DISSERTATION

Robot Motion Planning with Genetic Algorithms

ausgeführt zum Zweck der Erlangung des akademischen Grades
eines Doktors der technischen Wissenschaften
unter der Leitung von

Univ.Prof. Dipl.Ing. Dr.Dr.h.c.mult Peter Kopacek
E 325
Institut für Mechanik und Mechatronik

eingereicht an der Technischen Universität Wien

Fakultät für Maschinenwesen und Betriebswissenschaften

von

Dipl.-Ing. Mag. Georg Sommer
88 25 373
Liesingbachstrasse 21
A-1100 Wien

Wien, am 18. Oktober 2005

Acknowledgment

I want to thank my academic advisor, Univ.Prof. Dipl.Ing. Dr.Dr.h.c.mult Peter Kopacek for giving me the opportunity to work on that topic and for all the helpful comments and discussions.

Many thanks go to Dipl.Ing. Dr. Man-Wook Han for introducing me into the world of the Nomad 200 robot and his support during the work.

Special thanks go to Univ.Prof. Dipl.Ing. Dr.Dr.h.c.mult Peter Herbert Osanna for taking on the second supervision.

Finally, I want to thank my family for all their support during the work.

Kurzfassung

Die autonome Navigation ist eines der zentralen Forschungsfelder im Bereich der mobilen Robotik. Einen wichtigen Punkt zur Erreichung dieses Zieles stellt die Kollisionsvermeidung von Hindernissen dar. Daher ist die Planung von kollisionsfreien, zusammenhängenden Pfaden zwischen Anfangs- und Endpunkten eine fundamentale Voraussetzung für die autonome Fortbewegung von mobilen Robotern.

Das Gebiet der Bewegungsplanung für Roboter ist seit drei Jahrzehnten ein aktives Forschungsgebiet und zahlreiche klassische Algorithmen zur Pfadplannung wurden entwickelt. Ihnen allen gemeinsam ist, dass sie zuerst einen möglichst exakten Plan der Umgebung entwickeln und danach einen Algorithmus zum Auffinden eines optimalen Weges zwischen Anfangs- und Endpunkt darauf anwenden. Der Schwachpunkt dieser Methoden liegt darin, dass sie sehr unflexibel auf Änderungen in der Umgebung reagieren und sehr langsam in großen und komplexen Umgebungen sind.

Einen vielversprechenden Weg diese Probleme zu überwinden, stellen heuristische Methoden dar. Diese unterscheiden sich von den klassischen Pfadplanungsalgorithmen dadurch, dass sie eine Umgebungskarte mittels stochastischen statt deterministischen Methoden erstellen. Eine beliebte heuristische Methode sind Evolutionäre Algorithmen. Sie basieren auf dem Darwinschen Prinzip 'Überleben der Geeignetsten' und haben sich als eine effektive Optimierungsmethode für komplexe Suchräume herausgestellt.

In dieser Arbeit werden Genetische Algorithmen zur Pfadplanung eines synchrongetrieben Roboters verwendet. Wir zeigen, dass solche Algorithmen schnelle und robuste Werkzeuge für Planungsaufgaben in Umgebungen mit Hindernissen sind. Es werden zuerst Ergebnisse von Simulationsstudien vorgestellt, die anschließend auf der mobilen Roboterplattform Nomad 200 implementiert werden.

Abstract

Autononomous navigation is one of the key problems in the field of mobile robots. To achieve this goal an important factor is to prevent the vehicle from colliding with obstacles. Planning a collision free, feasible path from the starting to the goal point is therefore a fundamental requirement for autonomous navigation.

Robot motion planning has been an active research area for the last three decades and numerous classical path planning algorithms have been developed. They all have in common that they build a preferable exact map of the environment at first and then use a certain algorithm to find an optimal path to reach the desired goal. The drawback of these methods is that they are inflexible with respect to changes of the environment or target points and that they are rather slow in large and complex environments.

A promising way to overcome that difficulties is to use sampling-based or heuristic methods . They differ from the classical ones by constructing a roadmap with probabilistic or random techniques instead of a deterministic way. A popular heuristic optimization method are Evolutionary algorithms. They are based on the Darwinian principle 'survival of the fittest' and have demonstrated to be effective procedures in complex search spaces.

In this work we use Genetic algorithms for planning motions of a synchronous drive robot. We show that such planners are fast and robust tools for planning tasks in environments with obstacles. We first did some simulation studies and tested it afterwards on the Nomad 200 platform.

Contents

1. Introduction and Motivation

The world of robotics currently undergoes a breathtaking development. Since the first industrial robots, which were build in the late 50´s and early 60´s[1], we passed the 800 000 barrier of installed industrial robots in 2004 and expect a number of 1 000 000 not later than 2007 [1]. This growth was possible because of the rapidly fallen prices of industrial robots. A robot sold in 2004 for example, would have cost about a fourth of what a robot with the same performance would have cost in 1990. In many cases robots approximate pay-back periods of about 1-2 years.

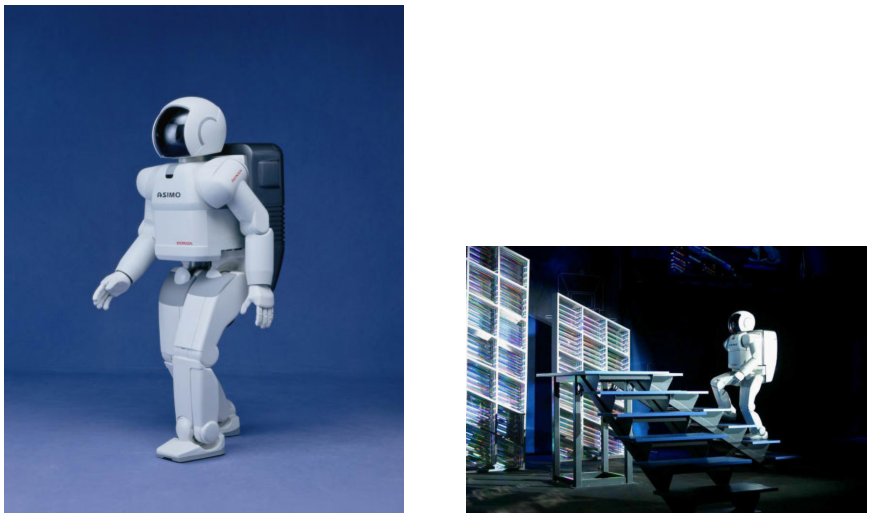
Industrial robots are important devices in a flexible automated production environment, due to their strength in repetition and precision at high velocities. They are primary used as handling devices, for arc welding, varnishing or assembling in an environment, where the objects to manipulate are fixtured to be in the right place at the right time. These manipulators lack any kind of intelligence and therefore their range of application is restricted to well defined tasks e.g. in production technology or transport systems. With the improvement of computational power and sensor systems the aim is to build more intelligent systems, which can plan, decide and act autonomously. Particularly with the demand of mobile systems, the ability to navigate purposefully in an environment needs some kind of intelligence.

[1]The first industrial robots were developed by George Devol and Joe Engelberger from *Unimation Company* .

The first examples of industrial-suited mobile robots were designed for transport tasks in assembly hangars. The locomotion of that AGVs (automated guided vehicles) based on induction loops, which were milled into or glued on factory floors. This kind of mobile vehicles are of course inflexible, to altering the route is costly and any unforeseen changes (such as objects blocking the path) can lead to failure in completing a task.

The only way to overcome these obstacles is to build autonomous robots, which have some kind of intelligence in order to reason about a *shapelarge scale* space, i.e. regions of space that are much larger than a system can observe from a single point. There has been made tremendous progress in the world of mobile robots within the past 10 years and the topic is still under rapid development. Today, we find mobile robots in various operational areas like transportation, cleaning, inspectations [97], guidance in museums [96], [95], [2] or in environments hostile to humans like underwater inspection [99] or mine detectors [4]. Great strides have also been made within the last 3 decades in the field of Human-computer Interaction [3]. This shows good promises, that in the nearly future robotic assistants could do some work for elderly or handicapped people such as fetching newspapers and mail, getting things out of high or low cabinets or carrying laundry, e.q. [98], [5]. As most of the presently used robots are driven by wheels or chains, the surmount of obstacles like steps, ditches or rough terrain is complicated or even impossible. The answer to that problem is to copy the kinematics and control system of humans, which leads to one of the most sophisticated topics in modern robotics, the area of Humanoid Robots.

The most famous representative is Honda's Asimo (Advanced Step in Innovative Mobility) [94], which appeared in public for the first time in 2000. The most impressive aspect of Asimo is it´s smooth and natural locomotion even when it climbs up steps (Fig.1.1). This is possible because of a predicted movement control (intelligent-WALK), that means, Asimo predict its next movement in real time and shift its center of gravity



(a) (b)

Fig. 1.1.: Honda's Asimo [94].

in anticipation[2]. The other technical facts of Asimo aren´t less impressive:

- 26 DOF (Degrees of Freedom) .
- Weight 43 kg , height 120 cm
- Vertical arm movement up to 105° .
- Velocity 1.6 km/h (walking), 3.0 km/h (running) .
- Recognise and respond to some 50 different calls, greetings and queries.

The development of Asimo was a milestone in the field of robotics and the continuous evolution in Hard- and Software technologies give hope, that mobile robots will be an important contribution to the improvement of life in human society in the 21st century. One the key aspects to realize this intention is the development of intelligent autonomous navigation

[2]That´s the same way how human beings walk .

concepts, which work fast and stable even in a noisy environment. Nearly all animate systems show some sort of structured navigation, i.e. they know where they are, plan a path to reach a goal point and are able to build a map[3] of the environment. Each of these aspects provides a large number of problems and challenges of there own and especially in combination with each other. In that work we will focus on the motion planning aspects, which is a fascinating and challenging research field for nearly 30 years.

1.1. Problem Definition

The problem of finding collision free paths between a starting and a goal point in an environment with randomly distributed obstacles is one of the key problems in robotics. In recent years it has been shown, that the insights in **path planning** algorithms are also interesting for other fields than robotics like computer graphics, virtual reality [6], industrial CAD, or even protein-ligand docking and drug design (see e.g. [7]). There exists a rich literature of classical path planning methods (Section 3.4). They all have in common that they build a preferable exact map of the environment at first and then use a certain algorithm to find an optimal path to reach the desired goal. This approach tends to be inflexible with respect to changes of the environment, target points or optimization goals. A further drawback of most of the classical path planning algorithms is their rapidly increasing computational time if the environment becomes too large and complex.

An auspicious way to overcome that difficulties is to use sampling-based or heuristic methods (Section 3.5) for motion planning. They differ from the classical methods by constructing a roadmap with probabilistic or random

[3] In this context, the word map denotes any kind of one-to-one mapping of the real world onto an internal representation. A well known example is the 2-dim street map of a city.

techniques instead of a deterministic way. A big advantage is that their complexity tends to be dependent on the difficulty of the path and much less on the global complexity of the scene.

A popular heuristic method for planning and searching problems are evolutionary algorithms. Evolutionary Algorithms are a class of global, parallel, stochastic and robust search methods founded on Darwinian principles. The central idea behind these techniques is the following: The fitness of a starting population will be improved from generation to generation through environmental pressure (Survival of the fittest). That means, given a function to optimize, we randomly generate a set of possible solutions (the starting population) and use the function values as an abstract fitness measure. In analogy to biological systems we apply the operations of selection, recombination and mutation to that population to improve the fitness from generation to generation until an optimum is reached. The planning of a path with evolutionary methods can therefore be represented in the following algorithm:

```
Create a number of different paths
DO
Measure fitness (e.g. the length of paths)
Select the fittest paths
Recombination to create offsprings
Mutation of some genes (as in nature)
UNTIL (an optimum is reached)
```

The challenging tasks are now to find an adequate data structure of the population, the optimal design of a fitness function and the optimal choice of some parameters for selection, reproduction and mutation. In chapter 5 we present the results of the genetic motion planning algorithms. We first did simulation studies to optimize the parameters of the GA and then tested it on a the robot platform Nomad2000 (see section 2.5.4).

The rest of the theses is organized as follows: In chapter 2, we give an survey about mobile robots and discuss the most important aspects of their Hard- and Software. In chapter 3 the Motion Planning problem will be formulated in a mathematical sense and some important classical planning methods will be introduced. In chapter 5 we discuss in detail the method of Genetic algorithms, which are the most popular representative of evolutionary algorithms. In Chapter 6 we summarize the results and give an outlook for further developments.

2. Mobile Robots

2.1. Introduction

Mobile robotics is a relatively young and interdisciplinary field, bringing together Engineers, Computer Scientists, Mathematicians, Physiologists and Physicians. It is an fascinating but sophisticated research field, because mobile robots have to consider all of the problems associated with static industrial robots but must also cope with a continually changing and uncertain environment. Therefore we are faced with manifold problems concerning kinematics, energy, vision systems and especially the intelligence for planning and acting autonomously or at least semi autonomously. There are three main questions, which a mobile robotic systems has to answer for itself:

- Where am I ?
- Where am I going ?
- How do I get there ?

Thus from an intelligent robot we expect the ability to move in it´s environment, to learn from previous experiences and to create some internal imagination of it´s world in order to reason about it and make some decisions [9]. In that sense, mobile robots are a physical embodiment of

what Computer Scientists call **intelligent Agents**[1], e.g. a piece of autonomous, or semi-autonomous proactive and reactive, computer software. The concept of 'agents' was first introduced in the 1970s in connection with systems of distributed artificial intelligence (DAI). The main characteristics of agent systems are [17],[18]:

- **Autonomy**, because agents should perform most of their tasks without interventions of humans. This implies that agents need some control over their actions and internal state.
- **Social Ability**, because agents should be able to act with other agents or humans.
- **Responsiveness**, because agents should observe their environment and respond to changes in a suitable time.
- **Proactivness**, because agents should act goal-oriented.
- **Adaptability**, because agents should learn from former experiences.
- **Mobility**, because agents should have the ability to chance their location if it is necessary to perform a task.
- **Veracity**, because agents must not knowingly communicate wrong information.
- **Rationality**, because agents are expected to act in order to achieve their goals.

As many tasks are too complicated for one agent, modern research focuses on the cooperation of many agents. Robot Soccer [12],[13] is a well-known example of these so-called Multi Agent Systems (MAS) [17], [15], [16] and has gained a lot of interest in the past years. The most important characteristics of such MASs are that each agent has incomplete information or capabilities for solving the problem and that there is no system

[1]The Hardware representation of Agents are also called Holons. The word Holon is a combination of the greek 'holos' (the whole) and 'on' (the particle), first introduced by Koestler in 1967 [8].

global control. Each agent is an autonomous entity and can act either cooperative or selfish. This provides the opportunity, that agent systems can share a common goal (like an ant colony) or pursue their own interests.

In opposite to Software Agents, robotic systems need also an interface for sensing the environment and actuators for acting in it. In Fig. 2.1 we find a detailed picture of a **Sense - Decide - Act** control scheme, which is common in mobile robotic systems. Sensors observe the surrounding area of the robot and convert these dates into useful information for the modeling system. This information is then used either to build a model of the environment or to compare it with an already existing model. The next step is to plan an action based on the robot´s model of the world (which is of course a simplified picture of the real environment) and finally to execute these actions by controlling the robot´s actuators. This Sense - Decide - Act process is repeated continuously until the final goal is reached[2].

In the next section we will discuss the most important hardware aspects of mobile robots, as far as it is necessary to understand their impact on planning and decision algorithms. There has been made a tremendous progress in robotic hardware recently, which is a necessary condition for developing intelligent agent systems.

2.2. Locomotion

Locomotion is the physical interaction between the robot and it´s environment. There exist countless possibilities of kinematical structures, de-

[2]In the last decade, that Sense - Decide - Act scheme has been criticized for a lot of reasons, especially due to the symbol grounding problem [22]. It states that symbols per se are meaningless and that reasoning based on such symbols is also meaningless. Therefore it´s impossible to develop real **intelligent** systems with such an control scheme. New approaches, like the **Behavior-Based Robotics** try to overcome these problems [25], [24], [23].

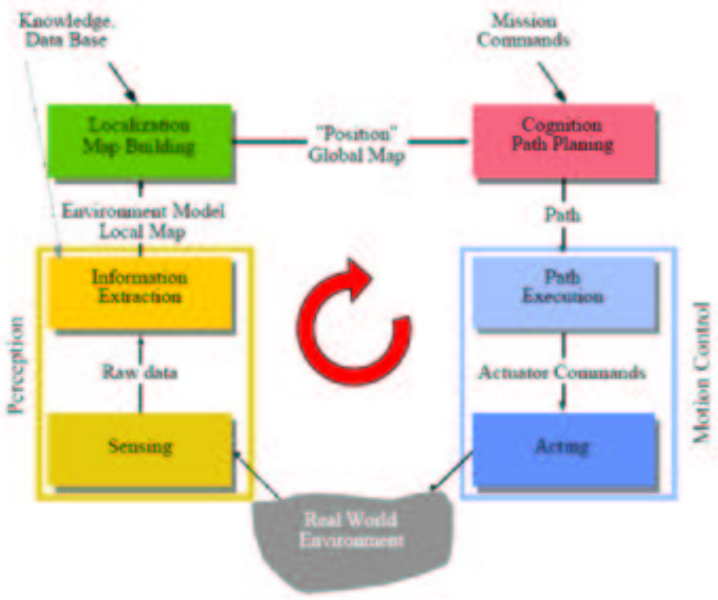


Fig. 2.1.: General Control Scheme for a mobile robot [10]

pendent on the application area and the environment where the robot has to operate. The most important questions for the design process concern the **stability**, the **characteristics of contact** and the **type of environment**. In the case of a flat ground the most efficient type of locomotion system is a wheel driven robot, i.e. two or more wheels are powered by DC motors and exploit the ground contact to move.

Differential Drive

Due to it´s simple steering mechanism, **Differential drive** is the most popular drive mechanism for ground-contact mobile robots. As depicted in Fig. 2.2 two wheels with distance l are fixed on a common axis. They are controlled by two separate motors and we suppose that there is no slipping. If we vary the two wheel speeds the robot must rotate about the point ICR (instantaneous center of rotation), which lies on the extended axis of the wheels. The distance R of the ICR with the center of the

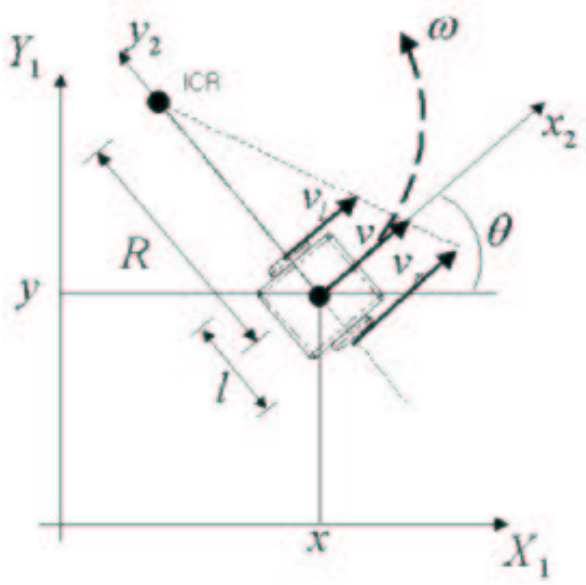


Fig. 2.2.: Differential drive robot [29].

wheels (x, y) thus is a function of $v_r(t), v_l(t)$. Let v denote the track velocity and ω the angular velocity, we find using the relation $\vec{v} = \vec{r} \times \vec{\omega}$:

$$\begin{aligned} v_r &= \omega(R + l/2) \ , \\ v_l &= \omega(R - l/2) \ . \end{aligned} \tag{2.1}$$

Solving this system, we get

$$\omega(t) = \frac{v_r - v_l}{l} \ , \tag{2.2}$$

$$R(t) = \frac{l}{2}\frac{(v_r + v_l)}{(v_r - v_l)} \tag{2.3}$$

$$V(t) = \omega \cdot R = \frac{1}{2}(v_r + v_l) \ . \tag{2.4}$$

There are two interesting special cases:

- $v_r = v_l \Rightarrow \omega = 0$ and $R \rightarrow \infty$, i.e. the path of the robot is a straight line.
- $v_r = -v_l \Rightarrow R = 0$, i.e. the robot rotates about the center of the wheels with $\omega = 2v$.

The possibility to turn around it´s own axis makes a differential driven robot attractive for navigation in narrow environments. For all other values of $v_r(t), v_l(t)$, the robot follows a curved trajectory with ICR as the center of rotation and the angular velocity ω. It´s important to note, that there is no combination of $v_r(t), v_l(t)$ that allows the robot to move sidewards along the wheels axis. This is an example of a **nonholonomic system**, which means, that adjacent areas in the environment are not directly achievable by the robotic system. Therefore it is much more complicated to plan feasible paths for nonholonomic systems as everyone knows, who has to drive a car into a small parking spot.

Starting from a pose x_0, y_0, θ_0, the control parameters $v_r(t), v_l(t)$ determine the pose $x(t), y(t), \theta(t)$ at any time t[3]. Integration over the time t gives us the pose for a robot with the velocity $V(t)$:

$$\begin{aligned} x(t) &= \int_0^T V(t)\cos(\theta)\ dt\ , \\ y(t) &= \int_0^T V(t)\sin(\theta)\ dt\ , \\ \theta(t) &= \int_0^T \omega(t)\ dt\ , \end{aligned} \tag{2.5}$$

where $V(t)$ and $\omega(t)$ can be inserted from Equ.(2.4) and Equ.(2.2).

The calculation of the **Inverse Kinematical Problem**, how to choose the control parameter to get a certain pose, is much more difficult and

[3]This is also called the **Forward Kinematics**.

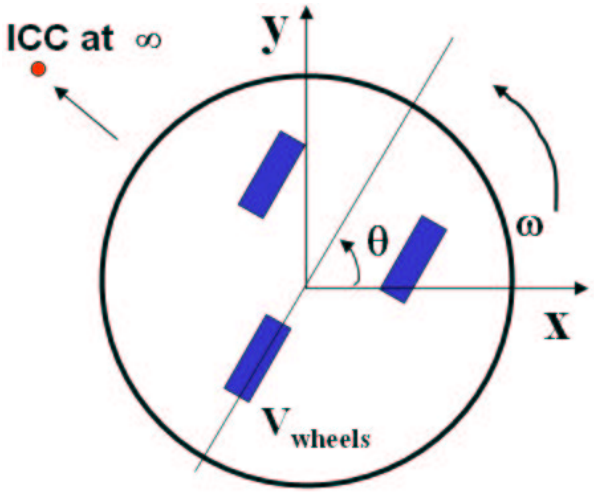


Fig. 2.3.: Synchronous drive robot [29].

analytical solutions exist only for special cases like $v_l(t) = v_l$, $v_r(t) = v_r$ and $v_l \neq v_r$.

Synchronous Drive

Synchronous (or **Synchro**) drive robots are usually equipped with 3 wheels, which are driven and can be steered. The wheels are arranged in a way, that they build an equilateral triangle and are coupled so that they turn and drive in the same direction (Fig.2.3). Usually two independent motors are used, one for driving and one for turning the wheels. In opposite to a differential drive, a synchro drive robot can drive in any desired direction[4] and is therefore a convenient model for an idealized point robot However the robot has to stop and realign it´s wheels and that´s the reason why it´s sometimes called an **almost holonomic** vehicle. As a synchro drive system rotates about it´s center with ω and drives with a velocity V_w we get the forward kinematics (see Equ.(2.5)):

[4]Therefore the shape is often cylindrical.

$$x(t) = \int_0^t V_w \cos(\theta)\ dt' \ , \tag{2.6}$$

$$y(t) = \int_0^t V_w \sin(\theta)\ dt' \ , \tag{2.7}$$

$$\theta(t) = \int_0^t \omega(t)\ dt' \ . \tag{2.8}$$

The inverse kinematics is similar to the differential drive and therefore we can find simple solutions only in special cases. Two of them are interesting:

- $V_w = 0$ and $\omega(t) = \omega \Rightarrow$ the robot rotates about it´s center.
- $V_w(t) = V_w$ and $\omega(t) = 0 \Rightarrow$ the robot moves along the direction of it´s wheels.

Ackerman (Car like) Drive

This driving concept is found in most cars today. There are two combined driven rear wheels and two front steering wheels, which can rotate on separate arms (Ackermann) or on a common arm (car like). An Ackerman Steering needs two independent motors, one for linear driving and one for rotation. The forward kinematics of the car robot can be calculated with the help of Fig. 2.4:

The center of the rear wheels drives with the velocity V, which leads to the following expressions for (x, y)

$$x(t) = \int_0^t V \cos(\theta)\ dt' \ , \tag{2.9}$$

$$y(t) = \int_0^t V \sin(\theta)\ dt' \ . \tag{2.10}$$

$$\tag{2.11}$$

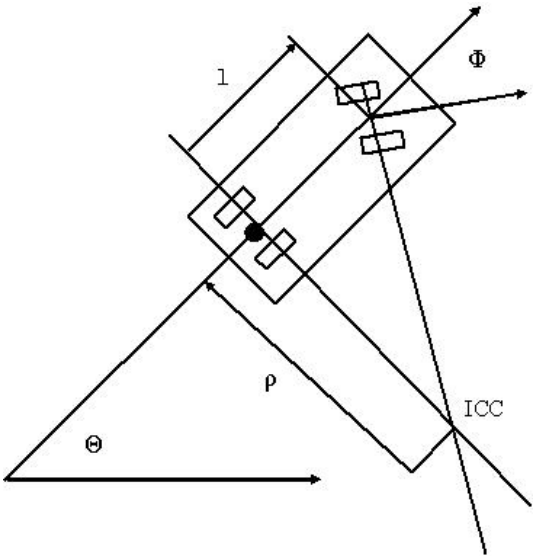


Fig. 2.4.: A car like robot

To derive an equation for $\dot{\theta}$ we first use the geometric relation (see Fig. 2.4)

$$\rho(\phi) = \frac{l}{\tan\phi} \ . \tag{2.12}$$

This relation is reasonable, because if the steering angle $\phi \to 0 \Rightarrow \rho \to \infty$ (the robot drives straight ahead) and if $\phi \to \pi/2 \Rightarrow \rho \to 0$ (the robot changes it´s direction instantly). As the latter case is only possible for a **tricycle**, an Ackerman steering has a maximum steering angle $\phi_{max} < \pi/2$, which implies a minimum turning radius ρ_{min}. If we fix the steering angle and consider an infinitesimal displacement ds of the rear wheel center we get $ds = \rho d\theta$. With $ds = V dt$ and Equ. (2.12) we find the relation for θ

$$\theta(t) = \int_0^t \frac{V}{l \cdot \tan\phi} \, dt' \ . \tag{2.13}$$

Fig. 2.5.: A Mecanum Wheel [101]

In Tab. 2.1 we shortly summarize the strength and weaknesses of the presented locomotion systems. All of them are common in mobile robot systems and it depends on the application areas which one will be preferred.

Omni-directional Drives

All of the above locomotion systems suffer from the kinematical restriction that they can´t drive sidewards[5]. In opposite to such **non-holonomic** systems omni-directional wheels allow movements in all directions from any configuration. Such systems have vast advantages in congested environments which are usually found in factories, warehouses or hospitals. Most of them are based on the ideas developed by the company Mecanum AB [26], [11] in the 70's. In general, Mecanum wheels consist of a driven central wheel, which is covered with a number of free rollers (Fig.2.5). The rollers are placed at an certain angle to the wheel axis. Therefore the force vector of the rotating wheels splits into a component forward to the direction and a component perpendicular to that. Depending on

[5]One exception is the Synchro drive which allows sidewards driving but only after stopping the translation and reorientation of the wheels.

Locomotion	Pros	Cons
Differential Drive	simple design, no turning radius	as there are two independent drive wheels, the control for moving a straight line can be very difficult.
Synchronous Drive	Simpler motion control due to two separate motors for translation and rotation. Straight line motion is guaranteed mechanically. Nearly holonomous system.	Wheel alignment is critical, especially on corrugated floors. The driving mechanism is mechanically more complex.
Ackermann Steering	Straight-line driving is stable. Simpler motion control due to two separate motors (as synchro drive).	Minimum turning radius. The inverse kinematics is quite complicated as it is a non holomomous system (Car parking). Rear driving wheels tend to slippage in curves.

Table 2.1.: Advantages and Disadvantages of the classical wheeled driving systems

each individual wheels direction and speed the resulting forces allow the movement in any direction of a 2D plane.

