Contracting Force Fields in Robot Navigation and
Extension to Other Problems
by
Alexandre F. HAAG
Ingenieur de l’Ecole Polytechnique - France
Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of
Master of Science in Mechanical Engineering
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 2000
© Massachusetts Institute of Technology 2000. All rights reserved.

Author .......................................................

Department of Mechanical Engineering
May 5, 2000

Certified by ..................................................

Jean-Jacques E. Slotine
Professor of Mechanical Engineering and Information Sciences
Professor of Brain and Cognitive Sciences
Thesis Supervisor

Accepted by ..................................................

Ain A. Sonin
Chairman, Department Committee on Graduate Students
Contracting Force Fields in Robot Navigation and Extension
to Other Problems

by

Alexandre F. HAAG

Submitted to the Department of Mechanical Engineering
on May 5, 2000, in partial fulfillment of the
requirements for the degree of
Master of Science in Mechanical Engineering

Abstract

This thesis derives results in path planning and obstacles avoidance using force fields.

Based on Contraction Theory, we prove convergence of a simple attractive force
field. Repulsive fields are, in general, not contracting in the Euclidean metric, but we
can define a contracting region when it is combined with another contracting field. We
then provide interesting results about superposition and combination of many force
fields. This theory also enables us to simply extend these convergence properties to
non-autonomous systems such as moving obstacles.

Next, we apply successfully breadth first search, dynamic programming and fast
marching method to create global force (or potential) fields. We compare these three
techniques, explain that there are in fact almost equivalent and present solutions to
improve their computational efficiency. This thesis also addresses the extension to
much more general problems than robotics, especially in optimization, and exposes
an original solution to the Traveling Salesman Problem that runs in \( o(N^{2.5}) \).

Finally, we introduce a new mathematical distance as a theoretical tool to better
understand the structure of the workspace. We present general results that will help
find more theoretical results about superposition of force fields. An efficient algorithm
that uses the sum of previously computed potential fields to build a new one is given.

As an application, we describe an implementation of a concurrent mapping and
localization problem on a simulated Khepera robot. We give some insights on the
problem and in particular we explain how to do very simply a gradient descent with
a real robot on our virtual potential field.

Thesis Supervisor: Jean-Jacques E. Slotine
Title: Professor of Mechanical Engineering and Information Sciences Professor of
Brain and Cognitive Sciences
Acknowledgments

I would like to express my gratitude to my advisor, Professor Jean-Jacques Slotine. He was a good source of support and ideas and he let me the freedom necessary to be inspired by the muses of creativity.

Special thanks go to my very close lab-mate Caroline, with whom I had a lot of interesting discussions on our research problems. My other lab-mates Lutz, Martin, Winnie and Jesse answered many questions I had and I thank them very deeply.

I furthermore appreciate the friendships I have found here: Gwenaelle, Jeff, Jeremy, Maxime, Gilles, Frank, Stephane and many others. They all provided numerous pieces of advice and comments and made this time thoroughly enjoyable.

Finally, I am very grateful to my whole family who supported and encouraged my work.

This work was supported in part by NSF-KDI grant 67562.
Contents

1 Introduction 6

2 Local Obstacle Avoidance 9
  2.1 Attractive Force Field ........................................ 9
  2.2 Repulsive Force Field ........................................ 11
    2.2.1 One, Basic Repulsive Point .......................... 12
    2.2.2 More Complex Repulsive Force Fields ................. 14
    2.2.3 Many Repulsive Points .................................. 17
    2.2.4 Repulsive Region ....................................... 18
    2.2.5 Moving Obstacles ...................................... 19
    2.2.6 Combination ........................................... 19
  2.3 Conclusion .................................................. 22

3 Global Path Planning 23
  3.1 Introduction ................................................ 23
  3.2 Three Different approaches ................................ 24
    3.2.1 Graph Search .......................................... 24
    3.2.2 Dynamic Programming .................................. 25
    3.2.3 Fast Marching Method .................................. 26
  3.3 Comparison ................................................ 27
  3.4 Improvement ................................................. 28
  3.5 Example: the Traveling Salesman Problem ................. 30
  3.6 Link to PDE theory .......................................... 33

4 A New Metric on the Workspace 35
  4.1 Motivation .................................................. 35
  4.2 Definition .................................................. 35
  4.3 Properties .................................................. 36
  4.4 Applications ................................................ 37
    4.4.1 Superposition of potential fields ...................... 37
    4.4.2 Convex functions ...................................... 40
    4.4.3 An efficient algorithm ................................ 40
5 Application : Concurrent Mapping and Localization 42
  5.1 Introduction ................................. 42
  5.2 Application of previous theories ..................... 43
    5.2.1 Exploring Phase ............................. 43
    5.2.2 Repositioning Phase ......................... 44
  5.3 Results ..................................... 45

6 Conclusions and future work 46
Chapter 1

Introduction

By force field we understand a function that to each state assigns a precise control. For a robot moving in the plane, this function can be viewed as a vectorial force, which makes the robot move. Even though we will often use the representation of the robot here, it is important to keep in mind that almost all results may be applied to more general cases like optimization or other graph search problems.

If this force field $F$ has no vorticity, which is usually the case, there exists a scalar function $\phi$ such that its gradient is the desired force field $F$. $\phi$ is called an Artificial Potential Field. Its use is one of the most efficient approaches for online obstacle avoidance to date and has been introduced by [9] and studied by many others. We will not make a big distinction between the scalar and the vectorial approaches in the following and, for example, an equilibrium point will be described either as a minimum of the potential field or as a zero of the force field.

The idea of seeing a control algorithm as a force field in the work -Cartesian- or joint space of the robot is several years old. But some biological experiments have brought new interest in this theory.

Many researchers have suggested a hierarchical control scheme for movement control, based on arguments of bandwidth (there are many sensors and muscles which need a large amount of information to function correctly), delays (nerves are slow, and communicating all that information would result in long delays, with subsequent control problems), and anatomy (the loop from muscles to spinal cord to sensors suggests heavily some low level control modulated by descending signals from the brain). Experiments by Bizzi [3] and others (Giszter [7], Mussa-Ivaldi [15] and Loeb [12]) have attempted to elucidate this hierarchy. They electrically stimulated the spinal cords of frogs and rats and measured the forces obtained at the leg, mapping out a force field in leg-motion space.

Surprisingly, they found that the fields were generally convergent (with a single equilibrium point) and uniform across frogs. The idea is that if the leg is free to move, it will move to the center of the force field, see Giszter [7]. Different fields therefore correspond to different postures.

They have found that the fields can be combined either by superposition -stimulating in two places in the spinal cord resulted in a field which is the linear vector superposition of the fields obtained when each point was simulated, or by "winner-takes-all"
- stimulating in two places and finding the resulting field coming from one of the two places (Mussa-Ivaldi [15]). The force fields remain of similar shape even after the frog is deafferated (Loeb [12]), which strongly suggests that the fields are caused by the interaction of the constant (there is no sensory feedback) spring-like properties of the muscles and the leg kinematics.

These findings lead researchers to suggest that these fields are primitives that are combined to obtain leg motion. Mussa-Ivaldi and Giszter [14] have shown that fairly arbitrary force patterns and so complex motions can be generated using a few of these force field primitives. This theory sounds very attractive to robotics engineers, whose realizations have performances far from those of the animal world.

From the Artificial Intelligence point-of-view now, there are three main reasons to use force fields or motion primitives: it is supposed to be quite simple to implement, computationally efficient and as powerful as animal motions. Furthermore, since it is working in real world on animal, we can expect that it will work on robots! The problem remains to find out more precisely how to use force fields and how to prove their convergence and stability.

From a control point-of-view, force field theory can be seen as a convenient and efficient way to plot and visualize the control policy at each state point.

Contraction theory has been introduced by Lohmiller and Slotine [13] recently. Intuitively, contraction analysis is based on a slightly different view of what stability is, inspired by fluid mechanics. Regardless of the exact technical form in which it is defined, stability is generally viewed relative to some nominal motion or equilibrium point. Contraction analysis is motivated by the elementary remark that talking about stability does not require one to know what the nominal motion is: intuitively, a system is stable in some region if initial conditions or temporary disturbances are somehow forgotten, i.e., if the final behavior of the system is independent of the initial conditions. All trajectories then converge to the nominal motion. In turn, this shows that stability can be analyzed differentially - do nearby trajectories converge to one another?- rather than through finding some implicit motion integral as in Lyapunov theory, or through some global state transformation as in feedback linearization. Not surprisingly such differential analysis turns out to be significantly simpler than its integral counterpart.

To solve the obstacle avoidance problem, we start by presenting basic calculations on force fields (chapter 2). Our aim in this chapter is to prove what kind of attractive and repulsive force fields are contracting and how we can use them for more complex problems (multiple obstacles, non-punctual obstacles and time varying environment). All these results apply more at the low or local level, in cases where an entire path is already planned at the global level. This is actually what chapter 3 is about: global path planning. We first present existing techniques to find efficiently a trajectory and the shortest path from one point to another in a maze or in more general problems. These techniques are Breadth First Search (BFS), Dynamic Programming (DP) and Fast Marching Methods (FMM). We show that they are almost equivalent and that they still can be improved. In particular we apply this improvement at the end of this chapter to an original modeling and efficient solving of the Traveling Salesman Problem. This is an algorithm that finds an exact solution of problem up to 26 cities.
in less than a minute. Chapter 4 introduces a new metric on the workspace and tries to build new mathematical tools useful for convergence and stability analysis. It derives general theoretical results and also presents a new fast algorithm to find a path from one point to another in a maze, by reusing the computation done before to plan other trajectories. We end this thesis with an application of some of the previous results: a concurrent mapping and localization problem is implemented on a simulated Khepera robot. The robot is exploring randomly its unknown environment, building an internal map at the same time and then using it to navigate to a desired position.
Chapter 2

Local Obstacle Avoidance

Obstacle avoidance is the problem of designing a controller that safely navigates the robot around forbidden points to avoid collisions. By local, we mean that the results presented in this chapter apply more to the cases where the main goal of the robot is to go to one “close” point (the attractive point), avoiding a few obstacles (the repulsive points). For example a really complicated maze will be better solved using other techniques like the ones presented in chapter 3.

