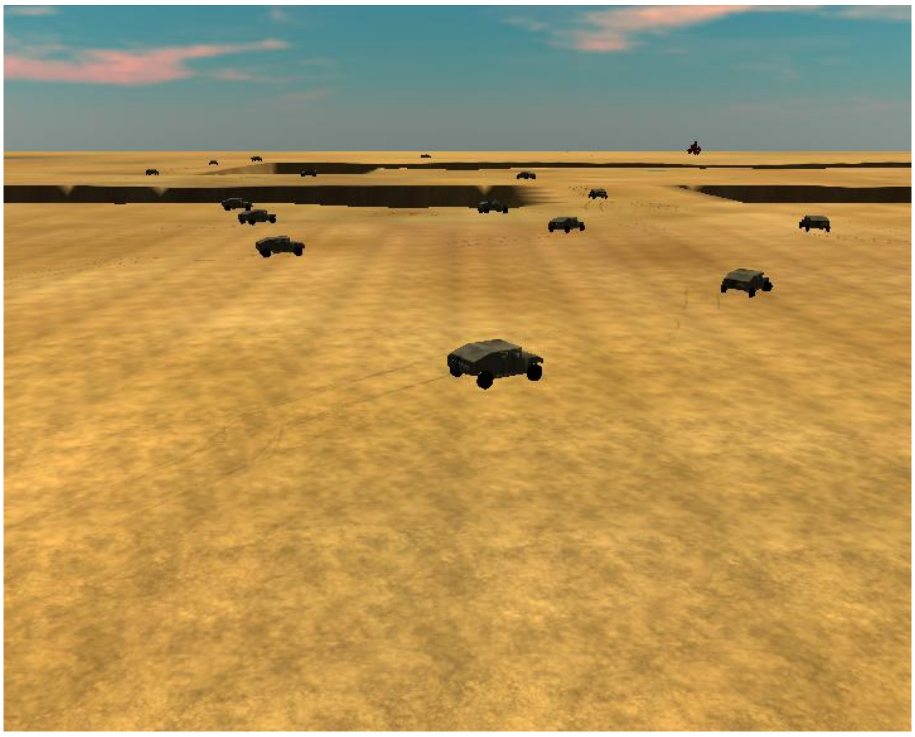
SWEDISH DEFENCE
RESEARCH AGENCY

FOI-R--1383--SE
October 2004
1650-1942

Scientific report

Magnus Lindhé

A flocking and obstacle avoidance algorithm for mobile robots



System Technology Division
SE-172 90 STOCKHOLM
Sweden

Swedish Defence Research Agency
System Technology Division
SE-172 90 STOCKHOLM
Sweden

FOI-R--1383--SE
October 2004
1650-1942

Scientific report

Magnus Lindhé

A flocking and obstacle avoidance algorithm for mobile robots

Issuing organization Swedish Defence Research Agency System Technology Division SE-172 90 STOCKHOLM Sweden	Report number, ISRN FOI-R--1383--SE	Report type Scientific report
	Research area code Combat	
	Month year October 2004	Project no. E6003
	Customers code Commissioned Research	
	Sub area code Weapons and Protection	
Author/s (editor/s) Magnus Lindhé	Project manager Peter Alvå	
	Approved by Monica Dahlén	
	Sponsoring agency Swedish Armed Forces	
	Scientifically and technically responsible Karl Henrik Johansson	

Report title

A flocking and obstacle avoidance algorithm for mobile robots

Abstract

Mobile robots cooperating in groups offer several advantages such as redundancy and flexibility and can sometimes perform tasks that would be impossible for one single robot. We have developed a new algorithm for navigating a group of robots to a predefined target while avoiding obstacles and staying together in a formation. The algorithm merges a potential-based method that provides the shortest unobstructed path to the goal with an approach from coverage control, where the centroids of Voronoi partitions are used to optimize the placement of sensors in a scalar field.

Our algorithm guarantees safety: there will be no collisions between robots or with obstacles of arbitrary shape. It has shown reliable goal convergence in simulations, and the robots form a hexagonal lattice when moving in open fields. Further, the algorithm is completely decentralized and the processing load for each robot is virtually unaffected by adding more group members.

This report provides a description of the algorithm and its properties, as well as results from simulations in both Matlab and Fenix, a graphical simulation environment developed at FOI, where the robots are modeled as the US Army HMMWV all-terrain vehicle.

Keywords

Flocking, Obstacle Avoidance, Car-like robots, Cooperative control

Further bibliographic information	Language English
ISSN 1650-1942	Pages 52
Distribution By sendlist	Price Acc. to pricelist Security classification Unclassified

Utgivare Totalförsvarets forskningsinstitut Avdelningen för Systemteknik SE-172 90 STOCKHOLM Sweden	Rapportnummer, ISRN FOI-R--1383--SE	Klassificering Vetenskaplig rapport
	Forskningsområde Bekämpning	
	Månad, år Oktober 2004	Projektnummer E6003
	Verksamhetsgren Uppdragsfinansierad verksamhet	
	Delområde VVS med styrda vapen	
Författare/redaktör Magnus Lindhé	Projektledare Peter Alvå	
	Godkänd av Monica Dahlén	
	Uppdragsgivare/kundbeteckning Försvarsmakten	
	Tekniskt och/eller vetenskapligt ansvarig Karl Henrik Johansson	

Rapportens titel

En navigationsalgoritm för flockar av autonoma mobila robotar

Sammanfattning

Mobila robotar som samarbetar i grupp ger flera fördelar, t.ex. redundans och flexibilitet. Dessutom ger det möjlighet att lösa vissa uppgifter som vore omöjliga för en ensam robot. Vi har utvecklat en ny algoritm för att förflytta grupper av robotar till ett förutbestämt mål, utan att de kolliderar med varandra eller med hinder. På vägen håller de ihop i en formation. Vi har kombinerat en potentialbaserad metod, som används för att hitta kortaste vägen till målet, och en yttäckningsmetod som utnyttjar tyngdpunkterna för Voronoi-regioner för att optimera placeringen av sensorer i ett skalärt fält.

Vår algoritm är bevisat säker mot kollisioner, oavsett form på hindren. Den har visat sig ge pålitlig konvergens till målet i simuleringar och robotarna bildar hexagonala gitterformationer när de rör sig på öppna ytor. Algoritmen är dessutom helt decentraliserad och mängden beräkningar som varje robot behöver göra ökar försumbart när fler gruppmedlemmar tillkommer.

Den här rapporten beskriver algoritmen och dess egenskaper och redovisar resultat från simuleringar i två olika miljöer. Dels har vi simulerat i Matlab och dels i Fenix, en grafisk datormiljö utvecklad av FOI, där robotarna har modellerats efter amerikanska arméns terrängfordon HMMWV.

Nyckelord

Formationer, hinderundvikande, robotbilar, samverkande robotar

Övriga bibliografiska uppgifter	Språk Engelska
ISSN 1650-1942	Antal sidor 52
Distribution Enligt missiv	Pris Enligt prislista Sekretess Öppen

Contents

Preface

This report, written at the Department of Autonomous Systems at the Swedish Defence Research Agency (FOI), constitutes my master's thesis. It is the final element of my Master of Science in Electrical Engineering at the Royal Institute of Technology (KTH).

The last years of my studies have been funded by the Armed Forces Engineer Programme, aiming at providing the Swedish Armed Forces with specialized officers who have both a military background as well as a Master of Science. As FOI represents the highest level of military research and development in Sweden, it was a natural choice to apply for a master's thesis project here.

The work has been supervised by Petter Ögren at the Department of Autonomous Systems and by Karl-Henrik Johansson at the Department of Sensors, Signals and Systems at KTH. Karl-Henrik Johansson has also acted as examiner.

A shorter version of this report was presented by the author at the Reglermöte 2004, at Chalmers University of Technology in Gothenburg, [1].

Work planning and experiences

This project got off to a flying start in the beginning of February, when Petter presented his idea of combining coverage control with obstacle avoidance. We decided to try to produce the outline of a paper by the beginning of March, so I had to quickly take in the papers on coverage control before starting on the Matlab code for the simulations. During the next four weeks I simulated the algorithm and came up with problems that Petter and I had to solve. After the basic framework was designed and tested, I spent some time documenting the first phase of the project and studying how the algorithm could be extended to sensor positioning in the target area.

The second phase was to implement it in Fenix, which made me have to brush up on my C++ skills before understanding the structure of the software. The actual programming took five weeks and then I spent two weeks documenting and making screenshots and films for the presentations. The final phase of the work has been to present it at the Reglermöte and work on the properties of formation stability and goal convergence.

During the work I have learned a lot about robotics in general and computational geometry and path planning in particular. It also gave me a better understanding of research methodology, which is not taught much in other undergraduate courses. Attending the Reglermöte was very rewarding, both professionally and personally, as I got some insight into current issues in the control community as well as inspiration for possible future postgraduate studies.

Acknowledgements

The first mentioning of course goes to my supervisor Petter, as thanks for all his scientific as well as moral support. He has always been available to help whenever innocent, albeit virtual, robots have lost their lives due to my mistakes or flaws in the algorithm. Most of the ideas contained in this report originate from him.

I also owe thanks to Emil Salling at FOI, who has spent many hours customizing Fenix for my application and editing the screenshot films that I showed at the Reglermöte. One day I will summon the courage to ask my future boss for a flatscreen like Emil's...

Had it not been for Karl-Henrik Johansson at S3, I would probably not have found this master's thesis project and he has been a valuable sounding board for the development of the algorithm as well as the presentation of it.

Finally I would like to thank all those who have made my time at FOI pleasant, be it as room-mates, Fenix technicians, spontaneous coffee-room lecturers, rewarding lunch company or otherwise. I hope to see you again in my future career!

Stockholm, 11th of June 2004

Magnus Lindhé

1. Introduction

1.1 Problem definition

The object of this project has been to develop and study a new algorithm for moving a group of robots from one place to another, without colliding with each other or any obstacles. While doing so the group should stay together, preferably in a geometrically defined formation. The algorithm was then to be tested in a realistic setting. FOI has a testbed for robotics applications, consisting of rebuilt radio-controlled cars and Fenix, a graphical computer simulation environment. The interface between the system and the controller is the same in both environments, so the same control program can be run both in the cars and in the computer. Since there are currently only two robots available in hardware, a decision was made to implement the system in Fenix. The developed algorithm is a combination of two recently presented research papers, on navigation functions, [2] , and coverage control, [3]. The idea was to extend them to get the desired formation properties and eventually study the issues of safety, goal convergence and under what conditions formations were formed.

The main difficulty when trying to achieve goal convergence, flocking and collision avoidance is that of prioritizing the sometimes contradictory requirements. Figure 1.1 illustrates the chosen order of priorities:

1. The highest priority is given to **safety** , i.e. collision avoidance.
2. The second priority is **goal convergence**. For collision safety the robots should of course not all reach the exact goal coordinates, but rather a set of equilibrium positions, tightly arranged around it.
3. Finally the robots should stay together in a flock while moving. We call this **flock cohesion**.

We consider a group of kinematic robots p_i that have the following dynamics:

$$\begin{aligned} p_i(k+1) &= p_i(k) + u_i(k) \\ ||u_i|| &\leq u_{max} \end{aligned}$$

Each robot is equipped with a sensor that can detect neighboring agents within the radius R_{max}. Equivalently, it can have some means of communication, used to inquire about the position of other robots within R_{max}. We believe this to be a more realistic assumption than the one made in [3] , where the sensors are assumed to have infinite range. It is also assumed that the positioning problem (described in the background section) has been solved, so that we have access to an accurate position estimate. All robots have processing capabilities that allow them to locally run the algorithm that will be presented. Finally they have knowledge of the location of all obstacles. This is not always true in real-world implementations, but then the robots can be equipped with a sensor for obstacles and the navigation function can be calculated with respect to all obstacles in view from the robot. All other areas are assumed to be free space. If the robot later discovers more obstacles, they can be added to the map and the navigation function is recalculated. This can lead to the robot going

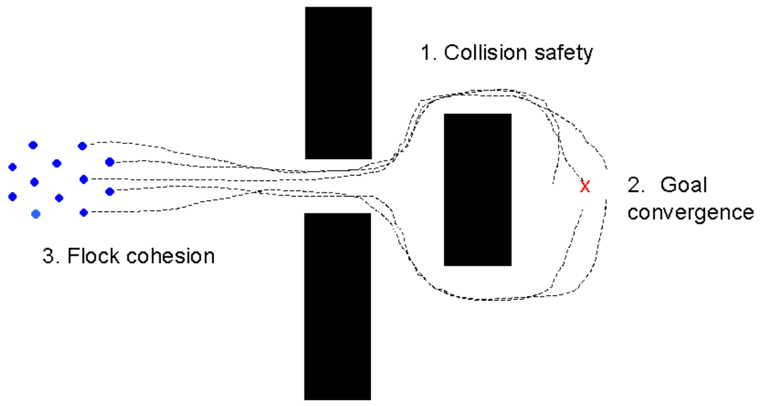


Figure 1.1: A group of robots moving around some obstacles before stopping around the goal, marked by an X. This illustrates the chosen order of priorities for the algorithm. Safety is given the highest priority, then comes goal convergence and finally flock cohesion.

into dead ends, but as soon as it discovers this the map will be redrawn and it will try another, presumably better, path.

Robots with more realistic dynamics, such as car-like vehicles, can also be controlled with the proposed algorithm. As an example of this, we use the following model in Fenix (deduced in section 5.4):

$$\begin{aligned}\dot{x} &= v\cos\theta\\ \dot{y} &= v\sin\theta,\\ \dot{\theta} &= \frac{v}{L}\tan\delta\\ \dot{v} &= u\end{aligned}$$

This requires a hierarchical controller structure, as depicted in Figure 1.2. The high-level algorithm uses the position of the vehicle and that of the neighbors to calculate the next waypoint as well as a region where the vehicle controller is free to manoeuvre. This ensures collision-safety even in the case of non-holonomic dynamics, when the vehicle may need to make parallel parking movements to reach its waypoint.

1.2 Report outline

The report starts with an introduction to robotics, in section 2. It motivates the use of robots in general and groups of robots moving in formations in particular. It also contains a presentation of some of the most important hardware building blocks used in robotics, to give an understanding of some considerations that have governed the development of the algorithm. The theoretical core of the report is contained in section 3. It briefly accounts for the concepts of navigation functions and coverage control before presenting our new algorithm. It ends with a section on proven or theoretically predicted properties of the algorithm.

Then two sections follow, describing the simulations first in Matlab and then in Fenix. They both present the results of the simulations and some problems that have been encountered when implementing the algorithm on that specific platform. The section on Fenix also contains some information on the structure of the software and an account of how a controller was developed to drive the car between waypoints designated by the higher-level algorithm.

Thus far the report considers the problem of only navigating the robots. In section 6 the problem is extended to also consider different sensing tasks in an area surround-

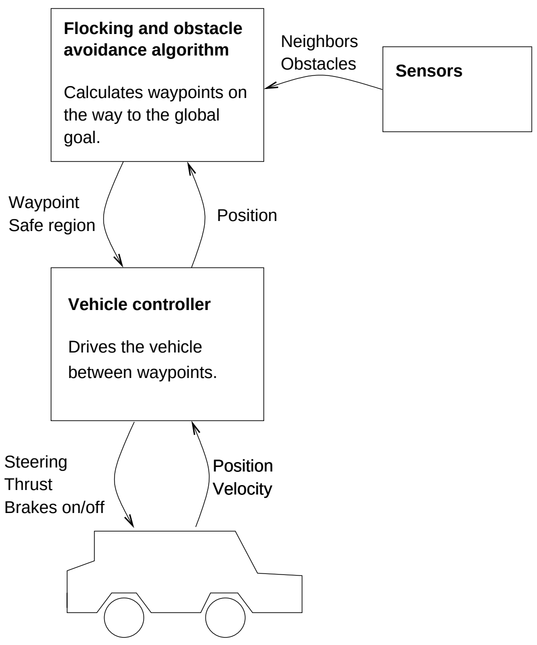

Flocking and obstacle avoidance algorithm
Calculates waypoints on the way to the global goal.
Neighbors
Obstacles
Sensors
Waypoint
Safe region
Position
Vehicle controller
Drives the vehicle between waypoints.
Steering
Thrust
Brakes on/off
Position
Velocity

Figure 1.2: The hierarchical controller structure used to control vehicles with more realistic dynamics.

ing the designated goal. Some sensor types that could be of interest in a military context are presented, as well as a simple way of switching between navigation and sensing within the framework of the algorithm. Finally there are some conclusions on the algorithm as a whole and an account of some issues that are still open to research.

1.3 Reader's guide

The background section is intended for readers who are new to the field of robotics. Those who want a motivation for formation movement and groups of robots can go directly to section 2.4. All readers should read section 3, as it contains the necessary theory to understand the algorithm, possibly with the exception of section 3.6.

Sections 4 (Simulations in Matlab) and 5 (Simulations in Fenix) are independent and can be read in arbitrary order. Subsection 5.3 is mainly intended for those who want to understand the software and/or develop it further.

Section 6, on deployment in the target area, can be understood separately except for the extension of the navigation algorithm, which requires that one has understood the theory in section 3.

2. Background

This section will provide some background information on robotics in general and mobile robots in particular for readers who are unfamiliar with the field. We give some motivations for the use of robots and then present the underlying technical challenges when designing a mobile robotic system. This will serve to explain the governing limitations when we focus on high-level autonomous navigation and formation keeping in the rest of the report.

Since this work has been carried out at the Swedish Defence Research Agency the main focus is on military applications and we have chosen to limit the scope to ground vehicles. The limitation lies not so much in the general theory as in the assumption on vehicle dynamics in the problem definition and the choice of interesting sensor types in section 6. As an example, an unmanned aerial vehicle (UAV) with aircraft-like design cannot stop and turn on the spot, and in underwater applications radio direction finding or computer vision are not as feasible and interesting as the standard method of acoustic detection and ranging.

2.1 Motivation for robotics

Robots should be used to relieve humans of tasks that fall under the four D:s of robotics: dirty, dull, dangerous or difficult [4]. Much like man has always developed technology to free time for other more challenging tasks, now the time has come for robots. Stationary industrial robots have become standard and many of the key technologies for mobile robots, such as power-efficient yet fast processors, sensors and communication circuits are getting cheaper and smaller by the day. This is driven by their use in consumer products, but as a side effect it makes robotics more economically attractive.