2.3. Sensing

Sensors are the interface between the robot and the environment. As mobile robots usually act in dynamic regions with frequently changing obstacles, sensor signals and their correct interpretation play a key role for a safe navigation. As each real sensor suffers from some physical limitations like **noise**, **returning of faulty signals** or **non-linearities** it is important to provide mobile robots with a lot of different sensor systems

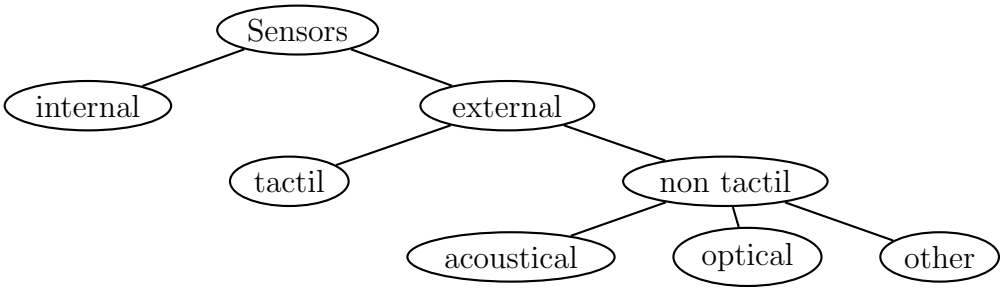
in order to get trustful signals. There exists a broad variety of sensor systems like

- Range Sensors (distance) ,
- Position Sensors (absolute position) ,
- Environmental Sensors (temperature, pressure, concentrations),
- Optical Sensors (image recognition),
- Internal Sensors (acceleration) .

In order to find the most useful sensor for an application one has to consider a lot of properties like

- **Speed of Operation**: the response time of a change in input .
- **Sensitivity**: ratio of change of Output to change of Input .
- **Robustness**: sensitivity to environmental influences .
- **Error Rate**: difference between 'real' data and measured ones .
- **Linearity**: ratio of input to output should be constant .
- **Range**: minimum and maximum values to be measured).
- **Resolution**: smallest observable increment in input.
- **Computational Requirements**: e.g. in the case of optical sensors.
- **Cost**: reaches from a few cents for a cheap infrared sensor to several thousand euros for an optical system.
- **Size, power, weight**: especially for mobile robots the dimension should be as small as possible.

The sensors can be classified into the following scheme:



Internal sensors monitor the robots internal state (e.g. acceleration) while external sensors monitor the robots environment. Tactile sensors detect physical contact with the robot, while non tactile sensors measure acoustical or optical waves, which are reflexed by distant obstacles. These sensors can be further subdivided into active systems (i.e. emitting energy) and passive systems (like a camera). Further on we can distinguish between local sensors, which are installed on the robot and global sensors, which are mounted outside the robot (e.q. a web-cam). Due to the fact, that no single sensor system is able to provide enough information for navigating safely, modern robot systems are equipped with different sensor systems which we will describe in the following.

Bumper

Bumpers are tactile sensors mostly based on micro switch or whisker technologies. In their simplest form they build an array of binary switches, which are embedded in a compliant material. After a physical contact one ore more of the switches are depressed and send a signal to the controller. More sophisticated bumpers use strain gauges or piezoelectric transducers, which return signals depending on the applied pressure. In some robots bumpers are directly connected to the e-stop circuit (e.g. Cybermotion Navmaster), providing the last possibility for collision avoidance, while in other systems they are just returning the position of the contact. As many

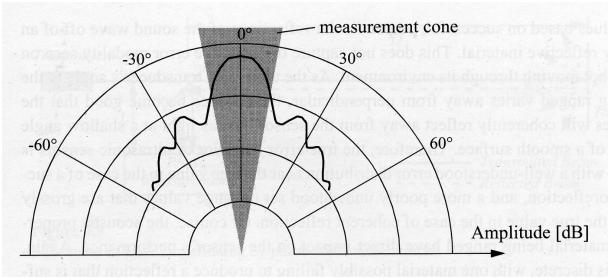


Fig. 2.6.: Cone-Shaped form of the sonar signal.

indoor robots are not equipped with active breaking systems or their inertia is too large for stopping within a small distance, bumpers are mostly not used for sensing reasons but for providing a force-absorbing material to avoid or reduce damage.

Sonar

Sonar sensors are active sensing systems which emit sound impulses and measure the time of flight and the difference in phase shift and attenuation of the reflected signals. Most of the commercial sonar systems based on ultrasonic waves at a frequency about $40-50$ kHz. They use a physical transducer both as an emitter and an receiver. Due to residual oscillations in the transmitter after generating an acoustic pulse it is unable to detect incoming signals for a short period of time (**blanking interval**). Therefore sonar sensors are blind for obstacles in short distances to the robot. Sonar sensors suffers a lot of further restrictions:

- Due to the cone-shaped form (Fig.2.6) of the signal an obstacle detected at distance d can be anywhere within that sonar cone .
- Most of the common objects are reflectively and therefore a detected signal might be the result of a series of reflections at different obstacles.
- The speed of sound depends on temperature and humidity.

In the last years their is an increasing interest of using RADAR Systems. Although they are still rather costly and bulky they are attractive due to the fact that they can provide some information about surface properties and geometry of obstacles.

Infrared Sensors

Infrared sensors are the simplest and cheapest type of non- contact sensors. They emit an infrared signal, which is often modulated with a low frequency to differentiate it from ambient light sources. In an ideal environment, where all objects are of the same color and surface structure, infrared sensors can easily be used for measuring distances d to obstacles as the intensity I of the reflected light is proportional to d^{-2}. This relation also explains why infrared sensors are inherently short range sensors. The drawback of such sensors is their strong dependency on obstacle colors, e.g. black bodies are nearly invisible to infrared sensors.

Laser Rangefinder

Laser Rangefinger sensors work similar to Sonar systems with the important improvment of using light instead of acoustic waves. The wavelength of that Laser sensors is near infrared light and therefore much smaller than sonar waves, which reduces the problem of specular reflections. The typical range of Laser systems is up to a several hundred meters, which make such systems also attractive for outdoor applications. There are three different methods of using Laser Range finders:

- **Time of flight**, which is analog to the sonar sensor. These system is very expansive because of the complicated electronics which is needed to resolve the picoseconds pulses.
- **Phase Shift**, means that a amplitude modulated light at a known frequency is emitted and the phase shift of the received signal is

measured. The distance D of the obstacle and the phase difference θ are related through

$$D = \frac{\lambda \cdot \theta}{4\pi},$$

where λ is the known wavelength. We see that the distance of an obstacle can be ambiguous as we get for all distances $D = \lambda \cdot n/2$, $(n = 1, 2, \ldots)$ the same angle θ. But in practice the range of the sensor is much lower than λ due to the attenuation of laser light in the air.

- **Triangulation** is a method of using geometric properties of the emitted and received signal. Today most of the receivers are CCD or CMOS cameras because they can recover distances to a large set of points, which gives information about the geometry of the obstacle. The emitter sends a structured light, like a laser stripe or another known pattern, onto the obstacle. In Fig. 2.7 we see that light source and the camera at a distance d enclose an known angle Ω. With the geometric relations $z/a = f/x'$ and $\tan\Omega = z/(d + x')$ find that the distance z can be recovered from the projection of the line x' in the camera as

$$z = \frac{fd}{f\cot\Omega - x'}$$

.

Due to their coherent light, which remains collimated over a large distance, Laser Systems are very accurate distance sensors over large distances. The disadvantage is, that they are blind to optically transparent materials like glass, which can be significant problem for indoor use (e.g. museums).

Visual Sensor

The ability to handle complex optical patterns is the most important sensor for humans and gets more and more important for technical systems,

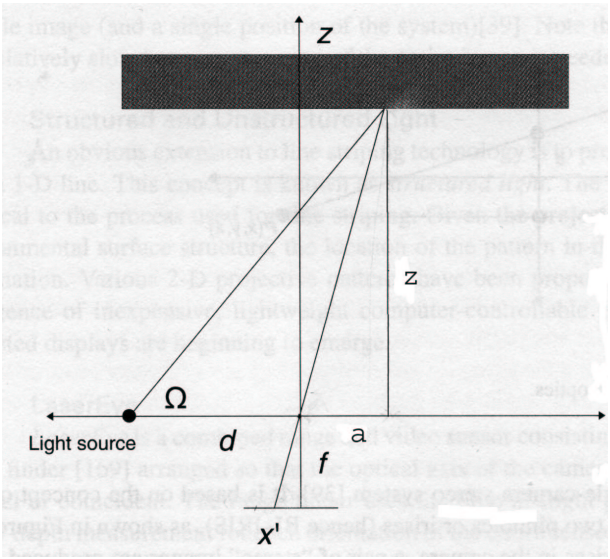


Fig. 2.7.: Principle of a laser based distance measurement [29].

too. Due to rapid developments concerning both the Hard- and Software of optical devices, a lot of progress had been made in the field of Machine Vision. The current Hardware technologies for optical systems are **CMOS** and **CCD** cameras. Both generate a two dimensional array where each pixel carries the gray level or color information. While the CCD (Charged Coupled Devices) Chip is an array of light sensitive picture elements, which will be readout line-by-line, CMOS (Complementary Metal Oxide Semiconductor) Chips readout each pixel. The advantages of CMOS over CCD are that the CMOS technology is simpler and therefore easier and cheaper to produce, that it consumes significantly less power and that there is no **Blooming** effect[6]. On the other hand, the advantages of CCD are to be more sensitive, they better compensate defective pixel and that the technology is older and therefore more mature. It is far beyond the scope of that work to give an overview of the countless possibilities to get information out of the gray-level matrix after a pic-

[6]A too bright illumination causes spill from one photo sites into the adjacent photo sites of the transfer row, which leads to incorrect values.

a	b	c
d	e	f
g	h	i

Table 2.2.: A 3x3 matrix of color information.

ture is taken. However, there are a few basic image processing techniques, especially for detecting edges and reducing noise, which are widely used and easy to understand and implement[7]. The fundamental principle of all that image processing operators is to modify individual pixels of the original matrix X by applying an matrix operator H, called the **convolution mask**. The new Matrix Y is the result of the discrete convolution $Y = X * H$, which is defined as:

$$Y(x,y) = \sum_{i=-\infty}^{i=\infty} \sum_{j=-\infty}^{j=\infty} X(x+i, y+j)H(i,j)$$

The convolution mask H is in general a $3\times3, 5\times5, \ldots$ matrix.

In Tab. 2.2 we have a small color image matrix, where the letters represent some numbers (e.g. $0 \ldots 255$ in a 8 bit gray level), and apply a few of the most common operators H to it:

- **Median filter**: We replace the middle pixel e by the median of the 9 values. This operation eliminates peaks, which are mostly the result of noise.
- **Prewitt Operator**: Edges are often associated with a sharp change in intensity. If the intensity is a continuously differentiable function we can apply the gradient operator. In the discrete case we can use the operator $\nabla = (\nabla_1)^2 + (\nabla_2)^2$, with

[7]There is a lot of image processing literature (see e.g. [19], [20]).

$$\nabla_1 = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} , \nabla_2 = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} .$$

If ∇ exceeds a present threshold it is retained otherwise discarded.

- **Sobel Operator**: is similar to the Prewitt operator and works also as a high pass filter, but with different definitions of $(\nabla_1)^2$ and $(\nabla_2)^2$

$$\nabla_1 = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} , \nabla_2 = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} .$$

- **Laplace Operator**: The disadvantage of the former operators is on the one hand that the derivative- like operators amplify any noise and on the other hand, that the threshold has to be set a priori. Using the Laplace Operator avoids this problem, because it is a second derivative operator and therefore zero at discontinuities like an edge. One possible representation of the discrete Laplace operator $\triangle$ is the following:

$$\triangle = \begin{pmatrix} -1 & 0 & -1 \\ 0 & 4 & 0 \\ -1 & 0 & -1 \end{pmatrix} .$$

2.4. Control

Closed loop control (Fig.2.8) is an essential topic in mobile robotics connecting actuators and sensors with the control software, in order to counteract a certain deviation from a desired behavior. When a robot shall move along a special trajectory, the encoder values are sent as actual

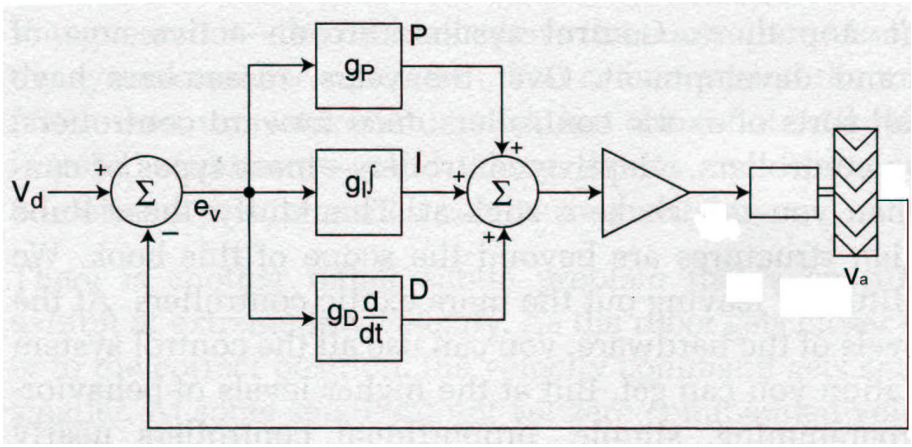


Fig. 2.8.: The closed loop control scheme of a PID Controller.

values to the controller. The speed setpoint is sent as a pulse width modulation (PWM) signal to the controller and differences between actual $v_a(t)$ and desired values $v_d(t)$ are measured. The task of the controller is to minimize this difference as rapidly as possible without generating oscillations of the system. A standard way to do this is to use a **PID** (proportional, integral, differential) controller, which can be written in the following way:

$$y(t) = g_p \cdot e_v(t) + g_I \cdot \int e_v(t)dt + g_D \cdot \frac{d}{dt}e_v(t) \ ,$$

where $e_v(t) = v_d(t) - v_a(t)$ is the error function and $y(t)$ the motor output function.

The tuning of the tree parameters g_p, g_I, g_D is a tricky matter but there exists a few guidelines that can be used for experimentally finding suitable values (see e.g. [105], [27]).

2.5. Important Aspects of Service Robots

As prices of Hard- and Software components fall steadily and simultaneously their capability and quality raises, robots leave more and more the manufacturing halls and become part of every day life. In opposite to a relatively narrow application area of industrial robots, these service robots occur in a tremendous variety concerning their design, functionality and use. A useful definition of service robots originates from the Fauenhofer IPA 1994 [21]:

A Service robot is a freely programmable mechanism, which performs a service task. A Service task is an activity, which isn´t made for producing real assets but accomplish a benefit for humans and facilities.

Although most of the 'non industrial' robot systems are experimental and used for education or research some mobile solutions are already commercially available. At the end of 2003 about 21.000 Service robots for professional use were installed [1]. The areas of application are for example:

- Underwater systems ($\sim 23\%$)
- Cleaning robots ($\sim 16\%$)
- Laboratory robots ($\sim 15\%$)
- Medical robots ($\sim 12\%$)
- Defense, Rescue and Security applications ($\sim 5\%$)
- Agriculture systems ($\sim 4\%$)

The value of the stock of professional service robots is estimated at $2.4 billion[8] Approximately 600.000 other units for domestic use and almost 700,000 units for entertainment and leisure were sold at the end of 2003 and and it is expected that several millions of these mobile platforms will be installed in the next few years [1]. These numbers show that besides the exciting research problems which mobile robots offer, there is an increasing market potential for such systems.

The increasing interest of mobile robots is based on the fact, that such systems are especially well suited for tasks that exhibit one of the following characteristics:

- Hostile or inaccessible environment, e.g. the deep sea or mine fields .
- Contaminated environments, e.g. in a nuclear reactor .
- Locations which are far away, e.g. other planets .
- Tasks with a very high fatigue factor, e.g. security control .

The next generation of service robots go beyond these applications and will display an enormous progress in the field of Human-Robot Interaction, which enables such systems for tasks like

- Support for elderly or handicapped people ,
- Edutainment ,
- Medical applications ,
- Cleaning of homes .

[8] It´s also interesting to throw a glance at the unit prices (in 2004) for professional service robots [1]; the most expensive robots are the underwater systems ($ $\sim$ 300.000) , followed by milking robots ($ $\sim$ 200.000). The average price of a medical robot is about ($ $\sim$ 150.000).

Fig. 2.9.: Care-O bot II, Frauenhofer Institute IPA [98]

Successful applications in these areas involve a highly secure and fault-tolerant technology. Due to the innovative character of mobile service robots, specifications and standards for the construction and operation of such systems are still lacking. Nevertheless, a lot of systems which are already in use now provide a high degree of functionality. In the following we want to describe 3 interesting representatives of intelligent mobile systems.

2.5.1. Care-O bot II

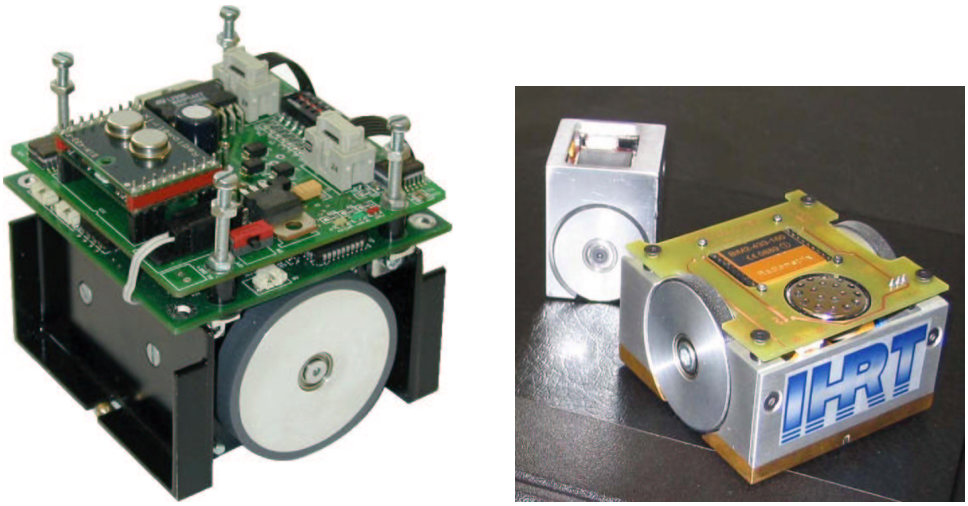
Care-O-bot II [100], developed at the Frauenhofer Institute (IPA) Stuttgart, is a mobile service robot which has the capability to interact with and assist humans in typical housekeeping tasks. It supports humans in tasks like grasping, holding, or lifting, can execute everyday jobs such as setting the table, operating the microwave, or simple cleaning tasks or control the home infrastructure as e.g. heating system, lights or doors.

The increasing number of elderly people and thus the number of people impaired by diseases or handicaps will rise quickly in the next future[9]. Therefore technical aids will play an important role for an adequate support for all elderly people.

The Care-O-bot is equipped with a differential drive including shaft encoders for motion tracking. It is able to move at a speed of up to 1.2 m/s. Four castor wheels are further used for keeping the robots upright. A gyroscope is integrated to the robot platforms to track their current orientations. A 2D laser scanner is attached to the front of each robot. It is used for self-localization, navigation, and obstacle detection.

As Care-O-bot interacts with handicapped people a redundant safety system is of great importance. It is equipped with a lot of additional safety sensors like a bumper at the bottom of the robots, several infrared sensors, a laser scanner and a magnetic sensor facing toward the ground, which is used to prevent the robot from leaving their assigned area[10].

The navigation is completely autonomous with the laser scanner used for detecting the current position. The robots integrate automatic self-test, start-up, and shutdown capabilities and can therefore easily be operated by untrained personnel. The communication system is based on wireless LAN and allows for a a remote control and diagnosis using a stationary PC.



(a) Roby-Go. (b) Roby-Go 2.

Fig. 2.10.: Roby-Go and it's successor, TU Vienna.

2.5.2. Roby-Go

The mobile robot **Roby-Go** [102], developed at the TU Vienna, is a differential drive system with 4.05 W electric drives with a speed of 8000 1/min and a maximum speed of 2.54 m/s which can be reached within 500 ms . It´s main application is as a soccer player in the category MiroSot (FIRA) but it also serves as an education tool for students in the field of control engineering and mechantronics as well as a test-bed for Multi-Agent Systems. The edge length of the cube is 75 mm and the robot is equipped with two wheels with a diameter of 48 mm and a width of 8 mm and weighs 450 g (with batteries). The power supply will be ensured by 9

[9] According to numbers of the Federal Statistical Office, in 2001 around 24 percent out of 82.5 million people living in Germany were senior citizens over 60 years. With demographic development continuing, the number of people over 60 years old will reach 35 percent of Germanys population in the year 2040. In 1999 the number of elderly people that were treated by one nurse was around nine. According to current predictions, in 2050, this ratio will lie at about 1:17.

[10] This area is surrounded by a magnetic band.

Fig. 2.11.: Transcar, an AGV from Swisslog [103].

$NIMH$ cells with a total capacity of 700 mAh by 1.2V, which gives enough power for driving 20 min with full speed. The electronic system is designed modular and with an open architecture consisting two PCB boards with C167CR-LM (Infineon) micro controllers with a CAN-Bus Interface and which are programmable by a serial interface. The communication of the robot with the environment is by a radio module using the serial interface. The required velocities of the left and right wheel will be send by a standard PC.

2.5.3. Transcar

The Swisslog AGV **Transcar** [103] is a transport system for a variety of bulk items through a hospital. The robots are ideally suited for moving routine, on-demand or large quantity items, like meals/soiled, dishes or surgical cases, between centralized functions such as kitchens, laundries and storerooms.

Transcar offers:

- Low profile, compact, fully symmetric bi-directional vehicles ,
- a contour-following laser guidance system for maximum safety, flexibility and cost efficiency ,

Fig. 2.12.: Nomad200

- a dual range non-contact laser obstacle detection ,
- an automatic lifting and transport of four-wheel carts of varying configurations and loads ,
- quick charging NiCD batteries for maximum vehicle availability ,
- advanced PC computer system directing all vehicle movement, tracking, diagnostics and information feedback .
- a communication system, which is a combination of wired and wireless LAN.

2.5.4. Nomad2000

The Nomad200 from Nomadic Technologies is an integrated mobile robot system widely-used in teaching and research in Robotics and Artificial Intelligence. It is based on a three wheel synchronous drive kinematics controlled by two motors, one for translation and one for rotation. Servo control is performed by a MC68008/ASIC microprocessor system.

It has a maximum translational speed of 0.6 m/s and a rotational speed of $60°/s$.

There are 6 different sensory systems installed: tactile, infrared, ultrasonic, Laser scanner, vision system and compass. The tactile system works with 20 switches, which are organized in two rings with 10 switches in each ring. The sonar system has 16 channels giving range information from 17 to 255 inches with 1 % accuracy over the entire range. It is based on a time of flight measuring method of acoustic signals at a frequency of 49.9 kHZ. The infrared system is build up of 16 sensors, which measure the reflective intensity up to 24 inches.

The vision system consists of a Sony XC-75 standard CCD camera with a resolution of 490 $\times$ 512 pixels, a frame grabber as an interface to the motherboard and a software library, providing low level vision algorithms for edge detection, line extraction, distance transformation, etc. Together with a Sick Laser scanner it builds a Laser range finder sensor. The operating range is from 18 to 120 inches and has a power requirement of 2000 mA at 12 V .

The Energy Systems consists of 5 12V-12Ah batteries providing 840 Wh usage, which is about 7h driving time for the robot.

Programming of the robot will be done off line on a standard Linux PC using the language C. Before the program is send to the robot via a wireless Ethernet Connection , it can be tested in a simulation environment (see App. B).

The Nomad200 is a perfect Hardware platform for testing various kinds of problems in the field of mobile robotics and will also be used as a testbed in that work.

3. Robot Motion Planning

One of the key challenges of autonomous robots are adequate planning strategies. The goal is to specify a task in a high level programming language, or as a long-term objective in everyday speech, and the robot is able to transform it into motion and control instructions. The fundamental task in motion planning is to find a path for a robot, whether it is a robot arm, a mobile robot, a humanoid or a virtual agent, from a start to a goal configuration without collisions. Thereby we are confronted with two kinds of constraints: **global constraints**, which arise from obstacles in the environment and **local constraints**, which arise from the kinematics of the robot. As the power and generality of planning algorithms increases, we are able to handle applications of increasing difficulty, like systems with high degrees of freedom, complicated geometric and differential constraints. In Fig.3.1 and 3.2 we find demonstrative examples that path planning isn't constricted only to mobile robots. Although the applications are very different[1], the tasks are in both cases the same: reaching a possible goal configuration from a given start configuration with respect to constraints.

Since the planned motions for robotic systems have to be executed in a world with physical laws, uncertainty and geometric constraints, path planning has also to consider various aspects of mechanics, control theory,

[1] On the one hand we are facing a **continuous** on the other hand a **discrete** planning problem.

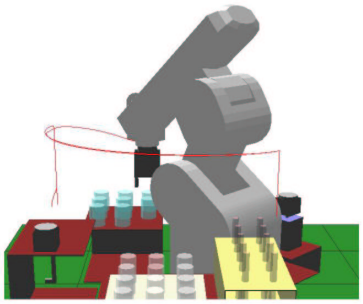


Fig. 3.1.: A continuous planning problem.

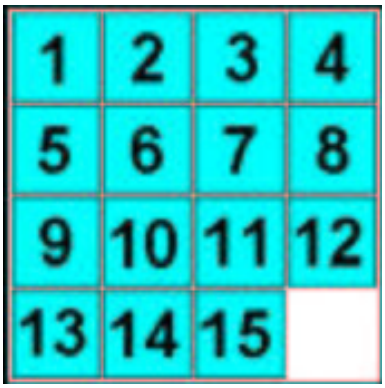


Fig. 3.2.: A discrete planning problem.

differential geometry and computer science. This leads to a further important aspect of autonomous motion; that a robot has to collect it´s correct position and orientation at each time. These tasks result in three problems, which an autonomously guided robot has to solve:

- **Map building**, i.e. finding a useful representation of the environment.
- **Target finding**, i.e. planning of 'optimal' paths.
- **Target tracking**, i.e. the robot has to know it´s correct position.

All of that points seem to be easy tasks for humans and animals (at least in a known environment), but they are extreme challenging exercises for artificial systems[2]. In this work the main focus lies on the path planning tasks but will also stripe the two other aspects of navigation as far as they are important for us.

[2]A prominent example is the First *Grand Challenge '04* field test of autonomous ground vehicles organized by DARPA (Defense Advanced Research Projects Agency), a rally from Barstow, California to Primm, Nevada about 160 miles across the Mojave dessert. Despite a $1 million prize none of the teams reached the goal, most of them failed after a few 100 meters [104].

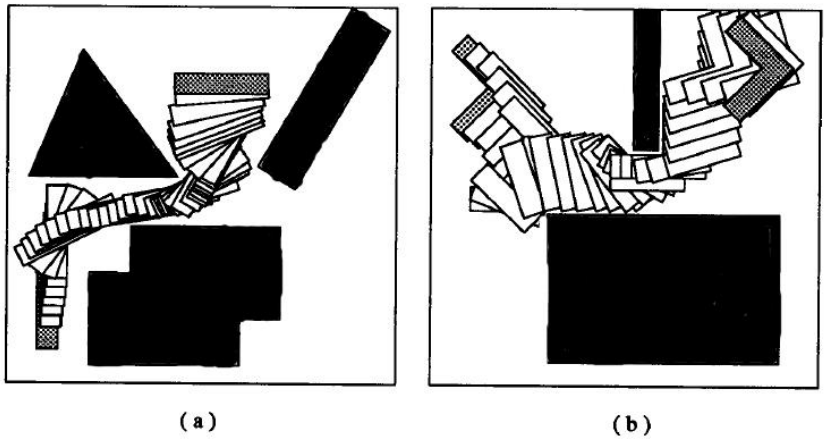


Fig. 3.3.: Basic Path Planning Problem of two robots with different shapes [31].

3.1. Path Planning of mobile robots

The general motion planning problem can easily be described in the following way: *Given an initial state, a final state and constraints of allowable motion. Find a collision- free path between these two states, which satisfies all the constraints.*

We can divide this problem in three parts:

1. Finding a free path in the space where the robot is moving .
2. Planning a trajectory, e.g. at each point the robot has to know it's orientation, velocity and acceleration .
3. Regarding the robot's geometry and kinematical constraints .

With that prerequisites the task is to find algorithms for collision free paths of the moving robots. A further aspect of planning is that we are not only searching for feasible paths but also for paths, which are optimal in some specified manner. This could include the following aspects:

- finding those paths of minimal length,
- finding paths of minimal costs or minimal energy consumption,
- finding "safe" paths in a dangerous environment,
- considering moving obstacles instead of fixed one,
- coordinate the motion of several robots,
- planning and coordination of sliding and pushing motions in order to achieve a special constellation of the obstacles,
- reasoning about uncertainty to build feasible sensory- based motion strategies.

