Autonomous navigation in dynamic uncertain environment using probabilistic models of perception and collision risk prediction.

Chiara Fulgenzi

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque

| | | | | | | | | | |

THÈSE

pour obtenir le grade de

DOCTEUR DE L'INPG

Spécialité : *Mathématiques appliquées*

préparée au laboratoire LIG et l'INRIA Rhône-Alpes, dans le cadre de l'École doctorale *Mathématiques, Sciences et Technologies de l'Information, Informatique (MSTII)*

présentée et soutenue publiquement par

Chiara FULGENZI

le 8 juin 2009

Titre :

Autonomous navigation in dynamic uncertain environment using probabilistic models of perception and collision risk prediction

Directeur de thèse :

Christian LAUGIER

Composition du jury :

M. Augustin LUX	Président
M. Jean-Paul LAUMOND	Rapporteur
M. Steven LAVALLE	Rapporteur
M. Emmanuel MAZER	Examinateur
M. René ZAPATA	Examinateur
M. Christian LAUGIER	Directeur de thèse
Mme Anne SPALANZANI	Co-encadrant

Abstract

In this document we address the problem of autonomous navigation in dynamic unknown environments. The key of the problem is to guarantee safety for all the agents moving in the space (people, vehicles and the robot itself) while moving toward a predefined goal. In contrast with static or controlled environments, high dynamic environments present many difficult issues: the detection and tracking of moving obstacles, the prediction of the future state of the world and the on-line motion planning and navigation.

We moved our research starting from the fact that the decision about motion must take into account the limits of sensor perception, the velocity and future trajectory of the moving obstacles and that it must be related with the on-line updating of the world perception. Our approach is based on the use of probabilistic frameworks to represent the uncertainty about the static environment and the dynamic obstacles and we have developed decision methods that take explicitly into account such information for the autonomous navigation task.

At first we focused our attention on reactive approaches. The developed approach is based on the integration between a probabilistic extension of the Velocity Obstacle framework within a dynamic occupancy grid (Bayesian Occupancy Filter). The dynamic occupancy grid gives a probabilistic and explicit representation of the perception limits and of the model error and noise while the Probabilistic Velocity Obstacles compute a probability of collision in time. The probability of collision and hence the choice of the next control, take into account limited sensor range, occlusions, obstacles shape and velocity estimation errors. Simulation results show the robot adapting its behaviour to different conditions of perception, avoiding static and moving obstacles and moving toward the goal.

In the second part of the document we move toward more complex strategies, integrating a Partial Motion Planning algorithm with probabilistic representation of the environment and collision risk prediction. We propose a novel probabilistic extension of the well known Rapidly-exploring Random Trees framework and we integrate this search algorithm into a partial planning anytime approach, updating on-line the decisions of the robot with the last observations. We consider at first the case of a short-term prediction based on a classic Multi-Target-Tracking algorithm. Secondly, we consider the case of a longer term prediction based on *a priori* known typical patterns. We compare

two kinds of patterns representation: the Hidden Markov Model representation and the continuous Gaussian Processes representation. The search and planning algorithm is adapted to these different kinds prediction and simulation results for navigation among multiple moving obstacles are shown.

Resumé

La navigation autonome en environnement dynamique représente encore un défi important pour la recherche en robotique. Le point central du problème est de garantir la sécurité de tous les agents qui se déplacent dans l'espace. Contrairement aux environnements statiques ou contrlés, o les techniques de planification globale peuvent tre adoptées, les environnement dynamiques présentent des difficultés majeures : la détection et le suivi des obstacles mobiles, la prédiction de l'état futur du monde et la planification et la navigation en ligne. Dans cette thèse nous abordons le problème de la prise en compte explicite des incertitudes liées à la perception de l'environnement et à la prédiction de l'état d'un environnement dynamique inconnu dans la decision des mouvements futures du robots. Dans la première partie du document nous proposons un algorithme réactif. L'approche développée est basée sur l'intégration de l'extension probabiliste de la méthode dite Velocity Obstacle (Obstacles de vitesse) et d'une grille d'occupation dynamique (le Filtre d'Occupation Bayesien, BOF). La probabilité de collision et le choix du contrle prennent en compte la portée des capteurs, les occultations, les incertitudes sur l'estimation de la forme et de la vitesse des obstacles. Dans la deuxième partie du document nous proposons une méthode de planification partielle : nous avons développé une nouvelle extension probabiliste de l'algorithme de recherche RRT et nous l'utilisons pour une approche de planification et re-planification "anytime", en mettant à jour les décisions du robot en relation avec les observations les plus récentes. Nous considérons d'abord le cas d'une prédiction à court terme ; dans un second temps nous abordons le cas de prédiction a moyen terme, basée sur les trajectoires typiques des obstacles, apprises a priori.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Autonomous robots are widely spread in industries, where their space of work is well protected, so that nothing can interfere with the moving robot. In order to introduce robots in everyday life environments, safe systems of navigation in dynamic and populated environment have to be developed. In the last decade a variety of mobile robots designed to operate autonomously in environments populated by humans has been developed. These robots have been deployed in hospitals, office buildings, department stores, and museums. Existing robotic systems are able to perform various services such as delivery, education, providing tele-presence, cleaning, or entertainment. Furthermore, there are prototypes of autonomous wheelchairs and intelligent service robots which are designed to assist people in their homes.

From another point of view, industries and governmental institutions are interested at the develop-

a b c

Figure 1.1: Various types of robots: (a) Robotic arms in car industry, (b) roomba, an autonomous vacuum cleaner (c) RHINO a museum tourguide.

ment of automatic methods for assist drivers in their cars and at the production of fully autonomous vehicles that could move safely in towns, among buildings, human driven vehicles and other road agents as cyclists and pedestrians. This interest in pushed by the fact that roads are a dangerous

environment: accidents on the road represent an high percentage of death in most of the developed countries; in urban scenarios, cyclists and pedestrians are at the first place in victims of drivers and cars. Also, in recent years, more and more electronics have been gradually put in cars to reduce the accidents rate and their incidence on people health (ESP, ABS, ACC ...). Only a few examples of autonomous driven systems have been produced and used: this is the case of the autonomously driven metros and little other experimental plateforms. For safety reasons, the use of these vehicles is usually limited to dedicated roads, where no other vehicles or people are supposed to move or stay. In 2005 and 2007 the American Defence Advanced Research Projects Agency (DARPA) have sponsorized a race for autonomous vehicles in desert environment first and in urban like environment then. In these competitions, multiple teams have been addressing the autonomous driving task or roads among static obstacles and other moving vehicles. Only a few of the presented vehicles were

a b c

Figure 1.2: Autonomous vehicles: (a) a driverless metro in Copenaghen; (b) Cycabs in a platoon; (c) Tartan-Racing, the winner or DARPA urban challenge 2007.

able to attend the end of the competition. This is due to the intrinsic complexity of the navigation task, expecially in urban and in general in highly dynamic environment.

Autonomous navigation in populated environments represents still an important challenge for robotics research. In contrast with static or controlled environments where global path planning approaches are suitable, high dynamic environments present many difficult issues: the detection and tracking of the moving obstacles, the prediction of the future state of the world and the on-line motion planning and navigation. The decision about motion must be related with the on-line perception of the world and take into account all the sources of uncertainty involved. In the last few years, the problem of incomplete, uncertain and changing information in the navigation problem domain has gained more and more interest in the robotic community, and probabilistic frameworks aimed at the integration and proper elaboration of such information have been developed.

1.1 Problem description

1.1.1 Autonomous navigation in dynamic environment

The problem addressed in this document is the autonomous navigation in an uncertain dynamic environment. Our aim is to develop techniques allowing a robot to move autonomously and safely in an environment which is not perfectly known *a priori* and in which static and moving obstacles are present. The task of the autonomous robot is to find and execute a continuous sequence of actions that leads it each time to a given position or goal avoiding collisions with the obstacles.
In a navigation problem the robot is concerned with the following subproblems:

Where I am?	Localization
Where are the obstacles?	Mapping
Where will obstacles be in the future?	Prediction
Is there a trajectory to the goal?	Motion Planning
How do I execute the trajectory?	Trajectory Following

The robot is equipped with sensors, intelligence and actuators to face and solve these problems:

- **Perception** The robot is equipped with proprioceptive and exteroceptive sensors. Proprioceptive sensors give the robot internal informations about its own state; in the case of a mobile wheeled robots, such sensors can be odometric sensors (encoders) and heading sensors (gyroscopes). Compasses and GPS give the robot information about its own state relative to the world (absolute heading and absolute position). Exteroceptive sensors give the robot information about the external world. This is the case of contact sensors, distance and bearing sensors, visual sensors, etc (light, noise sensors...). The robot needs models of itself and of the world to interpret the raw data received by the sensor and extract the necessary information to perform localization and mapping.

- **Prediction** To make decisions, the robot needs to know how the world evolves in the future. Again, we can distinguish prediction in internal and external prediction. Internal prediction is the prediction about the robot own state. External prediction tells the robot how the world changes into the future: these changing can be independent of the actions of the robot or can in part be correlated to them. For long time, the robotic community has addressed navigation problem assuming the simplifying hypothesis that the world is static except for the robot and its actions; however, when going to highly dynamic and unknown environments or considering more robots, the fact that the environment changes in time cannot be side stepped. Hence the need of prediction and of dynamic models of the world and of moving obstacles has arisen.

- **Decision** Given its kinematic model, the model of the system and the goal, the robot must find a way to reach the goal from the current position satisfying some constraints as avoid

collision with obstacles, reach the goal within some time, find the shortest path... The decision process must be based not only on the *a priori* information, but mainly on the information gathered by perception. This is the part this thesis is mostly concerned with.

– **Action** The robot puts in practice the decision taken with its actuators, in this case the locomotive motors which allow the robot to move in the environment. The action should be taken out in closed loop control so that in case of fault, the decision is corrected or the decision process is resumed.

We have particularly addressed the problem of putting in relation the decision and action process with a realistic perception input, assuming hat the robot has little *a priori* knowledge of the static environment and of the moving obstacles around it.
We looked at the problems and limits inherent to sensor perception and future prediction and turned ourselves toward environmental models that could express and update at best the information and uncertainty coming from perception so that the decision process could use and take advantage from them.

1.1.2 Dynamic environment modelling

The environment is the space in which the robot is supposed to operate. We call an environment *dynamic* when the positions occupied by some obstacles can change in time. This can be due to objects that: change position in time, change shape in time or appear or disappear in the workspace. A map of a dynamic environment useful for navigation is a model where the robot can distinguish the positions in which it can safely stay or move in the present and future time. Giving a map of a dynamic environment is not an easy issue because the shape of moving obstacles, their initial position and their future trajectories are often known only up to some limit or unknown. Usually, some *a priori* hypothesis is taken for example, on the appearance of the obstacles in the sensor data and on their behaviour or motion model. On the basis of such hypotheses and given at each instant the sensors data, the mapping algorithm must distinguish the free and occupied space around the robot; determine which are the moving obstacles and give an estimation of their future trajectories; update these informations on-line to follow new entering obstacles and, if possible, predict the future position of the obstacles.
Along the document we will describe and discuss the choice of different environmental and predictive models that at each chapter seemed the most adapted to represent the environment relatively to the information needed by the task of the robot and by the chosen decision approach.

1.1.3 Dealing with uncertainty

The basic problem of navigation assumes that the geometry of the robot, the geometry of the obstacles and their position are accurately known. It also assumes that the robot can exactly take

the chosen actions and that other obstacles motion is exactly known in advance. However, there is no real world robot or environment that can satisfy these assumptions: robot control and geometric models are imperfect, future is unknown. Furthermore, the robot could have little knowledge about its workspace, so it would have to rely on its sensors observations to build or refine its model. We will call *uncertain* an information that is inaccurate, incomplete, noisy or possibly wrong. In this paragraph we will analize the sources of uncertainty for an autonomous mobile robot in a dynamic environment and the consequences they can have on the navigation task. We will then analyse the ways uncertainty is usually represented. Finally, we will discuss on the opportunity to take into account such uncertainty and on the choice of using a probabilistic representation.

Sources

In usual robotics problem, uncertainty arise from all the components of a robotic system. In detail:

1. sensors: the sensors the robot is equipped with have a limited range and resolution and their measures are affected by noise and faults.
2. models : the mathematical models are a simplified representation of reality: they are incomplete and they correspond to a less or more important approximation; their reliability is time related. Some robotics applications also, are programmed to estimate on-line the parameters of the models: these parameters are initially unknown or largely estimated.
3. actuators : the actuators of the robot are not perfect and are affected by faults.

These factors are of big importance in the navigation problem as they heavily affect the decision process at all stages. Following the framework developed in [1], we can divide uncertainty in four classes:

– **Configuration Sensing** (*CS*) uncertainty: Due to the sensor limited accuracy and the actuators error, the state of the robot is not exactly observable. The robot must make an inference about its current configuration on the basis of the initial configuration and the actions history only (*dead reckoning*) or using the sensors observations also (localization).

– **Configuration Predictability** (*CP*) uncertainty: Due to the actuators error or wrong hypotheses on the environment, the state of the robot is not exactly predictable.

– **Environment Sensing** (*ES*) uncertainty: Due to the sensors limited range, the information the robot has on the world is limited to its neighbourhood; also, depending on the type of sensors used, there can be areas of occlusion. Due to the sensors accuracy, the shape and exact position of obstacles is not exactly known. In relation with the models used and with the amount and quality of *a priori* knowledge, the representation of the world can be more or less accurate: the models are built taking in account a trade-off between accuracy and complexity. For what

concerns the static world, the critic parameters in the case of a continuous model are the number and the dimensions of the primitives chosen to fit all the shapes in the environment; in grid models instead, the critic parameters are the dimension, the size and the discretization step of the grid.

– **Environment Predictability** (*EP*) uncertainty: For what concerns moving obstacles, models not only must specify the shape and dimensions of the obstacles, but also the way they move into the environment. Also, the robot needs to know if, when and where new obstacles can enter the workspace. Such models must integrate knowledge about the structure of the environment. To design such models can be a very difficult task. When the models are not known or too complex, simpler models are used whose validity is however limited to a restricted class of obstacles and to a short time period.

Each of this sources can be considered separately and most of the algorithms developed in literature face one simplified combination of the whole problem. In the following paragraphs we will compare the presented algorithm also on the base of the class of uncertainty that the algorithm allows to face. Our work focuses at first on environment sensing and environment predictability uncertainty.

Representation

In many applications, the way the uncertainty is taken into account and represented is a critic choice for the success of the tasks of the robot. Lets list the main representations used.

Deterministic : The uncertainty is simply not taken into account. The observations of sensor are used as true values and the state of the world is supposed to be known and perfectly predictable. This kind of representation can be used only in artificial or high controlled environments, where the model used can be considered complete and perfect. Deterministic models are the correspondent of so called *open loop* control systems, which are used when the tolerance of the problem is far away bigger that its uncertainty. However, whenever the computational power and perception capabilities are strong enough, these models are avoided, at least for safety reasons.

A second way to represent uncertainty but using deterministic models can be called **Implicit representation**. In this case the model of the world is deterministic, but the algorithm itself is designed to handle unexpected changes or states. The implicit model is, for example, at the basis of most reactive approaches and of planning and replanning approaches. Only the local environment is considered at each time step. A binary (eventually *worst case*) representation (occupied / free space) is given and the robot chooses only the next action to take. The fact that the robot has a finite visibility range and that there is little information about the future state of the obstacles is not explicitly modeled, but the algorithm faces these limitations iterating the

decisions at each time step. Another group of implicit approaches are the *feedback-planners*, which plan an action from all possible states.

Indeterministic or Possibilistic : An upper bound to uncertainty is assumed, but not additional information is used. Usually *worst case* analysis is performed to take safe decisions. This choice gives good results when the uncertainty bound is low enough to allow take safe and effective decisions. In this case the eventually available additional information is not used and the computation is simplified. However, the *worst case* analysis can restrict drastically the set of possible solutions giving sub-optimal results and even fail to find an existent feasible solution. If worst case analysis seems too pessimistic, *medium case* analysis can be performed, which however does not guarantee safe solutions.

Probabilistic : The system is supposed to know (or it is able to learn) some statistics about uncertainty and uses the statistics to make probabilistic estimations. Uncertainty is expressed in the form of a posterior probability density; *expected case* analysis is performed. The mathematical tools developed around probabilistic models are well theoretically founded and very powerful. The drawback of using these methods is that a big amount of high level information has to be collected and set into the statistical models of uncertainty and that the computation can become far too complex. The needed information for such models is not always available, so important and restrictive *a priori* hypotheses are used. When the computation become too complex with respect to the available means and time, approximated solutions are necessary.

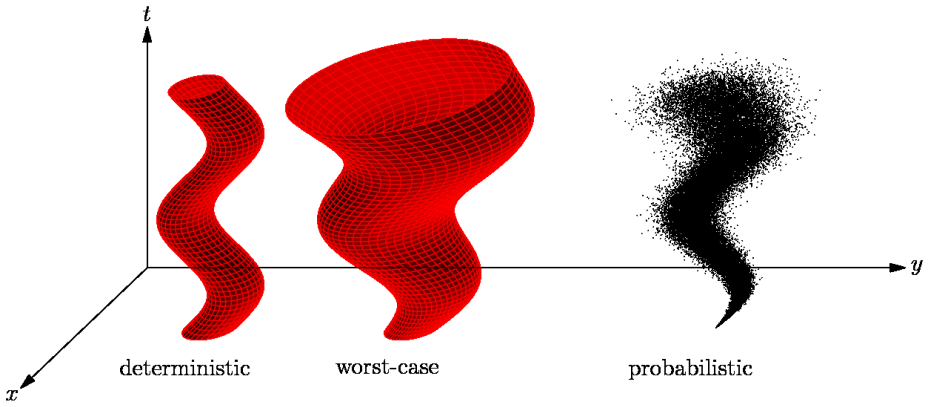


Figure 1.3: Uncertainty Representations.

Information Spaces

The *Information Space* approach consists in formulating the problem directly in the space of available information. In the case of worse case approach, the enlarged states are formed by the uncertainty

intervals of the original state. In the case of probabilistic approach, the enlarged space is formed by probability distributions over the original state. The idea of this formulation arises from the fact that for some tasks there can be no need to precisely estimate the state of the system. Instead, considerations on the solvability of the problem and search for optimal solutions can be formulated and performed in this enlarged space, whose state is always known. However, formulating and finding of solutions in this space is highly task-dependent and an open research problem.

Motivation

In the following of the document we made generally the choice to represent the uncertainty in the perception and prediction of the environment through probabilistic models. In the last decades, the probabilistic models have been more and more applied in automatic systems and in robotics. On one side, the development of computational resources and mathematical tools make it possible to use such complex models in reasonable time. On the other side, in many difficult but fundamental problems the probabilistic representation has revealed the only solution to develop robust and successful algorithms. For what concerns mobile robotics, we can take as examples the Localization problem and the Simultaneous Localization and Mapping (SLAM) problems, where Bayesian recursive inference (the Kalman Filters [2] first and the particle filters [3, 4] then) has been the fundamental step toward theoretical and practical solutions to the problem. Also, most applications around static world representation, target tracking and motion prediction are based on probabilistic models. Along the document we will justify the use of the chosen representations by mean of theoretical considerations and experimental comparison.

1.2 Developed approach and thesis contribution

The aim of this work is to develop an algorithm for safe autonomous navigation among static and moving obstacles. We moved our research from the following considerations:

- The **uncertainty** and **incompleteness** of the information perceived by the robot is not negligible and some mean to take it into account into the decision process should be introduced;
- The fact that the environment is **dynamic** cannot be ignored: the robot performance is influenced by the fact that obstacles move in the environment and the robot should be able to take safe and good decisions at anytime and act promptly in the dynamic environment.

During this thesis we developed decision methods which could handle and take advantage of the probabilistic information that directly comes from perception. The unknown dynamic environment is perceived and mapped by the robot: the uncertainty and incompleteness of the sensing information regarding both the static and dynamic environment should be properly mapped and probabilistic representations are chosen. The velocity of the obstacles, the uncertainty about static environment

information and future prediction is taken into account in the decision process and properly updated with the new incoming information. We focussed our attention on the uncertainty coming from environment sensing and prediction, discarding the uncertainty due to the robot configuration sensing and prediction (see §1.1.3).

At first we developed a reactive method, based on a dynamic occupancy grid: the proposed algorithm is based on the integration between a probabilistic extension of the Velocity Obstacle framework within the Bayesian Occupancy Filter. The dynamic occupancy grid gives a probabilistic and explicit representation of the preception limits and of the error and noise of the sensor model; the Velocity Obstacle framework provides the basis to compute the probability of collision associated to each control of the robot. A cell-to-cell approach has been developed: the probability of collision in time for linear velocities of the robot is computed. A navigation strategy has been implemented, where the choice of the next control depends on its probability of collision. Simulation results shows the robot avoiding static and moving obstacles and moving toward the goal and adapting its behaviour to different conditions of perception: the dynamic occupancy grid representation and the developed approach allow to take explicitly into account and observe the effect of limited sensor range, occlusion and velocity estimation errors. The algorithm is able to work both without an explicit object representation (cell-to-cell) or considering cell clusters as coherent objects. Results show comparison between the two techniques.

In the second part of the document we move to a more forward-looking strategy, integrating a Partial Motion Planning algorithm with probabilistic representation and prediction. We propose a novel probabilistic extension of the well known RRT framework: the configuration-time space is explored randomly incrementally growing a tree. A method is developed to compute the probability of collision due to the static and moving obstacle of each node and the probability of success of each path in the tree. These probabilities give a bias so that the growth of the tree is driven toward the most likely zones of the space. The search algorithm is integrated in an anytime planning and replanning approach: the probabilities of collision and the decisions of the robot are updated on-line with the most recent observations. The algorithm is applied to unknown dynamic environments. The probabilities of collision are computed on the basis of the probabilistic models which map the static and the dynamic obstacles: the static environment is mapped with an occupancy grid, which is updated on-line integrating the incoming information. The moving obstacles are also detected and tracked on-line; for what concerns prediction, we consider at first a short-term, linear prediction based on the motion model estimated by a classic Multi-Target Tracking algorithm. In a second moment we consider the case of a longer term prediction based on *a priori* learned typical patterns. We consider two kinds of patterns representation: the Hidden Markov Model representation and the continuous Gaussian Processes representation. The developed algorithm is adapted to these different kinds of prediction and advantages and drawbacks of these representation are discussed. Results show how the navigation algorithm deals with unknown static environment and multiple moving obstacles with

uncertain trajectories. Comparison among different prediction techniques is presented. The probabilistic framework developed form a solid basis for the integration of multi-sensor perception and hybrid prediction techniques. The possibility to extend the method to coordination among multiple robots is discussed. Part of the work here described has been published in peer reviewed conference papers:

- Chiara Fulgenzi, Anne Spalanzani, Christian Laugier, "Dynamic Obstacle Avoidance in uncertain environment combining PVOs and Occupancy Grid.", IEEE International Conference on Robotics and Automation, ICRA 2007 [5];
- Chiara Fulgenzi, Anne Spalanzani, Christian Laugier, "Combining Probabilistic Velocity Obstacles and Occupancy Grid for safe Navigation in dynamic environments.", Workshop on safe navigation in IEEE International Conference on Robotics and Automation, ICRA 2007 [6]; ,
- Chiara Fulgenzi, Christopher Tay, Anne Spalanzani, Christian Laugier, "Probabilistic navigation in dynamic environment using Rapidly-exploring Random Trees and Gaussian Processes.", IEEE/RSJ International Conference on Intelligent RObots and Systems, IROS 2008 [7];
- Chiara Fulgenzi, Anne Spalanzani, Christian Laugier, "Probabilistic Rapidly-exploring Random Trees for autonomous navigation among moving obstacles.", Workshop on safe navigation in IEEE International Conference on Robotics and Automation, ICRA 2009 [8];

1.3 Thesis outline

This documents presents 6 chapters including this introduction and final remarks.
Chapter 2 presents the developed reactive navigation method. After a presentation of the various state of the art reactive approaches, the Bayesian Occupancy Filter and the Velocity Obstacle method are introduced. While the Bayesian Occupancy Filter provides the map of the dynamic environment, a cell-to-cell method based on Velocity Obstacles is developed to compute the probability of collision in time for linear velocities of the robot. Simulation results show the robot adapting its behaviour to different degrees of sensing information.
Chapter 3 extends the horizon to partial planning methods. The planning methods and some of their extensions to handle uncertainty will be introduced. Expecially, the Rapidly-exploring Random Trees method will be detailed, as is at the basis of the proposed method. The prediction obtained by an on-line tracking method is delined. We present then our method for online planning and replanning among moving obstacles with uncertain trajectory. Results for motion planning on a real dataset of multiple loving pedestrian are shown and discussed.
Chapter 4 introduces the Hidden Markov Models and the Gaussian Processes methods to learn typical pedestrians or car paths in specific environments. The prediction issued by these two methods is delined and the navigation method detailed in the previous chapter is extended to take advantage

of these added information. Simulation results of navigation in closed environment among multiple obstacles following typical patterns are shown and discussed. Comparison between the different prediction methods is presented.

Chapter 5 presents the implementation details of the described algorithms and the simulator used for experimental results of the proposed methods.

Finally, **Chapter 6** ends the document with perspectives for future works and remarks on the given contribution.

Chapter 2

Reactive Navigation

Autonomous navigation in dynamic environment requires the system to be *reactive* to the changes of the environment. Uncertainty is a property inherent to most of real world dynamic environments: is somebody coming in from that door? Is that car going to brake or steer in the next few seconds? In many practical situations one does not have enough information to imagine the exact future and consequently to plan her own actions in distant future. In contrast with path planning approaches, where the problem is to find a complete path from the initial position to the goal in a known environment, many real world applications for autonomous navigation in dynamic environment are then based on *reactive collision avoidance* techniques.
The problem addressed by reactive approaches is the computation of only the next command of the robot. The usual approach is based on the definition of a function that specifies the "cost" of the possible controls or movements in terms of clearence with the surrounding obstacles and of direction of the desired goal. At each time step, the algorithm looks for the minimum cost and the correspondent control is executed. In order for the computation to be fast enough and allow the robot to be reactive to the changes in the environment, only the local information and a short-time period are considered, i.e. the cost function only depends on the one-step ahead prediction of the robot position and of the near-by obstacles. Also, important simplifying hypotheses are usually taken:

- The environment is static: all obstacles have zero velocity and there are no moving obstacles that can enter the scene during motion.
- The environment is perfectly observed: the uncertainty given by perception is not taken into account.

These assumptions further reduce the complexity of the decision problem. The major advantage then, is that the robot can promptly adapt itself to the changes in the observed environment; the developed algorithms are usually fast enough to allow navigation also in low dynamic environments.

On the other side, such simplifications present some important drawback: ignoring the time dimension and the obstacles velocity, the robot can put itself in dangerous situations. For example, the robot could stop in areas crossed by moving obstacles instead of preferring quite areas, or could find itself trapped by multiple moving obstacles in a situation of inevitable collision [9] (see Fig. 2.1). In presence of moving obstacles, the static assumption can induce the robot to discard feasible solutions or perform obscillating behaviour. Worse, if the robot cannot stop instantaneusly it could also cause a collision. To avoid this problem, the robot is usually kept far from obstacles by the use of safety boundaries which allow it to stop before collision. Too high boundaries could prevent any motion in cluttered environments while too low boundaries could lead to collision, so the choice of safety boundaries must be driven by a-priori information about the dynamic properties of the obstacles, which is a strict hypothesis on the environment and eventually on the capabilities of the robot to distinguish between different kinds of obstacles. In the scientific literature, only a few algorithms

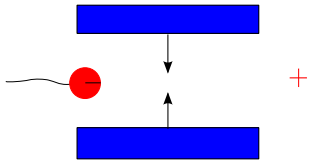


Figure 2.1: The robot goes toward inevitable collision in a shrinking passage.

have been proposed which take into consideration the velocity of the obstacles (*Velocity Obstacle*, *Dynamic Object Velocity*). This kind of algorithms are more suited to high dynamic environments, but are more computationally demanding and require more information. In particular a mean to extract moving objects from the sensors readings and to accurately estimate their shape and velocity is required. So these algorithms are extremely sensible to the precision of detection and velocity estimation in this phase.

The uncertainty and incompleteness raising from the limits of the sensors, the error of the detection and tracking algorithms and the possibility that the obstacles change their velocity are usually ignored. It is generally assumed that the uncertainty is neglectable with respect to the safety boundaries or that the iterations of the algorithm are fast enough to correct and update the information before the robot puts itself in danger. However, this depends not only on the sensor equipment and the robot dynamics, but also on the structure of the environment and the behaviour of the moving obstacles.

In this chapter we present a novel algorithm which both takes into account explicitly the dynamics of the obstacles and the uncertainties in perception. In particular we will focus our attention on the areas that are occluded or out of perception and the uncertainties in obstacles shape and velocity. The algorithm we propose is based on the velocity obstacles approach but takes in input a probabilistic space-time grid, the Bayesian Occupancy Filter. In the following paragraphs we will first present

some of the most successful approaches developed for reactive navigation and compare them. We will then present in more detail the Velocity Obstacle approach and the Bayesian Occupancy Filter, which are the background work for the developed approach. The navigation algorithm developed is then presented. Examples and performance analysis in challenging situations end the chapter.

2.1 History and State of the Art

In the this section we will resume some of the most used reactive methods following the chronological order of appearance. We will also briefly present path and trajectory deformation techniques that can be classified as hybrid methods which are aimed at reactively adapt paths or trajectories computed a-priori.

2.1.1 Reactive techniques

- **Potential Fields** (PF) Suggested by [10], the method imagines that obstacles exert repulsive forces on the robot, while the goal applies an attractive force. The force vector resultant by the sum of the two vectors at the robot position is calculated and used as the accelerating force.

- **Vector Field Histogram** (VFH) In a first step VFH [11] builds an occupancy grid, where the value of occupancy likelihood is increased in the cell in which the measure reading is found. Secondly, the grid is mapped in to a polar histogram around the robot location, each sector representing a possible moving direction. For each sector, an *obstacle density* is calculated summing the occupancy likelihood of each cell weighted by its distance from the center of the robot. A thresholding is applied in order to select directions with low obstacle density. The direction which is closest to the goal direction is selected, while the velocity is computed taking into account the distance to the obstacles. An extension called VFH^+ [12] considers instead circular trajectories.

- **Curvature Velocity and Lane Curnature Velocity** (CV) The problem is considered as a constrained optimization problem in the control space: the set of all the pairs (v, ω) of the traslational velocity v and the rotational velocity reachable from the present state is considered. The robot is assumed to move along arc of circles and the obstacles are approximated with circles. The method, developed in [13] looks for the command that maximizes a weighted sum of the speed of motion, goal heading and the maximum distance that can be travelled before encountering an obstacle. The implementation is based on a discretized approximation of this objective function. This method was extended in [14] with the name of Lane Curvature Method in which both linear and circular trajectories are considered. Also, larger free paths (lanes) are preferred, and better results are obtained in terms of obstacle clearance.

- **Dynamic Window** (DW) is a very popular approach developed by Fox et al. in 1997 [15] and has been used in multiple real world applications. The search space is limited to circular trajectories determined by the commands (v, ω) reachable from the present state. The algorithm proceeds in two step: first the set of admissible velocity (*Dynamic Window*) is found; secondly, the velocity of the set that maximizes the objective function is chosen. A velocity is admissible if it is reachable from the present state under the dynamic constraints of the robot and if the robot can stop before it reaches the closest obstacle on the corresponding curvature. Obstacles are considered static. The objective function takes into account the direction of the goal, the distance from the nearest obstacle along the trajectory and the velocity of the robot. In [16] the objective function used is a NF1 function, which by definition is a function which does not present local minima and extends the approach to a global navigation. To achieve the computation, the control space is discretized; the discretization step directly influences the computational time and the quality of results.

- **Nearness Diagram** (ND) Developed in [17], the algorithm distinguishes five situations or security levels and specifies different control strategies for each of them. Two polar diagrams are built around the robot: one gives the distances of the obstacles from the robot and the other the distances from center of the robot in each polar sector. The free walking areas (*valleys*) are extracted from the second diagram and one of them is selected according to the characteristics if the discontinuity and the goal direction. Finally the level of security is determined on the basis of the first diagram and the choice of the appropriate command is taken out. This approach allows to overcome some drawbacks of previous approaches for what concerns local minima due to totally visible U-shaped obstacles. It was also extended to global navigation with the use of an $NF1$ function.

- **Obstacle Restriction** (OR) Proposed by [18], the method proceeds in two steps:

  - a local procedure decides whether the motion should be directed toward the goal or should find an alternative subgoal;
  - motion restriction is associated to each obstacle and used to compute safe motion.

  In the first step, the robot checks on the fly if the *tunnel* to the goal is free or blocked. If it is free than is not necessary to use a subgoal, otherwise a set of subgoals is determined on the basis of the distribution of the obstacles and the closest to the robot with a free tunnel is selected. In the second step, a safety distance from the obstacles and the direction of the goal are used to restrict the desired motion of the robot. The motion is finally selected considering the limits of the motion restrictions and the direction of the goal. The advantage of this method is to avoid U-shaped traps and oscillations even in cluttered and dynamic environments.

- **Inevitable Collision State** (ICS) In [19], Fraichard proposes the concept of Inevitable

Collision States (ICS). A state is an ICS if whatever control or sequence of controls is applied next, the robot will inevitably enter in collision with some obstacle. The idea rises from the fact that the dynamic constraints of the robot could prevent it to avoid collision when it is in a collision-free position but the distance and the relative velocity between it and some obstacle does not allow an avoiding maneuver. So, a control can be safely applied to the robot only if the next state reached is not an ICS. Knowing the trajectory of the obstacles up to infinite time, the set of ICSs can be approximated studying a finite set of bang-bang maneuvers. Alternatively, given the desired control, the set of trajectories is applied to the expected state.

- **Velocity Obstacles** (VO) is the set of linear velocities of the robot that will be in collision with an obstacle at some time. The linear constant velocity of the obstacles is assumed to be known. The approach [20] consists in a simple geometric method to find the VO considering a finite time horizon. A navigation method can be realized using an objective function to choose the best velocity among the reachable velocities that are out of the VO. The algorithm has been extended to handle whatever velocity of the obstacles (Non-linear Velocity Obstacles) using discrete approximations [21]. Another extension called PVO considers a probabilistic representation of uncertainty for the obstacles velocities and presents interesting theoretical results in multi robot coordinated navigation [22].

- **Dynamic Object Velocity** (DOV) This approach, proposed in [23], maps the static and moving obstacles in the velocity-time space referred to the robot. This is done using a discrete set of curvatures for the robot path and finding the limits in time for which there would be collision with an obstacle. Considering obstacles which move with constant linear velocity, the Dynamic Object Velocity is built. The goal is mapped in the velocity space given the needed rotation to align the robot with the goal and the maximum velocity of the robot. The policy chosen is to reach the desired command as soon as possible, considering the constraints of the robot and the collision avoidance. First the desired command is checked for collision with the projection of the DOV in the velocity space: if it is in collision, the nearest, collision free command is chosen. Then, the reachable command set is considered and the nearest velocity to the objective one is picked. This velocity is checked for collision with the DOV is chosen if collision free.

2.1.2 Path and Trajectory Deformation

The idea of path deformation techniques is to adapt a global plan elaborated a-priori using the newly acquired information. Even if these approaches require an a-priori knowledge of the environment structure and a nominal global plan, they are often presented toghether with the reactive approaches, because the way the path is adapted is based on reactive techniques. Path deformation techniques have been proposed in the early 80s as an effective way to fill the gap between reactive

and path planning approaches. While thinking to a complete navigation algorithm, the interaction between global planning and reactive algorithm is infact a difficult issue: one must decide how much information should be used at the planning level and how much should be reserved to the reactive one, where to put local subgoals, how much the real trajectory can go far from the nominal one before replanning is needed.

- **Elastic Bands** (EB) are the first and one of the most successful approach belonging to this class. Proposed by Quinlan and Khatib in 1993 [24], the method imagines path as a band subjected to internal contraction forces and external repulsive forces. The contraction force removes oscillations along the path; the repulsive force is applied by the obstacles and assures obstacle clearance. These two forces deform continuously the initial path: the resultant path is smooth and adapts to new observed obstacles. In the practical implementation the band is represented by a sequence of spheres or *bubbles* with the same dimension of the configuration space; the center of the sphere is the chosen path and its radius is the distance with the nearest obstacle. The method has been used also in dynamic environments, even if the information about the dynamics of the obstacles is not used. By the fact that the time dimension is ignored, the velocity of the robot is maintained constant and the path can be moved far away from the nominal trajectory. in this case a new path must be elaborated by the planner.

- **Trajectory deformation** (TD) Recently, a method which takes into account the time dimension has been proposed [25]. The deformation is carried out both in the spatial and time dimension and the dynamics of the obstacles are taken into account. The algorithm recalls the elastic band strategy: the trajectory is discretized in a set of nodes that are moved under the influence of repulsive and attractive forces. The algorithm performance is regulated by two parameters which weight the internal forces so the trajectory is mostly deformed in time or in space respectively.

2.1.3 Comparison

In the next table the presented methods are resumed and compared. Only few of them take in consideration the fact that the environment is dynamic (first column). The second column asks if the algorithms take into account the kinematic and dynamic constraints of the robot. The third column shown wether the algorithms are applied to the continuous set of velocity or if discretization is necessary. Column *Uncertainty* shows if the approaches represent explicitly the uncertainty in the perception system. The last column shows '+' when the complexity of the algorithm is low or very low.
Also, only VFH and VO can give an explicit representation of uncertainty, to some extent: VFH consideres uncertainty in perception, but does not account for the uncertain velocity of the obstacle. VO, and especially the PVO extension, considers uncertainty in the shape of circular obstacles and

	Dynamic	Constraints	Cnt/Dscrt	Uncertainty	Complexity
PF	-	-	C	-	++
VFH	-	-	D	+	+
CV	-	+	D	-	+
DW	-	+	D	-	++
ND	-	-	D	-	+
ORM	-	-	C	-	+
ICS	+	+	C	-	–
VO	+	-	C	+-	++
DOV	+	+	D	-	-
EB	-	-	C	-	-
TD	+		D	-	–

Table 2.1: Reactive approaches comparison.

a discretized symmetric uncertainty on velocity. Most of the algorithms rely on their very low complexity: this allow them to be *fast enough* to react properly at the changes of the environment or to some error in perception in low dynamic environments. Apart from VO, the methods that account for the velocity of the obstacle present higher complexity. On the other side, VO is based only on linear velocities, i.e. does not properly considers the kinematic constraints of the vehicles.

