehicles with nonholonomic constraints. *Robotics and Autonomous Systems*, 52:115–131, 2005.

[125] R. Ramanath, W. E. Snyder, G. L. Bilbro, and W. A. S. III. Demosaicking methods for bayer color arrays. *Journal of Electronic Imaging*, 11:306–315, 2002.

[126] C. Samson. Time-varying feedback stabilization of car-like wheeled mobile robots. *International Journal of Robotics Research*, 12(1):55–64, 1993.

[127] C. Samson and H. Ait-Abderrahim. Feedback control of a nonholonomic wheeled cart in cartesian space. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1136–1141, Sacramento, CA, 1991.

[128] J. Sanchez-Reyes and J. M. Chacon. Polynomial approximation to clothoids via s-power series. *Computer-Aided Design*, 35:1305–1313, 2003.

[129] T. Sasaki, K. Morioka, P. T. Szemes, D. Brščić, and H. Hashimoto. Moving object tracking and an application in intelligent space. In *International Conference on Instrumentation, Control and Information Technology (SICE Annual Conference 2005)*, pages 2326–2329, Okayama, Japan, 2005.

[130] A. Scheuer and T. Fraichard. Collision-free and continuous-curvature path planning for car-like robots. In *Proceedings of International Conference on Robotics and Automation (ICRA '97)*, Albuquerque, U.S.A., April 1997.

[131] M. Seder, P. Mostarac, and I. Petrović. Hierarchical path planning of mobile robots in complex indoor environments. *Transactions of the Institute of Measurement and Control*, 2009. Accepted for publication.

[132] M. Seder and I. Petrović. Dynamic window based approach to mobile robot motion control in the presence of moving obstacles. In *Proc. of the IEEE International Conference on Robotics and Automation*, pages 1986–1991, Roma, Italy, 2007.

[133] M. Seder and I. Petrović. Integration of Focused D* and Witkowski's algorithm for path planning and replanning. In I. Petrović, editor, *Proceedings of the 4th European Conference on Mobile Robots*, pages 99–104, Mlini, Croatia, September 2009. KoREMA.

[134] A. Segovia, M. Rombaut, A. Preciado, and D. Meizel. Comparative study of the different methods of path generation for a mobile robot in a free environment. In *Proceedings of International Conference on Advanced Robotics*, Pisa, Italy, June 1991.

[135] D. H. Shin and S. Singh. Path generation for robot vehicles using composite clothoid segments. Technical Report CMU-RI-TR-90-31, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA, December 1990.

[136] K. G. Shin and N. D. McKay. Minimum-time control of robot manipulators with geometric path constraints. *IEEE Transactions on Automatic Control*, 30(6):531–541, 1985.

[137] M. Shiomi, T. Kanda, H. Ishiguro, and N. Hagita. Interactive humanoid robots for a science museum. *IEEE Intelligent Systems*, 22(2):25–32, 2007.

[138] R. Siegwart and I. R. Nourbakhsh. *Introduction to Autonomous Mobile Robots.* MIT Press, 2004.

[139] M. Simon, S. Behnke, and R. Rojas. Robust real time color tracking. In *Proceedings of The 4th International RoboCup Symposium (RoboCup'00)*, Melbourne, Australia, 2000.

[140] R. Smith. Open Dynamics Engine (ODE) library. `http://www.ode.org/`.

[141] M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis, and Machine Vision.* Brooks/Cole Publishing Company, second edition, 1999.

[142] P. Soueres and J. D. Boissonnat. *Robot Motion Planning and Control*, chapter Optimal Trajectories for Nonholonomic Mobile Robots. Springer, 1998.

[143] M. Sridharan and P. Stone. Structure-based color learning on a mobile robot under changing illumination. *Autonomous Robots*, 2007.

[144] C. Steger. Evaluation of subpixel line and edge detection precision and accuracy. *International Archives of Photogrammetry and Remote Sensing*, XXXII, Part 3/1:256–264, 1998.

[145] Sun Microsystems. What every computer scientist should know about floating-point arithmetic, 1992.

[146] R. Swaminathan and S. Nayar. Polycameras: Camera clusters for wide angle imaging. Technical report, Columbia University, Computer Science, 1999.

[147] K. Toyama, J. Krumm, B. Brummit, and B. Meyers. Wallflower: Principles and practice of background maintenance. In *Proceedings of the International Conference on Computer Vision (ICCV'99)*, Corfu, Greece, 1999.