For most of the real world problems, feasibility is already challenging enough; finding optimal paths is much harder and sometimes even impossible due to the fact that we either can´t formulate the right criteria for optimality or can´t find an sufficient fast algorithm. In such cases, finding feasible paths is the only goal we can reach, but fortunately these solutions are often sufficient feasible for practical applications.

If we neglect the kinematical aspects of the robot (*Trajectory Planning*) in a first step, we have to solve a completely geometric problem often called the *Basic Motion Planning Problem* (Fig.3.3). Under the assumption that the geometry of the robot and the geometry and location of the obstacles are perfectly known, a lot of algorithms to that "Free Flying Object" situation have been developed within the last 25 years[3]. They base on different theoretical assumptions and requirements concerning the following questions [29]

- **Environment and robot:** What is the structure of the environment, what about the shape of the robots ?

[3]Mostly in the field of Computer Science and Mathematics as there are a lot of search problems which can be traced back to path planning.

- **Soundness:** Is the planned trajectory collision free ?
- **Completeness:** Do the algorithms guarantee to find a path ?
- **Optimality:** Are the founded paths the optimal ones ?
- **Space or time complexity:** How much storage space and CPU time is taken to find a solution ?

Dealing with realistic path planning problems means that we have to simplify the mathematical model in order to get fast enough solutions, e.g. a common prerequisite is that the obstacles of the real world are mapped into convex polygons. Thus, we have always be aware of these simplifications and have to think about control mechanisms to repair the deviations from the real situation.

Basically we can classify path planning methods in **local** and **global** ones. Global methods plan in the whole environment, while local ones plan in a definite region and summarize their part solutions at the end. Despite the huge variety of classical planning algorithms they essentially base on three different ideas: the **roadmap**, **cell decomposition** and **potential field**. In the last years there has been an increasing interest in using **heuristic** or **probability based** planning strategies as for complex systems the classical methods proved to be too slow in finding feasible solutions. Before we will describe these methods in detail we want to introduce some important definitions and geometric concepts, which are necessary for a thoroughly understanding of motion planning problems [31].

3.2. Mathematical Background

We consider a robot $\mathcal{A}$ moving in a 2-dimensional environment, with some obstacles $\mathcal{B}_1, \ldots \mathcal{B}_n$. This environment where a certain robot operates is called the *workspace* $\mathcal{W} \subset \mathcal{R}^n$ (n=2,3). The robot $\mathcal{A}$ is a rigid object (with

a certain position and orientation), which can be described as a compact subset $\mathcal{W}$ of the $\mathcal{R}^n$ (n=2,3). In most of the cases the robot $\mathcal{A}$ will be a simple polygon with n vertices and n edges or a circle with radius r. The obstacles $\mathcal{B}_1, \ldots \mathcal{B}_n \subset \mathcal{R}^n$ are fixed rigid objects in $\mathcal{W}$. They will be represented as closed polygons (not necessary convex) or circles of radius r. In order to describe the motion of the robot we need a coordinate system which is fixed in the workspace $\mathcal{F}_\mathcal{W}$ (*world coordinate system*) and a frame $\mathcal{F}_\mathcal{A}$, which moves with $\mathcal{A}$. Since $\mathcal{A}$ is rigid, every point $a \subset \mathcal{A}$ has a fixed position with respect to $\mathcal{F}_\mathcal{W}$. But the coordinates of a in $\mathcal{W}$ depend on the position and orientation of $\mathcal{F}_\mathcal{A}$ relative to $\mathcal{F}_\mathcal{W}$.

As an example we consider the 2-dimensional flat plane with the standard Euclidean coordinate System ($\mathcal{R}^2$) and a square shaped robot $\mathcal{A}$ with vertices at $(-1,1),(1,1),(1,-1),(-1,-1)$. A translational movement of $\mathcal{A}$ about the vector $(1,1)$, denoted by $\mathcal{A}(1,1)$, leads to the new vertices $(0,2),(2,2),(2,0),(-2,-2)$. With this form of representation, a robot can be specified by listening the vertices of $\mathcal{A}(0,0)$. A different and in the case of rigid shapes more practical way to identify the position of a robot is by a reference point. This point is in general the origin of the coordinate system $(0,0)$ and can be inside (e.g. the barycenter) or outside the robot. Thus the notation $\mathcal{A}(x,y)$ specifies that the reference point is placed at (x,y).

A *pose* or *configuration* of a robotic system in the workspace means the position and orientation of $\mathcal{F}_\mathcal{A}$ with respect to $\mathcal{F}_\mathcal{W}$. Any point of the robot $\mathcal{A}$ has a fixed position in $\mathcal{F}_\mathcal{A}$ but it´s pose depends on the pose of $\mathcal{F}_\mathcal{A}$ relative to $\mathcal{F}_\mathcal{W}$. The set of all possible configurations is then called the *configuration space* $\mathcal{C}$. It is the space of all possible configurations of a robotic system. The topology of that space is usually not that of a Cartesian space[4]. In general $\mathcal{C}$ is described by a list of real parameters.

[4]To be more precise, $\mathcal{C}$ is a smooth manifold of dimension n. That means that the local topological and differential structures are isomorphic to $\mathcal{R}^n$. For the global structure of the manifold this is not necessary true. This means, that we can always find some smooth paths in a certain region but necessary in the whole configuration space.

Fig. 3.4.: A 2 DOF Articulated Robot and it´s configuration space

For example (Fig.3.4), the configuration space of an 2 DOF articulated robot is that of an torus.

Another example are car- like mobile robots, which have 2 translational and 1 rotational DOF. They can be described by the real parameters (x, y, θ), where (x, y) denote the position in workspace (e.g. of the center of gravity) and θ denotes the orientation of the vehicle. In that case $\mathcal{C}$ is $\mathcal{R}^2 \times S^1$, which has the topology of a cylinder[5]. In general, for each DOF a single parameter is needed and it´s obvious that each additional DOF complicates the structure of the configuration space (Fig.3.5).

As we are interested in moving objects we need a definition of a *path*, which will always be a continuous map from an initial to a final state. A path of $\mathcal{A}$ from the configuration q_{init} to q_{goal} is a continuous map

$$\tau : [0, 1] \rightarrow \mathcal{C} \ ,$$

with $\tau(0) = q_{init}$ and $\tau(1) = q_{goal}$.

A *trajectory* is a path parameterized by time:

$$\tau : t \in [0, T] \rightarrow \tau(t) \in C$$

[5]If we only consider the translation of a robot in $\mathcal{R}^2$ the configuration space is identical to the workspace. Nevertheless a strict disjunction between these two spaces is useful.

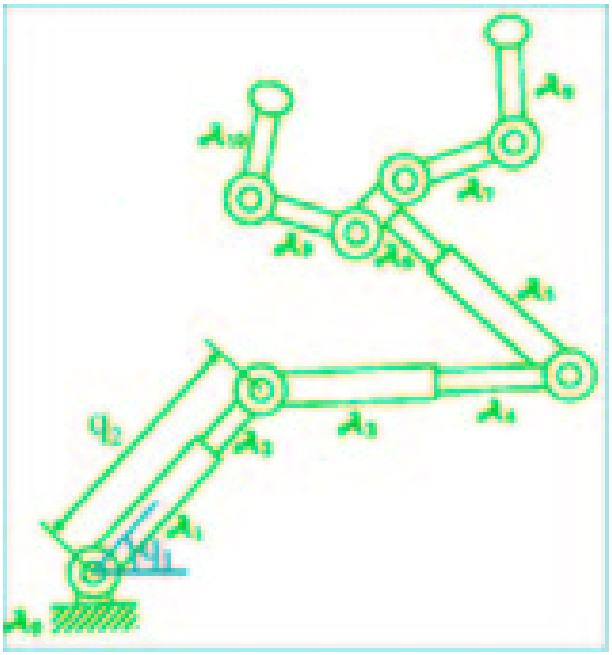


Fig. 3.5.: A 10 DOF Articulated Robot. $\mathcal{C}$ is $(S1)^7 \times R^3$

As a distance function we will use the metrics[6]

$$d(\vec{q_1}, \vec{q_2}) = \max_{a \in \mathcal{A}} ||a(\vec{q_1}) - a(\vec{q_2})||$$

where $||\vec{x}_1 - \vec{x}_2||$ denotes the Euclidean distance between two points $\vec{x}_1, \vec{x}_2 \in \mathcal{R}^n$.

As a specific example we consider $\mathcal{C} = \mathcal{R}^2 \times \mathcal{S}^1$. The two configurations are $a(\vec{q_1}) = (x_1, y_1, \theta_1)$ and $a(\vec{q_2}) = (x_2, y_2, \theta_2)$ with $\theta_1, \theta_2 \in [0, 2\pi)$. The distance is

$$d(\vec{q_1}, \vec{q_2}) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2')^2 + \alpha^2}$$

with $\alpha = \min\{|\theta_1 - \theta_2|, 2\pi - |\theta_1 - \theta_2|\}$. We see that $d > 0 \ \forall a(\vec{q})$ and $a(\vec{q_1}) \rightarrow a(\vec{q_2}) \ \Rightarrow d \rightarrow 0$.

This definition of a path implies that the moving object doesn´t have any

[6]The metrics induces a topology in $\mathcal{C}$, so that the term "continuity" is reasonable.

kinematic or dynamic constraints, thus we consider a *a free flying* object $\mathcal{C}$. A remark should be made about the differentiability of paths. Although we don´t need smooth paths in the basic motion planning problem there is little interest in generating paths which we know in advance to be infeasible. Thus we will only consider differentiable paths and discard non smooth paths.

The above considerations didn´t include the obstacles $\mathcal{B}_n$ of the workspace. Thus, we have to map the obstacles into the configuration space and to look for free paths in a geometrically constraint space. Every obstacle $\mathcal{B}_i$ $(i = 1, \ldots n)$ in the workspace $\mathcal{W}$ maps in $\mathcal{C}$ to a region

$$\mathcal{CB}_i = \{\vec{q} \in \mathcal{C} |\ \mathcal{A}(\vec{q}) \cap \mathcal{B}_i \neq \emptyset\}$$

which is called a C-obstacle. The union of all the C-obstacles

$$\bigcup_{i=1}^{q} \mathcal{CB}_i$$

is called the C-obstacle region and in opposite to that, the set:

$$\mathcal{C}_{free} = \mathcal{C} \setminus \bigcup_{i=1}^{q} \mathcal{CB}_i = \{\vec{q} \in \mathcal{C} |\ \mathcal{A}(\vec{q}) \cap \mathcal{B}_i = \emptyset\} \ ,$$

is called the free space. Any configuration in $\mathcal{C}_{free}$ is called a free configuration. A *free path* is therefore a map $\tau : \ [0,1] \rightarrow \mathcal{C}_{free}$. Therefore a configuration $a(\vec{q})$ is collision-free, or free, if the robot placed at $a(\vec{q})$ has null intersection with the obstacles in the workspace. A configuration is called *semi-free* if the robot at this configuration touches obstacles without overlap[7].

A major problem of the configuration space representation is that with increasing complexity the search space can be very large and therefore

[7]Such a configuration can´t be accepted for real paths because of uncertainties of the control system or problems by touching instable objects.

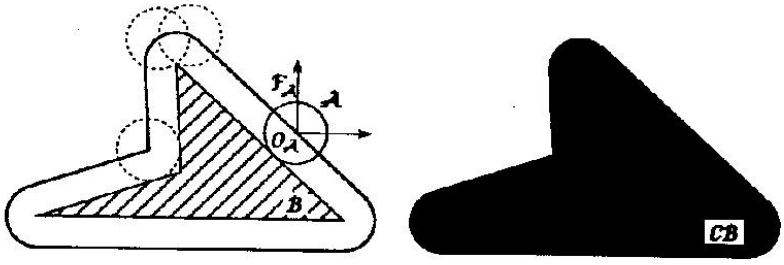


Fig. 3.6.: A moving disc $\mathcal{A}$ in $\mathcal{R}^2$ with a polygonal obstacle $\mathcal{B}$ [31].

time consuming to search. A common simplification is to assume that the robot can be approximated by a disk with radius r and is able of omnidirectional motion[8]. Based on that assumption we can enlarge the obstacles in the C space about the radius r and treat the robot as a point particle. The dilation operation is called **Minkowski Sum** and is an important tool for calculating $\mathcal{CB}$. In it´s general form the Minkowski Sum of two sets $\mathcal{A}$ and $\mathcal{B}$ is defined as follows:

Definition 3.2.1 *The Minkowski sum of to sets $\mathcal{A} \subset \mathcal{R}^2$, $\mathcal{B} \subset \mathcal{R}^2$, denoted by $\mathcal{A} \oplus \mathcal{B}$, is defined as*

$$\mathcal{A} \oplus \mathcal{B} := \left\{ \vec{a} \in \mathcal{A}, \vec{b} \in \mathcal{B} |\ \vec{a} + \vec{b} \right\} \ ,$$

where $\vec{a} + \vec{b} = (a_x + b_x, a_y + b_y)$.

As an first example we consider a robot $\mathcal{A}$, approximated by a disk of radius r, and an obstacle $\mathcal{B}$, approximated by a polygon, in $\mathcal{W} = \mathcal{R}^2$. The Minkowski Sum is received by navigating the disk along the surface of $\mathcal{B}$. Therefore the resulting $\mathcal{CB}$ is the obstacle dilated by the radius r (Fig.3.6)
.

Another example displays $\mathcal{CB}$ when $\mathcal{A}$ and $\mathcal{B}$ are convex polygons. $\mathcal{A}$ can translate freely but can´t rotate. Therefore $\mathcal{C}$ is $\mathcal{R}^2$ and for $\mathcal{CB}$ we get the

[8]This is sometimes called the *Point Robot Assumption.*

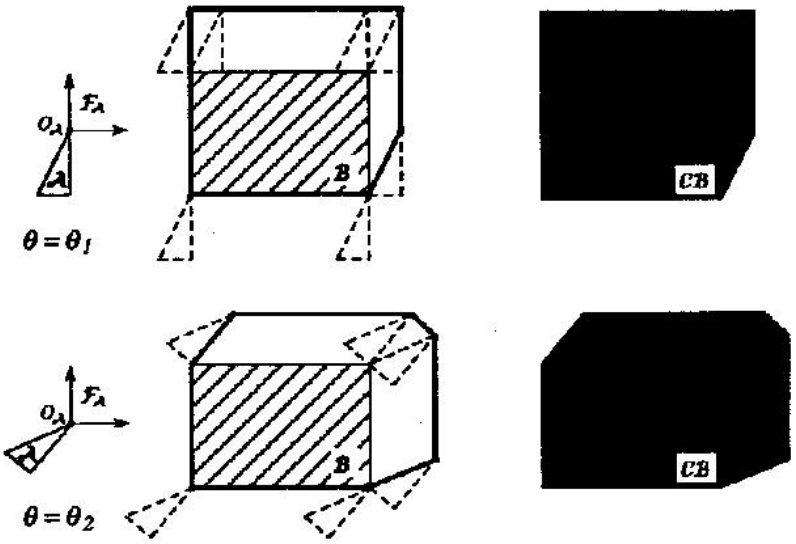


Fig. 3.7.: 2 configurations of $\mathcal{A}$ the related C-Obstacles [31].

shapes of (Fig.3.7). It´s important to mention that for each orientation of $\mathcal{A}$ we have a different shape of the C-obstacle.

As a last example we let $\mathcal{A}$ not only translate but also rotate freely. In that case $\mathcal{CB}$ complicates dramatically an will become a volume as we now have to introduce a third dimension ($\mathcal{R}^2 \times \mathcal{S}^1$ (Fig.3.8).

The most simple way to formulate an algorithm for computing the Minkowski sum of two convex polygons is to simply apply the definition 3.2.1 and compute $\vec{a} + \vec{b}$ for all pair $\vec{a}, \vec{b}$ of vertices and finally built the convec hull of all these sums. We want to give an alternative algorithm which is much faster and easy to implement [28]:

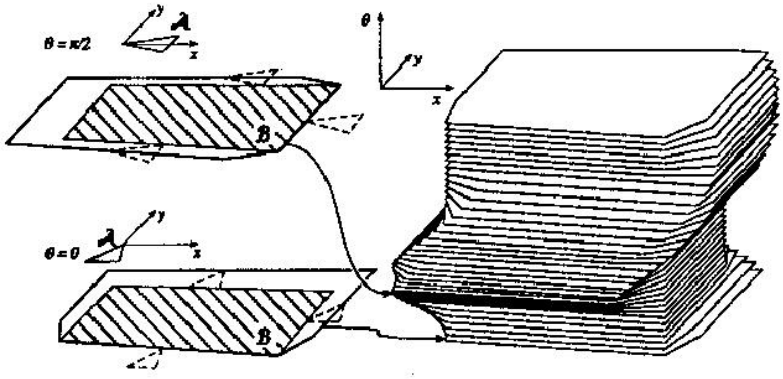


Fig. 3.8.: The free flying object $\mathcal{A}$ is also allow to rotate [31].

Algorithm 3.2.1 ***: MinkowskiSUM***

Input *:*

2 CONVEX POLYGONS $\mathcal{A} = v_1, \ldots, v_n,\ \mathcal{B} = w_1, \ldots, w_m$;

Output *:*

MINKOWSKI SUM $\mathcal{A} \oplus \mathcal{B}$;

01: $i \leftarrow 1,\ j \leftarrow 1$;
02: $v_{n+1} \leftarrow v_1$, $w_{m+1} \leftarrow w_1$;
03: ***DO***
04: $(i \neq n+1\ AND\ j \neq m+1)$;
05: *Add* $v_i + w_j$ *as vertex to* $\mathcal{A} \oplus \mathcal{B}$;
06: ***IF***
07: $angle(v_i, v_{i+1}) < angle(w_j, w_{j+1})$
08: $i \leftarrow (i+1)$;
09: ***ELSE IF***
10: $angle(v_i, v_{i+1}) > angle(w_j, w_{j+1})$
11: $j \leftarrow (j+1)$;
12: ***ELSE***
13: $i \leftarrow (i+1)$;
14: $j \leftarrow (j+1)$;
15: ***WHILE***
16: $i = (n+1)\ AND\ j = (m+1)$

The subroutine $angle(vw)$ denotes the angle of the vector $v\vec{w}$ with the positive x-axis.

We conclude the short survey about the Minkowski sum with to important theorems (without proof):

Theorem 3.2.1 *The C- obstacle of $\mathcal{B}$ is $\mathcal{B} \oplus -\mathcal{A}(0)$, where $-\mathcal{A} = -\vec{a} : \vec{a} \in \mathcal{A}$ means, that $\mathcal{A}$ is reflected at the origin.*

Theorem 3.2.2 *Let $\mathcal{A}, \mathcal{B}$ be two convex polygons with n and m edges. Then $\mathcal{A} \oplus \mathcal{B}$ is a convex polygon with at most $n + m$ edges*[9]*.*

3.3. Representing Space

An important goal in developing intelligent agents capable of complex planning tasks is that they have some internal representation of their environment. Such an internal map is necessary for reasoning about navigation, recognizing special regions or objects. We can divide such maps in different levels of abstractions:

- **Spacial Information**, is a straightforward representation of the space. The idea is to represent the space (2D or 3D) as a grid, where the pixels represent either a free space ($'0'$) or an obstacle ($'1'$). The big advantage is that there is no strong assumption of the objects in the environment, the disadvantage is that the storage space depends strongly on the resolution of the grid[10].

[9] and can be computed in $\mathcal{O}(m + m)$ time.

[10] E.g. in a $15 \times 15 \times 15$ m^3 environment with a grid length of 3 m, 125 cells are needed, whereas in a $100 \times 100 \times 100$ m^3 environment with a grid length of 1 cm, 10^9 cells are needed. Using more advanced data structures like Quadtrees or BSP [31] trees reduce the storage space.

- **Geometric information**, which is the direct product of sensor data and important for local planning tasks and collision avoidance. Geometric maps are build up of discrete geometric primitives like points, lines, splines or surfaces.
- **Topological information**, is an abstract representation of the environment,which lacks any metric data. The obstacles and the free space between it is represented as nodes an edges of a graph or tree. Such a representation is the requirement for a global path planner.
- Semantic information, is the ability not only to detect objects as geometric forms but also to understand their meaning. This is the most advanced level of abstraction an we are far away from satisfying results.

3.4. Basic Motion Planning Problem

The main ideas of the configuration space as displayed in section 3.2 can be summarized in a recipe:

- Represent the robot $\mathcal{A}$ as point in the configuration space.
- Map all the obstacles of the working space into this space.
- Find an optimal path free path from the initial to the goal configuration.

With the introduction of the configuration space we transform the basic motion planning problem from planning of object motion to of planning point motion. This is a completely geometric problem and a lot of methods have been developed within the last 2 decades in the field of algorithmic geometry. These methods are also referred to as exact algorithms, as in any cases they will find a solution to the problem or at least will report that no solution exists. In the following sections, we describe the most

important of them and analyse their strength and weakness [31], [34], [33].

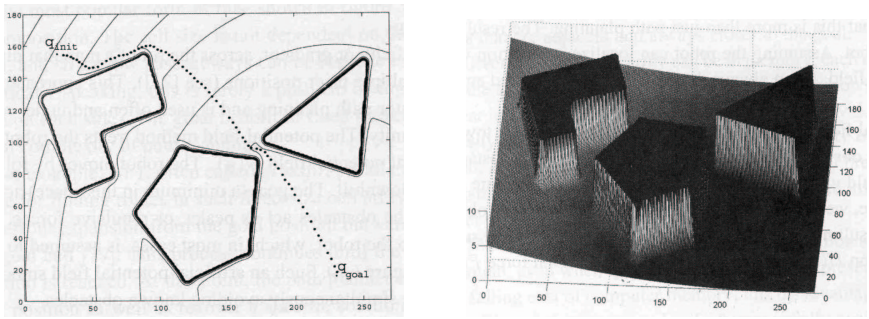
3.4.1. Potential Field Method

The key idea of the potential field method [35] is to treat the robot as a charged particle acting under the influence of external potential fields U. The obstacles carry the same charge as the robot and therefore cause repulsive forces, while the goal point is inversely charged an acts therefore as an attractor (Fig. 3.9). Such an artificial potential can be written as

$$U(\vec{q}) = U_{goal}(\vec{q}) + \sum_i U_i(\vec{q}) \ .$$

From the potential we can derive the artificial force, which is responsible for the trajectory of the robot

$$F = -\nabla U(\vec{q}) = -\begin{pmatrix} \partial U/\partial x \\ \partial U/\partial y \end{pmatrix} \ .$$



(a) Equipotential lines and the path of the robot.

(b) The resulting potential field.

Fig. 3.9.: The potential field method [34].

The remaining task is to find adequate definitions for the potentials U. A common choice for U_{goal} is

$$U_{goal}(\vec{q}) = \alpha \|\vec{q} - \vec{q}_{goal}\|^2 \ ,$$

and for U

$$U(\vec{q}) = \beta \|\vec{q} - \vec{q}_{obstacle}\|^{-1} \ ,$$

where $\| \cdot \|$ denotes the Euclidean distance between the robot in state $\vec{q}$ and the closest point on the obstacle. In order to reduce complexity the repulsive forces are only calculated for the next obstacles instead of summing up all influences. This method has a lot of special advantages:

- spatial paths are not preplanned and can be generated in real time ,
- the generated paths are smooth ,
- planning an control are directly coupled, which simplifies the control algorithm .

The potential field method suffers a major setback if the obstacle has a concave shape, because the robot can be trapped into the local minima of that potential function. A lot of improvements of the classical potential field method has been developed and implemented in robotic systems [36], [41], especially to improve the local minima behavior, but also to reduce the oscillations when passing a narrow space like a door. An interesting approach is the **randomized path planner** (RPP) [49], where the escape from local minima is managed with the help of random walks.

Nevertheless due to the problems of local minima, potential field planner will more often be used as local planner, while global planning uses graph-like methods.

3.4.2. Roadmap

The central idea behind this method is to plan all possible paths in a given two- dimensional configuration space with random distributed C-obstacles. The nodes of that non- directed graph are the initial and goal configuration and the vertices of the polygons. The nodes are connected with those links, that don´t intersect the interior of a polygon. Once the road network is build one can search for a suitable path between two points.

Visibility Graph Method

The earliest Roadmap planning method is the **visibility graph method** [38]. In Fig.3.10 we find a possible scenario, an initial and a final point and C- obstacles between it. The next step is to build up the visibility graph G = (V,E) (Alg. 3.4.1) as mentioned before . The problem of finding the shortest path is now reduced to finding the shortest path from the start to the end node in the graph. There exists a lot of efficient algorithms for searching in the graph of n nodes, e.g. the **Shortest Path Algorithm** of Dijkstra [30] or the A^* Algorithm [68], which leads to an overall complexity of $O(n^2)$. The general visiblity graph algorithm works in the following way:

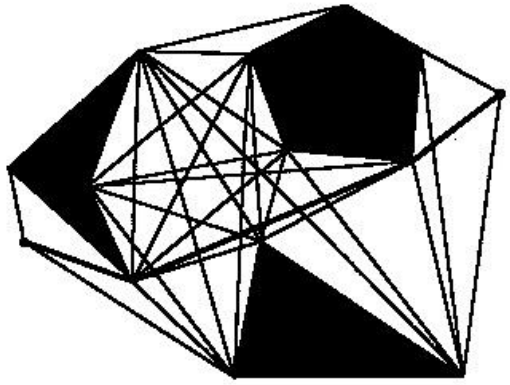


Fig. 3.10.: Visability Graph

Algorithm 3.4.1 ***: Simple visibility Graph Algorithm***

Input *:*
$V \leftarrow$ *all obstacles vertices in* $\mathcal{C}$;
$V \leftarrow$ *start and goal position* ;
Output *:*
shortest path in $G = (V, E)$;

01: ***FOR*** *every pair of nodes* $u, v \in \mathcal{C}$
02: ***IF*** *(edge* (u, v) *is an obstacle edge then)*
03: *insert* (u, v) *into* $G = (V, E)$;
04: ***ELSE***
05: ***FOR*** *every obstacle edge* $\{e\}$;
06: ***IF*** $\{(u, v)\} \cap \{e\} \neq \emptyset$
07: *then* ***GOTO*** *02* ;
08: ***ENDIF***
09: *insert* (u, v) *into* $G = (V, E)$;
10: ***ENDIF***
11: ***ENDFOR***
12: *Search in* $G = (V, E)$ *using A* or Dijkstra* ;

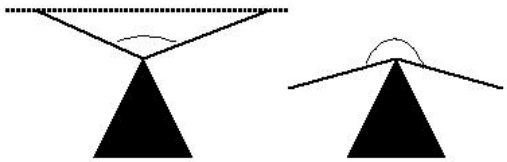


Fig. 3.11.: Edges with an angle $< 180°$ can be replaced by a shorter edge.

Before using the algorithm Alg. 3.4.1 it is advisable to remove the unnecessary edges from the graph as can be seen in Fig. 3.11. All edges with an included angle of $< 180°$ can be removed as there is a shorter connection possible.

The visability graph method is **complete**, e.g. if there exists a shortest path, it will be found. The disadvantage is, that the possible paths pass through the vertices and that the algorithm doesn't make any assumptions of the size of the robot. Therefore such paths don't have a safety distance to obstacles, but this is necessary if the robot moves through an environment, where it has to take care of them (e.g. if humans are walking around). It is possible to overcome this problem at least in part by applying the Minkowski sum to dilate the obstacles. However, for non circular robots this method is not complete, that means the path planner can't find a path even if there exits one.

Voronoi Diagrams

To overcome the problem of approaching too close to the obstacles, a well known technique from the field of computational geometry can be used, the **Voronoi diagram** [39]. The general idea of Voronoi diagrams is to maximise the clearance of fixed points, called sites. For the purpose of robot motion planning the sites will be enlarged to polygonal obstacles and we are searching for the maximal clearance between the robot and

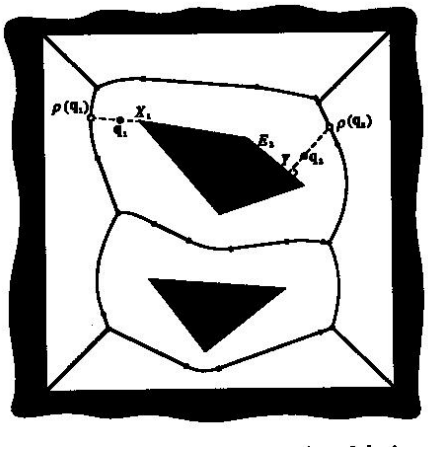


Fig. 3.12.: Voronoi Diagram [31].

the obstacles. The Voronoi diagram (also called **generalized** Voronoi diagram) is then defined as the locus of points equidistant to the closest two or more obstacle boundaries [37]. The paths between the edges of the obstacles are straight lines and around the corners they are parabolas. This leads to a path as shown in Fig. 3.12.