2.2 Motivation

On the basis of the intuition of ICS, VO and DOV, we think that the explicit representation of obstacle velocity is necessary to perform safe and efficient navigation in high dynamic environments. None of the methods presents above however, accounts for the uncertainty in perception. In a real world environment, and especially in presence of dynamic obstacles such as cars, pedestrians or other robots, the perception based on a realistic sensor equipment is affected by incompleteness and uncertainty. The extraction of moving object, the approximation in a circular shape (hypothesis used in all the approaches that account for dynamics) and the estimation of the velocity of the obstacles are, for instance, sources of errors. We think that a reactive algorithm needs an explicit representation of the environment dynamics and should be aware of the uncertainty and incompleteness of its perception. A *realistic* sensor assessment presents:

- a limited range;
- hidden areas;
- a finite accuracy;
- obstacles shape uncertainty;
- moving obstacles detection uncertainty;
- velocity estimation uncertainty;

At the purpose of integrate the perception uncertainty in the motion decision process, we need

at first a model of the world that is capable to express all these different characteristics and that is general enough to be applied in different scenarios and with different sensor settings. Differently from the methods that ignore the time dimension, the perception we need must be able to integrate on-line the incoming information, and continuously reestimate the velocity of the obstacles and refine the perception of the static and dynamic environment. Secondly, we need a method to measure how the velocity of the obstacles and the uncertainty in the perceived world affect the safety of the robot in future time. Finally we need a way to choose the appropriate input at each time step.
Our model of the world will be represented by a Bayesian Occupancy Filter (BOF, [26]). The BOF is an occupancy grid where the time dimension has been added to the 2D state space: the occupation of the environment and the velocity of the obstacles are represented by discretized probability distributions in the space-time grid. As classic occupancy grids, the BOF is a very generic structure, adapted to both indoor and outdoor environments and to different sensor settings. Sensor fusion is performed through proper theoretical probabilistic models; a priori and on-line acquired information are integrated using a probabilistic framework. In this representation, the incompleteness and uncertainties coming from the observations are naturally handled and explicitly representing with the use of probabilistic models, which are more informative in comparison with the worst-case approaches and allow a more deep interpretation of the world and hence a more intelligent performance.
On the basis of the BOF, our work will be to quantify the effective risk of collision of the various configurations in the state space and use this information to choose a suitable control for the robot. As we stated in the previous section, among all the reactive approaches described, only the VO, the DOV and the ICS take into account the dynamic nature of the world. Among these three approaches, the Velocity Obstacle only presents a low complexity, which is a very important characteristic as we want to extend the method to a probabilistic model. In this chapter we explain how we adapted the Velocity Obstacle approach to the world representation given by the BOF. We detail a cell-to-cell approach based on the combination between the BOF and the VO to estimate the *probability of collision* in future time for the controls of the robot. We will then obtain a reactive navigation algorithm choosing an objective function and the next safe control for the robot.

2.3 Proposed approach: Probabilistic Velocity Obstacles in B.O.F

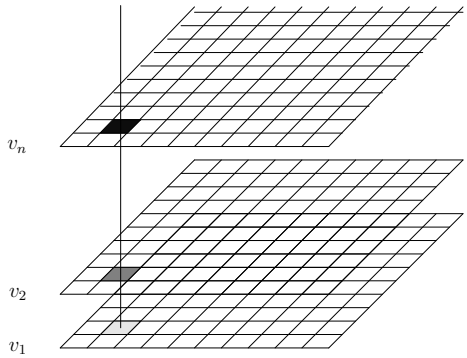
Paragraph 2.3.1 and 2.3.2 recall the previous work, describing respectively the Bayesian Occupancy Filter framework and the Velocity Obstacle approach. Paragraph 2.3.3 describes the generalisation of velocity obstacles to the cell-to-cell approach and how the probability of collision in time is computed, while 2.3.3 Paragraph 2.3.4, details how the control input is chosen to achieve obstacle avoidance.

2.3.1 Sensing Uncertainty: The Bayesian Occupancy Filter

The BOF is a discretized method to recursively estimate the state of a dynamic environment. The BOF is based on an occupancy grid where the velocity dimension has been added to the classical 2D representation of the Euclidean space. Since the BOF integrates dynamic obstacles, a prediction step need to be added to the classical occupancy grid updating iteration. The following paragraphs explain in more detail how the BOF is built and updated.

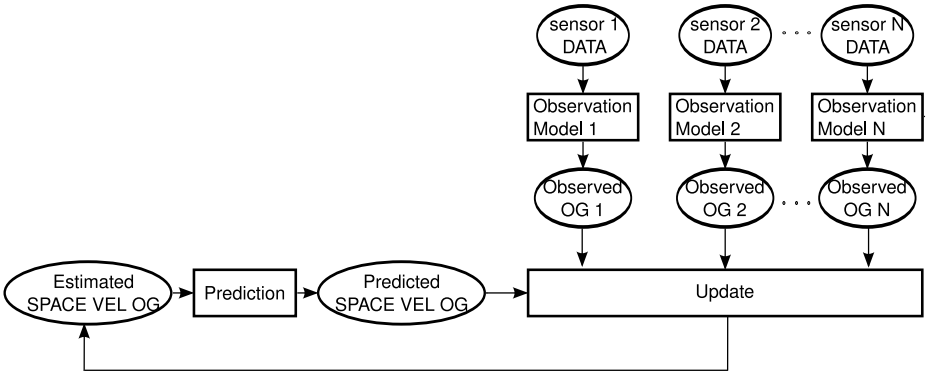
The space-velocity grid

Probabilistic occupancy grids are well known structures used for environmental representation. The 2D Euclidean space is divided in a finite number of cells, each representing a position in the plane. The state of the system $X(t)$ at time t is the list of the states of all the cells of the grid: Occ, when the cell is occupied or Emp if the correspondent space is free. Given a probabilistic sensor model $P(z(t)|x(t))$ where $z(t)$ is the current observation, the grid is updated following the Bayes rule. Under the hypothesis that each cell of the grid is statistically independent from its neighbourhood, each cell state estimation is updated independently [27]. To handle dynamic obstacles, each cell of the BOF maintains not only an estimation of its occupation probability, but also a discretized representation of the probabilistic distribution function (pdf) over velocities. A minimum and maximum velocity value is considered for eventual objects in the space, and the pdf is approximated by a finite histogram over uniform distributed velocity values v_n with $n = 1...N$. The discretization step is chosen according to the spatial and time discretization: given q the size of a cell and τ the time step, only integer velocities in terms of $\frac{q}{\tau}$ are taken into consideration. This choice is necessary in order to perform fast and rigorous prediction and updating steps and will be explained in the next paragraph.
The final grid can be seen as a tri dimensional grid where each slice represents a velocity value and the occupation of each cell represent the probability that the space is occupied by an obstacle at the correspondent velocity. Considering all the slices for a particular location, gives the probability of occupancy and the distribution probability over velocities for the space cell (see Figure 2.3.1).



The Bayesian Filter

Since the approach needs to handle moving obstacles, it is necessary to introduce a prediction step in the updating iteration of the classic occupancy grid. In the prediction step, the occupancy of



each cell is moved in according to the estimated velocity. In the estimation step, the data acquired from sensor readings are used to update the state of the grid.

Prediction step A constant velocity dynamical model is assumed. For each spatial cell $c = [i, j]$ and for each value of velocity $v_n = [di, dj]$, an antecedent cell is considered : $c_a(n) = [i - di, j - dj]$. Under the hypothesis that each cell is independent, the predicted occupation of each cell is computed as follows:

$$\hat{P}_c(Occ) = \sum_n P_{c_a(n)}(Occ) \cdot P_{c_a(n)}(v_n) \tag{2.1}$$

$$\forall n,\ \hat{P}_c(v_n) = \frac{P_{c_a(n)}(v_n)}{\sum_{n'} P_{c_a(n')}(v_{n'})} \tag{2.2}$$

The predicted probability distribution function of velocity of a cell c is obtained by a normalisation over all velocity probability values of each antecedent cell.

Estimation step Sensor data are acquired from one or more sensor and an observed occupation grid is built according to each observation model. The grid is finally updated following the Bayes rule:

$$P_c(Occ|z(t)) \propto P_c(z(t)|Occ) \cdot \hat{P}(Occ) \tag{2.3}$$

Usually only the occupation of the space is directly observed (using a distance sensor or a fixed camera); the pdf over velocities evolves in the prediction step (see equation 2.2) and modifies according to the occupancy observations. However, is the velocity is also directly observed an estimation step can be performed also for the velocity probability distribution function.

2.3.2 Velocity Obstacles

A Velocity Obstacle (VO) is defined as the set of linear instantaneous velocities of the robot which cause a collision with a given obstacle or a set of obstacles. A simple geometric method to compute velocity obstacles has originally been introduced by Fiorini and Shiller in [28] for the case of obstacles moving with constant linear velocity. In [29] the concept has been extended to obstacles with whatever trajectory, with the development of Non-Linear Velocity Obstacles (NLVO). In [22] the Probabilistic Velocity Obstacles (PVO) have been introduced, which can be used under velocity uncertainty and which provide a simple plateform for reactive navigation in uncertain environment, showing interesting results in reflexive or multiple agent coordinated navigation. Navigation algorithms based on velocity obstacles have been theoretically developed but no experimental results have been achieved.

Definition

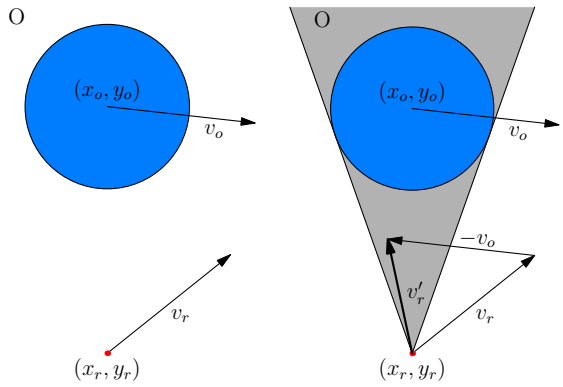


Figure 2.2: Collision Cone for a punctual robot and a circular obstacle with linear velocity v_o; v_r is in collision.

Lets consider a punctual robot r in $[x_r, y_r]$ free to move in the 2D plane, and an obstacle o of arbitrary shape, with centre in $[x_o, y_o]$ and constant linear velocity v_o. With these hypotheses, the configuration space of the robot $\mathcal{C}$ is equivalent to the Euclidean space. If the robot is circular with centre in (x_r, y_r) , the corresponding configuration space is given by a punctual robot in (x_r, y_r) and all the obstacles enlarged by the radius of the robot.
The velocity space is defined as the configuration space where linear instantaneous velocities are described by vectors attached to the centre of objects. The idea is to work in this space and determine the set of all linear velocities that lead the robot to a collision with the obstacle in future time:

$$VO = \{\overrightarrow{v}_r, \overrightarrow{v}_r \in \mathcal{V} | \exists t > 0, r(t) \cap o(t) \neq \emptyset\} \tag{2.4}$$

The Collision Cone CC_{ro} of the robot r relative to the obstacle o is defined as the set of all relative velocities $v'_r = (v_r - v_o)$ that leads the robot in collision with o in future time:

$$CC_{ro} = \left\{ v'_r | \exists t > 0, (x_r + v'_r \overrightarrow{i} \times t, y_r + v'_r \overrightarrow{j} \times t) \in o \right\} \tag{2.5}$$

where $\overrightarrow{i}$ and $\overrightarrow{j}$ are the unity vectors of x and y directions, respectively. The CC_{ro} is the positive angle with vertex in (x_r, y_r) and rays the right and left tangent to object o . From the geometrical point of view, to know if a velocity v_r is in collision with the obstacle o it is sufficient to verify if the relative vector $v'_r = (v_r - v_o)$ points in the CC_{ro} , i.e. check if the extension of the vector in the positive direction intercepts the obstacle. The velocity obstacle VO_{ro} is obtained translating CC_{ro} by the obstacle velocity v_o : all and only the velocities v_r pointing outside the cone are collision free. If more than one obstacle is present in the environment, it is sufficient to consider the union of each velocity obstacle [28] :

$$VO_r = \bigcup_{k=1\ldots K} VO_{ro_k} \tag{2.6}$$

Considering a single time t, the velocity obstacle $VO(t)$ relative to t is the set of linear velocities that lead the robot in collision with the obstacle at time t. From a geometric point of view, $VO(t)$ is the homothety of the enlarged obstacle with center in the robot position $r(0)$ and of scale $frac1t - t_0$. As the trajectories of the obstacles are not known up to infinite time, is usually not necessary to consider the VO as defined in equation 2.6, but only the subset that lead the robot to collision within a time horizon T_h. The choice of T_h should take under consideration the reliability of the obstacles model in future time and the kinematic constraints of the robot in the present state (time needed to stop). To entirely define the VO in $[t_0, t]$ it is sufficient to find its contours, that correspond to *contact velocities*. Geometrically, they are represented by the left and right contours of the collision cone defined in 2.5: these are given by the tangent lines from the robot center to the enlarged obstacles. In this lines (r right, l left) we can find the point that correspond to collision at the extremes of the chosen interval: $vo_r(t)$ and $vo_l(t)$. If the time elements are drawn into the cone, the contour of each homothety is divided in a lower part, nearer to the robot, and in an upper part. For $t = t_0$ the contour of the VO corresponds to the upper part of the edge of the homothetic transformation of the obstacle and is situated at infinite; for $t = T_h$ the contour of the VO corresponds to the lower part of the edge of the homothetic transformation of the obstacle in time T_h.

2.3.3 Cell-to-cell approach

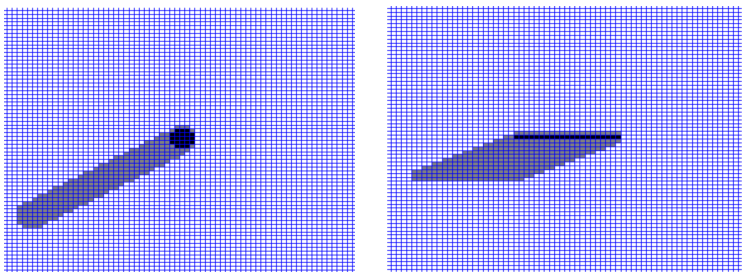
Lets consider a binary grid where the robot and the surrounding environment are represented. Each cell is labelled with a value $P(Occ) = 0$ if the cell is empty and $P(Occ) = 1$ if the cell is occupied by one obstacle. A special value is used to say that a cell is occupied by the robot $P(Occ) = R$. Also,

the velocity of each cell is deterministically known:

$$P_c(v_n) = \quad 1 \quad , \; if \; v_o = v_n \tag{2.7}$$

$$0 \quad , \; otherwise \tag{2.8}$$

The grid is relative to the robot. The robot and the obstacles are represented by connected clusters of occupied cells.
The velocities of the obstacles estimated by the BOF $v_n = [i \cdot \frac{q}{\tau}, j \cdot \frac{q}{\tau}]$ where $i, j \in \mathbf{N}$ are relative to the robot. So, the velocities v_r we study for the robot are actually accelerations with respect to the previous velocity values and are also integer values with repsect to the resolution of the grid: $v_r = [i \cdot \frac{q}{\tau}, j \cdot \frac{q}{\tau}]$ Following the framework detailed in the previous section, a velocity $v_r = [\Delta i_r, \Delta j_r$ leads the robot to a collision if it belongs to at least one of the $VO_{p_r c_o}$ between one of the points of the robot p_r and an occupied cell of the grid c_o. Given $[\Delta i_o, \Delta j_o]$ the velocity of an obstacle in the space, and $[\Delta_i, \Delta_j]$ an admissible velocity of the robot, each relative velocity $(\Delta' i, \Delta' j) = (\Delta i - \Delta i_o, \Delta j - \Delta j_o)$ is considered. This velocity corresponds to the vector attached to $p_r = [x_r, y_r]$ pointing $[x_r + \Delta' i, y_r + \Delta' j]$. As shown in Fig. 2.3(a), this velocity belongs to the VO relative to p_r if and only if there is at least an occupied cell with velocity $(\Delta i_o, \Delta j_o)$ in the positive direction of the extension of the velocity vector.



(a) (b)

Figure 2.3: Black cells represent the robot, while grey cells represent the SO for a given relative velocity and time interval. (a) circular robot (b) line robot.

To consider one whole cell of the robot, we make the union between the velocity obstacles generated by the points belonging to the cell. The contours of this set are the two parallel lines tangent to both the cells centred respectively in (x_r, y_r) and $(x_r + \Delta' i, y_r + \Delta' j$) in the positive direction. This region is the collision set of velocity v'_r relative to cell r and we denote it $CS(v'_r, r)$. All the cells that fall also partially in the collision set (and do not belong to the robot itself) have to be checked for occupancy. To compute the probability of collision of the relative velocity in an interval $[t_{k1}, t_{k2}]$, the set $CS_t(v'_r, r)$ of cells that should be traversed by the robot in the considered interval is computed. The distance covered by a robot cell in the considered interval in the velocity

direction is given by the norm of the velocity $\alpha = |\Delta' i| + |\Delta' j|$ times the length of the time interval $t_2 - t_1$. All the cells within the distance interval $[\alpha \times k_1, \alpha \times k_2]$ in the considered direction are checked for occupancy. To check the whole robot dimension this must be done for each cell of the robot. The considered velocity is in collision if it is found one occupied cell with velocity $(\Delta i_o, \Delta j_o)$ for at least one of the cell of the robot.

Compute the probability of collision

As detailed in §2.3.1, each cell of the BOF stores a probabilistic estimation of its state:

- a value of probability of occupation $P(Occ)$;
- a probabilistic distribution function on a histogram of possible velocities $P(v_n)$, $n = 1...N$;

Given a robot velocity v_r and an obstacle velocity v_n, the probability of collision of a cell r with a cell o in the $SO_t(v_{r'}, r)$ is computed multiplying the probability of occupation of the obstacle cell with the probability of the considered velocity: Considering all the cells of the robot, the maximum probability of collision in the considered interval $[t_0, t]$ is kept for each considered velocity:

$$P(t_{coll} \in (t_0, t] | v_r, v_n) = \max_{o \in SO_t} P_o(Occ) \cdot P_o(v_n) \tag{2.9}$$

where o is each cell in $SO_t(v_{r'}, r)$
To compute the probability of collision $P_{coll}(v_r)$ of the absolute velocity in the considered interval, all the possible velocities of obstacles have to be considered. This value is computed as follows:

$$P(t_{coll} \in (t_0, t] | (v_r)) = 1 - \prod_{n=1}^{N} (1 - P_{coll}(v_r, v_n)) \tag{2.10}$$

The previous equation is issued from the hypothesis that the probabilities of collision for different velocities are conditionally independent. This hypothesis is usually not fulfilled by the experiments: even if collisions with different objects can be considered independently, collisions probabilities with the same object at different velocities are correlated. This point is further analyzed in §2.3.3. The probability of collision in a time interval $(t_0, t_k]$ is the integral of the collision density over the interval. As we consider discrete time steps, we have $t_k = k * \Delta t$. Known the probability of collision in the interval $(t_0, t_{k-1}]$ and the probability of collision in a single time step $(t_{k-1}, t_k]$, the total probability of collision in $(t_0, t_k]$ can be computed recursively:

$$P(t_{coll} \in (t_0, t_k]) = P(t_{coll} \in (t_0, t_{k-1}]) + (1 - P(t_{coll} \in (t_0, t_{k-1}])) \cdot P(t_{coll} \in (t_{k-1}, t_k]) \tag{2.11}$$

with the hypothesis that $P(t_{coll} = t_0) = 0$. This estimate is again a conservative approximation that gives higher or equal probabilities of collision than if equations 2.9 and 2.10 are directly used

for the interval $(t_0, t_2]$. The probability of avoiding collision $(1 - P_{coll})$ falls exponentially in time. If there is not any obstacle in the perceived area, the rate of decay depends only on the characteristic of perception: in particular the probability of miss detection and false alarms and the dynamicity of the environment that is reflected in the probability of prediction error. When an obstacle or an hidden area is found in the direction of the considered velocity, the rate of decay changes and depends on the degree of belief of the obstacle presence.

Clustering

Classic dynamic environment representation involves the detection and tracking of moving obstacles. These approaches, usually based on a representation of the static environment and on a multi target tracking algorithm present some drawback in comparison with the BOF. The detection system of a multi target tracking algorithm must be usually trained to look for obstacles of a certain dimension, shape and velocity. Also, as the perception is supposed to be held from a moving sensor, static and moving object must be distinguished, which is not always a trivial problem. The association of an observation to one or more tracks, to a new object or finally to a false alarm is a complex task, especially in cluttered environments or if only low level information is available. The complexity scales with the number of the obstacles. In the BOF instead, there is no notion of objects: the complexity is independent of the number of objects in the environment and only depends on the resolution of the spacial and velocity discretization. On the other side, this means that we do not have any hypothesis of coherence among the cells. This makes the estimation of the velocity longer (in terms of number of subsequent observations) . Also, the velocity estimation is discretized, and the approximation can be worse than with multi tracking approaches. Different results can be obtained considering the correlation of the probabilities of collision given by cells that belong to the same obstacle. However, this information is not given by the grid itself. We need to cluster the cells in order to recognize the different objects. First, a double thresholding is applied in order to recognize free, occluded and occupied space. The occupied space is considered and a clustering algorithm is applied. We proceed in two steps:

- Position clustering: the near-by cells are connected using a 4 -connection (or 8-connection) recursive algorithm.
- Velocity Clustering: each cluster is checked for velocity coherence: if the velocity profiles of the cells belonging to the same cluster are too different, the cluster is divided again.

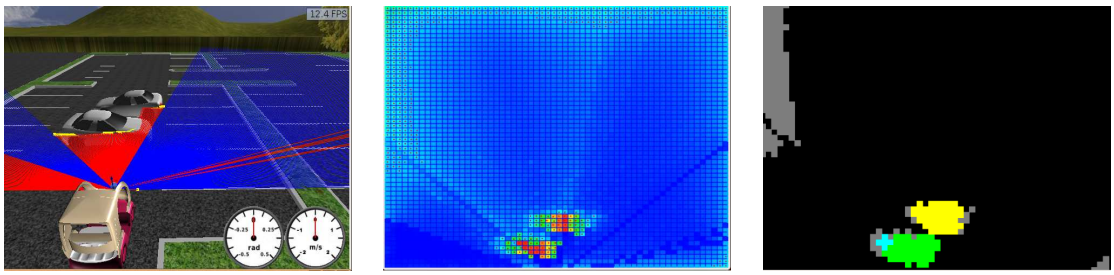
The second step allows a better performance of the algorithm in cluttered environment and in the case of low resolution. To perform the velocity clustering, the *distance* between the pdf of the velocities of the cell is measured. If the distance is bigger than a threshold, the cells are believed to belong to different objects and another cluster is initialized. A tracking algorithm can be applied: the center of mass of the clusters and their covariance are used as observations. The state of the

object is given by its position and velocity: the uncertainty in position is given by the occupancy probability of the cells themselves. The velocity uncertainty is estimated from the distributions in each cell. A grid is retrieved where each cell is labelled with the object it belongs to: a tracked object, free space, hidden space. The computation of the probability of collision which takes into account the object information can be performed as follows: Equation 2.10 is substituted by equation 2.13 or each object k:

$$\sum_{n=1}^{N} P(v_n, k) = 1 \tag{2.12}$$

$$\sum_{n=1}^{N} P_{coll}(v_r, v_n, k) \leq 1 - \prod_{n=1}^{N} (1 - P_{coll}(v_r, v_n, k)) \tag{2.13}$$

Equation 2.12 represents the integral over the set of possible velocities of object k; In equation 2.13, the first term is the correct estimate of collision probabilities due to object k given its pdf over velocities; the second tern is the estimate considering each velocity independently. The occluded space and free space are taken into account using equation 2.10. Fig. 2.4(a) shows a simulated environment: the Cycab is equipped with a laser range finder and perceives two cars: the nearest moving from left to right and another just behind, moving from right to left. Fig. 2.4(b) shows the dynamic occupancy grid computed: red stays for high probability of occupation, while blue is for low probability. Fig. 2.4(c) shows the clusters found on the grid, correspondent to the two cars.



(a) (b) (c)

Figure 2.4: (a) Two moving cars are perceived: the nearest one goes from left to right, the back one goes from right to left. (b) The perceived occupancy grid and mean velocity estimated in each cell. (c) The clusters obtained: black is free space, grey is occluded space, the two moving objects are extracted.

2.3.4 Navigation Algorithm: Choice of the control input

The aim of the reactive navigation algorithm is to find at each step the control that minimises the risk of collision and leads toward the goal position. An objective function is defined, which takes into consideration the difference between the robot motion direction and the goal direction and the

value of the linear velocity applied.

$$K(v) = \alpha \cdot |v| + \beta \cdot h(v, goal)$$

In the simple case, the function $h(v, goal)$ is just the difference between the velocity direction and the direction of the goal; in presence of local minima, however, or if some different optimisation parameter is considered, a different distance function could be defined in a previous phase of motion planning. The constants α and β are empirically chosen.
Different strategies can be employed to choose the next control and to set the trade-off between safety and goal convergence. We consider the robot safe if it can stop before running into a collision. This means a velocity v can be applied for an interval ϵ if the robot will not run into collision up to:

$$T_{safe}(v) = \epsilon + T_{brake}(v) \tag{2.14}$$

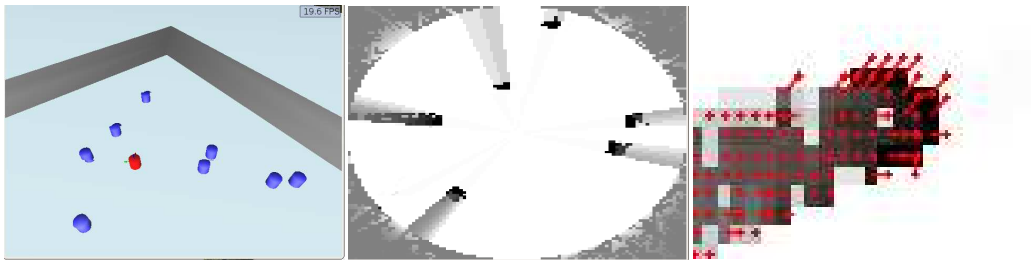
where $T_{brake}(v)$ is the minimum time to stop applying the maximum negative linear acceleration. One solution is to choose a time horizon T_h and set a probability threshold P_{safe}. In this case only the velocities that satisfy $P(t_{coll} \in (t_0, T_h]) < P_{safe}$ are retained and the one with bigger utility function is chosen. However, the choice of T_h and P_{safe} are arbitrary. If T_h is too high, some safe maneuvers could be forbidden; if it is too low, the robot could choose a dangerous maneuver. Likewise for P_{safe}. Also, in the emergency case when no velocity is safe enough, the robot would perform an emergence maneuver (braking) without any further information. The method described in the previous section gives us a tool to compute the probability of collision in time for admissible linear velocities of the robot. To choose the next control, the maximum value of T_{safe} for the discretized set of reachable velocities of the robot is found and for each reachable velocity the probability of collision is estimated in the interval $(t_0, t_0 + \max_v(T_{safe}(v))]$. As the BOF is estimated in the local grid, the current velocity of the robot must be added to that of the obstacles to find the true value. The probability of collision is computed separately for timesteps between $T_{safe}(V)$ and $\max_v(T_{safe}(v))$ and a new objective function is considered:

$$K^*(v) = K(v) * ((1.0 - P(t_{coll}(v) \in (t_0, t_0 + k\tau])) \cdot \frac{k\tau}{\max_v(T_{safe}(v))} \tag{2.15}$$

In this way, velocities that are safer for longer time are preferred, and the minimum time considered is their safe time. The algorithm is repeated at each time step, so that the robot is reactive to new obstacles and obstacles changing velocity.

2.4 Examples and Performance

Here we show some example of the algorithm in simple simulated environments. We first isolate different problems with the purpose to give an objective evaluation of the performance of the approach under different environmental situations and with different perception conditions and then show a simulation of the navigation algorithm. We consider a circular holonomic robot equipped with a 360° laser range finder and perfect localizaton in the space as in figure 2.5(a) (red object). The environment is formed by static obstacles of rectangular shape (grey objects) and moving circular objects with the same size of the robot (blue objects). Since we are working with a probabilistic representation, each cell has in general a positive probability of occupation: the free space scanned by the sensor is characterised by $P(Occ) \simeq 0$ while not sensed environment has $P(Occ) \simeq 0.5$ (Fig. 2.5(b) For what concerns velocities, on each cell we will have a pdf more and more extended as the prediction is less reliable: in this example, we considered a set of discretized velocities in the interval $v_x = [-3, 3]m/s$, $v_y = [-3, 3]m/s$ (Fig. 2.5(c))).



(a) (b) (c)

Figure 2.5: (a) holonomic robot in red, circular moving obstacles in blue and static obstacles in grey. (b) The estimated spatial grid; (c) Detail of estimated velocities.

For the space out of the local grid $P(Occ) = 0.5$ and a uniform pdf over velocities are considered. The following figures show the occupancy grids estimated by the robot under different perception conditions.

Limited Range

Figure 2.6(a) shows perception of an empty environment with a limited distance range. A maximum range of $20 \cdot q$ is considered. The algorithm run for a time long enough to stabilize the velocity distribution on the occluded cells, so that the mean and expected velocity in the distribution is zero. Figures 2.6 (b) and (c) show the results of probability of collision for increasing time instants considering linear velocities. Figures 2.6 (d-f) show the same setting in a scenarion with a static obstacle. The probability of collision is bigger for larger velocities in each direction and it grows with time: also if the robot stands still ($v_x = v_y = 0$) the probability of collision grows with time as

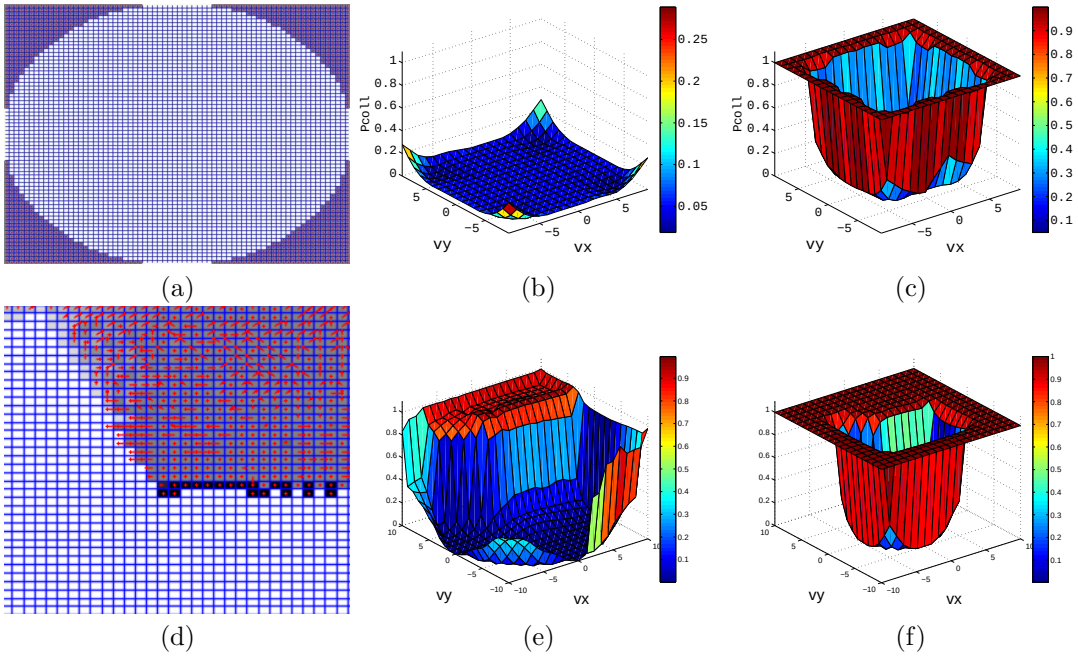


Figure 2.6: (a) Simulated occupancy grid: the robot in the centre perceives free space all around, with limited range. (b) The probability of collision for each velocity of the robot considering k=3, (c) k=5; (d-f) Same pictures in case that the robot perceives an obstacle.

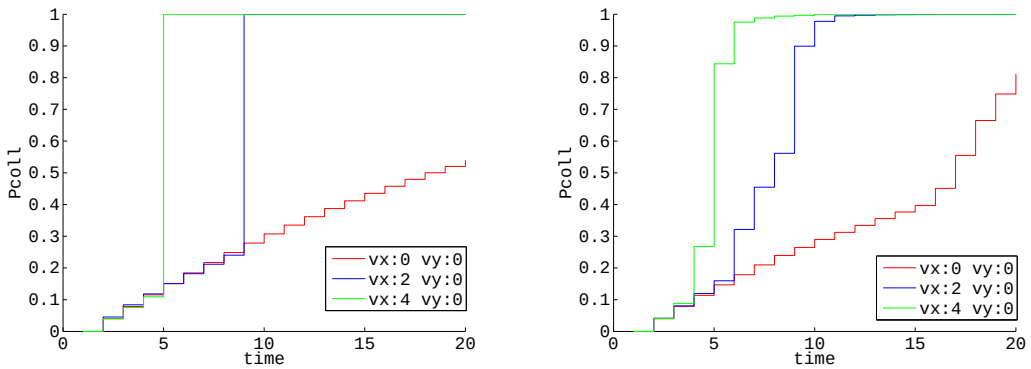


Figure 2.7: Probability of collision in time obtained with 10% probability of false alarm and of miss detection: (a) 5m far from a static obstacle (b) 5m far from an obstacle with uncertain velocity.

the hypothesis that some unseen obstacle could go toward the robot is considered. The uncertainty about obstacles velocity plays an important role in the increase of probability of collision in time: as shown in Fig.2.7 (a) if there is not any obstacle in the perceived area, the rate of increase of the probability of collision only depends on the characteristic of perception: in particular the probability of miss detection and false alarms and the dynamicity of the environment that is reflected in the probability of prediction error. When an obstacle or an hidden area is detected in the direction of the considered velocity, instead, the rate of changes in time and depends on the degree of belief of the obstacle presence and on the distribution of its velocity (Fig.2.7 (b))

Clustering

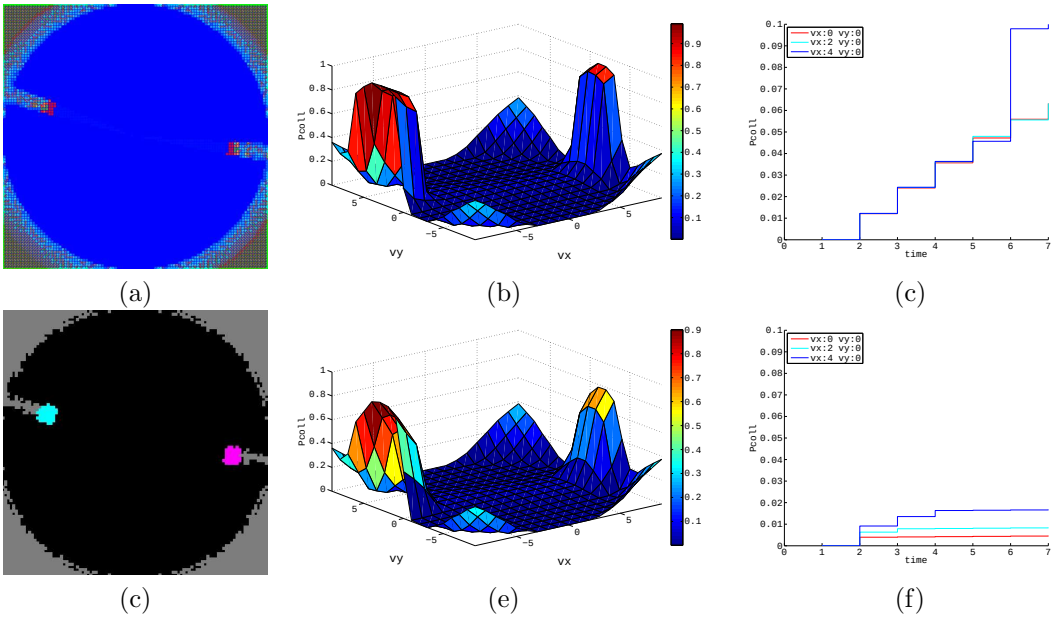
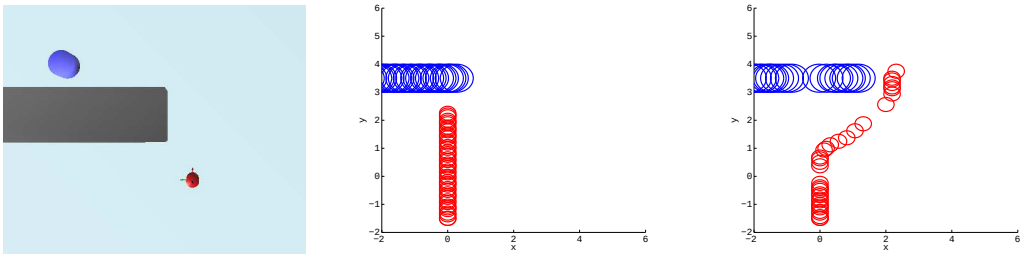


Figure 2.8: Comparison between perception and probability of collision in case of not clustered grid (a-b-c) and in case of clustering (d-e-f).

Fig. 2.8 show the results of the probability of collision in time in case of non clustered grid (Fig. 2.8 (a)) and in case of clustered grid (Fig. 2.8(d)). The probability of collision due to the obstacles are lower when clustering is applied: this is due to the fact that correlation between cells is considered and that velocity is better estimated. For the same reason, the evolution in time of the probability is steeper in the case of non clustered grid (Fig. 2.8(c) and (f)). The probability of collision due to *free* and occluded space is the same (as in the corners of Fig. 2.8(b) and (e)).