Here, we will show that the superposition of simple radial force fields works very well and is a very simple way to design control functions. Based on a new method to analyze stability (contraction theory [13]), we perform a separate study of different fields. Next, we propose different applications of the superposition of these fields: multiple obstacles and repulsive regions. Furthermore, the contraction theory and the use of the variable \( s = x + T \dot{x} \) enable us to derive very easily similar results for non-autonomous systems or moving obstacles.

The work presented here is done for a mass-point robot in a Cartesian space. But as long as its equation of the dynamic is similar to (2.2), another system can be solved with the same approach. For example, manipulator robots could use those techniques for their end effectors and one of the obstacles would be the arm of the robot itself. The calculations are usually done in two dimensions but we show how to extend the results for n-dimensional problems.

2.1 Attractive Force Field

An attractive point is used to set the goal the robot has to reach.

We present first the simplest case: there is only one attractive point and no obstacles in the environment (at least locally). Moreover we consider the robot as a mass point.

Let us first introduce a new variable, \( s \), which acts as a prediction of the position of the robot \( T \) seconds further, by including the velocity:

\[
s = x + T \cdot \dot{x}
\]

where \( T > 0 \), \( x \) is the position vector (possibly in joint space) and \( \dot{x} \) the velocity.
Figure 2-1: An attractive force field centered in (0.5, 0.5) with two initial vectors \( s \) and corresponding trajectories followed by the robot, according to equations (2.1), (2.2) and (2.3). Here \( T = 1 \).

\( T \) can also be seen as a factor that changes the relative importance of damping and stiffness: the higher \( T \), the higher the damping in the motion. There are many benefits in using \( s \). First we manipulate only a single variable instead of both the position and the velocity and moreover this will enable us to transform a second order system into a first order system. Secondly, the damping is embedded deeper into the control equations. Third, it is easier to use in contraction analysis. The introduction of the \( s \) variable is also related to the Sliding Control Theory [20].

We then have the fundamental equation of dynamic:

\[
F(x) = m \cdot \ddot{x} \tag{2.2}
\]

where \( F(x) \) is the force field created to control the robot and \( m \) the mass of the robot.

Finally, we can chose the control \( F(x) \) applied to the system by another vectorial function \( f(s) \):

\[
F(x) = \frac{m}{T} \cdot (f(s) - \dot{x}) \tag{2.3}
\]

Then, by differentiating (2.1) and replacing with (2.2) and (2.3), we get:

\[
\dot{s} = \dot{x} + T \ddot{x} = f(s)
\]

which is the usual form for a control equation. It is also a first order differential equation. This means that \( s \) is controlled by a velocity field instead of a force field for \( x \). It is thus much easier to visualize the behavior of the variable \( s \): it just follows the stream lines. For example, on figures (2-1), (2-2) and (2-3), the trajectories of \( s \) are just straight lines from the starting point towards the attracting point or away from the repulsive point.
From there, we can apply the contraction theory \cite{13}, which states that the previous system is contracting if:

\[
\frac{1}{2} (\frac{\partial f}{\partial s} + \frac{\partial f^T}{\partial s}) \leq -\beta I < 0
\]

For an attractive point, we chose: (the size of the square matrix $K$ is the number of dimensions of the space)

\[
f(s) = \begin{pmatrix}
k_1 & 0 & \vdots \\
0 & k_2 & \vdots \\
\vdots & \vdots & \ddots
\end{pmatrix} (s_0 - s) = K(s_0 - s)
\] (2.4)

If $k_1, k_2, ... > \beta > 0$, then the system is contracting and every trajectory $s(t)$ tends to $s_0$. This means that two different trajectories with initial conditions $s_1(0) = s_{01}$ and $s_2(0) = s_{02}$ will converge to the same trajectory $s_0(t)$.

Now we can go back from the $s$ state space to the $(x, \dot{x})$ state space. By looking at the definition of $s$, we see that $x$ is defined by a first order differential equation $x + T\dot{x} = s(t)$, which implies exponential convergence to a particular solution of the differential equation. Thus two trajectories in $(x, \dot{x})$ with two different initial conditions, will converge exponentially to two particular solutions that converge to the same solution (i.e. the solution of $x + T\dot{x} = s_0(t)$), which implies that the system has also a contracting behavior in $x$.

Another important question is how to choose the value of $T$. Basically, we do not want $s$ to be “too far” from $x$, so that the behavior of the real robot ($x$) is not too far from the controlled variable, $s$; but at the same time we want $T$ not too small to use the prediction effect of $s$. What we mean by “too far” depends of course on the scale of the problem: if the distances in the environment are small or if everything is moving fast, $T$ has to be small. For example, in our simulations we used $T = \frac{1}{2}$, in a $2\times2$ environment, with obstacles moving at a speed of 0.5.

For a basic attractive force field as in figure (2-1), $s_0$ is just defined by $s_0 = x_0$, the center of the field. But the above demonstration is also valid for non autonomous systems where the center of the force field is moving, that is where $s$ depends on $t$.

Since the contraction is conserved by addition, the sum of different force fields, contracting within the same metrics, is still a contracting field. In particular, we can add two attractive force fields and get a new attractive force field centered elsewhere. We do not know whether the force fields discovered in the frog are mathematically contracting, but motion primitives are a typical application of the previous result.

### 2.2 Repulsive Force Field

Repulsive points represent the obstacles the robot has to avoid. As it will appear in this section, repulsive force fields are much more complex than attractive ones: contraction is not straightforward and they can create local minima in the field.
2.2.1 One, Basic Repulsive Point

We assume the repulsive point is at 0. For a repulsive force field centered at another point, we may just consider \( s - s_0 \) instead of \( s \).

At the difference of the attractive case, the repulsive force field should have very low influence far from the center. The first function that comes to mind is something linear and as simple as \( f(s) = \begin{cases} F_0 - k||s|| & \text{for } ||s|| < \frac{F_0}{k} \\ 0 & \text{otherwise} \end{cases} \). There is a problem with this though. It is due to the fact that the control function depend on \( s \) and not directly on the position. This means that we apply to the robot the force corresponding not to where it is but to where the vector \( s \) points. Thus, for example, if the robot is on one side of an obstacle and \( s \) already points across the obstacle to the other side, the field will attract the robot towards the repulsive obstacle! To avoid that, we assume that this is not the case at time 0 and need to set the force to infinity at the center of the obstacle to ensure that the robot will always be sufficiently slowed down when approaching an obstacle\(^1\). The 1/r function is then a logical continuous function to use.

We still use the same definition of \( s \) as in (2.1) and the same equation for the control (2.3) as in previous section. Then if we choose the basic radial repulsive field \( f(s) = \frac{s}{||s||^2} \), the derivative of \( f \) is (in 2D, where \( s_x \) and \( s_y \) are the two components of \( s \)):

\[
\frac{\partial f}{\partial s} = \frac{1}{||s||^4} \begin{pmatrix} s_x^2 - s_y^2 & -2s_x s_y \\ -2s_x s_y & s_x^2 - s_y^2 \end{pmatrix}
\]

which is not definite negative. Therefore we cannot conclude to the contraction of such a field. It is actually obvious on a plot-out that a basic repulsive force field is not contracting and that any trajectory is going away from any other.

Even though, if we make a metric change and look at the same trajectories in polar coordinates (figure 2-3), we see that the trajectories are “not too far” from being contracting. Actually the trajectories of the \( s \) variable are horizontal straight lines, which stay at the same distance in \( \theta \) from each other and tends to the same \( r \). They do not contract to a single trajectory, but they also do not diverge from each other. Therefore, we had the idea that the presented repulsive force field is “almost” contracting and that when added to a strongly enough contracting field, the sum remains contracting.

To prove this mathematically, we look at the eigenvalues of the symmetric part of \( f \) ’s Jacobian, \( J_{rep} \). These two eigenvalues are \( \lambda_{rep} = \pm \frac{1}{||s||^2} \). If we do not consider the region too close to the repulsive point, \( \lambda_{rep} \) is upper bounded on the domain by, let’s say, \( \lambda_{rep\text{max}} \). A contracting force field is characterized by a symmetric part of it’s Jacobian, \( J_{contr} \), which is definite negative. Then \( J_{contr} \) is diagonalizable and its biggest eigenvalue \( \lambda_{contr\text{max}} \) is negative. To verify if the sum of the two fields is contracting, we check whether the sum of the symmetric parts of the Jacobians, expressed in the same metric, is definite negative. We have:

\(^1\)For a real robot, the force can not be infinite and it is important to change appropriately the controller in case \( s \) would point to the other side of the repulsive point.
Figure 2-2: basic radial repulsive force field with two initial vectors $s_1$ and $s_2$ and the corresponding trajectories followed by the robot.