In analogy to the visibility graph, the Voronoi algorithm is complete but the generated paths are longer as they include curved segments.

The property of a Voronoi planner, that it maximizes the distance to the obstacles should be regarded when short-range sensors are used, because they will mostly fail to detect the surrounding environment which leads to poor results in the case of self localization.

3.4.3. Cell Decomposition

The vertical cell decomposition method divides the configuration space into areas, or cells, that are free and areas that are occupied by obstacles (Fig. 3.13). At each edge of the bounding box or of an obstacle,

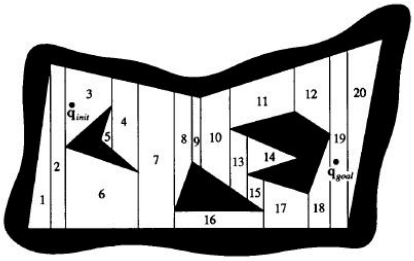


Fig. 3.13.: Vertical Cell Decomposition of $\mathcal{C}_{free}$.

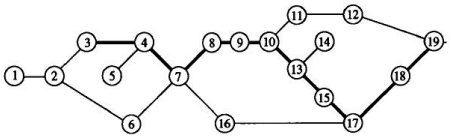


Fig. 3.14.: The resulting topological graph.

a vertical line is drawn upwards and downwards until an neighbored obstacle or the bounding box of the workspace is hit. If the union of the trapezoidal and triangular cells is exactly the free space, then we call it an **exact cell decomposition** otherwise an **approximate cell decomposition**.

After $\mathcal{C}_{free}$ is partitioned into n cells $\mathcal{C}_i$ a sample point $p_i \in C_i$ is chosen in each cell. The sample points can be selected at the centers of the cells or randomly. Then the so-called **connectivity graph** G = (V,E) is constructed as follows : Each of the sample points is a vertex of the graph and two adjacent vertices will be connected by a straight line path, which is the edge. The accessibility condition is satisfied because every sample point can be reached by a straight line due to the convexity of every cell. The resulting path τ passes all the points $p_0, p_1, \ldots, p_n$, where $\tau(0) = p_I$ and $\tau(1) = p_G$.

In analogy to the visibility graph the cell decomposition method fails, when the dimension of the configuration space gets higher or when the complexity of the scene is large, as the number of cells required becomes too large to be practical. It has been shown, that many general formulations of general motion planning problems are PSPACE-hard[11] and

[11]This means, that there is no deterministic algorithm, which needs no more than polynomial amount of storage space during the execution. A famous example for a

therefore there is no hope of finding a complete and universally valid solution. Nevertheless there are at least two good reasons to study such exact methods

- It often happens, that we are only interested in a special class of planning problems, e.g. only a translation of the robot or an environment with only a few obstacles. In that cases there exists a lot of special variants of roadmap or potential field based algorithms, which are fast, elegant and complete.
- From a theoretical point of view, it is both interesting and satisfying to know that there are complete algorithms for an extremely broad class of motion planning problems. Thus, even if a problem exceeds the limiting assumptions for exact planning algorithms, they provide both a basis for an approximate solution and theoretical upper bounds on the time and storage-space needed.

3.5. Sampling-based Planner

As the classical exact planning methods are limited to restricted problems, the focus of research switched to **sampling-based**[12] methods. They demonstrate a tremendous potential in solving many challenging high-dimensional problems efficiently at the expense of completeness. However, such planners can achieve a weaker form of completeness, the so called **probabilistical completeness**. This means, that if a solution path exists, the planner will eventually find it with a probability that converges to one when enough points are sampled. Current heuristic planning algorithms can be divided into two classes: **multiple-query** methods, where a generated roadmap can be repeatedly reused if the environment is static and **single-query** methods, where the $\mathcal{C}_{free}$ is perambulated for each query anew. The most prominent representative for the

PSPACE-hard problem is the **piano mover's problem** [40].

[12]They are also often called **heuristic** or **probabilistic** methods.

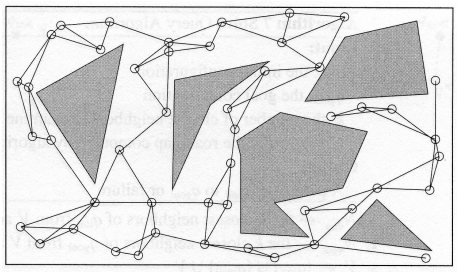


Fig. 3.15.: A roadmap for a point robot.

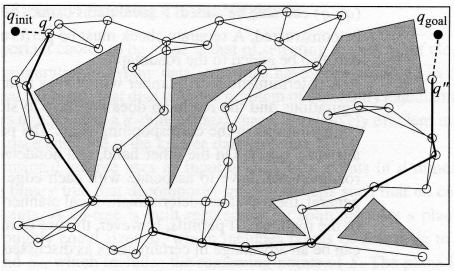


Fig. 3.16.: The calculated path of the PRM planner [34].

former method are **probabilistic roadmap planner**, while for the latter the method of **rapidly-exploring random trees** is getting increasingly important.

3.5.1. Probabilistic Roadmap Planner

The **probabilistic roadmap planner** (**PRM**), developed independently at different groups [42],[43],[45],[44], is a roadmap method where the path isn't generated in a deterministic but in a stochastic way.

The key idea is to choose a collection of random configurations in $\mathcal{C}_{free}$, which are the nodes of a graph G = (V, E). Then a number of pairs of those configurations will be connected by a path using a simple local motion planner. It is typical to choose an simple straight line interpolation between two configurations and check if this path is collision free. If this is the case than it will be added as an edge to the graph. As a result of the procedure Alg.3.5.1 (this is called the learning phase) we get a connected graph of $\mathcal{C}_{free}$ as shown in Fig. 3.15. In the following query phase, the start configuration s and the goal configuration g are added to V. Then a graph search tries to find the shortest possible sequence of edges between s and g and which will afterward be transformed into a feasible path in $\mathcal{C}_{free}$ (Fig. 3.16).

Algorithm 3.5.1 ***: PRM Algorithm***

Input *:*

n : number of nodes to put in the roadmap

k : number of closest neighbors to examine for each config.

Output *:*

A roadmap G = (V, E)

01: $V \leftarrow 0$, $E \leftarrow 0$;
02: ***WHILE*** *(*$|V| < n$*)*
03: $c \leftarrow$ *a (random) configuration in* $\mathcal{C}_{free}$;
04: $V \leftarrow V \cup \{c\}$;
05: ***END WHILE***
06: ***FOR ALL*** $c \in V$
07: $N_c \leftarrow k$ *closest neighbors to* c *according to* $|\cdot|$;
08: ***FOR ALL*** $c' \in V$
09: ***IF*** $(c, c') \notin E$ ***AND*** *local planner finds way betw.* (c, c') ;
10: $E \leftarrow E \cup (c, c')$;
11: ***END IF***
12: ***END FOR***
13: ***END FOR***

There are a lot of details to fill in this abstract scheme of algorithm 3.5.1: what are the optimal values for n and k, which local planner should be used, which is the best distance measure, how to select promising pairs of nodes to connect, etc. This is on the other hand a big advantage of the PRM, because it can be used for different classes of problems and can be generalized to a lot of special constrained systems. A common difficulty of the PRM planner the so-called **narrow passage problem**. This means, that the probability of random samples falling inside a small passage is very low. Therefore there are only little edges connecting the

nodes, which might lead to a discontinuous roadmaps. There exists several approaches to cope with that problem like sampling on the $\mathcal{C}_{free}$ boundary [50], [42], Gaussian sampling [51] or Bridge-test sampling [52], etc[13].

3.5.2. Rapidly-Exploring Random Trees

While PRM Planners have some difficulties with nonholonomic systems as they need a huge amount (typically tens of thousands) of connections between pairs of configurations, the method of **Rapidly-Exploring Random Trees (RRT)** [48], [46],[47] was originally developed for motion planning tasks under differential constraints. The key idea is to incrementally construct a search tree that gradually improves the resolution (Fig.3.17), but does not need to explicitly set any resolution parameters. The principle construction scheme of a RRT is to start from x_{init} and incremently add new vertices to the tree $\mathcal{T}$ that are biased by a randomly selected state (Alg.3.5.2). As shown in Fig.3.17 a random state x is chosen and the $EXTEND$ function (Alg.3.5.3) selects the nearest state $x_{near} \in \mathcal{T}$ to x. As in the case of PRM the meaning of "'nearest"' depends on the chosen metric ρ. The function $NEWSTATE$ makes a motion toward x. This motion depends on the **state transition function** $\dot{x} = f(x, u)$, where $\dot{x}$ denotes the derivative of the state with respect to the time and $u \in U$ is a element of the **input** set U. Integrating over a fixed time interval Δt we get for the new state $x_{new} \approx x + f(x, u)\Delta t$. The use of a state transition function allows a great flexibility regarding constraints. For holonomic planning, a useful definition is $f(x, u) = u$, with $\|u\| < 1$, while for nonholonomic planning, the next state is constrained due to the special choice of f. The $NEWSTATE$ function also includes a collision detection function to determine if a new state is element of $\mathcal{C}_{free}$ or not. If it is successful, the new state x_{new} is added to the tree $\mathcal{T}$. There can occur three possible situations :

[13] See e.g. [34] for further sampling strategies.

- **Reached**, i.e. x_{new} reaches x .
- **Advanced**, i.e. $x_{new} \neq x$ will be added to RRT .
- **Trapped**; i.e. $NEWSTATE$ fails to produce a state x_{new}.

As the probability for choosing a node for extension is proportional to the volume of its Voronoi region, the RRT tends to rapidly grow in the unexplored regions of $\mathcal{C}_{free}$.

Algorithm 3.5.2 ***: Build RRT Algorithm***

Input *:*

x_{init} : *root of the three*

n : *number of attempts to expand the tree*

Output *:*

$\mathcal{T} = (V, E)$, *rooted at* x_{init} *and has* $\leq n$ *configurations*

01: $V \leftarrow \{x_{init}\}$, $E \leftarrow 0$;

02: ***FOR*** $(i = 1$ *to* $i < n)$

03: $x_{rand} \leftarrow RANDOMSTATE()$;

04: ***END FOR***

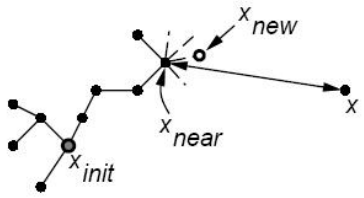


Fig. 3.17.: The $EXTEND$ function [33].

Algorithm 3.5.3 ***: Extend RRT Algorithm***

Input *:*

$\mathcal{T} = (V, E)$: *a RRT*

x : *randomly selected state biasing the growth of* $\mathcal{T}$

Output *:*

x_{new} : *a new configuration toward* x *or failure if* $x_{new} \neq x$

03: $x_{near} \leftarrow$ *closest neighbor of* $x \in \mathcal{T}$ *according to metric* ρ*;*
04: ***IF*** *(*x_{new} *is collision free)*
05: $V \leftarrow V \cup \{x_{new}\}$ *;*
06: $E \leftarrow E \cup \{(x_{near}, x_{new})\}$ *;*
07: ***IF*** *(*$x_{new} = x$*)*
08: ***RETURN****(Reached)* *;*
09: ***ELSE***
10: ***RETURN****(Advanced)* *;*
11: ***END IF***
12: ***END IF***
13: ***RETURN*** *(Trappped)* *;*

The key advantages of RRT's are :

- The expansion of RRT is heavily biased towards unexplored regions

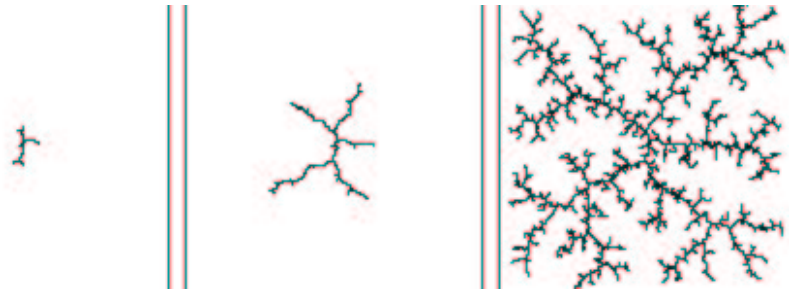


Fig. 3.18.: A RRT explores $\mathcal{C}_{free}$ [33].

of the state space .

- It has been proven that RRT is probabilistically complete under very general conditions.
- RRT are relatively simple, leading to a good performance .
- RRT always remain connected in $\mathcal{C}_{free}$.

Even though RRTs work well in many applications, they have several weaknesses, which cause them to perform poor in special cases. Therefore several extensions and improvements of the basic RRT has been proposed, like **bidirectional** versions, where two trees are growing from the initial and the goal configurations or **dynamic-domain RRT planner** [53] , which try to control the bias of the nodes in the tree near boundaries in order to improve the behavior in narrow passages. Further more there are recent works, which try to combine PRM with RRT in order to create highly efficient parallelized motion planners [55], [54].

3.5.3. Deterministic Sampling

In important question that arises in the field of sampling based planning methods is, whether randomization offers any advantages when applied to sampling strategies. This question seems absurd due the success of randomized path planning methods, however it has been shown [57], [58],

that **deterministic sampling** has some advantages over random sampling. Many PRM planners show a better performance by concentrating samples in a nonuniform way, e.g. along C-space boundaries [42] or medial axis [56]. Thus the key idea is to consider sampling as an optimization problem in which a set of points should be placed in $\mathcal{C}_{free}$ in an 'optimal' way. Therefore it seems, that the success of sampling-based motion planner over earlier combinatorial ones are primarily due to the fact that the are sampling-based, not due to the fact that they are usually randomized [59].

4. Genetic Algorithms

4.1. A brief history of Evolutionary Computation

Evolutionary Algorithms (**EA**) are a class of global, parallel, stochastic and robust optimization methods founded on the Darwinian principle **survival of the fittest**. They are applied successfully to a wide range of technical and scientific problems, especially when due to complexity more classical optimization methods fail. The first attempts for using biological computational principles go already back to the 50's [65], however since the 80's evolutionary principles are an intensive research topic in the field of Computer Science and Engineering. Basically the field of Evolutionary Computing is divided into four areas:

- Evolution Strategies (**Evolutionsstrategie**) [65] ,
- Evolutionary Programming [64], [62] ,
- Genetic Algorithms [69] ,
- Genetic Programming [71] .

For a long time those four fields evolved separately even though they have a mutual basis. With the Workshop 'Parallel Problem Solving from Nature' in 1990 [80] the boundaries had broken down and the various

topics begin to fuse to the research field **Evolutionary Computation**. In this context a lot of new techniques and improvements of the classic EA have been developed in the last few years [73], e.g. **memetic algorithms**, **coevolution**, ant **colony optimization**, **differential evolution**, **particle swarm optimization**, **cultural algorithms** and the combination of **neural networks** and **fuzzy logic** with evolutionary algorithms (**Computational Intelligence** [78]).

The usage of EA is reasonable in all those cases, where more traditional optimization methods fail or are too slow. Especially if the search space is very large and riddled with many local optima EA provide excellent results. Due to the fact that the general design of an EA is non problem-specific[1] there is a huge field of applications to practical problems, e.g.

- Robotics [86] ,
- Control System Engineering [81] ,
- Routing and scheduling [76] ,
- Surface Reconstruction [78] ,
- Production Planning [83] ,
- Automatic evolution of computer software [71] ,
- Maschine Learning [66] ,
- Traveling Salesman Problem [73] .

In the following sections we describe the most important aspects of EA. Although all kinds of EA have a basic structure in common, we will focus on Genetic algorithms as they are used for motion planning tasks in the next chapter.

[1] That's only half of the truth, because implementing of expert knowledge accelerate the performance of EA enormously.

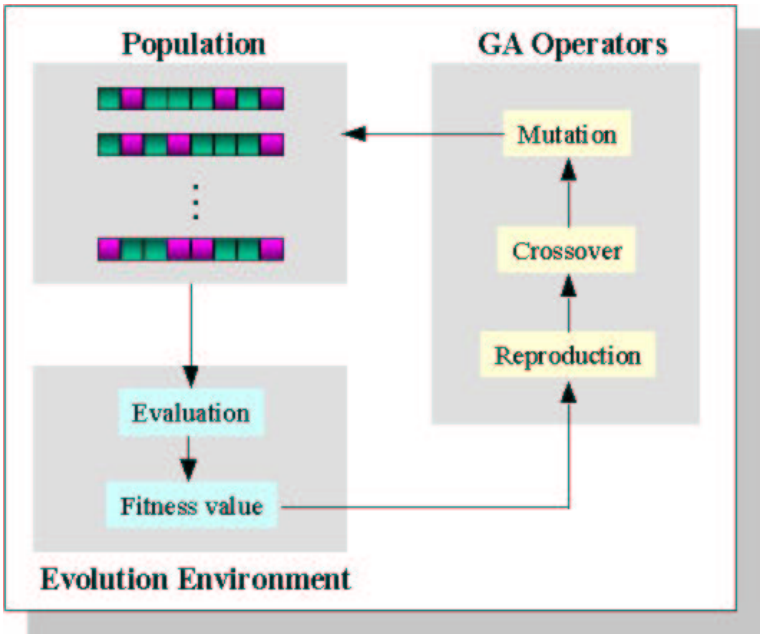


Fig. 4.1.: Scheme of a Genetic Algorithm

4.2. Elements of an Genetic Algorithm

The common underlying idea behind all EA is to copy the natural process of evolution in order to improve the fitness of a population. In other words, a starting population is given and environmental pressure shall improve the fitness until a satisfying solution is found. As we know from nature, evolution uses the operations of **selection, reproduction** and **mutation** (Fig. 4.1) to maximize the fitness of a population. The general scheme of an EA is shown in a pseudo code formulation in Alg. 4.2.1. It can be seen that such an algorithm falls in the category of **generate-and-test** until some optimum is reached. The evaluation function is a heuristic estimation of solution quality and the optimization process is driven by the evolutionary operators, which are applied with a certain probability.

Algorithm 4.2.1 ***: General Genetic Algorithm***

Input*:*

FITNESS FUNCTION $Fit(x)$

POPULATIONSIZE , CHROMOSOMES LENGHT ;

MUTATIONPROBABILITY, SELECTIONPROBABILITY ;

Output *:*

OPTIMAL INDIVIDUAL OF THE POPULATION $P(t)$

01: $t \leftarrow 0$;

02: *Create a Starting Population* $P(t)$;

03: $evaluation(P) \leftarrow Fit((P(t))$;

04: ***WHILE*** *(*$evaluation(P)$ ***NOT*** SATISFYING*)*

05: {

06: $P'(t) \leftarrow$ ***Selection*** $(P(t))$;

07: $P''(t) \leftarrow$ ***Recombination*** $(P'(t))$;

08: $P'''(t) \leftarrow$ ***Mutation*** $(P''(t))$;

09: $evaluation(P) \leftarrow$ FITNESS$(P(t))$;

10: $t \leftarrow t + 1$;

11: }

4.3. Coding

The starting point of an EA is the representation of the candidate solutions[2] as a **chromosome** or **gene**. Typically these candidates are represented by strings over a finite alphabet in Genetic algorithms, real-valued vectors in Evolution Strategies or trees in Genetic Programming. In a more formal language we call that representation a (bijective) mapping

[2]A candidate solution is a member of a set of possible solutions to a given problem.

φ from the search space (*Phenotype*) to the 'representation space' (*Genotype*) of the EA. The most common genotype representation of Genetic algorithms is standard binary or Gray coding, although integer and real number coding can be much more efficient for some problems. For example, if we plan a path on a grid, we might use the values (0,1,2,3) representing (North, East, South, West). Using binary coding in that example would make the chromosomes two times larger as we need two bits for each direction. The choice of an appropriate representation is a key issue in EA, because the whole optimization algorithm directly works in the genotype space. The length of each chromosome depends on the specific problem and can be chosen freely. Besides a fixed genome length, there are a lot of applications, where a variable length is more efficient as it reduces unnecessary operations at parts of the chromosome, which are irrelevant for the solution. As we will see in the next chapter, path planning is much more efficient when using variable length chromosomes.

The next question to answer is the choice of an appropriate initial population size. On the one hand, if the population size is too small, the algorithm may converge too quickly and could be captured in a local optimum. On the other hand, a large population size might cost too much CPU time as the number of operations explode. If there doesn't exist any problem specific knowledge the first generation of the n chromosomes will in general be chosen randomly.

In Tab.4.1 we find a typical example of a population consisting of m binary coded chromosomes of the length l. If their counterparts of the phenotype space are e.g. real numbers, then we have a mapping $\varphi : \mathbb{R} \rightarrow \mathbb{B}$.

In many cases where we have to transform binary coded strings $x = A_1 A_2 \dots A_l \in \mathbb{B}^l$ into real valued numbers $x_{rv} \in \mathbb{R}$ within the range $[lb, ub]$ we can use the following formula:

$x_1 =$	1	0	...	0	1
$x_2 =$	0	0	...	1	1
...	...	...	...	...	...
$x_n =$	1	1	...	0	0

Table 4.1.: A population with n binary coded chromosomes of the length m.

$$x_{rv} = lb + \frac{ub - lb}{2^l - 1} \sum_{j=0}^{l-1} A_{l-j} 2^j \ , \tag{4.1}$$

The accuracy of the real values x_{rv} depends therefore on the number of bits l and the size of the interval. In many cases GA are most successful when the encoding used is closed to the problem, i.e. a 'small' change in the phenotype space should implicate a small chance in the genotype space. Given a genotype with $l = 4$ for example and a phenotype $r \in [0, 15]$, we find that

$$0111 \doteq 7 \ .$$

In order to increase this phenotype number by one, all genotype numbers have to undergo changes

$$1000 \doteq 8 \ .$$

Cray Coding remedies this problem by ensuring that any adjacent points in the phenotype space are also adjacent in the genotype space.

Definition 4.3.1 ***(Gray Coding)***
Given a standard binary coded bitstring $A = A_1A_2...A_l \in \mathbb{B}^l$. The Gray coded bit string $A = A_1A_2...A_l \in \mathbb{B}^l$ arises from

$$B_i = \begin{cases} A_i & i = 1 \\ A_{i-1} \ \oplus \ A_i & i > 1 \end{cases}$$

where $\oplus$ denotes the XOR Operator.

To change a Gray Coded bitstring B into a standard binary bitstring C one has to apply to formula

$$C_i = \oplus_{j=1}^{i} B_j \ .$$

4.4. Fitness

In analogy to the Darwinian principle "survival of the fittest" only those pairs of chromosomes should reproduce, which continually improve the fitness of the best solution at each generation, as well as the average population fitness. A quality measure of an individual's fitness is given by the **objective function**. The selection probability for reproduction will be determined from that objective value of each chromosome and the objective values of the whole population. This measure is called the **fitness value** of an individual and the **fitness** or **evaluation function** transforms the objective values into the fitness values. To clarify this subtle distinction between the objective function $f(x)$ and the fitness function $F(x)$, let us consider a concrete example: The task is to find the optima of a real valued function $g(x)$. The fitness of each individual is measured by inserting the decoded bitstrings (independent variables) into $g(x)$ and looking for their function values. As the fittest individuals are those with the highest numerical values of $G(x)$ (and should therefore obtain the highest probability to survive), we can use $g(x)$ directly as a fitness function. In the opposite case of finding the minima, we can't directly use $g(x)$ as a fitness measure, because the fittest individuals are now those with the lowest values of $g(x)$. Thus, we need a transformation of high objective values into low fitness numbers and vice versa, which could be done e.g. by using $F(x) = -g(x)$.
it is also possible to define fitness measures in non-numerical problems, like finding the correct sequence of amino acids that will fold to a desired three-dimensional protein structure. In that case the objective function could measure the potential energy for each sequence and the fittest chromosome

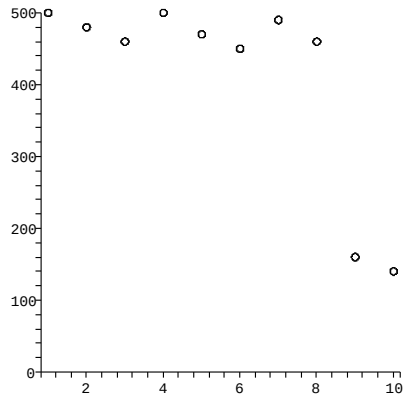


Fig. 4.2.: Two highly fit individuals dominate the population

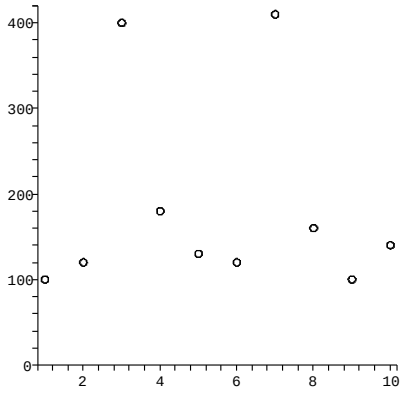


Fig. 4.3.: Many fit individuals hardly differ from each other

could be that one with the lowest potential energy. As one can imagine especially in non-numeric problems the fitness function isn´t unique and finding an appropriate fitness measure can be a challenging task.
In general we can distinguish between two methods of fitness assignment, **proportional** and **rank-based**. **Proportional** fitness assignment means that each individual is assigned a fitness value which is proportional to it´s objective value. Pohlheim [75] specifies the four most important assignments:

linear scaling:	$F(x) = a \cdot f(x) + b$
linear dynamic scaling	$F(x) = a \cdot f(x) + b(t)$
logarithmic scaling	$F(x) = b - \log f(x)$
exponential scaling	$F(x) = (a \cdot f(x) + b)^k$

$a, b, k \in \mathbb{R}$ some problem specific parameters.

The disadvantage of the proportional assignment can be seen in Fig.4.2 and 4.3

Two highly fit individuals (in the case of a minimization problem) in Fig.4.3 dominate the whole population and might cause rapid convergence

to sub-optimal solutions. This is also called **premature convergence**. On the other hand in Fig.4.2 we have little deviation in the fitness values of the population and therefore **no selection pressure** towards the fittest individuals.

To overcome these problems, **rank-based** fitness assignments were suggested [61], where each individual is sorted according to it´s objective value. The fitness of individuals will then be calculated with:

$$F(x_i) = 2 - \sigma + 2 \cdot (\sigma - 1) \cdot \frac{x_i - 1}{N - 1} \ , \tag{4.2}$$

where x_i is the position of the individual i, σ the **selective pressure**, **a real number which is typically chosen in the interval** $[1.1, 2.0]$ **and should determine the bias**, N the population size. Using (4.2) with selection pressure $\sigma = 2$ for example, the best ranked chromosome has fitness value 2 (independent of N) and the worst ranked chromosome has $F = 0$ (so it has no chance of reproduction).

Another possibility of order is a **non-linear ranking:**

$$F(x_i) = \frac{N \cdot X^{i-1}}{\sum_{j=1}^{N} X^{j-1}} \tag{4.3}$$

X can be determined by

$$0 = (\sigma - 1) \cdot X^{N-1} + \sigma \cdot X^{N-2} + \ldots + \sigma \cdot X + \sigma$$

If we compare these ranking methods with $\sigma = 2$ we find that the non-linear ranked fitness values of the better individuals are slightly beyond their linear ranked counterparts. In the case of the worst ranked individuals this effect turns around. The main application range of non-linear ranking is for $\sigma > 2.0$. In figure (4.4) we find for $\sigma = 3.0$ for example, that the fitness of the better is much higher than the fitness of the

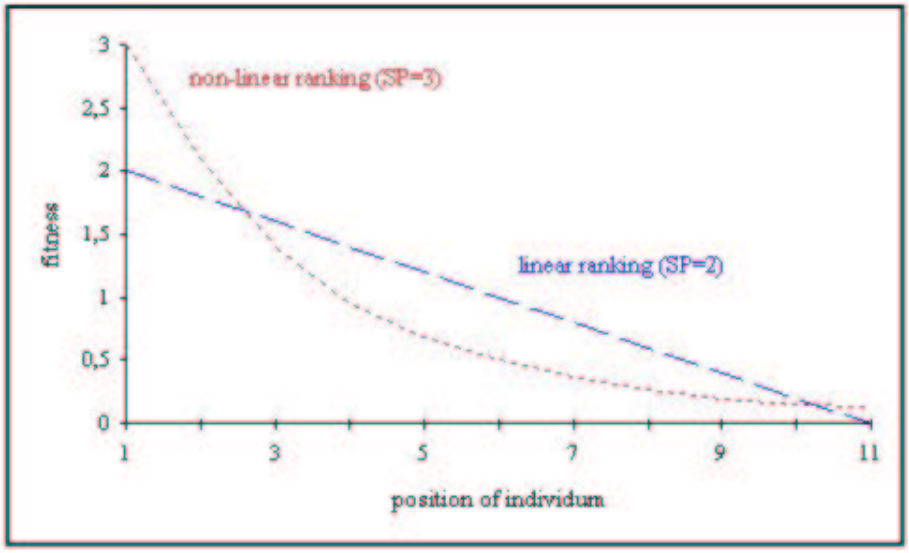


Fig. 4.4.: Linear versus Non-linear Ranking [75].

worse individuals. Thus we have biased the fitness landscape towards the fitter individuals but each one has a positive probability for selection.