Industrial robots can carry out tasks where humans are exposed to high noise levels, uncomfortable working positions or unhealthy environments such as solvents, spray paint or radioactivity. They also make new products and methods of manufacturing available by giving unprecedented precision to welding and cutting operations. In military applications, mobile robots can do dangerous disposal of explosive ordnance, exposed reconnaissance missions or long-distance transportation. The US Defence Advanced Research Project Agency (DARPA) has shown great interest in the field, issuing the DARPA Grand Challenge, a desert race between Los Angeles and Las Vegas for autonomously navigating offroad cars [5]. The best car managed to drive 11 kilometres before getting off course, which serves to underline the difficulties involved in autonomous navigation.

A more peaceful use for robots will be to aid in caring for the numerous after-war generation of Swedes that will soon be retired and can be expected to need more health care. Simple or unergonomic tasks may be performed by robots to give medical personnel more quality time with the patients. In space, robots have landed on Mars before the technology allows humans to safely travel there. Because of the long distances the robots cannot be remotely operated from Earth, but instead get high-level commands such as waypoints or predefined experiments to perform. Results are then transmitted back to scientists on Earth [6].

2.2 Challenges in mobile robotics

By mobile robot we mean a robot that is not attached, mechanically or electrically, to anything and that is completely self-contained. Such a robot needs to be able to displace itself, find its position, detect obstacles or other objects, communicate with a supervisor and/or other robots and finally process all the data into sensible outputs. All of these requirements present challenges that are subject of inter-disciplinary research.

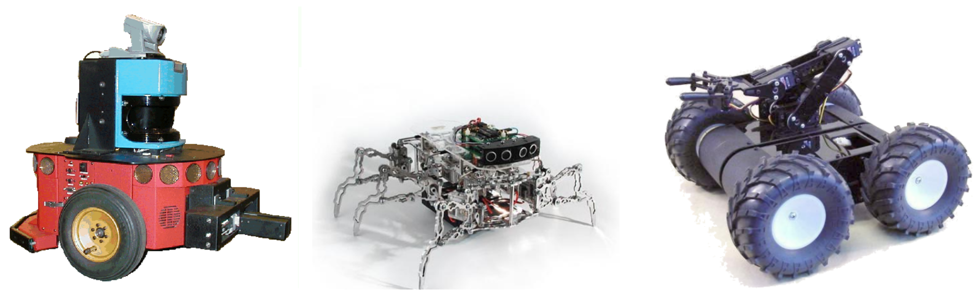


Figure 2.1: Examples of mobile research robots. Left: The Pioneer 2 from ActiveMedia. The “coffee-brewer” is a laser scanner, above which a camera is mounted. Middle: The BrainStem Bug from Acroname, Inc., a six-legged walking robot with ultra-sound sonars for obstacle detection. Right: The Lynxmotion 4WD2 all-terrain chassis with L5 Arm, from Lynxmotion.

2.2.1 Actuators All components that are used to mechanically execute the outputs of the control system are called *actuators*. In the case of robotics they can be wheels, tracks or legs for moving, manipulator arms for handling objects, motors turning a camera head and so on. For the purpose of this overview we will restrict the presentation to popular systems for propulsion of mobile robots.

The three most common methods of traction for mobile robots are wheels, tracks and legs. Wheels have the advantage of being reliable, simple and allowing high speeds. Tracks work better in uneven or soft terrain but are less suited for dead reckoning as they tend to slip, especially when turning. They also use more energy and require more maintenance than wheels. Legged robots are still in the research phase and require much more power and sophisticated control. The advantages are that they allow the robot to access very difficult terrain, a key feature in future applications such as nuclear power plant maintenance and earthquake victim search and rescue. A legged robot would be able to go wherever a human could.

A system on wheels can have some different configurations, representing tradeoffs between ease of control and other desirable properties such as speed, cheap actuators and agility. Figure 2.2 depicts three popular configurations. Model A has all wheels turnable and powered, which gives very good maneuverability at the expense of a more difficult mechanical design. B is called a “unicycle” and is very popular for simple experiment robots since it contains very little mechanics. Note that the two powered wheels are individually actuated, enabling the robot to turn on the spot. Drawbacks are that it cannot move sideways without turning first, and the passive castor wheels make it unsuited for rough terrain. Finally C has “car-like dynamics”, with powered back wheels and steerable front wheels. It can go fast and there are ready-made platforms ranging from cheap radio controlled cars to full-size trucks, but controlling it is more difficult. Moving sideways requires a whole sequence of back-and-forth manoeuvres, as in parallel parking.

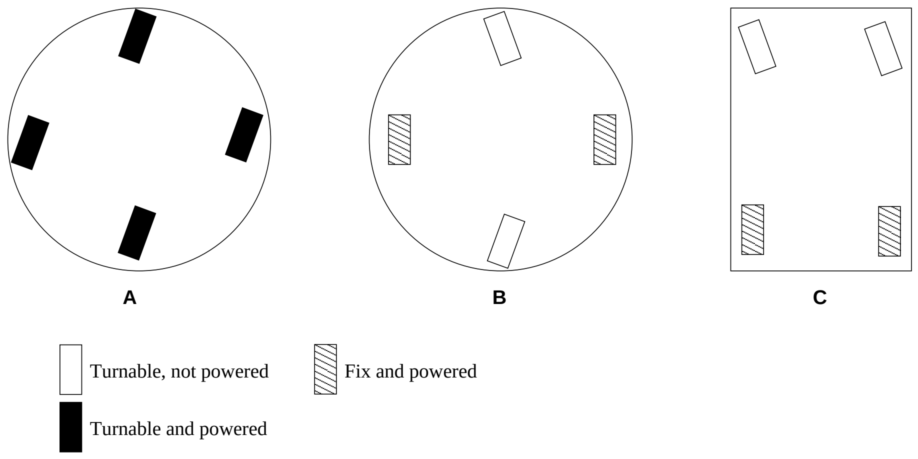


Figure 2.2: Three common wheel configurations for mobile robots.

2.2.2 Positioning In order to be useful and to find its goal, a robot must know its position. This can be in the sense of the robot knowing its coordinates, be it global coordinates or relative to e.g. a room, or in a topological sense, e.g. knowing that it is "in the corridor between the kitchen and the library". Tomatis et al., [7], have mixed the two methods in a robot that navigates to the correct room in an office building by first using a topological approach and then orients itself inside the room using local coordinates. They remark that this is very similar to the way a human would do it: when going to a room he orients himself coarsely by counting passed doors or corridors, while inside he switches to exact positioning to place the coffee mug in the coffee machine.

The basic method for positioning is usually dead reckoning, by simply counting the number of revolutions of the wheels or tracks, or in case of legs, by counting steps and using more complex geometry. It has the advantage of being simple, but since it involves integration it inevitably suffers from accumulating errors. Slippage, uneven surfaces and sensor errors will eventually make the dead reckoning position estimate drift away from the true position. The same is true for more advanced inertial navigation systems containing gyros, accelerometers and/or magnetic flux meters used as compasses. They can be more or less accurate but always have some finite drift, giving rise to a growing error.

To correct the dead reckoning one needs to use a parallel system where the error is bounded, see below. The standard approach is then to fuse the data from both systems by using a Kalman filter, [8]. This can give a better position estimate and also an estimate of the drift that is used to improve the performance of the inertial navigation system.

One such system that can be used in parallel is satellite navigation, most popularly GPS. It works only in outdoor applications where there is a free line of sight to a large portion of the sky, thus there are problems in dense cities and no reliable estimate at all indoors. As an example of what accuracies can be obtained we take the Lassen LP GPS receiver from Trimble, a small low-power GPS receiver suited for robotics applications. It has a stated horizontal uncertainty of less than six metres for 50% of its measurements and less than nine metres for 90% of the measurements, [9]. There are several more advanced approaches, such as receiving both carrier frequencies of the GPS system and use the extra information to correct for atmospheric effects or tracking the phase of the signal, which can give accuracies in the order of centimetres, [10]. Another commonly used technique is differential GPS (DGPS) with a stationary receiver at a surveyed position, producing corrections that can be used in real-time or for post-processing to significantly improve the accuracy of other receivers in the proximity (tens of kilometres from the stationary receiver).

In a military context the GPS system has some important weaknesses. First of all it is controlled by the United States, which makes it possible for the US military to degrade the accuracy of the signal (as was done with the publicly available signal until 2000). If civilian GPS receivers are used against US troops in a conflict, they could decide to degrade the signal again or even make it unavailable, which would have an impact also on countries that do not participate in the conflict. Secondly, the signal is very weak once it reaches Earth, so even very small jammers could have a substantial impact on the performance of GPS receivers in a large area. These problems make it necessary not to depend entirely on GPS for critical positioning of military systems.

A consequence of the design of the GPS system is that a receiver that has localized itself also has access to a very accurate time estimate. In the Lassen LP the errors are in the order of less than a microsecond [9]. If one has the possibility to use specifically placed beacons or other transmitters with known locations, they can be used in much the same way as the satellites of the GPS system. The beacons can transmit light, sound or radio waves and the method typically requires some signal processing to filter out echoes and surrounding noise before calculating a position estimate.

Many research robots also have sonar-type sensors, emitting a signal and then listening for echoes that indicate objects. They can be fairly unsophisticated ultrasound sonars that have the advantage of also providing good warning for collisions, or more advanced laser scanners that can produce a high-resolution 3D image of the immediate surroundings. A normal radar would fit into this category too, but its bulkiness and coarser resolution make it more interesting in the case of UAVs. By looking for characteristic landmarks such as walls, corners or trees a sonar-type sensor can contribute to the positioning. A problem with ultrasonic sonars is that very smooth surfaces tend to deflect the waves much like a mirror deflects light, without giving the diffuse echo that can be detected by the sonar.

In recent years there has been rapid development in the field of computer vision, strongly affected by the abundance of cheap digital cameras and computing power. Fitting a robot with a camera can aid the positioning, although even the most sophisticated systems today depend on highly controlled environments such as clear roadside markings and colour-coded objects or using the image to avoid movement or clearly distinguishable obstacles only. Human image recognition contains several layers of sophisticated image processing, intuition and experience that are far from what can be artificially achieved today.

A related research area is that of simultaneous localization and mapping, SLAM. Here the robot has no a priori knowledge of the environment, but is given the task of building its own map and positioning itself in it as it goes along.

2.2.3 Control and processing The robot needs a "brain", making sense of all the sensor data, applying a control algorithm to it and finally producing output to the actuators. The rapid development in computer technology has made it possible to fit even small robots with considerable processing capacity, without using too much of the precious electric power. The programs running in a robot are often divided hierarchically, where the lowest level consists of driving the actuators and filtering sensor data. Higher-level programs make strategical decisions on paths, goals and priorities. An important issue when designing software for robots is that of safety. There must be low-level safety routines that can quickly detect potentially dangerous conditions without the need for extensive calculations. Furthermore the system needs to be stable to avoid the risk of hang-ups that could disable the safety routines.

2.2.4 Communication Adding a means of communication to a robot can enhance its performance considerably. It can relay sensor data to an operator, reduce the need for on-board computation by communicating with a central controller and coordinate actions within a group. As with processing, technology makes rapid progress giving cheaper, smaller and less energy consuming circuits. Communication is most commonly done through radio links, that can range from transponders or Bluetooth circuits with ranges of a few metres to the directional antennas and high-power transceivers used to communicate with robots in space. Networks can have a fix configuration or be ad hoc, where the topology is dynamic. Depending on what nodes are within range at a given moment, messages can then be relayed different paths to increase the overall range of the system, at the expense of more demanding network protocols. In a military context, the problem of jamming requires that the system does not fail without communications or has countermeasures, for example directional antennas or frequency-hopping radios. An enemy can also be expected to use deception, i.e. sending false messages, and to use triangulation to localize the transmitter.

2.3 Autonomous navigation

The objective of autonomous navigation is to be able to tell a robot *where* to go, but let it figure out on its own *how* to get there. There are two dominating approaches to the subject; planning and reaction. Planning depends heavily on modeling the world and, in a classical control or optimization manner, finding the best solution by applying mathematical methods to the model. The resulting output is then used in the real world and the difference between the predicted and actual outcome is fed back into the algorithm to correct the world estimate. There exist many proven methods for analyzing planning algorithms, but the drawbacks are the high computational demands as the world model gets more complicated.

The other approach is based on reacting directly to sensor input from the surrounding world. It borders on the domains of artificial intelligence, AI, and is motivated by the study of animal and human behaviour. Arkin, [11], uses the terms attention, behaviours and intentions. Attention controls the sensors so that they focus on what is important at the moment. This can for example mean a camera focusing on a suspected obstacle or microphones tuning to the sound of walking feet when the robot expects to meet someone. The point of focusing attention is to reduce the amount of sensor data that has to be processed. Then the robot has a whole set of behaviours, such as avoiding obstacles, going towards the goal, slowing down in crowds and so on. The behaviours are fed by all relevant sensors and output a recommended actuator response. Finally the intentions of the robot has to accumulate all the recommendations into one action. If the intention is to reach the goal at any cost, the robot will listen to the "towards the goal" behaviour unless maybe the "obstacle avoidance" gives a very strong impulse to stop, because of a massive brick wall ahead. On the other hand a robot with the intention of guiding visitors through a museum should give more weight to the "slow down in crowds" behaviour than arriving at the goal.

Figure 2.3 schematically depicts these building blocks of a reactive control system. The advocates of reactive control underline that "the world itself is the best model" and must be superior to theoretical approximations, plus the lesser need for computation that gives cheaper hardware and faster reactions. Drawbacks are that the methods of reactive control are difficult to analyze and this gives problems in guaranteeing properties of the algorithms and makes tuning a trial-and-error process.

2.4 Multi-robot cooperation and formations

Using several robots that cooperate in a group has several advantages over solving the same task with one single robot, [12]:

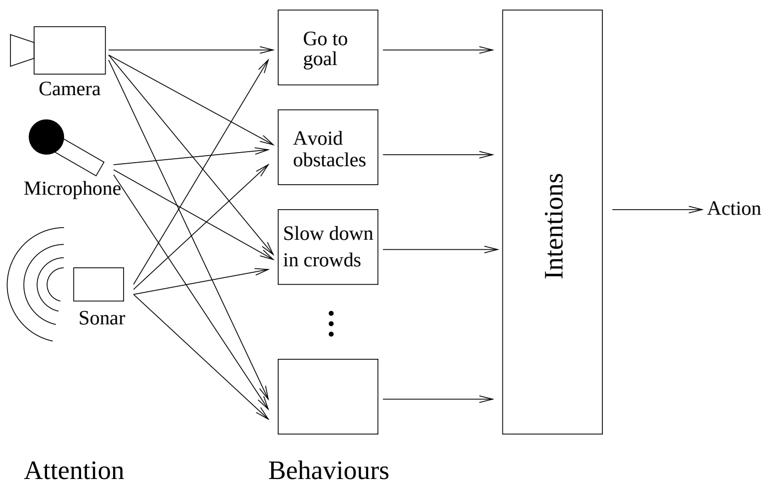


Figure 2.3: The structure of a reactive control system, with sensors governed by attention, different behaviours that recommend actions and finally the intentions that decide what recommendation(s) to follow.

1. Redundancy: If one robot is destroyed or gets stuck, the group as a whole can still function, albeit with reduced performance.
2. Simplicity: Instead of making one robot loaded with advanced sensors, some tasks can be performed by several simpler robots. They can be serially manufactured to lower the overall cost.
3. Flexibility: Groups of simple robots can easily be rearranged or partitioned in subgroups to adapt to new environments or functions.
4. Distributed sensing: Some tasks such as area surveillance or sweeping are better solved by groups of sensors than a single one. Robots in prescribed formations can also do triangulation and satellites can do so called deep space interferometry, which requires sensor spacings far wider than what can be achieved on a single satellite.

When using groups of robots, it is often beneficial to encode some form of group cohesion in their navigation algorithms. The groups are then typically categorized as *flocks* or *formations*. In a flock, the members stay together but can have arbitrary positions inside the group. This is opposed to the term formation, which is used to describe a rigid configuration of the participating robots. There are some reasons why moving in flocks or formations can be of interest. First, it facilitates communication within the group and second, it makes it easier to supervise for an operator, much like a kindergarten teacher tries to keep the children together during an excursion. Second, if the group has a sensing task, it may require that they move in a prescribed formation to coordinate their sensors and get full coverage. Finally, if something happens to one of the robots, it will be easier to help it if the others are nearby. Of course there are also situations in a military context when it can be dangerous to move in flocks, since it may increase the risk of detection. And perhaps the best strategy for crossing a minefield is not to move in a flock but on a line, following the path of the first robot.

3. Flocking with obstacle avoidance

In this section we describe the developed algorithm for moving a group of agents to a target area while maintaining group cohesion and not colliding with obstacles or other agents. The words robot and agent will be used interchangeably in the following text. We will also use the term formation in a looser sense than described above, to describe a group where the robots have assumed a prescribed configuration, but are free to break it if needed to ensure goal convergence.

3.1 Earlier approaches

There are several approaches to moving in formations and avoiding obstacles described in the literature, while the combination of the two is less studied. Many existing schemes use potential-based methods. In these, every neighbor is assigned a potential with a minimum at the desired inter-agent distance and the agent is controlled according to the negative gradient of the sum of all potentials. It strives to move to a position where the potential has a minimum. Obstacles are given potentials too, that are added to the sum. This approach can lead to an agent being pushed into an obstacle by a group of neighbors, whose common influence may overcome the repulsion from the obstacle. A way to overcome this is by giving the obstacle a repulsive influence that approaches infinity as the distance to it goes to zero. On the other hand, this risks producing the effect of an agent being pushed into a neighbor due to the much stronger influence from an obstacle.

Many of the schemes described above also have the disadvantage of representing obstacles as single points. This works well for circular obstacles, but in case of more general shapes the approach is less successful. The obstacle can then be represented by a virtual neighbor placed at the point on the obstacle boundary closest to the agent. Non convex boundaries can give abrupt changes of the position of the virtual neighbor, making the agent trajectory irregular and inefficient.

3.2 Combining coverage control and navigation functions

The idea of our algorithm is based on a proposition by Cortes et al., [3], combined with the concept of navigation functions, [2]. We divide an area $Q \subseteq \mathbb{R}^2$ into so called Voronoi regions around every agent. The Voronoi regions are then intersected with the obstacle-free space before the centre of mass is calculated for every region, weighted with the navigation function. The navigation function is a scalar function that roughly maps every free point in Q to the length of the shortest obstacle-free path to the goal. Finally the agents move to the respective centres of mass of each Voronoi region before the procedure is repeated. If a Voronoi region is unbounded in any direction, we add virtual mirror neighbors, whose positions are those of all neighbors, but mirrored in the position of the agent and at a fixed distance.