Figure 2-3: Same as figure (2-2) but in polar coordinates. Note that the trajectories for $s$ are horizontal straight lines.
Figure 2-4: The two trajectories "contract" to the same point on this force field consisting of a basic attractive field and a basic repulsive field.

\[
\forall x \neq 0, \quad x^T J_{\text{contr}} x < \lambda_{\text{contr max}} \|x\|^2 < 0
\]
\[
\forall x \neq 0, \quad x^T J_{\text{rep}} x < \lambda_{\text{rep max}} \|x\|^2 < \|\lambda_{\text{rep}}\| \|x\|^2
\]
thus, \[
\forall x \neq 0, \quad x^T (J_{\text{rep}} + J_{\text{contr}}) x < (\lambda_{\text{rep max}} + \lambda_{\text{contr max}}) \|x\|^2
\]

Then a sufficient condition for the system to be contracting is

\[
(\lambda_{\text{rep max}} + \lambda_{\text{contr max}}) < 0.
\]
This can be achieved either by strengthening the contracting force field (more negative \(\lambda_{\text{contr max}}\)) or weakening the repulsive force field (smaller \(\lambda_{\text{rep max}}\)) or increasing the size of the excluded domain around the repulsive point. In the last case however, if the contracting field is an attractive one, it is better not to exclude this attractive point from the contracting domain. It is also interesting to note that the region excluded could correspond to the real size of the non-punctual obstacle. Finally, since the non-contracting region is repulsive and bounded, any trajectory starting in this region is going to its boundary and then to the contracting domain; and this also guarantees that any trajectory starting in the contracting domain will end up in it. Figure 2-5 shows the importance of this second condition; the trajectory can go out of the contracting domain, in which case the distance between two trajectories is not necessarily decreasing, but it will return to it after a finite time. Thus every trajectory ends up at the desired attractive point.

This result is confirmed by simulations and can be viewed on figure 2-4.

2.2.2 More Complex Repulsive Force Fields

First Note that the equilibrium point does not correspond exactly to the center of the attractive force field because the repulsive field has still a small influence at this point. Thus once a target point and an obstacle are defined, the equation \(f_{\text{attr}}(x) = f_{\text{rep}}(x)\) must be solved to find the position \(x\) at which the attractive point has to be set.

A simple way to avoid that is to continuously cancel the repulsive force field outside
Figure 2-5: The top plot shows the trajectory in \((x,y)\) coordinates; the middle plot is in the \(s = (s_x, s_y)\) space and the bottom one is the distance between the two trajectories over time. One can see that when one of the trajectories is leaving the contracting domain (outside the circle), the distance can increase.
a given ball of influence. We can, for instance use the function \( f(s) = \max(\frac{s}{||s||^2} - \frac{1}{R_0}, 0) \). The eigenvalues of the symmetric part of the Jacobian of \( f \) are, inside the ball of radius \( R_0 \), the same as in the previous subsection \( (\lambda_{rep} = \pm \frac{1}{||s||^2}) \) and are null outside. We could construct a function that would have the same properties and that is moreover smooth but this does not seem to be necessary.

Another thing we want to take into account is the size of the obstacle. Since in real world, obstacles are not points, we propose the following function: \( f(s) = \frac{s}{||s||^2 - r_0} \), \( ||s|| > r_0 \), which is explained in more details in the section “Repulsive Region”.

Also, It is easy to verify that this result is valid for an N-dimensional space. In fact, the eigenvalues of the Jacobian are still \( \lambda_{rep} = \pm \frac{1}{||s||^2} \), the positive one being a (N-1)-order eigenvalue.

Finally, if we change the reference frame and use the cylindrical coordinates \((r, \theta)\), we have the new expressions of the different equations:

\[
s_{r\theta} = M \cdot s_{xy} = \begin{pmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{pmatrix} \cdot \begin{pmatrix} s_x \\ s_y \end{pmatrix}
\]

expressed on \( r, \theta \): \( s_{r\theta} = \begin{pmatrix} r + T\dot{r} \\ T\dot{\theta} \end{pmatrix} \), and thus the derivative, \( \dot{s_{r\theta}} = T\left( \begin{pmatrix} \frac{1}{T}\dot{r} + \dot{\theta} \\ \frac{r}{T}\dot{\theta} + \dot{r} \end{pmatrix} \right) \)

the fundamental equation of dynamic in cylindrical coordinates becomes:

\[
F_{r\theta} = m \begin{pmatrix} \dot{r} - r\dot{\theta}^2 \\ 2\dot{r}\dot{\theta} + r\ddot{\theta} \end{pmatrix}
\]

now it appears that if for the control, we use:

\[
F_{r\theta} = \frac{m}{T} f(s) - m \begin{pmatrix} \frac{1}{T}\dot{r} + r\dot{\theta}^2 \\ -\dot{r}\dot{\theta} \end{pmatrix}
\]

then combining these equations leads to the equation of the system is: \( \dot{s_{r\theta}} = f_{r\theta}(s_{r\theta}) \), which without subscript is the classical form for a control equation:

\[
\dot{s} = \dot{x} + T\ddot{x} = f(s)
\]

Now, let’s assume

\[
f_{r\theta} = \begin{pmatrix} \frac{1}{s_r} \\ -\epsilon s_{\theta} \end{pmatrix}
\]

where \( \epsilon > 0 \). we choose \( \epsilon \) very small so that it is significant from a mathematical point of view but negligible in practice.

Then \( f_{r\theta} \) is contracting and the system is contracting for \( \theta \neq \pi \) and the use of \( \frac{1}{r} \) means the field is repulsive. By adding another term \( A = \begin{pmatrix} \frac{1}{r}\dot{r} + r\dot{\theta}^2 \\ -\dot{r}\dot{\theta} \end{pmatrix} \) in the control equation (2.6), we have constructed a contracting repulsive force field. Unfortunately,
Figure 2-6: Existence of a local minimum in front of a group of three obstacles: The trajectory whose initial $s$ is in the non-contracting domain ends up in a local minimum.

this metric is not really practical when there are other force fields centered elsewhere. Indeed the trajectories in a basic attractive force field (straight lines in Cartesian space) are $acos$ functions in the $(r, \theta)$ metric centered at the repulsive point: there are not contracting in the whole space.

### 2.2.3 Many Repulsive Points

To represent other obstacles in the environment, we add other repulsive force fields. Almost as before, they will be described by the function $f = (\frac{s-s_i}{\|s-s_i\|^2})$, where $s_i$ is the center of the repulsive force field number $i$. The eigenvalues of the corresponding Jacobian are $\lambda_{rep,i} = \pm \frac{1}{\|s-s_i\|^2}$. The system is then contracting in a given domain if we have at each point inside this domain: $(\lambda_{contr} + \sum \lambda_{rep,i}) < 0$. This equation defines the contracting domain and as in the previous section, it can be modified by strengthening the contracting force field (more negative $\lambda_{contr \ max}$) or weakening the repulsive force field (smaller $\lambda_{rep}$).

The main problem with multiple force fields is the possible existence of local minima different from the global minimum (figure 2-6). But in our case, the previous proof ensure us that any local minimum is outside the contracting domain, because there can be only one equilibrium point in a contracting domain. Therefore if the robot starts in the contracting domain (which is easy to check) and it remains in it then it will go to the attractive point. In this case, the difficulty is to predict whether the robot will remain in this contracting domain or not. However, we know that if the robot starts in a ball centered at the equilibrium point and included in the contracting domain, it will always remain in this ball and contract to the equilibrium point.

To avoid local minima, there also exist the solution of using harmonic functions as potential field. Harmonic functions are functions whose second derivative verifies
the Laplace equation $\nabla^2 \phi = 0$. See [6][4][10], for instance, for more precise results on harmonic functions. Their use comes from the fluid analogy and they have the fundamental properties that first the sum of two harmonic functions is still an harmonic function and secondly that they achieve their extremum only on the boundary of objects. In two dimensions, $\phi(X) = k \log(\sqrt{x^2 + y^2})$ is the most commonly used harmonic function (\(k\) is positive for an attractive potential field and negative for a repulsive one). We can see that the repulsive force fields we used before are harmonic: $\text{grad} \phi = 1/r$. Unfortunately the attractive force field presented at the beginning of the chapter (equation (2.4)) is not harmonic, thus we cannot conclude anything about their sum. If we use for the attractive force field $f(s) = \frac{s}{||s||^3}$ the sum is a harmonic potential, but this field is not contracting. This is easily proved by remarking that the trace of the Jacobian of any Harmonic field is null (by definition) and by the fact that changing the metric does not change the trace (if the metric change does not depend on time). Thus at least one of the eigenvalues of the Jacobian is positive or null.

In an n-dimensional space a typical harmonic potential is $\phi(X) = k \left( \sum \frac{1}{x_i^2} \right)^{n/2}$. These kinds of fields are a bit different from the $\frac{1}{x^2}$ fields proposed in the previous subsection, but their greatest eigenvalues are still bounded by $\left( \sum \frac{1}{x_i^2} \right)^{n/2}$ and thus we can apply the same proof to get the contraction property.