4.5. Population

The role of the population is to hold the fittest individuals and bias it towards an optimal solution. It is the population which is responsible for the process of evolution because single individuals are static objects and don't change or adapt. An important aspect is the **diversity** of a population. There are a lot of possibilities to measure the diversity like the number of different phenotypes or genotypes or the entropy. A high diversity means, that a large part of the search space is checked, thus reducing the probability of trapping into local minima. On the other hand we need a bias towards a fitter population, which could be done e.g. with an **elitist** operation, that means the best individual of a current

population is seed to the next generation or the worst individual is chosen to be replaced by another one.

The optimization process of EA is mainly based on three primary operators: **selection, crossover** and **mutation**. While the operation of selection is always applied to the whole population, crossover and mutation act on individuals. Further on, the selection process according to fitness is an **exploitative** resource, whereas the crossover and mutation operators are **exploratory** resources. EA combine the exploitation of past results with the exploration of new areas of the search space and the effectiveness of finding a solution strongly depends on an appropriate mix of exploration and exploitation.

4.6. Selection

Selection is the process of choosing the optimal individuals for reproduction and thus for offspring generation. This can be done in a **probabilistic** way, i.e. each individual can be chosen with a definitive probability or in **deterministic** way, i.e. the best fitness- ranked individuals are chosen. At first sight the deterministic selection method seems to be superior but as already mentioned nature needs diversity and therefore in most cases a probabilistic selection method is preferred. In analogy to the fitness assignment we distinguish between **fitness proportional** selection and **rank based** selection. Among a lot of methods discussed in the literature, there are four prominent selection operations:

- Roulette wheel selection
- Stochastic universal sampling
- Tournament selection
- Truncation selection

To compare these algorithms and discuss their strengths and weaknesses we introduce some measures of performance ([61]):

- **bias:** difference between individuals actual and expected selection probability,
- **spread:** range of possible values of offspring of an individual,
- **selective pressure:** probability of the fittest individuals being selected compared to the average probability of selection of all individuals,
- **Loss of diversity:** proportion of individuals that are not selected.

4.6.1. Roulette wheel selection

This is the simplest kind of selection scheme. The individuals are mapped to contiguous segments of a line, such that each individual's segment is equal in size to its fitness. The relative fitness or selection probability $Pr()$ of an individual x_i is given by the absolute fitness of the individual $F(x_i)$ divided by the fitness of the actual population

$$Pr(x_i) = \frac{F(x_i)}{\sum_{i=1}^{N} F(x_i)} \ . \tag{4.4}$$

As a next step a random number is generated and that individual whose segment spans the random number is selected. The process is repeated until the desired number of individuals is obtained. In the example shown in Tab.4.2 there is a population of 11 individuals with their given relative fitness value and want to choose 6 individuals for offspring production. In comparison we also listed the fitness values of a rank-based fitness assignment[3] (Equ. 4.2).

[3]In that case we have to enlarge the random generator to the interval $[0.0, 2.2]$.

Number of individual	1	2	3	4	5	6
Rank- based fitness value	2.0	1.8	1.6	1.4	1.2	1.0
Selection probability	0.18	0.16	0.15	0.13	0.11	0.09
Number of individual	7	8	9	10	11	
Rank- based fitness value	0.8	0.6	0.4	0.2	0.0	
Selection probability	0.07	0.06	0.03	0.02	0.0	

Table 4.2.:

Fig. 4.5.: Roulette Wheel Selection [75].

For selecting the individuals 6 independent uniform distributed random numbers between $[0, 1]$ are generated, e.q. 0.81, 0.32, 0.96, 0.01, 0.65, 0.42. Fig.4.5 shows the result of the algorithm - the individuals with number 1, 2, 3, 5, 6, 9 has been selected. As we expect, the selection probability is much higher for fitter individuals, nevertheless there is a small (but > 0) probability for choosing an individual with a low fitness value.

The roulette-wheel selection algorithm provides a zero bias but does not guarantee minimum spread because of the usage of random numbers. Although it is quite unlikely it could happen that all trials chose the worst possible individual.

4.6.2. Stochastic universal sampling

Stochastic universal sampling (SUS) overcomes the disadvantage of no guaranteed spread. As in roulette-wheel selection the individuals are

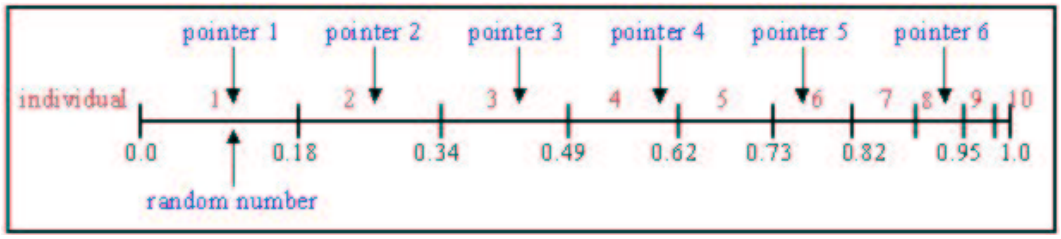


Fig. 4.6.: Stochastic universal sampling [75].

mapped to contiguous segments of a line, such that each individual's segment is equal in size to its fitness. In opposite to the previous method, equally spaced pointers are placed over the line as many as there are individuals to be selected. Let N be the number of elements to be selected, then $1/N$ is the distance between the pointers. The first pointer is given by a random number in the interval $[0, 1/N]$. In our example we want to choose 6 individuals, i.e. the distances of the pointers are $1/6 = 0.167$. A sample of one random number between 0 and 0.167 gives 0.1 and thus we get the individuals $1, 2, 3, 4, 6, 8$ (Figure 4.6).

This method, which needs only one random number, provides a zero bias and guarantees a minimal spread. It is the best known fitness proportional selection method.

4.6.3. Tournament selection

The previous two methods need the knowledge of the entire population in order to select individuals. In some situations, e.g. if the population size is very large or distributed in a special way, obtaining this knowledge is either highly time consuming or impossible. Further more, in game playing strategies for example, we can't quantify the (absolute) strength of a single individual, that is, a particular strategy, but we can compare any two of them. Tournament selection is an operator, which does not require

any knowledge of the global population. Instead of using an ordering relation of the whole population it ranks any two individuals. A certain number q of individuals is chosen randomly from the population and the best individual is selected by comparing their fitness values. This process is repeated as often as the new population is complete. If q is quite large, then there is more chance that individuals of above-fitness will be chosen for the new population. Due to it's simplicity and the fact that the selection pressure can be controlled by q, the tournament selection operator is probably the most popular selection operator in modern applications of Genetic algorithms.

4.6.4. Truncation selection

Truncation selection is directly connected to the rank-based fitness assignment. Each individual is sorted according to it´s fitness and only those one will be selected which exceed a defined threshold. This method selects the best individuals with a probability of 1 and cancels the remaining one.

4.6.5. Elitism

"**Elitism**" is an addition to many selection methods that forces EA to retain some of it´s best individuals at each generation. Such individuals could be lost if they are not selected or could be destroyed by mutation. It has been shown, that for many applications the search speed can be greatly improved by keeping the fittest individuals in the population.

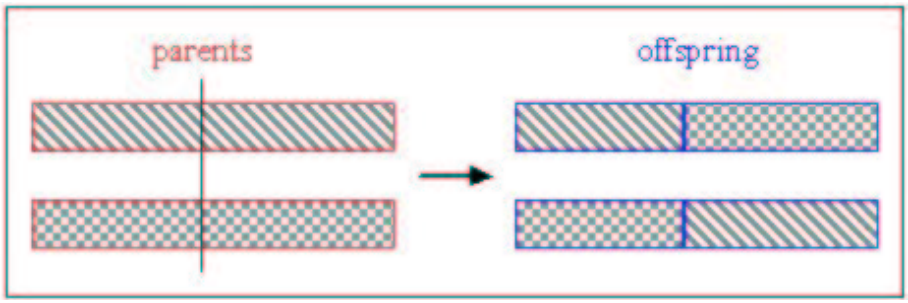


Fig. 4.7.: Single Point Crossover

4.7. Crossover

The basic operation for producing new chromosomes is that of **crossover**. Like in nature the offsprings inherit positive characteristics from both´s parents genetic material and thus improve their fitness. Recombination is usually applied probabilistically according to the so-called **crossover rate**, which is typically in the range of $[0.5, 1.0]$. In general crossover happens between two parents, though their exists versions of three or more parent cases, too.

The simplest of the crossover operator is the **single-point crossover** (Fig. 4.7): A single crossover point is chosen at random in $[0, n-1]$ (n is the length of encoding), splitting both parents into two parts. The offsprings are generated afterward by exchanging the tails of the parent chromosomes.

In that case of **multi-point crossover** m crossover points are chosen at random and sorted in ascending order. Then, the segments between successive crossover points are exchanged in order to produce new offsprings. (Figure 4.8). A further interesting variant is called **uniform crossover**. This method is implemented by generating a string of n random variables. In each position of that string, if the random value is below a threshold, the gene is ihnerted from the first parent, otherwise

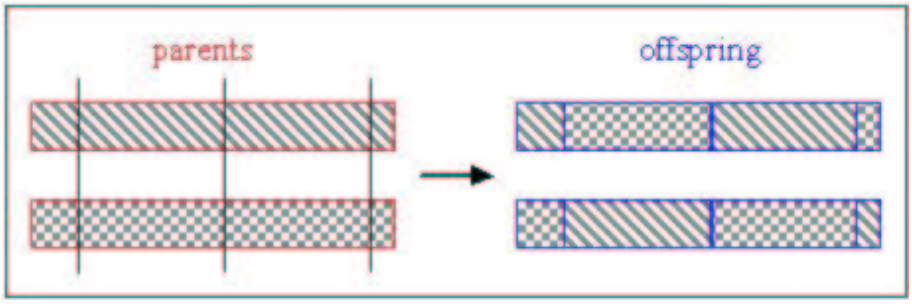


Fig. 4.8.: Multi Point Crossover

from the second. The second offspring is generated by using the inverse mapping.

Besides these two operators there are a lot of other crossover algorithms discussed in the literature, e.g. *Parameterized Uniform Crossover*, *Shuffle*, *Reduced Surrogate* or *Intermediate Recombination*. Which one of these operators performs best depends on the specific problem, particularly if there are known patterns within a chromosome. The randomly mixed offsprings can strengthen or weaken such dependencies and might therefore lead to an undesirable behavior.

4.8. Mutation

In natural evolution, mutation is a random process where one allele of a gene is replaced by another to produce a new genetic structure. In Genetic Algorithms, mutation is randomly applied with low probability, typically in the range 0.001 and 0.01 in opposite to other EA, where mutation is the primary operator for modifications. Although considered as a background operator in GA, the role of mutation is often seen as providing a guarantee that the probability of searching any given string will never be zero and acting as a safety net to recover good genetic material that may be lost through the action of selection and crossover. The effect of

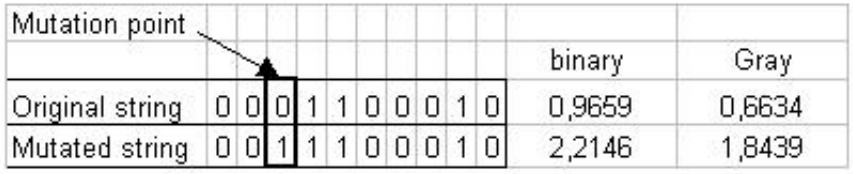


Fig. 4.9.: Mutation of a binary string

mutation on a binary string is illustrated in Figure 4.9 for a 10-bit chromosome representing a real value decoded over the interval [0, 10]. It is important to see the effect of different coding schemes, in our example the standard binary coding and gray coding. In the case of integer- or floating- point representations it is common to change the allele value of each chromosome randomly within it's domain (lower (lb) and upper (ub) bound):

$$< x_1, \ldots, x_n > \rightarrow < \bar{x}_1, \ldots \bar{x}_n >, \; with \; x_n, \bar{x}_n \in [lb, ub]$$

With non-binary representations, mutation is achieved by either perturbing the gene values or random selection of new values within the allowed range.

4.9. Reinsertion and Termination

Once the new population is generated we determine the fitness and start the EA again. If fewer individuals are produced by recombination than the size of the original population, then the fractional difference between the new and old population sizes is termed a *generation gap*. In the case where the number of new individuals produced at each generation

is one or two, the EA is said to be *steady-state* or *incremental*. If one or more of the fittest individuals are deterministically allowed to propagate through successive generations then the EA is said to use an *elitist strategy*.

Because all EA are stochastic search methods, it is difficult to formally specify convergence criteria. A common practice is to terminate the EA after a prespecified number of generations and then test the quality of the best members of the population against the problem definition. If no acceptable solutions are found, the EA may be restarted or a fresh search initiated. In order to avoid premature convergence, operations increase the variance within a population are often introduced. An example is the **prevention of incest** [87], which means, that crossover is only allowed if the difference (e.g. the Hamming distance in the binary case) between two chromosomes is above a threshold.

4.10. Schemata

One of the key ideas in developing EA is to enable a parallel search in the hyperplanes of the search space. The chromosomes that belong to such hyperplanes exhibit a similar structure called a **schema** [70]. A schema is a template that divides the strings into equivalence classes. For example, a 3 bit binary coded chromosome schema $1 * 1$ represents the so-called **instances** 111 and 101. In general, a population of N individuals, with binary coded chromosomes of length l, contains a number of schemata between 2^l and $N \cdot 2^l$. The use of schemata offers two advantages: firstly, they are useful for describing those components of a chromosome that provide high fitness and secondly, they allow the exploration of a much higher number of strings than those contained in the population. Schemata also show the influence of the genetic operators on chromosomes. In the case of templates like $1 * * * * * * 1$, the probability of being broken down by crossover is far higher than in cases of $* * * * 11 * * * *$. A major result

in the theory of EA is , that schemata with short length and high fitness , will disproportionate reproduce. Therefore EA search a solution space quite efficiently. Although the theory of schemata has been criticized a lot in recent years [82], it is a helpful tool for understanding some aspects of how EA work.

4.11. Strengths and weaknesses of EA

As already mentioned at the beginning of that chapter, EA have been successfully implemented in many applications. Thus, we want to summarize the strengths of EA:

- **Parallelism**. In opposite to other search methods like hill climbing or simulated annealing, EA are intrinsically parallel due to the fact, that multiple offsprings are generated, which explore the search space in many directions at once. The parallelism of EA goes even beyond this, when considering the schema theorem. By evaluating a particular string, which is a representant of many schemata, all the corresponding hyperspaces are sampled at the same time. Therefore, a EA that explicitly evaluates a small number of individuals is implicitly evaluating a much larger group of individuals.
- **Multimodal spaces**. Due to the parallel search behavior, EA are predestinated for problems, where the space of all potential solutions is truly huge and cluttered with many local optimas.
- **Information exchange**. As crossover is the key element, candidate solutions exchange information between them, which avoids the problem of searching only in a immediate vicinity without reference to what other individuals may have discovered.
- **Simultaneous parameter manipulation**. Many real-world problems cannot be stated in terms of a single value to be optimized, but

must be expressed in terms of multiple objectives. Due to parallelism EA are well suited to find solutions for such **multiobjective optimization** problems [84], [85].

Nevertheless EA also have some limitations, however, all of these can be more or less overcome and none of them threatens the validity of biological evolution.

- **Robustness**. The definition of an adequate representation of the given problem is a key question in EA. In general there exits a lot of possibilities to represent the candidate solutions, but the chosen representation must be robust; i.e., it must be able to tolerate random changes such that fatal errors do not consistently result.
- **Fitness function**. The choice of the fitness function must be carefully considered so that higher fitness is attainable and leads to a better solution for the given problem. If the fitness function is chosen poorly or defined imprecisely, the EA may be unable to find a solution to the problem, or may end up solving the wrong problem.
- **Parameter**. The choice of the parameters like population size, chromosome length, crossover rate, mutation rate, etc is a tedious procedure as there doesn't exist a formal way how to do that. Although there has been made some progress in recent years, the optimal choice of the parameters remains a procedure of trial and error.
- **Premature convergence**. If an individual that is more fit than most of its competitors emerges in a early generation, it may reproduce so abundantly that it drives down the population's diversity too soon, leading the algorithm to converge on the local optimum. This is an common problem in small populations, where even chance variations in reproduction rate may cause one genotype to become dominant over others.
- **Performance**. As EA are not designed for special optimization tasks, their performance is usually slower than other techniques. In

those cases, where problem specific algorithms exist, they always perform better as they don't waste time for testing the fitness of sub-optimal solutions. Furthermore, due to the stochastic nature of a EA, the solution will only be an estimate and there exists no guarantee that the global optimum will be found.

5. Genetic Path Planning

As we have seen in Chapter 3, the motion planning problem can in many cases be formulated as an optimization problem, like minimizing the path length or the power consumption when driving between a starting and a goal point. Therefore EA are also predestinated for solving complex motion planning tasks in constrained environments. We will show, that Genetic Algorithms are flexible a fast and flexible tool for designing feasible paths. In opposite to the classical paths planners, **Genetic path planners** (**GPP**) don't need an exact representation of the environment as they perform an adaptive search on populations of candidate vehicle's actions. In the following we describe our GPP algorithm as well as some popular genetic planners used in the literature.

5.1. Representation

We consider our path planning problem on a $n \times n$ grid, where each square cell has the 1, which represents the diameter of the robot Nomad200. Using a grid has the advantage, that it is straightforward to represent paths as strings of numbers, denoting the passed cells. However, the generated paths are not smooth and ignore minimal turning radius in the case of car like robots. Therefore the generation of those paths is two-staged: at first the edges of the path are generated, which are a sequence of position vectors, e.g. (x_i, y_i) in the Euclidean plane, and secondly a smooth curve

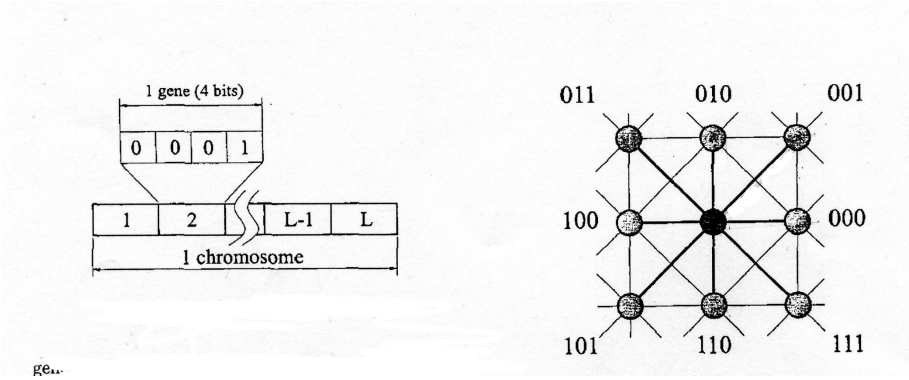


Fig. 5.1.: A 4-bit encoding.

will interpolate these points (e.g. **Spline interpolation**). These smooth curve, which runs from the starting to the end point, respects the kinematical constraints of the vehicle and eliminates the discontinuities at the vertices.

To represent the path in the genotype space can be done in different ways. Using a binary representation [88], [89] we need at least two bits to represent four directions of the plane, e.g **00** $\Leftrightarrow$ $MoveEast$, **01** $\Leftrightarrow$ $MoveNorth$, **10** $\Leftrightarrow$ $MoveWest$, **11** $\Leftrightarrow$ $MoveSouth$. A typical chromosome of a certain length n, then describes the path in each of the denoted direction, starting from the most significant bit, like **[10010011]** , which denotes the four actions of the robot $(MoveWest), (MoveNorth), (MoveEast), (MoveSouth)$.

To enable diagonal paths, too, we have to introduce at least a 3 bit representation, with **000** $\Leftrightarrow$ $MoveEast$, **001** $\Leftrightarrow$ $MoveNorthEast$, **010** $\Leftrightarrow$ $MoveNorth$, etc. In [91] a fourth bit was introduced, which denotes moving or not moving. Thus, we have a representation of the chromosomes as shown in Fig.5.1.

In our path planner we use instead of an binary an integer representation. We use the numbers $(1, \ldots 8)$ to represent the directions, where **1** $\Leftrightarrow$ $MoveEast$, **2** $\Leftrightarrow$ $MoveNorthEast$, etc. This representation has the advantage that the chromosome lengths are much smaller and therefore

less time is needed for separation and recombination in the crossover process.

Another possible representation is a floating point encoding, where the single entries into the chromosome are the (relative) coordinates of the robot's position [90], [92]. In such a case a sequence like $(3.2, 4.3, 1) \rightarrow (5.2, 3.7, 1) \rightarrow (7.2, 4.2, 0), \ldots$, where the third entry is 0 or 1, which provides information on feasibility of the point and the following path or not. This representation has the advantage, that it can faster explore the configuration space as it isn´t restricted to next neighbor movement. Nevertheless it is rarely used, because the classical EA operators, mutation and crossover are note applicable and have to be tailored for that kind of representation.

The next question to answer is the optimal length of the chromosomes. As the length of the path isn't known in advance, a fixed string size would be a severe restriction to the genetic planner. The reason is that a sequence of possible movements, which result in a connected path, vary in length. This happens when the start and goal positions vary or the number and size obstacles change. A further reason to use variable length chromosomes is shown in Fig. 5.2 and 5.3. The task of the robot is to move from the start cell S to the goal cell G, where the path length is fixed. In the picture we find two different paths, one with the sequence of **[3555]**, the other with **[5177]**. As we can see, path 2 never has a chance to reach the goal and will therefore in general be removed although it might have some useful sequences of actions in it.

Thus, variable length chromosomes are much more natural for path planning tasks, although the implementation of some genetic operations is a little bit more tricky. The initialization procedure works now as follows:

A random number $\alpha \in [lb, ub]$ is chosen, which defines the length of the chromosome. The lower bound lb of α is the length l of the bounding box and the upper bound ub is l^2, so that the robot could cover each

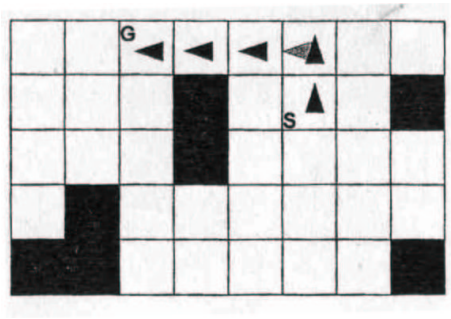


Fig. 5.2.: A path of length 4 achieves the goal point [88].

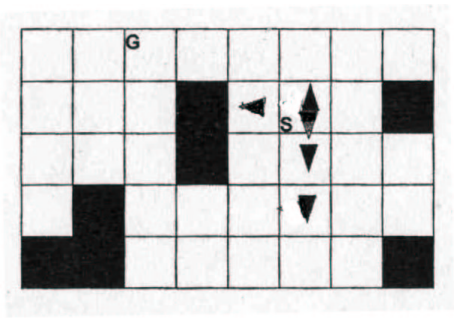


Fig. 5.3.: A path of the same length miss the goal [88].

cell of the grid if necessary. Then, α random numbers between $(1, \ldots 8)$ are generated, defining the chromosome. The box size has been chosen between 10 and 20 units, which result in a chromosome length between 10 and 400. The population size has been set to 60 and the total number of generations is up to 300.

5.2. Fitness function

The key point in designing a GPP is the definition of the fitness respectively the objective function. The measuring of the fitness is even more complicated as we have seen that there exists a lot of parameters of mobile robot tasks, which can be optimized. In that work, we want to optimize the robot's path with respect of the length and the feasibility, e.g. the shortest possible, collision free paths, which achieve the goal point, should be the fittest ones. The objective function will be chosen in such a way, that the fittest individuals are those with the lowest fitness values.

A simple measure is to count each step a chromosome produces and weight it with a damping factor, which should punish those steps which collide with an obstacle ([91]). An example for such a type of function could be

$$f = \sum_{i=1}^{l} d_i \cdot (1 + w_i) \ ; \tag{5.1}$$

where the weight factor w could be defined as follows

$$w = \begin{cases} N \gg 1 & \text{if the cell is occupied by an obstacle} \\ 0 & \text{if the cell is free} \\ M < 0 & \text{if the target is achieved} \end{cases}$$

where $M, N \in \mathcal{N}$ and the distance d_j is designed as

$$w = \begin{cases} 1 & \text{if horizontal or vertical step} \\ \sqrt{2} & \text{if diagonal step} \end{cases}$$

Although the objective function is very simple, it works well especially in environments with a small number of obstacles.

Another possible way to define a fitness function is to include a 'curvature' into the path [90], in order to guarantee a relative smooth movement of the robot. Such a function is designed for floating point encoding, where the single entries are the (relative) coordinates of the robot.

$$f = \alpha \cdot \|p\| + \beta \cdot curv(p) + \gamma \cdot coll(p) \ ; \tag{5.2}$$

where $\alpha, \beta, \gamma \in \mathcal{R}$ are weight factors and

- $\|p\| = \sum_{j=1}^{n-1} d(v_i, v_{i+1})$ is the total distance of the path and $d(v_i, v_{i+1})$ the distance between two adjacent points .
- $curv(p) = max\{s(m_i), i = 2, \ldots n-1\}$ is the maximum 'curvature' at a vertex, where $s(m_i)$ is defined as

$$s(m_i) = \frac{\theta_i}{min\{d(v_{i-1}, v_i), d(v_i, v_{i+1})\}} \ ;$$

with $\theta_i \in [0, \pi]$ is the angle between the line segments (v_{i-1}, v_i) and (v_{i-1}, v_i).

- $coll(p) = max\{c_i, i = 1, \ldots n-1\}$, where

$$c_i = \begin{cases} g_i - \tau & \text{if } g_i \geq \tau \\ e^{a(\tau - g_i)} - 1 & \text{otherwise} , \end{cases}$$

g_i is the minimum of the line segment (m_i, m_{i+1}) to all detected obstacles, τ is a 'safty distance' parameter and $a \in \mathcal{R}$. This term punishes paths with line segments approximating too close to obstacles.

The goal of the evolutionary algorithm is, to minimize this function .

In this work, the fitness function is constructed in the following way: First of all we have to take into account that the fitness of an individual depends on the length of the path and the positional error of the path from the goal position. Therefore we take the weighted sum of these two parts

$$f_1 = \alpha \cdot length + \beta \cdot poserror ,$$

where $length = \sum_{i=1}^{n}(x_{i-1}, x_i)$, the sum of all segments between two edges of the path and

$$poserror = \sqrt{(x - x_g)^2 + (y - y_g)^2}$$

is the Euclidean distance between the actual and the desired position of the path. In the same way we can use the Manhatten distance function for measuring the positional error:

$$poserror = |x - x_g| + |y - y_g| \ .$$

This term is also important to penalize those paths, which cross the goal point due to remaining path sequences in the chromosome. The optimal chromosome should end at the goal point and shouldn't generate any further steps. The two coefficients α and β can be chosen in the following way:

- if $poserror$ is large $\rightarrow \beta \sim \alpha$, e.g. the length of the path and the error of the goal position are equally weighted in the fitness function.
- if $poserror \rightarrow 0 \Rightarrow \beta \rightarrow 1$ and $\alpha \rightarrow 0$, e.g. we are searching for the shortest path of all paths achieving the goal point.

To achieve these restrictions we can use the following relations [88] :

$$\alpha + \beta = 1 \ , \qquad \beta = 0.5 \cdot e^{-poserror} + 0.5 \tag{5.3}$$

Equ.5.3 forces the path primarily to the desired goal point and later in to minimize the path length. We could also choose e.g. $\beta = 0.2 \cdot e^{-poserror} + 0.8$, which bias the path search even more towards finding the goal point instead of finding a short path.