The new algorithm has several advantages over existing schemes. First it guarantees collision avoidance, both with other agents and obstacles. Second, it often gives goal convergence in the sense that all agents will be gathered around the goal after a finite time. Finally, simulation results as well as theoretical analyses of simplified settings indicate that a flock of agents moving in an open field will exhibit group cohesion in terms of forming a hexagonal lattice formation.

In the next sections we describe the coverage control algorithm and navigation functions in more detail.

3.3 Coverage control

Cortes et al., [3], study the problem of positioning a number of sensors

$$P = \{p_1, p_2 \dots p_N\},\ p_i \in Q,$$

in order to observe an area, Q, which is assumed to be a convex subset of $\mathbb{R}^2$. Let the probability density for a detectable event be $\phi(q)$ and the performance of the sensors decrease with the distance from the sensor to the event. A cost function can then be formulated as

$$H_V(P) \quad = \quad \int_Q \min_i ||q - p_i||^2 \ d\phi(q).$$

This can be interpreted as the expected squared distance from an event to the closest sensor, i.e. $E(\min_i ||q - p_i||^2)$.

It turns out that the gradient of this function with respect to the sensor positions p_i is

$$\frac{\partial H_V(P)}{\partial p_i} = 2M_{V_i}(p_i - C_{V_i}),$$

where

$$\begin{aligned} M_V &= \int_V \phi(q)\ dq, \\ C_V &= \frac{1}{M_V}\int_V q\phi(q)\ dq. \end{aligned} \tag{3.1}$$

The quantities M_V and C_V are the generalized mass and centre of mass (centroid), respectively, and V_i, $i = 1, \dots, N$, are the Voronoi regions associated with P, as defined next.

Definition 3.3.1 (Voronoi partitions) *The collection $V(P) = \{V_1, V_2, \dots, V_N\}$ is a Voronoi partition corresponding to the points $P = \{p_1, p_2, \dots, p_N\}$ if*

$$V_i = \{q : ||q - p_i|| \leq ||q - p_j||,\ \forall j \neq i\},$$

where $||.||$ denotes the Euclidean norm. The sets V_i are denoted Voronoi regions.

Cortes et al. propose a solution to the coverage control problem on feedback form using a gradient descent control law:

$$\dot{p}_i = -k_{prop}(p_i - C_{V_i}).$$

Via a LaSalle argument it is then shown, among other things, that p_i converges asymptotically to the set of critical points of H_V, i.e. points where $\frac{\partial H_V(P)}{\partial p_i} = 0$.

3.4 Navigation functions

The concept of navigation functions was introduced by Rimon and Koditschek, [13]. The navigation function, $NF : Q \subseteq \mathbb{R}^2 \to \mathbb{R}$, is an artificial potential that maps all obstacle-free points in Q to a scalar value. It has only one minimum, local as well as global, located at the goal.

The original navigation function is, however, not very well suited for computation. Ögren et al., [2], have suggested a modified version that is piecewise differentiable and has local maxima of measure zero. It is constructed as follows:

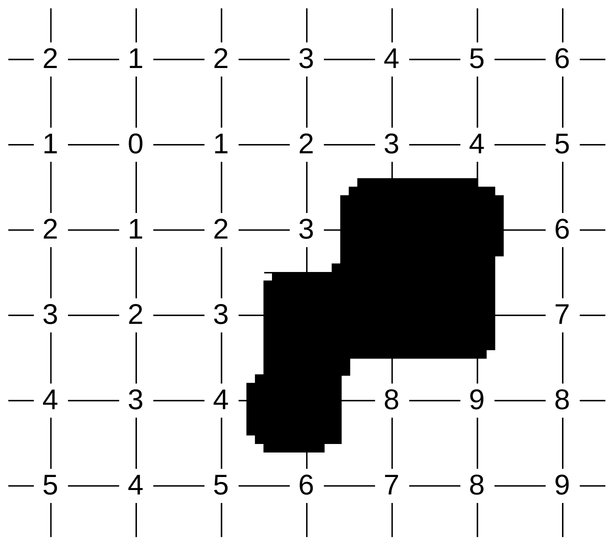


Figure 3.1: An example of a navigation function in a subset of Q. The grid spacing is assumed to be 1 and the goal point by definition has the value 0. Note how the two points of value 9 form a ridge of measure zero.

1. Make a grid covering Q and remove all vertexes and edges that are on the inside of obstacles.
2. Designate one of the vertexes as the goal.
3. For each remaining vertex, calculate the shortest distance to the goal.
4. For points inside grids, NF is obtained by linear interpolation of the values at the three adjacent vertices. The grid is divided along a diagonal, chosen so that the resulting two triangles have the same slope or meet in a maximum, not a minimum. Finally the value is obtained by interpolation on the vertices of the triangle that covers the point.

Figure 3.1 shows an example of navigation function values in a subset of Q around the goal.

3.5 Proposed algorithm

As mentioned in the problem formulation, we consider a group of kinematic robots p_i that have the following dynamics:

$$\begin{aligned} p_i(k+1) &= p_i(k) + u_i(k) \\ ||u_i|| &\leq u_{max} \end{aligned}$$

To calculate the control of vehicle $j \in \{1 \dots N\}$ we let

$$P_j = \{p_i : ||p_i - p_j|| \leq R_{max}, i \neq j\}$$

define the set of neighbors within sensing radius. We also define S_j as the sensor coverage area of agent j, i.e $S_j = \{q : ||q - p_j|| \leq R_{max}\}$, and we let L_j be all parts of Q that are within the line of sight from agent j. The following algorithm is carried out by vehicle j at each iteration:

0. **Initialization:** Set d to the desired inter-vehicle distance. The design parameter k_ϕ is discussed below, but can normally be set to $k_\phi = 1$. Finally choose a small scalar $\epsilon > 0$, which will be the minimum step length.

1. **Mirror neighbors:** If p_j is not inside the convex hull of its sensed neighbors, P_j, i.e. the Voronoi region is unbounded in some direction, then for each $p_i \in P_j$ create a new momentary *mirror neighbor* $\hat{p}_i$ as

$$\hat{p}_i = p_j - d\frac{p_i - p_j}{||p_i - p_j||}.$$

   Let $\hat{P}_j = P_j \cup \{\hat{p}_i\}$.

2. **Voronoi region:** Calculate V_j from the position of the neighboring vehicles and mirror neighbors, $\hat{P}_j$.

3. **Centroid:** Calculate C_V according to (3.1), with $\phi(q) = e^{-k_\phi \cdot NF(q)}$ and $V = V_j \cap L_j \cap S_j$.

4. **Choice of target:** Calculate $p_j{}'$ from the following optimization problem:

$$\begin{aligned}
\min_{p_j{}'} \quad & ||p_j{}' - C_V||^2, && (3.2)\\
\text{s.t.} \quad & NF(p_j{}') < NF(p_j) - \epsilon, && (3.3)\\
& p_j{}' \in V_j, && (3.4)\\
& p_j{}' \in L_j, && (3.5)\\
& ||p_j{}' - p_j|| \le R_{max}/2. && (3.6)
\end{aligned}$$

   If there is no feasible solution, set $p_j{}' = p_j$.

5. **Execution:** Apply the control $u_j = p_j{}' - p_j$ and repeat from step 1.

One could make a few remarks on the algorithm:

Remark 3.5.1 (Mirror neighbors) *The purpose of the mirror neighbors is to achieve flocking in open areas. Without them the vehicles at the boundary of a flock will tend to "float away". This effect is reversed by introducing the mirror neighbors.*

Remark 3.5.2 (Decentralization) *The algorithm is completely decentralized as it only requires that the robot has knowledge about its surroundings, up to the sensor radius. There are two alternative ways for the robots to know the positions of their neighbors: Either they have a means of communicating their position to all neighbors, or every robot is equipped with a sensor that can measure the positions of all nearby robots. Avoiding ambiguities in the Voronoi partitioning requires that all robots make the partitioning at the same time instances (which can be solved by accurate on-board clocks or centralized time-keeping as in the GPS system) or that the update interval is sufficiently short in relation to the speed of the robots.*

3.6 Properties of the algorithm

In this section we give some theoretical results on the properties of the proposed algorithm. We consider the issues of safety, goal convergence and formation stability. We also suggest some modifications that have not yet been integrated into the algorithm. As stated above, the highest priority has been given to safety so that neither the robots nor stationary objects in the surroundings will be damaged. This is guaranteed by the following proposition.

Proposition 3.6.1 (Safety) *There will be no collisions with obstacles or other vehicles.*

Proof. The Voronoi regions will be correct at least to the radius of $R_{max}/2$, since the sensor can see to the radius of R_{max}. Thus the inner part of the Voronoi region, to which the step is restricted by (3.4) and (3.6), will never be entered by another agent. The step will furthermore never collide with an obstacle due to (3.5). □

This result can also be expanded to a robot with non-zero dimensions. The obstacles then need to be grown by one robot radius and all Voronoi region boundaries have to be moved inwards by the same distance. Even if two robots step to the very boundaries of their Voronoi regions, there will still be a margin that prevents a collision.

Under very special circumstances, described in section 3.6.1, the robots can find themselves in situations where they have not reached the goal but still cannot find feasible points to go to. This seems to lead to a stable, but not asymptotically stable, dead-lock that stops some robots from reaching the goal. Such a complete locking has never occurred in simulations, instead the algorithm has produced reliable goal convergence in all tested environments. The following result describes this:

Proposition 3.6.2 (Goal convergence) *The agents will move towards the goal until they reach a configuration such that the optimization problem (3.2)–(3.6) has no feasible solution for any agent.*

Proof. This follows from condition (3.3) and the fact that $NF(x) \geq 0$, with equality only at the goal. □

We would ideally want to prove that the robots will stay together in a collected formation when moving in open fields, where NF has a constant gradient. Simulations have showed that the robots tend to form a hexagonal lattice, which seems logical as it provides a configuration where all inter-robot distances are d, as encoded in the mirror neighbor mechanism. In the following analysis, when referring to the hexagonal lattice formation we shall mean the formation depicted in Figure 3.2. There are still some open research problems, but judging from the special cases where we have been able to prove formation stability and the appealing behaviour in the simulations, we have confidence in the mechanism of mirror neighbors.

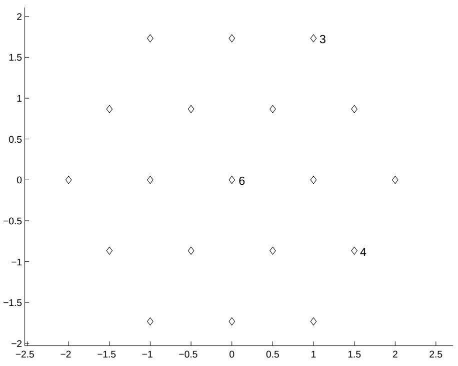


Figure 3.2: We denote the above formation the hexagonal lattice formation. In the figure, some agents are numbered according to how many neighbors they have at distance d.

The following lemma shows that the hexagonal lattice formation is a stationary configuration, but does not guarantee stability.

Lemma 3.6.3 (Hexagonal Lattice) *Assume constraint (3.3) is not active. If the vehicles are in a open area with constant NF gradient, then the hexagonal lattice formation with inter vehicle distances d is a stationary configuration for all vehicles.*

Proof. If all the vehicles are positioned in the hexagonal lattice formation, then so will all the mirror neighbors. Thus all vehicles will have identical perfect hexagons for Voronoi regions. The fact that the gradient of NF is constant in the region makes the NF values of two different hexagonal regions differ by a constant term only, $NF(q_1) = NF(q_2) + M$, where q_1 and q_2 are the same positions in different hexagons. This makes $\phi(NF(q_1)) = \phi(NF(q_2)+M) = e^{-k_\phi(NF(q_2)+M)} = \phi(NF(q_1))e^{-k_\phi M}$, i.e. ϕ differs by a constant factor. Finally, since a constant factor does not influence C_V the motion of all vehicles will be identical and the configuration is stationary. □

In one dimension it can be proved that two robots will converge to the inter-vehicle distance d:

Lemma 3.6.4 (One-dimensional two vehicle stability) *Let two vehicles move in one dimensional space towards the goal. If they are within sensing range of each other, all other vehicles or obstacles are beyond the distance R_{max} and constraint (3.3) is inactive, then the two vehicles will converge exponentially to the preferred distance d.*

Proof. In one dimension for $x \leq 0$ we get $NF = -x$ and thus $\phi(x) = e^{k_\phi x}$. For symmetry reasons, both Voronoi regions will have width c. A region having its leftmost boundary at $x = b$ has the following mass and centroid:

$$\begin{aligned} M_V &= \int_b^{b+c} e^{k_\phi x}\, dx = \frac{e^{k_\phi b}}{k_\phi}(e^{k_\phi c} - 1) \\ C_{V_x} &= \frac{1}{M_V}\int_b^{b+c} x\, e^{k_\phi x}\, dx \\ &= b + \frac{1}{k_\phi\ (e^{k_\phi c} - 1)}[e^{k_\phi c}(k_\phi c - 1) + 1] \end{aligned}$$

Apparently the relative position of the centroid in the Voronoi region does not depend on b. We are thus free to place the origin of our new coordinate system inbetween the two vehicles and let the equal distance to each of them be $a/2$. After one iteration the new inter-agent distance a_{new} becomes

$$a_{new} = C_{V_2 x} - C_{V_1 x} = 0 - (-\frac{d+a}{2}) = \frac{d+a}{2}.$$

Assuming that $a = d + \delta$ we see that

$$a_{new} = \frac{d+a}{2} = \frac{d+d+\delta}{2} = d + \frac{\delta}{2}.$$

Thus the deviation from the preferred distance is halved at every iteration step and the convergence is exponential. □

The fact that the two lemmas above show results that are independent of k_ϕ indicates that we can get clues to the general behaviour of the formation by using the simplification $k_\phi = 0$.

Lemma 3.6.5 (Three-vehicle flocking) *Assume there are three vehicles within sensing range of each other and all other vehicles and obstacles are beyond the distance R_{max}. If these three vehicles are controlled according to (3.2) with $k_\phi = 0$, assuming the constraint (3.3) is inactive, then the vehicles will converge to an equilateral triangular formation with the side d.*

Proof. Consider vehicle p, and note that its Voronoi region will be a parallelogram due to the placement of the mirror neighbors. This set will furthermore satisfy

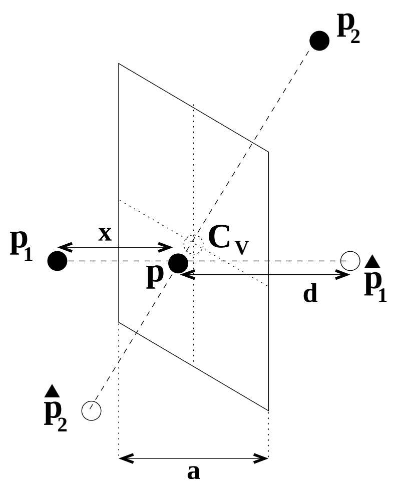


Figure 3.3: The setting for Lemma 3.6.5. The solid lines denote the Voronoi region. Using $k_\phi = 0$, C_V is at the geometric centre of the parallelogram.

constraint (3.5) and assume initially that it also satisfies constraint (3.6). In a parallelogram, the centroid is at equal distances from any pair of two opposite edges. Without loss of generality we consider one vehicle with distance x to one of its two neighbors and a being the separation of the two sides, as depicted in Figure 3.3.

A motion from p_j to the centroid C_V now yields the following change in the x direction.

$$\begin{aligned}\Delta x &= \frac{a}{2} - \frac{x}{2},\\ &= \frac{1}{2}(\frac{d}{2} + \frac{x}{2} - x),\\ &= \frac{1}{4}(d - x).\end{aligned}$$

We see that $x \to d$ and thus all three sides of the triangle tend towards length d. This qualitative behavior can also be seen to hold in cases where constraint (3.6) is active. This is due to the fact that the circle is centered at p_j. □

In a more general setting, with an arbitrary number of robots, we can show that a robot far away from the formation will move towards it:

Lemma 3.6.6 (General flock cohesion) *If the vehicles are controlled only according to (3.2), with $k_\phi = 0$ and disregarding (3.3) and (3.6), then a vehicle not contained in the convex hull of its neighbors, P_j, will move such that at least one neighbor will be closer than, or at a distance d, and at least one will be at d or farther away.*

Proof. Let the position of a vehicle, not contained in the convex hull of its neighbors, be p_j and the positions of all its neighbors, P_j, be p_i. Assume $||p_j - p_i|| > d$, for all i. Since p_j is not in the interior of the convex hull of its neighbors, we can draw a line through p_j, separating the neighbors from their mirror images. Let L be the one such line that maximizes the distance to the closest mirror neighbor. We will now argue that C_V is on the side of L containing the real neighbors.

If the neighbors were at a distance d from p_j, then L would split V_j into two parts differing only by a 180-degree rotation, having C_V somewhere on the line L. Furthermore, V_j would have two corners where L intersects its boundary.

Moving the neighbors back to their true positions will therefore only grow the part of V_j on the neighbors' side, thus moving C_V towards this side as well.

Therefore, p_j will move towards the neighbors, until the assumption $||p_j - p_i|| > d$ no longer holds. The opposite argument, with $||p_j - p_i|| < d$, is completely analogous. □

The main difficulty when analyzing the algorithm is that it is nonlinear and the influence of one neighbor generally depends on the positions of other neighbors as well. While being difficult to analyze, it is well suited for computer simulations, so we have simulated a setting with a group of robots in a hexagonal lattice formation. We then perturbed the position of one robot at a time, while keeping all others fix. The original configuration is illustrated in Figure 3.2.

Three different positions in the formation are indicated in Figure 3.2, numbered according to how many close neighbors they have. We have analyzed every such position in the lattice numerically by perturbing the agent while fixing all neighbors. The mirror neighbors are of course not fix, since their positions depend on the position of the agent being tested. For every relative perturbation Δp_j from the original position p_j we plotted the ratio

$$U = \frac{||C_{Vj} - p_j||}{||\Delta p_j||} \geq 0.$$

This gave a scalar field $U : \mathbb{R}^2 \to \mathbb{R}$. If, for small perturbations Δp_j, the scalar field is less than one, this means that the centroid will always be closer to the original position than the perturbed position. As the agent will go to its centroid, this means that it will eventually return to its original position after a perturbation. This does not constitute a formal proof, since we have only perturbed one agent and stability requires that all agents can be perturbed simultaneously and since we have only sampled U at a finite number of points. However, as we will see below this method has produced some interesting results.