[148] N. Ukita and T. Matsuyama. Real-time cooperative multi-target tracking by communicating active vision agents. *Computer Vision and Image Understanding*, 97:137–179, February 2005.

[149] J. van den Berg. *Path Planning in Dynamic Environments.* PhD thesis, Utrecht University, 2007.

[150] G. Vegter. *Algorithms and Data Structures*, volume 519/1991 of *Lecture Notes in Computer Science*, chapter Dynamically maintaining the visibility graph, pages 425–436. Springer Berlin / Heidelberg, 1991.

[151] D. J. Walton and D. S. Meek. A controlled clothoid spline. *Computers & Graphics*, 29(3):353–363, 2005.

[152] L. Wang, K. Miura, E. Nakamae, T. Yamamoto, and T. Wang. An approximation approach of the clothoid curve defined in the interval $[0, \pi/2]$ and its offset by free-form curves. *Computer-Aided Design*, 33:1049–1058, 2001.

[153] Z. Wang and D. Garlan. Task-driven computing. Technical Report CMU-CS-00-154, School of Computer Science, Carnegie Mellon University, 2000.

[154] R. Wein, J. P. van den Berg, and D. Halperin. The visibility–Voronoi complex and its applications. *Computational Geometry: Theory and Applications*, 36(1):66–78, 2007.

[155] M. Weiser. The computer for the twenty-first century. *Sci. Am.*, 265(3):94–104, 1991.

[156] E. Welzl. Constructing the visibility graph for n line segments in $O(n^2)$ time. *Information Processing Letters*, 20:167–171, May 1985.

[157] Wikipedia the free encyclopedia. Radio-frequency identification. `http://en.wikipedia.org/wiki/Rfid`. Accessed 2009-12-08.

[158] Wikipedia the free encyclopedia. Time-of-flight camera. `http://en.wikipedia.org/wiki/Time-of-flight_camera`. Accessed 2009-12-07.

[159] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.

Abstract

Localization, Motion Planning and Control of Mobile Robots in Intelligent Spaces

This thesis investigates application of mobile robots in environments provided with ambient intelligence to form the so-called Intelligent Spaces (iSpaces). The purpose of the iSpace is providing various services to its users, where mobile robots provide numerous possibilities, such as load delivery, visitor guidance etc. Thus the focus of the thesis is set to developing the capability of the space to fully utilize mobile robots, with the emphasis on development of low-level algorithms that can benefit by using intelligent space advantages, such as precise localization and mapping and increased computational power. In this way a method for fast and precise mobile robot localization using distributed cameras is developed. The developed algorithm enables high framerates and at the same time high measurement precision and accuracy as it works in subpixel precision. The algorithm was tested in multiple experiments and performed with very good results. Second, a fast and flexible robot mobile robot motion planning method appropriate for application in iSpaces is developed. The developed motion planner is decoupled and consists of four stages: path planning, path smoothing, trajectory planning and trajectory tracking. The method can replan the path in real time, which enables planning in dynamic environments.

Keywords: Intelligent Spaces, mobile robotics, global vision, motion planning, trajectory tracking

Sažetak

Lokalizacija, planiranje gibanja i upravljanje mobilnim robotima u inteligentnim prostorima

Disertacija obrađuje primjenu mobilnih robota u prostorima opremljenim prostornom inteligencijom, tvoreći na taj način tzv. inteligentne prostore (iProstore). Svrha iProstora je pružanje usluga korisnicima prostora, gdje mobilni roboti pružaju brojne mogućnosti, primjerice dostavu, vođenje posjetitelja itd. Stoga je težište disertacije stavljeno na razvoj mogućnosti prostora da u potpunosti iskoristi mobilne robote, s naglaskom na razvoj algoritama niže razine koji mogu iskoristiti prednosti inteligentnih prostora, kao što su precizna lokalizacija i kartiranje te povećana računalna snaga. U prvom redu, razvijena je brza i precizna metoda lokalizacije korištenjem sustava raspodijeljenih kamera (tzv. sustav globalnog vida). Razvijeni algoritam omogućava obradu velikog broja slika u sekundi, i istodobno visoku preciznost i točnost mjerenja, budući da radi u području potpiksela. Algoritam je ispitan kroz više eksperimenata koji su pokazali vrlo dobre rezultate. Zatim je razvijena brza i fleksibilna metoda planiranja gibanja mobilnih robota pogodna za inteligentne prostore. Razvijeni planer gibanja je raspodijeljen i sastoji se od četiri faze: planiranje putanje, izglađivanje putanje, planiranje trajektorije i slijeđenje trajektorije. Metoda omogućava replaniranje putanje u stvarnom vremenu, čime je omogućeno planiranje u dinamičkim prostorima.