Occlusion

The following example shows the trajectory of the robot in the case of a moving obstacle hidden by a wall. Fig. 2.9 (a) shows the simulated scenario. The obstacle is moving to the right , while the robot



(a) (b) (c)

Figure 2.9: (a)The obstacle moves to the right, the robot goal direction is north: (b) the trajectory followed when there is no occlusion (c) the trajectory followed in case of occlusion.

goal is up in the vertical direction. Fig. 2.9 (b) shows the trajectory followed by the robot when if the wall is removed and the obstacle is perfectly visible. The robot follows a straight trajectory, adapting its velocity to the one of the obstacle. Fig. 2.9 (c) shows the trajectory followed by the robot when the obstacles come out suddenly from the occluded space. The robot moves to the right *before* it sees the obstacles: it does that because of the uncertainty in velocity of the hidden space. Then it accelerates to pass in front of the obstacle.

Velocity uncertainty

Fig. 2.10 shows another scenario and the results obtained with different characteristic in perception. In figure (a), the robot has a good estimation of velocity and a large visibility range. In this case the robot performs the obstacle avoidance passing near the obstacles and reaching the goal with a short trajectory. In figure (b) a Gaussian uncertainty on obstacles velocity is considered. Since the beginning the robot tries to keep its trajectory further away from obstacles; the path results longer as the robot performs wider curves to avoid collisions. In figure(c) the visible range is short($30 \cdot q$) and the occluded zones hide obstacles and their shape. The robot goes slower; as it approaches the crossing it still doesn't see the obstacle on the left and enters the crossing; when it perceives the obstacles it is forced to escape from it and reaches the goal only after waiting the right obstacle to leave the crossing. Tests with a lower range cause the robot to perform an emergency maneuver (brake and stop) before facing the crossing.

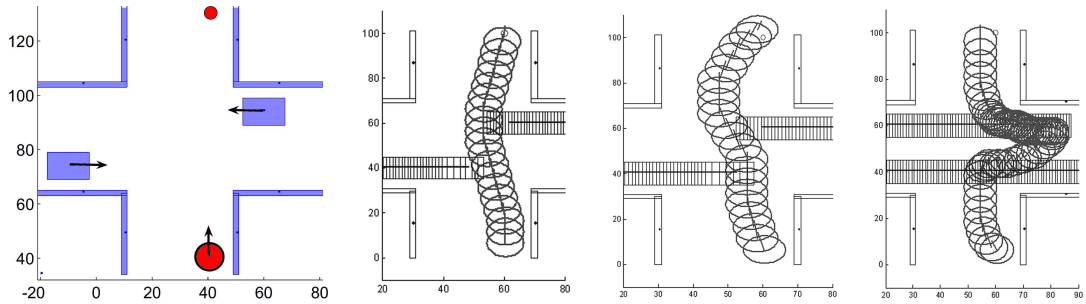


Figure 2.10: (a) A cross-road scenario. Trajectories followed: (a) with long visibility range and good velocity estimation (b) with uncertainty in the velocity estimation (c) with short visibility range.

2.4.1 Complexity considerations and simplifications

The complexity to compute the probability of collision of each velocity v in a time interval $(t_0, t_k]$ depends on:

- the ratio between the dimension of the robot and the discretization step of the spatial grid;
- k: the number of timesteps to be considered for each velocity;
- N: the number of possible velocities for the obstacles.

The complexity of the algorithm can be fully determined of line and does not depend on the environmental conditions (number of obstacles). The set of cells to check for each relative velocity in each time step $SO_{t_k}(v'_r)$ can be determined a priori and off-line. Each velocity can be studied independently and parallelization is possible in this phase. Also, is possible to use a safety threshold in the probability of collision and stop the computation when its probability of collision is bigger than the threshold. For what concerns the number of velocities to study at each iteration, the search space is reduced to the velocities that can be reached within the next time step. This *dynamic window* is centred around the actual velocity of the robot and is limited by the maximum acceleration that can be exerted exert and the maximum allowed velocity in function of the direction.

2.5 Conclusions

In this chapter we proposed a method to compute the probability of collision in time for linear velocities of an holonomic robot and a reactive algorithm to perform obstacle avoidance in dynamic uncertain environment. The novelty of the method consists in the explicit consideration of uncertainty in the perception system, rising both from the errors and noise in the model of the environment and from occlusions, sensor range, noise and failures. The input to the algorithm is a space-time

occupancy grid, which is highly reactive to the environmental changes and is well suited to be applied in various sensor settings. The developed algorithm computes a probability to collision in time working directly in the . The dynamic and kinematic constraints of the robot are so taken into account and the study is reduced to the current reachable velocities. We considered linear velocities and an holonomic robot. However, we can easily generalize the approach to robots that move along straight trajectories but that are constrained in linear acceleration and changing direction rate. This is achieved by reducing the set of admissible velocities. Another important class of robots can be represented by robots moving along arc of circles. For this class, the controls are couples (v, ω) with v the linear velocity and ω the angular velocity. The generalization to robots moving along arc of circles requires then some approximation. With the hypothesis of a short timestep and limited velocity, arcs of circle can be approximated with straight lines. The linear control correspondent to (v, ω) is given by:

$$v_x = -v \cdot cos(\omega \frac{\tau}{2}) \tag{2.16}$$

$$v_y = v \cdot sin(\omega \frac{\tau}{2}) \tag{2.17}$$

where τ is the timestep.

For what concerns the navigation algorithm, the simulation results show that the robot is able to navigate among static and moving obstacles facing unexpected situations and moving toward the goal. The robot adapts its behaviour to the quality of information received and modifies its trajectory according to the incomplete and uncertain perception of the environment. In particular simulation experiments show that the algorithm is able to represent explicitly the uncertainty and face problems due to short range, occlusion and velocity uncertain estimation. With some difference in the accuracy of performance the algorithm is able to maintain the safety issues and work without an explicit object representation. This enhances the qualities of the method as it is able to work independently from a target tracking algorithm, which could introduce false alarms, miss detection and velocity estimation errors and needs a high computational power with respect to the raw data. We have shown how the clustering on the grid introduces a more accurate estimation of object velocity, which could improve the final navigation performance.

However, due to the very reactive nature of the algorithm and to the limited knowledge of the environment, there is no guarantee that the robot achieves the goal or that it doesn't put itself in emergency conditions that could have been avoided with a deeper interpretation of the world. As other reactive algorithms, our approach is subjected to **local minima** problems: the minimal cost at a certain time step can be given to a command that is optimal locally but that produces suboptimal global behaviour. U-shaped obstacles for example, represent a classical trap for robots driven by reactive methods (see Fig. 2.11. Another problem comes from the perception of velocity. While the presented method can be used independently from a target tracking algorithm, with the

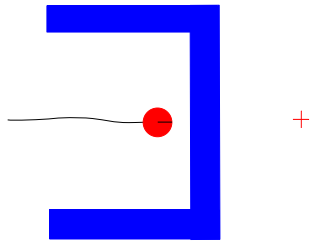


Figure 2.11: The robot is trapped in a local minimum.

need of a-priori models, the complexity and errors that these approach can introduce in the process, the concept of "object" can be useful in the determination of velocity and can lead to more complex and realistic motion models than the constant velocity one. Also, the use of the BOF requires the velocity of the obstacles to be discretized, with a loss of accuracy that influences the performance of the robot.
In order to achieve a better performance, we should integrate the information of the probability and time to collision in a motion planning algorithm able to face more complex scenarios, combining a priori knowledge and on-line perception.

Chapter 3

Motion Planning in changing environment

In the previous chapter we saw how reactive approaches are effective in dynamic environments; however we are aware that they present convergence and safety problems and lead to suboptimal solutions. Hence, we move our attention to motion planning strategies. The purpose of motion planning is to use the prior knowledge to choose the best set of actions to take in order to accomplish a task from the present state. It is intuitive then, that the *quality* of the plan strictly depends on the quality of the prior knowledge: if we do not know the present state or the future, choosing an appropriate plan can be impossible; if the situation changes the plan can become infeasible. With [30] we can affirm that the more incomplete the prior knowledge, the less important the role of planning.

In Section 1.1.3 we classified the sources of uncertainty in four classes: Configuration Sensing (CS), Configuration Predictability (CP), Environment Sensing (ES) and Environment Predictability (EP) uncertainty. A real-world system is usually affected by all four of these sources, but depending on the quality of the mechanical system (sensors, actuators) and of the complexity of the environment and the corresponding models, one or the other source affects the navigation problem with more or less importance.

In this chapter we will address the problem of motion planning in *changing* environments. In the context of navigation, we call an environment *changing* when the mapping relative to our application changes in time. Considering a static environment, when some unexplored zone is observed, free areas and new obstacles can appear: the information changes during navigation. Within a dynamic environment instead, the information changes also independently of the robot motion: an unknown dynamic environment presents more strict constraints in terms of real-time decision and time of reaction than an unknown static environment.

In this chapter we introduce a more looking-forward strategy with respect to purely reactive approaches in order to better integrate the *a priori* knowledge with the on-line perception. We will face the problem of a robot navigating among moving obstacles, taking into account environment sensing uncertainty and environment prediction uncertainty. The static environment is perceived from a local point of view, and the information is used to estimate a global occupancy grid. The moving obstacles are detected and tracked: the prediction of their future trajectory is based on a short-term motion model estimated on line by the tracking algorithm.
In section 3.1 we recall the basic planning problem and the algorithms developed to address it. In section 3.2 we give a detailed formulation of our problem and present the choices made to address it. Section 3.3 presents the developed approach, an extension of a well known sampling based method to the case of a probabilistic description of the configuration-time space; we introduce our approach to handle changing and dynamic environments, and how the algorithm proposed is adapted to integrate new incoming information. Section 3.5 ends the chapter with examples and simulation results.

3.1 State of the art

The classical motion planning problem is generally defined as follows:

> *Given a robot $\mathcal{A}$ that moves in a workspace $\mathcal{W}$ containing obstacles, an initial configuration q_{init} and a goal configuration q_{goal} find, if it exists, a feasible continuous sequence of configurations starting at q_{init} and terminating at q_{goal}. If such a solution does not exist, tell it in finite time.*

The path is then defined as the continuous function from q_{init} to q_{goal}. The path is always *feasible* if the robot is a "free flying object". However, if the motion of the robot is constrained (*eg* by non-holonomic constraints...) the problem becomes much more complex and the distance between configurations should be designed to represent the admissible controls needed to pass from one configuration to another. The basic formulation of the problem considers only static obstacles and a perfect knowledge of the workspace. The general approach to solve the problem is to consider the configuration space $\mathcal{C}$ and to distinguish in it the space of free configurations $\mathcal{C}_{free}$ and the space of configurations that are in collision with some obstacle $\mathcal{C}_{obs}$.
The complexity of general classes of planning problems has been theoretically faced and solved since the early $'80s$. In [31] it has been shown that the 3 dimensional planning problem with an arbitrary number of obstacles is PSPACE-hard if the velocity modulus is bounded, NP-hard otherwise. It has successively been shown [32] that the problem with bounded modulus is also NP-hard. A problem is said to be NP when it could be solved in polynomial time by a *nondeterministic Turing machine*, or equivalently if all the positive solutions can be verified in polynomial time given the right information. As it is unknown if the NP class is equal or bigger than the P (polynomial time) class, it is impossible to say if a polynomial algorithm to solve the problem exists. NP-hard means

that NP is a lower bound class, i.e it means that the problem is at least NP, but it is not known if the problem also belongs to NP or to a more complex class. It then appears unavoidable that a complete general algorithm would require at least a running time exponential in the dimension of $\mathcal{C}$. On the other side, the existence of complete algorithms to solve the planning problems give an upper bound for the complexity of the problem. In [33] a general algorithm was developed whose running time is doubly exponential in the dimension of $\mathcal{C}$; this bound was lowered by Canny's algorithm described in [32] whose running time is exponential in dimension [34].

3.1.1 Planning in known environment

We can distinguish two main classes of path planning approaches: **combinatorial** and **sampling based** approaches.
Combinatorial algorithms are complete algorithms that find paths through the continuous configuration spaces without approximation. The general idea is to find a way to capture and represent the connectivity of $\mathcal{C}_{free}$, so that given an initial and a goal configuration, if they can be connected through $\mathcal{C}_{free}$ a solution is found, otherwise it does not exist. These methods are said to be *complete* as they are able to find a solution or to tell if a solution does not exist in finite time. The problem of combinatorial algorithms is that an exact and explicit representation of $\mathcal{C}_{obs}$ is not always simply achievable, especially in high dimensional spaces and in constrained systems (non-holonomic etc..). These approaches can reveal too complex to be applied to problems with high dimensionality: this is the case of manipulators with many degrees of freedom, mobile robots in large environments, multiple coordinated robots. Since the most efficient general algorithm ever developed is exponential in the dimension, complete algorithms can be effectively used only for the simpler problems. On the other hand, more efficient algorithms have been punctually developed for specific problems [30].
At the beginning of the $'$90s another family of approaches have begun to be developed: sampling-based methods. The basic idea of sampling-based methods is to avoid the explicit construction of $\mathcal{C}_{obs}$ and conduct the search among $\mathcal{C}$ with a sampling scheme. A collision detection module probes the explored space; if the space is free, the configuration is recorded and the algorithm attempts to connect it to the space already explored. Sampling-based methods are more general than combinatorial algorithms, but are not complete due to the fact that the space is not systematically represented. Two new weaker notions of completeness are considered:

Resolution Completeness: If a solution exists, the algorithm finds it in finite time; if a solution does not exist, the algorithm may run forever.

Probabilistic Completeness: If a solution exists, the probability to find it in finite time converges to one.

The algorithms which present resolution completeness are deterministic algorithms which sample the space densely: as the number of iterations tends to infinity, samples come arbitrarily close to

any configuration. Random-sampling algorithms instead, can only be dense with probability one, and are probabilistically complete. The success of these methods is mostly due to their simplicity of implementation and the fact that, thanks to the high computational power of today's computers, they can solve most of complex and high dimensional problems in reasonable time.

Combinatorial Motion Planning

- **RoadMaps** The idea is to capture the connectivity of the configuration space with a network of one dimensional curves (*roadmaps*). Various methods have been proposed to build the initial roadmap: visibility graphs [35, 36], Voronoi diagrams [37], freeway nets [38], silhouettes [32]. Once the initial map has been built, the algorithm can satisfy multiple queries: at each query, the initial configuration and the goal are connected to the net; if a path can be retrieved through the roadmap, the solution is found; otherwise the algorithm reports failure.

- **Cell Decomposition** [33, 39, 40] $\mathcal{C}_{free}$ is exactly partitioned into a set of non overlapping regions called cells. The adjacency relation among the cells is represented in a connectivity graph. Given a planning problem, the connectivity graph is searched for a *channel* from the cell containing the initial configuration to the cell containing the goal configuration.

- **Potential Fields** [41, 42] These methods are an extension of the reactive navigation methods described in Chapter 2: the robot is regarded as a particle under the influence of artificial forces that attract it to the goal and repel it from the obstacles. The previous knowledge of the environment is here used to design potential functions without local minima, when possible.

Sampling Based Motion Planning

- **Discretized** $\mathcal{C}_{free}$ [43, 44] The method consists in discretizing the configuration free space with a set of cells. A discrete search is then conducted incrementally from the initial configuration with a general search algorithm (A^*). The success of the search can depend on the resolution of the grid. Thus, if a solution is not found, the search can be repeated refining the resolution.

- **Randomized Potential Fields** [45, 46] The algorithm is issued from the potential field context for reactive navigation which has been described in Chapter 2. In this case the search proceeds incrementally; if a local minimum is detected, random walks are used to escape it.

- **Probabilistic RoadMaps** (PRM) [47] In a preprocessing phase, a dense sequence of samples is drawn in $\mathcal{C}$. Each sample that does not belong to $\mathcal{C}_{obs}$ is stored and the algorithm tries to connect it with the roadmap using a simple local planner. In the query phase q_{init} and q_{goal} are processed. If they can be connected with the roadmap, a search is performed for the

sequence that connects these two vertices. If they cannot be connected, it is not possible to tell whether a solution exists.

- **Ariadne's clew** [48] The basic idea is that a compromise must be made between finding an optimal solution and global search. The algorithm grows a tree of configurations alternating an explore and a search phase. In the explore phase, the algorithm randomly selects an explored node and finds the furthest configuration that is possible to connect with a simple local planner. In the search phase, the algorithm tries to connect the new node with the goal.
- **Rapidly-Exploring Random Trees** (RRT) [49] Inspired by the precedent algorithm, this method grows a search tree and periodically tries to connect it to the goal. A configuration is sampled in the space according to a predefined or a random sequence. Then, an attempt is made to extend the tree toward the sampled configuration. Periodically, instead of a random sample, the goal is considered.

Planning in a known dynamic environment

One extension of the basic problem is to consider moving obstacles in the workspace. In this first extension, the knowledge of the workspace is still considered perfect: the shape, position and trajectory of the moving obstacle are known. In this case, the problem cannot be solved just by choosing a sequence of configurations, but the time dimension must also be considered. The plan must specify the configurations to be attended at specific time instants. The new space of search is the configuration-time space $\mathcal{CT}$: in this space, the obstacles are represented by static regions. The approaches presented above can be easily extended to this space; however, since time is not reversible, the planning method must take this new constraint into account. Also, if kinematic and dynamic constraints are considered, they must be translated into geometric constraints of the path along the time dimension.

3.1.2 Planning in an uncertain environment

The problem we are going to face in this chapter is the uncertainty about the geometry, the location and the trajectory of the obstacles. This means we are considering only the Environment Configuration and Environment Prediction uncertainties which come from perception and we discard the uncertainty coming from the localization and execution problems. Independently of whether there are moving obstacles in the environment or not, we can distinguish two cases. If the uncertainty is bounded and low a worst-case approach is suitable: the obstacles in $\mathcal{C}$ or in $\mathcal{CT}$ are just enlarged by the amount of the related uncertainty and the deterministic approaches presented above can be applied without modification.
If the amount of uncertainty is more important, it is possible that all feasible solutions are lost. In this case, the robot needs to obtain information using its sensors at execution time. This is the

case of both the exploration problem in static environment (where the robot is supposed to have little or no *a priori* knowledge about the workspace) and in the navigation problem in dynamic environment (where the robot does not know the future trajectory of the moving obstacles). If the state space is discretized and the number of states reachable by the system is finite and tractable, it is possible to address the problem by the use of complete discretized approaches. Accordingly to the representation and the degree of uncertainty, different approaches are suitable:

Complete algorithms in discrete $\mathcal{C}$

- **Feed-back plans** In feed-back plan approaches uncertainty is not explicitly modeled. Instead, the fact that the future state may be unpredictable is accounted for by ensuring that the robot knows which action to apply from each state of the system. The solution is called *strategy* and is a function γ from the state space directly to the control space: $\gamma : \mathcal{S} \rightarrow \mathcal{U}$. In this way, whatever the next state of the system, the robot will have computed in advance the best action to take.
- **Markov Decision Processes** (MDP) In the game theoretic and MDPs approaches, the uncertainty is modeled as an action taken by a particular agent that can be called *nature*. This action influences the next state of the system and is unknown. The state of the system is perfectly observable at each instant. The problem is then modeled as a sequential game against nature. A reward (or a cost) is assigned to each state, giving advantage to the goal and penalizing any failure state and can depend on the nature action/uncertainty. If probabilistic uncertainty is used, the problem is referred to as a Markov Decision Process (MDP). A weighted graph can be obtained using the probabilistic motion model from each state. Solving the problem means finding a policy that maximizes the expected reward. This can be done applying graph search algorithms, like A^* and value iteration.
- **POMDP** If not only the outcome of an action is unknown, but also the state of the system is not perfectly observable, the problem can be formulated as a Partially Observable MDP (POMDP). The algorithms used to solve the MDPs, however, cannot be easily generalized to this new class of problems; also, the complexity of the problem scales exponentially with the number of states. For instance, task-dependent and approximate solutions have been proposed in [50].

Complete algorithms in discrete $\mathcal{C}$ need the state of the system and the control space to be discretized and their complexity scales badly with the number of possible states: in practice these algorithms are used only for low dimensional problems.
When the *a priori* knowledge on the workspace and on the obstacles motion model is too small, the complexity of the problem is too high to be addressed with a complete algorithm in reasonable time. In this case the robot may interleave planning and execution phases. If the world is static,

the algorithms proposed for the basic formulation of the planning problem can be easily extended: the robot computes an initial path with the expected state of the world and it begins to execute the path; when something different or unexpected is perceived it can stop and has time to plan a new path according to the new model of the world. In order to slim the replanning phase, some algorithms have been proposed to take advantage of the previous planning phases and to optimize the new search modifying the plan locally, where the unexpected change has occurred. In the case of a finite and discretized state space, a complete algorithm can be applied:

- **D*** Proposed by Stentz A. in [51] the algorithm is a generalization of the A^* search algorithm in the case of partially known environments. In a finite discretized configuration space, an initial cost to pass from a configuration to a near one is given. A path is initially found and the robot begins to execute the path. If a change in one cost is detected, only the affected configurations are considered and the optimal path is updated in a reduced time. The algorithm finds the optimal path with the given information.

Sampling Based algorithms

When the state space is large or unbounded and when the discretization is difficult to apply, as in the case of non-holonomic robots, sampling based algorithms can be used:

- **Anytime RRTs** Proposed in [52], the algorithm plans a complete path at the beginning using the classical RRT method. If the path is invalidated by the observations at execution time, the tree (which is maintained in memory) is locally "repaired": the invalid nodes are deleted, while new configurations are searched to repair the tree from the root to the branches that go toward the goal but have been isolated by the unexpected obstacles.
- **Partial Motion Planning (PMP)** [9] The algorithm takes explicitly into account the time constraints and the safety issues to which the decision process is subjected in a dynamic environment. A tree is grown using a classic sampling based algorithm, but a node is added to the tree only if it is not an Inevitable Collision State (see §2.1). In this way, PMP is able to give a safe partial path at anytime. During execution of the partial path selected, another partial path is elaborated starting from the end of the previous elaborated path.
- **Particle-RRTs** The algorithm extends the RRT algorithm to an uncertain environment propagating the uncertainty into the planned path. The tree is extended with different likely conditions sampled from a probabilistic model, as in particles filters. Each vertex of the tree is a cluster of the simulated results. The likelihood of successfully executing an action is quantified and the probability of following a full path can be determined. The practical implementation addressed in [53] considers a robot in an environment with uncertain slippage.

In Anytime RRTs, the time needed for initial planning and replanning is not strictly bounded. This is not a strict constraint in static or quasi-static environments, but becomes a vital parameter

in dynamic environments. When considering an uncertain dynamic world, the robot must react to the changes before putting itself and people or vehicles in danger. The PMP approach takes the time constraint explicitly into account: renouncing to completeness properties, the algorithm computes a safe partial trajectory in the available time. These two sampling based methods are however based on a deterministic representation of the world; there is still little work on how to face prediction uncertainty in large state spaces, and only recently some attempts have been done to extend sampling-based approaches to continuous state spaces with probabilistic uncertainty, as in the case of Particle-RRTs.

3.1.3 Comparison

The path planning methods presented mainly rely on deterministic representations of the world. If they are used in a dynamic environment, either it is assumed that the trajectory of the moving obstacle is known *a priori*, or the path planning is combined with a reactive method to face the unexpected changes of the environment. In the case of deterministic representation of the future trajectory of the obstacles, the problem of uncertainty is completely skipped: either the information gathered is assumed as *true* or a worst-case approach is used; in this last case a maximum boundary for uncertainty is determined on the basis of the *a priori* information, such as the motion capabilities of the obstacles. While a probabilistic model requires much more information than a deterministic one (since it is based on statistics), it is also much more expressive and it is able to *quantify* the risk of the robot in each configuration. The behaviour of worst-case planners can be imitated in probabilistic approaches just by assuming a threshold in the probability values.
Among the algorithms presented above, we compare in this paragraph only the algorithms used for planning in uncertain environments (presented in section §3.1.2): table 3.1 shows the main approaches, if they are able to represent (+) the various kind of uncertainty or not (-) and their answer in terms of time constraint, i.e. if they are adapted not only to the general unknown environments but especially to navigation among dynamic obstacles.

To use complete algorithms, the state space of the robot (configuration and controls) has to be

		ES	EP	CS	CP	Dynamic
Complete	D*	+	+	-	-	-
	MDP	-	+	-	+	-
	POMDP	+	+	+	+	-
Sampling Based	AnytimeRRTs	-	+-	-	-	-
	ParticleRRTs	+	-	-	+	-
	PMP	-	+-	-	-	+

Table 3.1: Comparison between planning methods.

discretized. The complexity of the algorithms scales with the dimensions of the environment, which

can prevent them to be used among moving obstacles. In particular the POMDP formulation, which presents the largest range of uncertainty representation, also requires the discretization of the observation space, and exact or approximate algorithms have been developed only for very low dimensional scenarios. Sampling-based methods operate in a continuous state space, which is a more natural approach in real world navigation. Also, they seem more adapted to the dynamic environment and to handle time-constraints, as they are expressly dedicated to high dimensional spaces. Only little work has been done till now to adapt these methods to handle uncertainty and real-time constraints:

- Any-time RRTs lacks both the explicit representation of uncertainty and it does not really take into account the real-time constraints: neither the time for the initial plan, nor the time for the subsequent replanning is bounded, but the approach is based on the consideration that the algorithm should be fast enough, independently of the distance of the goal and the obstacles.
- Partial Motion Planning presents a valid solution for real-time navigation in dynamic environments but lacks an explicit representation of uncertainty.
- Particle-RRT is an effective method to integrate uncertainty in the decision process, but is not easily extensible to integrate in real-time new information and so to handle the evolution of uncertainty in environment sensing and prediction.

3.2 Motivation

The aim of our thesis is to propose a method to navigate autonomously among static and moving obstacles while handling the uncertainty due to the robot's perception of the environment and the real time constraints due to the dynamic and evolutive nature of the information. The problem presents the following characteristics:

1. The position of the robot is known.
2. The future configuration of the robot deterministically depends on the current configuration and the controls applied.
3. The static environment is not deterministically known. The occupation of observed parts of the environment is probabilistically estimated.
4. There are obstacles that move in the workspace and that can enter it during navigation.
5. The shape of moving obstacles is probabilistically estimated.
6. The state of moving obstacles (position and velocity) is probabilistically estimated.

7. The future position of moving obstacles can only be probabilistically predicted according to an estimated motion model and typical environmental patterns.

In terms of the uncertainty classification made in 1.1.3, we address the **environmental sensing uncertainty** ES (see points 3-5) and **environmental prediction uncertainty** EP (see points 6, 7). We will discard the configuration sensing CS and configuration prediction CP uncertainty (respectively points 1 and 2). As the environment is dynamic, we need to address also the **time-constraint** problem, making use of an anytime algorithm. In synthesis:

	ES	EP	CS	CP	Dynamic
Needed Properties	+	+	-	-	+

Table 3.2: Needed Properties

As shown in table 3.1, complete methods presented in §3.1.2 take into account probabilistic uncertainty, but are too costly in terms of computational power required by the dynamic environment. Also, the dimension of the space is initially unknown and changes during the task; the discretization required, expecially in the action space, is not adapted to maneuvering in cluttered environments. We will then use a sampling-based approach. In the previous paragraph we showed that all the sampling approaches presented are based on the RRT algorithm, that all present some of the needed properties, but none of them satisfies all our constraints. We need then to develop a new algorithm which contemporaneously is able to represent and update uncertainty in real-time with the changes of the environment. Among the sampling-based path planning algorithm presented in 3.1.2, the RRT algorithm presents some important advantages:

– it easily handles non-holonomic constraints;

– it is incremental, which seems a natural requirement when exploring an unknown environment.

We propose to use this algorithm as basis for the exploration of the configuration-time space and introduce two basic novelties:

– the probabilistic representation of the configuration-time space;

– the real-time updating of such a representation and the associated decision process.

In Particle-RRT, the probability distribution of the future configurations of the robot is represented by a set of particles. Extending this method to the uncertainty in the environment sensing and prediction would mean sampling a space of dimensions N_oM for each searched configuration-time of the robot, where N_o is the number of variables representing the state of each moving obstacle and M is the number of moving obstacles. The complexity of sampling is linear in each dimension, but exponential in the number of dimensions: this kind of approach scales badly with the number

of obstacles.
We propose to represent the uncertainty in the static environment by mean of an occupancy grid; the uncertainty in the dynamic environment is instead given by independent probability distribution functions, that can be represented by continuous or discrete forms each. This approach is linear with the number of variables in the system state. In Section §3.3.1 we present in detail the classic RRT framework. In Section §3.3.2 we propose an extension of the algorithm to maintain a probabilistic representation and search and take decisions over such a space.
In Section §3.3.3 we will integrate this method in an iterative planning and execution method inspired by the PMP framework: we present the way to update on-line the representation of the probabilistic world and consequently adapt the robot's decisions.

3.3 Contribution: Probabilistic RRTs

The algorithm described in the previous paragraph lies on a deterministic representation of the environment, i.e. $\mathcal{C}_{free}$ and $\mathcal{C}_{obs}$ are deterministically and *a priori* known. As stated in previous sections however, the robot's knowledge about the environment is incomplete and uncertain. To take into account these limitations, we have proposed a probabilistic representation of the state of the world, which evolves with successive observations. On the basis of the RRT algorithm we developed then a probabilistic planner.

3.3.1 Rapidly-exploring Random Trees

The Rapidly-exploring Random Tree (RRT), introduced in [49] is a well known randomized algorithm to explore large state spaces in relatively short time. The algorithm incrementally builds a search tree and gradually improves the resolution among the state space: in the limit the tree densely covers the space (see Fig. 3.1). The algorithm is proven to be probabilistically complete. Algorithm

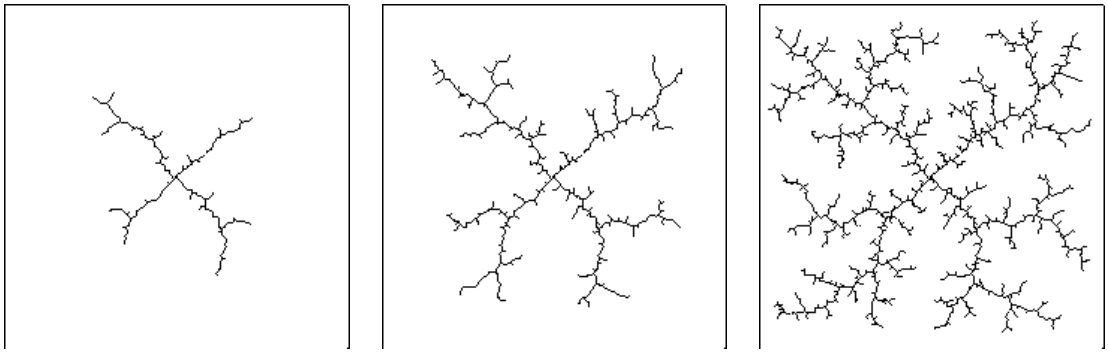


Figure 3.1: Rapidly Exploring Random Tree, courtesy of http://msl.cs.uiuc.edu/rrt/

1 resumes the RRT algorithm for single query planning. At each iteration, a point p is randomly

Algorithm 1: RRT

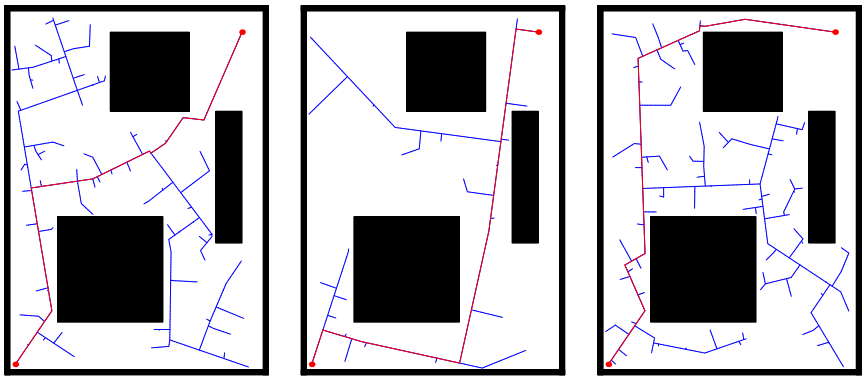
```
1 T.init(q_0);
2 while q_goal ∉ T do
3     p = ChooseState(q_goal);
4     q = T.NearestNeighbor(p);
5     [u_new, q_new] = T.Extend(q, p);
6     if q_new ∈ C_free then
7         T.addNode(q_new);
8         T.addEdge(q, q_new, u_new);
9 end
```

chosen from the state space and the algorithm tries to extend the current search tree toward that point (line 3). The nearest node q to p within the nodes of the search tree is chosen for extension (line 4). A local planner searches for an admissible control u_{new} to apply for a time interval Δt and that results in a node $q_{new} \in \mathcal{C}_{free}$ (line 5) . If the transition from q to q_{new} is collision free, q_{new} is added to the tree (lines 7 and 8). The search can continue until the tree is considered sufficiently dense. In the case of single-query planning, the goal of the algorithm is to find a path from a given initial configuration to a specific goal. In this case it can be unnecessary to spread the tree all over the state space: the root of the tree is the initial configuration, and the algorithm tries to extend the tree toward the goal. In order to speed-up the exploration, some bias is used: for instance, p can be chosen to be the goal once on a given number of iterations or with some probability. In algorithm 1 the ending condition of the algorithm (line 2) is just the goal to be reached and the path from the initial state to the goal is retrieved. In different implementations of the algorithm however the tree can continue to grow for all the available time and the best (shortest) path is chosen or it can stop once one of the found paths is good enough with respect to some constraint previously defined (length, smoothness). Figure 3.2 shows the described algorithm applied to a point robot in a known static environment. The initial position of the robot is at the left bottom corner, while the goal is at the right top corner. Images 3.2(a),(b) and (c) show in blue the search tree and in red the chosen path obtained in 3 runnings of Algorithm 1 which give different results.

ChooseState()

Different techniques can be used to efficiently choose the point toward which extend the tree. On one side the search must be driven toward the goal, on the other side some space must be explored in order to avoid local minima.

- Random Sampling: at each time step, a sample is randomly drawn in the state space. The bias to the goal is obtained setting a probability to pick the goal and choosing at each iteration



(a) (b) (c)

Figure 3.2: RRT basic algorithm applied to a point robot in a known static environment.

whether the goal or a random sample should be picked. If some *a priori* information is available, a non-uniform distribution function can be designed to draw the sample more efficiently.

- Dense Sampling: the samples are systematically called from a list, improving the resolution with the iterations. The bias to the gal is obtained interleaving the list with the goal at a predefined frequency.

The frequency with which the goal is chosen can be seen as a measure of the ratio between the time we want to spend in searching for a solution and the time we want to spend to explore the space. The best value for this bias is highly dependent on the characteristics of the environment: obstacle density, size and position.

3.3.2 Probabilistic Search algorithm

Let's assume an uncertain configuration space, where we cannot recognize $\mathcal{C}_{free}$ and $\mathcal{C}_{obs}$, but for each point q, a probability of collision $P(q, q')$ is estimated for each q' that can be reached from q using a simple control for a fixed time. $P(q, q')$ represents the probability of collision that encounters the robot when passing from q to q'. The map can be interpreted as a cost map. The cost of traversing a sequence of configurations is not additive on the single configurations: considering the sequence $\pi(q_N) = q_0...q_N$ where each configuration q_n is connected to the next one q_{n+1} by an edge, the probability to reach q_N without entering in collision is computed recursively considering

independent probabilities of collisions in each node.

$$P_{coll}(\pi(q_N)) = P_{coll}(\pi(q_{N-1})) + (1 - P_{coll}(\pi(q_{N-1}))) \cdot P_{coll}(q_{N-1}, q_N) = \quad (3.1)$$

$$= 1 - \prod_{i=1}^{N}(1 - P_{coll}(q_{i-1}, q_i)),\ P_{coll}(q_0) = 0 \quad (3.2)$$

$$L(\pi(q_N)) = 1 - P_{coll}(\pi(q_N)) = \prod_{i=1}^{N}(1 - P_{coll}(q_{i-1}, q_i)) \quad (3.3)$$

The probability of traversing the tree from q_0 to q_N without collision is called the **probability of success** of the partial path $\pi(q_N)$.
To perform a search we need to define a weight for each node in order to choose the best one to extend. A random point p is chosen from the state space and the algorithm tries to extend the current search tree toward that point. Each node is labelled with a weight w which accounts not only for the expected length of the global path $\pi(q_N, p) = q_0, ...q_N, ...p$ passing through q_N, but also for the probability of success of the path. First of all, the probability of success is normalized with respect to the number of nodes in the path. This step allows to have a measure of the "quality" of the paths that is independent of their respective length:

$$\tilde{L}_\pi(q_N) = \sqrt[N]{L_\pi(q_N)} \quad (3.4)$$

The computed weight is proportional to the normalized probability of success of the path and inversely proportional to its expected length:

$$\tilde{w}(p, q_N) = \frac{\tilde{L}_\pi(q_N)}{D(q_N, p)} \quad (3.5)$$

$$w = \frac{\tilde{w}}{\sum_{i=1}^{N} \tilde{w}} \quad (3.6)$$

The function $D(q_N, p)$ is an estimation of the length of the global path. Only the first part of it, from q_0 to q_N, is known. The length of $\pi(q_N, p)$ is unknown, and an heuristic $traj(q_N, p)$ must be used, depending on the kinematic constraints of the robot. For holonomic robots a simple Euclidean distance can be used; for car-like robots, a trajectory from q_N to p is computed using a combination of a circular arc at maximum steering to align the robot with the direction of p and a straight line to reach it.