### 2.2.4 Repulsive Region

One idea here is to use many repulsive points to model a repulsive region instead of just repulsive points. The problem is to define where the robot can go between two points and where it can't because it is supposed to be the same obstacle. In other words we have to find out how many points are needed to represent one obstacle. By the way, we also have to check that if two obstacles are close but not enough to block the robot, this representation of force fields will allow the controller to drive the robot between these two obstacles. Figure 2-7 shows the kind of problems we may encounter when there are two repulsive points. It also reminds us that when using the $s$ variable, it has to point outside a forbidden domain at the beginning.
Another solution is to use functions that are infinite on the border of the obstacle, so that the robot can never get inside the region defined by the obstacle. For a circular obstacle, a good function will be \( f(s) = \frac{s - s_0}{\|s - s_0\|^2 - R^2} \) where \( s_0 \) is the center and \( R \) the radius of the obstacle. In the same way as in the previous section, we can show that this field is contracting when added with an attractive force field, except in a disk around the obstacle. The radius of this disk is the same as previously plus \( R \). This is easily shown by noting that the eigenvalues of the Jacobian are \( \lambda_{rep} = \frac{1}{\|s - s_0\|^2 - R^2}, -\frac{\|s - s_0\|^2 + R^2}{\|s - s_0\|^2 - R^2} \), that the second one is always negative and that the first one is bounded outside a disk of radius greater than \( R \).

### 2.2.5 Moving Obstacles

This is one big advantage of contraction. Indeed the contraction theory tells us that if a non-autonomous system is contracting at every time then it is also contracting over time. This comes from the fact that we are only looking at the Jacobian, which contains no derivative over time. Thus all the previous results are valid if the attractive point is moving (interception problem or movement defined by a global planner) and/or if the repulsive points are moving (the obstacles may represent people walking on the sidewalk and the robot has to avoid them). Since the velocity of the attractive or repulsive point is embedded in \( s \), it does not change anything in the equations! Moreover the center of a repulsive or attractive field is in \( s_0 = x_0 + T\dot{x}_0 \), which is not where the obstacle or the target is now, \( x_0 \), but where it will be in \( T \) sec. Thus this simple controller naturally looks ahead to detect a possible collision and avoid it.

Figure 2-8 shows different shots of trajectories of robots avoiding a moving obstacle.

Since the non-contracting region is also moving with the obstacle, it may be more difficult to check that the robot does not stay in it. But for example, in the case where there is only one repulsive point and one attractive point, there is still no other stable equilibrium point than the attractive one.

### 2.2.6 Combination

Now that we have enough material on repulsive and contracting force fields, we can address the issue of choosing \( T \) exposed in section 2.1. The problem is that \( T \) defines for how far in time the prediction is made and is chosen more or less arbitrarily. Or for one given \( T \), the robot may not see an obstacle that appears very close at the last moment or may “forget” an obstacle, if \( T \) is too high. On the other hand, if \( T \) is too low, the prediction is not enough ahead in time and the robot may detect obvious collisions only at the last moment. Thus it would be useful to be able to use different variables \( s_i \) corresponding to different time \( T_i \). Let us do it for \( i = 2 \). We have then the two following contracting systems:

\[
\begin{align*}
\dot{s}_1 &= f_1(s_1) \\
\dot{s}_2 &= f_2(s_2)
\end{align*}
\]
Figure 2-8: Read from left to right and top to bottom to see two different trajectories of a robot trying to avoid the obstacle represented by the small circle. The two trajectories are very close at the beginning but one “prefers” to slow down at go behind the obstacle.
where $s_i = x + T_i \ddot{x}$ and where $f$ is the repulsive force field (for example an attractive plus a repulsive force fields as defined previously). We want to combine them and define a new equation for the system:

\[
\frac{\dot{s}_1 + \dot{s}_2}{2} = \dot{x} + \frac{T_1 + T_2}{2} = \frac{f_1(s_1) + f_2(s_2)}{2}
\]  

which, plugged into the equation of dynamic (2.2), gives the equation of control to apply to the robot:

\[
F(x) = m\ddot{x} = \frac{2}{T_1 + T_2} \left( \frac{f_1(s_1) + f_2(s_2)}{2} - \dot{x} \right) = \frac{2}{T_1 + T_2} f(s)
\]  

Thus, we can see the system (2.7) as :

\[
\begin{align*}
\dot{s} &= s_1 + s_2 \\
\dot{s} &= f_1(s_1) + f_2(s_2)
\end{align*}
\]

We want to guarantee the contraction of the system when both $s_1$ and $s_2$ are in the contracting domain of $f_1$ and $f_2$. To do that we express the system as the combination of :

\[
\begin{align*}
\dot{s}_1 &= \dot{x} + T_1 \ddot{x} = \frac{s_1}{T_1 + T_2} + \frac{T_1}{T_1 + T_2} (f_1(s_1) + f_2(s_2)) = g_1(s_1, s_2, t) \\
\dot{s}_2 &= \dot{x} + T_2 \ddot{x} = \frac{s_2}{T_1 + T_2} + \frac{T_2}{T_1 + T_2} (f_1(s_1) + f_2(s_2)) = g_2(s_1, s_2, t)
\end{align*}
\]

By differentiation, we get:

\[
\begin{align*}
\dot{\delta s}_1 &= \frac{\delta s_1 - \delta x}{T_1 + T_2} + \frac{T_1}{T_1 + T_2} \left( \frac{\partial f_1}{\delta s_1} \delta s_1 + \frac{\partial f_2}{\delta s_2} \delta s_2 \right) \\
\dot{\delta s}_2 &= \frac{\delta s_2 - \delta x}{T_1 + T_2} + \frac{T_2}{T_1 + T_2} \left( \frac{\partial f_1}{\delta s_1} \delta s_1 + \frac{\partial f_2}{\delta s_2} \delta s_2 \right)
\end{align*}
\]

which can be written:

\[
\begin{pmatrix}
\dot{\delta s}_1 \\
\dot{\delta s}_2
\end{pmatrix} = \frac{1}{T_1 + T_2} \begin{pmatrix}
-I + T_1 \frac{\partial f_1}{\delta s_1} & I + T_1 \frac{\partial f_2}{\delta s_2} \\
-I + T_2 \frac{\partial f_1}{\delta s_1} & I + T_2 \frac{\partial f_2}{\delta s_2}
\end{pmatrix} \begin{pmatrix}
\delta s_1 \\
\delta s_2
\end{pmatrix} = J \begin{pmatrix}
\delta s_1 \\
\delta s_2
\end{pmatrix}
\]

Note that, in general, this is a block matrix equation if the dimension $N$ of $s$ is greater than one, which is usually the case. The $1$’s are actually identity matrices for $N > 1$.

But let us first consider the case where $s$ is a scalar. The symmetric part of $J$ has

\[
\frac{1}{2(T_1 + T_2)} \left[ T_1 \frac{\partial f_1}{\delta s_1} + T_2 \frac{\partial f_2}{\delta s_2} - 2 \right] \pm \sqrt{\left( T_2 \frac{\partial f_1}{\delta s_1} + 2 \right)^2 + \left( T_1 \frac{\partial f_2}{\delta s_2} + 2 \right)^2 + T_1^2 \frac{\partial f_1^2}{\delta s_1} + T_2^2 \frac{\partial f_2^2}{\delta s_2} - 4}
\]

for eigenvalues. Since a real symmetric matrix is always diagonalizable, we are guaranteed that the term under the square root is positive. However it is much harder to prove that the two eigenvalues are negative. A 3D plot of the function $\lambda(\frac{\partial f_1}{\delta s_1}, \frac{\partial f_2}{\delta s_2})$ shows that $\lambda$ is only positive when the two derivatives are very different from each
other or when one of them is positive, in all other cases, which are also the most common, the greatest eigenvalue is negative and thus the system is contracting. In the other cases, we will need to find a new metric in which the system might be contracting (so far, we have not find simple metric that works).

For the general N-dimensional case, we have to consider $\mathbf{J}$ as a block matrix and the eigenvalues as matrices. These depends on the Jacobians $\frac{\partial f}{\partial \mathbf{x}}$ and their transposes. Then the same base change matrix expanded by block from 2x2 to 2Nx2N will transform the symmetric part of $\mathbf{J}$ into
\[
\begin{pmatrix}
\lambda_1 & 0 \\
0 & \lambda_2
\end{pmatrix},
\]
diagonal per block. When both $\lambda_1$ and $\lambda_2$ are definite negative so is $\frac{\mathbf{J} + \mathbf{J}^T}{2}$ and the system is contracting.

2.3 Conclusion

Force field is a very efficient, useful and practical method for environment with few obstacles or when looking only at a region that has few obstacles (local problem). Moreover, with the contraction analysis of these force fields it is straight forward to derive similar convergence results for non-autonomous systems. Since the attractive point can move too, force fields could serve as a low-level controller that follows a defined trajectory. But to plan this defined trajectory or to control a robot in a bigger, more complicated environment, we need, to avoid local minima, to have a global approach and to use other techniques, which are presented in the next chapter.
Chapter 3

Global Path Planning

3.1 Introduction

This chapter starts with a presentation of three different efficient methods used in path planning, namely Breadth-First Search, Dynamic Programming and Fast Marching Method. Then we compare them and show small advantages of each. But the main result of the first sections is that these three algorithms are actually almost identical and that their differences reside only in details of implementation and in their representations. This allow us to keep only one representation, which is almost that of Fast Marching Method: a potential field over the state space, which gradient can be viewed as a force field and that has only one minimum. The next section proposes an improvement of these algorithms, which is particularly interesting when the dimensionality of the problem is high. As an application of this remark, we solve the Traveling Salesman Problem using an original modeling, viewing the state space as an hypercube.