The last term, we have to take into account is a measure for collisions. This term counts the numbers of collisions and the 'depth of penetration' into the obstacle. To measure the latter, we have to build the convec hull of the obstacle vertices and count the number of steps within it. As we are only interested in collision free paths$p$, we need a strong weight factor for collisions. This can be achieved by using the e- function. Therefore we find for the fitness function

$$f(p) = (\Omega - \beta \cdot \epsilon \cdot poserror(p) + (1 - \beta) \cdot length(p)) \cdot e^{-\cdot coll(p)} \ , \tag{5.4}$$

where Ω is a parameter and ϵ is an additional parameter for forcing the path the achieve the gaol point. We set ϵ to values between 1 and 5. The advantage of such a fitness function is that we only need to choose two

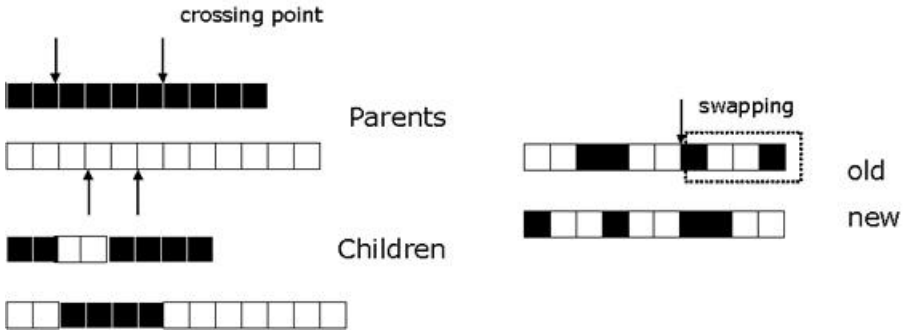


Fig. 5.4.: The two-point crossover and the swapping operator.

parameters, while the other ones are already determined. The task now is to find a maximum of that function, which leads to a feasible path for the robot.

5.3. Genetic Operators

In our GPP we used the following Genetic operators:

Crossover
We use a two-point crossover in our GPP. As we use variable length chromosomes this operator is similar but not identical to the classical two-point crossover operation. The operator works in the following way: two crossing points are randomly chosen for each chromosome. Then the strings between these points are cut and exchanged. This will also change the length of the chromosomes as shown in Fig. 5.4 The crossover operator is applied by a probability p_c.

Swapping
The swapping operation divides randomly the single chromosome in two parts and exchange these positions (Fig.5.4) . As this is a background operator it is applied by a small probability p_s.

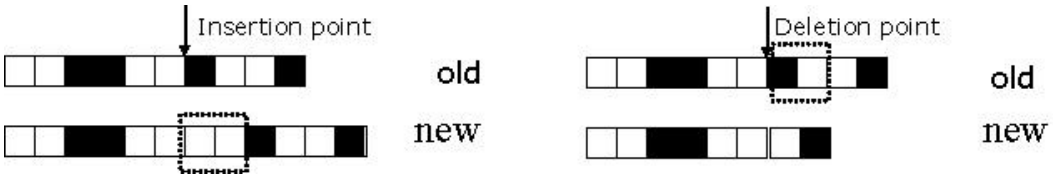


Fig. 5.5.: The insert and delete operator.

Mutation

As we don't work with a binary representation, we can't use simple bit-flipping process. Instead we replace an existing entry with a new random number in the interval $[1, \ldots, 8]$. Mutation plays a key role to diversify the population. It is therefore not necessary that mutation improves a single solution. The mutation operator is applied by a probability p_m.

Insertion

The insertion operation chooses randomly a sequence of actions and insert it into an existing chromosome (Fig.5.5). This operation helps to stretch those chromosomes, which produce in principle feasible but too short paths. This operator is also applied with a small probability p_i.

Deletion

This operator works in the opposite way as the insertion operator as it deletes a sequence of random length from a chromosome (Fig.5.5). As this operator might lead to useless short paths, which never has a change of achieving the goal, it has be applied with a small probability p_d.

Knowledge based Operators

Although not implemented in our GPP, some authors [90], [93] suggest the implementation of additional operators, which make use of some problem-specific knowledge, e.g. of the environment.

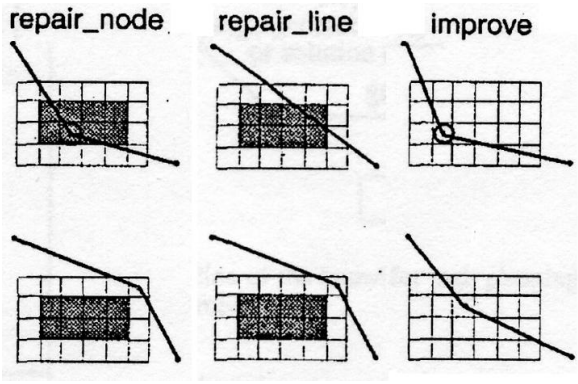


Fig. 5.6.: Three specialized Genetic operators.

In Fig.5.6 we find some demonstrative examples. The operator *repairnode* is used to move a node fallen into an obstacle out of it and to a best grid around an obstacle. To find an optimal grid outside the obstacle, a local search operation is used. The *repairline* operator works in a similar way and repairs an infeasible line segment by introducing a suitable node outside the obstacle. The *improvement* operator optimizes already feasible line segments by moving suboptimal nodes to better grid positions, in order to minimize path distances.

These operators can improve the quality and feasibility of the solutions significantly [93]. However, the price to pay is performance as these additional feasibility checks consume computational time. Therefore such operators are especially helpful in congested environments, where to chance of finding feasible solutions at all is low.

Choice of the parameters

A significant problem in designing EA is the choice of the parameters. As there exists no mathematical algorithm for correct parameter settings, we have to do exhaustive experiments to find useful values. Nevertheless, because of much effort in theoretical research as well as in practical implementations, there exits some standard values, which often need only small adaption. In our experiments we find out that the following settings yield

acceptable results. The population size = 60, the number of generations = up to 300, the crossover rate $p_c = 0.6$, the mutation rate $p_m = 0.05$, the $swappingrate = 0.1$, $insertionrate = 0.01$ and the $deletionrate = 0.1$. We use a $q = 4$ tournament selection and elitism, to keep back the best chromosome in each generation. The primary goal is to generate collision free paths that achieves the goal point. Out of that set of paths, the shortest of them will give the optimal solution.

5.4. Simulation in an environment without obstacles

The first case of our experiments consists a 20×20 box without obstacles. The starting point is $(0, 0)$, which is the lower left corner of the box. The goal point is $(20, 20)$, which is the right upper corner. Fig.5.7 shows the generated path after 10 generations. The simulations run on a Pentium IV 1.5 GHz, 512 MB RAM and a Windows XP operating system. The CPU time was 0.061 s for 10 generations and the best chromosome is given as [**12223331312233233112222**]. We see that either the shortest possible path is generated nor the goal point is achieved. After 20 generations the optimal path has been generated by the GPP (Fig.5.8). The CPU time was 0.101 s and the generated sequence is [**22222222222222222222**].

An important feature of the GPP is its relative fast convergence as can be seen in Fig.5.10. On the abscissa we find the number of generations and on the ordinate the corresponding fitness value. For the sake of clarity we plot the inverse fitness values, so that the fittest est individuals are those with the smallest values. There is a sharp decline of the function converging to 0 after 20 generation. We also plotted the average fitness of each generation (dashed line), which of course shows the decline of the fitness value, too. But there can also bee seen, that the average values of the later generations show a slight increase of the fitness function. Thus

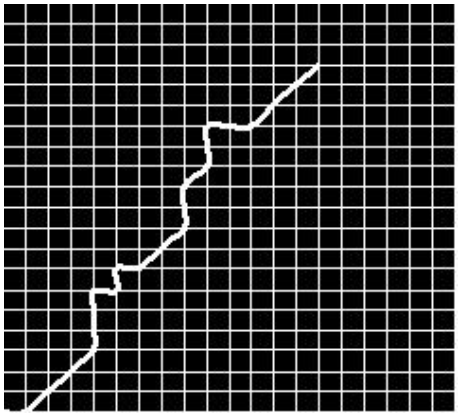


Fig. 5.7.: The generated path after 10 generations.

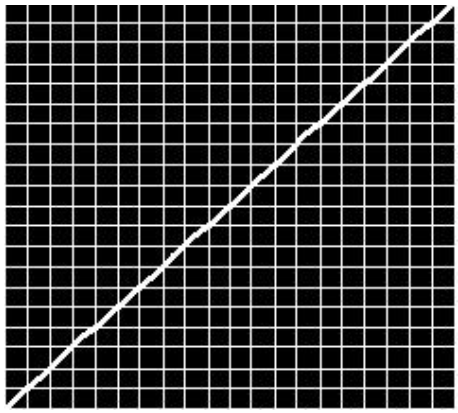


Fig. 5.8.: The generated path after 20 generations.

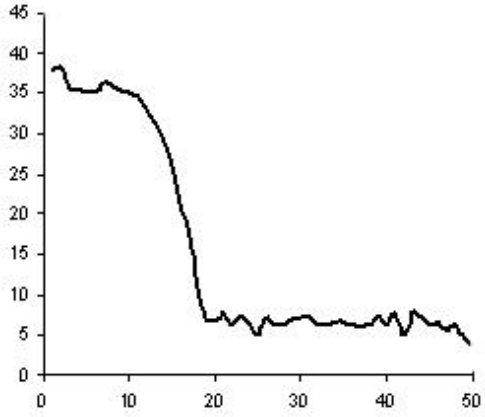


Fig. 5.9.: The positional error as a function of generations.

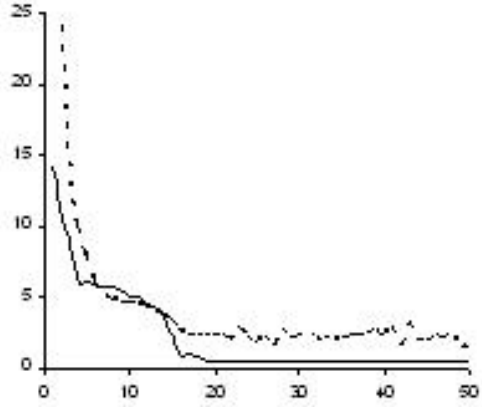


Fig. 5.10.: The average fitness as a function of generations.

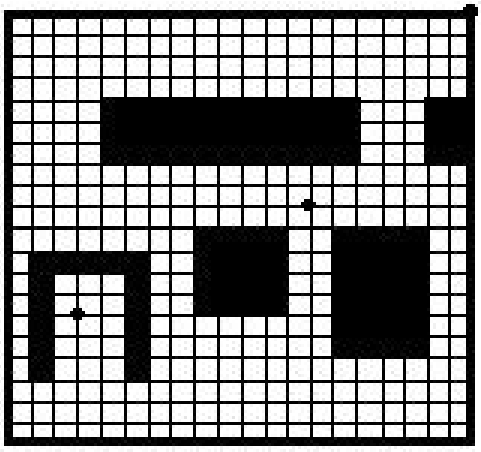


Fig. 5.11.: An environment with some obstacles.

it is important to mention, that the production of arbitrary generations does't necessary improve the average fitness. Actually it can happen, that additional generations worsen the result.

In Fig.5.9 we show the average position error of each generation. We again find a sharp decline after 20 generations, representing the optimal solution.

5.5. Simulation in an environment with obstacles

The next task is to plan a path in an environment with obstacles as shown in Fig.5.11. The starting point in that case is $(3, 6)$, where the origin of the coordinate system again is situated in the lower left corner, and the goal point is at $(13, 11)$ The challenge of that scene is the U-shaped obstacle around the starting point and the relatively narrow passages between the rectangular obstacles.

After 10 generations we find the path Fig. 5.14 (a), with a large position error and a twisting path. Nevertheless the solution already finds a way out of the shape after the short CPU time of 0.03 s . The parameter ϵ of Equ.(5.4) has been set to 1 and as a further constraint we use the condition that the path length has to be at least a minimum number otherwise the fitness is set to zero. The reason to do this is, that there is a local minimum around the starting point, which results in no or only a small motion. This is some kind of 'expert knowledge', which is extracted after a few simulation runs, and an example of how the search procedure can be accelerated by introducing some useful conditions. In the figures Fig. 5.14 (b), (c), (d) we find the evolution of the paths as a function of the calculated generations. As the paths are calculated by random operations, we can't expect only straight lines as in the case of visibility Graphs. In some cases we have to smooth the path in a post-processing phase.

It should also be noted, that the shown paths are only examples of a larger class of possible ways. As a GA is a stochastic process, we can't generate the same paths at every run. Especially, due to the use of variable length chromosomes, it sometimes happens that the chromosomes will be too large, which results in a overflow error. In such cases, the algorithm doesn't return any path and the genetic procedure has to start again.

Another severe difficulty is the correct parameter tuning of the genetic operators and of the fitness function. If *poserror* is too dominant, the planner will generate paths, which are a direct connection between the start and the goal points, thus ignoring the obstacles in the environment. On the other hand, if *poserror* is weighted too low, than the generated paths show long detours or won't even reach the goal point. As in the case of an environment with no obstacles, we find, that the GPP converges relatively fast (Fig. 5.12) and at least for environments with a complexity of our setting, we find optimal solutions after maximal 300 generations. Thus, calculating up to 1000 generations won't improve the resulting paths

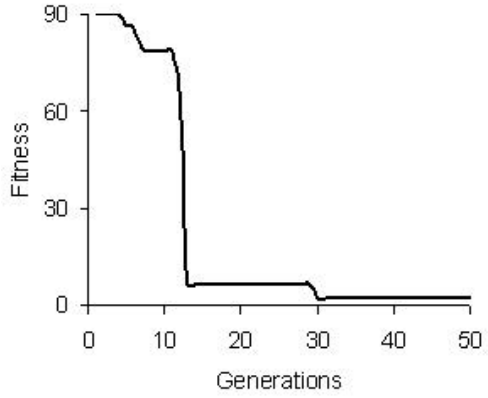


Fig. 5.12.: The average position error.

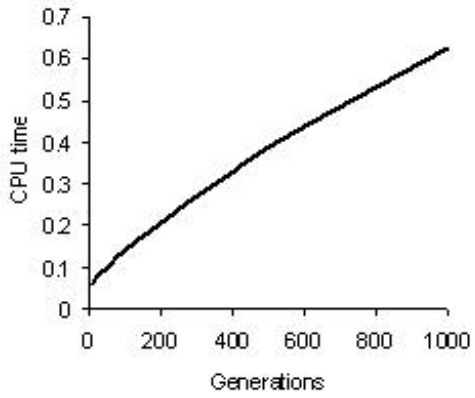
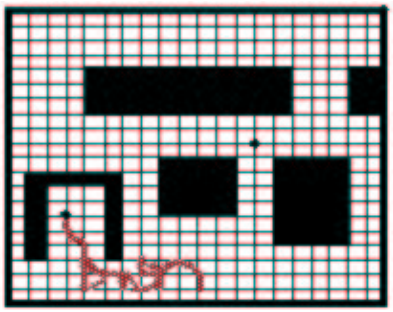


Fig. 5.13.: The average CPU time.

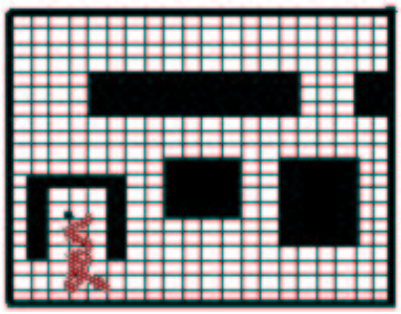
as can be seen in Fig. 5.14 (e). In Fig. 5.13 we show the average CPU time as a function of the number of generations. The resulting curve show, that even in the case of 1000 generations the CPU time is below 1 s and therefore fast enough for planning tasks of mobile robots in relatively large environments. But the CPU time on it's own isn't very meaningful as we have to take the successful experiments into consideration, too. The GPP produced an optimal path in about 80 % of the experiments, a positional error less then 4 in about 75 % and failed to produce a path in about 8%.

Value of randomized planners

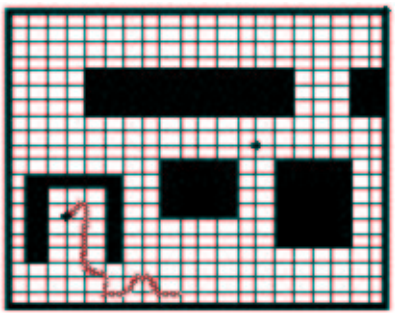
Since the seminal work of Barraquand and Latombe on the RPP [49], randomization is ubiquitous in planning algorithms. In the case of GPP it is a necessary element due to the nature of evolutionary algorithms. An inescapable feature of randomized algorithms is their lack of repeatability: no two runs will generate the same results. This has some advantages and disadvantages when applied to the field of robot motion planning. It is positive that sometimes the randomized algorithm will be 'lucky' and solve a problem very quickly. In the other side, if it takes a long time to solve a particular problem, there is hope that it will be



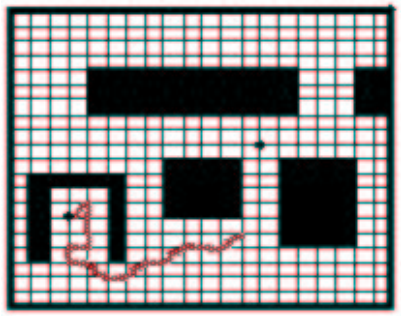
(a) Path after 10 generations.



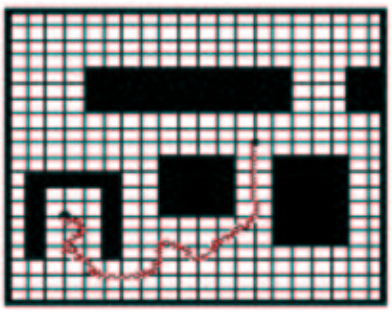
(b) Path after 20 generations.



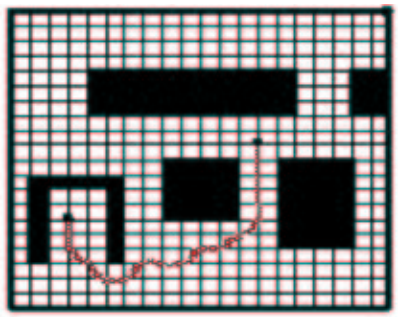
(c) Path after 50 generations.



(d) Path after 100 generations.



(e) Path after 300 generations.



(f) Path after 1000 generations.

Fig. 5.14.: Path Planning in an environment with obstacles.

faster next time. In the case of a deterministic algorithm, like Visiblity graph or cell decomposition, this is not the case. If a deterministic algorithm performs poorly once, it will always perform poorly for that problem.

On the other hand, the lack of repeatability caused by randomization can easily lead to a misinterpretation of the performance of the algorithm as some important aspects might be overlooked. In a deterministic algorithm it often can be discovered quickly because a single execution may be enough to reveal them. This results in a greater carefulness in the algorithm design and implementation process, respectively occurring errors can more easily be discovered. Further more, a greater understanding of high-level algorithmic operation is possible since there is no random noise in its performance and operation.

Especially in the case of Evolutionary algorithms, where the performance is influenced by a lot of parameters, it is sometimes difficult to detect design or implementation errors. Further more, the performance of the algorithm depend on the existence of a good random number generator. Therefore it is sometimes difficult do find out, why a genetic algorithm performs worse in a special case and a lot of runs have to be made, in order to draw conclusions from some statistical patterns.

Path Quality

As already mentioned, the quality of randomly generated paths is sometimes ugly. A resulting path can make long detours and contain many redundant motions. Another problem is the fact, that we plan our motions on a grid, which leads to first-order discontinuities at the nodes of the path.

A standard method to overcome these problems is to smooth the path in a post-processing phase. This can be done by choosing pairs of configurations on the path (not necessarily nodes of the path) and trying to

replace the path between these two points by the path resulting from calling a local (exact) planner. After the replacement of the old sections of the path, again a collision checker has to proof the feasibility of the new path. Unfortunately, smoothing only partially solves the problem. It does reduce the length of the path in open areas but it often cannot correct long detours around obstacles and it doesn't make the path first-order continuous .

We can't get the problem of avoiding long detours completely under control as the primary focus of the GPP is to avoid collisions and reach the goal point. A way out of that, is to make a few simulation runs and choose the shortest path out of it. Another possibility is to set a maximal path length as an upper bound and discard all other generated paths.

Nodes in the generated path on a grid introduce first-order discontinuities in the motion. To avoid this problem, one has to find smooth curves connecting the nodes. One possible solution is to proceed as follows: Let e_1 and e_2 be two consecutive edges of the path and m_i the midpoint of e_i $(i = 1, 2)$. In order to get a differentiable path, we replace the part of the path between m_1 and m_2 by a circle arc. This arc will have its center on the bisecting line of e_1 and e_2, will touch e_1 and e_2 and have either m_1 or m_2 on its boundary. Doing this for each consecutive pair of edges results in a smooth path. If the new path collide with an obstacle, we can make the circle smaller, pushing it more towards the node between the edges.

Another possibility is to use more advanced types of curves like *B-splines*. A B-spline curve is expressed as

$$\vec{P}(u) = \sum_{i=0}^{n} \vec{P}_i \cdot N_{i,k}(u) \quad (t_{k-1} \leq u \leq t_{n+1}) \ ,$$

where

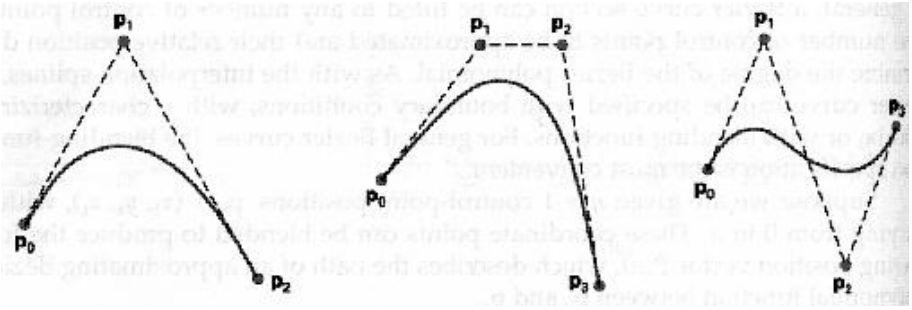


Fig. 5.15.: Some examples of B-spline curves.

$$N_{i,k}(u) = \frac{(u - t_i) \cdot N_{i,k-1}(u)}{t_{i+k-1} - t_i} + \frac{(t_{i+k} - u) \cdot N_{i+1,k-1}(u)}{t_{i+k} - t_{i+1}} \ ,$$

$$N_{i,1}(u) = \begin{cases} 1 & \text{for } t_i \leq u \leq t_{i+1} \\ 0 & \text{otherwise} \ , \end{cases}$$

and $\vec{P}_i$ denotes the position vector of the i-th vertex. A few examples of possible B-splines are shown in Fig. 5.15. With the help of such curves we are able to optimize the paths, in order to enable a smooth robot motion.

5.6. GPP with the Nomad 200

In the final section we show, that the simulation results can't be directly transmitted to the real robot system Nomad 200. Besides the the planning tasks we have additional problems to solve when working with a real robot system. In Fig. 5.16 we find the different building blocks of mobile robot navigation:

- **Perception**, the interpretation of the sensor data, in order to get meaningful information out of them .

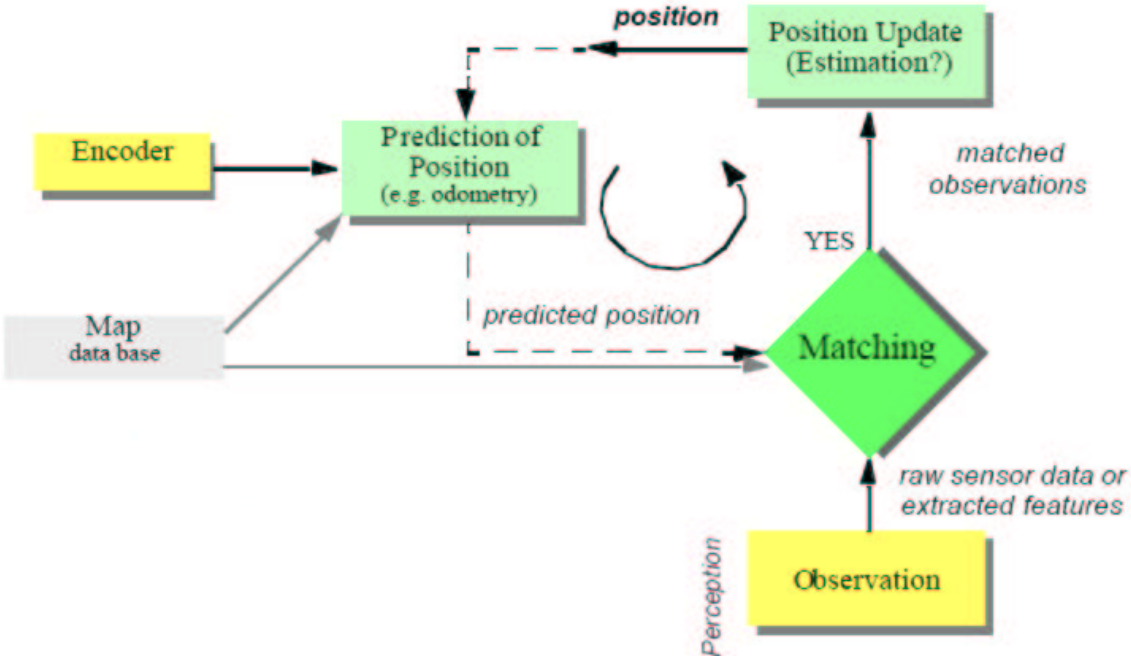


Fig. 5.16.: General schematic for mobile robot localization [10].

- **Localization**, the robot must determine it's position .
- **Position Prediction**, the path planning problem.
- **Motion Control**, the control of the motor output, in order to keep the desired path.

The localization problem has two challenges: Sensor noise and aliasing. When using a color CCD Camera for example, then the different objects can be recognized by analysing the different RGB values. But this information depends on the illumination of the environment. When the sun is hidden by the clouds, the measured RGB values are completely different and this method of object recognition doesn't work any more. The second problem we are faced with, is the non uniqueness of sensor readings, so-called **sensor aliasing**. In opposite to human sensor systems, which receive mostly unique inputs in each unique local state, there is a many-to-one mapping from environmental states to the

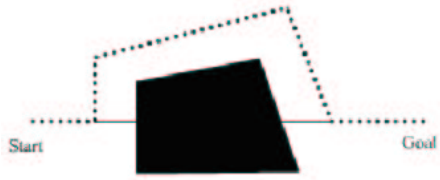


Fig. 5.17.: Kollision avoidance a).

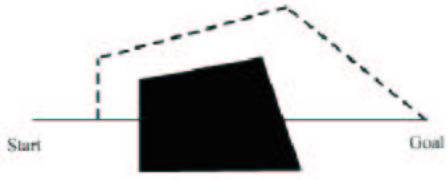


Fig. 5.18.: Kollision avoidance b).

robot's input. Thus, the robot's percepts cannot distinguish between these states.

There are a lot of methods to localize the robot: Markov and Kalman filter localization, which are extremely popular strategies for indoor robots. Besides there are a lot of other localization techniques like Landmark-based navigation, mosaic-based navigation or route-based navigation.

Another important aspect of navigation is collision avoidance of obstacles, which do not exist in the pre-planned path. The task is to steer the robot around obstacle on a detour as optimal as possible. The minimum time and power consumption should be taken into consideration. During his trip robot meets movable and stationary obstacles. Whether the obstacle is movable or stationary the collision avoiding approach should be different.

Two possible collision avoiding procedures are considered in case the obstacle lies on pre-planned path.

- Robot follows the contour of the obstacle - like wall following - until he meets the pre-planned path (Fig.5 a). This approach can be easily implemented. But this approach is presumed that the working environment is static.
- Robot sets a virtual goal position and moves to position as long as there is no obstacle to goal position (Fig.5 b).

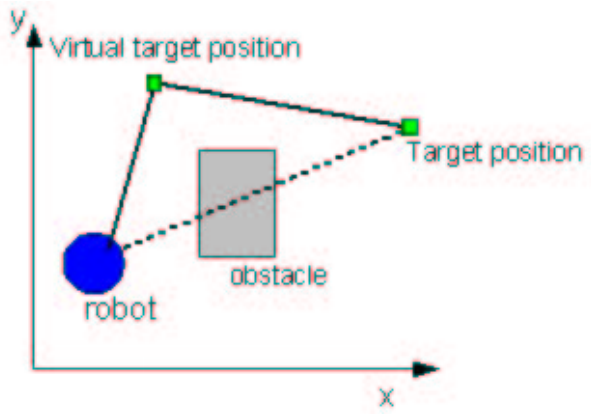


Fig. 5.19.: Virtual target position.

In this work the second procedure is used. While the mobile robot is moving along the pre-calculated path - a straight line between a present and a target-position - the sensor system of the robot detects the environment and it's changes. In our case for example (Fig.5.19), the robot detects an obstacle along his moving direction, sets a virtual target position and tries to reach this position. After the position is reached, the robot moves to the actual target position again.