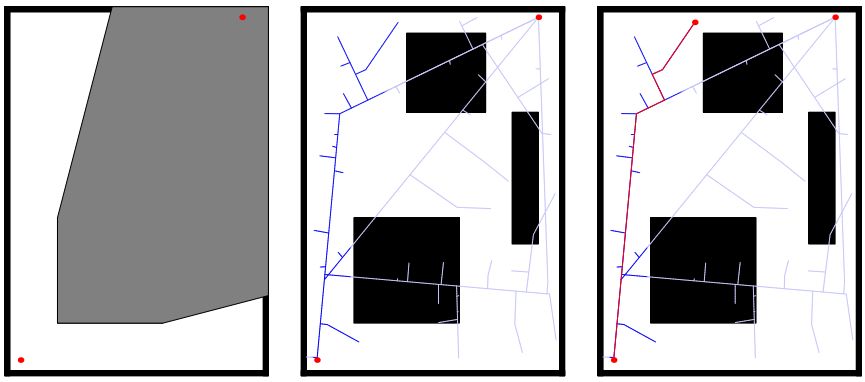
$$D(q_N, p) = \sum_{i=1}^{N} |q_i - q_{i-1}| + traj(q_N, p) \quad (3.7)$$

The node with the highest weight is the *best* node over the tree and is chosen to be expanded. Normalizing the weight over the nodes of the tree, the node could also be chosen drawing a random node proportionally to the normalized weight. This allows the algorithm to come out of local minima

in the case of a cluttered environment, but gives worse results in the case of a free environment. The local planner acts as in the classic RRT algorithm. All nodes are added to the tree, or a lower threshold L_{safe} can be chosen to avoid the exploration to move toward too unlikely zones. The search can continue until the time is out or until some other condition is satisfied: for instance the algorithm can require that q_G is in the tree or that a path with a minimal safety has been found. Algorithm 2 resumes the proposed approach. Figure 3.3 shows the performance of the algorithm

Algorithm 2: Probabilistic RRT

```
1 T.init(q_0);
2 while q_goal ∉ T do
3     P = ChooseState(q_goal);
4     q = T.BestNode(P);
5     [u_new, q_new, P(q_new)] = T.Extend(q, P);
6     T.addNode(q_new);
7     T.addEdge(q, q_new, u_new), P(q_new);
8 end
```



(a) (b) (c)

Figure 3.3: Probabilistic RRT basic algorithm applied to a point robot in an unknown static environment.

in a simple static environment, with an holonomic point robot. On the left are the initial position and the current space perceived by the robot: the area behind the nearest obstacle is completely occluded (grey). In the center is the tree grown in the available time: darker blue is for more likely paths. On the right is the partial path chosen according to the weight of the nodes. The algorithm can easily be used for differently shaped robots, non-holonomic robots and in a dynamic environment. Figure 3.4 shows a partial tree for a car-like robot (rectangle in red) in an acquired occupancy grid. In this case, the controls to grow the tree are given by pairs $(a, \dot{\phi})$ where a is an admissible linear acceleration and $\dot{\phi}$ an admissible steering rate. These controls are considered constant on the timestep τ. The goal for the robot is set at some distance in front of it; each time

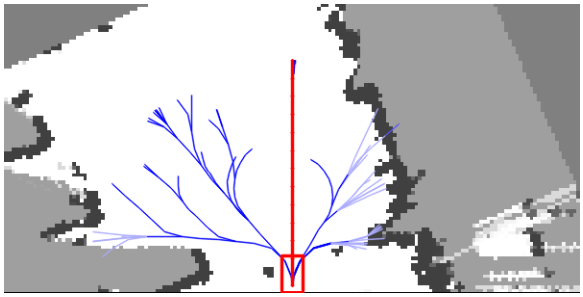
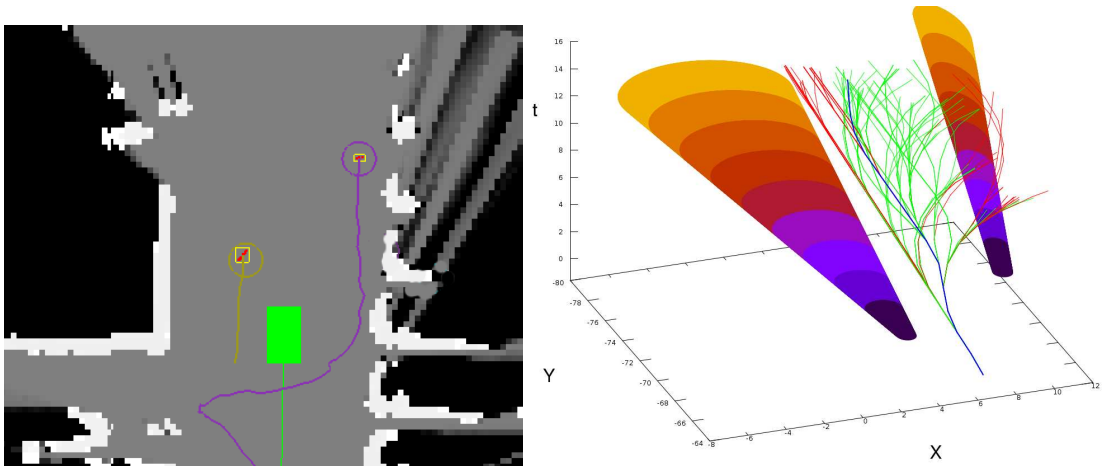


Figure 3.4: Probabilistic RRT algorithm applied to a non-holonomic robot in static environment. The search tree is shown in blue, lighet colour is for lower likelihood. The chosen path is shown in red. A discretized set of controls has been used to generate the tree.



(a) (b)

Figure 3.5: Example of Probabilistic RRT in an uncertain dynamic environment: (a) the occupancy grid and two moving obstacles; (b) the prediction of the obstacles, represented by ellipses with axes once the standard deviation and the grown search tree.

a point is chosen in the space, the best control is chosen as the best among a discretized set $(a, \dot{\phi})$. In the case shown in the figure, a uniform discretization of the admissible interval has been used, where 5 was the cardinality of the set of the linear accelerations and 7 the one of the steering rate. The color of the branches reflects the probability of success of the associated path: the lighter the colour, the lower the probability.

Figure 3.5 shows a partial tree grown in a dynamic environment. On the left the static occupancy grid and two tracked moving obstacles along with their estimated trajectories. On the right the search tree is presented in the time-space coordinates: the predictions of the moving obstacles are represented at each instant by ellipses with axes corresponding to the standard deviation interval according to the prediction coming from the target tracking algorithm: the superposition of these ellipses generates the truncated cones shown on the figure. The green and red branches are the

explored configurations of the robot. A lower threshold has here been imposed to distinguish safer (green) and less safe (red) paths. The path chosen at the current timestep is shown in blue. In both Figure 3.3 and Figure 3.5 the goal had been set far in front of the robot.

3.3.3 Planning and replanning in dynamic uncertain environment

The knowledge of the robot is evolutive: new information is integrated in time, during navigation. To take advantage of the new information, the robot must adapt its decisions as soon and as often as possible. The Partial Motion Planning is a framework that takes explicitly into account the fact that in dynamic environment, the time available for planning is inevitably limited. Our contribution extends this original approach to take into account the evolutive nature of the information and improve the chosen path.

3.3.4 Partial Motion Planning

In a dynamic environment, the robot needs to choose a set of actions that have to be executed at a certain instant or in a certain window of time. If we consider for example a car that enters a lane, the maneuver chosen has to be executed in the time window in which the lane is free. Again, if we consider a car that overtakes another one, the plan should be executed before the other car changes its behaviour: it turns on an indicator, it changes velocity. So, the time available to perform planning depends:

- On the trajectory of the moving objects;
- On the time-validity of the models used.

Most of the developed planning algorithms do not take explicitly into account this issue. The idea of PMP [9] is to guarantee that after any fixed interval the trajectory planned by the robot is collision free, i.e. each configuration considered is not an Inevitable Collision State (ICS, see 2.1). After each planning cycle, the planned trajectory is generally just a partial trajectory: the convergence properties of global planning are lost, but the time constraints and the moving obstacles are properly taken into account: the length (in time) of the computed trajectory is limited by the validity of the motion model of the dynamic obstacles and eventually by the fact that the available time is not long enough to find a solution to the global problem. PMP can be seen as a reactive algorithm extended to more time steps with the important difference that the final state is guaranteed not to be an ICS, under the hypotheses considered. A fixed time interval is used. During this time, a search tree is randomly grown, on the model of the RRTs. A node is added to the tree only if it is not an ICS. However, as there does not exists a practical algorithm to prove that a state is an ICS, a set of braking trajectories are checked from the node: if it exists a braking trajectory that is collision free, the node is proven not to be an ICS and is added to the tree. When the available time is over, the nearest node to the

goal is found and the trajectory from the root is retrieved. The robot begins to execute the path; at the same time, it begins to search for the next trajectory: the model of the world is updated with the incoming information and the new tree considered has its root in the final configuration of the trajectory computed at the previous cycle. The planning algorithm works in parallel with execution.

Problems

This algorithm presents some limits:

- The tree is reset at each cycle: each new cycle does not take any advantage from the previous search.
- The models are deterministic: the model of the static and dynamic world are supposed to be perfectly known in the time horizon considered.
- The cycle time and the time horizon are fixed: the algorithm cannot adapt this interval to the actual conditions of the environment.
- The ICS check for each node is conservative: checking only braking trajectories, for instance, prevents the robot from accelerating and pass before obstacles.

Our approach presents an adaptable time horizon for planning. The time for the planning iterations depends on the length of the previous computed trajectory and on the on-line observations. The search tree is updated on-line, so that the search performed at the previous time step is the base for the new search. Safety of a path is guaranteed studying braking trajectories only for the last state of the path.

3.3.5 Proposed Solution: Probabilistic Partial RRTs

At the beginning of navigation, we suppose that the robot has some time to observe the environment and to plan before moving. From the initial state s_0, the tree is grown using the probabilistic RRT algorithm described in §3.3.2. At the end of the available time, the tree has reached depth N. At each of the explored partial paths a weight is associated using equations 3.5 and 3.6. The best path (with the higher weight) is picked out. To guarantee that the robot does not choose a dangerous path, we can specify the maximum allowed risk using a safety threshold. The chosen path is a sequence of N_2 configurations to be attended, where $0 \leq N_2 \leq N$. While the robot executes the path, the tree is updated and grown. At each timestep, the nodes of the tree that are not reachable (in the past) are pruned out and the probability of success of each path is updated with the new information coming from the observations. In the remaining available time, the tree is grown. At the same time, the robot acquires new observations and compares the planned path with the new estimation: if

the path is still safe, the robot continues to follow it; if an unexpected event is detected, a new safe path is retrieved from the search tree. If no safe path is found within the available time, the robot proceeds to a braking manoeuvre. In a very unpredictable environment, the time horizon will be shortened by the unexpected events and the algorithm will act like a purely reactive algorithm. On the opposite, in the case of static known environment, the algorithm will allow longer and longer time to grow the tree, and the result would be nearer to the solutions of a global planner.

Algorithm 3: Probabilistic Partial RRT

```
1  start_time = clock();
2  T.init(q_0);
3  while (clock() − start_time < available_time) do
4      p = ChooseState(q_goal);
5      q = T.BestNode(p);
6      [u_new, q_new, P(q_new)] = T.Extend(q, p);
7      T.addNode(q_new);
8      T.addEdge(q, q_new, u_new), P(q_new);
9  end
10 return(T.BestNode(q_goal));
```

3.3.6 Update the tree

When the robot moves, it observes the environment and updates its estimation with the incoming observations. The cost of crossing the tree changes and the tree needs to be updated. The update consists in three steps:

1. Prune the tree: the new root is the position of the robot and nodes that are in the past are deleted; the probability of reaching the nodes is updated, taking into account that the robot has already crossed part of the tree.

2. Update the weight of the nodes: when a change in the probability of collision is detected, the weight of the correspondent nodes (and of their subtree) must be updated.

3. Retrieve the best path.

If the considered environment is dynamic we need the robot to do these operations in real-time. In better words we need to know how much time is available for updating and how to allocate it. In the first step, the present state of the robot is considered. The tree is pruned so that only the subtree attached to the state of the robot is maintained. When the probability to pass from a configuration

q_0 to q_i changes, the weight of the subtree attached to q_i is updated using the following equations:

$$L(\pi(q_i, q_N)) = \frac{L(\pi(q_0, q_N))}{L(\pi(q_0, q_i))} \tag{3.8}$$

$$\begin{aligned} \hat{L}(\pi(q_0, q_N)) &= L(\pi(q_0, q_i))(1 - \hat{P}(q_i, q_j))(1 - P(q_j, q_N)) = \\ &= L(\pi(q_0, q_i))(1 - \hat{P}(q_i, q_j)) \prod_{k=j+1}^{N} (1 - P_{coll}(q_{k-1}, q_k)) \end{aligned} \tag{3.9}$$

Equation 3.8 is used once when the tree is pruned. In the equation q_0 is the old root, q_i is the new root and q_N is one node in the family of q_i. This first update is due to the fact that the robot has already moved from q_0 to q_i, and it is equivalent to the probability of reaching q_N from q_0 when the probability of reaching q_i is 1.
Equation 3.10 is used when the observations have revealed some difference with the prediction. The zones in which some difference have been detected are considered and the affected nodes are updated. In this case q_i and q_j are respectively the start and ending configuration in which a change in the probability of collision has been detected, from $P(q_i, q_j)$ to $\hat{P}(q_i, q_j)$.

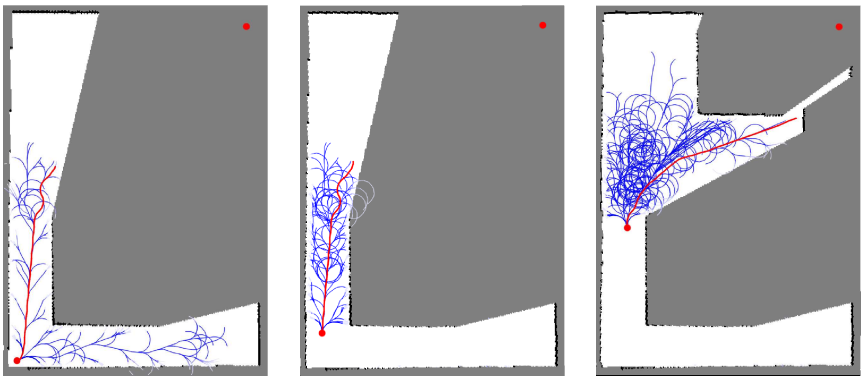


Figure 3.6: Updating and growing the tree during environment exploration.

In Fig. 3.6, the on-line updating of the tree is shown at 3 instants during navigation. At the beginnning, the most likely paths are explored in the two possible directions and the most promising one is chosen: the path with the highest weight according to equation 3.6 is drawn in red in Fig.3.6(a). Fig.3.6(b) shows the tree after some steps: the tree has been updated: the branch in the right direction has been cut as it became unreachable and the tree has been grown a little toward the promising direction. Fig.3.6(c) shows the tree and the new partial path found when a bigger portion of the space is visible.

(a) (b)

Figure 3.7: (a)The vehicle in the parking lot at INRIA Rhône-Alpes and a moving pedestrian; (b) The extracted occupancy grid, with the vehicle in green and the tracked moving obstacle in yellow.

3.4 A car-like robot among moving obstacles

In this section we briefly describe the perception and prediction aspects and detail the navigation algorithm in the more specific case of a car-like robot navigating among moving obstacles in an outdoor environment.

3.4.1 Perception

Simultaneous Localization and Mapping

The first and fundamental issue for every autonomous mobile robot is to be able to self-localize with respect to the static environment. If a sufficiently precise map of the environment is available, the robot compares its observations with the map in order to understand its current position. When the *a priori* knowledge of the environment is incomplete, uncertain, or not available at all the robot must build the map during navigation and simultaneously, it must use the same map to compute its position. The problem is referred to as the *Simultaneous Localization And Mapping* (SLAM) problem and has gained a relevant position in robotics research since a first closed formulation of the problem was introduced in [2]. The complexity of the SLAM problem comes from the strong correlation between the localization and the mapping tasks: on one side, the robot pose has to be estimated with respect to the map; on the other one, the map itself is built with respect to the estimated robot pose. The problem has been deeply investigated in literature and different solutions have been proposed; for a comprehensive overview see [54, 55, 56]. The approach we employed is based on the work of Vu [57]: here we give a brief description of the approach. A more detailed description of the algorithm and parameters used to obtain results can be found in chapter 5.
The representation of the static environment is based on an absolute occupancy grid, whose origin

is attached to the initial position of the center of rotation of the robot. The observations gathered by the exteroceptive sensors of the robot (in our case a laser range finder) are used to estimate the occupation of each position in the environment. The robot position is first estimated thanks to the fusion between the information coming from the GPS sensor (if available) and the odometric data. Then, an Iterative Closest Point algorithm (ICP, [58]) is applied, trying to align the laser readings with the estimated grid. Hence, the position of the robot can be refined. As the robot is supposed to move in large environments, parts of the occupancy grid that are far away from the actual position of the robot and of the goal are "forgotten" after some time. This has the purpose of limiting the memory requirements to store the grid and the errors due to very large loop closing. Some of the laser beams could have hit moving obstacles: in this case, it is expected that a cluster of readings come from an area that have been estimated as free. These readings are filtered out before updating the occupancy grid and form the hypothesis for a moving obstacle observation. These observation are then elaborated by a Multi Target Tracking algorithm.

Multiple Target Tracking

The aim of a Multiple Target Tracking algorithm is:

- to associate a list of observations to a list of objects;
- to initialize, maintain and update on-line the list of moving objects, their state and track.

The algorithm must cope with non detections and false alarms and in general also with changes in the moving direction and velocity of obstacles. The detailed description of the state of the art algorithms is beyond the scope of this document. An extended review can be found in [59].
The Target Tracking algorithm we use is a Multi Hypothesis Tracking based on a Global Nearest Neighbour data association and a set of Kalman Filters [60]. The state of the obstacles is given by their position in the horizontal plane (x, y) and their velocity in the 2 directions (v_x, v_y). Obstacles are supposed to be circular and their orientation is supposed to be the same of the velocity direction. In lack of further information, the motion model is a constant acceleration motion model.

$$X(t+1) = A \cdot X(t) + B \cdot U(t) \tag{3.10}$$

where X is the state of the obstacle, A is the dynamic matrix, B is the input matrix, U is

$$X = \begin{pmatrix} x \\ y \\ v_x \\ v_y \end{pmatrix} \qquad A = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad B = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad U = \begin{pmatrix} a_x \\ a_y \end{pmatrix} \tag{3.11}$$

the acceleration and is usually set equal to the last observed difference between the predicted and

observed velocity. As the pedestrians can easily change velocity and direction, the noise associated to the measures must be big enough to allow a good data-association. Given the set of tracks at the previous step and the set of observations, each observation is associated with an existent track or a new track. First the global solution that minimizes the *distance* between the observation and the one step ahead prediction is computed. The observations that are not associated with any track are hypotheses for new tracks, which must be confirmed by the subsequent observations; tracks that have not been observed are maintained for a short time before being deleted [61]. On the basis of the Kalman filter, the estimation of the state of each tracked pedestrian is given by a Gaussian probability distribution whose mean and covariance matrix are recursively estimated by the filter.

3.4.2 Prediction

Inoreder to achieve motion planning and navigation, we need to predict the occupation of the space in the future. Each time a node must be added to the tree, the occupation at the corresponding configuration and time is checked among the grid and the moving obstacles. The grid is supposed to map the static environment, so there is no need of prediction; the moving obstacles prediction is performed on the basis of the motion model estimated by the tracking algorithm. The hypothesis that new obstacles can enter the scene during motion is also considered.

Gaussian Prediction

Tracking algorithms based on the Kalman filter (KF) are constituted by linear motion models and Gaussian estimations of the state of the objects.

$$P(O_i(t)) \leftarrow \mathcal{N}(X_i(t), \Sigma_i(t)) \tag{3.12}$$

where $P(O_i(t))$ is the probability distribution representing obstacle O_i at time t, and is a Gaussian ($\mathcal{N}$) of mean $X_i(t)$ and covariance matrix $\Sigma_i(t)$.
The predictions based on the linear motion model A and on the noise statistics associated to the motion model are always Gaussian and can be analytically computed from the last estimation. Considering time $t + k$:

$$\hat{X}(t+k) = A \cdot X(t+k-1) = A^k \cdot X(t) \tag{3.13}$$

$$\Sigma(t+k) = A^T \cdot \Sigma(t+k-1) \cdot A + Q = (A^T)^k \cdot \Sigma(t) \cdot A^k + \sum_{j=0}^{k} (A^j \cdot Q) \tag{3.14}$$

where Q is the covariance matrix of the zero mean Gaussian noise associated to the dynamical model which is chosen *a priori*.

New obstacles entering the scene

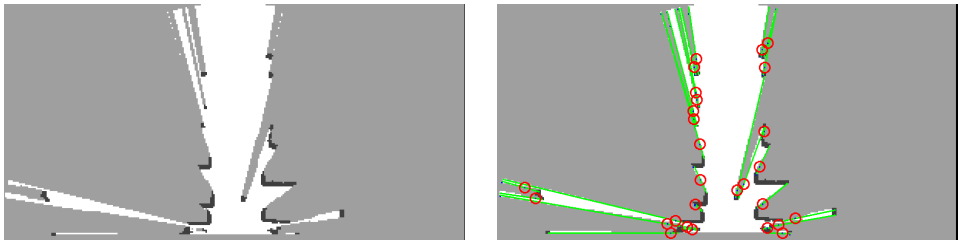


Figure 3.8: (a)A partial grid map; (b) The extracted doors (green lines) and the supposed new entering obstacles (red circles).

In dynamic environments, obstacles can enter or exit the workspace during the navigation task. Also if partial planning is used, it should be taken into account that new obstacles can enter the the workspace and interfere with the next motions of the robot. A usual approach is to pass to a reactive behaviour when an unexpected obstacle is encountered and the chosen path is invalidated. However it is possible to predict with some confidence from where and when some obstacle may enter the scene and perform a more robust planning on the basis of this information. We must:

- Distinguish from where a new obstacle may come.

- Apply a probability to the fact that an obstacle may enter and a motion model.

For the first problem, two techniques can be considered. We can suppose that an obstacle can enter from anywhere in the non-observed space. In this case, new obstacles are considered all around the robot just beyond the maximum sensor range or the first encountered obstacle. Using a worst-case hypothesis, the obstacles are supposed to move toward the robot. Otherwise we can search for specific areas from where an obstacle may enter. This technique is based on some assumptions about the observed space and the size, shape and behaviour of the obstacles. In this case, the robot must be able to recognize on-line the *doors* with its sensor perception only. In the parking environment where we tested our algorithm, we assumed that obstacles may enter only from hidden areas: i.e. they cannot traverse static obstacles. Given a partial map and the point of view of the robot, these regions are easily extracted. For each hidden region, the partial map is observed to see if the area around the door is occupied, free or occluded. If the area is occupied, the considered interval is discarded: the area is occupied by a static obstacle that is hidden only from the current point of view. If the area is free or occluded ($P_{occ} \leq 0.5$), and it is also wider than the minimum radius of the obstacles, a door is recognized. In Figure 3.8 the possible *doors* for new entering obstacles are shown as green lines. The probability of some new obstacles entering in the workspace can be

modeled as an homogeneous Poisson process.

$$P[(N(t+\tau) - N(t)) = k] = \frac{e^{-\lambda\tau}(\lambda\tau)^k}{k!} \qquad k = 0, 1, \ldots, K \tag{3.15}$$

where $k = N(t+\tau) - N(t)$ describes the number of events in time interval $]t, t+\tau]$; the rate parameter λ, is the expected number of arrivals per unit time. This parameter could be derived from a learning phase or fixed *a priori*. The probability that at least one obstacle enters the scene is given by:

$$P[N(t+\tau) - N(t) \geq 1] = 1 - e^{-\lambda\tau} \tag{3.16}$$

When performing prediction, an obstacle is initialized just behind the door, in the nearest possible point with respect to the robot actual position. In Figure 3.8 the possible positions of new obstacles are shown as red circles. The probability of occupation corresponding to the obstacle grows with the length of the time of prediction according to equation 3.15. A motion model is designed for each obstacle using a worst case approach: the obstacle is supposed to move toward the robot at its maximum speed. A stochastic noise in both direction and velocity is added to the model to take into account the other possibilities of motion.

3.4.3 Motion Planning

Choosing a point in the space: function ChooseState()

In this section we discuss the implementation of the function ChooseState() (see Algorithm 2, line 3) in the considered case. At the first iteration, the goal is always chosen: if a straight path to the goal is free, it is found first and the vehicle path is more efficiently chosen. In subsequent iterations, the goal is chosen with a random sampling in a space that grows around the robot proportionally with the depth of the tree and the dynamic possibilities of the robot. In general, the depth of the tree (time-length of the partial paths) is sufficient for the robot to reach the sampled point.

Nearest Node for non-holonomic robots

In this section we discuss the implementation of the function $D(q_N, p)$ (see 3.3.2) in the case of a non-holonomic robot with car-like constraints: this function allows to find the NearestNeighbor() among the nodes in the tree. In the case of unconstrained motion it is supposed that the system can move in all directions along line segments and the nearest node to a point in a 2D space can be easily found using Euclidean distance. In case of non-holonomic constraints, instead, as in a car-like robot, the system moves along curves and some point of the configuration space is not reachable at all. So, it might be too difficult and computationally expensive to implement an exact algorithm to find the nearest node to a configuration and the best admissible control to reach it. In this case, approximate solutions can be adopted. In our choice the function $D(q_N, p)$ is the length of a simple

feasible trajectory for the robot, composed by a maximum steering manoeuvre to align the robot with the point p and a straight line path.

Probabilistic Collision Checker

Given the representation of the static environment and of the moving obstacles we must find the probability of occupation of each explored configuration of the robot q and each transition from q to q', where $q' = Aq + u$. This probability $P_{coll}(q, q')$, is used to compute the likelihood of each node (see Eq. 3.1).

Let us consider a car-like robot, of rectangular shape and whose controls are defined by couples (v, ω) where v is the linear and ω is the angular velocity. Given the configuration of the robot $q = (x, y, \theta)$ and a control, the trajectory performed in an interval τ is given by the simple car-like model. In the case of straight motion ($\omega = 0$) the trace of the robot is the rectangle elongated by the length of the path. In case of $\omega \neq 0$ the trace of the robot depends on the position and the width of the wheel axes. The area is non-convex and delimited by circular arcs centred in c and line segments, as in Fig. 3.9(a). In continuous state space, the set of possible controls and ending states is infinite: it is not possible then to define $P_{coll}(q, q')$ for each q and each q'. In general, instead, a probability of occupation is estimated for each point in the space-time through a parametrization or a discretization of this space.

In our approach, the representation of the static environment is based on a spatial occupancy grid: the probability of collision given by the static obstacles is computed as the maximum probability of occupation among all the cells that fall also partially into the area swept by the robot:

$$P(coll(q, q', \mathcal{G}) = \max_{S} P_{occ}(i, j) \tag{3.17}$$

where $S = \{(i, j)\}$ is the set of cells traversed by the robot (see Figure 3.9(b)). As the grid is

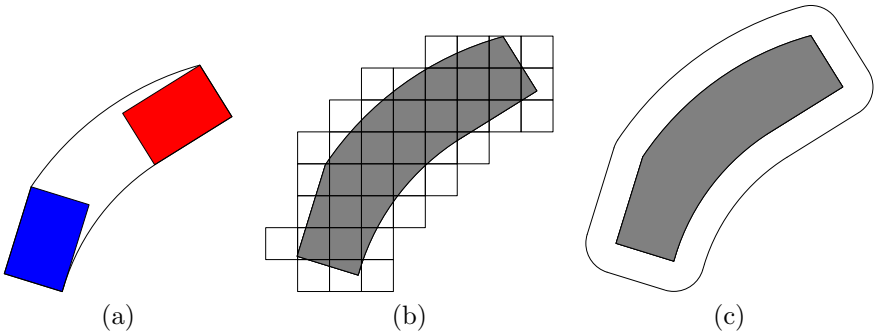


Figure 3.9: a) Area swept by the robot in a steering maneuver; b) cell coverage; c) enlarged by radius for a moving obstacle.

considered static there is no need of prediction and the probability of collision only depends on the

considered configuration, and not on the time instant. However, since the grid itself is updated at each time step, the probability of collision due to the static environment can change in subsequent observations.

When considering moving obstacles, the probability of occupation of the space must be given by the prediction at the considered time instant. Considering a circular moving obstacle o_n, the area swept by the robot is enlarged by the radius of the obstacle (Fig. 3.9(c)). The obstacle moves in the considered time interval; in our case the motion model of the obstacles are given by linear functions with Gaussian noise see §3.4.2. We consider the mean function and the growing covariance in the time interval. The probability of occupation is given by the integral of the predicted probability distribution function over the enlarged area:

$$P(t_{coll} = t|q, q', o_n) = \int_{Area} P(o_n(t)) \tag{3.18}$$

We suppose that the time interval is short enough in comparison with the size and the velocity of each obstacle, so that it is sufficient to approximate the time integral with the maximum value between the beginning and at the end of the time interval. This approximation is not sufficient when the object size is small and its velocity is big, if compared with the robot size and velocity. If the distance between the initial position and the ending position of the obstacle is big in comparison, it is necessary to subdivide appropriately the time interval and choose the maximum over the considered steps. The distribution is sampled proportionally with the probability and the ratio between the samples that fall respectively inside and outside the area gives the probability of collision. The precision of the approximation depends on the number of samples used: however, the more the samples, the larger the time to compute the integral.

Considering N moving objects, the motion of each object is considered independent from the others: this is a simplifying hypothesis. Under this hypothesis, the total probability of collision is computed as follows:

$$P(t_{coll} = t|o_1....o_N) = 1 - \prod_{n=1}^{N} (1 - P(t_{coll} = t|o_n)) \tag{3.19}$$

The total probability of collision considering both the static and dynamic obstacles is computed as follows:

$$P(t_{coll} = t|o_1, .., o_N, \mathcal{G}) = P(t_{coll} = t|\mathcal{G}) + (1 - P(t_{coll} = t|\mathcal{G}) \cdot P(t_{coll} = t|o_1....o_N)) \tag{3.20}$$

The underlying hypothesis is that, again, collision with a static and a moving obstacle are mutually exclusive.

Choose a path: safety issues

Given a search tree, the best path to choose is the one with the highest weight considering equations 3.5 and 3.6 for $p = q_G$. However, in this way it is still not guaranteed that the robot chooses a path that the user can consider *safe* enough. The choice can then be restricted adopting a lower threshold L_{safe}. The advantage is that we have a maximum limit for the acceptable risk and that the algorithm is faster and more accurate as all the time is spent in the safest solution. On the other side, such a threshold could reduce the set of feasible solutions to an empty set. In this case, either the robot must perform an open-loop emergency maneuver or it must rely on another, more reactive algorithm.

3.5 Results

The planning algorithm has been tested with real data acquired on a car-like vehicle equipped with a laser range finder and a differential GPS (Cycab [62]). During the experiment, the robot is manually driven in an outdoor environment and perceives static obstacles and moving pedestrians. Using the algorithm developed in [57, 61] and detailed in Chapter 5, the robot localizes itself, builds an occupancy grid map of the static environment and tracks the moving obstacles. The probabilistic predictions of the future state of the obstacles are computed using the constant velocity motion model, the position, velocity and covariance matrix estimated by the tracking algorithm. To test the planning algorithm we defined a goal 30 meters ahead the robot at each observation cycle and let the algorithm run in parallel with the on-line mapping and tracking: the tracking algorithm runs at about $10Hz$, the planning algorithm runs at $2Hz$. The output of both the mapping and planning algorithm are recorded: the occupancy grid, the tracked pedestrians, the state of the Cycab, the pedestrian predictions, the search tree and the chosen partial path. At the end of the dataset, each recorded partial path is tested with the observations gathered during the navigation: a *virtual* robot is moved through the map obtained by the mapping algorithm. The pedestrian observations are shown in the occupancy grid and are compared with the predictions based on the target tracking at the previous time steps. Figure 3.10 shows some obtained results. Pictures labelled with *tracking* are the outputs of the mapping and target tracking algorithm: a gray scale is used to show the probability of occupancy, darker colour stands for higher occupation probability. The robot is the green rectangle, while moving obstacles are shown as coloured ellipses along with their track. Pictures labelled with *plan* show the output of the prediction and the planning algorithm at a certain instant: the robot and the occupancy grid are shown; the future predictions of moving obstacles are shown by ellipses of ray twice the standard deviation estimated by the filter respectively in the x and y direction. The search tree is shown: the darker the color the higher the likelihood of the path. The chosen path is drawn in red. Pictures labelled with *validation* show the output of the validation process: the full red circles are the position of the pedestrians observed in future time

(with respect to the instant when planning is performed). These observations can be compared with the corresponding predictions at planning time and with the chosen partial path. The figure shows the case where an obstacle starts to move toward the left in front of the robot. In the first line, the prediction is not very accurate but still it is accurate enough to allow a safe plan (second line). In the third line, the velocity of the pedestrian is better estimated: the prediction is more accurate: this time the robot chooses to pass on the right of the obstacle, performing a safe maneuver (fourth line). Figure 3.11 shows a scenario where two pedestrians cross their trajectories in front of the robot. The robot is stopped and the obstacles are very near to it (Fig. 3.11(a-d)). In the first instants, no movements are possible: Fig 3.11(e) shows the search tree where the most likely maneuver is to stand still. After some instants, the obstacles move away and the robot can plan a path toward the goal, straight ahead (Fig 3.11(f)). Fig3.11(f-g) shows the validation of the chosen path.

However, the linear motion model is not always sufficient to describe pedestrian movements. In some cases, the short observation time before observation does not allow to accurately estimate the velocity; in some other cases the obstacles change direction abruptly, invalidating the motion model and the consequent prediction. Fig 3.12 shows a computed partial path in the case where the future predictions are not accurate: the obstacle first moves to the right, then stops and begins to move to the left. The path is computed before the robot can observe the new movement direction. At validation time, the computed path leads to collision. That is why it is necessary to update the collision probabilities and adapt the path to the new incoming information. Another similar situation is when an obstacle enters the scene just after the partial path has been computed. Also in this case it is possible that the computed path, safe with respect to prediction, becomes dangerous at execution time. Fig 3.13 shows such a case: the robot computes the path in the grid without moving obstacles (a,d); a pedestrian enters the scene just after this instant (b) and crosses the road in front of the robot (c). The computed path becomes dangerous in comparison with the acquired predictions. Fig 3.14 shows the case where the hypothesis of new entering obstacles and the *doors* extraction has been considered (see §3.4.2) . The partial plan is computed on the same data used for Fig 3.13. This time multiple hypotheses of moving obstacles have been drawn in the hidden free space Fig 3.14(a): the planned paths are shorter and keep further away from the obstacles. The position of the new entering obstacle is *caught* by one of the hypotheses (Fig 3.14(c)).

3.6 Conclusions

We presented an algorithm to perform navigation in dynamic environment which integrates a probabilistic representation of the uncertainty due to perception and the approximation of the motion models. Results show that the algorithm is able to compute safe trajectories in real time taking into account the static and moving obstacles perceived and the uncertainty in prediction of a real

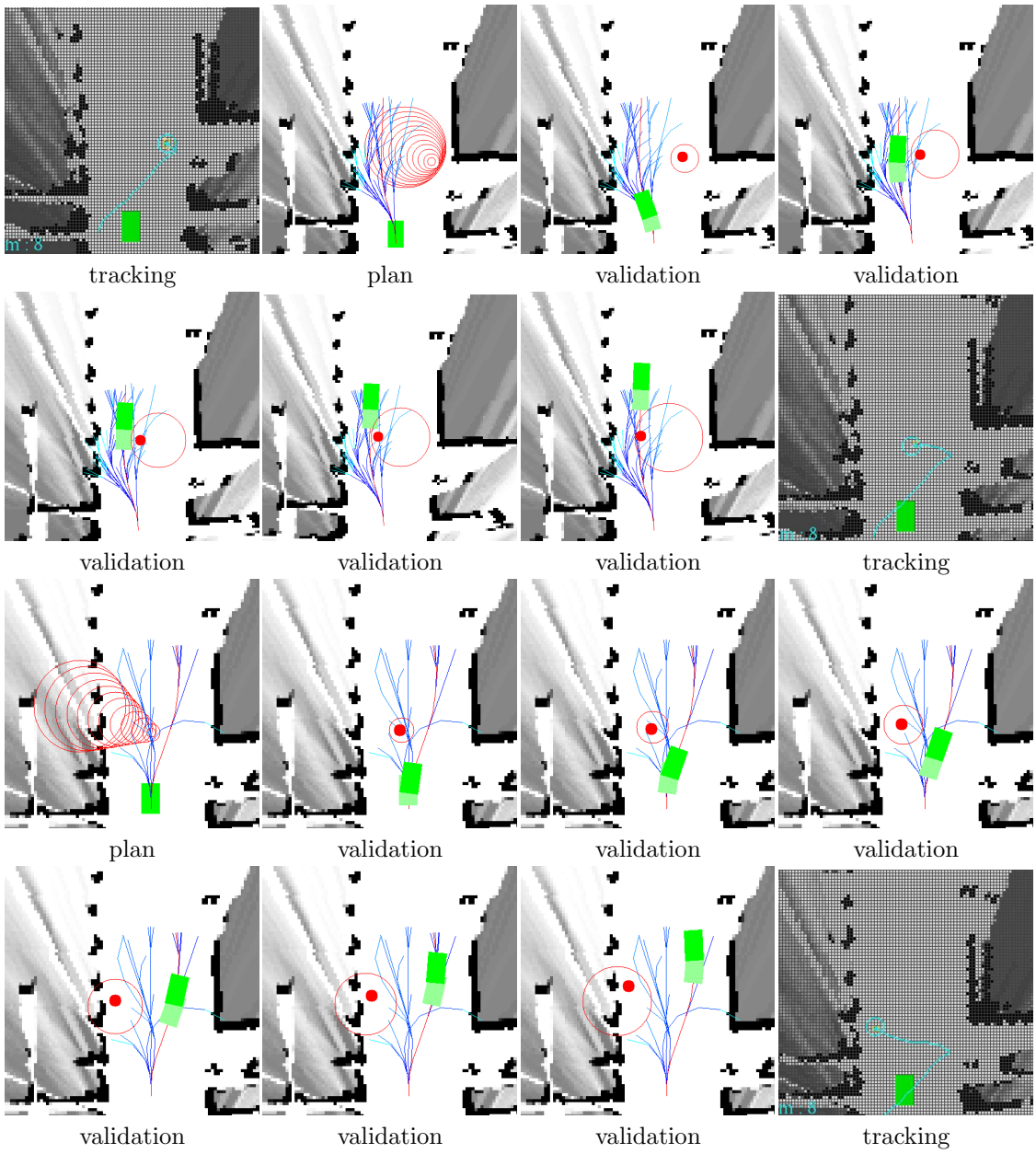


Figure 3.10: Results with a real dataset. The static environment is mapped and the moving obstacles are tracked (tracking). The algorithm explores the state space and chooses a trajectory (plan). The chosen trajectory is compared with the real acquired observations (red full circles) and with the prediction (red empty circles) (validation).

dataset. constraints imposed by the dynamic environment as it is an anytime algorithm: at each time step a solution is ready for the robot to perform safe and efficient navigation. The algorithm

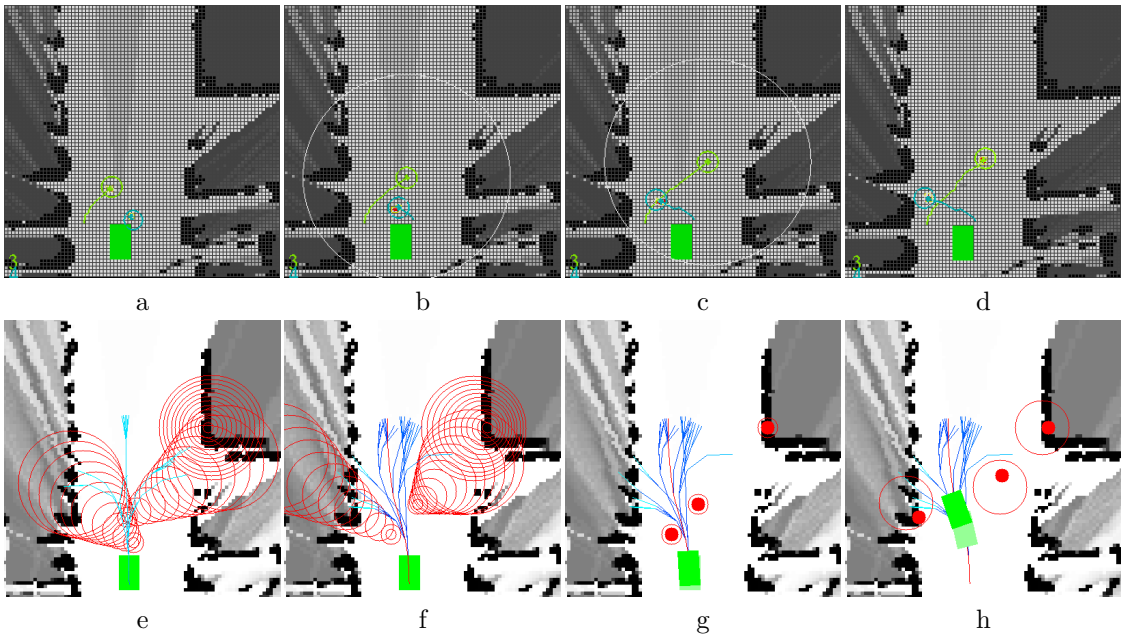


Figure 3.11: Partial planning in presence of two pedestrians: the Cycab is stoppped and obstacles cross in front of it (a-d). At the beginning no movements are possible (e), but in next timesteps a safe trajectory is found (f). The trajectory is validated with observations from tracking (g-h).