It was found that with the current strategy for mirror neighbors, position 4 in the lattice is not asymptotically stable. When the agent is perturbed so that it is just inside the convex hull of its neighbors, it has no mirror neighbors. This means that its Voronoi region will extend far outside the formation, until it is truncated at R_{max}. This is illustrated in Figure 3.4, where the truncation at R_{max} is not taken into account. The agent will break away from the formation when it moves towards the distant centroid. In the next iteration it will find itself outside the convex hull of its neighbors, create mirror neighbors and move towards the formation again. Despite this effect, in Figure 4.3 a group moving over an open field displays an almost perfect formation. We think that this could be because the sensor radius $R_{max} = 3$ is so small in comparison to the preferred distance $d = 2$ that the protruding Voronoi region does not extend very far. The effect is more evident in Fenix, where the sensor radius has been chosen to twice the preferred inter-robot distance. On open fields, vehicles at the edge of the formation exhibit a zig-zagging motion that may be stable in a limit cycle sense, but not asymptotically stable.

We decided to test a modified strategy for the mirror neighbors: An agent always mirrors all neighbors within the distance $1.5\,d$. If it is still not inside the convex hull of its neighbors and the mirror neighbors, it mirrors all other neighbors within the sensing radius. Using this strategy, we tested all positions in the lattice, with promising results. We got $U < 1$ for all displacements $||\Delta p_j|| < 0.25\,d$, which indicates that all positions are asymptotically stable. We also found that

$$||\Delta p_j|| \to 0 \Rightarrow U \to 0.75.$$

It seems intuitive that U should approach the same value for all positions, since for very small displacements all agents experience similar surroundings of six symmetrically distributed neighbors and mirror neighbors. This modified mirror neighbor strategy has not yet been integrated into the algorithm and tested in simulations of a complete moving group.

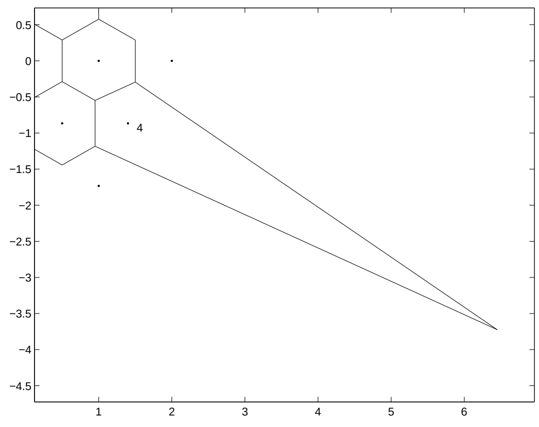


Figure 3.4: When the agent at position 4 is perturbed inwards, it has no mirror neighbors. Its Voronoi region then extends too far and its centroid is at a large distance from the formation. This makes the agent leave the formation.

The above lemmas and simulations lead us to believe that, when using the modified strategy for mirror neighbors, the robots will gather to form a formation when moving over an open field where the gradient of NF is constant. This is also indicated by the simulations described in the following sections.

3.6.1 Dead-locks Under very special conditions one or more robots can find themselves in situations where they have not reached the goal but still cannot find feasible points to go to. We will first study how this can happen to one single robot.

A single robot that gets very close to a corner before turning it, can find that there is no point within its line of sight that satisfies the condition that the navigation function has to decrease by at least ϵ in every step, (3.3). The situation is described further in Section (4.2.3) below. There are feasible points where NF decreases, but not by as much as prescribed by condition (3.3). The condition was originally included to guarantee an upper bound on the goal convergence time, but as this has proved insufficient the obvious suggestion would be to remove it. We have not fully analyzed this yet, but as described in section 4.2.3, we have already included a mechanism in the simulations that has the same effect. Another reason for requiring that the navigation function decreases monotonically is to get more appealing trajectories, where the robots never step backwards, and this is of course sacrificed if (3.3) is removed.

There have also been situations where several robots have blocked each other when rounding corners or entering passages. Two examples of such situations are depicted in Figure 3.5. This has never led to a permanent blocking in the simulations, as round-off errors and other numerical effects have eventually moved the agents enough to resolve the blocking. This indicates that the blockings are at least not asymptotically stable, as is confirmed for case A by the following analysis.

Example 3.6.7 (Dead-lock stability) *A blocking situation as depicted in Figure 3.5 A, where the Voronoi boundary L has a $-45°$ slope and $NF(p_1) = NF(p_2) < NF(q_c) + \epsilon - \delta$, where δ is a small arbitrary scalar, is not asymptotically stable.*

Proof. Note that all points satisfying (3.3) are to the right of the obstacle but below L, so they belong to the Voronoi region of agent p_2 but is out of its line of sight. As the level curves of NF all have $\pm 45°$ slope, there is no point within the line of sight from p_2 that satisfies condition (3.3) unless the agent moves above the line L. So for small movements of any of the agents, agent p_2 will remain stationary.

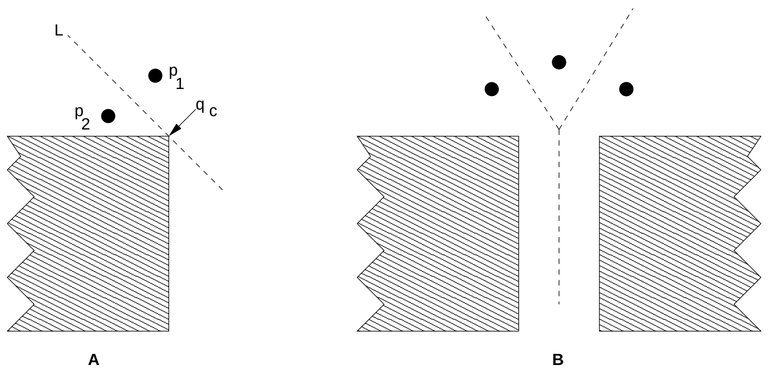


Figure 3.5: Two examples of situations when several robots are blocking each other. The dashed lines depict the boundaries of their Voronoi regions and the goal is located in the downwards direction.

To study how agent p_1 is affected by small movements of both agents, we will without loss of generality study small movements of L. It can be translated (as in Figure 3.6 A) or rotated (Figure 3.6 B), or a combination thereof. Assume that V_1 is the Voronoi region of agent p_1 and that $p^\star$ is the point in V_1 such that

$$NF(p^\star) \leq NF(q) \,\forall\, q \in V_1,$$

i.e. the best position in V_1.

In Figure 3.6 A we see that a small translation $\alpha < \delta$ of L, corresponding to a small movement of both agents, will lead to a decrease α of $NF(p^\star)$, which is not enough to fulfill constraint (3.3).

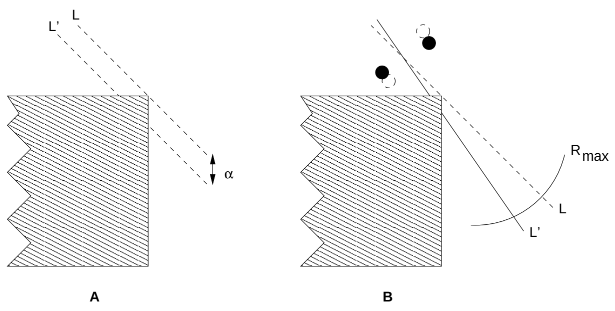


Figure 3.6: The two possible types of movements for the separator line L between two agents; translation (A) and rotation (B).

Even for very small clockwise rotations of L, as depicted in Figure 3.6 B, $NF(p^\star)$ will decrease with a large value. But for $p^\star$ to be a feasible point, it has to fulfill constraint (3.6), i.e. be within the distance $R_{max}/2$ from the agent. So for small displacements of the agents, L will not rotate enough to reveal points that simultaneously satisfy constraints (3.3) and (3.6). Consequently, even a combination of a sufficiently small rotation and a translation will not lead to agent p_1 finding a feasible target point, so agent p_1 will also remain stationary for small movements of any of the agents. Thus, for a sufficiently small displacement, none of the agents will move. That means that the equilibrium is stable but not asymptotically stable. □

Even though the blockings do not seem to be asymptotically stable, they still delay the goal convergence considerably. This is most clearly seen in Fenix where many robots are to pass a narrow bridge, as described in section 5.6. We have therefore devised a modification of the algorithm that will allow robots to detect a blocking and give way to others. The symmetry can easily be broken using individual ID numbers, assigned to every robot.

The suggested modification can be described as:

- Remove constraint (3.3).
- If a robot takes a step in a direction where NF increases **and** there is at least one neighbor within its sensing radius that has a lower ID, it signals that it will stand still during a predefined period of time (typically one iteration).
- During this time, all surrounding robots can intrude on its space by calculating asymmetric Voronoi regions. Instead of making boundary lines half-way between the two robots, they can for example use 95% of the distance and leave 5% to the robot giving way.

The reason to take a step *away* from the goal is typically that there is a congestion ahead, so the above criteria should be able to detect that. Simulations have showed promising results, where a group of robots typically approach a narrow passage, take one step back and then stand still with the exception of the one with the lowest ID. Then all others start moving again, back away and stand still to let the robot with the next lowest ID through, and so on. We have made preliminary simulations of this algorithm under the same conditions as described in Figure 4.1 and it decreased the number of iterations required for goal convergence approximately from 500 to 200. Testing this more thoroughly and analyzing it in theory is an interesting direction of future work.

4. Simulation in Matlab

In this chapter we present some results from computer simulations, performed in Matlab. A group of 20 robots is simulated in two different settings. First we study an environment with irregular and non convex obstacles to underline the advantage of not having to make geometric assumptions about obstacle shape. We then test the same group in an open field to explore the formation properties.

4.1 Results

Figure 4.1 shows four snapshots of a group of 20 robots, moving from the starting point in the lower right corner of the area to the goal in the upper right part, marked by an x. The vehicle positions are depicted by stars for the first and third snapshot and by dots for the second and fourth snapshot. The robots first perform a split/rejoin manoeuvre, then squeeze the formation when passing the corridor and finally gather around the goal. Due to the constraint $NF(k+1) < NF(k) - \epsilon$, the agents are distributed only in the third quadrant around the goal. In Figure 4.1 the preferred inter-agent distance is $d = 1$ and the maximum sensing radius is $R_{max} = 3$. The parameter k_ϕ is chosen to 1.

Figure 4.2 shows the same setting, but with $k_\phi = 0$. This removes the influence of the NF on C_V and the only thing driving the vehicles towards the goal is constraint (3.3). This causes the vehicles to move much slower, but as can be seen in the figure, the spacing during the corridor traversal is somewhat wider. The snapshots are not taken at the same time instances in the different Matlab plots.

To explore the flocking behavior in detail two simulations are run in an open area with no obstacles. Figure 4.3 shows an almost perfect hexagonal lattice resulting from the settings $k_\phi = 1$, $d = 2$ and $R_{max} = 3$. This was indicated, but not guaranteed, by the preliminary results in the theoretical analysis. The distance between agents agrees well with the preferred distance used.

Surprisingly enough, setting $k_\phi = 0$ gives less group cohesion, as seen in Figure 4.4. The transversal inter-agent distances are fairly correct, but the flock seems to have drifted apart in the direction of motion. The explanation is to be found in constraint (3.3). If k_ϕ is high enough, (3.3) will be satisfied by C_V itself, and the resulting behavior will be as if the constraint were not there. If, however k_ϕ is low enough, as in the $k_\phi = 0$ case, the vehicles would stand still in perfect formation if it were not for constraint (3.3). The constraint forces the vehicle to move in a direction other than C_V, thus obstructing the formation maintenance.

4.2 Implementational aspects

This sections describes some issues that we faced when implementing the algorithm.

4.2.1 Numerical integration The formula for the centroid (3.1) requires calculating a surface integral over the intersected Voronoi region $\hat{V} = V_j \cap L_j \cap S_j$. To do this we made a grid

$$G = \{q : q = p_j + 0.25\,(n\,e_x + m\,e_y),\ n, m \in \mathbb{Z}\} \tag{4.1}$$

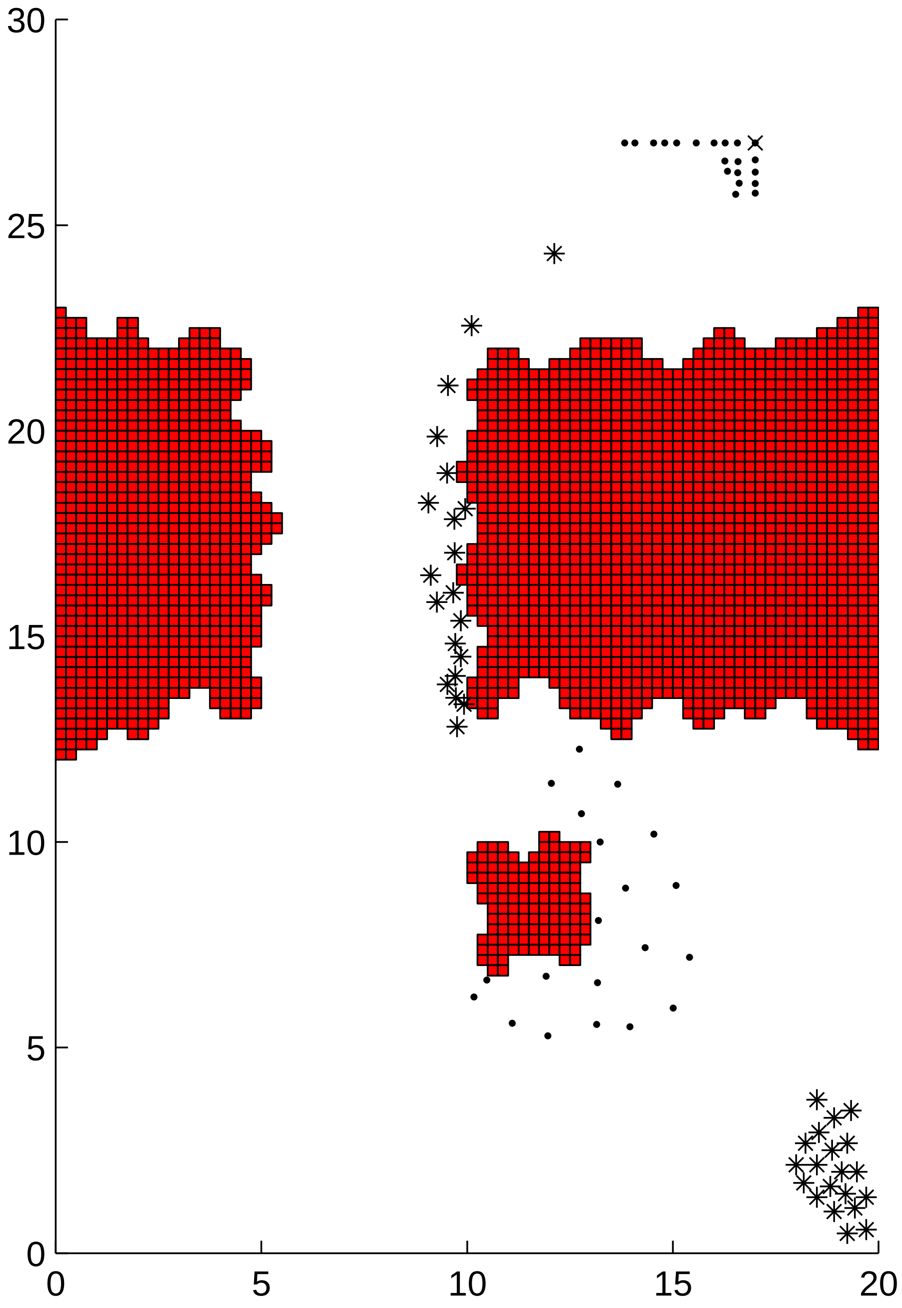


Figure 4.1: A group of 20 agents moving around irregular obstacles. The agents are depicted by stars for the first and third snapshot and dots for the second and fourth. The parameters are $R_{max} = 3$, $d = 1$ and $k_\phi = 1$. This gave the fastest goal convergence, at the expense of flock spacing in the corridor between the obstacles.

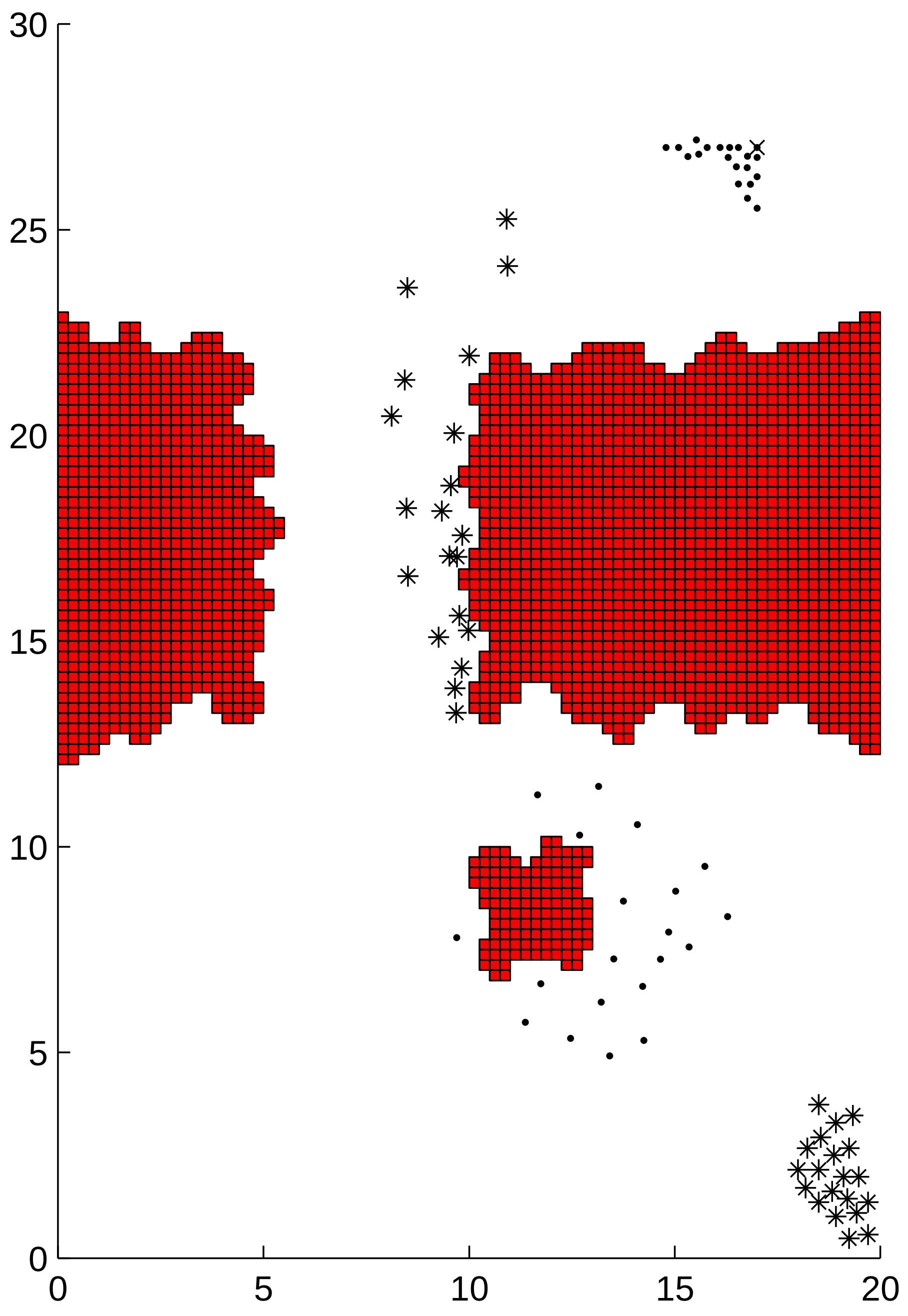


Figure 4.2: A group of 20 agents moving around irregular obstacles. The agents are depicted by stars for the first and third snapshot and dots for the second and fourth. The parameters are $R_{max} = 3$, $d = 1$ and $k_\phi = 0$. The group takes longer to reach the goal, but the inter-agent spacing is more appealing.