In order to have a concrete problem to talk about, we will present here the problem of a robot in a maze trying to find the shortest path from one point to another (figure 3-1). The maze is defined by a grid with obstacles (0 if free space or 1 if there is an obstacle at that point) or with costs to go to the next grid point (positive real numbers defining the cost at each point). When talking about mazes, we mean general ones, not those that have only one solution to go from the entry to the exit (those problem have little interest!). There are two main differences with chapter 2: first the problem is now a discrete one and second, the force defined by the field does not exactly correspond to the one that could appear in an equation of dynamic like (2.2) but more to a direction to the next state. But of course it is fundamental to keep in mind that the interest of this theory is that it can be extended to virtually every problem that can be transformed into a graph on which the best decisions have to be made at each successive state.

For example, if considering the previous problem as a dynamic problem (e.g. controlling the acceleration and not the displacement at each step), the velocity must be introduced in the state. Therefore a state is no longer defined as a position \((x, y)\) but as a 4D state \((x, y, v_x, v_y)\). It is a bit harder to represent and a slice of the state
space (figure 3-2) shows that the metric is all but Cartesian: two neighbor points in the sense of this problem (two states that can be successive) are not necessarily neighbors in the common sense. But we can still imagine a “robot” searching a shortest path in this 4D maze. We can also introduce the time as a 5th dimension, which enables to have a time dependent environment: moving obstacles are just as easy, if we know in advance how they will be moving. Another possible extension is the introduction of the probability. In this case the maze is not represented by zeros and ones whether there is an obstacle or there is not one, but by any number between zero and one representing the probability that there is an obstacle. These numbers will then be used as costs for the algorithms.

### 3.2 Three Different approaches

The aim of this section is to show that three different, well-known types of algorithms, with three different origins and representations have actually the same fundamental structure (i.e. the same order of computational time, same way of going through every point on the grid, only explanation and details on implementation are different).

#### 3.2.1 Graph Search

The breadth-first search explores a graph $G=(V,E)$ across the breadth of the frontier between discovered and undiscovered nodes. The later a node is examined the more distant it is from the starting node $s$. 

---

Figure 3-1: The shortest path from (1,4) to (5,8).
That is, the algorithm discovers all nodes at distance $k$ from the starting node $s$ before discovering any nodes at distance $k+1$.

As a result one gets the shortest paths to each node $v$ of $V$ that is reachable from $s$. A shortest path from $s$ to $v$ is in this case the path which consists of the lowest number of edges.

To describe the current state of the breadth-first search every node can be colored white, gray or black. At the beginning all nodes are white. When a node gets visited for the first time it gets colored non-white. That is, all gray and black nodes have been visited at least once.

An already visited node gets the color black, if all its child nodes have already been visited (which means that it is colored non-white). On the other hand, an already visited node gets the color gray, if it exists at least one child node, which is not visited yet (which means that it is white).

Therefore the gray nodes describe the frontier between the visited and not yet visited nodes. This representation with color is very useful to explain the similarity with the fast-marching method.

In the case of weighted graphs, the corresponding algorithm is known as Dijkstra’s algorithm. It uses a heap of the vertices, ordered by distance, instead of a set of discovered vertices.

3.2.2 Dynamic Programming

Dynamic Programming is a technique for efficiently computing recurrences by storing partial results. It is an application of the principle of optimality (whatever the initial state is, remaining decisions must be optimal regardless the state following

Figure 3-2: This is a cut of the 4D space $(x, y, v_x, v_y)$ corresponding to the maze problem with dynamic. It shows the neighbors for three points.
from the first decision). See [2] for more information on dynamic programming in control theory.

It is derived from the Hamilton-Jacobi-Bellman Partial Differential Equation:

\[ 0 = \min_{u \in \mathcal{U}} [g(x, u) + \nabla_t J^*(t, x) + \nabla_x J^*(t, x)' f(x, u)] \text{ for all } t, x \]

where \( x \) is the state, \( u \) the control, \( t \) the time, \( J^*(t, x) \) the cost-to-go function, \( f(x, u) \) the transition function, \( g(x, u) \) the cost function, \( \nabla_t \) the partial derivative with respect to \( t \) and \( \nabla_x \) the partial derivative with respect to \( x \).

Usually the computation starts from the end (from the target point) and finds backwards the optimal decision at time \( n-k \) so that the path from \( n-k \) to \( n \) remains optimal. But if the problem is symmetric (in the sense it is as easy to find all the points that can be reached from a given point as to find all the points from which a given point can be reached), dynamic programming can be use in the same direction as BFS. But apart this difference in direction, DP visits the points on the graph in the same order as BFS: from the end point it goes to all points where the robot could have been one step before, that is all points for which the endpoint is a neighbor, when it has calculated the cost for these points, it looks at all points where the robot could have been 2 steps before the end etc...

### 3.2.3 Fast Marching Method

The Fast Marching Method is part of the Theory of Curve and Surface Evolution developed recently by J.A. Sethian [19]. The idea here is to simulate the propagation of a wave from the endpoint through the entire state space (the propagation speed depending on the cost of going from one point to another and ignoring reflection and dispersion effects) and to record the time \( T(x, y) \) at which the wave pass at each point. These times \( T \) represent the shortest time (or length or smallest cost) to go from the endpoint to the particular point \( (x, y) \). \( T \) is a potential Field and a steepest gradient descent on it gives the shortest path to go to the chosen endpoint. Figure 3-3 shows a plot of this potential field on the same maze as in figure 3-1; it is easy to imagine how the wave has propagated from the target point, between obstacles, in the whole maze. By construction this field has only one minimum: the parent of a node has always a smallest value and only the endpoint has no parent (reminder: we assume that there is no negative cost).

This algorithm is based on the Eikonal Equation:

\[ |\nabla T| = F(x,y) \text{ where } T \text{ is the time of arrival of the wave and } F \text{ is the cost (or the inverse of the celerity) at point } (x,y). \]

The approach and the representation are very different from BFS and DP, but by looking into more details we can see that the simulation of the wave is done exactly in the same way: from the endpoint the algorithm measures the time to go to all neighbor points, then from all these new points, it looks one step further, finds the neighbors and calculates the time to go the next points. A good link between the two theories is to see the algorithm as an application of the Huygens principle in optic: at

---

1This \( T \) has nothing to do with the one in the definition of \( s \) in chapter 2 except the unit (time).
each step we consider the point being treated as a new light source (or wave source) propagating to its neighborhood.

### 3.3 Comparison

First of all it is interesting to note that virtually nobody mentioned that these techniques are so similar and that the structure of the computation done as an implementation of those algorithms is the same. In each case, the computer will look almost in the same order at the same points and their corresponding costs.

The main difference is the way intermediate results are stored during the process. In the BFS case the program stores for each vertices (or node or point explored) the parent it comes from, whereas in the two other algorithms, only the number of steps or the cost to go to this particular point is stored. This difference has three implications. First, a small advantage for BFS is that once you have gone through the whole graph, the time to explicitly output the shortest path is in $o(1)$ at each step. For the DP and the FMM this time is in $o(D)$ where $D$ is the number of possible predecessors. But this is still very fast and it only represents a small fraction of the total cost, which comes from the backwards computation. Second, in BFS we said that the parent is stored for each vertices, that is a number between 0 and the total number of parents, let’s say $o(n^D)$ where $D$ is the dimension of the graph and $n$ is the diameter of the graph. For the two other cases however only the number of steps is stored, which is a number between 0 and the maximum length, $o(n)$. The space required to store a number is on the order of its logarithm, thus BFS requires $D$ times more memory space. However if the cost function is real instead of integer, the memory space needed to store the cost may become larger.

Dealing efficiently with small changes in the environment, like adding or removing an obstacle, points out another disadvantage of BFS. Indeed if only the parent and
not the cost is kept for each node, the whole graph has to be recomputed to take into account the cost change of an edge. For the two other methods it is very easy to correct the output when a cost is decreasing or when an obstacle is removed. In this case we just need to start a “wave propagation” from the point that has changed and update all following points that have then a lower cost than before.

Remark: The points that have been updated after an obstacle changed can be seen as the “shadow” of this obstacle. It is also the set of points whose shortest path to the goal goes through this point that has changed.

If the cost of an edge is increasing, or if an obstacle is added, the problem is much more complicated but still doable. The solution we propose is a bit tricky and consist first to simulate a very small cost decrease of the edge that is changing and apply the previous algorithm to find all the points that depend on the changed point (this is its “shadow”). We know that only the points in the shadow domain might need to be updated. Thus we just need to put all the points on the border\(^2\) of the shadow in a sorted queue and do the wave propagation algorithm from here, on the shadow. Of course every new point found has to be inserted according to its cost in the ordered queue. In a word, if the intermediary costs are stored, it is easy and fast to correct them after a small change in the environment.

Last but not least with the DP and FMM, we keep more information in memory during the backwards computation, which can be used during the forward process, when making the path explicit. At each state we can find out all the controls that will lead to a path of shortest distance, whereas in BFS only one parent (which is now a child in the forward process) was stored. Thus it is easy to build a forward tree containing all possible shortest paths and choose between them:

- according to another constraint ;
- by going through some interesting points ;
- by choosing compliant trajectories of N different robots in the same environment (this is faster than the exact solution using a 2N dimensional problem ).

Another similar problem, which is easily solved with FMM is the cat-mouse problem where the cat has to plan a trajectory to intercept the mouse, which follows a known trajectory. Just simulates a wave from the current position of the cat. This will give us a sort of cone (a surface similar to figure 3-3 for which the z coordinate correspond to the lowest time t needed to go to the corresponding (x,y) point). The trajectory of the mouse is an increasing line in this (x,y,t) space. Where they intersect is the target where the cat has to go to catch the mouse.