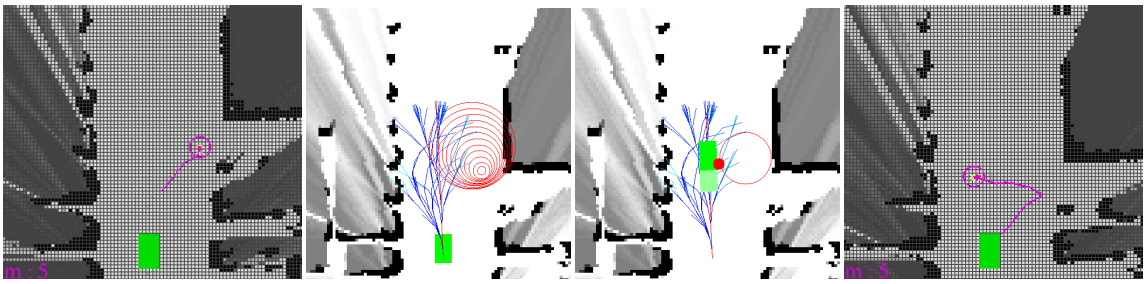


Figure 3.12: An obstacle waiting a bit and changing direction: the prediction is false and the resulting planning leads to collision.

respects the time constraints imposed by the dynamic environment as it is an anytime algorithm: at each time step a solution is ready for the robot to perform safe and efficient navigation. The prediction are linear predictions based on a constant acceleration motion models. The simplicity of these models leads to not very accurate predictions, and short time validity. The use of more complex models (Interactive Multiple Models [61] ...) could give better results in terms of prediction. However, prediction based on target tracking remains valid for a short time period. This fact makes the robot enlarge the tree of search in the nearspace and plan shorter paths. This translates in a

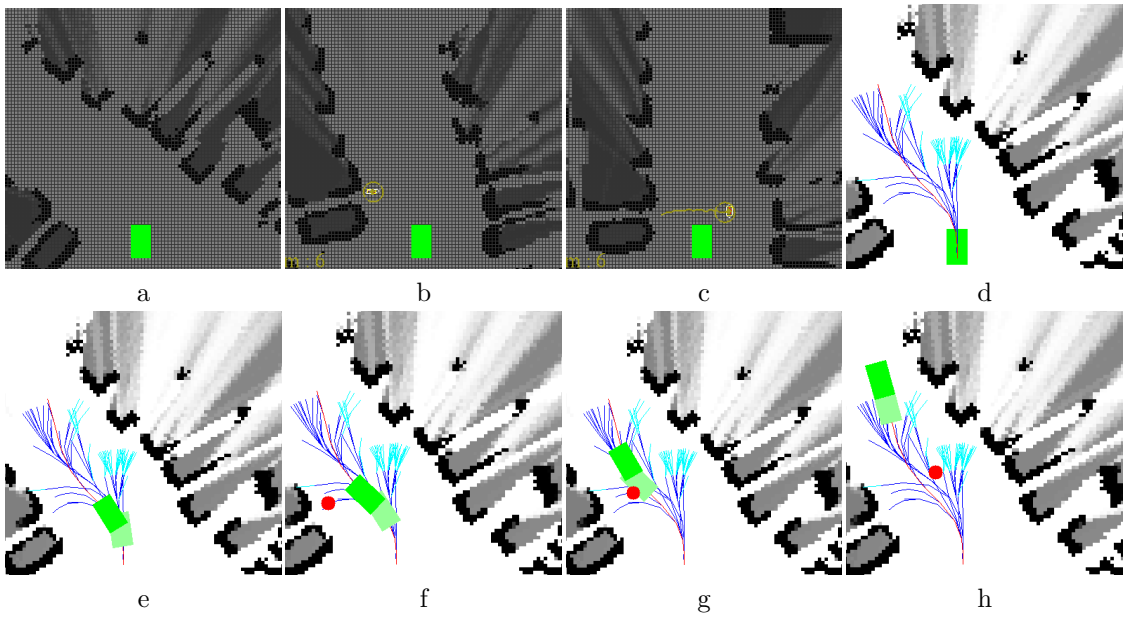


Figure 3.13: The partial path is computed without obstacles (a,d). An obstacle suddenly appears (b-c), the partial path becomes dangerous (d-h).

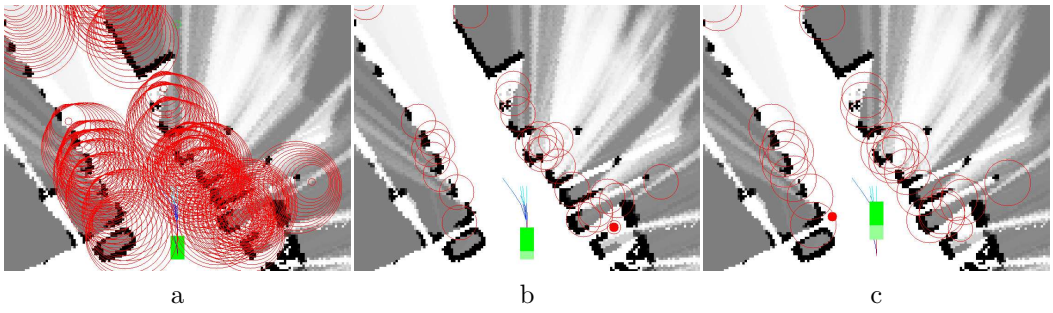


Figure 3.14: The hypothesis of hidden obstacles entering the scene is taken into account. (a) The prediction hypotheses, the searched tree and planed path. (b-c) validation of the planned path: (c) a new obstacle appears on the left, around the predicted position.

less efficient navigation than in static or more simple environment.

Chapter 4

Navigation among obstacles with typical motion models

Up to this point, we have looked at moving obstacles as non-rational objects which move freely in the environment. In the previous chapter, we made a simple hypothesis on the dynamics of these objects and their motion model was estimated only by mean of their immediate previous observations. However, pedestrians and vehicles moving in a given environment usually do not move at random but often engage in typical behaviours or motion patterns. Pattern based approaches give more reliable predictions in the medium-term with respect to the kinematic models. This information can be used to improve the navigation performance of robots.
In this chapter we extend our probabilistic partial planning to handle a medium-term prediction coming from pattern based motion models. In Section 4.1 and Section 4.2 we briefly present the general techniques to learn, represent and predict typical patterns, and the state of the art in motion planning among obstacles with known typical motion. In Section 4.3, we present how we adapted our probabilistic motion planning to the Growing Hidden Markov Model representation, as developed in [63] (§4.3.1) and to the Gaussian Process representation, as developed by Tay [64] (§4.3.2) and we close the chapter with Section 4.4 presenting simulation results.

4.1 State of the Art

Kinematic and dynamic models of the kind used in Chapter 3 explain the motion of objects in terms of physical properties and laws; they describe the object's motion as the evolution of its state when it is subject to a *control*. These approaches present some important limits:

1. kinematic models are incomplete;
2. objects are able to change their control dynamically.

The incompleteness comes from the fact that these models ignore the external factors that influence motion, for instance static obstacles. Secondly, most categories of dynamic obstacles can change dynamically their control: this is the case of pedestrians, human driven vehicles, robots ... For these reasons, motion models based on simple kinematical transition functions produce predictions that are sound only in a short time interval.
Another family of approaches is based on the fact that in a given environment, rational obstacles usually move with a specific intention. This assumption is related to the existence of typical motion patterns across the environment. The idea is to observe the environment and learn the possible typical patterns realized. Such motion patterns can be used as motion models for prediction (*pattern based motion models*).
An early work on navigation in changing environments and in presence of typical motion presents the idea to divide the state space in *hazardous* and *shelter* regions [1]. A shelter designates an area in which the robot is guaranteed to avoid collision, while an hazardous region designates an area in which other obstacles can move. The cost of traversing an hazardous or dynamic region directly corresponds to the risk of encountering a moving obstacle.
In more complex environments however, this representation may reveal too poor: there may be no shelter areas at all, or they can be interleaved with the hazardous ones, so that having a spatial and temporal hint of where moving object actually are becomes a necessity. The learning of typical patterns and the representation of pattern based motion models has been the subject of extensive study and many different approaches have emerged; roughly, we can distinguish the techniques of representation in two main categories: Trajectory Prototypes and Discrete State-Space Models.

4.1.1 Trajectory Prototypes

Trajectories observed in the environment are grouped in clusters by similarity. To each cluster corresponds a typical motion pattern and a single trajectory prototype is used to represent the cluster.

Representation Typical trajectories are usually represented as a sequence of points in the continuous state-space. Most approaches do not model the time variable explicitly and assume that the points in the sequence are regularly spaced in time. Sometimes, a measure of the "width" of the cluster is also included in the representation [65, 66]. In [67] a probabilistic model is proposed in which the width of the trajectory is represented as the variance of the distance between the trajectories that belong to the same cluster. Another probabilistic model of width has been proposed by [68]: every point of the trajectory prototype is modelled as a Gaussian and it is assumed that all such Gaussians have the same covariance matrix. A novel approach has been proposed by Tay in [64] where trajectories are represented by Gaussian Processes. In this case both the typical trajectory and its "width" (mean and covariance) are probabilistically estimated by a proper Bayesian framework. This work is recalled with more

detail in §4.3.2.

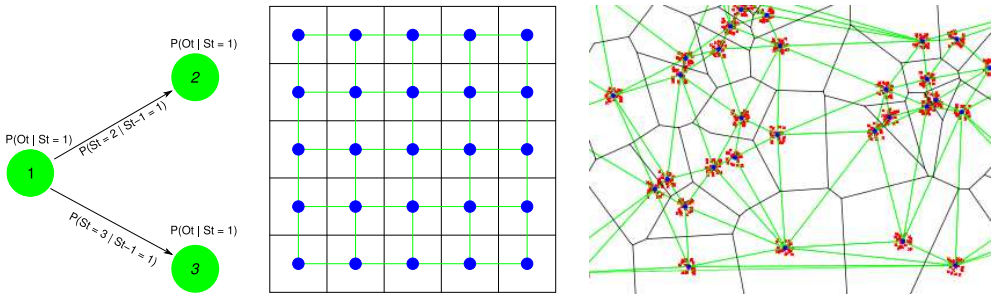
Learning In the learning phase, one has to: a) determine the number of clusters; b) find clusters; and c) build trajectory prototypes.
The clustering algorithms can be classified in model based and pairwise clustering techniques. Given the number of clusters, the data are classified running Expectation Maximization (EM, [69]) in order to maximize some global measure (*eg* data likelihood). Model based clustering algorithms need to know *a priori* the number of clusters: some approaches simple ignore the problem [70], while others use a greedy search to propose an initial guess and then add or delete clusters [68]. Most model based clusterings assume vectors of equal length in input: resampling is then necessary, so that trajectories of different length present the same number of input points.
In the pairwise clustering techniques, data elements are processed by pairs using a *distance* measure: if the data are similar enough, they are grouped in the same cluster, otherwise another cluster must be initialized. These approaches automatically determine the number of clusters, but do not offer any guarantee of satisfying a global optimality criterion.
Once the clustering process is over, the typical trajectory and eventually its width is determined choosing respectively the mean trajectory and a deterministic envelope or a variance measure of the cluster.

Prediction consists essentially in finding the cluster which best corresponds to the partially observed trajectory and using the prototype as the prediction of future motion. In [64] and [67] different approaches are proposed to return a probability distribution on all the clusters according to their similarity.

Navigation Literature on navigation with pattern based motion models is quite poor. We can cite [71], where the robot applies an A^* algorithm to find a path on a 2D space cost grid: the cost of passing through a cell at time t is given by the probability of collision plus the probability that a person covers the cell at that time. The algorithm is applied in an office-like environment where each typical pattern is represented by a fixed number of Gaussians that specify the probability of a position at a stage of the trajectory. Replanning is performed whenever information changes.

4.1.2 Discrete State-space Models

The basis of discrete state-space approaches are **Markov chains**, which represent time as a discrete variable and model motion in terms of a finite number of discrete states. Markov chains may be represented by directed graphs, where nodes represent discrete states and edges represent transitions from one state to another in a single time step (see Fig. 4.1(a)). Each edge is labeled with the transition probability, which follows the Markov assumption: knowing the present state, the future and



(a) (b) (c)

Figure 4.1: (a) A basic three-state HMM, with transition probabilities attached to edges and observation probabilities attached to nodes. (b-c) Discretizations of the space: (b) a uniform HMM graph: each state corresponds to a cell and an edge to a cell border; (c) a non-uniform discretization (Voronoi regions) which minimizes data distorsion.

past states are independent. Observation uncertainty is addressed using a discrete Bayes filter: the Hidden Markov Models (HMM) [72]. Transition and observation probabilities are often represented as multinomial and Gaussian probabilities.

Representation The HMMs represent a continuous state space through a discrete approximation in a *topological map*. Fig. 4.1 shows two possible space discretizations: in figure (b) the space is partitioned in a uniform cell grid: the corresponding topological map presents a node for each cell in the grid and an edge through each cell border; in figure (c) the partition of the space is obtained by a vector quantization algorithm on the basis of the observation, so that the distortion of the data is minimized. The subregions in which the space is partitioned are called Voronoi regions:

$$\mathcal{V}_j = \{x \in M | d(x - c_j) \leq d(x - c_i) \forall i\} \tag{4.1}$$

where M is the data manifold, $C = \{c_1 ... c_k\}$ is a finite set of reference vectors and $d(x, c - i)$ is a distance measure such as the square error or the Euclidean distance. Among the quantization algorithms we can cite the k-means algorithm [73], the Self Organizing Map [74] and the Topology Representing Networks [75], which work with an *a priori* defined number of regions; the Growing Neural Gas [76] and the Instantaneous Topological Map [77] take into consideration a variable number of units.

Motion patterns are usually represented as chains with directed edges going only in one direction. Every chain constitutes an individual HMM [78] or all patterns can be regrouped in a single HMM formed by many non-connected chains [79].

Learning In the learning phase one has to: a) learn the graph structure b) estimate the parameters of the different probability distributions.

The standard learning technique for HMMs is the Baum-Welch algorithm [80], a specialization

of the well known EM algorithm. This is however a parameter-only learning algorithm, and the model's structure is assumed to be *a priori* known. The classic algorithm is designed for off-line use. Structure-learning and incremental extensions of the Baum-Welch algorithm are presented in [81] and [82]. In [63] a *learn and predict* technique is presented where a topology learning network and an incremental parameter learning algorithm are integrated and evolve on-line with the available observations. This work is recalled with more detail in §4.3.1.

Prediction Applying Bayesian inference, the belief of the object's state is updated with the newly gathered observations and prediction is obtained by projecting the belief into the future. This may be done on state by state and observation by observation basis; however, depending on the HMM's structure, exact inference may not be fast enough to enable realtime use, and approximate methods are applied [78].

Navigation In [79] pre-learned motion patterns are represented by Hidden Markov Models. As in the case of trajectory prototypes, the best path is planned using A^* in a space-time grid where the cost of traversing a cell is given by the probability of collision with static obstacles plus the probability that a moving obstacle covers the cell.

4.2 Motivation

The aim of this chapter is to use our probabilistic partial planning with pattern based motion models. In Chapter 3 we underlined that the prediction based on target tracking only was limited to a short-time period and to linear behaviour of the obstacles. Once we have introduced typical patterns, the predictions are expected to be more reliable in the medium term and the behaviour of the obstacles to be better taken into account by the robot; in [79], the use of these motion models is shown to improve the navigation performance in comparison with the case of a model based only on target tracking.

Presented navigation methods based on typical patterns use complete methods (A^*) to compute a global plan and replan whenever possible. However, as stated in the previous chapter, the problem of A^* and of all complete methods is that the computational time depends on the environment structure and obstacles: these methods are more adapted to low dynamic environments, where the information does not change frequently, the obstacles velocity is limited and the robot can stop often and plan its future movements. The complete algorithms can also be usefully employed when the static environment is limited to a restricted known area. In a highly dynamic environment we need an *anytime* algorithm, which is able to give a feasible solution when the information changes (a new obstacle enters the scene, an obstacle change its behaviour). Among the developed approaches we have chosen to test our probabilistic planning with two frameworks: the Growing HMM framework and the Gaussian Processes representation. The advantages of the Gaussian Processes representation with respect to other trajectory propotypes approaches is that they present a solid probabilistic

theory for the representation of the mean and covariance of the different paths and the future prediction. Also, trajectories are represented by continuous functions, which allows to use different time steps for prediction and for observation, thereby limiting the necessary interpolations to the learning phase. Finally, the Gaussian representation allows the prediction to be very fast and computationally cheap.
We tested the algorithm also with an discrete approach based on HMMs. We consider the pattern representation developed in [63]: the patterns are represented in terms of "goals" i.e. of positions to reach in the environment: this helps explaining some unusual paths of the obstacles as pieces of typical paths, which is useful in prediction. Also, the rational process of obstacles under motion becomes similar to the one of the robot, and may be applied in perspective to a multiple robot case. Finally, the GHMM framework is a "learn and predict" approach, so that there is no need of a previous learning phase and the set of typical patterns adapts to the changes of the environment. We will not directly use this property in this chapter, however it is important to notice that this feature would be interesting for a robot operating in a quasi-static environment, where the static structure of the environment changes "slowly" in time, modifying the typical patterns of obstacles and the robot could communicate with the learning architecture to update on-line the set of patterns and the reliability of prediction.

4.3 Partial Motion Planning with typical trajectories

At a given instant, the robot knowledge about the state of the environment is represented by:

1. An occupancy grid: represents the structure of the static environment around the robot, according to the previous observations;
2. A list of moving objects, their estimated position, velocity and previous observations;
3. An estimation of the state of the robot (position, velocity);
4. A set of typical patterns which represent the motion models for the obstacles;
5. A goal state.

The occupancy grid and the state of the robot are estimated by a Simultaneous Localization And Mapping algorithm based on scan matching [57]. The position, velocity and track of the obstacles is estimated thanks to a Multiple Target Tracking algorithm, as in Chapter 3. The typical patterns are supposed to be learned by an off-board platform and to be known by the robot. The motion planning algorithm used is the same presented in Chapter 3. However, slight modifications have been used in order to take advantage of the medium-term prediction. Paragraph 4.3.1 presents the Hidden Markov Model approach and how the Partial Probabilistic Planning is adapted to the

prediction issued by such a model. The same is done in paragraph 4.3.2 for the Gaussian Processes approach.

4.3.1 Motion planning with Hidden Markov Models

The Growing Hidden Markov Models [63] is a "learn and predict" approach for typical pattern prediction: as more observations are available, the structure of the HMM graph evolves to fit the new observations. For the moment we will consider however that the current structure of the graph is stable and known by the robot: in this case, the approach is more similar to a classic discrete-state approach. The model makes the assumption that obstacles move in the environment with the aim to reach a specific goal. The states of the model are extended states composed by the position, velocity and intended goal of the obstacles. In this way, paths with different velocities (for instance traversed by pedestrians and cars) are attached to different graphs even if they follow the same path. In the learning phase, the goal is identified by the ending state of the observed trajectories and each typical pattern is represented by a Markov graph. Each graph is composed by all the different trajectories that lead to the same goal: this representation is more synthetic that the ones used in classical discretized approaches, where a pattern is identifyied with essentially a sequence of positions or states and has the advantage to explain unusual paths formed by "parts" of observed typical patterns. When more than one typical pattern is present, a set of graphs is used and they are all grouped in a single HMM.

Observation model

Only the position is directly observed. The observation model is:

$$P(O_t|S_t) = P(O'_t, O''_t|S_t) = P(O'_t|S_t)P(O''_t|S_t) \quad (4.2)$$

$$P(O'_t|S_t) = G(\mu_t, \Sigma) \quad (4.3)$$

$$P(O''_t|S_t) = U_{O''_t} \quad (4.4)$$

$$P(O_t|S_t) = \frac{1}{Z}G(\mu_t, \Sigma) \quad (4.5)$$

where O' represents the observation over the position and O'' represents the observation of the velocity and the goal. The two observation components are assumed to be independent (eq. 4.2): the observation model attached to the position is modelled by a Gaussian (eq. 4.3); since the velocity and goal are not observed, the corresponding observation model is represented by a uniform distribution (eq. 4.4): the same probability value is given to all velocities and goals, so the expression can be rewritten substituting a normalization constant Z for the uniform distribution (eq. 4.5).

Estimation step

Given an object observation O_t, the belief of the state S_t is reestimated by a discrete Bayesian filter.

$$P(S_t|O_{t-1}) = \sum_{S_{t-1}} P(S_t|S_{t-1})P(S_{t-1}|O_{t-1}) \tag{4.6}$$

$$P(S_t|O_t) = \frac{1}{Z}P(O_t|S_t)P(S_t|O_{t-1}) \tag{4.7}$$

The estimation of the extended state $P(S_t|O_t)$ is obtained by multiplying the likelihood of the observation $P(O_t|S_t)$ by the one step ahead prediction $P(S_t|O_{t-1})$; Z is a normalization constant. This last one is represented by the discrete probability distribution function in equation 4.6 over the nodes of the graph: the estimation at $t-1$ is propagated one step ahead using the transition probabilities $P(S_t|S_{t-1})$ with which the edges of the graphs are labelled (see Fig.4.1(a)). The extended state is not directly observable, the likelihood of the prediction $P(O_t|S_t)$, instead, depends only on the position of the obstacle. The velocity and the goal are estimated as a consequence of subsequent observations and estimations over the different Markov graphs.

Forward Prediction

Forward prediction is performed by propagating the estimated state to the required time horizon $t+k$:

$$P(S_{t+k}|O_t) = \sum_{S_{t+k-1}} P(S_{t+k}|S_{t+k-1})P(S_{t+k-1}|O_t) \tag{4.8}$$

The sum over the set of S_{t-1} is the sum over the nodes of all the graphs, considering the possible positions, velocities and goals. The motion model $P(S_t|S_{t-1})$ is given by the probabilities with which the edges of the graphs are labelled (see Fig.4.1(a)). Forward prediction is performed recursively. To exploit the advantages of the medium term prediction, the prediction, planning and update should be fast enough to allow the search tree to grow in far future time. Since the exact inference may be too costly in our case, a probability threshold is chosen and the distributions are truncated when they fall below the threshold; the number of states to consider is reduced.
With respect to a target tracking based motion model, the prediction issued from HMM graphs takes into account the static structure of the environment and the intentions of the objects, leading to non-linear models and multimodal distributions that explain better the behaviour of the obstacles in the medium-term.

Planning

From the point of view of navigation we are interested in the probability of a particular point in the state space to be occupied at a certain time in the future. This may be seen as the probability to

observe a given state in the future and is computed from the predicted state:

$$P(O_{t+k}|O_t) = \frac{1}{Z} \sum_{S_{t+k}} P(S_{t+k}|O_t) P(O_{t+k}|S_{t+k}) \tag{4.9}$$

where $t + k$ is the considered time horizon, S the state, O the observation and Z a normalization constant. Given the state of the robot, the probability of collision is obtained by considering independent collisions for the static environment and for each object separetely. We proceed as shown in 3.4.3: while for the static environment the occupancy grid $\mathcal{G}$ is considered, for the moving objects, the prediction based on HMM is considered. Our implementation considers a uniform discretization of the space, of the kind in Fig.4.1; if the step and the alignement of the graph is the same of the one of the static grid, a space-time grid can be obtained by combining the static probability of occupation with the prediction. However this is not the general case, as the static grid is built by the robot during exploration, and its resolution depends on the computation capabilities and sensor quality, while the HMM graph is built by an off-board plateform during the learning phase and could be used indifferently by various robots with different settings. A uniform probability of occupation is considered inside each region. The original GHMM approach provides a Voronoi region graph, where the distribution of observations inside each region is expected to be a normal distribution around a mean. The generalization from a uniform grid to a Voronoi region structure is easily achieved by considering the irregular polygons of the regions instead of the uniform grid. In this case, given the set of regions traversed by the robot and their mean and covariance, the probability of collision is obtained by approximating the integral with the sum over a set of samples. The total probability of collision is computed following equations 3.19 and 3.20 that are here recalled:

$$P(t_{coll} \in T|o_1...o_N) = 1 - \prod_{n=1}^{N} (1 - P(t_{coll} \in T|o_n)) \tag{4.10}$$

$$P(t_{coll} \in T|o_1...o_N, \mathcal{G}) = P(t_{coll} \in T|\mathcal{G}) + (1 - P(t_{coll} \in T|\mathcal{G})) \cdot P(t_{coll} \in T|o_1...o_N) \tag{4.11}$$

4.3.2 Motion Planning with Gaussian Processes

Gaussian Processes

A Gaussian process is a generalization of the Gaussian probability distribution in function space. Given the set of Gaussian distributed random variables $\{f(x_1), f(x_2), ..., f(x_N)\}$, it can be represented mathematically using the mean function and covariance function [83]:

$$f(x) \sim G(m(x), k(x, x')) \tag{4.12}$$

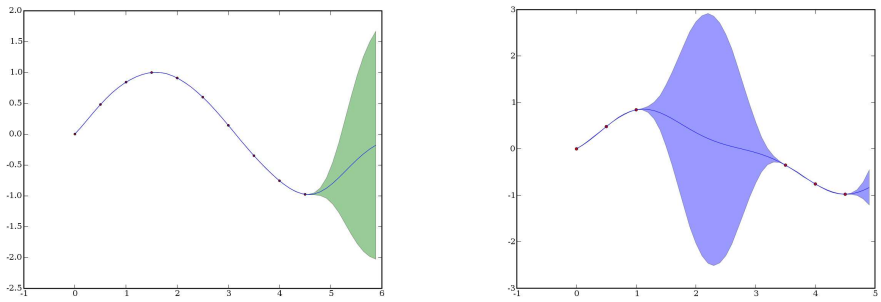
$$m(x) = E[f(x)] \tag{4.13}$$

$$k(x, x') = E[(f(x) - m(x))(f(x') - m(x'))] \tag{4.14}$$

Where $G(\mu, \Sigma)$ represents a Gaussian distribution with mean μ and covariance Σ. $k(x, x')$ is the covariance function with domains from the input space. $E[.]$ stamds for expected value. A pertinent property of the covariance function is that it has to be positive semidefinite. In [64], the squared exponential covariance function is adopted:

$$k(x, x') = \theta_0^2 exp - \frac{(x - x')^2}{\theta_1^2} + \theta_2^2 \delta(x - x') \tag{4.15}$$

Where $\delta(\cdot)$ is the Dirac delta function. The chosen covariance function is stationary since it is a function of the difference, $x - x'$. Thus, covariance of paths are assumed to be the same throughout the input space. $\Theta = \{\theta_0, \theta_1, \theta_2\}$ are parameters for the covariance function, also known as the hyperparameters for the Gaussian process. The hyperparameters influence the form of the Gaussian process. Intuitively, the hyperparameters set a characteristic length scale for the Gaussian process covariance function and this length scale can be viewed as how much the difference $x - x'$ in input space has to be before there is a "significant"' change in $f(x) - f(x')$. A short characteristic length scale gives a highly fluctuating Gaussian process whereas a longer length scale gives lower fluctuations. In the GP framework instead, trajectories are represented by continuous functions: the



a b

Figure 4.2: (a) Future prediction (green area) given by GPs in the case of a partial observed trajectory; (b) Regression (purple area) on a portion of unobserved trajectory.

typical velocity is learned and represented in the trajectory prototype; predictions at a given instant t are obtained using the ratio between the typical and the observed velocities.

Pattern Representation

It is assumed that the observation of paths belonging to an exemplar path are generated by a Gaussian process. The movements in the x and y axes are assumed to be independent, so each prototype is represented with 2 Gaussian processes, one for each axis respectively. The mean of these Gaussian processes is the mean function representing the path.

A single observation of a path is represented as two vectors (x_n, y_n) of dimension D, where D is the number of positions observed along the path. One vector represents the sequence of positions along the cartesian x axis and the other for the corresponding sequence in the y axis. The likelihood based on the N training sequences (x_n, y_n) is then:

$$L_x = \prod_{n=1}^{N} G(x_n|\mu_x, \Sigma_x) \tag{4.16}$$

$$L_y = \prod_{n=1}^{N} G(y_n|\mu_y, \Sigma_y) \tag{4.17}$$

Where x_n and y_n are vectors of x and y positions for the nth observation. μ_x, μ_y, Σ_x and Σ_y are the mean vectors and covariances of the x and y positions for the typical path. In almost all cases, the sequence of observations of positions are of different length for different path observations. In this case, a fixed dimension D can be chosen and the D positions can be obtained by choosing D points uniformly distributed along the interpolated path.
Since each observed path corresponds to two D dimensional vectors and a typical motion path is Gaussian distributed, the observed paths are Gaussian distributed from the generative point of view. A single typical motion path is a D dimensional Gaussian distribution and this can be easily extended to the case of representing several typical motion paths using a mixture model. Considering K components, the likelihood based on the N training data sequences is then:

$$P(x|z, \mu_x, \theta) = \prod_{n=1}^{N} \prod_{k=1}^{K} G(x_n|\mu_{x,k}, C(\theta_k))^{Z_{nk}} \tag{4.18}$$

$$P(y|z, \mu_x, \theta) = \prod_{n=1}^{N} \prod_{k=1}^{K} G(x_n|\mu_{y,k}, C(\theta_k))^{Z_{nk}} \tag{4.19}$$

In these equations z is the vector of component weights, θ the Gaussian process hyperparameters, μ the mean function of the Gaussian process, and $C(\theta_k)$ the covariance matrix of the Gaussian process parameterized by θ. The complete model is a hierarchical probabilistic model where random variables include cluster component weights and cluster means unlike the standard Gaussian mixture model.

Learning

As the hierarchical model is intractable, approximate learning methods are used. For practical purposes, it is desirable to have point estimates for the cluster component weights and the hyperparameters of the Gaussian processes. This leads to a learning approach closely related to Expectation-Maximization (EM) where in the E-step, variational mean field type learning is used. In the M-step maximization of the log likelihood over the space of parameters is performed. To

determine the number of cluster components, training is performed by assuming a large number of cluster components. The training result gives weights for each cluster, and the clusters with small weights are removed. For details, refer to [64].

Prediction

When performing path prediction, the input is a partially observed path of dimension $M < D$. For the case of a D dimensional Gaussian with x_1 of dimension M and x_2 of dimension $D - M$:

$$P'(x_1, x_2) \sim G\left(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}\right) \tag{4.20}$$

The probability of a partial path observation of dimension M belonging to a Gaussian of dimension D is evaluated by integrating over the $D - M$ dimensions of the Gaussian distribution to yield the marginal Gaussian distribution:

$$P'(x_1) \sim G(\mu_1, \Sigma_{11}) \tag{4.21}$$

This probability is evaluated for each cluster k. The prediction of a path x_2 given observation x_1 can be obtained by the Gaussian conditional distribution for each cluster k:

$$P'_k(x_2|x_1) \sim G\left(\mu_2 + \Sigma_{21}\Sigma_{11}(x_1 - \mu_1), \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{21}^T\right) \tag{4.22}$$

The prediction obtained considering all the clusters is then a Gaussian mixture: the weigth w_k of each cluster k is given by the likelihood of the observation sequence x_1 given the considered Gaussian process normalized over the set of clusters:

$$P(x_1|G_k) = G(x_1, \mu_{k,1}, \Sigma_{k,11}) \tag{4.23}$$

$$w_k = \frac{P(x_1|G_k)}{\sum_{k' \neq k} P(x_1|G'_k)} \tag{4.24}$$

where $G(x_1, \mu_{k,1}, \Sigma_{k,11})$ is the Gaussian Process k, $G(\mu_k, \Sigma_k)$ evaluated for sequence x_1 (see also equation 4.21). In order to reduce the number of considered clusters, either an *a priori* maximum number of hypotheses is fixed and the ones with the higher weights are retained, either an appropriate threshold is defined according to the Mahalanobis distance betwwen the observation sequence and each GP. Figure 4.3.2 shows an example of the prediction where the predicted path mean and variance are represented by 'bars'. Clusters for prediction were selected according to the χ-square statistic corresponding to the 95% confidence interval. For each cluster, the Gaussian distribution of the predicted path can be obtained using equation 4.22.
Considering a moving robot observing the moving object, the sequence of observations may be related not only to the beginning of a trajectory, but to any sub-segment of the typical pattern. In

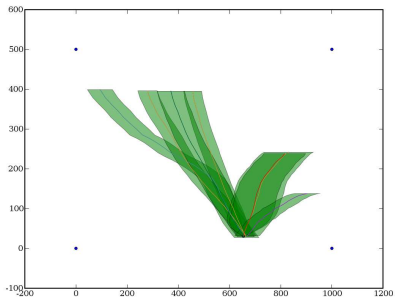


Figure 4.3: A prediction based on Gaussian Processes: red lines are the means and green bars represent the variance. The chosen set of GPs represents the 95% confidence interval.

this case, before evaluating the probability that the partial path observation belongs to a typical trajectory representation it is necessary to estimate which is the interesting part of the trajectory, i.e. at which point of the learnt trajectory the observation has begun. In our implementation, the sequence of observations is translated into the typical pattern and the minimum square error is found: the corresponding position of the partial path is considered for the probability computation. To account for different object velocities, the mean velocity of the observed object is computed on the basis of the observation sequence and their rate; each pattern presents a typical step given by the uniform sampling used in the learning phase. The ratio between the typical step and the object velocity is used to compute the prediction at the desired time.

Planning

The set of typical patterns is represented by the list of associated Gaussian Processes. Let us consider M moving objects and K the learned typical patterns. For an obstacle O_m, the predicted position at time t is estimated by a Gaussian mixture of $K_1 \leq K$ components, where K_1 is the number of typical patterns that have been maintained for prediction after gating. Considering each component k separately, the associated probability of collision $P_{cd}(k, m)$ is calculated considering the integral of the predicted Gaussian over the area swept by the robot.. In practice, the probability distribution is sampled and the integal approximated. For a single obstacle m and a Gaussian process k :

$$P_\pi(s_N, m, k) = 1 - \prod_{n=0}^{N} (1 - P_{cd}(s_n, m, k)) \tag{4.25}$$

Considering first all the GPs for one moving object and then integrating on all the objects we have:

$$P_\pi(s_N, m) = \sum_{k=1}^{K_1} l_{k,m} \cdot P_\pi(s_N, m, k) \tag{4.26}$$

$$L_\pi(s_N) = \prod_{m=1}^{M} (1 - P_\pi(s_N, m)) \tag{4.27}$$

where we considered independent collision events for each obstacle. L_{π_N} is the probability that the robot traverses the path without entering in collision. From this likelihood, the paths are weighted and the best is chosen as explained in §3.3.

4.3.3 New obstacles entering the scene

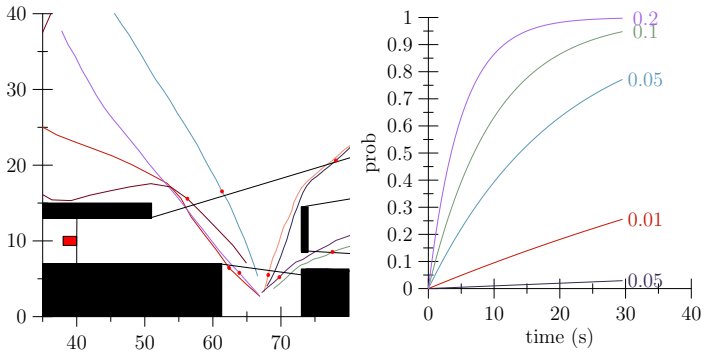


Figure 4.4: Points from where new obstacles are expected to enter the field of view of the robot and evolution in time of the probability that some new obstacles enter with varying λ.