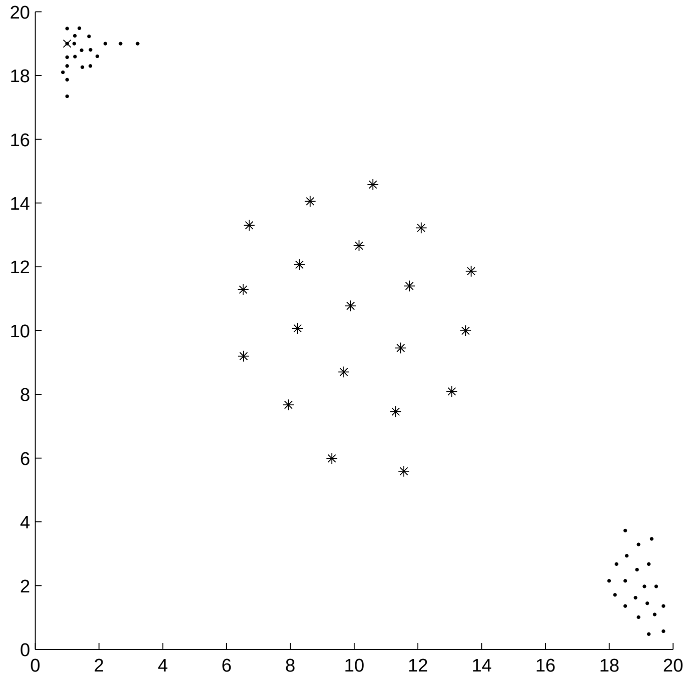


Figure 4.3: A group of 20 agents moving in a free field. The agents are depicted by dots for the first and third snapshots, and by stars for the second. The parameters are $R_{max} = 3$, $d = 2$ and $k_\phi = 1$. The group assumes an almost perfect hexagonal lattice formation.

i.e. a grid with spacing 0.25 that contains p_j. The motivation for choosing grid spacing 0.25 was that is also the spacing of the grid used for calculating the navigation function. All points in the grid were then tested against conditions (3.4)–(3.6) and the mass and centroid were calculated as

$$M_V = \sum_{q \in \hat{V} \cap G} \phi(q) \tag{4.2}$$

$$C_{Vx} = \frac{1}{M_V} \sum_{q \in \hat{V} \cap G} x_q \phi(q) \tag{4.3}$$

$$C_{Vy} = \frac{1}{M_V} \sum_{q \in \hat{V} \cap G} y_q \phi(q). \tag{4.4}$$

4.2.2 Line of sight Condition (3.5) as well as finding the intersected Voronoi region for integration as described above require a method to determine if a given point is within line of sight from the agent. Since the obstacles are given on bitmap form there is no vector representation of their boundaries, so the only information that can be extracted from the map is whether a specified point is occupied by an obstacle or not. The method also has to be reasonably fast, as the test is to be performed on all points included in the integration.

To satisfy these requirements we chose to use an approach from computer graphics, [14]. The idea is to study only the first octant, i.e. all points in the angle interval 0–45° from the agent. For every point we make a binary matrix of which other points are obscured if the generator point is occupied by an obstacle. The matrix corresponding to the generator point (1, 0.75) is illustrated in Figure 4.5. Points that correspond to a one in the matrix are plotted, those that correspond to a zero are

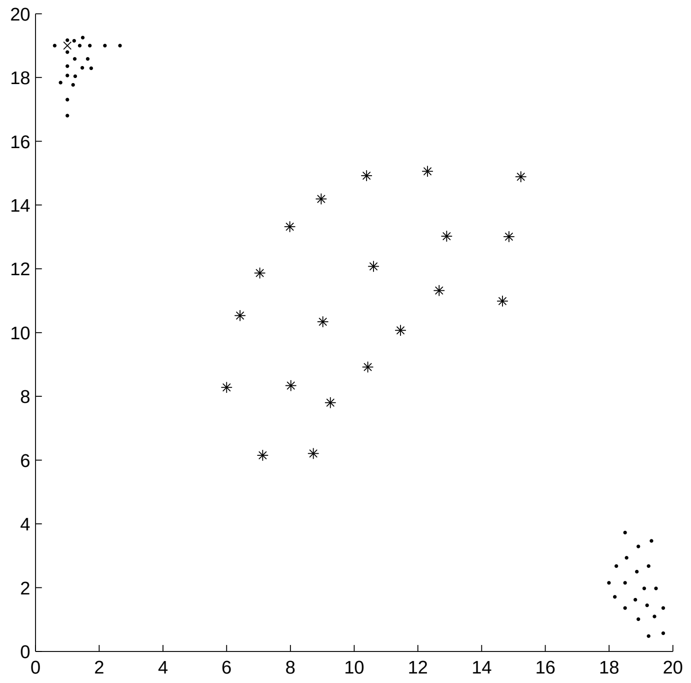


Figure 4.4: A group of 20 agents moving in a free field. The agents are depicted by dots for the first and third snapshots, and by stars for the second. The parameters are $R_{max} = 3$, $d = 2$ and $k_\phi = 0$. Due to the requirement that NF has to decrease monotonically, the formation keeping is disturbed.

not. The generator point is marked by a star (but is set to zero in the matrix). Since we have information on the obstacles only in the grid points, the calculation is very conservative, assuming that the obstacle occupies one grid length above and below the generator point. The analysis is performed only for generator points in the first octant (including the 45° diagonal), but points outside this sector may be marked as obscured.

By transposing the matrices, flipping them up–down and left–right we can use the same set for analyzing occupied points in any octant, thereby reducing the amount of memory required for the precalculated data. Finally all matrices corresponding to occupied points are superimposed on the grid, using an AND operation, leaving only the points that are not obscured by any obstacle.

4.2.3 Finding the optimal step Step five in the algorithm is formulated as an optimization over all points that fill the conditions (3.3)–(3.6), but in the actual implementation we have to make a finite list of candidate points. We use three types of candidate points, described below in order of priority.

As the first candidate we use the centroid itself, since it is obviously the choice minimizing $||p_d - C_V||^2$. The only difficulty is testing condition (3.5), i.e. that there is free sight from the agent to the centroid. We have chosen a safe approach, testing the four points in G (4.1) that surround the centroid. If they are all within line of sight, the centroid is considered to be feasible too and the agent steps directly there.

If that is not the case we test which of the feasible points in G have a lower navigation function value than the present position, and give each of them a scalar value $||p_d - C_V||^2$. Among all remaining candidates, we step to the one with the lowest value.

When trying this approach we discovered that agents sometimes got stuck when

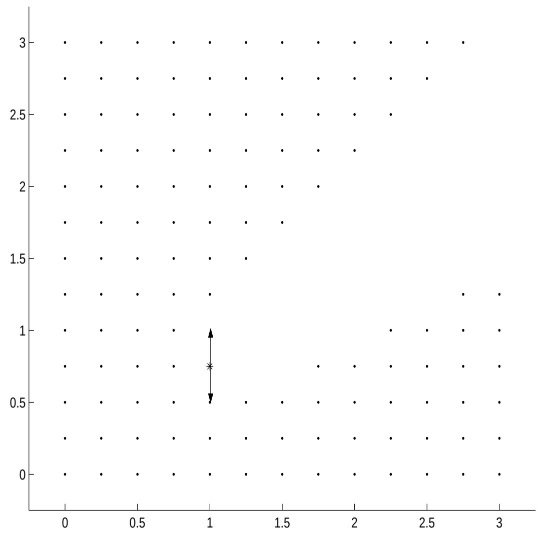


Figure 4.5: An illustration of the table showing what points are obscured, viewed from the origin, if the point with coordinates (1, 0.75), marked by a star, is occupied. The double arrow indicates the assumed obstacle shape, one grid length above and below the generator point.

turning sharp corners. The situation is depicted in Figure 4.6 where the goal is situated far up left, outside the figure. The agent is denoted by a diamond shaped symbol, standing in the grid square just below the corner. The thin lines are level curves of $NF(q)$ and the dots are the points in G. Due to a "ridge" in the navigation function, no point belonging to G has a lower navigation function value than the present position, so the agent would stand still. In situations like this we introduce four special candidate points

$$\begin{aligned}
(x,y)_{right} &= (x^{+} + 0.9 \cdot (x^{+} - x), y), \\
(x,y)_{up} &= (x, y^{+} + 0.9 \cdot (y^{+} - y)), \\
(x,y)_{left} &= (x^{-} + 0.9 \cdot (x^{-} - x), y), \\
(x,y)_{down} &= (x, y^{-} + 0.9 \cdot (y^{-} - y)),
\end{aligned}$$

where (x, y) are the coordinates of the agent and x^{+} is the x-coordinate of the vertex in the obstacle grid closest to the right of the agent. The other coordinates x^{-}, y^{+} and y^{-} are defined analogously.

The special candidate points are thus mirror images of the agent, mirrored in the four sides of the obstacle grid square that the agent stands in. But because of the 0.9 factor the mirror images are closer to the sides than the agent, so one of them always has a navigation function value that is a little lower than that of the present position. This candidate point is then chosen, although the decrease of the navigation function might not be as large as prescribed by (3.4). It is important to note that the creation of the special candidate points does not violate the presented algorithm, but is a consequence of the discretization needed for computer implementation. Equations (3.2)–(3.6) are formulated as an optimization over a continuous set, so we are free to test any candidates in the set. The violation instead lies in the special candidate point not fulfilling equation (3.4), which also represents a form of discretization. A suggestion on how to overcome this in future versions was given in section 3.6.

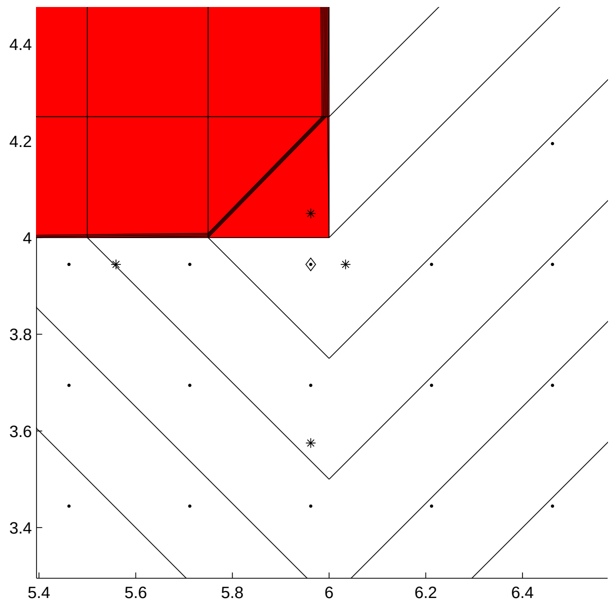


Figure 4.6: An agent about to turn a corner, making use of one of the four special candidate points denoted by stars. The agent is denoted as a diamond shape, the dots are points $q \in G$ and the lines are level curves of the navigation function $NF(q)$. The goal is far up left, outside of the figure.

4.3 Practical experiences

The aim of this section is to briefly account for some issues that we have encountered during the development of the algorithm. Hopefully it could be useful for someone trying to implement the algorithm or develop it further.

4.3.1 Trimming the integration regions to a convex shape The first and greatest problem has been that of rounding sharp corners. Our initial thought was that just intersecting the Voronoi regions with the obstacle-free space and then integrate over this area, V, to find C_V might move the centroid away from the corner enough for the robot not to collide with it. So in this early version, the robot always stepped directly to the centroid, without checking if it was safe. This did not always work, instead the robots in some situations tended to cut the corners as depicted in Figure 4.7.

We quickly concluded that one way to ensure an obstacle-free step to the centroid would be to make sure that the integration region was always convex. So we worked for a while on strategies for trimming the region into always being convex while still preserving enough of its original shape to get forward motion at all times. It proved difficult, so eventually we instead reformulated the problem into an optimization framework (3.2)–(3.6). This gave the possibility to explicitly require that the path chosen be within line of sight, thereby guaranteeing us against obstacle collisions.

Another problem with the first approach was that agents could “see through” thin walls, including points that were within R_{max} but on the other side of obstacles in the integration. This could lead to an agent stepping right into or through a wall, but was solved when we required that all points in the integration region had to be within line of sight.

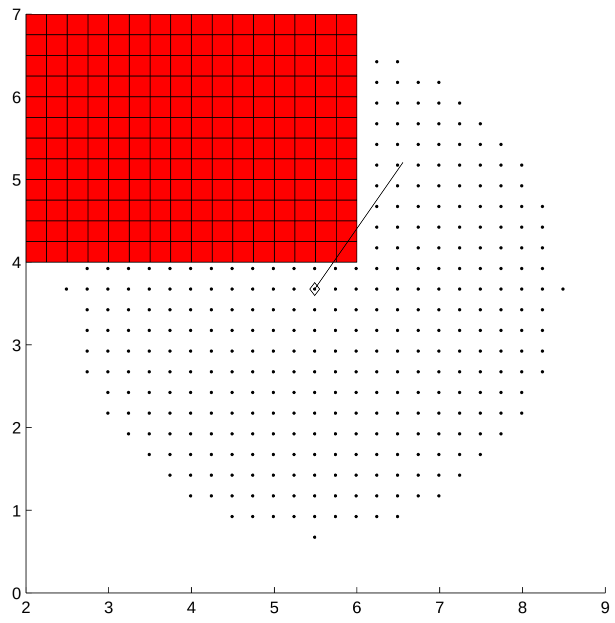


Figure 4.7: An agent cutting a corner because there is no constraint that the target point of a step has to be in line of sight from the agent. The agent is depicted as a diamond shape, the dots are points that belong to the integration region V and the line shows the calculated step. The goal is outside the figure, far up left.

4.3.2 Voronoi regions in vertex form When calculating the Voronoi regions we first used an elaborate scheme for finding the vertexes of the region. This included also finding "virtual" vertexes, where an unbounded region was truncated at the distance R_{max} from the agent. It was based on the idea of first finding the separator lines half way between the agent and every neighbor, and then walking along the innermost polygon formed by the lines (or by arcs centered at the agent and with radius R_{max}). This lead to long calculations and numerical problems when several lines crossed very close. So we eventually settled for another representation of the Voronoi region; that of normal vectors of the separator lines. Every neighbor p_i to agent p_j generates a separator line with normal vector

$$\begin{aligned} e_i &= \frac{p_j - p_i}{||p_j - p_i||}, \\ \text{and a scalar} \\ b_i &= [p_i + \frac{1}{2}(p_j - p_i)] \cdot e_i. \end{aligned}$$

Any point q then belongs to the Voronoi region if and only if $q \cdot e_i \geq b$ for all neighbors i. The condition can be formulated as a matrix multiplication to test for all separator lines at the same time, so it executes very efficiently in Matlab.

5. Implementation in Fenix

As a complement to testing our algorithm in Matlab, we wanted a more realistic environment. FOI has developed a system consisting of physical robots and a graphical computer simulator, sharing a common controller interface. So an application programmer can make a program for the computer environment and then reuse the same code in the physical robots. The simulator software is called Fenix, and this chapter describes the system and our implementation.

5.1 System overview

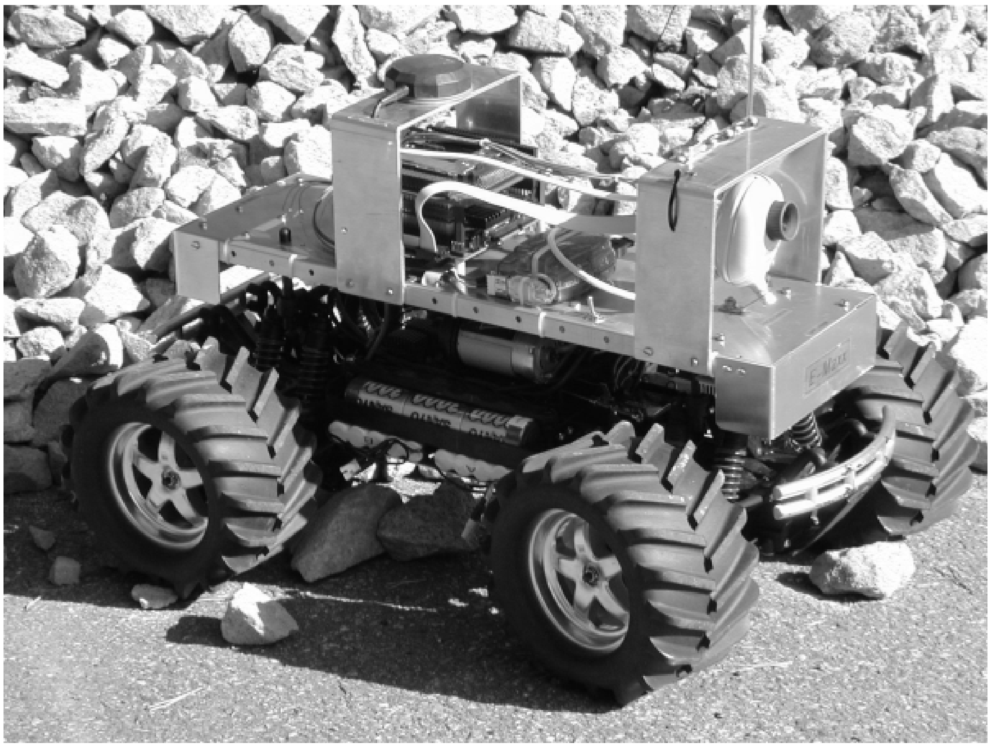


Figure 5.1: One of the radio-controlled cars at FOI.