3.4 Improvement

The cost of the previous algorithms depends on two main factors: the size or diameter of the problem and its dimension. The dimension has an exponential effect on the

\[^2\text{More precisely, the border is the set of points that are not in the shadow domain but are connected to a point in the shadow domain}\]
Figure 3-4: The stripped region represent the points explored at the time the algorithm stops, when using two waves (2D case). This has to be compared to the area of the big circle, corresponding to one wave.

computational cost, but this is also the reason why the following improvement can be significant.

If the goal is to find only one shortest path and find it as fast as possible, the algorithm will stop as soon as it finds one. But in this case the algorithm can be significantly improved by considering two waves propagating, one from the starting point and one from the target point. The algorithm will stop as soon as the two waves meet. To quantify the gain in computational time, we will suppose that our state space is like the usual Cartesian space. Thus in the usual case (one wave), the number of point explored is in the order of the volume of a ball of radius \( r \), whereas in the second case, it is in the order of the volume of two balls of radius \( r/2 \). Looking at figure (3-4) makes it easier to understand; in short, in the first case, the time is \( o(r^D) \) and in the second case, \( o(2(r/2)^D) \): the gain is \( 2^{D-1} \), where \( D \) is the dimension of the state space.

It has been verified on simulations that in the 2D maze problem, the program runs about twice as fast and in the dynamic problem (4D), it is possible to find the shortest path more than eight times faster\(^3\). If the problem would be to navigate in a 3D maze, typically the robot arm avoiding natural obstacles, we would have a 6D state space and the gain using two waves would be up to \( 2^{6-1} = 32 \) times faster. This can make a real difference when implementing a control in real time.

The gain of using two waves is the most important on problems where the complexity comes from a large dimension instead of a large diameter\(^4\) of the state space. To verify this assumption, an algorithm to solve the Traveling Salesman Problem has been implemented.

\(^3\)Since the maze is not a perfect Cartesian space, the gain depends actually on the configuration and the choice of the initial and target points and can vary from 0 up to 10.

\(^4\)By diameter, we mean its definition in metric theory: \( \sup d(x, y) \ x, y \in \text{state space} \).
3.5 Example: the Traveling Salesman Problem

As explained before, the gain of using two waves is particularly high for problems with high dimensionality and low diameter. We present here a modeling and a solution to the Traveling Salesman Problem (TSP) that have these characteristics. The problem is the following: given a finite number of "cities" along with the cost of travel between each pair of them, find the cheapest way of visiting all the cities and returning to your starting point. This is believed to be NP-complete, which means that nobody has found an algorithm that runs in polynomial time. Basically it comes from the fact that there is no really better algorithm than checking all possible tours. The following solution is still non-polynomial but it is much less expensive. This is because it avoids recurrent computation and mainly because it checks if the node corresponding to the complementary trip has already been explored.

The problem of N+1 cities is modeled as a unit hypercube (figure 3-5) of dimension N. The cities are numbered from 0 to N, we choose the city number N as the starting city for the tour and try to find the shortest open tour (or trip) that visits every other N cities exactly once. The solution to the TSP is then the trip whose length plus the distance from the last city back to the starting city is the smallest. To do that we divide the problem in intermediary problems consisting on finding the shortest trip starting from city N, going through a given subset of cities and ending at a given city.

On the hypercube, each node represents a set of cities. A node can be viewed either as a set of N booleans indicating whether it contains city #i or as an integer, which is the decimal value corresponding to this set of bits (the second way is interesting to save memory when implementing the algorithm). A state, which corresponds to an intermediary trip, is a node plus a terminating city: state = (node, lastCity). On the hypercube, the cities are represented by the directions; going from one node to another along direction i, means adding the city i in the trip (note that cities are numbered from 0 to N). For example if we go from node (1,0,1,0) to node (1,0,1,1), we have gone along direction 3 and the second state represents a trip going from city N (4, start city) through cities 0, 2 and to 3.

Now we have formulated the problem as a graph and we can use the previous methods. The idea is, at each step, to take the found state (trip) with the shortest length (and of course not treated yet) from a queue and to put all the following states with their corresponding length in the queue. By following states we mean the states that contain just one more city. If the following state already exist, then just update its length and its place in the queue if it is smaller.

Once a state (a trip) is treated, its length is the shortest possible length to go to this state, that is to make a trip starting from city N going through all cities contained in state.node and ending at state.lastCity. This is shown using recurrence. Suppose that we are processing state $S = (S.node, S.lastCity)$. Since we are treating nodes in their order according to their length, and there is no negative distance between two cities, all the states S could come from are either already treated or have a longer or equal length. For the recursive proof, we assume that all previous states have a length corresponding to the shortest trip. We are also sure that we tested trips corresponding to S with all possible last but one cities (actually, we have not tested
Correspondance between "Add city #" and direction in the hyperspace.

Figure 3-5: (Above) the hypercube representing the Traveling Salesman Problem; each circle represents a different state. A state is represented by a set of cities (a node) and an ending city (the coming direction).

(Below) The representation of the cities in the usual Euclidean 2D space.
### Table 3.1: Comparison between the two algorithms.

N is the number of cities minus one. Time is in ms on a Sun Sparc 10. Time does not mean too much. It is more precise to look at the number of states created (discovered) and at the number of states treated. In the two waves case the results vary a lot from one tour to another; to compare them, we used tours whose best solutions are similar (i.e. always on a square of width 1000).

<table>
<thead>
<tr>
<th></th>
<th>One Wave</th>
<th>Two Waves</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Avg time</td>
<td>created</td>
</tr>
<tr>
<td>10</td>
<td>535</td>
<td>5120</td>
</tr>
<tr>
<td>11</td>
<td>1393</td>
<td>11264</td>
</tr>
<tr>
<td>12</td>
<td>3654</td>
<td>24576</td>
</tr>
<tr>
<td>13</td>
<td>10786</td>
<td>53248</td>
</tr>
<tr>
<td>14</td>
<td>29264</td>
<td>114688</td>
</tr>
<tr>
<td>15</td>
<td>84934</td>
<td>245760</td>
</tr>
<tr>
<td>16</td>
<td>251455</td>
<td>524288</td>
</tr>
<tr>
<td>17</td>
<td>Out of Mem</td>
<td>1231</td>
</tr>
<tr>
<td>18</td>
<td>2077</td>
<td>37664</td>
</tr>
<tr>
<td>19</td>
<td>3341</td>
<td>59193</td>
</tr>
<tr>
<td>20</td>
<td>4847</td>
<td>73590</td>
</tr>
<tr>
<td>22</td>
<td>10309</td>
<td>148747</td>
</tr>
<tr>
<td>24</td>
<td>38930</td>
<td>431580</td>
</tr>
<tr>
<td>25</td>
<td>95855</td>
<td>802815</td>
</tr>
</tbody>
</table>

The hypercube has $N^2N^{-1}$ edges, which is also the number of states. At each step, when processing a state, we do in average $N/2$ operations thus the cost of this algorithm is $o((N^2)^2N^{-2})$.

Now we can use our improvement which consists in simulating two waves, one starting from the start city and another starting from the end city. This is suppose to divide the size of the problem by 2 and thus give a cost $o((N^2)^22N^{-2}) = o(N^22\frac{N^2-5}{2})$. Results are shown in table 3.1. They confirm the predicted cost. However it is not easy to compare tour of different length, because the "difficulty" of a tour depends a lot on how the cities are spread on the plane. These results are obtained with tours where cities are located on a square. So the cost order for the two waves algorithm is the cost for the average case.

It is interesting to notice that, since the problem is symmetric, the second wave
has exactly the same characteristic has the first (it is the same to search for a tour
going in one direction or the other) and checking if the two waves have met only
consist on checking if the complementary node has already been found in the one
wave case. Such a positive check gives us a possible tour, which is already a good
tour; we then have to keep the program running until we are sure we have got the
best tour. This is the case when the length of the current state being tested is greater
than the half of the best tour found up to now.

The hypercube representation let us imagine a massively parallel chip that could
be design to solve the TSP in hardware. This chips would consist of plenty of very
simple processors, one per node on the hypercube, or a total of $2^N$. Each of them
would have to manage the memory cases corresponding to the costs to go to the
states it represents and then send these costs to its neighbors. However this idea has
not been examined deeply, and we do not guaranty there is no technical problem to
do such an electronic chip. More practically, this algorithm may be implemented on
existing parallel computer. Indeed these are often build by connecting $2^n$ processors
on a hypercube structure of dimension $n$. The number of processors is much smaller
than $2^N$, what we would need if we would like to have one processor per node, but
the hypercube has the interesting property that “sub-cubes” are connected to each
other with an hypercube structure. Thus it is very easy to allocate one processor per
sub-cube; this sub-cube will have a dimension of $N - n$, where $N+1$ is the number
of cities in the TSP and $2^n$ the number of processors.

Notes: the algorithm in the one wave case is very close to the one introduced
by Bellman in 1960 [1] and Held in 1962 [8]. But to the best of my knowledge, nobody
has mentioned the improvement of simulating two waves or just checking whether the
complementary of an intermediary tour has already been found. The representation
of a state space for the TSP as an hypercube is also new and interesting and helps un-
derstanding how the algorithm works. There are plenty of algorithms on the traveling
salesman problem, but the best that find exact solutions for every instance can solve
only up to 70-cities problems. Moreover, they often use a combination of different
techniques and have a local minimum approach whereas the algorithm presented here
search for a global minimum.

### 3.6 Link to PDE theory

Since the future position of the robot depends on its derivative (speed and accelera-
tion) and in a force field these derivatives depends on the position, the equation of the
robot is clearly a Partial Differential Equation (if the dimension is more than one).