In contrast with what was said in the precedent chapter (see §3.4.2), it is not necessary in this case to recognize on-line the *doors* in the environment. As the obstacles are supposed to move along typical patterns, they can enter the field of view only from points along these patterns. Their motion model is again defined by the motion pattern itself. As in §3.4.2, the probability that at least one obstacle enters the scene is given by:

$$P[N_k(t+\tau) - N_k(t) \geq 1] = 1 - e^{-\lambda_k \tau} \tag{4.28}$$

where $N_k(t+\tau) - N_k(t)$ is the number of pedestrians entering from typical pattern k in time interval $(t, t+\tau]$. A different frequency coefficient λ_k is chosen for each typical pattern k. These coefficients can effectively be defined on the basis of the observations gathered in the learning phase, and can change in time: for example it is possible to consider different coefficients for different hours during

the day or different days in a week etc... The obstacles are then supposed to move along one pattern or another with probability proportional to the learned λ_k. Figure 4.4 shows an example of where moving obstacles are expected to enter the field of view relatively to the current observation point of the robot and to the pre-learned typical patterns. Black blocks are static objects (walls and tables), the green rectangle is the robot and the coloured lines are the means of the typical patterns (represented here by Gaussian Processes). Green points represent the positions from where new obstacles are expected to enter the scene according to the current field of view, whose contour is drawn in black.

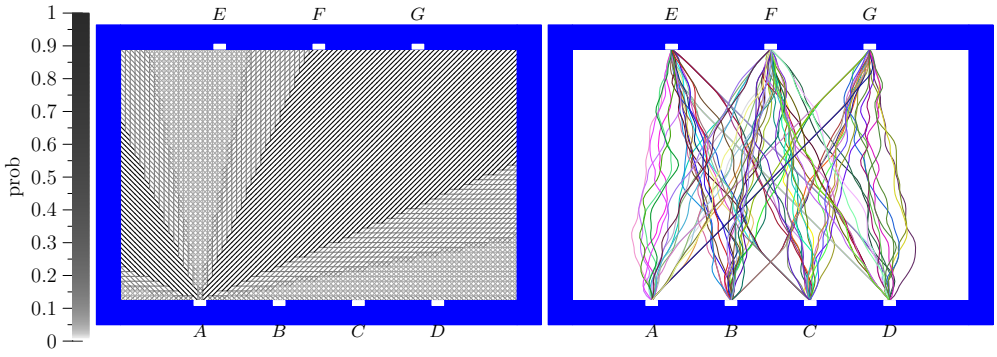
4.4 Examples and performance

We tested the algorithm in simulation for both the HMM and the GP representation. We simulated a robot with dimensions and kinematical constraints similar to the Cycab. The aim of these results is to prove that the planning and navigation algorithms allow the robot to move safely among moving obstacles and that the medium term prediction is useful to generate more intelligent and safer paths. At this aim we simulated perfect perception, localization and execution in order to isolate the problems due to misdetections, sensor limits and actuators inaccuracy, but give a better idea of the performance of the algorithm in ideal conditions. The static environment is deterministically known.

4.4.1 HMM Results

A rectangular environment has been simulated. A certain number of doors is simulated for the two long sides of the rectangle. Obstacles are supposed to enter from a door and to exit by another door in the opposite side. The space has been discretized in a uniform cell grid of step $0.5m$. A 4-connected HMM graph has been built on the grid for each goal: the probability to pass from a state to another depends on the decrease of the distance to the goal between the origin state and the destination one. A certain amount of noise is applied so that states that present nearly the same decrease in distance are given the same probability. The probability is then normalized over the set of edges coming out from the origin node. A set of 1000 trajectories has been randomly simulated on the basis of the graph: for each trajectory the enter door and the exit door are initially chosen. Given a state of the obstacle, the next state is drawn proportionally with the edges probability of the graph correspondent to the chosen goal. The position of the obstacle inside the cell is chosen by a smoothing filter.
The simulated robot has the same dimensions and kinematic and dynamic contraints of the Cycab. Perception is assumed perfect: the obstacles are represented by circles of $0.30m$ radius whose position is always known; no occlusion or finite range is considered. Localization and execution are also assumed as perfect processes. The robot has to cross the environment multiple times avoiding the

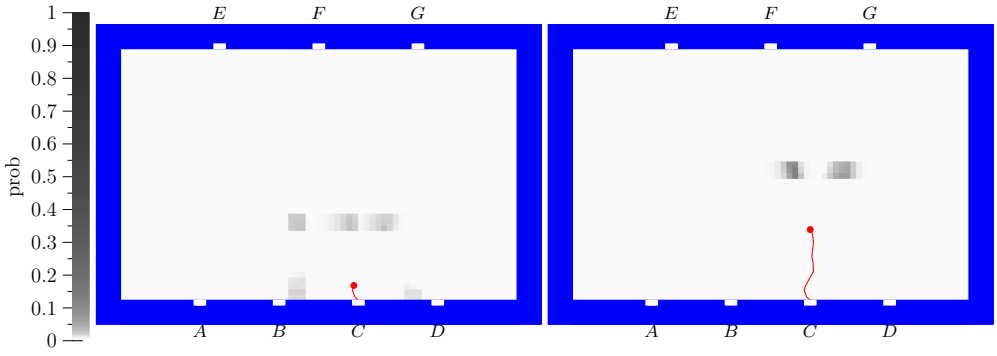
obstacles and reaching successively goals randomly drawn in the rectangle. Figure 4.5(a) shows the environment and the generated graph for goal A (a). The lines are the edges connecting the states of the graph. Only the edges with probability $p > 0.1$ are shown. The direction of the edges is not shown but is intended toward the goal A . Darker lines show connections with stronger probability. In Figure 4.5(b) the set of drawn trajectories is shown.



(a) (b)

Figure 4.5: (a) Example of HMM graph toward goal A; (b) Dataset of trajectories generated.

Prediction



(a) (b)

Figure 4.6: An object and its trajectory is observed (red): predictions for 10 timesteps ahead based on HMM graphs.

Figure 4.6 shows the prediction for 10 timesteps ahead given by the HMM graphs on the basis of previous observations and considering all the possible goals. In contrast with a general Bayesian Filter based on a kinematic motion model, the probability distribution is non-linear and multimodal and depends on the structure of the environment. It is possible to notice how the typical patterns are taken into account: we can see how the probability is distributed toward the possible goals of the object. The red line is the observed trajectory: in (a) multiple goals are possible (A, B, D, E, F, G)

and the prediction is shared between them; in (b), the observed trajectory lets only two goals (F or G) appear more likely.

Planning

Figures 4.7, 4.8 and 4.9 show the robot dealing with multiple obstacles crossing the free environment at successive instants in time: red circles are the obstacles and in black is their trajectory; the robot is the green rectangle, while the green spot is the current goal. Figure 4.7 shows four particular behaviours of the robot that have emerged during navigation. Also if multiple moving obstacles are present, the robot trajectory is "disturbed" by only one or two obstacles at the time. In the first column, Figure 4.7(a), the robot deviates from the initial planned path and accelerates to cross the space before obstacle A passes. In the first image the plan is directed toward the goal; in the second image, the partial path avoids the obstacle by accelerating and passing in front. In the 3-rd and 4-th image, the refinement of prediction allows the robot to improve the path toward the goal. In 5-th and 6-th image, the robot safely overcomes the obstacles and reaches the goal. In the second column, Figure 4.7(b), two sequences are shown: in the first 3 images, the robot deviates its path to let the obstacle pass first; in the last 3 images, the robot slows down and stops (5-th image), lets the obstacles pass and accelerates again to reach the goal (6-th image). The third column shows two sequences where the robot avoids moving along the possible paths of obstacles. In the first 3 images, the robot makes a large turn to avoid encountering obstacle A. In the 4-th image, the robot must adapt at first to obstacle B, then slows down and let obstacles C and D pass along their path before reaching the goal.

Figure 4.8 shows 16 images from an avoiding sequence. At each image, the obstacles that are nearer to interfere with the robot trajectory have been labelled with a letter. Figure 4.9 shows 16 images from a similar sequence with the difference that each time, the planned path has been grown up to a maximum depth of $13s$ instead of $5s$. Since the 3-rd figure, the robot takes into account the obstacles that are coming from the bottom side of the environment; in subsequent images, the path, of varying length, is adapted to the uncertainty in prediction and recursively refined till the robot reaches the goal. The algorithm runs with a frequency of $2Hz$. The robot avoids the obstacles and adapts the path toward the goal taking into account the refinement of the prediction at subsequent instants. Table 4.1 shows results obtained making the robot reach 100 goals in the rectangular environment and simulating various number of pedestrians. Due to the randomized nature of the planning algorithm, experiments have been repeated several times, and the average results are shown. Pedestrians move independently of the robot and of each other, they follow passively a predefined trajectory. A collision is detected when the distance between the rectangular robot and the position of the pedestrian is less than $30cm$. All collisions were detected when the robot is stopped: the robot does not collide with static obstacles and the collisions with moving obstacles happen when the robot is already stopped (third column). The number of *passive* collisions augments with the

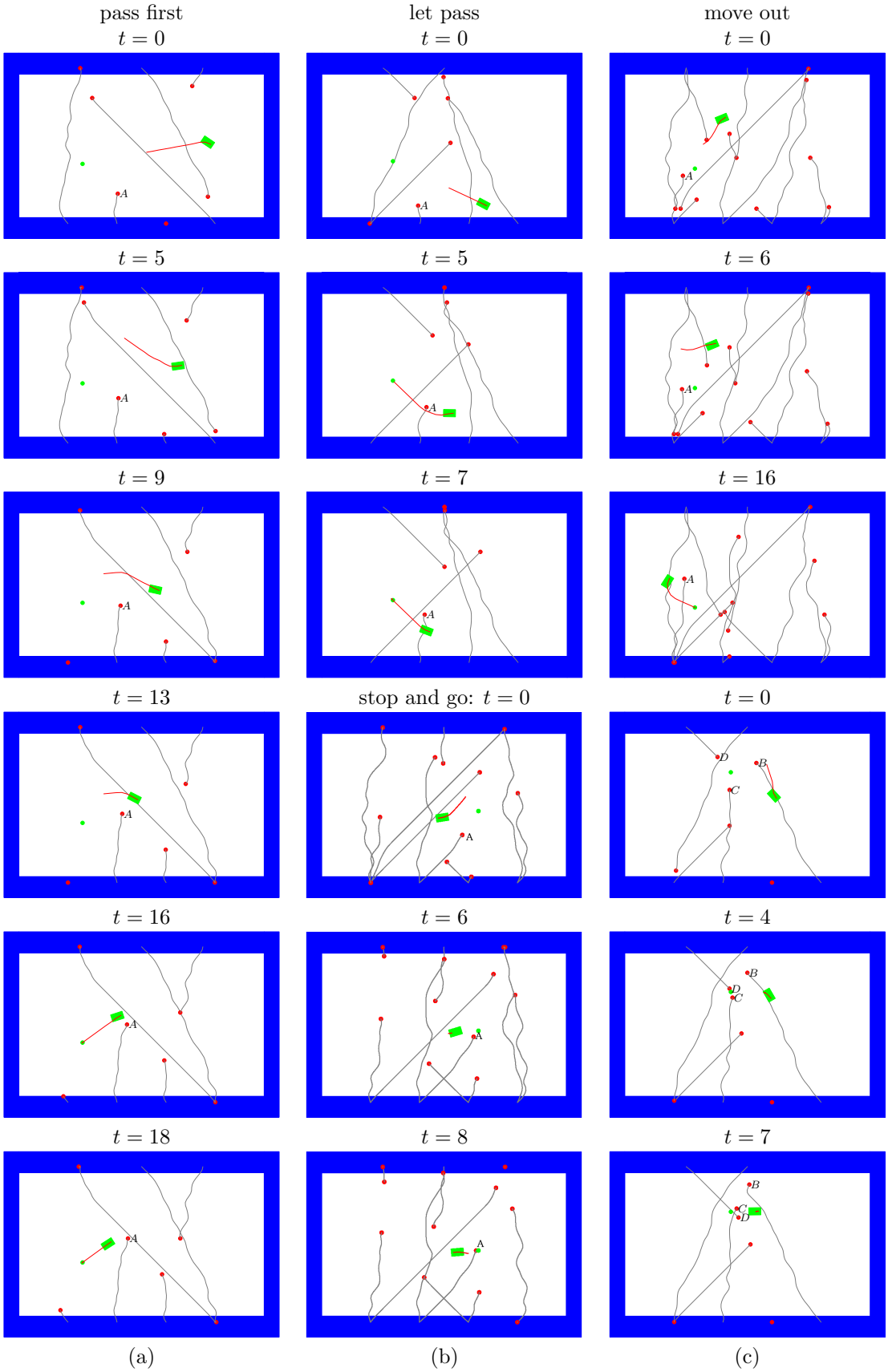


Figure 4.7: Navigation among multiple moving obstacles: (a) the robot passes before an obstacle; (b) the robot stops to let an obstacle pass; (c) the robot moves out of the obstacle path.

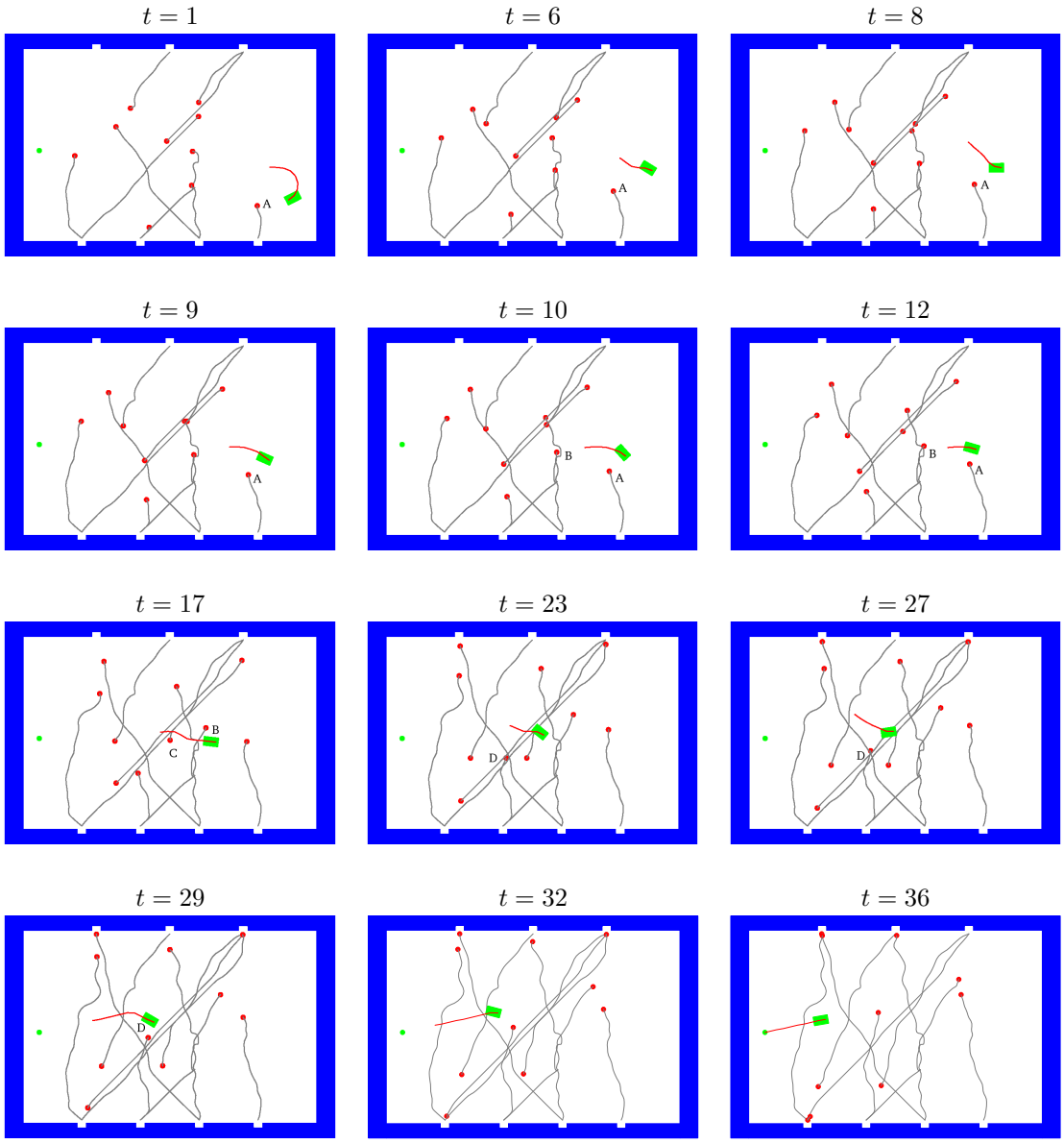


Figure 4.8: Simulation of planification and execution in a closed environment with moving obstacles following typical patterns: example with a maximum depth set to 5s.

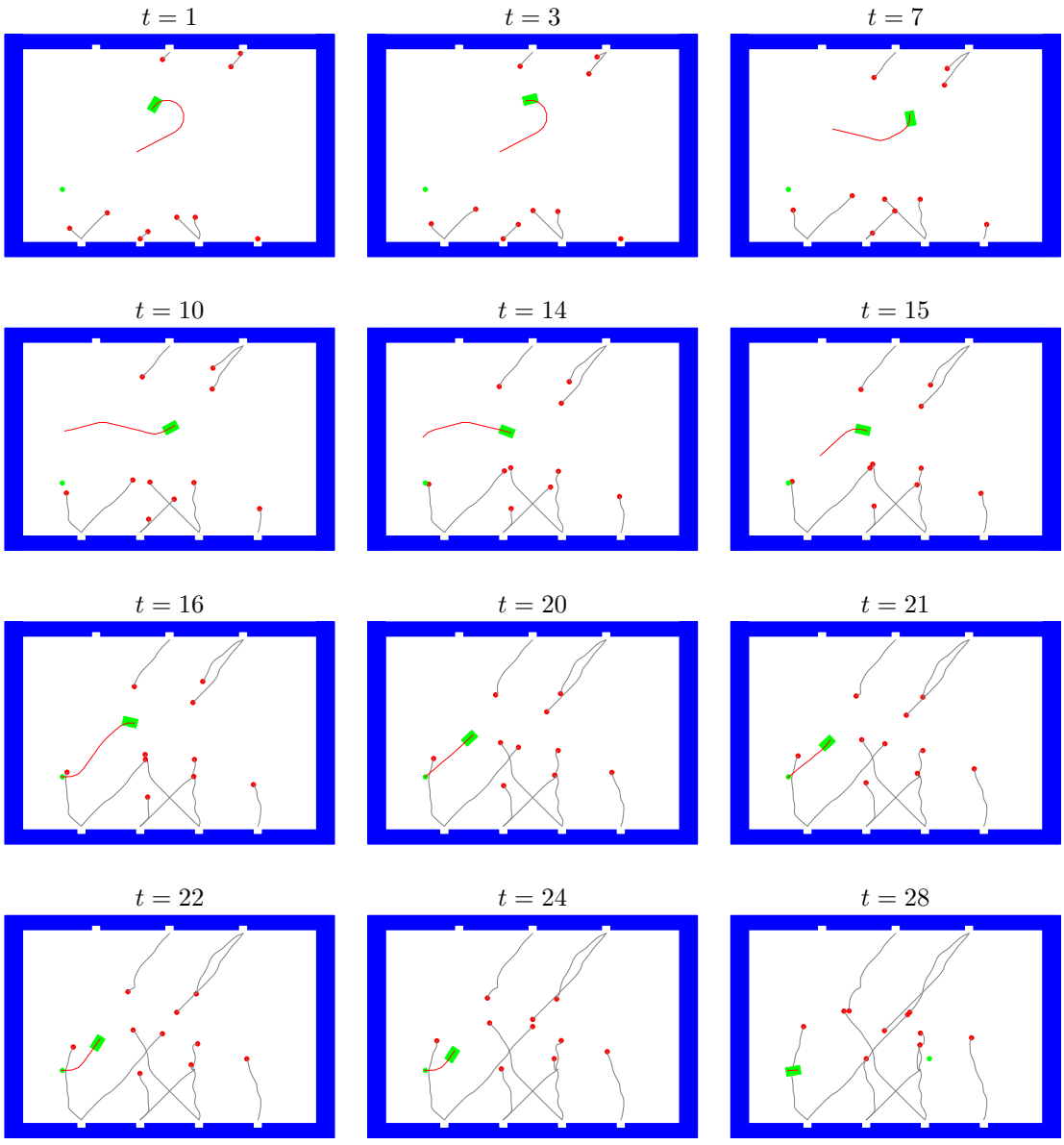


Figure 4.9: Simulation of planification and execution in a closed environment with moving obstacles following typical patterns: example with a maximum depth set to 13s.

number of obstacles in the environment. This is mostly due to the dimensions and the physical capabilities of the robot, which can find itself trapped by multiple moving obstacles or between a moving and a static obstacle with the impossibility to fastly move backward or accelerate or steer as much as necessary to avoid collision. In this case the safest maneuver is to stop as soon as possible. It is possible to notice that the obstacles simply follow their trajectory ignoring the presence of other obstacles and of the robot. In a real environment one could argue that the detected collisions would be avoided by the obstacles and that in general a collision means that the robot "bothered" the obstacle along its preferred trajectory. Also, a collision can be caused by the position of a goal near a door and the sudden appearence of a pedestrian. Tests done giving to the robot littler dimension or augmenting the maximum steering angle or just allowing a bigger ending distance from the goal show a sensible reduction in the number of collisions. The fourth column of the table indicates the mean time needed for the robot to accomplish the navigation task; last column indicates the percentage of this time relative to the case whhere there are no moving obstacles in the environment.

# pedestrians	# collisions	# collisions with $v_r \neq 0$	total time (s)	% time
0	0	0	887	1
4	0.2	0	1184.5	1.33
6	2	0	1219.5	1.37
8	3.2	0	1420	1.60
10	6	0	1550	1.75
12	7	0	1628.5	1.84

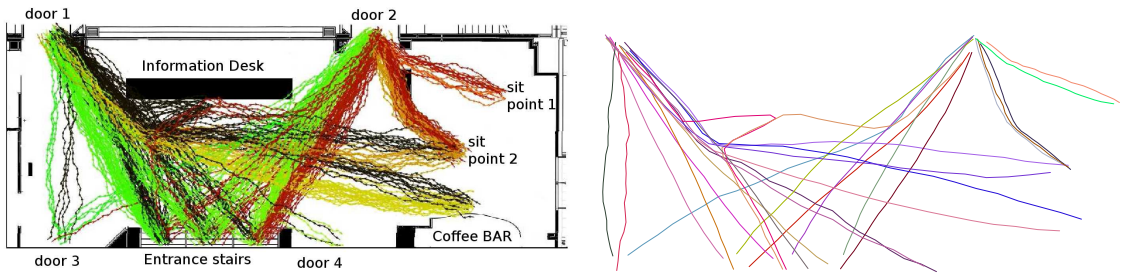
Table 4.1: Results for 100 goals reached and different numbers of pedestrians. Average results with HMM prediction, obtained by repeating the experiment 10 times.

4.4.2 GP Results

A dataset of real pedestrian trajectories recorded in the entrance hall at INRIA Rhône-Alpes have been processed and a set of GP have been learned, see Fig 4.10. This set is supposed to be known by the robot. An environment similar to the entrance hall has been simulated. The recorded trajectories have been linearly interpolated and simulated pedestrians move along the recorded data. It is assumed perfect perception, localization and execution. The robot moves around the environment, looking for goals that are randomly and successively drawn.

Prediction

Figure 4.11 shows the prediction obtained for one of the trajectories of the dataset. Column (a) shows the object observations (red points) and the mean of the typical paths considered (colored lines); column (b) shows the prediction at future time steps, given by ellipses with axis 3 times the



Dataset Learned GPs' means

Figure 4.10: (a) Trajectory dataset in the hall at INRIA Rhône Alpes and (b) learned GPs

standard deviation; column (c) shows the estimated likelihood of each of the GPs considered. Figure 4.12 makes the comparison between the covariance of the prediction issued from a target tracking method and the covariance issued form the GP representation. While the covariance of the target tracking grows to infinity, each of the modes of the GP prediction gets stable after a while; if after some observations, the distribution can be considered unimodal, the covariance of the prediction is limited and becomes inferior to the one given by Target tracking in medium term.

Planning

We tested the probabilistic planning with GP in conditions similar to that applied for the HMM example. We simulated the robot navigating among circular obstacles with trajectories that are chosen randomly from the trajectory dataset. The static environment is supposed to be free and the perception of the robot is simulated. The timestep chosen is of $0.5s$. Planning and execution run in parallel. Figure 4.13 shows some snapshot from the obtained results. The robot is the red rectangle and perceives the circular obstacle (red full point). The goal of the robot is at the bottom of the image (black circle). Green paths represent the mean of the Gaussian Processes: the likelihood of each GP is estimated at each time step on the basis of the previous observations; lighter colour is for lower likelihood. The tree explored by the robot is given by the blue lines. Again, lighter blue means lower likelihood. Circles represent the prediction for the obstacle for each associated Gaussian process and at successive timesteps. Fig. 4.13(a) shows the planning at the first timesteps, where the few observation on the obstacle give multiple likely patterns; in figure 4.13(b) the robot moves toward the goal, while the obstacle moves toward the upper left corner and the prediction gets better around one GP only. In figure 4.13(c) the moving obstacle has moved behind the robot, while another one appears near the goal: this is the first observation of the new obstacle and the path is still not adapted to this new entry. Fig. 4.13(d), shows how the search tree is grown and the path is adapted to the new situation. Similarly to table 4.1, table 4.2 summarizs the results obtained with the Gaussian Processes representation and prediction.

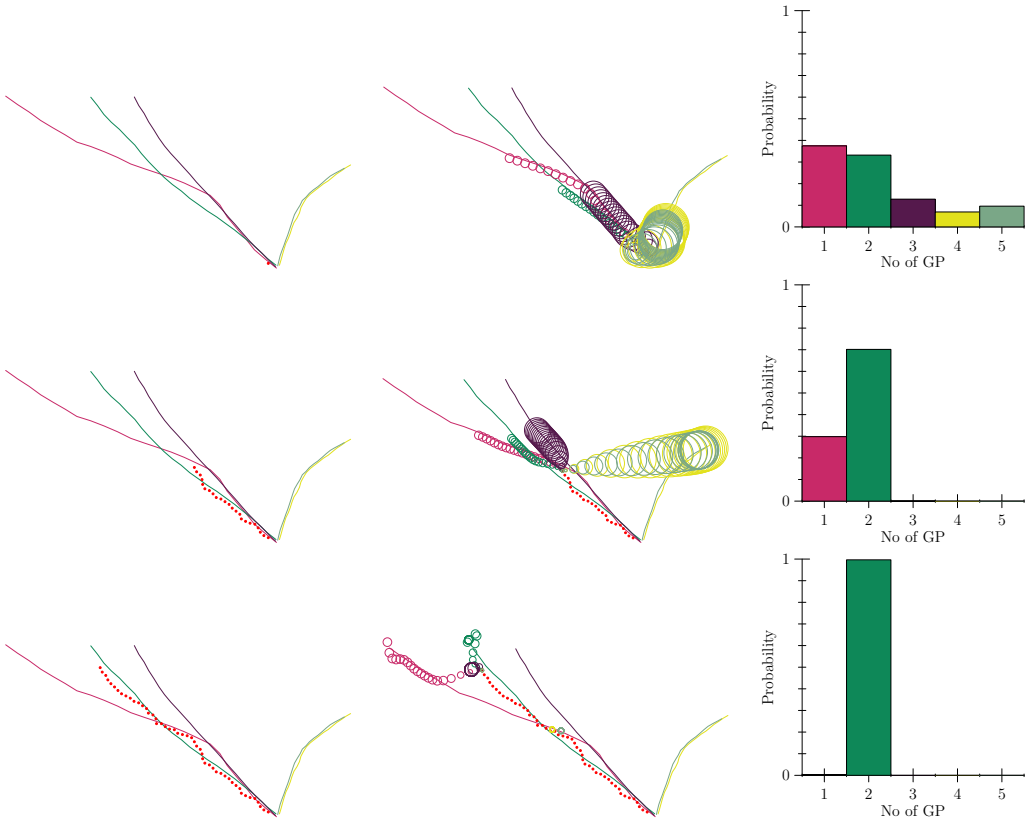


Figure 4.11: Gaussian Processes representation: the colored lines in (a-b) are the means of the typical patterns considered; given the observations (red points in (a-b)), the prediction for future timesteps is obtained (b). The probability of each pattern is estimated (c). Only GPs with probability > 0.01 are shown at each time.

# pedestrians	# collisions	# collisions with robot vel.$\neq 0$	total time (s)	% time
0	0	0	663.5	1
4	1.5	0	754	1.14
6	2.3	0	951.5	1.43
8	3.8	0	1094.8	1.65
10	6.3	0	1187.6	1.79
12	7.4	0	1240.7	1.87

Table 4.2: Results for 100 goals reached and different number of pedestrians. Average results with GP prediction, obtained repeating the experiment 10 times.

4.5 Conclusions

In this chapter we applied the navigation method developed in the previous chapter to the information given by two typical patterns representation: the Hidden Markov Models and the Gaussian

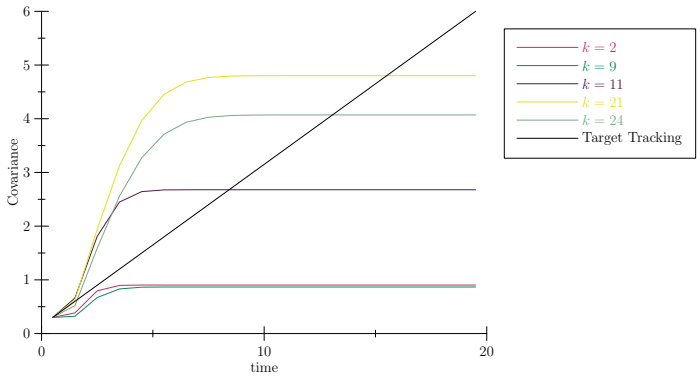


Figure 4.12: Comparison of the covariance of the prediction obtained with a target tracking approach versus GP representation of the typical patterns.

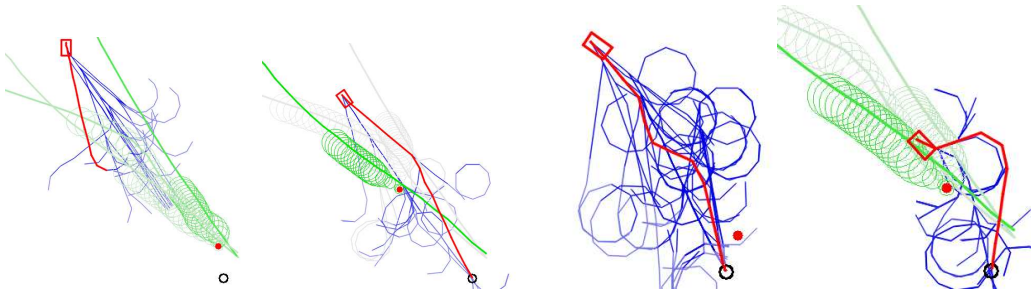


Figure 4.13: The robot moves in a simulated environment with a moving obstacle. The prediction of the obstacle is given by a Gaussian mixture based on the pre-learned Gaussian processes (green). The exploration tree maintains an estimation of the likelihood of the path that is adapted to the incoming observation.

Processes representation. As a conclusion, we first compare the HMM and the GP representations from the navigation point of view based on the probabilistic partial planning; secondly we underline the advantages and drawbacks of the typical pattern prediction method with respect to the target tracking methods. Finally we discuss the achieved results and some possible improvement ot the algorithm.

4.5.1 Comparison between HMM and GPs representation

The results obtained with HMM and GP are comparable in terms of time to reach the goal and collision avoidance. From the algorithmic point of view, each of the two methods present peculiar properties that make one or the other advantageous in different environments. The learning and prediction of moving obstacles trajectories with HMM is used in many applications in scientific and

commercial projects: the reliability of such a method is somewhat well extablished. On the other side, GPs are quite a new field in trajectory clustering andrepresentation, so the effectiveness of this method is yet to be proven.
For what concerns HMMs, our implementation of the method does not take properly into account the velocity of the obstacles. Since the velocity is part of the state, a graph is required for each different velocity: the velocity is discretized in the model. A second important point is that the prediction is discretized in time and the discretization step depends on the observation rate in the learning phase. This can cause some problems in the navigation phase: the timestep for prediction should be ideally a multiple of the observation timestep and the velocity of the tracked obstacle has to be approximated according to the discretization. Also, the choice of prediction timestep is conditionned by the observation in the learning phase and the prediction is less accurate in time.
In the GP framework instead, the trajectory models are continuous: the velocity is estimated during navigation and its value is directly used for prediction. Secondly, the prediction is also continuous: this allows to better decouple the navigation algorithm and the learning and representation of the typical trajectories; there is no more need to use the same or a multiple timestep and the on-line observation does not need to be synchronized with learned discrete observation points. Finally, in the GPs framework, the prediction is represented by a mixture of Gaussians instead of a discretized function. In relation with our algorithm, this allows to update the probability of collisions stored in the tree in a more efficient way than in the case of the HMM representation: in the HMM case (see §4.3.1), to the probability of collision needs to be recomputed from scratch. In the GP case (see §4.3.2) the probability of collision in updated only by changing the weight of each pedestrian mode, discarding the difference in mean and covariance.

4.5.2 Typical Pattern based models: advantages and drawbacks

The advantages of predicting the obstacles position on the basis of the typical patterns are:

- more reliability in the medium and long term;
- finite dispersion of the uncertainty in the space in the future prediction;
- a more precise idea of where new obstacles may come from.

From the point of view of navigation, this means:

- longer planned paths;
- reduced necessity of replanning;
- better global performance (shorter global path, shorter time to reach the goal).

The drawbacks of such an approach are mostly due to the fact that the prediction is based on a learning phase, which must be carried out by an off-board dedicated architecture and that the obtained information is strictly correlated with the considered environment. Also, we can notice that in all presented approaches the learning phase is presented as a **batch** process which is carried out before any prediction is possible using observation from fixed, **off-board** sensors. These are two important and heavy issues on the system: the need of a sensor plateform to properly observe the environment and the need of a period of time when the robot is not operational and only learning is performed.

For the first problem, we can imagine that the observations may be directly carried out by the on-board sensors of the robot. However, this strictly depends on the robot equipment, the environment structure, the robot position: if the robot is moving, the uncertainty about its position also affects the sensor readings, and in general only partial trajectories would be observable. Furthermore, the robot should navigate without the proper prediction information, meaning that the kinematic and dynamic models will not be completely substituted by the typical pattern based ones. As far as we know, at this time there is no work which presents results of typical motion patterns learning with on-board sensors only. Another solution would be to apply typical learned behaviours or patterns to similar environments in various locations: we can expect, for example that a pedestrian moves in the same way in different towns: walking along the sidewalks and crossing the road at specific points such as crosswalks; for what concerns cars, they are expected to move along one lane and to slow down, stop or turn with some probability when in proximity of specific areas.

For the second problem, in [63] a "learn and predict" approach is presented: the learning of the typical patterns is carried out in an incremental way, so that it is possible to have some prediction since the first observation. On one hand, this reduces the time needed by a learning-only phase and allows the set of typical patterns to adapt in time to the potential changes of the environment. On the other hand, the reliability of predictions at an early phase is not guaranteed, so that a navigation strategy in such an environment should take into consideration both the typical patterns prediction along with its reliability and the motion model based on kinematics.

It is finally possible that some obstacle does not follow the previously learned patterns. This can be due to an unsufficient learning or an unusual behaviour. From the point of view of the robot, the challenge is the same and if an obstacle for which the Mahalanobis distance is higher than the threshold for all the typical paths, the partial planning should be performed on the basis of the motion model issued by the linear target tracking, as detailed in chapter 3. However, if the new observed pattern is actually systematically adopted, it is opportune to perform another learning phase and to add the new pattern to the typical ones in order to maintain the advantages of this

navigation method.

4.5.3 Improving issues

Our algorithm has proven to be an effective tool for navigation among moving obstacles which follow typical patterns. To test the algorithm in more realistic scenarios however, it is necessary to fuse the information coming from target tracking with the one given by the typical pattern: only in this way the algorithm would be more robust to obstacles with an unusual or not learned behaviour. This issue will be addressed in Chapter 6. For the algorithm to be tested in road scenarios, where vehicles present typical patterns, it would be necessary to model the road rules direrctly into the planner, so that in each situation only allowed maneuvers are taken into consideration.

Chapter 5

Implementation and experimental setup

5.1 The Cycab

The Cycab (Fig. 5.1) is an electric robotic plateform. It was initially developed by INRIA and then commercialised by Robosoft. The Cycab is a car-like mobile platform which can be equipped with multiple sensors, for proprioceptive and exteroceptive measurements. It can be driven manually using a joystick or automatically. The intelligence system is constituted by two microcontrollers whose task is to control the velocity and steering angle of each wheel and an on-board computer which give commands to the microcontrollers and connects the various sensors. For our simulation experiments and during data acquisition we considered the following sensor set up:

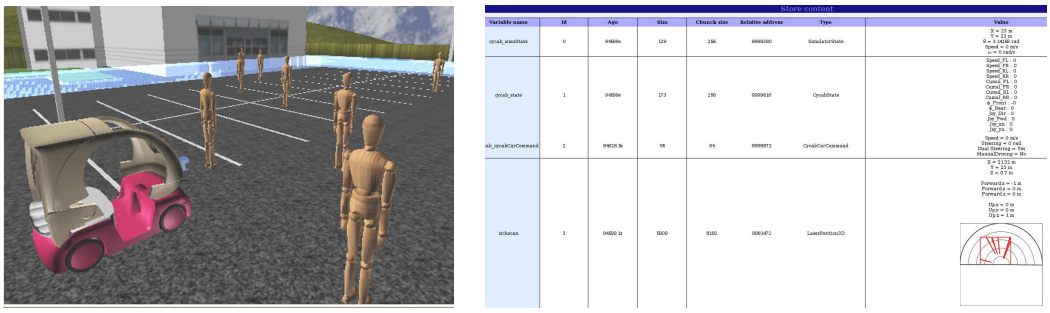
– **a laser-range finder Sick**: situated in front of the Cycab, at 50cm from the ground level, it observes an half circle in front of the Cycab on a plane parallel to the ground and up to a maximum distance of 80m.