Ključne riječi: Inteligentni Prostori, mobilna robotika, globalni vid, planiranje gibanja, praćenje trajektorije

Curriculum Vitae

I was born on the 25th July, 1978 in Zagreb, Croatia. After the primary school in Pregrada, I finished the math-oriented high school in 1996 in Krapina. The same year I started the undergraduate studies at the Faculty of Electrical Engineering and Computing of the University of Zagreb (FER), Croatia. In 1999 the Faculty council appointed me to finish the undergraduate studies with emphasis on scientific research. I graduated in September 2001. Since November 2001 I have been a student of graduate scientific studies at FER and have worked at the Department of Control and Computer Engineering, FER. In 2005 I started my PhD studies, and I passed the qualifying doctoral exam and the public talk in February 2006 and July 2009, respectively.

I was a teaching assistant on the following undergraduate courses: Digital and Nonlinear Control Systems, Nonlinear and Optimal Systems, Process Measurements, Basics of Digital Computers, Computer Architecture 1. Currently I am involved in the undergraduate courses Remote and Distributed Control Systems, Automatic Control, Laboratory and Skills – Matlab and Computer Controlled Systems.

During my undergraduate studies I received the Croatian Ministry of Science, Education and Sports scholarship for especially gifted students. The Faculty council of FER awarded me with recognition Josip Lončar for the second year of my undergraduate studies. I am the main developer of FER robot-soccer team ACT-Croatia, where I won the 5$^{\text{th}}$ place at 10$^{\text{th}}$ European Championship in Robot Soccer, Zürich/Linz 2008.

The main areas of my scientific interests are intelligent spaces with emphasis on mobile robot application, localization of mobile robots using computer vision, mobile robot motion planning based on computational geometry and control of mobile robots. I published one paper in CC-indexed journal *Robotics and Autonomous Systems*, two papers in journal *Automatika* and 15 papers in proceedings of scientific conferences.

Životopis

Rođen sam 25. srpnja 1978. u Zagrebu. Osnovnu školu sam pohađao u Pregradi, a prirodoslovno-matematičku gimnaziju završio sam 1996. u Krapini. Iste godine započinjem dodiplomski studij na Fakultetu elektrotehnike i računarstva (FER) u Zagrebu. Godine 1999. Fakultetsko vijeće odobrilo mi je završetak studija s naglaskom na znanstveno-istraživačkom radu. Diplomirao sam u rujnu 2001. s prosjekom ocjena 4,62. Od studenog 2001. godine student sam poslijediplomskog znanstvenog studija na FER-u i radim u Zavodu za automatiku i računalno inženjerstvo. Godine 2005. upisao sam doktorski studij na FER-u, u veljači 2006. položio sam kvalifikacijski doktorski ispit, a u lipnju 2009. održao javni razgovor.

Sudjelovao sam u nastavi iz sljedećih predmeta: Digitalni i nelinearni sustavi upravljanja, Nelinearni i optimalni sustavi, Procesna mjerenja, Osnove digitalnih računala, Arhitektura računala 1, dok trenutno sudjelujem na predmetima Sustavi za daljinsko i distribuirano upravljanje, Automatsko upravljanje, Laboratorij i vještine – Matlab te Računalno upravljanje sustavima.

Tijekom dodiplomskog studija primao sam državnu stipendiju za osobito nadarene studente. Fakultetsko vijeće mi je dodijelilo priznanje Josip Lončar za osobiti uspjeh na drugoj godini studija. Glavni sam istraživač FER-ova robonogometnog tima ACT-Croatia, s kojim sam osvojio 5. mjesto na Europskom prvenstvu u robotskom nogometu, Zürich/Linz 2008.

Glavna područja mog znastvenog interesa su inteligentni prostori s naglaskom na primjeni mobilnih robota, lokalizacija mobilnih robota računalnim vidom, planiranje gibanja mobilnih robota primjenom računalne geometrije te upravljanje gibanjem mobilnih robota. Objavio sam jedan rad u CC časopisu *Robotics and Autonomous Systems*, dva rada u časopisu *Automatika* te 15 radova u zbornicima znanstvenih skupova.