These equations defining a force field are each derived from PDE’s. It is thus
natural to have a look at other PDE’s that have been studied in the literature to
generate a field that will drive the robot to the target point, through an optimal path
if possible and in a reasonable computational time.

Laplace’s equation $\Delta \phi = 0$ is very simple and well known. The corresponding
field $\phi$, solution of this equation, is said to be a Harmonic Potential and has some
nice properties (the sum of to harmonic functions is still harmonic and it achieves its
extrema only on the boundary). It is the solution of an irrotational and incompressible flow.

Other PDE that could be studied for our problem is the equation of light \( \frac{\partial^2 \phi}{\partial z^2} = -c \frac{\partial^2 \phi}{\partial t^2} \). It has the advantage of being conserved by summation and it represents also a wave. But since it is a second order equation, its computation cost is much harder (in the order of \( N^2 \) instead of \( N \) ). The transport equation \( \frac{\partial \phi}{\partial z} = -c \frac{\partial \phi}{\partial t} \) could solve this last problem but unfortunately its solutions are not always appropriate to model complex control functions. Heat transfer and diffusion-convection equations could also be considered.
Chapter 4

A New Metric on the Workspace

4.1 Motivation

A better, more usable, more practical representation and formalization of the structure of the space (which point is connected to which other, and how far they are from each other) seems to be necessary to analyze and understand what is really happening when the robot moves in the optimal force field. There is already a structure defined at the local level by the cost function but we would like to have tools to work at the global level.

The main motivation is to predict in which cases we can simply add already computed optimal fields to get another force field that has its minimum at a desired point and if possible that is optimal. These tools might also help to do more accurate convergence proofs of some navigation algorithm or trajectory planner.

We present thereafter very general results and unfortunately the theory need to worked further but the algorithm exposed at the end of the chapter seems to be very good in practice.

4.2 Definition

We define a new mathematical distance $d$ from a point $A$ to a point $B$ in the robot’s environment by the length of one of the shortest path from $A$ to $B$.

Note: If our goal is to find the fastest trajectory from $A$ to $B$, then we will define $d(A,B)$ as the shortest time needed to go from $A$ to $B$. In general “length” should be understand as the total cost of a path, regardless of the unit of this cost.

We verify that $d$ is a distance:

$d(x,y) = 0 \Rightarrow x = y$ : this is true as long as the cost function is always strictly positive ;

$\forall x,y , d(x,y) = d(y,x)$ : true for all symmetric graphs, which is the case for the 2D maze ;

$\forall x,y,z , d(x,z) \leq d(x,y) + d(y,z)$ : if this was not the case we could find a shortest path from $x$ to $z$ by going through $y$, contradicting the definition of $d(x,z)$. 

35
Since we have defined a distance, the space in which the robot evolves is now called a Metric Space.

Note that the 4D maze case, for instance, does not fit into this theory because the corresponding graph is not symmetric (i.e. the fact that from a point $X_1 = (x_1, y_1, v_{x1}, v_{y1})$ you can go to a point $X_2 = (x_2, y_2, v_{x2}, v_{y2})$ does not imply that you can go from $X_2$ to $X_1$). And thus the function giving the shortest path from one point to another does not satisfy the second condition in the definition of a distance: $d(X_1, X_2) \neq d(X_2, X_1)$.

4.3 Properties

The main mathematical theory developed in the literature on metric space is Topology. Although this is more about open, closed, compact sets and continuity, which are not very useful for our discrete problem, there are other results that could be interesting. Some of them are:

1. The use of $k$-Lipschitz functions: $f$ s.t. $\exists k > 0 \ s.t. \ \forall x, y \ d(f(x), f(y)) \leq k d(x, y)$ If $k < 1$, $f$ is said to be $k$-contracting.

2. The fixed point theorem, which states that, in a complete metric space, a $k$-contracting function has one and only one fixed point $x$ that is $\exists! x \in E \ s.t. \ f(x) = x$.

The next step is to construct a vectorial space out of this metric space, that is compatible with the definition of the norm corresponding to our distance. This means that we have to define an external law $\cdot$ and an addition $+$ or a scalar product $\langle, \rangle$. Since we have already a distance, it is easy to define the corresponding norm and that explains why we only need to define either the addition or the scalar product. They are connected by the formula:

$$||a + b||^2 = \langle a + b, a + b \rangle = ||a||^2 + ||b||^2 + 2 \langle a, b \rangle$$

$$\iff \langle a, b \rangle = \frac{1}{2}(||a + b||^2 - ||a||^2 - ||b||^2)$$

The scalar product will enable us to do geometry instead of only topology. For example with the scalar product comes the notion of orthogonality, of collinearity and of projection. Viewing the elements of the workspace as vector along with the definition of the external law give us the notion of straight lines (which are usually not straight in the sense of the usual Euclidean space). We can then also use the well-known Cauchy and Minkovski inequalities:

Cauchy-Schwartz : $\forall (x, y) \in E^2, \ | \langle x, y \rangle | \leq ||x|| \ ||y||$

Minkovski : $\forall (x, y) \in E^2, \ ||x + y|| \leq ||x|| + ||y||$

For the external law, we can imagine the following definition for a continuous, unbounded (in the sense of the new distance) space:
Let $k \in \mathbb{R}$ and $v \in E$,

If $k < 1$, $k \cdot v$ is defined as a point $v'$ such that $v'$ is on the shortest path from 0 to $v$ and $d(0, v') = k \cdot d(0, v)$

If $k > 1$, $k \cdot v$ is defined as a point $v'$ such that $v$ is on the shortest path from 0 to $v'$ and $d(0, v) = \frac{1}{k} d(0, v')$

If $k = 1$, $v' = k \cdot v = v$; and if $k = 0$, $v' = k \cdot v = 0$

There always exist such a $v'$, but it is not always unique, so another constraint need to be added so that it is defined in a unique way.

For the scalar product, unfortunately, we didn't find any definition that would stay coherent with our definition of the distance and it is the same problem to define an addition. The question is whether it is possible to find one or not, that is whether our metric space has the structure of a vectorial space or not.

Example of calculation that can be done using the properties of the distance:

let $c_i \in \mathbb{R}^+$, s.t. $\sum c_i = 1$

let $a_i \in E$

Prove that if there exist $A \in E$ such that

$$\forall x \in E, \sum c_i d(a_i, x)^2 = d(A, x)^2 + \sum c_i d(a_i, A)^2$$  \hfill (4.1)

then $A$ is unique. This equation is equivalent to: there exist an optimal field centered in $A$ which square is the weighted sum of the square of different potential fields centered in $a_i$. For the proof, we suppose that there exist two points $A$ and $B$ verifying (4.1). Then, we have for $A$ with $x = B$:

$$\forall x \in E, \sum c_i d(a_i, B)^2 = d(A, B)^2 + \sum c_i d(a_i, A)^2$$

and for $B$ with $x = A$

$$\forall x \in E, \sum c_i d(a_i, A)^2 = d(B, A)^2 + \sum c_i d(a_i, B)^2$$

by adding the two equations, we get:

$$0 = d(A, B)^2 + d(B, A)^2$$

which implies that $d(A, B) = 0 \iff A = B$ q.e.d.

### 4.4 Applications

#### 4.4.1 Superposition of potential fields

One of the applications of this theory could be to find the shortest path from every point to any other very efficiently. When the optimal potential fields have been computed for a few target points (at least the dimension of the problem plus one, example: figure (4-1)), the weighted sum of these fields is a new potential field. The question is to determine whether it has only one minimum and whether it is optimal. The idea behind this is that when we compute a new potential force field using the
Figure 4-1: The three different optimal potential fields used to create a new potential field by combination in figure 4-3 and 4-2.

Propagation techniques from chapter 3, we do a lot of computation we have already done before to compute other force field. Indeed for each computation the simulated wave passes on the same points every time, adding the same corresponding cost.

This works well for the basic Euclidean space, without obstacles with the usual Euclidean distance. A 3D-paraboloid is a potential field for the 2D space and if following the steepest gradient from every point leads to the center of the paraboloid. Moreover, the sum of paraboloids with different centers is a new paraboloid (use the analytical equation of a paraboloid and see that the sum of polynomials of degree 2 is still a polynomial of degree 2). Unfortunately doing the same in the new metric space does not work for the whole space; figure(4-3) shows the sum of the square of three optimal force fields, you can see that in this case it is a “good” field for the entire space, in the sense that there is no local minimum, but this is not a general rule. Taking the square of the fields corresponds to a kind of Norm 2 (figure(4-3)), but we can do the same with a combination inspired from Norm 1 (sum of absolute value) or infinite norm (the highest value at each point, also called winner-takes-all, see figure(4-2)) or everything in between. According to simulations, the method corresponding to the Norm 2 \( \text{NewPot} = \sqrt{\sum c_i \text{OldPot}_i^2} \) seems to give the best results.
Figure 4-2: Superposition (winner-takes-all) of the three force fields shown in figure 4-1.

Figure 4-3: The potential field obtained by adding the square of the three fields shown on figure 4-1.
4.4.2 Convex functions

This is not completely surprising because the square of an increasing function is convex. A convex function verifies:

$$\forall x, y \in E, \forall \lambda \in [0, 1] \ f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y)$$

And it has the two following interesting properties:

1. The sum of convex functions is convex;
2. A convex function has only one global minimum;

In this case, it is a bit as if we were adding two paraboloids to get a surface having the same shape. Thus using convex function would be a good thing. Unfortunately the square of the potential fields are not perfect convex function in the Euclidean space and we cannot prove they are convex according to the new metric because we were not able to define the sum of two vectors. Convex functions would be a very useful application of the construction of a vectorial space.