The localization methods mentioned before has the restriction, that they need human effort to install the robot into space. In order to behave like an intelligent agent, we would like that the robot itself could autonomously explore the environment,interpret it and build a map for planning a path. This problem is also called **SLAM** (simultaneous localization and mapping) and is one of the most difficult problems to solve in the field of mobile robots. Although SLAM has not yet been fully perfected, it is starting to be employed in unmanned aerial vehicles, autonomous underwater vehicles, planetary rovers and newly emerging domestic robots. As GPP doesn't need an exact map of the environment, they are relatively robust against some noise of the sensor data and uncertainties of the obstacle's position. Therefore the use of GPP attenuates to some extent the SLAM problem.

6. Conclusion and Outlook

In this work, Genetic Algorithms where used for planning paths of mobile robots. As the applications of mobile robots increase rapidly, the task of finding feasible paths in complex environments with numerous obstacles is a necessary condition for autonomous navigation.

Since the classical path planning algorithms are mostly too slow for complex environments, heuristic methods receive an increasing interest in the path planning community. Besides the sample based algorithms, like RPM or RRT, Genetic Path Planners provide a promising tool for planning tasks. As such planners doesn't reason about what actions have to be done to move safety, rather generating feasible paths by an adaptive search, they are ideal candidates for rapid motion planning. Therefore even in the case of noisy information of the environment, GPP provide some usable results.

The features of our path planning algorithm is that it uses variable length chromosomes, an integer representation of the robot's movements, a fitness function forcing the positional errors and unfeasible steps quickly to zero and some advanced genetic operators, which accelerate the search notable.

There are many directions in which this work can be proceed. Instead of using fixed obstacles, one could extend the genetic path planner to moving obstacles. Since GA are often used for scheduling tasks, the coordination of multiple robots motion would also be an interesting case to study. Using

some expert knowledge improves the performance of the genetic algorithm - therefore the implementation of Fuzzy rules for control tasks is also an interesting challenge for future work. As the Nomad 200 is a almost holonomic system, the use of GPP in a non-holonomic system, like a car like robot or a robot with trailers could also provide some new insights in the functioning of genetic algorithms.

A. Source Code of the GPP

In the following chapter we present the code of the GPP as we used in the simulation. It is written in C using the LCC-Wedit Win32 Ver. 3.3. compiler. We didn't implement any GUI in order to keep the program small and simple. For the same reason, we skipped all those parts of the program, which doesn't deal directly with the path planner. In the following flowchart we list all the building blocks, which are needed to control the real system.

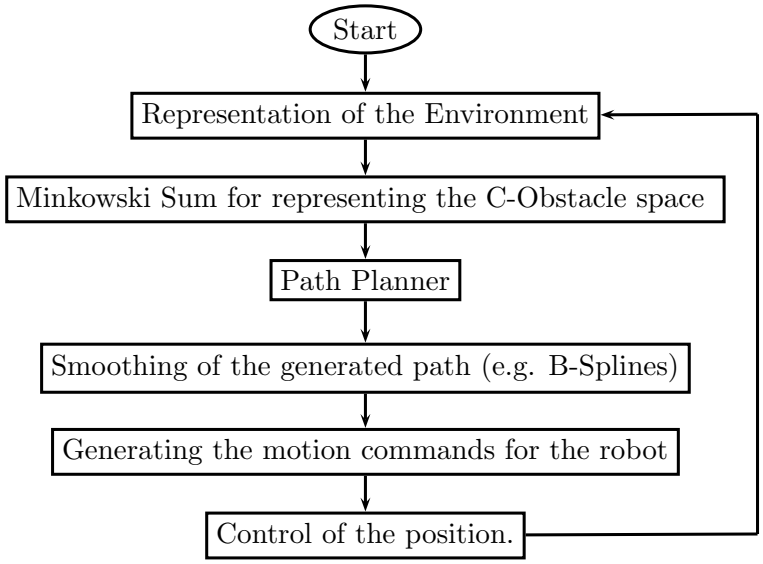


Fig. A.1.: Building Blocks of the robot navigation systems.

```
/*************************************************************/
/* This is a  genetic algorithm implementation where the      */
/* evaluation function takes positive values only and the      */
/* fitness of an individual is the same as the value of the    */
/* objective function                                          */
/*************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>

/* Change any of these parameters to match your needs */

#define POPSIZE 60            // population size
#define MAXGENS 100           // max. number of generations
#define SELECT_PROB 0.6      // probability for crossoverselection
#define PMUTATION 0.08        // probability of mutation
#define q 4                   // Number of tournaments of the q-tourn. selection
#define SWAP 0.2             // probability of swapping process
#define INSERT 0.1            // probability of insert. a random sequence
#define DELETE 0.05          // probability of deleting  a random sequence
#define BETA 1
#define ALPHA 10*abs(X_START-X_GOAL)       //parmaeter of the fitness function
#define TRUE 1
#define FALSE 0
#define BOX 40         // The length of the square, where the robot can operate
#define X_START 3.0    // Start and Endpoints
#define Y_START 6.0
#define X_GOAL  20.0
#define Y_GOAL  20.0

#define RND   rand()
#define RNDE ((float) rand()/((float) RAND_MAX+1))  //Random number between
[0,1]

FILE *galog;                     // an output file
FILE *bestway;                   // an output file with the best way


int generation=0;               // number of generations, from 0 to MAXGENS

struct genotype                   // genotype (GT), a member of the population
{
      int    *gene;               // a string of numbers [1-8], which represents
                                  // the different directions to drive
      double fitness;             // GT's fitness
      int    length;              // number of chromosoms in each gene
      double step;                // 1 if the step is along the axis and
                                  // sqrt(2) if it is a median
      double way;                 // total length of the way
      double xcoord;              // x-coordinate
      double ycoord;              // y-coordinate
      double collision;           // number of collisions with obstacles
      double p_error;             // Positional Error from the desired goal
};

struct genotype population[POPSIZE+1];    // population
struct genotype newpopulation[POPSIZE+1]; // new population replaces old one

struct point
{
      double x;
      double y;
} ;


/* Declaration of procedures used by this genetic algorithm */

void initialize(void);
int  randval(void);
void evaluate(void);
void clear(void);
void keep_the_best(void);
void elitist(void);
void select(void);
void crossover(int a, int b);
void swap(void);
void insert(void);
void delete(void);
void mutate(void);
void report(void);
void direction( double chromo, int mem);


int obstacle(int a, int b);


/*************************************************************/
/* Initialization function: Initializes the values of genes    */
/* within the variables bounds. It also initializes (to zero)  */
/* all fitness values for each member of the population. It    */
/* reads upper and lower bounds of each variable from the      */
/* input file `gadata.txt'. It randomly generates values       */
/* between these bounds for each gene of each genotype in the  */
/* population. The format of the input file `gadata.txt' is    */
/* var1_lower_bound var1_upper bound                           */
/* var2_lower_bound var2_upper bound ...                       */
/*************************************************************/

void initialize(void)
{
int i,j;
int chromolength;

// initialize variables within the bounds

for (i = 0; i <= POPSIZE; i++)
      {
            chromolength = rand()%(BOX*(BOX-1)+1) + BOX ;
            //lenght of the chromosom, choosen randomly , between BOX + BOX^2
            population[i].gene = malloc(chromolength*sizeof(int));
            population[i].length = chromolength;
            population[i].xcoord = X_START;
            population[i].ycoord = Y_START;
            population[i].step = 0;
            population[i].way = 0;
            population[i].collision = 0;
            population[i].fitness = 0;
            population[i].p_error = 0;
```

```
        for (j = 0; j <population[i].length; j++)
            population[i].gene[j] = randval();  //string of directions to drive
      }
}
/*************************************************************/
/*Random value generator:Generates a value                   */
/*between 0 and 8 (the directions to drive)                   */
/*************************************************************/

int randval(void)
{
int val;
val = (int)(rand()%8+1);
//numbers, representing [1,..,8] the possible directions (East,NorthEast,...)
return(val);
}


/*************************************************************/
/* Evaluation function: This takes a user defined function.  */
/* Each time this is changed, the code has to be recompiled. */
/* The current function is:                                  */
/*************************************************************/

void evaluate(void)
{
int i,j, k;
double w;

clear();

  for (j = 0; j < POPSIZE; j++)
      {
           for (i = 0; i < population[j].length; i++)
           {
             direction(population[j].gene[i], j);
               population[j].way += population[j].step ;

      /*********simple collision check ******************************/
        if(obstacle(population[j].xcoord,population[j].ycoord)==1)
          population[j].collision += 1.0;      //counts the collisions and
                                               //the penetration into obstacle
      /*************************************************************/
           }
               //fitnessfunction has to be maximized
      population[j].p_error = sqrt(pow(population[j].xcoord - X_GOAL,2)
                             + pow(population[j].ycoord - Y_GOAL,2));
                        w  = 0.5*pow(2.7, -population[j].p_error)+ 0.5;

      population[j].fitness = (ALPHA - w*BETA*population[j].p_error
                -(1-w)*population[j].way)*pow(2.7,-population[j].collision);

           if(population[j].way  < abs(population[j].xcoord - X_GOAL))
              population[j].fitness =0;
           if(population[j].p_error  < 10)
            population[j].fitness =0;

             // two additional constraints, in order to
     }
}
```

```
/*************************************************************/
/*    obstacle                                               */
/*                                                           */
/*************************************************************/

int obstacle(int a, int b)   //int population[a].xcoord, int
population[a].ycoord)
{
      int i;
      struct point vertex_obstacle1[2];
      struct point vertex_obstacle2[2];
      struct point vertex_obstacle3[2];
      struct point vertex_obstacle4[2];
      struct point vertex_obstacle5[2];
      struct point vertex_obstacle6[2];
      struct point vertex_obstacle7[2];


            vertex_obstacle1[0].x = 1;
            vertex_obstacle1[0].y = 3;
            vertex_obstacle1[1].x = 2;
            vertex_obstacle1[1].y = 9;

            vertex_obstacle2[0].x = 2;
            vertex_obstacle2[0].y = 8;
            vertex_obstacle2[1].x = 5;
            vertex_obstacle2[1].y = 9;

            vertex_obstacle3[0].x = 5;
            vertex_obstacle3[0].y = 3;
            vertex_obstacle3[1].x = 6;
            vertex_obstacle3[1].y = 9;

            vertex_obstacle4[0].x = 8;
            vertex_obstacle4[0].y = 6;
            vertex_obstacle4[1].x = 12;
            vertex_obstacle4[1].y = 10;

            vertex_obstacle5[0].x = 14;
            vertex_obstacle5[0].y = 4;
            vertex_obstacle5[1].x = 18;
            vertex_obstacle5[1].y = 10;

            vertex_obstacle6[0].x = 4;
            vertex_obstacle6[0].y = 13;
            vertex_obstacle6[1].x = 16;
            vertex_obstacle6[1].y = 16;

            vertex_obstacle7[0].x = 18;
            vertex_obstacle7[0].y = 13;
            vertex_obstacle7[1].x = 20;
            vertex_obstacle7[1].y = 16;



      if
        (
        ((a >= vertex_obstacle1[0].x) && (a <= vertex_obstacle1[1].x) && (b >=
vertex_obstacle1[0].y) &&
        b <= vertex_obstacle1[1].y) ||
       ((a >= vertex_obstacle2[0].x) && (a <= vertex_obstacle2[1].x) && (b >=
vertex_obstacle2[0].y) &&
        b <= vertex_obstacle2[1].y) ||
```

```
      ((a >= vertex_obstacle3[0].x) && (a <= vertex_obstacle3[1].x) && (b >=
vertex_obstacle3[0].y) &&
        b <= vertex_obstacle3[1].y)  ||
      ((a >= vertex_obstacle4[0].x) && (a <= vertex_obstacle4[1].x) && (b >=
vertex_obstacle4[0].y) &&
        b <= vertex_obstacle4[1].y)  ||
      ((a >= vertex_obstacle5[0].x) && (a <= vertex_obstacle5[1].x) && (b >=
vertex_obstacle5[0].y) &&
        b <= vertex_obstacle5[1].y)  ||
      ((a >= vertex_obstacle6[0].x) && (a <= vertex_obstacle6[1].x) && (b >=
vertex_obstacle6[0].y) &&
        b <= vertex_obstacle6[1].y)  ||
      ((a >= vertex_obstacle7[0].x) && (a <= vertex_obstacle7[1].x) && (b >=
vertex_obstacle7[0].y) &&
        b <= vertex_obstacle7[1].y)  ||
        (a <= 0 || b <= 0 || a >= BOX || b >= BOX)              //out of BOX x
BOX
         )
       return 1 ;
      else
            return 0 ;

}
/*************************************************************/
/* Clear the chromosome entries for the next run           */
/*************************************************************/

void clear(void)
{
      int i;
      for (i = 0; i < POPSIZE; i++)
            {
             population[i].step = 0;
             population[i].way = 0;
             population[i].collision = 0;
             population[i].xcoord = X_START;
             population[i].ycoord = Y_START;
            }
}


/*************************************************************/
/* direction of the robot */
/*************************************************************/

void direction( double chromo, int mem)
{
      switch(((int)chromo))
            {
            case 1:                                                 //right
                  {
                        population[mem].xcoord ++;
                        population[mem].step = 1.0;

                        break;
                  }
            case 2:                                                 //up-right
                  {
                        population[mem].xcoord ++;
                        population[mem].ycoord ++;
                        population[mem].step = sqrt(2.0);

                        break;
                  }
            case 3:                                                 //up
                  {
                        population[mem].ycoord ++;
                        population[mem].step = 1.0;

                        break;
                  }
            case 4:                                                 //up-left
                  {
                        population[mem].xcoord --;
                        population[mem].ycoord ++;
                        population[mem].step = sqrt(2.0);

                        break;
                  }
            case 5:                                                 //left
                  {
                        population[mem].xcoord --;
                        population[mem].step = 1.0;

                        break;
                  }
            case 6:                                                 //down -left
                  {
                        population[mem].xcoord --;
                        population[mem].ycoord --;
                        population[mem].step = sqrt(2.0);

                        break;
                  }
            case 7:                                                 //down
                  {
                        population[mem].ycoord --;
                        population[mem].step = 1.0;

                        break;
                  }
            case 8:                                                 //down-right
                  {
                        population[mem].xcoord ++;
                        population[mem].ycoord --;
                        population[mem].step = sqrt(2.0);

                        break;
                  }
            }
}


/*************************************************************/
/* Keep_the_best function: This function keeps track of the    */
/* best member of the population. Note that the last entry in  */
/* the array Population holds a copy of the best individual    */
/*************************************************************/

void keep_the_best()
{

int i,j;
int index_best = 0; /* stores the index of the best individual */
int best;
```

```
best = population[0].fitness ;

   for (i = 0; i <= POPSIZE; i++)
      {
        if(population[i].fitness > best)
              {
        best = population[i].fitness;
          index_best = i;
       }
}


                /* once the best member in the population is found, copy the
genes into
                   the last chormosome of the current generation. */


        population[POPSIZE].gene =
malloc(population[index_best].length*sizeof(int));
            for(i=0;i<population[index_best].length;i++)
                  population[POPSIZE].gene[i] = population[index_best].gene[i] ;

                  population[POPSIZE].length    = population[index_best].length;
                  population[POPSIZE].way       = population[index_best].way;
                  population[POPSIZE].p_error   =
population[index_best].p_error;
                  population[POPSIZE].collision =
population[index_best].collision;


}

/*****************************************************************/
/* Elitist function: The best member of the previous generation */
/* is stored as the last in the array. If the best member of    */
/* the current generation is worse then the best member of the  */
/* previous generation, the latter one would replace the worst  */
/* member of the current population                             */
/*****************************************************************/

void elitist()
{
int i;
double best, worst;                     /* best and worst fitness values */
int index_best=0, index_worst=0;         /* indexes of the best and worst
member */

best  = population[0].fitness;
worst = population[0].fitness;

for (i = 0; i < POPSIZE; i++)
     {
      if(population[i].fitness > best)
            {
         best =  population[i].fitness;
         index_best = i;
          }
      if(population[i].fitness < worst)
            {
            worst = population[i].fitness;
          index_worst = i;
            }
      }
```

```
/* if best individual from the new population is better than */
/* the best individual from the previous population, then    */
/* copy the best from the new population; else replace the   */
/* worst individual from the current population with the     */
/* best one from the previous generation                     */


if (best >= population[POPSIZE].fitness)
       {
      population[POPSIZE].gene =
malloc(population[index_best].length*sizeof(int));
    for (i = 0; i < population[index_best].length; i++)
            population[POPSIZE].gene[i] = population[index_best].gene[i];

      population[POPSIZE].fitness = population[index_best].fitness;
      population[POPSIZE].length = population[index_best].length;
      population[POPSIZE].way = population[index_best].way;
      population[POPSIZE].p_error = population[index_best].p_error;
    }
else
    {
      population[index_worst].gene =
malloc(population[POPSIZE].length*sizeof(int));
    for (i = 0; i < population[POPSIZE].length; i++)
            population[index_worst].gene[i] = population[POPSIZE].gene[i];

      population[index_worst].fitness = population[POPSIZE].fitness;
      population[index_worst].length = population[POPSIZE].length;
      population[index_worst].way = population[POPSIZE].way;
      population[index_worst].p_error = population[POPSIZE].p_error;
    }

}



/*****************************************************************/
/* q-tournament selection
*/
/* makes sure that the best member survives                   */
/*****************************************************************/

void select(void)
{
  int i,j;
  int index, temp;

      for(i = 0; i < POPSIZE; i++)
      {
            index = RND%POPSIZE ;

        for(j=0;j<q; j++)
             {
                   temp = RND%POPSIZE ;
                   if(population[temp].fitness > population[index].fitness)
                   index = temp;
             }



                 newpopulation[i].gene =
malloc(population[index].length*sizeof(int));
                       for(j=0;j<population[index].length;j++)
```

```
            newpopulation[i].gene[j] = population[index].gene[j] ;
            newpopulation[i].length  = population[index].length;
            newpopulation[i].way     = population[index].way;
            newpopulation[i].fitness = population[index].fitness;
            newpopulation[i].p_error = population[index].p_error;

      }

           for(i = 0; i < POPSIZE; i++)      //copy  newpopulation to old one ,
              {
                   population[i].gene =
malloc(newpopulation[i].length*sizeof(int));
                       for(j=0;j<newpopulation[i].length;j++)
                        population[i].gene[j] = newpopulation[i].gene[j] ;

                   population[i].length  = newpopulation[i].length;
                   population[i].way     = newpopulation[i].way;
                   population[i].fitness = newpopulation[i].fitness;
                   population[i].p_error = newpopulation[i].p_error;
            }

}


/*************************************************************/
/* Crossover, 2 random points are selected and the allele    */
/* within them are exchanged                                 */
/*************************************************************/

void crossover(int a, int b)
{

      int crosspoint[4];
      int i,temp;
      int *temp_1, *temp_2;
      int length_1,length_2;

                                        //choose 4 random crossing points
      for(i=0;i<2;i++)
      crosspoint[i] = RND%population[a].length;
      for(i=2;i<4;i++)
      crosspoint[i] = RND%population[b].length;


      for(i=0;i<4;i+=2)      //crosspoint[1]<crosspoint[2] AND
crosspoint[3]<crosspoint[4]
            {
       if(crosspoint[i]> crosspoint[i+1])
                 {
            temp = crosspoint[i];
          crosspoint[i] = crosspoint[i+1];
          crosspoint[i+1] = temp;
           }
           }

      length_1 = crosspoint[0]+ (crosspoint[3] - crosspoint[2])+
(population[a].length-crosspoint[1]);
      length_2 = crosspoint[2]+ (crosspoint[1] - crosspoint[0])+
(population[b].length-crosspoint[3]);

      temp_1 = malloc(length_1*sizeof(int));
      temp_2 = malloc(length_2*sizeof(int));
```

```
            for(i=0;i<crosspoint[0];i++)
         temp_1[i] = population[a].gene[i];

            for(i=crosspoint[0];i<crosspoint[0] + (crosspoint[3]-
crosspoint[2]);i++)
               temp_1[i] = population[b].gene[crosspoint[2]+(i-crosspoint[0])];

                for(i=crosspoint[0]+(crosspoint[3]- crosspoint[2]);
                      i<(crosspoint[0]+crosspoint[3]- crosspoint[2]) +
population[a].length - crosspoint[1];i++)
                  temp_1[i] = population[a].gene[crosspoint[1]+(i-
(crosspoint[0]+crosspoint[3]- crosspoint[2]))];

            for(i=0;i<crosspoint[2];i++)
         temp_2[i] = population[b].gene[i];

            for(i=crosspoint[2];i<crosspoint[2] + (crosspoint[1]-
crosspoint[0]);i++)
               temp_2[i] = population[a].gene[crosspoint[0]+(i-crosspoint[2])];

                for(i=crosspoint[2]+(crosspoint[1]- crosspoint[0]);
                      i<(crosspoint[2]+crosspoint[1]- crosspoint[0]) +
population[b].length - crosspoint[3];i++)
                   temp_2[i] = population[b].gene[crosspoint[3]+(i-
(crosspoint[2]+crosspoint[1]- crosspoint[0]))];

            for(i=0;i<length_1;i++)
                  population[a].gene[i] = temp_1[i];
                  population[a].length = length_1 ;

            for(i=0;i<length_2;i++)
                  population[b].gene[i] = temp_2[i];
                  population[b].length = length_2 ;

}

/*************************************************************/
/* Mutation: Random uniform mutation. A variable selected for */
/* mutation is replaced by a random value between lower and   */
/* upper bounds of this variable                             */
/*************************************************************/

void mutate(void)
{
int i, j;
double x;

for (i = 0; i < POPSIZE; i++)
      for (j = 0; j < population[i].length; j++)
            {
            x = RNDE;
            if (x < PMUTATION)
                  population[i].gene[j] = randval();

            }
}

/*************************************************************/
/* Swap divides the chromosome randomly into two parts        */
/* and exchanges these two parts                             */
/*************************************************************/

void swap (void)
```

```
{
int i,j, crosspoint;
int *temp_1, *temp_2;
double x;

for (i = 0; i < POPSIZE; i++)
  {
      x = RNDE;
       if (x < SWAP)
             {
               crosspoint = RND%population[i].length;
                temp_1 = malloc(crosspoint*sizeof(int));
                  temp_2 = malloc((population[i].length-
crosspoint)*sizeof(int));
              for (j = 0; j < crosspoint; j++)
                temp_1[j] = population[i].gene[j];
                    for (j = crosspoint; j < population[i].length ; j++)
                temp_2[j-crosspoint] = population[i].gene[j];

                    for (j = 0; j <population[i].length-crosspoint ; j++)
                population[i].gene[j] = temp_2[j];
                    for (j = population[i].length-crosspoint; j <
population[i].length ; j++)
                population[i].gene[j] = temp_1[j-(population[i].length-
crosspoint)];
            }
      }

}


/*************************************************************/
/* Insert chooses a random point an inserts a random sequence */
/* of allele into it                                         */
/*************************************************************/

void insert(void)

{

int i,j, insertpoint, insertlength;
double x;
int *temp_1, *temp_2;

for (i = 0; i < POPSIZE; i++)
  {
      x = RNDE;
       if (x < INSERT)
          {
           insertpoint  = RND%population[i].length;
           insertlength = RND%population[i].length;
             temp_1 = malloc(insertpoint*sizeof(int));
             temp_2 = malloc((population[i].length-insertpoint)*sizeof(int));

                    for (j = 0; j < insertpoint; j++)
                temp_1[j] = population[i].gene[j];
                    for (j = insertpoint; j < population[i].length ; j++)
                temp_2[j-insertpoint] = population[i].gene[j];

             population[i].gene = malloc((population[i].length +
insertlength)*sizeof(int));

              for (j = 0; j < insertpoint; j++)
                population[i].gene[j] = temp_1[j] ;
               for (j = insertpoint; j < insertpoint + insertlength; j++)
                        population[i].gene[j] = rand()%8+1 ;
                   for (j = insertpoint + insertlength; j < insertlength +
population[i].length; j++)
                        population[i].gene[j] = temp_2[j - (insertpoint +
insertlength)] ;
                        population[i].length = insertlength +
population[i].length;
              }
      }
}


/*************************************************************/
/* Delete chooses a random point an deletes a random sequence */
/* of allele from the chromosome                             */
/*************************************************************/


void delete(void)   //Problem: The length of the chromosome can get 0 !

{

int i,j, deletepoint, deletelength;
double x;
int *temp_1, *temp_2;

for (i = 0; i < POPSIZE; i++)
      {
      x = RNDE;
       if (x < DELETE)
          {
             deletepoint  =  RND%population[i].length;
            deletelength =  2; // a fixed length that the chromosom won´t get too
small -  RND%population[i].length;
             temp_1 = malloc(deletepoint*sizeof(int));
             temp_2 = malloc((population[i].length-deletepoint)*sizeof(int));

                    for (j = 0; j < deletepoint; j++)
                temp_1[j] = population[i].gene[j];
                    for (j = deletepoint + deletelength ; j <
population[i].length ; j++)
                temp_2[j- (deletepoint + deletelength)] = population[i].gene[j];
                  population[i].gene = malloc((population[i].length -
deletelength)*sizeof(int));

                  for (j = 0; j < deletepoint; j++)
              population[i].gene[j] = temp_1[j] ;
                for (j = deletepoint; j <  population[i].length-deletelength;
j++)
                        population[i].gene[j] = temp_2[j- deletepoint]  ;
              population[i].length = population[i].length - deletelength ;
            }
      }

}



/*************************************************************/
/* Report function: Reports progress of the simulation. Data   */
/* dumped into the  output file are separated by commas        */
```

```
/*************************************************************/

void report(void)
{
int i;
double best_val;            /* best population fitness */
double avg;                 /* avg population fitness */
double stddev;              /* std. deviation of population fitness */
double sum_square;          /* sum of square for std. calc */
double square_sum;          /* square of sum for std. calc */
double sum_p_error;         /* sum of positional error */
double avg_p_error;               /* avg positional error    */
double sum_way;                   /* sum of path lengths */
double avg_way;            /* avg path length */
double sum_coll;             /* sum of collisions */
double avg_coll;             /* avg collisions */
double sum;                 /* total population fitness */

sum         = 0.0;
sum_square  = 0.0;
sum_p_error = 0.0;
sum_way     = 0.0;
sum_coll    = 0.0;

for (i = 0; i < POPSIZE; i++)
      {
      sum         += population[i].fitness;
      sum_square  += population[i].fitness * population[i].fitness;
        sum_p_error += population[i].p_error;
        sum_way     += population[i].way;
        sum_coll    += population[i].collision;
      }

avg = sum/(double)POPSIZE;
square_sum = avg * avg * POPSIZE;
stddev = sqrt((sum_square - square_sum)/(POPSIZE - 1));
best_val = population[POPSIZE].fitness;
avg_p_error = sum_p_error/(double)POPSIZE;
avg_way = sum_way/(double)POPSIZE;
avg_coll = sum_coll/(double)POPSIZE;

fprintf(galog, "\n %d; %.3f; %.3f; %3f; %3f; %3f", generation +1,
        best_val, avg, avg_p_error, avg_way,avg_coll);
}

/*************************************************************/
/* Main function: Each generation involves selecting the best */
/* members, performing crossover & mutation and then          */
/* evaluating the resulting population, until the terminating */
/* condition is satisfied                                     */
/*************************************************************/


int main(int argc,char *argv[])
{



int i, goal=0;
int j;
int index_best=0;
double best ;
```

```
srand( (unsigned)time( NULL ) );

if ((galog = fopen("galog.csv","w"))==NULL)
      {
      exit(1);
      }

if ((bestway = fopen("bestway.txt","w"))==NULL)
      {
      exit(1);
      }


 initialize();
 evaluate();
 keep_the_best();


      while(generation<MAXGENS)
            {

            select();

            for(i=0;i<POPSIZE;i+=2)
              {
            if(RNDE < SELECT_PROB)   //two chromosoms are selected for crossover
with a probability SELECT_PROB
          crossover(i,i+1);
            }
      mutate();
            swap();
            insert();
            //delete();
            evaluate();
        elitist();
            report();
            generation++;
            }


 best = population[0].fitness ;
  for(j=0;j<=POPSIZE ;j++)
      {
      if(population[j].fitness > best)
            {
            best = population[j].fitness;
            index_best = j;
         }
        }

population[index_best].xcoord = X_START;
population[index_best].ycoord = Y_START;

      for (i = 0; i < population[index_best].length; i++)
            {
            direction(population[index_best].gene[i], index_best);
            fprintf(bestway,"\n%3.0f; %3.0f",population[index_best].xcoord,
population[index_best].ycoord);
            }

fprintf(galog,"\n\n Simulation completed\n");
fprintf(galog,"\n Best member: \n");
fprintf(galog,"\n\n Best fitness = %3.3f",population[index_best].fitness);
```

```
fprintf(galog,"\n\n Weglaenge = %3.3f",population[index_best].way);
fprintf(galog,"\n\n Position Error = %3.3f",population[index_best].p_error);
fprintf(galog,"\n\n Collisions = %f",population[index_best].collision);

fclose(galog);
fclose(bestway);

printf("Success\n");

    return 0;
}
```

B. Programming the Nomad 200

In principle we have two possibilities to run the Nomad from a program (Fig. B.1):

- **Direct Mode**: the program communicates directly with the robot daemon .
- **Client Mode**: the program communicates as a client to the server. This mode will always be used when testing a new program.