Figure 5.1: The Cycab plateform in front of INRIA Rhône-Alpes

– 4 **encoders**: which measure the rotation angle of 4 measure wheels free of mass attached to each of the Cycab wheels respectively.

– **a GPS**: which communicates with a constellation of satellites to give the absolute position of the Cycab.

5.2 Cycab Simulator: CycabTK, and Hugr



(a) (b)

Figure 5.2: (a) CycabTK simulator (b) Hugrstore screenshot.

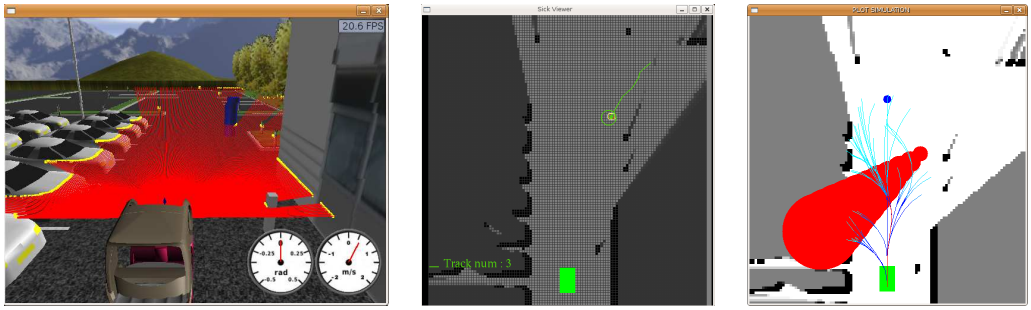
A Cycab simulator has been developed by e-motion team [84]. This simulator is intended to simulate hardware and low-level drivers, in order to produce a temporal behaviour (refresh frequency, scheduling...) similar to what can be found on the Cycab. Furthermore, a hierarchy of C++ classes has been developed in order to keep a consistent interface between the simulated Cycab and the real one. Sensors and environment are also simulated, so that complete applications can be developed on this test bed.

The simulator is based on MGengine, an opensource simulation engine which is intended to be an highly dynamic and evolutive 3D engine, object-oriented and realtime [85]. The Cycab ToolKit (CycabTK) offers various robots and sensor drivers, in particular a Cycab driver, a Sick sensor driver, a camera driver, a stereo camera driver, and a GPS driver. Data provided by each sensor and the variables of the robot are simulated and shared on a memory segment, the *Hugrstore*. Hugr is a middleware library providing tools for applications to share information by simply reading and writing data-objects. It is based on shared memory map called store where data are to be written or read by applications. The Hugr framework enables the 3D simulator, the control application, the sensors drivers etc... to run concurrently regardless of how those programs are designed: It also provides an interface to the real robot: the sensor data can be read and the controls data can be written or sent to the real Cycab using the same commands and protocol of the simulator. Finally it is possible to record and replay data.

Hugr is also being integrated on the other robotic platforms at INRIA-Rhône-Alpes, as the automatic wheel-chair. In the simulator different objects (static and dynamic) and robots can be represented. In Chapter 2, we used an holonomic robot model while, in Chapter 3 and 4, the Cycab model has been used.

5.3 Implementation of the navigation algorithm: complete navigation architecture.

The mapping and target tracking algorithm, along with the planning algorithm and a perfect execution model have been tested in the simulator letting the pedestrians move along trajectories generated in a similar way of what explained in §4.4.1. To run the whole architecture on the real robot it is still necessary to develop a trajectory tracking algorithm and to test if the computational resources now available on the Cycab are sufficient to let all the algorithms (mapping and tracking, planning, execution, user feedback) run in parallel at an acceptable frequency in dynamic environment. If this would not be possible, a network version of Hugr is under development so that different algorithms could run on dedicated hardware while sharing information. In Figure 5.4 the



a b c

Figure 5.3: Screenshot from simulation results: (a) The Cycab and an obstacle moving in the simulated environment in Cycabtk. (b) The static map and tracked objects issued from the mapping algorithm. (c) The planning algorithm output.

navigation architecture implemented is shown. Using a shared memory system, the various processes can run in parallel at different frequencies. The store gives at each time the last available value of each variable. Each module works separately and asynchronously, sharing the needed information through the store. The variables published on the store are the followings:

Cycab: The robot publishes on the store the data coming from each of its sensors:

- **Odometry**: The Cycab is equipped with an optical encoder in each of its four wheels. Odometric data are elaborated from the lmicrocontrollers of the Cycab and consist in:

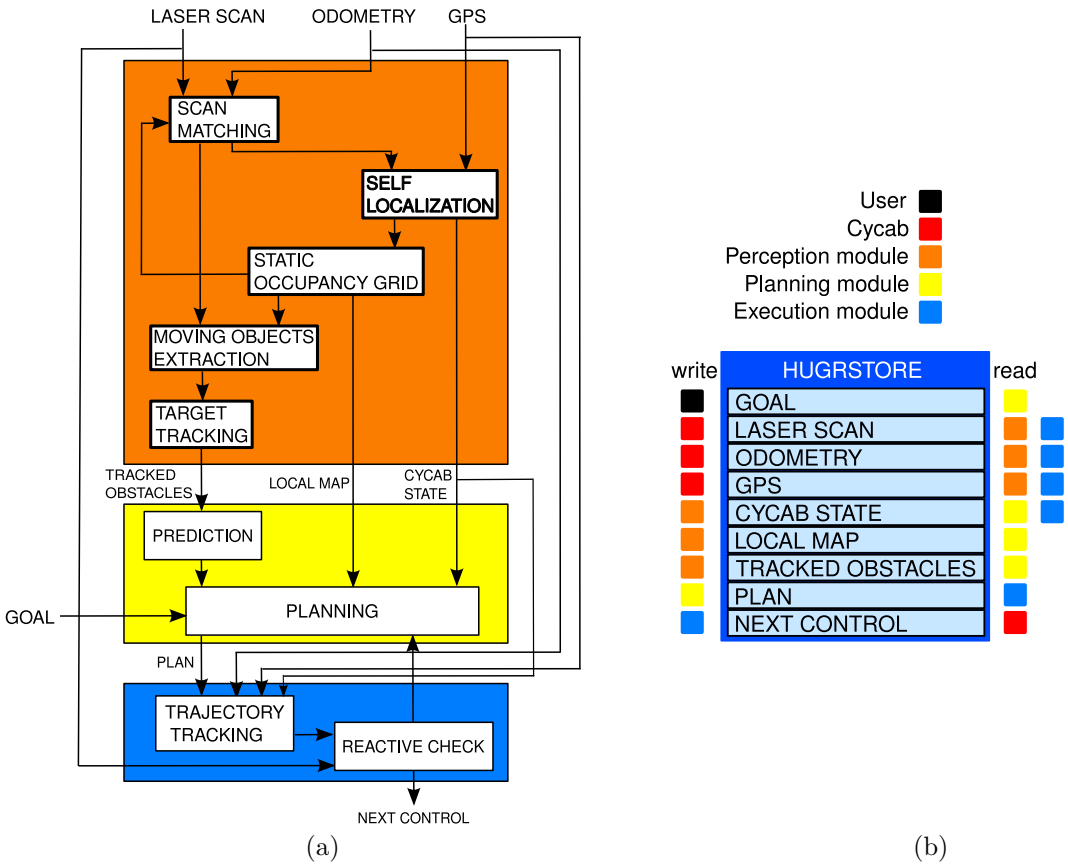


Figure 5.4: (a) Navigation architecture: at the top the sensor information coming from the robot; in orange the perception module; the output if the complete architecture are the controls for the robot motion (v, ω); (b) Access of the various processes to the shared memory segment.

- $(d_{Rr}, d_{Rf}, d_{Lr}, d_{Lf})$ distances covered by the 4 wheels:
  R=right, L=left, r= rear, f = front;
- $(v_{Rr}, v_{Rf}, v_{Lr}, v_{Lf})$ linear velocity of each wheel;
- (ϕ_f, ϕ_r) steering angles of the front and rear axes respectively;
- t acquisition time, referred to beginning of navigation.

Data are written in the shared memory by the Cycab, at a frequency of about $100Hz$. Odometric data are used by the perception module, and by the execution module to estimate the state of the Cycab.

– **GPS**: The Cycab is equipped with a GPS: the collected data are triangulated with another fixed based GPS, so to have centimetric precision over the position. GPS data consist in:

- $(north, east, height)$ absolute coordinates of the Cycab;

- (v, θ) estimated velocity and orientation;
- (h, m, s) hour, minute and second of data acquisition, referred to day time.

Data are published in hugrstore at a frequency of about $10Hz$. GPS data are used by the perception module, and by the execution module to estimate the state of the Cycab.

– **Laser Scan**: A laser-range finder Sick is mounted in front of the Cycab, at 50cm height from the ground level. It observes an half circle in front of the Cycab on a plane parallel to the ground. Sick data consist in:

– configuration:
- N number of beams;
- $(\alpha_{min}, \alpha_{max}, \alpha_{step})$ minimal and maximal angle of field of view, angular resolution;
- $(d_{min}, d_{max}, d_{step})$ minimal and maximal measurable distance, range resolution;
- (I, i_{min}, i_{max}) tells if sensor returns intensities, the minimal and maximum intensity values;

– data:
- $(d_1, ..., d_N)$ measured distances;
- $(i_1, ..., i_N)$ received intensities.

Data are passed to the store at a frequency of about $37Hz$. Laser data are used by the perception module to build the static occupancy grid and to refine the position and orientation of the robot; they are also used by the execution module to check if there is some obstacle too near the robot and if an emergency braking maneuver if needed.

Perception module (orange): is dedicated to elaborate the information coming form the sensors and gives in output a local occupancy grid, a list of moving obstacles and an estimation of the state of the robot (with respect to absolute coordinates, if GPS is available or in coordinates relative to the initial position of the robot).

– **Cycab State**:
– (x, y, θ) estimated position and orientation of the center of the rear wheel axis;
– (v, ϕ) linear velocity and steering angle of the front wheels;

– **Local Map**:
– (x_0, y_0, θ_0) position of cell $i = 0, j = 0$ in the Cycab reference frame;
– *step* discretization step;
– $\mathcal{G} = \{p_{ij}\}_{i=1,..,I;j=1,...,J}$ probabilities of occupancy of each cell in the grid;

– **Tracked Obstacles**: List of objects, each described by:

- id identifier of the object;
- $(x, y, dim_{XX}, dim_{YY}, dim_{XY})$ position and dimension (approximated by an ellipse, described by a symmetric matrix;
- (v_x, v_y) linear velocity;
- (Q_{XX}, Q_{YY}, Q_{XY}) uncertainty in constant velocity motion model, described by a symmetric matrix;

The algorithm used by this module has been introduced in Chapter 3 and detailed in section §5.4. This module works at a frequency of about 8 to 10 Hz, which can slightly vary in dependence of the size of the grid sent to the shared memory and the number of moving obstacles tracked.
The outputs of the perception module are written in Hugrstore and read by the planning module. The estimation of the state is also used by the execution module, respectively to compute the best control for trajectory tracking.

Planning module (yellow): takes in input the static map, the tracked obstacles, the estimated position of the robot and the goal defined by the user. After performing prediction on the basis of the motion models issued from tracking, the search tree is updated and grown and a partial trajectory is elaborated and passed in output. The partial trajectory consists of:

- **Plan**: $\pi = \{(q_n, t_n)\}_{n=1,...N}$, where
  - $q_n = (x, y, \theta, v, \phi)$ are the desired Cycab states;
  - t_n are the correspondent timestamps, referred to the navigation time.

The algorithms described in Chapter 3 and Chapter 4 have been implemented in C++. The partial planning algorithm have been tested with the real dataset acquired driving the Cycab manually (see Fig. 5.6(a)) among moving pedestrians and results have been presented in §3.5. This module works at a frequency of about $2Hz$. The partial trajectory is read by the execution module, as soon as it is written.

Execution module (blue): Given the estimated position of the robot at a certain instant, the odometric data, the GPS data, and the desired trajectory of the Cycab, this modules gives in output the best control of the robot. This is aimed at maintaining the real trajectory of the robot as near as possible to the desired one. The control to be passed next is checked reactively (REACTIVE CHECK) with the current laser scan; if there is an obstacle too near to the robot the robot is supposed to be in danger and the control is substituted by an appropriate braking maneuver. A control for the robot is constituted by:

- **Next Control**: (v, ϕ) linear velocity and steering angle of the front wheels.

The steering angle of the rear wheels is always set to zero. The Cycab microcontrollers translate the control into power values for each motor. This module works at a higher frequency with respect to the planning module and lower frequency with respect to the odometry writing rate. For the results obtained in the precedent chapters, we assumed perfect execution meaning that the planned controls applied to the Cycab give exactly the desired state. This module has not been yet developed. We will give some idea for future work on how to address the problem of execution error and trajectory tracking in Section 6.2.1.

User feedback The user gives the position of next goal and is shown the path the Cycab is going to execute. The user can stop the execution with a brake maneuver and ask for a new path. In static environment, the user is asked to confirm the path before execution.

- **Goal**: $(north, east)$ Goal position, expressed in geodetic coordinates when GPS is available or relative coordinated otherwise.

5.3.1 Time evolution of the algorithm

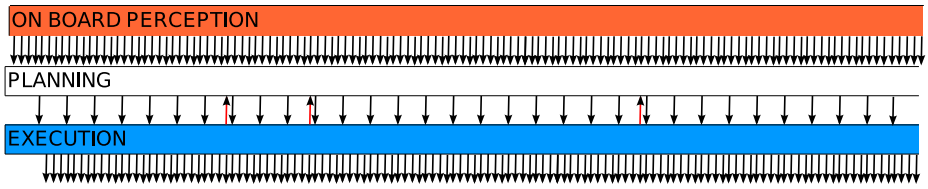


Figure 5.5: Scheme of the time evolution of the navigation algorithm.

Figure 5.5 shows the time evolution of the navigation algorithm in time. Arrows show the outputs of each module: each module works at a different frequency. In particular:

- the perception module works at its own, high frequency;
- the execution module frequency is higher than the planning frequency: the next control is corrected at higher frequency using the odometric and localization data;
- the planning can be re-synchronized due to an unexpected event detected by the reactive algorithm in the execution module (red arrows pointing upwards).

5.4 Mapping and multi target tracking algorithm

5.4.1 Self Localization

On the basis of the GPS and odometric data a first estimation of the state of the Cycab is computed. The initial position and orientation are the absolute reference frame for the whole navigation algorithm: given the GPS data at the beginning of navigation, each GPS point correspond to a position in the frame referred to the position and orientation of the Cycab at the beginning of navigation. On the basis of the acquisition time of the different kinds of data, the new position is estimated. The linear displacement increment and the orientation increment are computed from the odometric data:

$$\Delta_s = \frac{(d'_{Rr} + d'_{Lr} + d'_{Rf} + d'_{Lf})}{4} \tag{5.1}$$

$$\Delta_\theta = \frac{d'_{Fr} - d'_{Fl}}{L} \tag{5.2}$$

where d' is the difference between the current odometric data and the data acquired at the same time of the last GPS acquisition; L is the length of the rear wheels axis. The new position estimation is given by the last GPS acquired position, incremented by Δ_s and Δ_θ.

5.4.2 Scan Matching

The position estimated by the GPS and odometric data is refined matching the laser data with the estimated grid. Considering a certain number of previous estimated poses (empirically set to 7) and corresponding acquired laser scans, the hitting points of each laser beam are computed. The pose estimation issued from the odometric and GPS data is used as a guess for the new position. Alignment is iteratively performed between the previous scans and the current scan using Iterative Closest Point algorithm [58]. The pose obtained after alignment is passed as output.

5.4.3 Static Occupancy Grid

A global static occupancy grid is built. The grid is initially fixed at the absolute reference frame. The discretization step used is $0.2m$. Initially the probability of occupation of each cell is setted a 0.5. After the acquisition of a laser scan and the estimation of the robot position, the grid is updated. Considering a single laser beam, all the cells traversed along the direction of the beam are considered and their occupation is updated using the Bayes' rule and the sensor properties. A sensor model is designed:

- $P_{obs} = P_{md}$: the probability of miss detection P_{md} is associated with the observation of each cell traversed by the beam without reflection;

- $P_{obs} = 1 - P_{fa}$: the complement of the probability of false alarm P_{fa} is associated with a cell in which the beam has been reflected;
- $P_{obs} = 0.5$ is associated to each cell beyond the reflected beam.

The estimation of the probability of occupations P are recorded in terms of the logarithm of odds:

$$logOdd(P) = log\left(\frac{P}{1-P}\right) \tag{5.3}$$

$$P = 1 - \frac{1}{1 + \exp(logOdd(P))} \tag{5.4}$$

Given the log odd of a cell computed at the previous step $logOdd_{-1}$, and the log odd from observation $logOdd_{obs}$, the updated probability is computed as follows:

$$logOdd = logOdd_{-1} + logOdd_{obs} - logOdd(0.5) \tag{5.5}$$

The local grid is extracted from the absolute grid: the position of the center point of each cell in the local grid is considered and the correspondent occupation in the absolute grid is retrieved. The used local grid is $15m$ large from the center of the robot in the left and right direction, $3m$ long from the center to the back of the robot direction and $35m$ long from the center to the front of the robot direction.

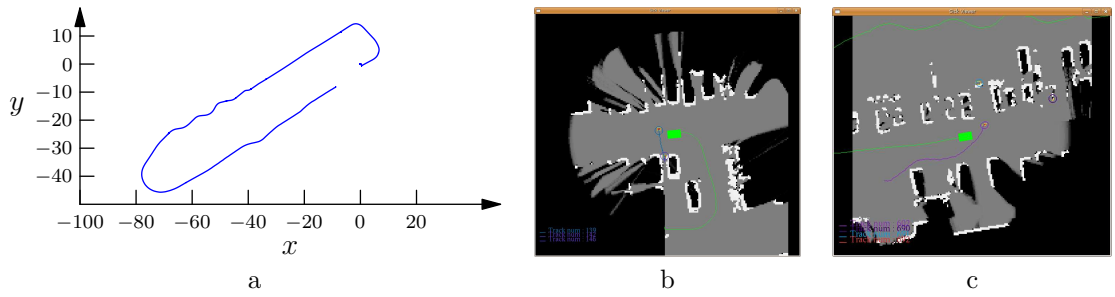


Figure 5.6: (a) Cycab trajectory obtained by GPS point measures; (b-c) Static obstacles and tracked pedestrians obtained along the trajectory and the global map.

5.4.4 Detection of moving obstacles

Given the current laser scan and the estimated position, the positions where laser beams have been reflected are computed. These positions are compared with the occupancy grid: if the correspondent cell has a probability of occupation lower than P_{md} the point is considered as a possible moving obstacle. For a moving obstacle to be identified, a minimum number of detected points is fixed (3 in our implementation). Detected points are said to belong to the same moving object if their relative

distance is less than a predefined threshold ($0.3m$ in our implementation). An object observation is computed for each cluster and is represented by the mean position of the cluster and the maximum dimension, expressed by a 2×2 matrix: objects observations are approximated by ellipses represented by the matrix.

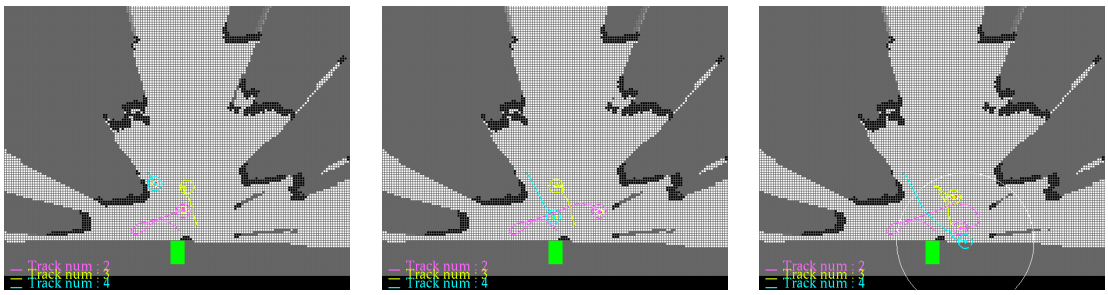
5.4.5 Target Tracking

The list of moving object observations is passed to the target tracking algorithm. The tracker maintains a list of objects and their tracks described each by a Kalman filter. Each existent track can be associated to at most one observation. First, the one step ahead prediction of each object is performed and the Mahalanobis distance between the prediction and the observations is gated in order to reduce the number of possible assignments. Then, a Global Nearest Neighbour data association method is used [59]: the possible global association hypotheses are weighted considering a combination of the distances between the observation and the predictions. In our implementation we used a simplified version of the Multi Hypothesis Tracking algorithm, and we chose to maintain only the hypothesis with the highest weight instead of a set of hypotheses. Several cases could appear:

1. An observation is associated with a track: the obstacles state is updated using the Kalman filter;

2. A track has no observation associated: the reestimated state of the obstacle is given by the prediction; a counter is initialized so that tracks that are unobserved for long time are forgotten;

3. An observation is not associated to any track: a hypothesis of new track is initialized; in order to discard some false alarms, a counter is initialized and only objects that are observed during 3 consecutive instants are retained.



a b c

Figure 5.7: Mapping and target tracking example with three moving obstacles in front of the robot;(c) track number 4 disappears behind the robot.

It is also possible to run the target tracking algorithm on-line with the simulator. For this purpose, pedestrians have been modelled by cylinders of radius $0.25m$, that are easily trackable and give measures near to the realistic ones (see Fig. 5.3).

Chapter 6

Perspectives and Conclusions

6.1 Contributions

In this thesis we focussed on the problem of navigation in unknown dynamic environment. In particular we stated the importance for the robot to have knowledge of the limits of the information it has about the surrounding world and we addressed the issue to integrate into the decision process a probabilistic representation of the uncertainty due to the limits of perception of the environment and to the prediction of the moving obstacles trajectories.
As discussed in Chapter 2 and 3 most of the developed approaches in literature and real world applications are based upon a deterministic or worst case approach, making the assumption that the information of the robot is good enough to allow navigation. On the other side, some probabilistic approaches exist but, due to the complexity of the probabilistic representation, are limited to the simplest and lowest dimensional problems. Also, most developed probabilistic approaches are complete methods that do not take explicitly into account the real-time constraints of navigation in dynamic environment or the evolutive nature of the gathered information during environment exploration. To address these issues, we developed first a reactive and then a partial planning techniques aimed at the computation of the risk or probability of collision in order to choose a good trajectory for the robot in terms of navigation safety and efficiency in the goal reaching task. We took into account a probabilistic representation of the static world, given by an occupancy grid, and different kinds of probabilistic prediction: from the one-step ahead prediction, used in Chapter 2, to the short term Gaussian prediction (Chapter 3) and to the medium term predictions represented by HMMs or GPs (Chapter 4). We considered that the environment representation and predictions naturally evolve in time due both to the continuous observations of the robot, that refines its estimation, and to the changing of behaviour of the obstacles (obstacles that enter the environment, obstacles that change velocity or trajectory). We used the observations gathered from the sensors to compute the risk of collision and update it according to the evolving information so that the choice of the path

is adapted to the environmental situation at anytime.
The contributions of this thesis work are:

1. **A reactive navigation algorithm based on a dynamic occupancy grid representation of the environment and a probabilistic computation of the risk of collision.**
   In Chapter 2 we showed with simulation experiments that our algorithm is able to compute the probability of collision in time taking into consideration the quality of the sensor information and the environmental changes. We showed with simple problems that the algorithm can compute in real-time safe and goal directed controls. We compared the results obtained with the simple grid and applying a clustering algorithm on it. We demonstrated that the probabilistic representation of the perception uncertainty is important for the safety and efficiency in the navigation task and that the robot adapts its behaviour to the accuracy of the world model.

2. **A probabilistic extension of the Rapidly-exploring Random tree algorithm with on-line updating.**
   In Chapter 3 we moved toward a partial planning strategy following the idea that a more looking forward strategy could give more efficient and safer performance taking advantage from the information estimated by a tracking algorithm. After reviewing the state of the art for motion planning in dynamic environment and under uncertainty, we took in consideration partial planning and sampling based methods which could satisfy anytime constraints and incremental knowledge of the environment, in contrast with complete methods. We developed a probabilistic extension of the Rapidly-exploring Random tree algorithm: the configuration time space is explored; a tree is grown randomly, but each node is weighted with respect to the probability of collision and the length of the correspondent path: the search is then biased toward the safer zones in the environment that (probably) lead to the goal. We then describe how to update the tree during navigation so that the choice of the path is coherent with the most recent estimation of the state of the world. We used a real tracking dataset with multiple pedestrians and a car-like robot to show the results obtained with our algorithm.

3. **The integration of the probabilistic partial planning with medium term prediction represented either by Hidden Markov Models or Gaussian Processes.**
   In Chapter 4 we introduced the hypothesis that obstacles move following typical motion patterns a priori known by the robot. We presented two methods to represent such typical behaviours, namely the Hidden Markov Model approach and the Gaussian Processes approach. These two probabilistic frameworks present many differences between each other and represent well the state of the art in typical pattern learning, representation and prediction. We adapted the navigation algorithm developed to the two representations, in order to take advantage from the different characteristics of the prediction issued by the two probabilistic

framewoks. We presented and compared simulation results obtained with simulated trajectories and real pedestrian trajectories in indoor environment, showing the robot planning and navigating in real-time, avoiding collision or at worse stopping before colliding with the "blind" moving obstacles.

We think that the introduction of the probabilistic representation of the world present and future state and the computation of the related risk of collision in a partial planning framework represents an important step toward autonomous mobile robot navigating safely in populated environments. We are aware that many improvements and refinements to the algorithm can be done and are necessary to use the developed approach in a realistic scenario. In Section 6.2 we will describe some of such improvements. We think also that the introduction of the probabilistic representation coupled with a partial and anytime algorithm opens interesting perspectives for future work, considering multiple autonomous vehicles operating in the same environment which could achieve coordination even in the case of reduced communication possibilities. We will introduce and briefly discuss this issue in Section 6.3.

6.2 Future Work

6.2.1 Execution error and trajectory tracking

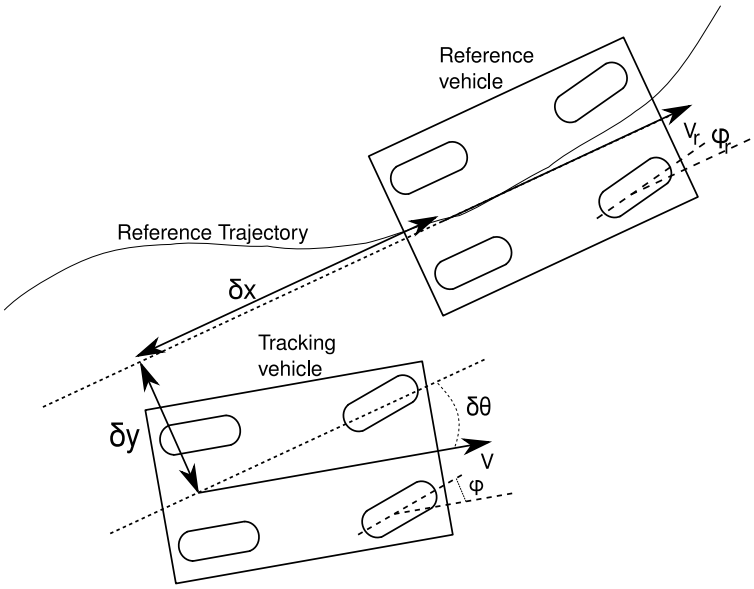


Figure 6.1: Variables involved in the trajectory tracking problem.

To test the algorithm on the real platform and to achieve a robust performance, a trajectory

tracking algorithm is needed. This algorithm has the aim to maintain the trajectory of the robot as near as possible to the nominal trajectory computed by the planner, compensating the delays and errors of the real system with respect to the deterministic execution model used in the planning phase. Also, our partial planner gives in output a list of states and desired controls at discretized instants in time: it is assumed that controls are constant along a timestep and can change instantaneously at the beginning of the next timestep. This of course is not possible and such a trajectory is not admissible in the strict sense: some interpolation is needed to overcome the discontinuities of the reference trajectory.

The robot estimates its own state: this can be done with the relative measures issued by the SLAM algorithm or by the absolute measures given by the centimetric GPS, when this high precision sensor is available. Since both the SLAM algorithm and the GPS work at a lower frequency that the odometry, the estimation can be refined at each instant adding the odometric distances acquired in the meanwhile, and assuming that in such a brief period of time, the error accumulated by the odometric sensors is negligible. The estimated configuration of the robot is then $X(t) = (x, y, \theta)$.

The desired trajectory is defined by the desired state of the robot at a certain instant and a reference control: respectively $X_r(t) = (x_r, y_r, \theta_r)$ and $U_r(t) = (v_r, \phi_r)$. The trajectory tracking error is defined as the difference between the real and desired state (see also Fig 6.1):

$$\delta X = X(t) - X_r(t) = (\delta x, \delta y, \delta\theta) \tag{6.1}$$

The problem of trajectory tracking is to compute the best control $U(t) = (v, \phi)$ for which the system converges toward the reference trajectory, given the error δX and the reference control U_r.

This problem is a classic problem in the automatic controls and robotic community. We present here some characteristics of the problem and some proposed solutions.

It is well known [86] that non-holonomic systems cannot be stabilised about a point by a smooth state feedback control law. Consequently, other types of non-linear feedback controls have been designed: time-varying feedbacks [87] and dynamic feedbacks have been successfully used for tractor trailer and car-like robots. The tracking problem – or stabilization of trajectories – found some earlier constructive results in the Kanayama control rule [88]:

$$\begin{pmatrix} v \\ \omega \end{pmatrix} = \begin{pmatrix} v_r cos(\delta\theta) + K_x \delta x \\ \omega_r + v_r(K_y \delta y + K_\theta sin(\delta\theta)) \end{pmatrix} \tag{6.2}$$

the approach is based on linearization and dynamic feed-back linearization. The stability of the control rule is proven using a Lyapunov function. K_x, K_y and K_θ are parameters whose value can be chosen to design the convergence of the system (fast, non-oscillatory answer). This rule is defined for the control $U = (v, \omega)$ where ω is the angular velocity instead of (v, ϕ). However, since ω can be deduced from ϕ and inversely, the Kanayama rule is adapted to our case. The drawbacks of such

rules are due to the smallness of region of feasibility, due to the linearization of the control law. Other tracking control laws based on local linearization methods have been proposed which achieve local [89] asymptotic stabilization. Global asymptotic stabilization was obtained via state feedback in [87] and via state and output feedback in [90]. In [91] a near-optimal, globally and exponentially stabilizing control is proposed.
In each method, the decision over the control parameters regulates the trade-off between the robustness of stability and fast convergence. It is possible that the proposed controllers will present a lack of flexibility when compared with the real system (large) errors and the trajectory model. It would be then necessary to apply more flexible controller with smaller stability guarantees, as proposed by [92] or a Bayesian controller, as proposed by [93].

6.2.2 Integration of short-term and medium-term prediction

In Chapter 3 and Chapter 4 we proposed respectively to take into consideration a prediction based on the motion model estimated on the basis of target tracking and a medium term prediction based on pre-learned typical patterns. We presented these models as stand alone techniques and in 5 we presented the architecture to a complete navigation algorithm taking into account the prediction coming from target tracking. However it would be suitable to integrate both prediction techniques in one only navigation structure so to exploit at maximum the available information in different situations. In all environments, the robot needs a target tracking method to be aware of the presence, the position and the velocity of the moving obstacles; medium term prediction instead is related to particular environments and it is also possible that some obstacle does not follow the pre-learned patterns; the best would than be to fuse the information coming form target tracking with the information coming from the pattern based medium term prediction when possible. We propose then to modify the prediction module to appropriately integrate the short term and the medium term prediction as shown in Figure 6.2. While the short term block is the same presented

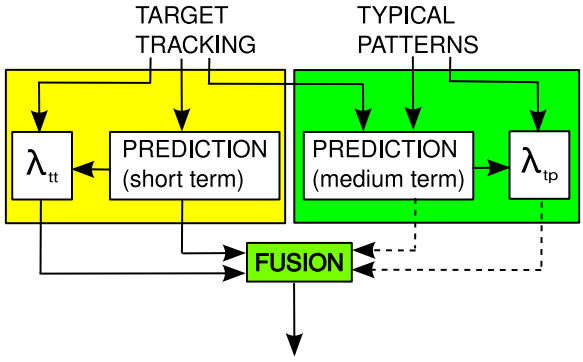


Figure 6.2: Prediction and fusion modules.

in Chapter 5, the block at the right performs prediction on the basis of the observed trajectories of the obstacles and the pre-learned typical patterns. The medium term prediction is obtained as detailed in Chapter 4 in dependence on the representation of the typical patterns themselves. The observations come from the tracking algorithm, which gives to the prediction module the actual position and velocity along with the associated covariance and the previous trajectory. The bottom block performs the fusion between the two predictions. To fuse these information it is necessary to weight the two models according to their *reliability*, separately at each time instant and for each object for which the prediction is required. The reliability measure should reflect the quality of the model with respect to the observations gathered, by considering statistical information conveyed in prior and current prediction and classification results and is expressed in Figure 6.2 by the vectors λ_{tt} for the target tracking and λ_{tp} for the typical patterns respectively. For example the kynematic model based on target tracking is not reliable for medium term: the maximum reliable time horizon changes largely for pedestrians and different kind of vehicles etc... On the other side, the pattern based models are less precise in the short term; furthermore, the fact that obstacle could assume an atypical behaviours or that the set of pattern learnt is not complete should not be neglected, especially in the case of on-line learning.

Target Tracking Prediction Reliability

The short term prediction block gives in output the sequences of predictions for each time horizon $k' = 1...k$ along with a sequence of confidence coefficients $\lambda_{tt}(t + k')$ for each object, where it must be $0 \leq \lambda_{tt}(t + k') \leq 1$, so that it is possible to compare it with the weights coming from the medium term prediction block.
The motion models estimated by target tracking techniques are usually linear and based only on the object recent movements. The accuracy of prediction in such models decreases with the distance of the prediction horizon from the present time, due to the model noise parameter set in the tracking filter. Also, these models are usually simplistic with respect to the real motion of objects, so the premise of using such models for predictions is that the model has a finite, *short* validity in time. The time length of this validity is highly application dependent. In practice, in the case of pedestrians, a linear (constant velocity or constant acceleration) motion model can be considered valid only for one or a few seconds. A common choice is to set a priori a maximum time horizon k_{tt} after which the prediction is not taken into consideration and instead the kynematic possibilities of the obstacle (maximum velocity in each reachable direction) are used (worst case approach). In principle, k_{tt} can be estimated on-line comparing previous predictions with the actual observations and setting a threshold on the minimum admissible prediction accuracy. So it can be set:

$$\lambda_{tt}(t + k') = 0, \ \forall k' > k_{tt} \tag{6.3}$$

while, for $k' \leq k_t t$, the coefficient depends on the prediction accuracy.
A usual measure for the accuracy of prediction is given by the expected data likelihood, i.e. given a prediction for a certain time horizon $t+k$, the accuracy is given by the likelihood of the effective observation at $t+k$:

$$E[L_p] \simeq \frac{1}{n}\sum_{n=1}^{N} P(O_{n,t+k}|O_{n,t}) = P(O_{n,t+k}|S_{t+k})P(S_{t+k}|O_t) \tag{6.4}$$

$$\lambda_{tt}(t+k) = E[L_p] \tag{6.5}$$

where E stays for the expected value which is approximated by the average over a finite set of N observations. To use a metric that has a direct geometric interpretation, the average error can be considered and is computed as the expected distance between the prediction for a certain time horizon and the effective observation. Considering a Gaussian prediction $G(\mu_{t+k}, \Sigma_{t+k})$, the average error for the single prediction is given by:

$$E[e] = ||O_{t+k} - \mu_{t+k}||^{1/2} \tag{6.6}$$

$$\lambda_{tt}(t+k) = \frac{1}{1+E[e]} \tag{6.7}$$

The Mahalanobis distance extends this notion and considers also the prediction uncertainty:

$$D_M(O_{t+k}, P(S_{t+k}|O_t)) = \sqrt{(O_{t+k} - \mu_{t+k})^T \Sigma_{t+k}^{-1} (O_{t+k} - \mu_{t+k})}. \tag{6.8}$$

$$\lambda_{tt}(t+k) = \frac{1}{1+D_M(O_{t+k}, P(S_{t+k}|O_t))} \tag{6.9}$$

This measure also may be generalized for a complete data set containing N prediction-observation sequences. Equations 6.7 and 6.9 give the confidence coefficient for each chosen measure respectively.