One thing that explains the method of superposition does not work in the general case is because it does not take into account symmetry. This is also why we presented plots with a symmetric maze (a cross). If the problem is symmetric and if the different potential fields are spread out symmetrically, the resulting potential field is symmetric and the symmetric of a minimum is also a minimum, thus not a global one. Symmetry is only one kind of problem. More generally, when two different parts of the simulated wave are merging, we might get into trouble.

4.4.3 An efficient algorithm

So we did not find a general method for superposition but there is a very efficient practical method to use it.

Imagine a robot traveling many times in the same maze. For the first three or so target points, it computes the optimal potential force field as explained in chapter 3. But it keeps it in memory after it has reached these points. When it has a couple of such potential fields in memory and when it is assigned a new target point $T$ it does the following:

1. Check if we have already computed potential fields $p_i$ such that there exist coefficients $c_i$ s.t. $\sum c_i \text{grad} P_i = \vec{0}$. If NO, compute a new potential field using basic techniques from chapter 3. If YES go to 2.

2. Consider the new potential $\sum c_i P_i$ where the $c_i$ and the $P_i$ are the same as in 1. Make a virtual gradient descent on this new potential, which is very cheap compared to the computation of a new optimal field. If the descent ends up in a local minimum, compute a new field. If the gradient descent ends up in $T$, you just need to do the same gradient descent with the real robot on this new potential. This will not be an optimal path but something near-optimal.

Reading this algorithm leads to the thought that there are too many conditions so that we can effectively gain from the superposition technique, but it appears that in
the real world, a weighted sum of potential fields is very often enough to avoid local minima. See figure 4-3 for an example of such summed fields.
Chapter 5

Application : Concurrent Mapping and Localization

5.1 Introduction

Over the last two decades or so, the problem of acquiring maps in indoor environments has received considerable attention in the mobile robotics community [21]. The problem of map building is the problem of determining the location of entities of interest (such as: landmarks, obstacles), often relative to a global frame of reference (such as a Cartesian coordinate frame). To build a map of its environment, a robot must know where it is relative to past locations. Since robot motion is inaccurate, the robot must solve a concurrent localization problem, whose difficulty increases with the size of the environment (and specifically with the size of possible cycles therein). Thus, the general problem of map building is an example of a chicken and egg problem: to determine the location of the entities of interest, the robot needs to know where it is. To determine where it is, the robot needs to know the locations of the entities of interest. To better understand the challenge of the problem, you can imagine yourself in a dark room filled with a lot of objects, you have to go around feel with yours hands where there is something and remember it so that you can find it later again or avoid it when you want to go to another place in the room.

This work has been done on a simulated Khepera robot. It is a miniature mobile robot with functionality similar to that of larger robots used in research and education. Khepera was originally designed as a research and teaching tool for a Swiss Research Priority Program at EPFL in Lausanne. It allows real world testing of algorithms developed in simulation for trajectory planning, obstacle avoidance, preprocessing of sensory information, and hypotheses on behavior processing, among others. It is a 50 mm diameter robot, with two DC motors, eight infrared proximity sensors (six in the front and two in the back) and battery, processor, memory.

The Khepera Simulator (figure 5-1) is a free-ware package allowing to develop controllers for the mobile robot Khepera using C or C++ languages. It includes an environment editor and a graphical user interface. The information received by the sensors and send to the motors simulates pretty well the noisy signals found on the
real robot. Moreover, if you own a Khepera robot, you will be able to switch very easily between the simulated robot and the real one. It is mainly aimed at teaching and research in autonomous agents.

5.2 Application of previous theories

There are two main parts in my implementation of concurrent mapping and localization on the Khepera Simulator. The first one consists of more or less random travel to explore the environment whereas the second is aimed at resetting accumulated error by going to a known position.

5.2.1 Exploring Phase

At the beginning and during the first 5000 steps, the robot goes more or less randomly. It goes straight and when it approaches an obstacle it avoids it using the proximity sensors: if it detects an obstacle on the right it turns to the left and if it detects one on the left it turns to the right. When it "sees" an unexplored area (i.e. white on the map) it tries to explore it: instead of going straight it turns toward this region.

That is for the navigation algorithm or the trajectory planner. But at the same time the robot has to build a map of the environment and update its position on this map. The basic idea to achieve that is very simple. The position \((x, y, \theta)\) is updated by integrating the desired speed sent to the motor by the controller and at each step the information from the proximity sensors are plotted on the internal map. But the
difficulty comes of course from the noise on information from the sensors and to the motors.

To deal with that we introduce the notion of probability that there is an obstacle at a given point. In other word a pixel on the map is not white or black to indicate that there is an obstacle or there is not one but is any gray tone in between to indicate the probability that there is an obstacle at the point corresponding to this pixel. To achieve that, at each step we update the map by averaging the information from the sensor and from the previous map.

During navigation we also want to use the information we have already collected on the map to correct the position and minimize its error. Thus when the proximity sensors have detected an obstacle, we look a bit further on the internal map in the direction of the sensor to see if we have already found a obstacle. If yes, we assume that they represent both the same obstacle and we correct the estimated position of the robot accordingly.

### 5.2.2 Repositioning Phase

Every 5000 steps the robot goes back to its initial position whose exact coordinates are precisely known. Moreover it is easy to be sure the robot is really at the point it thinks it is, because the initial position is in front of a wall. Thus we can precisely correct the $y$ and $\theta$ coordinates. For more precision, we could have chosen the repositioning point in a corner to also update $x$.

To return to the initial position we use the map we have build and the global path planning techniques developed in chapter 3. The robot first stops to compute the "return function" or potential field. This takes less than a second for a 100x100 grid map and gives the result shown on figure 5-2. This computation simulates one wave starting from the initial position (i.e. where the robot wants to return) for which the propagation speeds is given by the value on the build map. In other words, the internal map is used as a cost function to go from one pixel/point to its neighbors. This means that the highest the probability of an obstacle, the highest the cost to go and the lowest the velocity of the wave: the robot will avoid region where an obstacle is highly probable.

After the potential is computed, the robot slowly descents the gradient of this field towards the know position.

When implementing path planning on a real robot, there is an easy and elegant way to interface the results of the backward computation (the potential field) and the actual command of the actuators. Imagine two (in the 2D case) virtual sensors $s_{right}$ and $s_{left}$ that are placed on each side of the robot and that measure the values of the created field. The following simple control will drive the robot to the minimum of the field, which is the target point:

$$command_{right} = k \times (s_{right} - s_{left}); \quad command_{left} = k \times (s_{left} - s_{right}).$$

Indeed when the right sensor detects a higher value than the left one, the right motor will run faster than the left one, making the robot to turn left until both sensors have the same value, which means that the direction of the robot is collinear with the
The potential field created to drive the robot back to its initial position.

Thus the gradient descent and the global path do not even need to be computed explicitly.

Finally when the robot is close enough to the desired position, it turns so that it goes in the \( \theta = 0 \) direction. Then it goes forward until it is in front of the wall and when the sensors on the right have the same values as the ones on the left it correct its estimated position. Now it can start a new exploring phase.

## 5.3 Results

After 20 000 steps (i.e. a couple of minutes) the internal map build by the robot looks like the right part of figure 5-1, which is a pretty good representation of the real world shown of the left side and precise enough to be used for navigation in this environment. Moreover the error on the coordinates remains always under 10cm in a 1m square maze. It is of interest to note that the most painful error is the one on \( \theta \) because it is hard to correct during the navigation phase and because a small error on \( \theta \) can induce a high error on the \((x, y)\) coordinates.

This results are obtained only on a simulator and even though its simulates a 10% noise on the motors and on the sensors, the noise is not “perfect”. In fact noise is produce by a computer and cannot be perfectly random; especially the percentage of noise on the two motors is the same, which means that when the robot sends a command to go straight it goes exactly straight on the simulator, which may be a bit different on a real robot. However the simulated robot still has to deal with noisy signals in all other cases.
Chapter 6

Conclusions and future work

We have presented here different methods to find a path in a given environment, all based on force fields. Those were usually illustrated by example in robotics and especially in mobile, Cartesian robotics, but most of them can be easily extended to more general problems like optimization, graph search or robot arms.

Contraction Theory has been successfully used to prove convergence of force fields methods and chapter 2 gives some insights on this recent theory. Although these may not be well suited for very large problems, they can be used efficiently for local obstacle avoidance. There is here still further work to be done, for instance study other types of force fields, understand better the dynamic of the system in $s$, define methods to take into account the saturation of the actuators, etc.

Chapter 3 presents more a small improvement and some general analysis but on a domain, which is broad, well studied, very useful and that has a lot of applications in the industry. After presenting and comparing three different methods (Breadth First Search, Dynamic Programming and Fast Marching Method), we propose a significant improvement and apply it to the Traveling Salesman Problem, where it is particularly interesting.

We then introduce a new metric as a useful mathematical tool in chapter 4. It is, to the best of my knowledge, completely new and still needs further work but we already found an algorithm that can save a lot of computation. We explained why and we gave some starting points to find better methods that could reuse some computation when traveling many times in the same environment; the basic idea is to simply add different potential fields. We cannot prove that it is possible to construct a vectorial space or that there is a general method to combine different potential fields but we have good reasons to believe it.

Finally some more practical work has been done: concurrent mapping and localization has been tested on a simulated Khepera robot. It uses techniques developed in the previous chapters to prove that they can be practically used.
Bibliography