The Graphic User Interface (GUI) provides a convenient access to the real and simulated robots, and to the representation of the world, as shown in Fig. B.2. Through the GUI, the user can send commands to robots, monitor command execution by seeing the robot actually moving on the screen, visualize instantaneous and cumulated senor data. The software runs on a Linux operating system and shows four windows (see Fig. B.3). The left window shows the simulated robot environment, the middle window the real robot path and the left windows the state of the infrared and sonar sensors.

As already mentioned, the Nomad 200 is equipped with a variety of sensors, which are controlled by the micro controller and sent to the server. The actual states of the robot are saved in the following parameters:

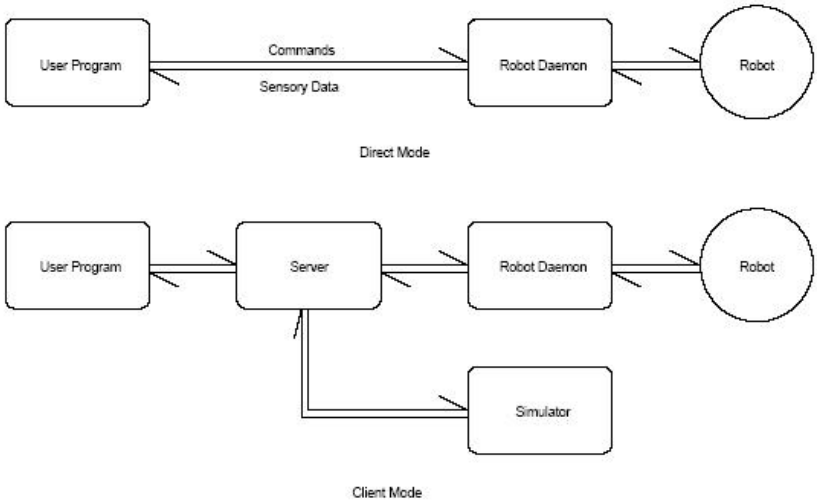


Fig. B.1.: Programming the Nomad in two possible modes.

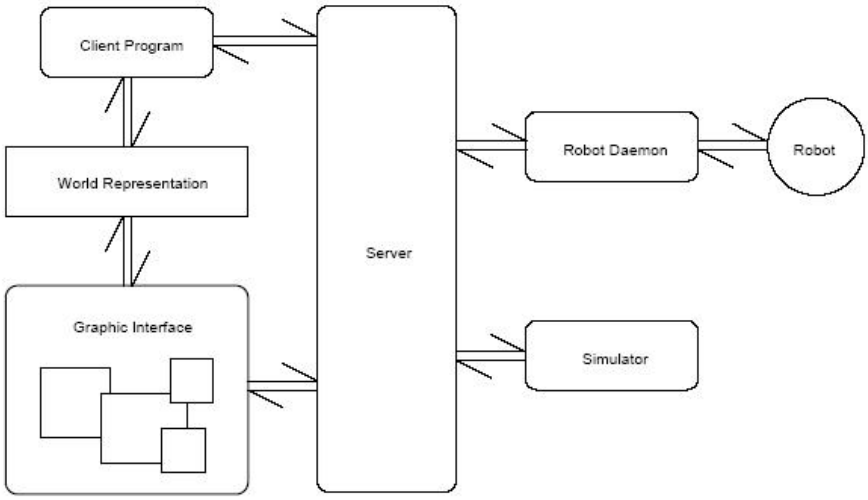


Fig. B.2.: The graphic interface of the programming environment.

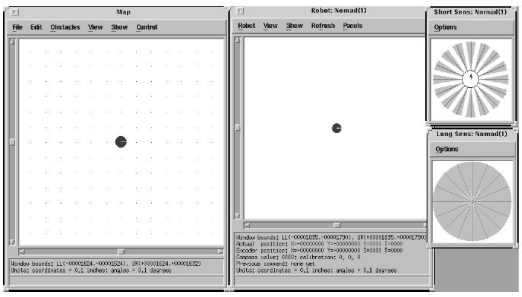


Fig. B.3.: Nomad200 Simulation Environment

- Position(`STATE_CONF_X, STATE_CONF_Y`)
  `STATE_CONF_X` : x-coordinate of the robot
  `STATE_CONF_Y` : y-coordinate of the robot
- Orientation(`STATE_CONF_STEER, STATE_CONF_TURRET`)
  `STATE_CONF_STEER` : Steering angle
  `STATE_CONF_TURRET`: Turret angle
- Sensor states (`STATE_SONAR_0 .. STATE_SONAR_15, STATE_IR_0 .. STATE_IR_15, STATE_BUMPER`)
  `STATE_SONAR_0 ... STATE_SONAR_15` : Distance measure of the 16 sonar sensors
  `STATE_IR_0 ... STATE_IR_15` : Distance measure of the 16 infrared sensors
  `STATE_BUMPER` : Bumper data

The sonar and infrared sensors are placed around the robot (Fig. B.4,B.5) and change their shape, when approximating an obstacle (Fig. B.6,B.7).

The Control Software is written in ANSI C++, whereas a variety of additional commands are used. Programming the Nomad requires the following steps:

- Establish communication with a robot .
- Initialize the robot and its sensors .

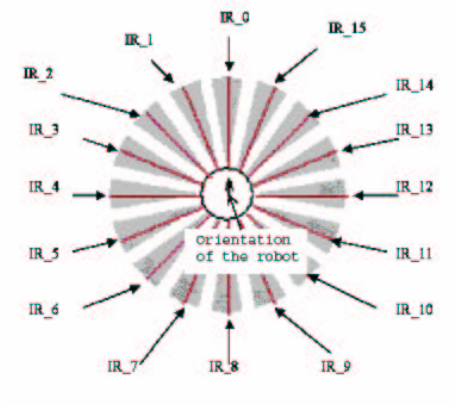


Fig. B.4.: The adjustment of the IR Sensors.

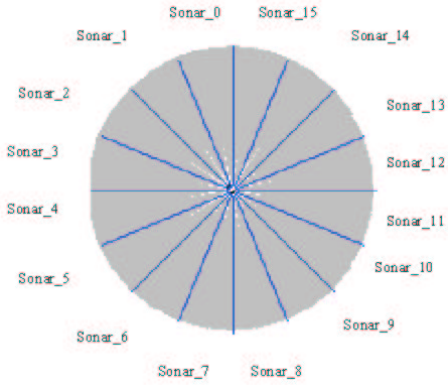


Fig. B.5.: The adjustment of the IR Sensors.

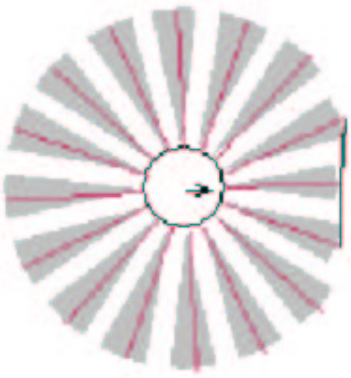


Fig. B.6.: IR Sensor State.

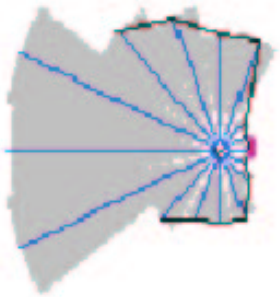


Fig. B.7.: Sonar Sensor state.

- Repeat until done
  - Send motion and sensing commands to the robot
  - Get motion and sensing data from the robot
- Disconnect from robot .

In general there exist three basic classes of robot specific commands :

- Communication commands, in order to establish a connection between the robot and the server.
- Motion commands to move the robot and to obtain its current configuration .
- Commands to configure and readout the sensory data.

A common property of almost all of the commands is that they update the global vector `State`. The value returned by the functions themselves is `TRUE` if the command was successfully transmitted to the robot and state information came back correctly. It is **not** an indication that the command was successfully completed. The reason of this is that commands are executed asynchronously: the function itself will return immediately, and while the intended action starts on the robot, the program will move on to the next instruction. For instance, if one send `pr(1000,0,0)`, a command that tells the robot to move forward by 2.54 m, the command will return immediately (and probably even before the robot is actually moving). If the program's next line is `pr(-1000,0,0)`, this will cause the robot to stop the previous motion and start this next one, requesting a move into the opposite direction. The only exceptions are the commands `zr` and `ws`, that initialize the robot's encoders and wait for the robot to stop, respectively. To get some impression of how the programming of the Nomad works, we show a simple example :

```
#include "Nclient.h"

void main()
```

```
{
connect_robot(1);
zr();
sp(50,0,0);
pr(1000,0,0);
while(State[STATE_CONF_X)<1000)
gs();
disconnect_robot(1);
}
```

This program:

- connects to the robot
- initializes it using the command `zr`
- sets the translational speed to 0.127 m/s, the speeds of the two rotational degrees of freedom to zero
- translates the robot by 2.54m
- gets the robot state during the motion
- disconnects from the robot

The include file `Nclient.h` contains the prototypes of the robot commands.

The Robot window (Fig.B.8) allows interactive control of a robot. At the bottom of the window we find information about the current robot position, compass value, and the last command issued. In position information, `X` and `Y` are the coordinates, `S` is the steering direction in degrees, `T` is the turret direction in degrees. Degrees range from 0 to 360 with 0 as the horizontal right. The simultaneous display of the actual position and that position, which is calculated via the encoder, can monitor the effects of external disturbances on the robot.

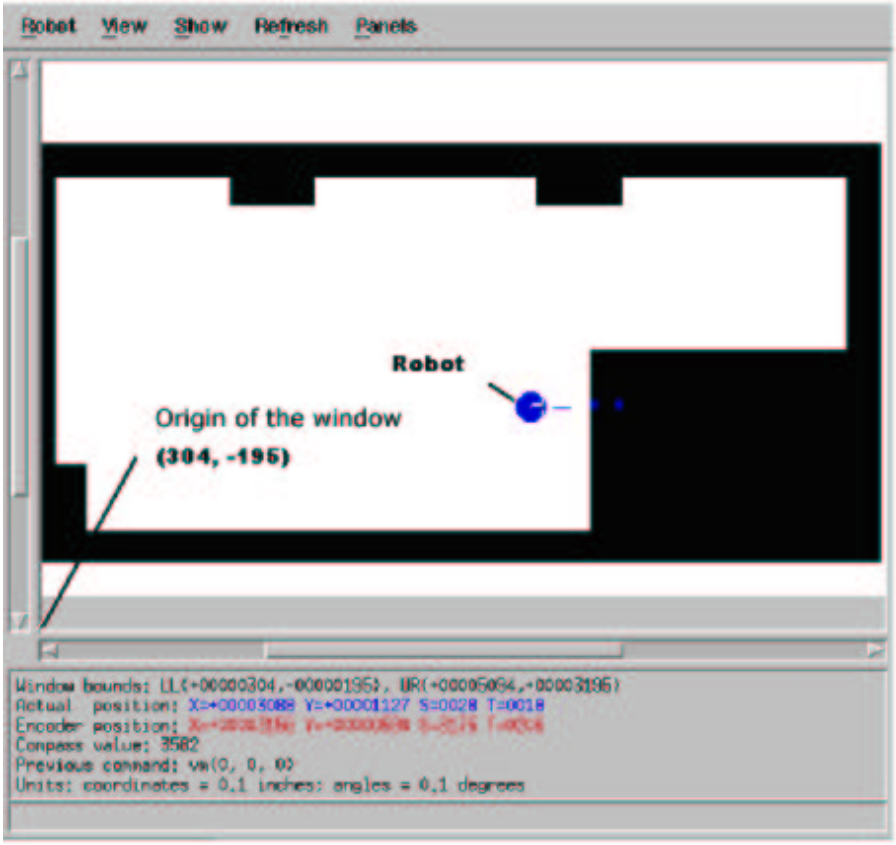


Fig. B.8.: The robot window

List of Figures

Bibliography

[1] World Robotics (2004). *Statistics, Market Analysis, Forecasts, Case Studies and Profitability of Robot Investment*. United Nations Economic Commission for Europe, Geneva .

[2] Burgard W., Cremers A., Fox D., Hahnel D., Lakemeyer G., Schulz D., Steiner W. and Thrun S. (1998). *The Interactive Museum Tour-Guide Robot*. In Proceedings of the 15th National Conference on Artificial Intelligence. Madison, Wisconsin. AAAI Press.

[3] Kiesler S., Hinds P. (2004). *Introduction to This Special Issue on Human-Robot Interaction* Human-Computer Interaction Vol. 19 No. 1 2 (1-8) .

[4] Kopacek P., Han M.-W., Putz B. Schierer E., Würzl M., (2004). *A concept for a humanoid demining robot.* Proceedings of the International IARP Workshop on Robotics and Mechanical assistance in Humanitarian Demining and similar risky interventions, 16-18. June 2004, Brussels-Leuven/Belgium .

[5] Miller D., Slack M. (1995). *Design and testing of a low-cost robotic wheelchair.* Automomous Robots Vol. 2 77-88.

[6] Kuffner J.J. Jr. (1999). *Autonomous Agents for Real-time Animation* . PhD theses, Stanford University CA .

[7] Apaydin M.S., Brutlag D.L., Guestrin C., Hsu D., Latombe J.-C. (2002). *Stochastic Roadmap Simulation: An Efficient Representation and Algorithm for Analyzing Molecular Motion.* Proc. RECOMB'02, Washington D.C., 12-21.

[8] Koestler A. (1967). *A Ghost in the Machine.* Arkana Books, London.

[9] Nehmzow U. (2003). *Mobile Robotics. A Practical Introduction* Springer, London.

[10] Siegwart R., Nourbakhsh I.R. (2004). *Introduction to Autonomous Mobile Robots.* MIT Press, Cambridge, MA.

[11] Bräunl T. (2003) *Embedded Robotics.* Springer, Berlin.

[12] Kim J.-H., Kim D.-H., Kim Y.-J., Seow K.-T. (2004). *Soccer robotics.*(Springer Tracts in Advanced Robotics) Springer, Berlin.

[13] Han M. W., Kopacek P., Putz B., Würzl M., Schierer E. (2003). *Robot Soccer - A First Step to Edutainment.* Proceedings of RAAD'03. 12th International Workshop on Robotics in Alpe-Adria-Danube Region, Cassino .

[14] Nardi D. (Edit.) (2005). *RoboCup 2004: Robot Soccer World Cup VIII.* Springer, Berlin.

[15] Weiss G. (Edit.) (2000). *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence.* MIT Press, Cambridge, MA.

[16] Wooldridge M. (2002). *Introduction to MultiAgent Systems.* John Wiley & Sons.

[17] Paolucci M., Sacile R. (2005). *Agent-Based Maufacturing and Control Systems* . CRC Press, Boca Raton.

[18] Wooldrige M,, Jennings N.R. (1995). *Intelligent Agents: theory and practice.* Knowledge Eng. Rev. 10, pp. 115.

[19] Gonzales R.C., Woods R.E. (2002). *Digital Image Processing.* Prentice Hall.

[20] Horn B. (1986). *Robot Vision.* MIT Press.

[21] Schraft R.D., Hägele M., Wegener K. (2005). *Service Roboter Visionen.* Hanser, München.

[22] Harnad S. (1990). *The Symbol Grounding Problem.* Physica D 42: 335-346.

[23] Pfeifer R., Scheier C. (2001). *Understanding Intelligence.* MIT Press, Cambridge, MA.

[24] Brooks R.A. (1991). *Intelligence without Reason.* Proc. IJCAI 91 (1) 569-595 .

[25] Arkin R. (1998). *Behavior Based Robotics.* MIT Press, Cambridge, MA.

[26] Ilon B.E. (1975). *Wheels for a course stable selfpropelling vehicle movable in any desired direction on the ground or some other base.* US Patents and Trademarks office, Patent 3,876,255 .

[27] Åström K, Hägglund T. (1995). *PID Controllers: Theory,Design and Tuning.* Instrument Society of America, Research Triangle Park NC.

[28] Berg M. de et al. (2000). *Computational Geometry.* Springer.

[29] Dudek G., Jenkin M. (2000). *Computational Principles of Mobile Robotics.* Cambridge University Press .

[30] Dijkstra E. (1955). *A note on two problems in connexion with graphs* Numerische Mathematik, Vol.1 (3) 269-271 .

[31] Latombe J.-C. (1991). *Robot Motion Plannning.* Kluwer .

[32] Laumond J.-P. (Edit.)(1998). *Robot Motion Planning and Control .* Springer LNCIS 229 .

[33] LaValle S.M. (2005). *Planning Algorithms.* To appear, http://msl.cs.uiuc.edu/planning

[34] Choset H., Lynch K.M., Hutchingson S., Kantor G., Burgard W., Kavraki L., Thurn S. (2005). *Principles of Robot Motion.*MIT Press, Cambridge, MA.

[35] Khatib O. (1986) . *Real-time obstacle avoidance for manipulators and mobile robots.* Int. Jour. Robotics Research 5, 90-98 .

[36] Khatib M., Chatila R. (1995) . *An extended potential field approach for mobile robot sensor-based motions.* Proc. Intell. Auton. Syst. IAS 4, IOS Press, Karlsruhe, 490-496.

[37] Lee D.T., Drysdale III R.L. (1981) *Generalized Voronoi diagrams in the plain.* SIAM J. Comput. 10 (1), 73 - 87 .

[38] Lozano- Perez T., Weley M. (1979) . *An algorithm for planning collision-free paths among polyhedral obstacles.* Communications of the ACM 22(10) , 560-570.

[39] Aurenhammer F. (1991). *Voronoi diagramms - A survey of a fundamental geometric structure.* ACM Computing Surveys 23, 345-405.

[40] Reif J. (1979). *Complexity of the mover's problem and gernalizations.* Proc. 20th IEEE Symp. Found. Comput. Scien. 421-427.

[41] Feder H.J.S., Slotin J.-J.E. (1997). *Real-Time path planning using harmonic potentials in dynamic environments.* Proc. IEEE Int.Conf. on Robotics and Automation, Albuquerique, NM.

[42] Amato N., Wu Y., (1996). *A randomized roadmap method for path and manipulation planning.* Proc. IEEE Int. Conf. on Robotics and Automation, 113- 120.

[43] Kavraki L., Latombe J.C. (1994). *Randomized preprocessing of configuration space for fast path planning.* Proc. IEEE Int. Conf. on Robotics and Automation, 2138-2145.

[44] Sv̌estka P. (1997). *Robot motion planning using probabilistic roadmaps.* PhD thesis, Utrecht Univ.

[45] Overmars M.H. (1992). *A random approach to motion planning*, Technical Report RUU-CS-92- 32, Dept. Comput. Sci., Utrecht Univ., Utrecht, Ned.

[46] LaValle S.M. , Kuffner J.J. (1999). *Randomized kinodynamic planning.* IEEE Int. Conf. Robotics and Automation, 473-479 .

[47] LaValle S.M. , Kuffner J.J. (2001). *Rapidly-exploring random trees: Progress and prospects.* In B. R. Donald, K. M. Lynch, and D. Rus, [edit.], *Algorithmic and Computational Robotics: New Directions*,A K Peters, Wellesley, 293-308.

[48] LaValle S. M. (1998). *Rapidly-exploring random trees: A new tool for path planning.* TR 98-11, Computer Science Dept., Iowa State University.

[49] Barraquand J., Latombe J.-C. (1990). *A Monte-Carlo algorithm for path planning with many degrees of freedom.* In Proc. IEEE Int. Conf. Robot. & Autom., 1712-1717.

[50] Amato N. M., Bayazit O. B., Dale L. K. , Jones C., Vallejo D. (1998).*OBPRM: An obstacle-based PRM for 3D workspaces.* Proc. of the Workshop on Algorithmic Foundations of Robotics, 155 - 168.

[51] Boor V., Overmars N. H., van der Stappen A. F..(1999). *The Gaussian sampling strategy for probabilistic roadmap planners.* In Proc. IEEE Int. Conf. Robot. & Autom., 1018-1023.

[52] Hsu D., Jiang T., Reif J., Sun Z. (2003). *The bridge test for sampling narrow passages with probabilistic roadmap planners.* Proc. IEEE Int. Conf. Robot. & Autom.

[53] Yershova A., Jaillet L., Simeon T., LaValle S.M. (2005). *Dynamic-Domain RRTs: Efficient Exploration by Controlling the Sampling*

Domain. Proc. IEEE International Conference on Robotics and Automation, to appear .

[54] Strandberg M. (2004). *Augmenting RRT planners with local trees.* IEEE Intern. Conf. on Robotics and Automation, 3258-3262 .

[55] Plaku E., Bekris K. E., Chen B. Y., Ladd A. M., Kavraki L. E. (2005). *Sampling-Based Roadmap of Trees for Parallel Motion Planning.* IEEE Transactions on Robotics, to appear .

[56] Wilmarth S. A., Amato N. M., and Stiller P. F. (1999). *MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space.* IEEE Int. Conf. on Robotics and Automation, 1024-1031.

[57] Branicky M.S., LaValle S.M. (2002). *On the relationship between classical grid search and probabilistic roadmaps.* Proc. Workshop on the Alg. Found. of Robotics .

[58] Branicky M.S., LaValle S.M., Olson K., Yang L. (2001). *Quasi-randomized path planning.* Proc. IEEE Int. Conf. on Robotics and Automation, 1481-1487 .

[59] Lindemann S. R., LaValle S. M. (2004). *Current issues in sampling-based motion planning.* In P. Dario and R. Chatila, [edit.], Proc. Eighth Int. Symp. on Robotics Research. Springer-Verlag, Berlin. To appear.

[60] Baker J. E. (1985). *Adaptive Selection Methods for Genetic Algorithms.* Proc. ICGA 1, pp. 101-111 .

[61] Baker J. E. (1987). *Reducing bias and inefficiency in the selection algorithm.* Proc. 2nd Int. Conf. on Genetic Algorithms and their Applications, pp. 14-21.

[62] Fogel. L. J., Owens A. J., Walsh M. J. (1966). *Artificial Intelligence through simulated evolution.* John Wiley, New York.

[63] M. Maxfield, A. Callahan , L.J. Fogel (Eds.) (1965). *Biophysics and Cybernetic Systems.* Proc. of the 2nd Cybernetic Sciences Symposium, 131 - 155.

[64] Friedberg R. M. (1958). *A Learning Machine: Part I.* IBM Journal of Research and Development, 2(1), 2-13.

[65] Friedman G.J. (1956). *Selective feedback computers for enigneering synthesis and nervous system analogy.* Master Theses, University of California, Los Angeles.

[66] Goldberg D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley Publishing Company .

[67] Goldberg D. E., Deb K. (1991). *A comperative analysis of selection schemes used in Genetic Algorithms.* Rawlins G.J.E. [Edit.] *Foundations of Genetic Algorithms.* Morgan Kaufmann Pub., San Mateo, USA, 69-93.

[68] Hart P., Nilsson N., Raphael B. (1968). *A formal basis for the heuristic determination of minimum cost paths .* IEEE Transanctions on Systems Science and Cybernetics, vol SSC-4 No.2 (8) 100-107.

[69] Holland J.H. (1969). *A new kind of turnpike theorem .* Bulletin of the American Mathematical Society, 75(6), 1311 -1317.

[70] Holland J.H. (1975). *Adaption in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, MI.

[71] Koza J.R. (1992). *Genetic Programming: in the programming of computers by means of natural selection.* MIT Press, Cambridge, MA.

[72] Michalewicz Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs.* Springer.

[73] Weicker K. (2002). *Evolutionäre Algorithmen .* Teubner.

[74] Mitchell, M. (1996). *An introduction to Genetic Algorithms.* MIT Press, Cambridge, MA.

[75] Pohlheim H. (2000). *Evolutionäre Algorithmen.* Springer.

[76] He L., Mort. N. (2000). *Hybrid genetic algorithms for telecommunications network back-up routeing.* BT Technology Journal, 18(4), 42-50.

[77] Rizki M., Zmuda M., Tamburino L. (2002). *Evolving pattern recognition systems.* IEEE Trans. on Evolutionary Computation, 6(6), 594-609 .

[78] Schwefel H.-P., Wegener I., Weinert K. (Eds.) (2003). *Advances in Computational Intelligence.* Springer.

[79] Vankeerberghen P., Smeyers-Verbeke J., Leardi R., Karr C.L., Massart D.L. (1995). *Roboust Regression and Outlier Detection for nonlinear Models using Genetic Algorithms.* Chemometrics and Intelligent Laboratory Systems, 28 (1995) 73-87 .

[80] Schwefel H. P., Männer R. [Edit.] (1990). *Parallel Problem Solving from Nature: 1st Workshop.* Lecture Notes in Computer Science No. 496, Springer.

[81] Fleming P., Purshouse R.C. (2002). *Evolutionary algorithms in control systems engineering: a survey.* Control Engineering Practice, 10, 1223-1241.

[82] Reeves C., Rowe J. (2002) . *Genetic Algorithms: Principles and Perspectives.* Kluwer, Norwell MA.

[83] Liu M., Wu C. (2003). *Scheduling algorithm based on evolutionary computing in identical parallel machine production line.* Robotics and Computer Integreted Manufacturing 19, 401-407 .

[84] Coello C. (2000) *An updated survey of GA-based multiobjective optimization techniques.* ACM Computing Surveys, 32 (2), 109-143.

[85] Fonseca C.,Fleming P. (1995). *An overview of evolutionary algorithms in multiobjective optimization.* Evolutionary Computation, 3(1), 1-16.

[86] Nolfi S., Floreano D. (2000). *Evolutionary Robotics.*, MIT Press, Cambridge, MA.

[87] Eshelman L.J. , J.D. Schaffer (1991). *Preventing premature convergence in Genetic algorithm by preventing incest.* R.K. Belew, L.B. Booker [Edit.], Proc. 4th Int. Conf. GA (1), 115-122.

[88] Nearchou A.C. (1999). *Adaptive navigation of autonomous vehicles using evolutionary algorithms.* Artificial Intelligence in Engineering 13, 159-173.

[89] Nearchou A.C. (1999). *A Genetic navigation algorithm for autonomous mobile robots.* Cybernetics and Systems 30, 629-661.

[90] Xiao J, Michalewicz Z., Zhang L., Trojanowski K. (1997). *Adaptive Evolutionary Planner/Navigator for Mobile Robots.* IEEE Trans. on Evolutonary Computing 1(1) 18-28.

[91] Tu J., Yang S.X. (2003). *Genetic Algorithm Based Path Planning for a Mobile Robot.* Proc. of IEEE Int. Conf. on Robotics & Automation, 1221 - 1226.

[92] Panda A.M., Dash R.R., Mishra S., Singh K.C. (2000). *Off-line and On-line path planning of mobile robot in an environment of static obstacles using evolutionary computation algorithm.* Jour. Electrical & Electronics Engineering, Australia, 20(3), 211-223.

[93] Hu Y., Yang S. (2004). *A knowledge based Genetic algorithm for path planning of a mobile robot.* Proc. IEEE Int. Conf. Robotics & Automation, New Orleans, LA, 4350-4355.

Internet Links

[94] http://www.honda-robots.com .

[95] http://www-2.cs.cmu.edu/∼illah/SAGE/index.html

[96] http://www.care-o-bot.de/MuseumRobots.php

[97] http://www.ipa.fhg.de/Arbeitsgebiete/robotersysteme/service/service_sich.php

[98] http://www.ipa.fhg.de/Arbeitsgebiete/robotersysteme/service/service_haushalt.php

[99] http://www.whoi.edu/marops/vehicles/argo/index.html .

[100] http://www.care-o-bot.de .

[101] http://robotics.ee.uwa.edu.au/eyebot/doc/robots/omni.html .

[102] http://www.robosoccer.at/robygo/frameset-eng.html .

[103] http://www.swisslog.com/hcs-index/hcs-systems/hcs-agv.htm .

[104] http://www.darpa.mil/grandchallenge/index.html

[105] http://newton.ex.ac.uk/teaching/CDHW/Feedback/Setup-PID.html .