Typical Patterns Prediction Reliability

The medium term prediction block gives in output the sequences of predictions for each time horizon $k' = 1...k$ along with a sequence of confidence coefficients $\lambda_{tp}(t+k')$ for each object, where $0 \leq \lambda_{tt}(t+k') \leq 1$.
The pattern based models are based on large datasets of observation sequences, so it is expected that their prediction in distant time is more accurate and reliable. As for the target tracking case, the accuracy of prediction can be measured using the maximum likelihood criterion or looking at the average error, computed as the expected distance between the prediction for a certain time horizon and the effective observation.
However, typical patterns based prediction is reliable only if the object is effectively following one of the learned patterns and if the system is capable to choose the correct one. In [94] the *abnormality* of

an observation sequence is measured using the normalized log likelihood of the observation sequence for a given behaviour model. Given the observation sequence $O = [o1, ..., o_t, ..., o_T]$ and a set of typical patterns $M = [m_1, ..., m_I]$ the abnormality is computed as follows:

$$P(O|M) = \sum_{i=1}^{I} \frac{N_i}{N} P(O|m_i) \tag{6.10}$$

$$l_t = \frac{1}{t} log P(O_t|M) \tag{6.11}$$

$$Q_t = (1-\alpha)Q_{t-1} + \alpha(l_t - l_{t-1}), \;\; Q_1 = l_1 \tag{6.12}$$

$P(O|M)$ is the likelihood of observing sequence O given the model M, where N_i is the number of training trajectories belonging to the ith behaviour and N is the total number of training trajectories; equation 6.11 gives the computation of the normalized log-likelihood for instant t; finally, in equation 6.12 Q_t is the abnormality measure. A threshold Th_A is imposed, and abnormality is detected if $Q_t < Th_A$: in this case, the object does not follow any of the learned patterns in M. In this case, it should be setted:

$$\lambda_{tp}(t + k') = 0, \;\; \forall k' = 0, ..., k \tag{6.13}$$

The prediction and the decision should than be based on the tracking based model only. In the case on on-line learning and prediction the abnormal patterns observed could be used as training trajectories for the initialization of new patterns.
In [94] the reliability of a decision on the sequence O to belong to a specific pattern $m_i \in M$ is computed with a Likelihood Ratio Test:

$$r_i = \frac{\frac{N_i}{N} P(O_t|m_i)}{\sum_{j \neq i} \frac{N_j}{N} P(O_t|m_j)} \tag{6.14}$$

If the output prediction is given by only the most likely pattern, than it should be set:

$$i = \underset{j=\{1,...,I\}}{\text{argmax}} (r_j) \tag{6.15}$$

$$\lambda_{tp}(t + k) = r_i \cdot \frac{1}{1 + D} \tag{6.16}$$

where D represents one of the accuracy measures described in the previous paragraph.
In [95], it is argued that a large difference in the classifier outputs reflects a greater confidence. The reliability of an observation sequence belonging to a specific pattern is then measured by two *dispersion* measures: an instantaneous dispersion (equation 6.17), which evaluates the observation probability of the model for the single observation at t and a temporal dispersion (equation 6.18)

which evaluates the reliability of the observation sequence.

$$L_t = L(log(P(o_t|M))) = \frac{2}{I_b(I_b-1)} \sum_{i=1}^{I_b} \sum_{j=1}^{I_b} log\left(\frac{P(o_t|m_i)}{P(o_t|m_j)}\right) \quad (6.17)$$

$$R_t = \sum_{\tau=1}^{T} \alpha(\tau) L_{t-\tau\Delta t} + \epsilon_t \quad (6.18)$$

In equation 6.17, I_b is the number of best hypotheses being considered among the set of I patterns in $M = [m_1, ..., m_I]$.
In equation 6.18, T is the length of the temporal window being considered, and ϵ_t is a white noise with zero mean; $0 \leq \alpha(.) \leq 1$ is a discount factor for past observations and is determined as a function of the autocorrelation of the L_t series and the variance of ϵ.
In Chapter 4, both in the case of HMMs and GPs representation, we used a mixture of models to predict trajectories.
In the case of GPs based models we used the likelihood of the observation normalized to the set of patterns to weight each pattern in the Gaussian mixture. To detect an abnormal pattern it is possible to set directly a lower threshold to the Mahalanobis distance between the observation sequence and each pattern or to study the abnormality measure proposed in equation 6.12. If the object trajectory is considered abnormal, the estimated coefficient is zero (eq. 6.13). Otherwise, the prediction is given by the mixture obtained by the typical patterns above threshold Th_A, opportunely weighted. Inthis case, the dispersion of equations 6.17 and 6.18 can be used to measure the reliability of the prediction for each time horizon:

$$\lambda_{tp}(t + k') = \frac{1}{1 + \exp(-R_t)} \quad (6.19)$$

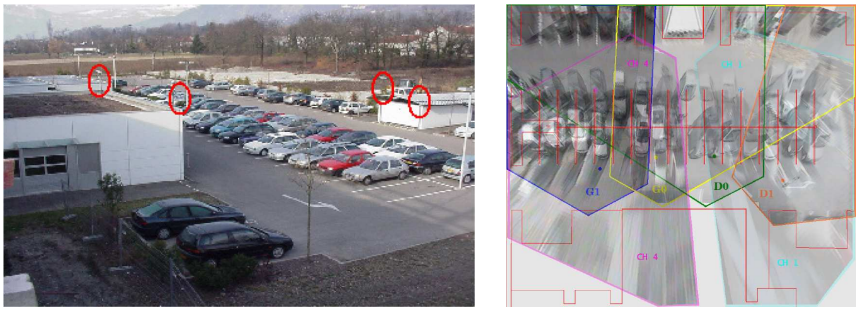
In the case of HMMs, a typical pattern has been represented by a graph leading to a goal state. The abnormality measure defined in 6.12 can be used referring the likelihood in eq. 6.10 to the graph of each goal; similarly, 6.17 and 6.18 can be applied.

Fusion

Given the predictions and their relative coefficients for each time horizon, fusion is performed choosing at each timestep the prediction with the maximum confidence coefficient. This choice is leaded by the fact that the both predictions are based on the same observations, and so cannot be considered as statistical independent. If the two sources of information can be considered independently one from the other, then the predictions at each time step are fused using Bayes' rule and multiplying the two prediction weighted by their confidence coefficient after normalization. If both confidence coefficients are set to zero, the prediction is performed according to the kynematic possibilities of the obstacle.

6.2.3 Communication with an off-board plateform: Parkview

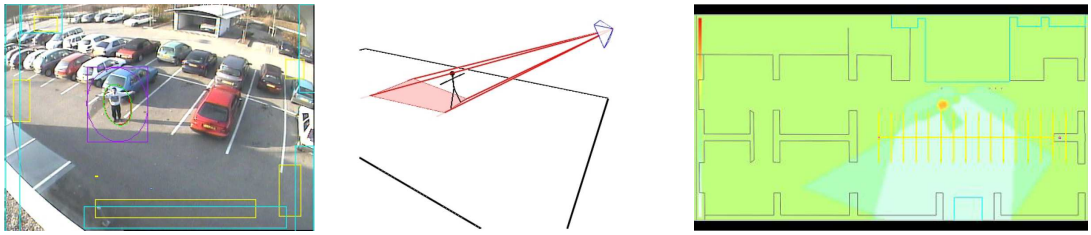
Parkview is an experimental plateform in the car-park at INRIA Rhône-Alpes (see Figure 6.3). The plateform is constituted by 4 cameras that look at the parking from different points of view, elaborate the images and send data to a central processing unit. Some zones of the parking are covered by the frustrum or more than one camera. Each camera elaborates the images acquired performing



(a) (b)

Figure 6.3: (a) The INRIA car park and the location of the cameras; (b) the ground plane with the frustrum of each camera.

background subtraction and detecting the moving areas that are identified with moving obstacles. A bounding box is associated to each moving obstacle. Using the projection matrix associated to the camera position and under the hypothesis that moving obstacles lie on the flat ground, the bounding box is projected into the ground plane. The space around the basis is considered occupied, while the space behind the bounding box is considered occluded. A sensor model has been built to fuse the information coming from each camera and an occupancy grid is built (see Figure 6.4) The occupancy grid can then be clustered and moving objects can be extracted and tracked (see [96]). This plateform is aimed at helping an autonomous vehicle to move autonomously and safely



(a) (b) (c)

Figure 6.4: (a) A tracked pedestrian; (b) The projection of the bounding box into the ground plane; (c) The occupancy grid obtained by the fusion of the information coming from two cameras: red stays for high probability of occupation, green for $P \simeq 0.5$ and white for free space.

in the area, providing a survey of the whole area and communicating to the vehicle the state of the

moving obstacles. The advantages of letting the robot communicating with an off-board plateform reside in the complementarity of the information gathered by the two intelligent systems. The off-board cameras in fact, have a different point of view on the workspace and can gather information about moving obstacles that are hidden from the robot point of view. Another advantage is that it is possible to move the medium-term prediction block from the robot to the off-board plateform (see Figure 6.2.3). In this way, not only the robot gains some computational time, but the whole architecture becomes more general: different plateforms in different environments can communicate with the robot in the same way; there is not need for the robot to know the particular environment and the related typical patterns, which are instead stored by the off-board plateform. Another advantage is that the typical patterns can be updated on-line by the plateform, without necessity to inform the robot or robots that move in the environment. Also it is possible to perform navigation at the same time of the learn and predict procedure, considering the appropriate reliability for prediction. The fusion between the on-board information and the off-board prediction should be

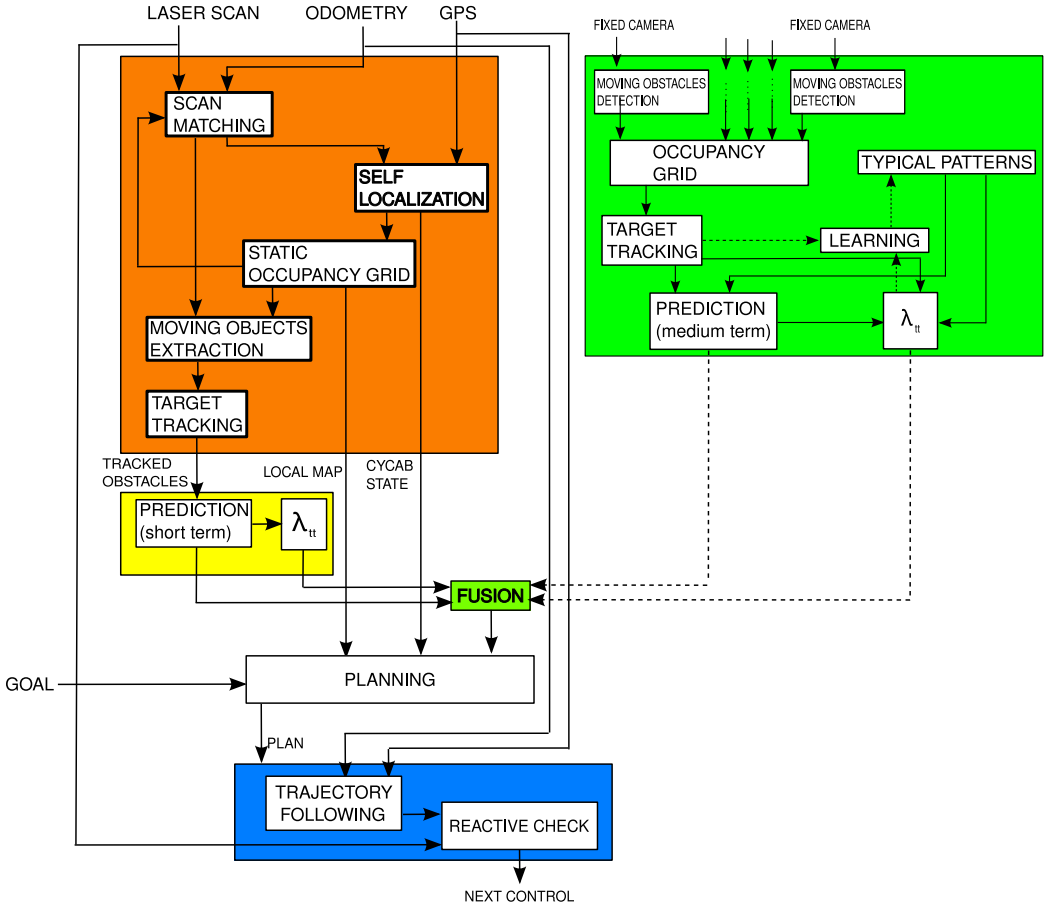


Figure 6.5: The navigation architecture in presence of the off-board Parkview plateform.

performed weighting the on board and the off-board reliability information (see §6.2.2). However, since the acquisition sensors and their points of view are different, all predictions are supposed to be performed in the GPS reference frame; It is also necessary to implement an algorithm to correctly associate each object observed by the off-board plateform with an object tracked by the robot or an object that is hidden from the robot point of view: a possible approach is the global nearest neighbohr algorithm used to associate the new observations with the tracked obstacles in the perception module (see §5.4).
Figure 6.6 shows the time evolution of the navigation algorithm in time. Arrows show the outputs

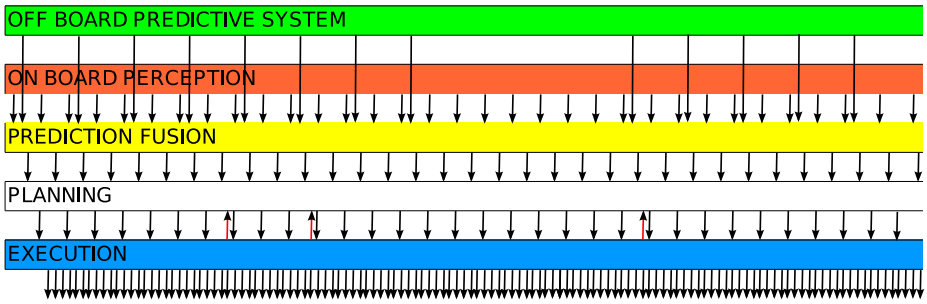


Figure 6.6: Scheme of the time evolution of the navigation algorithm.

of each module: each module works at a different frequency. In particular:

- it is expected that the off- board plateform prediction is slower than the on board once, due to the communication band limit;
- the execution module frequency is higher than the planning frequency: the next control is corrected at higher frequency using the odometric and localization data;
- the off board information is not synchron with the robot one and it is expected that there can be laps of time where no communication is possible (communication errors, absence of off-board plateform);
- the planning can be re-synchronized due to an unexpected event detected by the reactive algorithm in the execution module;

6.3 Perspectives: coordination among multiple, independent agents

In chapter 4 we introduced the hypotheses that object do not move at random in the space, but follow specific patterns. In particular, the HMM approach recalled made the assumption that objects

move in the space with the intention to reach specific goal positions. At the aim of prediction and navigation we also assumed that multiple objects in the environment move independently from each other and from the robot. However, it is a common experience that in the real world, people and vehicles do not move independently, but interact with each others and adapt their own motion to the actions of others, in order to avoid conflict situations. An autonomous robot moving in a dynamic environment modifies itself the behaviour of the others, with its presence and actions. Under these assumptions (objects move with the intention to reach a specific goal and objects interact with each others), each moving obstacle could be regarded to as a rational agent and the whole system could be modelled as a *Multi Agent System*. In particular, the intention to reach a goal position or to follow a specific path can be represented by a *utility function*, and it is assumed that the agent takes actions that maximize its utility function, having some knowledge and prediction of the state and intentions of other agents in the system.
From the point of view of the robot, it is possible to perform motion planning or obstacle avoidance in such environment with all the methods cited or developed in this thesis, however a more realistic prediction and a so more efficient navigation could be achieved when considering the interactions between people or vehicles and with the robot: this information could be exploited by the robot to better predict the future state of the world; future actions could be planned taking into account also the *reactions* of the dynamic obstacles in the environment. As a model of people interactions and people-robot interaction is hard to build and not available at present time –at our knowledge –, we continue our discussion considering multiple robots with independent goals, navigating in the same environment. The problem we could then address would be that of planning the actions for a set of autonomous robots having independent goals but moving in the environment (Multi Robot motion planning). Depending on the degree and representation of the information about the environment and about the other agents state and intentions, and depending on the possibility of explicit communication, various approaches have been developed in literature. Section 6.3.1 presents a brief and general overview of such methods; section 6.3.2 presents some ideas to extend the developed approach to the described case.

6.3.1 Multi robot coordination overview

Multiple-robot motion planning approaches are often categorized as centralized or decoupled.
A **centralized approach** typically constructs a path in a joint configuration space, given by the Cartesian product of the configuration spaces of the individual robots. For a feasible exploration of this large space, good heuristics or sampling based methods are needed. The path planning frameworks presented in Chapter 3 can then be easily extended to the multirobot case. Many approaches use potential field methods to guide the search [45, 46]. Other methods restrict the motions of the robots to reduce the size of the search space: for example in [97] the trajectory of each robot lies on an independent roadmap.

However, these methods do not necessarily represent an efficient use of resources: either the planner runs on a central machine which must be able to communicate with all the robots, either each robot plans itself for all the team, and again full communication is required among the team to send the sensor information of each robot to each other robot. In real world systems however, communication can represent a problem: bandwidth is always limited, which can cause latency between information sending and reception and so interfere with the time constraints of coordinated motion; also, emission power (and maximum communication distance) is limited; in some domains communication is not available at all (underground). It is then of vital importance to limit the communication to the strictly necessary information and to provide a model of when communication is possible. When the environment is unknown, centralized approaches have the advantage to use the information coming from all the team, and the plan of each robot depends of the total information acquired. If communication is not available, but each robot has a model of all other robots, only the common information should be used by each one to elaborate the same plan among the team. This is a drastic choice: in general the global plan is based on less information then the one gathered by each of the robots, so this method can often fail to find a feasible solution.

A **decoupled approach** typically generates paths for each robot independently. The simpler choice is to consider the interactions between the robots only in a second step: for example, the path of each robot is chosen avoiding the static obstacles and reaching to the goal, then the velocities along the path are adjusted (globally or locally) in order to avoid collision [98]. Other approaches generate a priority queue among the robots: the plans are computed one at the time following the queue; the robots whose trajectories have already been planned are considered as independent moving obstacles and their trajectory is avoided [99].

In more complete and recent approaches, robots planning their motion must take into account not only their own observations of world state, but also the possible observations and actions of teammates. Such a reasoning is a s strictly interconnected and complex process and seems to require an infinite recursion of beliefs to be modelled by each member of the team (I think that you think that I think...). An alternative is to limit the recursion to an arbitrary level that seems to respect the rational process of other agents, and that can be different from an agent to another: 0 level corresponds to an obstacle ignoring the presence and actions of other obstacles; level 1 corresponds to an obstacle making predictions of the others future actions on the basis of its observation before planning; level 2 corresponds to an obstacle prediction other obstacle motions in dependence of each of its possible actions etc...

A closed formulation of the problem is provided by Game Theory [100, 101]. In particular, non-cooperative game theory [102] is the study of how self-interested agents choose actions in the presence of other agents who are also choosing actions (strategic decision making). When trying to solve a game, one attempts to find an *equilibrium* where each player (interacting agent) has chosen a strategy that is the best given the other's strategies and which is then unlikely to be changed. Depending

on the if players have the same utility function (games with common payoffs), and of the quality of information (games of imperfect information, games of incomplete information) different models and solution methods have been proposed. In case of partial observability of the world state, the problem can be modelled as a Partially Observable Stochastic Game (POSG). POSGs generalize notions of single-stage games and Markov decision processes to both multiple agents and partially observable worlds. However POSGs are computationally intractable for all but the smallest problems. An interesting Bayesian game approximation to POSGs in which robots reason only a limited time ahead about uncertainty in world state is proposed in [103].
In the middle between centralized and decoupled approaches, a certain number of frameworks [104] suggests to exploit the topology of the team and apply a centralized approach only locally, where the system is known with more certainty and communication is easier; the composition of the groups is then modified on-line according to the changing positions of the robots.

6.3.2 Ideas for future work

We think that the probabilistic and partial framework developed in chapters 3 and 4 could be an interesting base to address the problem of multiple robots trying to reach independent goal positions or favourite trajectories while dealing with an uncertain knowledge of the environment and eventually of the state and intentions of other robots. However, we are aware that it exists a very large spectrum of research possibilities and it is not our aim to give an exhaustive list of possible approaches. In the following paragraphs, we only sketch some ideas to combine the approaches developed in this thesis with some of the approaches recalled in the precedent section. For simplicity we consider an homogeneous set of robots.

Centralized approach

A first possibility is to recur to a centralized approach: the state space where the plan is searched is composed by the joint configuration space of all the robots interacting in the environment, the state of each robot being constituted by its position and velocity and its goal. The search tree would be then grown in this space and the weight of the nodes would depend on a combination of the utility function related to the path of each robot, this last one given directly by the weight we have chosen for our algorithm (see Eq 3.5). Assuming full communication, each robot could communicate its state and observations to a central or supervising unit which would then build a map with the shared information, plan and update a partial plan for the system and communicate to each robot the actions it has to take. Alternatively, each robot could act as the central unit: each robot would communicate its state, goal and observations to all other robots; the same tree would be grown separately by each robot if all the robots have the same model and computational capabilities and are synchronised with each other.
If full communication is not available the task becomes significantly harder. The robots should find

a common plan on the basis only of the common knowledge. The problem of such an approach is that the dimension of the search space grows with the number of robots and that the common knowledge could be too little to allow planning in distant future.

Decoupled approach

Taking under consideration a decoupled approach, each robot could plan its motion considering other robots as independent moving obstacles. A first approach would be that each robot just track the robots in its field of view and plans its actions according to the approach described in Chapter 3. A second scenario can be imagined if each robot knows all the goals in the environment and has a favourite pattern to follow and reach its goal. To achieve planning with the hypothesis to follow a typical pattern, the search algorithm should be modified in order to opportunely weight the states along and "around" the favourite pattern, in terms both of space and time. Other robots in revenge, could be regarded as independent obstacles moving along known typical patterns, as in the approach described in Chapter 4: each robot estimates a probability distribution of the state and intentions of other robots and makes prediction on the basis of the a priori known patterns.
It is possible then to think to deepen the level of recursion in the prediction: for each searched action, the probabilistic prediction of the moving obstacles should be modified in dependence of the action itself. As we assumed that the robots are all the same, each robot can maintain an estimation not only over the state, but also the over the knowledge of other robots. Given its own action, the robot can then simulate the planning step of all other robots and define a new predicted probability distribution. With increasing complexity, the recursion level can be deepened at will.

Bibliography

[1] S. M. LaValle and R. Sharma, "On motion planning in changing, partially-predictable environments," *Int'l J. Robotics Research*, vol. 16, pp. 775–805, 1997.

[2] R. Smith and P. Cheeseman, "On the representation and estimation of spatial uncertainty," *Int. Journal of Robotics Research*, no. 5, pp. 56–68, 1986.

[3] A. Doucet, N. Freitas, and N. Gordon, *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, 2001.

[4] M. S. Montemerlo, "Fastslam: a factored solution to the simultaneous localization and mapping problem with unknown data association," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, USA, 2003, chair-Whittaker,, William and Chair-Thrun,, Sebastian.

[5] C. Fulgenzi, A. Spalanzani, and C. Laugier, "Dynamic obstacle avoidance in uncertain environment combining pvos and occupancy grid," in *IEEE International Conference on Robotics and Automation, ICRA*, 2007.

[6] ——, "Combining probabilistic velocity obstacles and occcupancy grid for safe navigation in dynamic environments," in *Workshop on safe navigation, IEEE International Conference on Robotics and Automation, ICRA*, 2007.

[7] C. Fulgenzi, C. Tay, A. Spalanzani, and C. Laugier, "Probabilistic navigation in dynamic environment using rapidly-exploring random trees and gaussian processes," in *IEEE/RSJ International Conference on Intelligent RObots and Systems*, 2008.

[8] C. Fulgenzi, A. Spalanzani, and C. Laugier, "Probabilistic rapidly-exploring random trees for autonomous navigation among moving obstacles," in *Workshop on safe navigation, IEEE International Conference on Robotics and Automation, ICRA*, 2009.

[9] S. Petti and T. Fraichard, "Safe motion planning in dynamic environments," in *IEEE IROS*, 2005.

[10] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, vol. 2, pp. 500–505, Mar 1985.

[11] J. Borenstein and Y. Koren, "The vector field histogram-fast obstacle avoidance for mobile robots," *Robotics and Automation, IEEE Transactions on*, vol. 7, no. 3, pp. 278–288, Jun 1991.

[12] I. Ulrich and J. Borenstein, "Vfh+: reliable obstacle avoidance for fast mobile robots," *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, vol. 2, pp. 1572–1577 vol.2, May 1998.

[13] R. Simmons, "The curvature-velocity method for local obstacle avoidance," *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, vol. 4, pp. 3375–3382 vol.4, Apr 1996.

[14] N. Y. Ko and R. G. Simmons, "The lane-curvature method for local obstacle avoidance," *Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on*, vol. 3, pp. 1615–1621 vol.3, Oct 1998.

[15] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *Robotics and Automation Magazine, IEEE*, vol. 4, no. 1, pp. 23–33, Mar 1997.

[16] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach," in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, vol. 1, 1999, pp. 341–346 vol.1.

[17] J. Minguez and L. Montano, "Nearness diagram navigation (nd): a new real time collision avoidance approach," *Intelligent Robots and Systems, 2000. (IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, vol. 3, pp. 2094–2100 vol.3, 2000.

[18] J. Minguez, "The obstacle-restriction method for robot obstacle avoidance in difficult environments," *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pp. 2284–2290, Aug. 2005.

[19] T. Fraichard and H. Asama, "Inevitable collision states. a step towards safer robots?" *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, vol. 1, pp. 388–393 vol.1, Oct. 2003.

[20] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *International Journal of Robotics Research*, vol. 17, pp. 760–772, 1998.

[21] F. Large, "Navigation autonome d'un robot mobile en environnement dynamique et incertain," Ph.D. dissertation, Université de Savoie, 2003.

[22] B. Kluge and E. Prassler, "Reflective navigation: individual behaviors and group behaviors," in *IEEE International Conference on Robotics and Automation, ICRA*, 2004.

[23] E. Owen and L. Montano, "Motion planning in dynamic environments using the velocity space," in *IEEE International Conference on Intelligent Robots and Systems, IROS*, 2005.

[24] S. Quinlan and O. Khatib, "Elastic bands: connecting path planning and control," *Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on*, pp. 802–807 vol.2, May 1993.

[25] V. Delsart and T. Fraichard, "Navigating dynamic environments using trajectory deformation," in *IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, France Nice, 2008. [Online]. Available: http://hal.inria.fr/inria-00293505/en/

[26] C. Coué, "Modéle bayésien pour l'analyse multimodale d'environnements dynamiques et encombrés : Application à l'assistance à la conduite en milieu urbain," Ph.D. dissertation, Institut National Polytechnique de Grenoble - INPG, France, 2003.

[27] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, pp. 46–57, Jun. 1989.

[28] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *Int. Journal of Robotics Research*, no. 17, pp. 711–727, 1998.

[29] F. Large, C. Laugier, and Z. Shiller, "Navigation among moving obstacles using the nlvo : Principles and applications to intelligent vehicles," *Autonomous Robots Journal*, vol. 19, no. 2, pp. 159–171, September 2005.

[30] J. C. Latombe, *Robot Motion Planning*. Dordrecht, The Netherlands: Kluwer, 1991, vol. SECS 0124.

[31] J. H. Reif and M. Sharir, "Motion planning in the presence of moving obestcles," in *IEEE Symposium of Foundations of Computer Science*, 1985, pp. 144–154.

[32] J. Canny, "Constructing roadmaps of semi-algebraic sets I," *Artificial Intelligence Journal*, vol. 37, pp. 203–222, 1988.

[33] J. T. Schwartz and M. Sharir, "On the Piano Movers' Problem: II. General techniques for computing topological properties of algebraic manifolds," *Advances in Applied Mathematics*, vol. 12, pp. 298–351, 1983.

[34] J. F. Canny, *The Complexity of Robot Motion Planning*. Cambridge, MA: MIT Press, 1988.

[35] N. J. Nilsson, "A mobile automaton: An application of artificial intelligence techniques," in *1st International Conference on Artificial Intelligence*, 1969, pp. 509–520.

[36] V. Akman, *Unobstructed shortest paths in polyhedral environments.* New York, NY, USA: Springer-Verlag New York, Inc., 1987.

[37] C. ó'Dúnlaing, M. Sharir, and C. K. Yap, "Retraction: A new approach to motion-planning," in *STOC '83: Proceedings of the 15th annual ACM symposium on Theory of computing.* New York, NY, USA: ACM, 1983, pp. 207–220.

[38] R. Brooks, "Solving the find-path problem by good representation of free space," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 13, no. 3, pp. 190–197, 1983.

[39] J. T. Schwartz and M. Sharir, "On the Piano Movers' Problem: I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers," *Communications on Pure and Applied Mathematics*, vol. 36, pp. 345–398, 1983.

[40] F. Avnaim, J.-D. Boissonnat, and B. Faverjon, "A practical exact planning algorithm for polygonal objects amidst polygonal obstacles," in *Proceedings IEEE International Conference on Robotics & Automation*, 1988, pp. 1656–1660.

[41] O. Khatib, "Commande dynamique dans l'espace opérational des robots manipulateurs en présence d'obstacles," Ph.D. dissertation, Ecole Nationale de la Statistique et de l'Administration Economique, France, 1980.

[42] ——, "Real-time obstacle avoidance for manipulators and mobile robots," *International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

[43] T. Lozano-Pérez, "Automatic planning of manipulator transfer movements," *IEEE Transactions on Systems, Man, & Cybernetics*, vol. 11, no. 10, pp. 681–698, 1981.

[44] R. A. Brooks and T. Lozano-Pérez, "A subdivision algorithm in configuration space for find-path with rotation," *IEEE Transactions on Systems, Man, & Cybernetics*, vol. SMC-15, no. 2, pp. 224–233, 1985.

[45] J. Barraquand and J.-C. Latombe, "A Monte-Carlo algorithm for path planning with many degrees of freedom," in *Proceedings IEEE International Conference on Robotics & Automation*, 1990, pp. 1712–1717.

[46] ——, "Robot motion planning: A distributed representation approach," *International Journal of Robotics Research*, vol. 10, no. 6, pp. 628–649, Dec. 1991.

[47] L. E. Kavraki and J.-c. Latombe, "Probabilistic roadmaps for robot path planning," 1995.

[48] P. Bessiere, J.-M. Ahuactzin, E.-G. Talbi, and E. Mazer, "The "ariadne's clew" algorithm: global planning with local methods," *Intelligent Robots and Systems '93, IROS '93. Proceedings of the 1993 IEEE/RSJ International Conference on*, vol. 2, pp. 1373–1380 vol.2, Jul 1993.

[49] S. LaValle and J. Kuffner, J.J., "Randomized kinodynamic planning," *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, vol. 1, pp. 473–479 vol.1, 1999.

[50] A. Foka and P. Trahanias, "Real-time hierarchical pomdps for autonomous robot navigation," *Robot. Auton. Syst.*, vol. 55, no. 7, pp. 561–571, 2007.

[51] A. Stentz, "The focussed d* algorithm for real-time replanning," in *In Proceedings of the International Joint Conference on Artificial Intelligence*, 1995, pp. 1652–1659.

[52] D. Ferguson and A. T. Stentz, "Anytime rrts," in *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '06)*, October 2006, pp. 5369 – 5375.

[53] N. Melchior and R. Simmons, "Particle rrt for path planning with uncertainty," in *2007 IEEE International Conference on Robotics and Automation*, April 2007, pp. 1617–1624.

[54] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, 2005.

[55] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping (slam): part i," *Robotics and Automation Magazine, IEEE*, vol. 13, no. 2, pp. 99–110, 2006.

[56] T.Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (slam): part ii," *Robotics and Automation Magazine, IEEE*, vol. 13, no. 3, pp. 108–117, 2006.

[57] N. A. Trung-Dung Vu, Olivier Aycard, "Online localization and mapping with moving object tracking in dynamic outdoor environments," in *IEEE Intelligent Vehicles Symposium*, Istanbul, 2007.

[58] P. Besl and H. McKay, "A method for registration of 3-d shapes," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 14, no. 2, pp. 239–256, Feb 1992.

[59] R. F. P. S. s. Blackman, *Design and Analysis of Modern Tracking Systems*. Norwood, MA, USA: Artech House, 1999.

[60] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[61] J. Burlet, "Suivi multi-objets adaptatif : Application la classification de comportements de mobiles," Ph.D. dissertation, Institut National Polytechnique de Grenoble, France, December 2007.

[62] C. Pradalier, J. Hermosillo, C. Koike, C. Braillon, P. Bessière, and C. Laugier, "The cycab: a car-like robot navigating autonomously and safely among pedestrians," *Robotics and Autonomous Systems*, vol. 50, no. 1, pp. 51–68, 2005.

[63] D. A. Vasquez Govea, "Incremental learning for motion prediction of pedestrians and vehicles," Ph.D. dissertation, Institut National Polytechnique de Grenoble, Grenoble (Fr), February 2007.

[64] C. Tay and C. Laugier, "Modelling paths using gaussian processes," in *Proc. of the Int. Conf. on Field and Service Robotics*, 2007. [Online]. Available: http://emotion.inrialpes.fr/bibemotion/2007/TL07

[65] D. Makris and T. Ellis, "Finding paths in video sequences abstract," in *British Machine Vision Conference*, 2001, pp. 263–272.

[66] I. Junejo, O. Javed, and M. Shah, "Multi feature path modeling for video surveillance," *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, vol. 2, pp. 716–719 Vol.2, Aug. 2004.

[67] D. Vasquez and T. Fraichard, "Motion prediction for moving objects: a statistical approach," in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, New Orleans, LA (US), April 2004, pp. 3931–3936. [Online]. Available: http://emotion.inrialpes.fr/bibemotion/2004/VF04

[68] M. Bennewitz, W. Burgard, and S. Thrun, "Learning motion patterns of persons for mobile service robots," *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, vol. 4, pp. 3601–3606 vol.4, 2002.

[69] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977.

[70] W. Hu, D. Xie, T. Tan, and S. Maybank, "Learning activity patterns using fuzzy self-organizing neural network," *Systems, Man, and Cybernetics, Part B, IEEE Transactions on*, vol. 34, no. 3, pp. 1618–1626, June 2004.

[71] M. Bennewitz and W. Burgard, "Adapting navigation strategies using motion patterns of people," in *In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA*. IEEE, 2003, pp. 2000–2005.

[72] L. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, Feb 1989.

[73] S. Lloyd, "Least squares quantization in pcm," *Information Theory, IEEE Transactions on*, vol. 28, no. 2, pp. 129–137, Mar 1982.

[74] T. Kohonen, *Self-Organizing Maps*, ser. Springer Series in Information Sciences. Springer Verlag, 1995, vol. 30.

[75] T. Martinetz and K. Schulten, "Topology representing networks," *Neural Netw.*, vol. 7, no. 3, pp. 507–522, 1994.

[76] B. Fritzke, "A growing neural gas network learns topologies," in *Advances in Neural Information Processing Systems 7*. MIT Press, 1995, pp. 625–632.

[77] J. Jockusch and H. Ritter, "An instantaneous topological mapping model for correlated stimuli," *Neural Networks, 1999. IJCNN '99. International Joint Conference on*, vol. 1, pp. 529–534 vol.1, 1999.

[78] M. Walter, A. Psarrou, and S. Gong, "Learning prior and observation augmented density models for behaviour recognition," in *BMVC*, 1999.

[79] M. Bennewitz, W. Burgard, G. Cielniak, and S. Thrun, "Learning motion patterns of people for compliant robot motion," *The International Journal of Robotics Research (IJRR)*, vol. 24, no. 1, 2005.

[80] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains," *The Annals of Mathematical Statistics*, vol. 41, no. 1, pp. 164–171, 1970.

[81] N. Friedman, "Learning belief networks in the presence of missing values and hidden variables," in *Proceedings of the Fourteenth International Conference on Machine Learning*. Morgan Kaufmann, 1997, pp. 125–133.

[82] Y. Singer and M. K. Warmuth, "Training algorithms for hidden markov models using entropy based distance functions," in *Advances in Neural Information Processing Systems 9*. MIT Press, 1996, pp. 641–647.

[83] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[84] Http://cycabtk.gforge.inria.fr/wiki/doku.php?id=start.

[85] Http://mgengine.wiki.sourceforge.net/.

[86] R. Brockett, R. S. Millman, and S. H. J., "Asymptotic stability and feedback stabilization," *Differential Geometric Control Theory*, pp. 181–191, 1983.

[87] C. Samson and K. Ait-Abderrahim, "Feedback stabilization of a nonholonomic wheeled mobile robot," in *Intelligent Robots and Systems '91. 'Intelligence for Mechanical Systems, Proceedings IROS '91. IEEE/RSJ International Workshop on*, Nov 1991, pp. 1242–1247 vol.3.

[88] Y. Kanayama, Y. Kimura, F. Miyazaki, and N. T., "A stable tracking control method for an autonomous mobile robot," in *IEEE International Conference on Robotics and Automation, ICRA*, 1990.

[89] G. Walsh, D. Tilbury, S. Sastry, R. Murray, and J.-P. Laumond, "Stabilization of trajectories for systems with nonholonomic constraints," in *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, May 1992, pp. 1999–2004 vol.3.

[90] Z. P. Jiang, "Lyapunov design of global state and output feedback trackers for nonholonomic control systems," *Int. Journal of Control*, no. 73, pp. 744–761, 2000.

[91] Z. Qu, J. Wang, C. Plaisted, and R. Hull, "Global-stabilizing near-optimal control design for nonholonomic chained systems," *Automatic Control, IEEE Transactions on*, vol. 51, no. 9, pp. 1440–1456, Sept. 2006.

[92] P. Morin and C. Samson, "Trajectory tracking for non-holonomic vehicles: overview and case study," June 2004, pp. 139–153.

[93] C. Pradalier, "Navigation intentionnelle d'un robot mobile," Dissertation, Inst. Nat. Polytechnique de Grenoble, Grenoble (FR), September 2004.

[94] T. Xiang and S. Gong, "Incremental and adaptive abnormal behaviour detection," *Computer Vision and Image Understanding*, vol. 111, no. 1, pp. 59–73, 2008.

[95] M. Makkook, O. Basir, and F. Karray, "A reliability guided sensor fusion model for optimal weighting in multimodal systems," in *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, April 2008, pp. 2453–2456.

[96] O. Aycard, A. Spalanzani, J. Burlet, C. Fulgenzi, T. D. Vu, D. Raulo, and M. Yguel, "Pedestrians tracking using offboard cameras," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, Oct. 2006, pp. 513–518.

[97] S. M. LaValle and S. A. Hutchinson, "Optimal motion planning for multiple robots having independent goals," *IEEE Trans. on Robotics and Automation*, vol. 14, pp. 912–925, 1996.

[98] O. Donnell and Lozano-Pèrez, "Deadlock-free and collision-free coordination of two robot manipulators," in *IEEE International Conference on Robotics and Automation, ICRA*, 1989.

[99] Erdmann and L.-P. erez, "On multiple moving objects," *algorithmica*, vol. 2, pp. 477–521, 1987.

[100] G. Owen, *Game theory.* Philadelphia, PA: W.B. saunders Company, 1968.

[101] T. Fudenberg and J. Tirole, *Game Theory.* Cambridge, MA: The MIT Press, 1991.

[102] ——, *Dynamic noncooperative Game Theory.* New York: Academic Press, 1982.

[103] R. Emery-Montemerlo., "Game-theoretic control for robot teams," Dissertation, Robotics Institute, Carnegie Mellon University, 2005.

[104] C. M. Clark, "Probabilistic road map sampling strategies for multi-robot motion planning," *Journal of Robotics and Autonomous Systems*, 2005.