The physical robots in the FOI testbed for robot controllers are redesigned radio-controlled cars (Figure 5.1). Every car is equipped with a card-PC, actuators for thrust (and thus braking) and steering, a WLAN transceiver for short-range communication and a GPS receiver for navigation. The platform is rugged enough for outdoor terrain use, although the surface needs to be reasonably smooth because of the diameter of the wheels. At the moment only two cars are operational, so to better demonstrate the flocking properties of our algorithm we decided to implement it in Fenix. It has a ready-made car model whose actuators have the same interface as the actuators of the physical car. The information available to the controller is also the same as in reality. The controller can extract the position of the car (although there is no error as in a GPS measurement) and, as will be explained in subsection 5.3, inter-agent communication is simulated by using a global list of positions where every single agent can only get a list of its neighbors within a specified radius. As described in the background section, a side-effect of having a GPS receiver on the car is that it has access to a very accurate time estimate. This is used to synchronize all cars and make them plan their next step at the same time. This is crucial for avoiding ambiguities in the Voronoi partitioning of the available space.

In this section the terms target and goal will be used for two different things. The goal is the global goal determined before executing the program and it is the same for all robots. The term target refers to the individual target that is assigned to each individual robot by its flocking algorithm, and it is updated at every replanning interval.

5.2 Implementational issues

This section describes some problems that arose when implementing the algorithm in a more realistic environment, and the solutions to them.

In Section 3.5 is described how we find the centroid of every Voronoi region by integration:

$$C_V = \frac{1}{M_V} \int_V q\phi(q)\ dq$$

with

$$\phi(q) = e^{-k_\phi \cdot NF(q)}.$$

This worked well in Matlab, where the world had the dimensions 10×10 units. The highest NF value was around 30, so (given that $k_\phi = 1$) $\phi \in (9 \cdot 10^{-14}, 1)$, which lies well within the range of the floating-point type of Matlab. But when we enlarged the world by almost a factor 1000, this was no longer true. The C++ variable types and mathematical functions could not handle numbers that small. Just changing k_ϕ, i.e. scaling the navigation function, did not work either, as it gave centroids that almost coincided with the geometric centroids. The solution was normalize the navigation function in the region around the agent and use

$$\phi(q) = e^{-k_\phi \cdot (NF(q) - NF(p_j))},$$

where p_j is the position of the agent. This way we both reduced the deviation from 1 and distributed the NF values more evenly around 1 to make better use of the full range of the floating-point types.

We have also expanded all obstacles by 4 m to compensate for the width and length of the robots. Our algorithm ensures that the centre of the robot does not collide with anything, but this sometimes lead to the robot going too close to the edge of an obstacle.

5.3 Software structure

The application programmer makes a C++ class that implements a controller for the car and is called at every update interval by the simulation engine. In principle the same code could then be transferred to the card-PC of the physical cars and used there as well. The structure of this software is described in this section, which is mainly intended for someone who wants to reuse the existing code or get hints for a similar programming task. It assumes that the reader has basic knowledge of C++ and object-oriented programming and it does not add significantly to the understanding of the more abstract overall algorithm.

All objects in the simulated world are created by an object factory, i.e. an underlying program that takes input from a script file and then creates as many instances

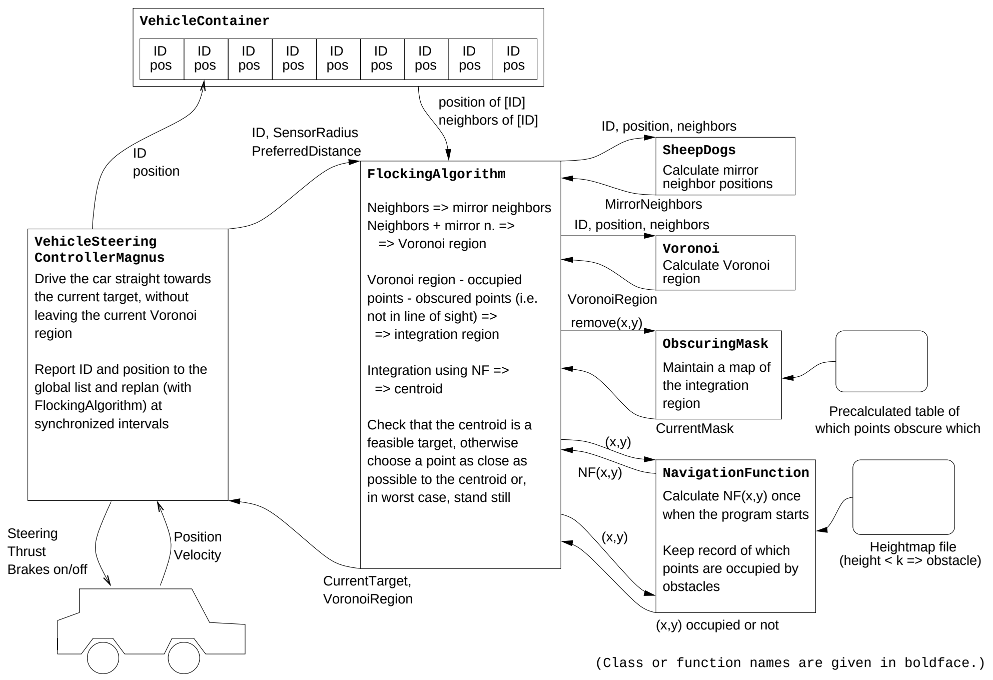

VehicleContainer
ID pos
position of [ID]
neighbors of [ID]
ID
position
ID, SensorRadius
PreferredDistance
FlockingAlgorithm
Neighbors => mirror neighbors
Neighbors + mirror n. =>
=> Voronoi region
Voronoi region - occupied points - obscured points (i.e. not in line of sight) =>
=> integration region
Integration using NF =>
=> centroid
Check that the centroid is a feasible target, otherwise choose a point as close as possible to the centroid or, in worst case, stand still
VehicleSteering ControllerMagnus
Drive the car straight towards the current target, without leaving the current Voronoi region
Report ID and position to the global list and replan (with FlockingAlgorithm) at synchronized intervals
ID, position, neighbors
SheepDogs
Calculate mirror neighbor positions
MirrorNeighbors
ID, position, neighbors
Voronoi
Calculate Voronoi region
VoronoiRegion
remove(x,y)
ObscuringMask
Maintain a map of the integration region
CurrentMask
Precalculated table of which points obscure which
(x,y)
NF(x,y)
NavigationFunction
Calculate NF(x,y) once when the program starts
Keep record of which points are occupied by obstacles
(x,y)
(x,y) occupied or not
Heightmap file
(height < k => obstacle)
Steering
Thrust
Brakes on/off
Position
Velocity
CurrentTarget,
VoronoiRegion
(Class or function names are given in boldface.)

Figure 5.2: Software structure of the car controller

of different objects as the file specifies. It also connects the objects to each other and initializes them as described in the script. An example of this is that the car is described as a chassis, connected by springs to four wheels. Then actuators are initialized and connected to the wheels and finally an instance of the **VehicleSteeringControllerMagnus** class is created and given handles to the actuators. When the script file specifies that 20 cars should be created, 20 similar instances are set up and connected, with one VehicleSteeringControllerMagnus object each. A manager class places the cars at different positions so that they will not collide from the start.

The basis of the car controller is the VehicleSteeringControllerMagnus class, that issues the actual actuator commands. They are of three types; first the steering command that has to be in the interval $(-1, 1)$, with -1 meaning full steering to the left. Then there is the thrust command that is also in the interval $(-1, 1)$, with 1 meaning full forward thrust and -1 full reverse thrust. Finally there is a separate braking command that overrides the thrust command. It has only two states: on or off, and when the brakes are on the thrust does not matter, the wheels are braking hard. The VehicleSteeringControllerMagnus class contains a few important variables:

- *mIDNumber*: The global ID of the car that is controlled by this instance. It is issued by the VehicleContainer class as described below.
- *drivingMode*: The state of the controller state machine, described in subsection 5.4.
- *planningMode*: The state of the replanning state machine, responsible for recalculating the current target and Voronoi region at synchronized intervals. It also reports the position of the car to the global list (VehicleContainer) before replanning, so that all positions in the list are updated.
- *currentTarget*: The coordinates of the current target that the controller should drive to.

- *mVoronoiRegion*: The Voronoi region that the car has to stay inside to avoid colliding with other cars.

The workings of the controller are described in subsection 5.4, but its task is to drive the car straight to the current target, without leaving the Voronoi region. The higher-level flocking algorithm guarantees that this straight path is free of obstacles.

The **VehicleContainer** class has only one global instance. It maintains a list of all cars and their position. The first time a car reports to the list it is issued an ID number that will identify it during the current execution of the program. The list contains global information on all cars, but they can only access the list by specifying their ID number and a sensor radius, and then a list of neighbors within the sensor radius from the position of the car is returned. So the controller only has access to local information.

The actual flocking algorithm as described in earlier sections is run in the **FlockingAlgorithm** function. Every execution of the function calculates one iteration step in the algorithm. It first compiles a list of possible mirror neighbors by calling the **SheepDogs** function. Then the mirror neighbors are combined with the actual neighbors and the whole list is used to create a **VoronoiRegion** instance, containing the boundaries of the Voronoi region around the car.

Then an integration grid is created, centered at the position of the agent and containing only points that are within the sensor radius R_{max}. It is created by and stored in an **ObscuringMask** object. Every point in the grid is tested to see if it lies inside the Voronoi region and is free from obstacles. If not, the ObscuringMask object is given the command to remove the point from the grid *as well as the points that are obscured by the point in question.* This is done using precalculated masks that are stored in memory to make the process faster. When all points are tested the centroid is calculated by equations (4.2–4.3).

For the calculation of the centroid, the algorithm needs the navigation function, NF. It is maintained by a global instance of the class **NavigationFunction**, that gets its original data from the same file that the graphics engine uses to determine the contours of the ground. This is an image file (in Windows 24-bit bitmap format), where dark colours mean low areas and light colours indicate high regions. Regions with an altitude below a predefined threshold are considered as obstacles. (This might appear counter-intuitive, but we chose to use ditches and holes as obstacles for two reasons: first of all it is self-indicating if a car has hit an obstacle, as it gets caught in the ditch, and secondly this makes it easier to see all cars than in the case of walls.) The NavigationFunction object also contains coordinates for the global goal. When these coordinates are changed or initialized, the values of NF are calculated over the whole world.

Finally the FlockingAlgorithm function has to check that the centroid is in the line of sight and that it its NF value is an improvement compared to standing still. If not, all points that were evaluated in the integration are tested to see if they are feasible candidates for being the next target. Among the feasible candidates the one closest to the centroid is chosen. If there are no candidates among the integration points, four special candidates are computed, as described in 4.2.3, and tested according to the same criteria as the centroid. If there are no candidates whatsoever, the next target is chosen to the current position, to make the car stand still. The final choice of target is returned together with the current Voronoi region to the calling VehicleSteeringControllerMagnus object.

5.4 Robot dynamics and controller

The low-level controller, implemented in the VehicleSteeringControllerMagnus object, has the following task:

- Drive the car from an arbitrary position and orientation to the target specified by higher level algorithms.
- The orientation of the car at the target is not important.
- There is line of sight from the start position to the target, but if the car has to manoeuvre it must not leave its Voronoi region.

Since the controller does not know in what direction the car will have to go in the next iteration, we decided not to put any constraints on the final orientation of the car. Instead we prioritized to quickly reach the target.

The car model in the simulation environment is modeled after the US Army HMMWV all-terrain vehicle that is four-wheel driven, weighs 2.3 tons and measures about 5 by 2 metres. Since its physical dimensions are so much larger than those of the radio-controlled cars the simulated world has been enlarged to a width of several kilometres. To make this section more general we have avoided specifying numerical values, but the values used for our simulations are listed in section 5.5.

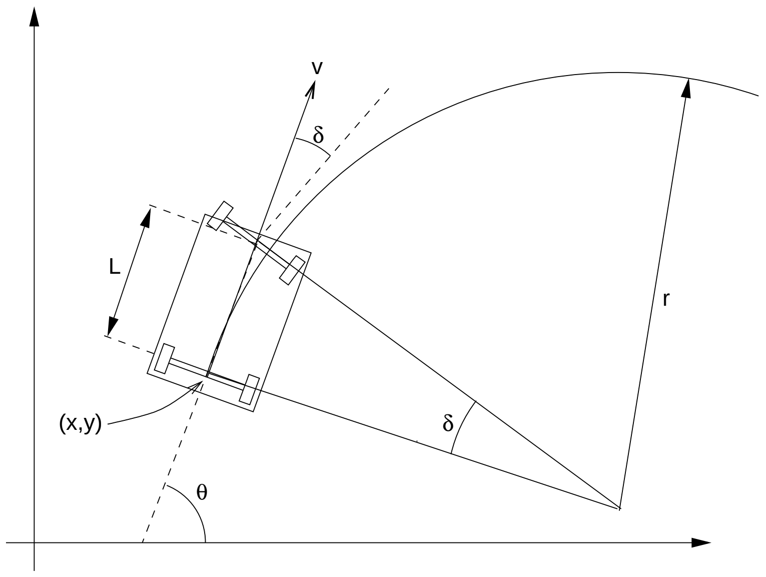


Figure 5.3: A model for the car.

Our car model is depicted in Figure 5.3. When steering, the wheels actually turn each around their own vertical axis and not around a common axis as in the model, but the effect is the same. How does the heading θ change over time? Let us assume that the car moves with a constant steering angle δ. It will then follow a circle with radius r, and basic trigonometry tells us that

$$\frac{L}{r} = \tan\delta. \tag{5.1}$$

During the time $\mathrm{d}t$ the car will travel the distance $v\,\mathrm{d}t$ along the circle. The heading then changes as

$$v\,\mathrm{d}t = r\,\mathrm{d}\theta \Rightarrow \frac{\mathrm{d}\theta}{\mathrm{d}t} = \frac{v}{r} = \frac{v}{L}\tan\delta.$$

The coordinates of the rear of the car evolve depending on θ and v and to account for the inertia of the car we added an integrator to the speed dynamics. The whole model becomes, [15]:

$$\begin{aligned}\dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta, \\ \dot{\theta} &= \frac{v}{L}\tan\delta \\ \dot{v} &= u\end{aligned}$$

In [15] it is shown that the time-optimal path for such a vehicle consists of a number of circular arcs of minimum turning radius, combined with straight line segments. Both the position and orientation of the car at the start and end points can be specified, but as stated above, we decided to relax this by just specifying the final position. Using these results, we decided on a simple and robust controller that starts by turning towards the target with a minimum turning radius and a constant speed, chosen not to give slipping, and then to drive straight ahead at a higher speed until it reaches the target. The speed is maintained by a simple P-controller:

$$u = k_v\,(v_{desired} - v).$$

The surface of the simulated environment is rather slippery, so if the car uses too much thrust there is a risk of slipping. We therefore chose $k_v = 0.5$, which empirically proved a good compromise between fast acceleration and avoiding slipping even when the reference signal, $v_{desired}$, changed fast. After a few accidents when the above P-controller tried to slow down by reversing the thrust and thereby lost the grip, we added a simple rule that if the speed is more than 2 m/s above the desired (when the P-controller saturates), we use the brakes to slow down instead, as this gives a much better grip. When turning, the steering angle δ is given its maximal or minimal value, depending on direction, and the speed is adapted to avoid slipping while still efficiently performing the manoeuvre. When the car is turned towards the goal it switches to driving straight ahead with a higher speed and a P-controller that directs the car towards the target:

$$\delta = \frac{\beta}{\delta_{max}} \tag{5.2}$$

where β is the angle between the velocity vector and the relative vector from the car to the target, chosen to be positive when the target is to the right of the car.

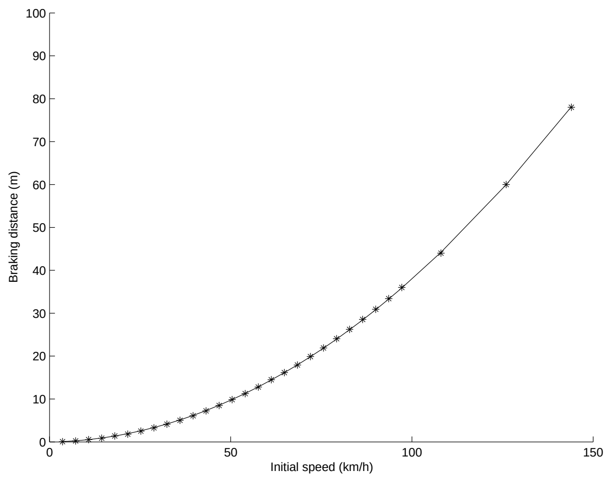


Figure 5.4: The braking distance, s m, as a function of speed, v m/s. The stars denote measured values and the line is the interpolating curve $s = 0.033v + 0.048v^2$.

Finally the car needs to be stopped at the target and this is done by “bang-bang control”. We constantly predict the braking distance as a function of the current speed and hit the brakes when that distance plus five metres (a safety margin) remains to the target. When testing the brakes at different speeds we obtained the data in Figure 5.4. We made a least-square fit with a second-degree polynomial, as the braking distance is usually a function of the kinetic energy, which in turn is proportional to the square of the speed. It showed that the constant term was in the order of a few centimeters, so we neglected it and made a new least-square fit to a simpler polynomial of the type

$$s = a\,v + b\,v^2,$$

where we got the values a=0.033 s and b=0.048 s^2/m. The applicability of this approach is confirmed by the excellent fit of the curve, depicted in Figure 5.4.

The controller above worked well, except for so called small-scale controllability problems, i.e. when the target was very close to the original position of the car. Everyone who has driven a car knows that it is easier to drive it to a point fifty metres away than a point just one metre to the side. The problem of small-scale controllability concerns areas that cannot be reached by the above scheme of a forward turn and driving straight. Figure 5.5 depicts the different zones around the car, particularly the near zones N_L and N_R where this occurs. The radius R is the turning radius of the car and was determined by testing on flat ground. All other target locations could be reached by the controller. So we added two backing states when the car drives backwards in a circle of minimum radius until it points in the direction of the target. To reach targets in N_L, the car reverses with full right steering along the boundary of N_R, which makes its forward direction sweep over the whole of N_L. The opposite works for targets in N_R. The car then drives straight ahead to the target as described above.

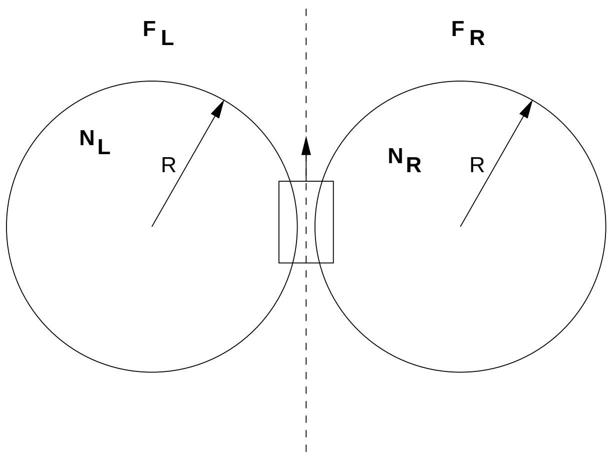


Figure 5.5: The different target zones for the car controller. $N_L \notin F_L$ and $N_R \notin F_R$. R is the turning radius of the car. A target right behind the car can be considered to belong to any of the zones F_R or F_L.

Then the problem of not leaving the Voronoi region remains. As the higher-level algorithm guarantees that the target will be inside the convex Voronoi region, this will only be an issue if the car has to turn before driving straight ahead. When doing this it constantly predicts its position two time steps ahead and checks that it will be inside the Voronoi region and not occupied by an obstacle. If the predicted position is not feasible, the car reverses both its forward velocity and steering, so a right forward turn becomes a left backwards turn and vice versa. This is what a human driver does when parallel parking. If the car risks leaving the Voronoi region when reversing to reach a target in a near zone, it drives straight ahead for 10 m (or until it reaches another Voronoi boundary) and then replans the whole manoeuvre. This is not necessarily optimal, but it was an easy way to find a strategy that works even in this very odd case. Figure 5.6 illustrates three possible trajectories of the controller.

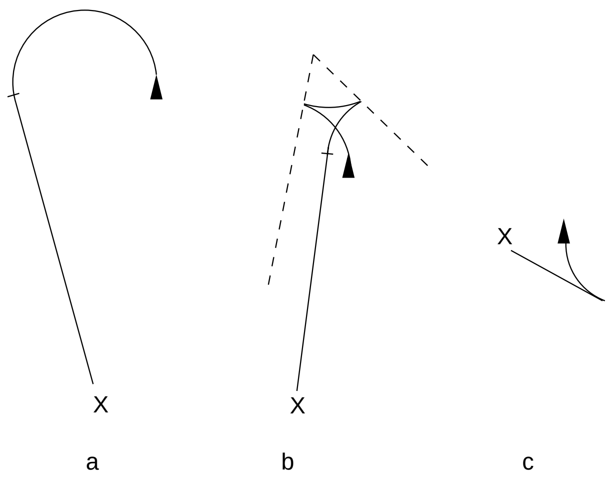


Figure 5.6: Three trajectories produced by the car controller. The triangle shows the initial position and orientation of the car, and the x is the target location. In a) the car has sufficient space to turn, while in b) the dashed line shows the boundaries of the Voronoi region that must not be passed. In c) the target is in the near left zone, so the car reverses before driving straight ahead.

We implemented this as a state machine, as described in Figure 5.7. To prevent unnecessary manoeuvering when the car was very close to the target we chose a distance D, inside which the car is considered "close enough" so it stands still. After every replanning, i.e. a new calculation of target and Voronoi region, the state machine is reset to the START state. If only one or very few cars are used, the targets are sometimes placed so far ahead that the cars have not reached the targets, and thus have high speeds, when replanning. When this happens we decided to skip the low speed turning phase and make the course corrections at high speed instead to make the motion smoother. To avoid skidding off the desired path because of too steep turns, this is only allowed if the new target is within an angle α from the forward direction.

5.5 Values used for the simulations

We have deliberately avoided specifying values of many constants mentioned in the above text, so as to make it more general and applicable to any size and type of car-like robot. In this section we specify the values for the simulations that will be discussed in the next section.

Figure 5.8 shows a map of the simulated world, where the black areas are obstacles. We chose to use ditches, about 15 m deep, as obstacles instead of walls or hills for two reasons. Firstly it enables us to overlook the whole course and see all cars at once. Secondly, if a car hits a wall it might bounce off the obstacle without the observer noticing it, while if it falls into a ditch it stays there.

Table 5.1: Values used for the simulations in section 5.6.

Quantity	Variable name	Value	Remark
Grid spacing for integration and NF		10 m	
Width and length of the world		2048 m	
Maximum height of the world		20 m	
Integration factor	k_ϕ	0.05	Defined in Section 3.5.
Replanning interval	t_r	1 s	A short interval gives smooth motion, but longer intervals improve formation stability.
Sensor radius	R_{max}	200 m	
Preferred distance	d	100 m	This is adapted to the granularity of the integration grid.
Car controller update interval	t_c	50 ms	This was set by the simulation engine.
Near-target distance	D	10 m	This is how close to the target the car controller is "satisfied" and stops.
Straight ahead angle		10°	Within this angle from the forward direction the controller considers the target to be straight ahead, and uses the control law (5.2).
High-speed turn angle	α	30°	See the controller state diagram (Figure 5.7).
Maximum speed		110 km/h	The desired speed when driving straight ahead towards the target.
Curve speed	v_c	20 km/h	The desired speed when making minimum-radius turns.
Prediction horizon		1.5 s	When making turns, the controller predicts the position of the car this far ahead in time, to check that the position is safe. It is chosen to be the approximate braking time when traveling at curve speed.
Turning radius	R	10 m	See Figure 5.5. The actual radius was measured to 8 m, so this includes a margin.
Maximum turning angle	δ_{max}	30°	Estimated value, but agrees well with (5.1).

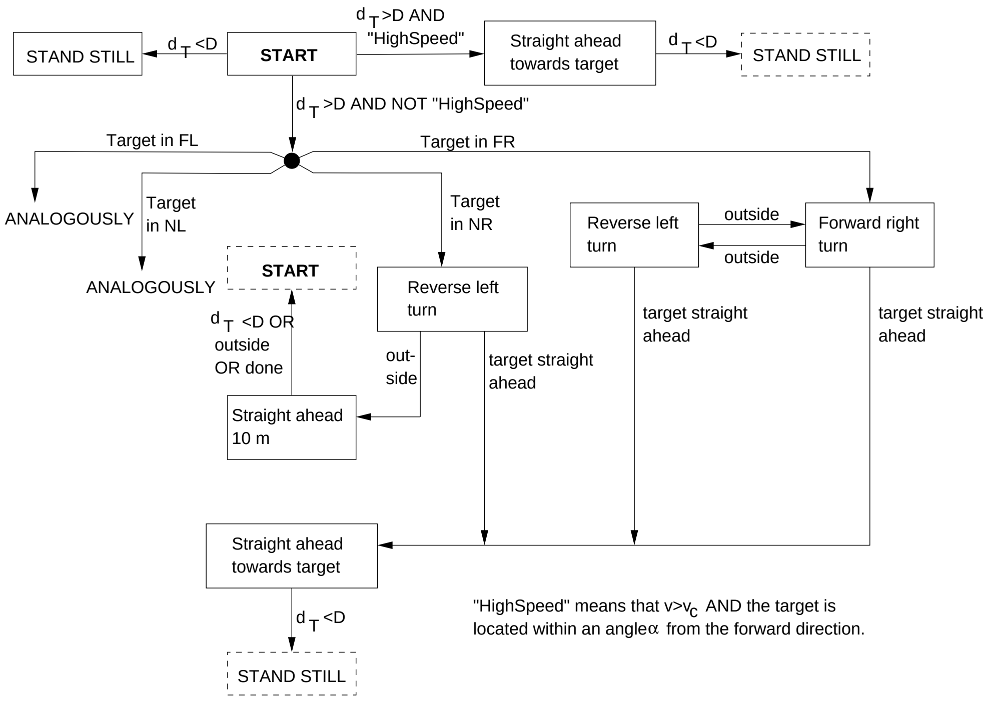


Figure 5.7: State diagram for the car controller, where the left part is omitted as it is analogous to the right side. The zones are named according to Figure 5.5, d_T is the distance to the target and D is the distance "close enough" to the goal. Dashed boxes are references to another state, repeated for clarity. The speed v_c refers to the speed used for minimum-radius turns. The condition "outside" means that the predicted position of the car is outside the Voronoi region.

5.6 Results

We have simulated three scenarios in the world described in Section 5.5: First we tried letting one single car start at position A and drive to the goal, then we started a group of ten cars at A to see how they coped with the obstacles and finally we started a group of 20 cars at B to study formation stability.

The single car reached the goal in 1 min 13 s. When turning the corners of the obstacles it slowed down to the curve speed v_c, which delayed the driving considerably but increased safety. It hit the target head on, without any need to make additional corrections before coming to a complete stop.

The group of ten cars also arrived safely at the goal, and reached the configuration depicted in Figure 5.9 after 2 min 45 s. There was a tendency of the cars blocking each other when driving onto the narrow bridge at C, but it never led to a locking situation with all cars standing still. There were no collisions even at the start, when the cars were lined up only ten metres from each other. The small-scale behaviour of the controller worked well, carefully backing or turning away from the others. We tried starting the group from different positions around the world and the obstacle avoidance worked well, except for some rare cases when the cars stood still for a long time waiting for their "turn" to pass a narrow bridge. When standing still the cars slide slowly to the side due to imperfections in the simulation engine, and this could lead to tipping over the edge into a ditch. This sliding mechanism has, on the other hand, also resolved some locking situations when several cars have blocked each other at narrow entrances.

When simulating the group of 20 cars, the computer could not run the simulation

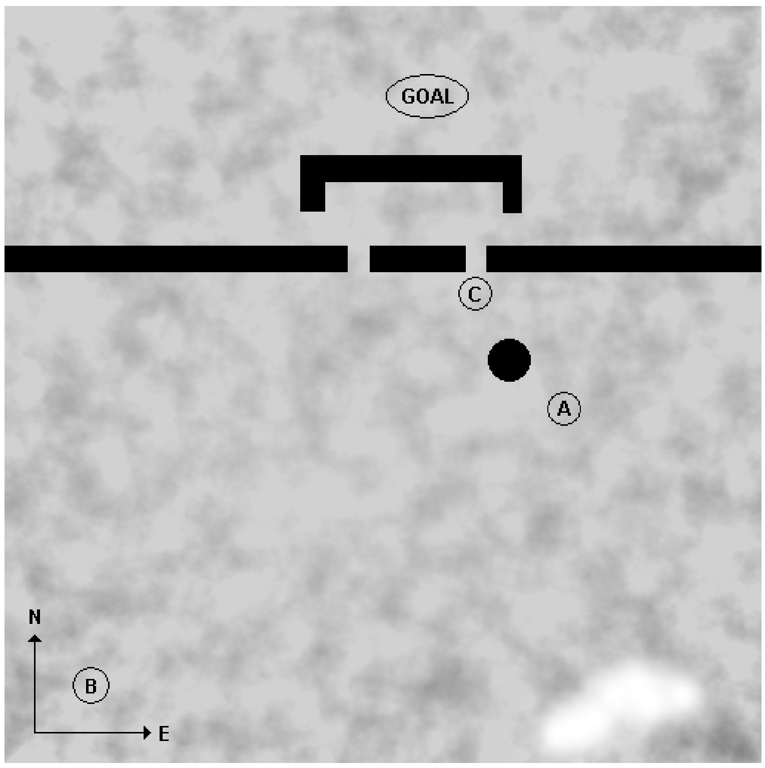


Figure 5.8: A map of the simulated world that measures 2×2 km^2 . The black areas are obstacles in the shape of ditches. The goal is indicated, as well as starting points A and B and the entrance of a narrow passage at C.

in real time due to the increased processing load. In a physical, truly distributed implementation this would not be a problem, as the only added complexity for the individual agent is that of computing more boundaries for the Voronoi region. It was discovered that the stability of the formations depended on the replanning interval t_r of the algorithm. Long intervals gave the cars time to reach their targets between every iteration, so the algorithm worked much like in the simplified Matlab environment. As can be seen in Figure 5.10, the cars do not form a truly hexagonal lattice but stay well collected. The reasons that they do not form a lattice are probably that the surface is uneven, which acts as a disturbance, and that robots on the side of the formation tend to zig-zag ahead as described in section 3.6. When the replanning interval was shortened the cars rarely made it to their targets. This smoothed the motion since they did not stop, but it disturbed the formations much like the case described in section 4.1, as the cars did not really go to the centroids of their Voronoi regions.

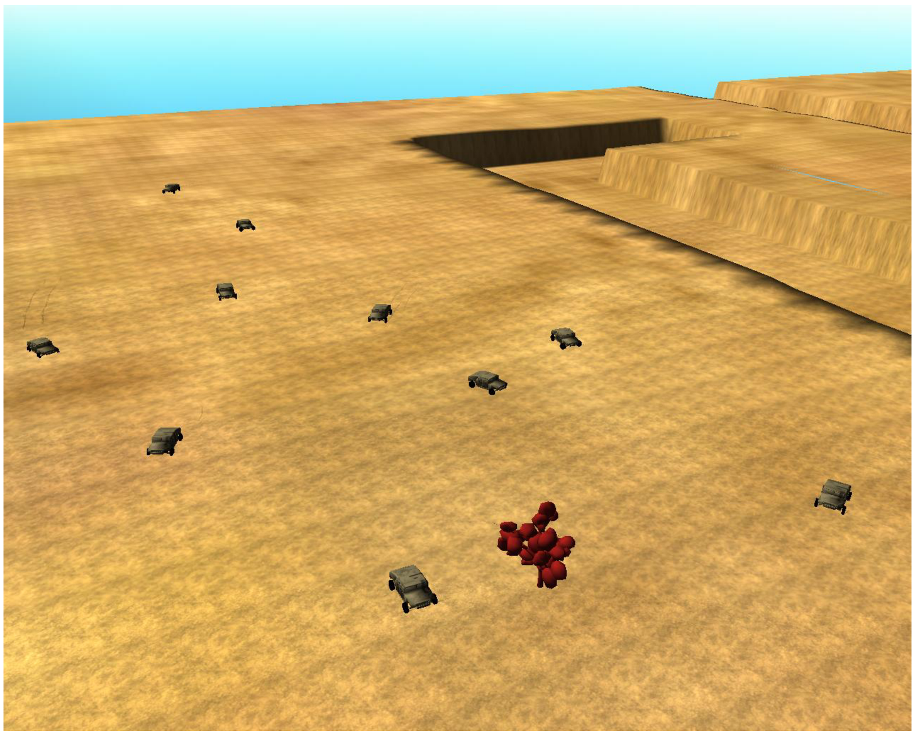


Figure 5.9: A screenshot from the simulation program showing ten cars at the goal, marked by a tree.

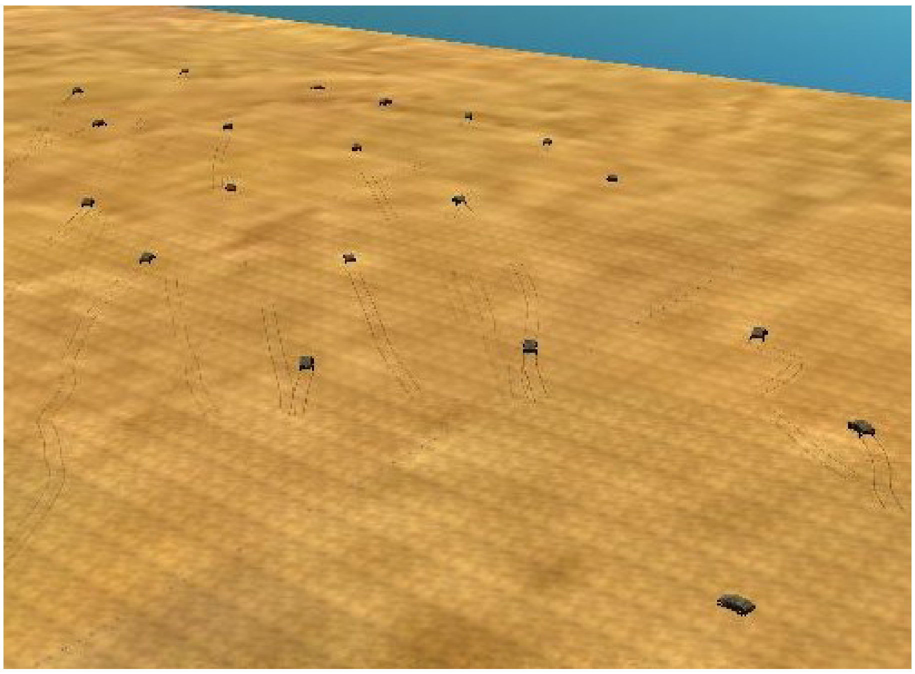


Figure 5.10: A screenshot from the simulation program, showing twenty cars moving in a loose formation. The goal is located outside the picture, far up to the left.

6. Sensor deployment

The preceding sections have described an algorithm for moving a flock of robots from arbitrary starting points to a goal, using sensors mainly as a means of detecting obstacles or other robots. We now study how to use the sensors to detect an external event, once the group has reached the goal area. A simple way of switching from navigation to a sensor-coverage strategy is presented and this can be done in the same algorithm framework as described above.

This is a useful approach for military applications, where the most popular use for mobile robots so far is as sensor platforms. Robots can be sent into dangerous areas, they can stay inactive and wait for very long periods of time without any need for supplies and UAVs can be made much smaller and thus stealthier than manned aircraft containing support systems for the pilot. As the costs for the needed hardware decrease even further, small robots can be used in masses for missions that were earlier too unimportant or unqualified for military personnel. An example of this is given by Rybski et al., [16] , who describe a system of *rovers*. They are larger robots that carry small *scouts* that can be launched in areas of interest. The scouts can roll or jump with limited control and are equipped with sensors and radio links to relay sensor data to the rover.

6.1 Sensor types

There are many sensor types that can be of interest in military applications, such as:

- Cameras, working in the infrared or visible spectrum. The images can be fully or partially processed by the robot or simply fed to an operator that does the interpretation. An example of partial processing can be detecting motion or tank-like objects in the image and then alerting an operator, which reduces the workload of the operator in case of a large amount of sensors.
- Artificial noses sniffing for explosives, even those who are contained in underground mines. [17]
- Down-looking radar, used for minesweeping.
- Microphones, listening for the sound of artillery firing. The location of the firing unit can then be determined by solving for the time of arrival of the sound wave to different robots.
- Antennas, used both to detect and possibly record enemy radio traffic and also for locating the transmitter.
- The robot itself can be used as a “sensor” for brute minesweeping. The operator simply drives along a desired path to see if the robot triggers a mine.

The behaviour of the individual robot as well as the flock has to be adapted to the type of sensor used. Artificial noses and down-looking radar have a limited footprint, so the robots need to sweep the area of interest. Others are non-isotropic, such as cameras or directional microphones, and thus require that the direction of the robot is controlled as well as the position. There is also a class of sensors used for triangulation, be it with antennas or microphones. The positions of the sensors then have to be chosen not only with respect to the distance to the transmitter but also the angle to get good resolution in the estimated transmitter position. In the next section we will describe how our algorithm can easily be modified to maximize the combined sensing performance of the whole group for a class of isotropic sensors.

6.2 Switching from navigation to sensing

The algoritm described in section 3.5 is largely based on the coverage control algorithm suggested by Cortés et al., [3]. They consider isotropic sensors working in an environment where a detectable event can occur with a given probability density. They formulate a cost function for the sensor performance of the whole group and show that it is minimized by a gradient descent control law. We replace the probability density with a navigation function to get goal convergence, but one can easily switch to a sensing behaviour by just going back to using the probability density. The mirror neighbor mechanism must be inhibited, but the framework of the algorithm remains the same.

To test this we introduced an Area of Deployment (AoD), a circle with radius R_{AoD} centered at the goal. In this area we assumed a simple probability density

$$\phi(q) = e^{-||q-q_G||^2},$$

where q_G is the goal point. When the robots have entered the AoD they switch to the following algorithm:

1. Calculate V_j from the position of the neighboring vehicles.
2. Calculate C_V according to (3.1), with $\phi(q) = e^{-||q-q_G||^2}$ and $V = V_j \cap \{q : ||q-p|| \leq R_{max}\}$.
3. Calculate p_d from the following optimization problem:

$$\begin{aligned} \min_{p_d} \quad & (p_d - C_V)^2, && (6.1) \\ \text{s.t.} \quad & p_d \in V_j, && (6.2) \\ & ||p_d - p|| \leq R_{max}/2. && (6.3) \end{aligned}$$

   If there is no feasible solution, set $p_d = p_j$.

4. Apply the control $u_j = p_d - p_j$ and repeat from the top.

The above algorithm can be modified to include obstacle avoidance, but we chose to require that the AoD is chosen by the operator to be an obstacle-free area. The reason for this is that otherwise the robots could get caught in non convex obstacles, since the gradient of the probability distribution does not “flow around” obstacles like the NF. Constraints (6.2)–(6.3) still guarantee safety from inter-robot collisions, though. A simulation, using the same obstacles as in Figure 4.1, with the goal at the coordinates (17,27) and $R_{AoD} = 3$ showed the result illustrated in Figure 6.1.

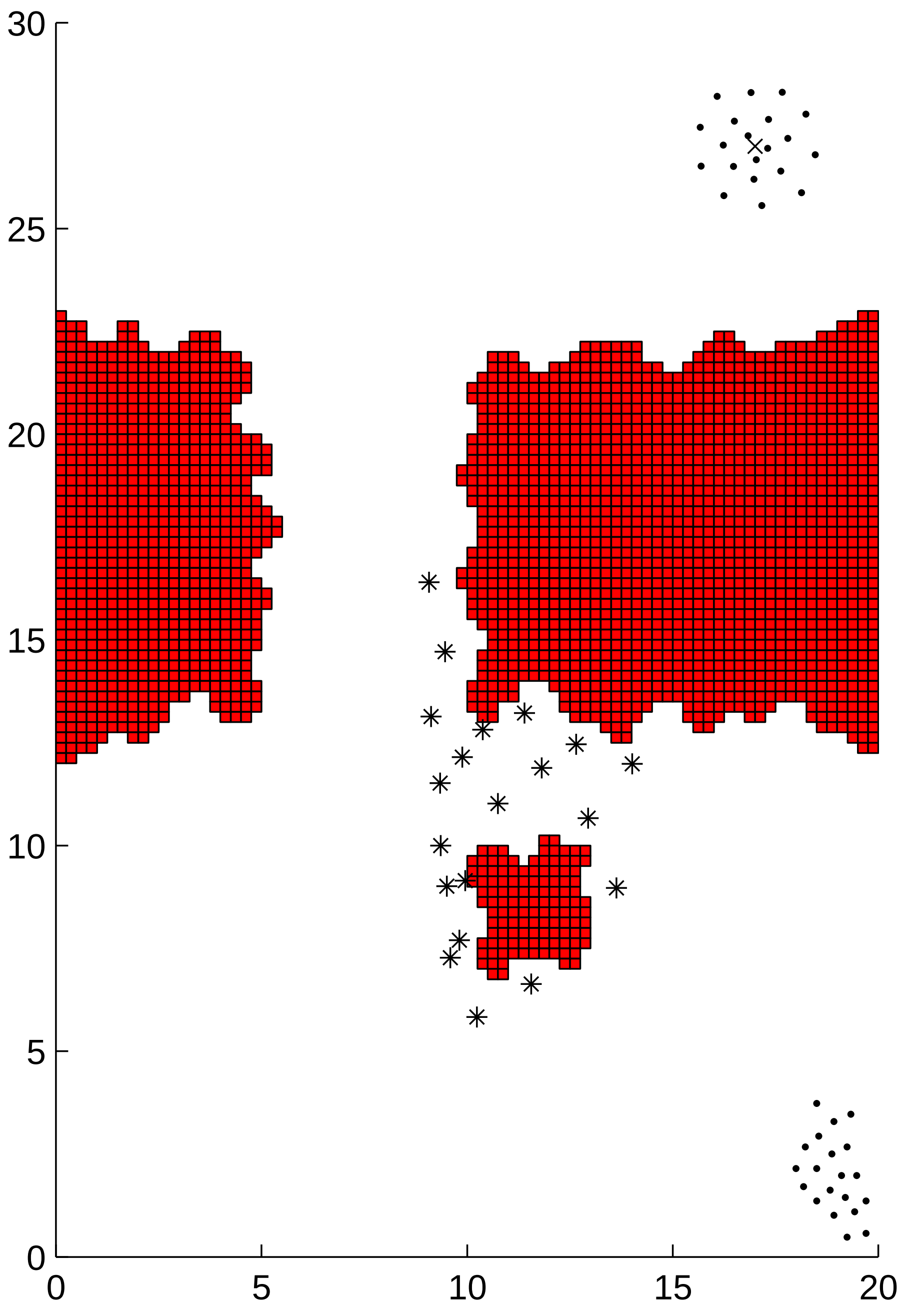


Figure 6.1: A group of 20 agents moving around irregular obstacles and then switching to a sensing behaviour in the Area of Deployment around the goal, marked by an x. The agents are depicted by dots for the first and third snapshot and stars for the second. The parameters are $R_{max} = 3$, $d = 1$, $k_\phi = 1$ and $R_{AoD} = 3$.

7. Conclusions

This report presents a new algorithm for navigating groups of robots from one place to another without colliding with each other or obstacles, regardless of their shape. When the robots are moving in open fields they stay together in a formation that facilitates inter-robot communication and provides good overview for an operator. We use navigation functions to find the shortest unobstructed path to the goal and the centroids of Voronoi regions to avoid collisions while still inducing motion towards the goal. Finally we have added the concept of mirror neighbors, which keeps the flock together and produces formations in open fields. Simulations have shown that the algorithm performs well and we have been able to prove that there will be no collisions. It provides a clear order of priority between the different requirements, where safety has precedence over goal convergence, which in turn is more important than formation keeping.

The algorithm has the advantage of being decentralized, meaning that every robot needs only information about its closest surroundings and there is no central coordinator. The computational demands are almost unaffected by the number of participating robots, which makes the algorithm suitable for scaling to large groups. Finally one can easily switch from navigation to sensing behaviour within the framework of the algorithm by just replacing the navigation function by a probability distribution of the event that is to be detected.

If the algorithm is to be used for controlling underwater vehicles or other kinds of robots that can move in three dimensions, it can be extended to allow for that. All components, such as the navigation function, Voronoi regions, centroids and mirror neighbors can be generalized to higher dimensions in a straightforward manner. As the algorithm is designed for a kinematic robot model (that can assume any velocity without limitations in acceleration), it is not suited for controlling vehicles such as UAVs. As was the case for the cars in Fenix, this requires a lower-level controller that steers the robots between the waypoints generated by our algorithm. This lower-level controller also has to handle situations when the waypoint is not moving, which could prove difficult in the case of an aircraft.

7.1 Suggestions for future work

Ideas for future work can roughly be divided into two categories: studying and improving the navigation algorithm and adding more behaviours when the group switches to sensing in the target area.

Concerning the navigation algorithm, we still have no complete proofs of formation stability. Formally, the question could be stated as: Under what conditions will a flock of robots following the proposed algorithm form a hexagonal lattice formation when moving over an open field where the navigation function forms a plane? And will that formation be stable? To avoid dead-locks, it would also be of interest to modify the algorithm to allow the robots to give way based on some criterion. This could simply be the ID of each robot, as explained in Section 3.6.1, who has the lowest NF value or maybe a reaction-diffusion system as described in [18]. To further reduce the problem of crowding around bottlenecks, the preferred distance d could be adapted according to the speed of the robot or the whole group. When going fast, they could maintain large distances, while in narrow spaces it would be acceptable to go closer. The distance d could also be controlled by an operator, who switches the value depending on the terrain and threat level. While navigating, there could also be “fuzzy” obstacles, such as enemy sites or other places associated with a cost, rather than physical obstacles. Maybe one can add some scalar penalty function to the NF before calculating the centroids, so as to make the robots reluctant to go close to that point.

When the robots reach the goal area and switch to a sensing behaviour, it would be of interest to expand the proposed sensor-placement strategy to allow for non-isotropic sensors or triangulation tasks. The robots could form a giant antenna array, either for localizing a jammer or avoid it or for transmitting home despite heavy jamming. Maybe an antenna array can be formed iteratively by the robots moving around while transmitting, until the resulting radiation pattern has the desired form?

Finally one could imagine that the probability density function (PDF) mentioned in section 6 is dynamically updated according to sensor data. A discovered event (e.g. an enemy vehicle) can generate a narrow peak in the PDF, which will attract the robot whose Voronoi region it belongs to. No other robots will be affected by it. If the single pursuing robot loses contact with the event, its position is predicted, with an added uncertainty. In lack of new sensor readings the peak in the PDF is gradually smoothed and when the pursuer cannot cover all of it due to the limited sensor radius, more robots will be attracted and will help in reaquiring the event. If the event is discovered again, the peak is narrowed again and the closest robot takes over the pursuit. Such a sequence of detection, prediction and reaquisition is illustrated in Figure 7.1.

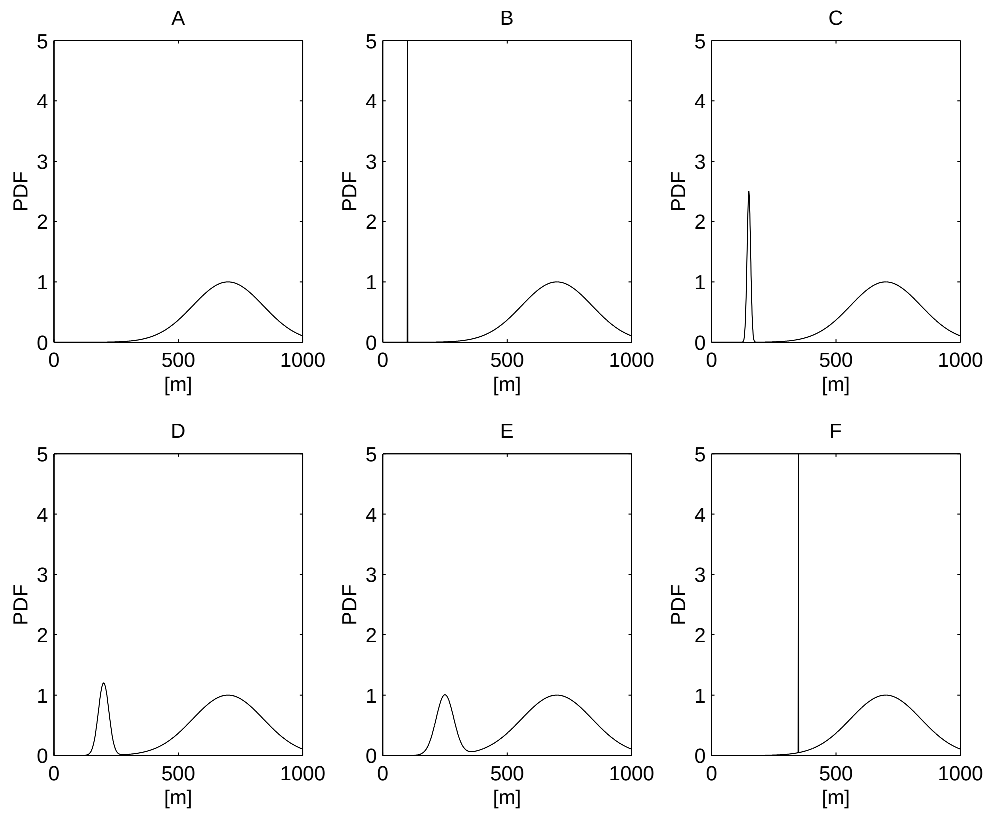


Figure 7.1: A sequence showing in one dimension how the PDF is dynamically updated to reflect the sensor data. Figure A shows the PDF before the robots have detected anything. They will distribute evenly around the peak at $x = 700$m. In B, a robot has detected a target, which generates a peak in the PDF. If there are no more indications, the peak is moved according to the estimated velocity of the target and widened to represent the increasing uncertainty of its position. This is depicted in Figures C-E. Finally, a robot detects the target again in Figure F, which leads to a new peak in the PDF.

Bibliography

[1] M. Lindhé, P. Ögren and K-H. Johansson. Flocking with obstacle avoidance using methods from coverage control. Presented at the *Reglermöte 2004* at Chalmers University of Technology, Sweden, May 26-27, 2004.

[2] P. Ögren and N. E. Leonard. A Convergent Dynamic Window Approach to Obstacle Avoidance. To appear in *IEEE Transactions on Robotics and Automation.*

[3] J. Cortés, S. Martinez, T. Karatas, F. Bullo. Coverage Control for Mobile Sensing Networks. Accepted for publication in *IEEE Trans. on Robotics and Automation.*

[4] J. L. Fuller. Robotics: Introduction, Programming, and Projects. Prentice Hall, 1999.

[5] Ny Teknik. "Arméns nya robotbil kör ökenrally". *http://www.nyteknik.se/art/27529*, last visited on 2004-03-15.

[6] C. R. Weisbin, J. Blitch, D. Lavery, E. Krotkov, C. Shoemaker, L. Matthies and G. Rodriguez. Miniature Robots for Space and Military Missions. *IEEE Robotics & Automation Magazine*, vol. 6, no. 3, pp. 9-18, 1999.

[7] N. Tomatis, I. Nourbakhsh, K. Arras, R. Siegwart. A Hybrid Approach for Robust and Precise Mobile Robot Navigation with Compact Environment Modeling. Proceedings of the 2001 *IEEE International Conference on Robotics & Automation*, Seoul, Korea, May 21-26, 2001.

[8] R. G. Brown and P. Y. C. Hwang. Introduction to random signals and applied Kalman filtering. John Wiley & Sons, 1992.

[9] Datasheet for the Lassen LP GPS receiver module, from Trimble Navigation Limited. *http://www.trimble.com/lassenlp.html*, last visited on 2004-05-24.

[10] Datasheet for the Trimble 5800 RTK GPS surveying system, from Trimble Navigation Limited. *http://www.trimble.com/5800.html*, last visited on 2004-05-24.

[11] R. C. Arkin. Behavior-Based Robotics. Massachusetts Institute of Technology, 1998.

[12] Y. Uny Cao, A. S. Fukunaga and A. B. Kahng. Cooperative Mobile Robotics: Antecedents and Directions. Published in *Autonomous Robots*, volume 4, issue 1, 1997.

[13] E. Rimon and D. Koditschek. Exact Robot Navigation Using Artificial Potential Functions. *IEEE Transactions on Robotics and Automation.* pp. 501-518, Vol. 8, No. 5, October 1992.

[14] J. Hall. A Fast Algorithm for Calculating Shading and Visibility in a Two-Dimensional Field. *http://www.cs.pdx.edu/ idr/graphics/los.html*, last visited on 2004-03-11.

[15] D. Anisi Optimal Motion Control of a Ground Vehicle. Master Thesis Report at the Swedish Defence Research Agency, July 2003. (Report number FOI-R–0961–SE)

[16] P. E. Rybski, N. P. Papanikolopoulos, S. A. Stoeter, D. G. Krantz, K. B. Yesin, M. Gini, R. Voyles, D. F. Hougen, B. Nelson and M. D. Erickson. Enlisting rangers and scouts for reconnaissance and surveillance. *IEEE Robotics & Automation Magazine*, vol. 7, no. 4, pp. 14-24, 2000.

[17] Information on the research on artificial noses by the Walt Group at Tufts University, MA. *http://ase.tufts.edu/chemistry/walt/research/projects/ExplosivesDetectionPage.htm*, last visited on 2004-05-28.

[18] Y. Ikemoto, Y. Hasegawa, T. Fukuda and K. Matsuda. Zipping, Weaving : Control of Vehicle Group Behavior in Non-signalized Intersection. Proceedings of the 2004 *IEEE International Conference on Robotics & Automation*, New Orleans, LA, April 2004,