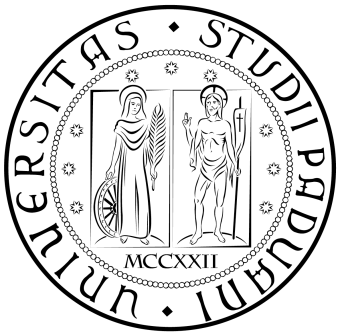

UNIVERSITÀ DEGLI STUDI DI PADOVA

FACOLTÀ DI INGEGNERIA
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA MECCATRONICA

TESI DI LAUREA MAGISTRALE

CONTRIBUTION IN DESIGN AND IMPLEMENTATION OF AN AUTONOMOUS MOBILE ROBOT.

Relatore: Ch.mo Prof. Roberto Oboe

Correlatore: Ch.mo Prof. Cyril Novales

Correlatore: Ing. Nicolas Morette

Laureando: Nicola Carretta
620030-IMC

ANNO ACCADEMICO: 2012-13

Summary

This paper is the result of six month thesis work performed in the robotic team inside the PRISME laboratory. This last one is a research laboratory, situated in Bourges (France), one of the four campuses of University of Orleans. In this thesis the candidate has done a study centred on the autonomous robot mobile navigation and robot's control architecture. The candidate works on some points to performing simulations, validate and implement algorithm for autonomous robot navigation, based on direct kinematic model and predictive control approach. This navigation method was conceived by the team of PRISME laboratory established on the occasion of a national french project: PROTEUS Project, that will be described in the next chapter. This document is divided in following points:

- After a brief introduction of laboratory research fields and robots model that have been performed, the first chapter will show the Proteus overview work and the main proposed scope of this project: it provides a complete toolset including simulation environment and middleware. Will be presented a robotic prototyping platform and sensors that were used for practical demonstration of navigation.
- In the second chapter a robot's control architecture conceived by Prisme laboratory will be presented. It consists in a hierarchical, modular and multi-layer architecture of function modules that allows a mobile robot to plan its tasks and to react to events.
- In the third part the *pilot* functional block of robot's control architecture will be presented. The chapter explains a reactive method, based on a deformable virtual zone “placed” around the robot to recognize and avoid obstacles.
- The fourth chapter treats the problem of robot localization into a known environment. The robot absolute position into a map will be used from navigation algorithm presented in this thesis. The localization algorithm has been implemented and tested in the robot platform, using a deterministic optimization algorithm.
- In the fifth chapter there will be an introduction about navigation tasks for autonomous mobile robots and a classification of navigation methods.
- The sixth chapter will describe a navigation method by direct kinematic model conceived by the team of Prisme laboratory that the candidate has had the pleasure of working with. The chapter will show step-by-step the navigation method principle and lastly will explain the use of a stochastic optimization algorithm (simulated annealing).
- In the seventh chapter there will be robot navigation results, obtained thanks c/c++ language implementation using RTMaps middleware. Will be presented a scenario that shows the robot navigation tasks such as reach way-points and avoiding obstacles. Will be presented robot performance into environment, useful to carry out settings of simulated annealing giving high probability to find suitable solutions.
- The last chapter gives the conclusions of the work and future proposals to be developed for the Proteus project.

Contents:

1 Introduction

The purpose of the thesis activity has contributed to achieve the implementation of an autonomous robot mobile. The robot must be able to perceive correctly the environment and react, depending on the level of autonomy, in order to planning trajectories and determine what movements will need to achieve its goal. The goals for an autonomous robot mobile could be dedicated for people's transport, surveillance, clearing explosive and more other. These performing tasks could be achieved with less human presence and moving in the environment without any external assistance.
This thesis is centred on implementation, test and validation of robot localization and navigation algorithms. Some algorithms have been tested under simulation with MatLab, and on the real robot mobile: WifiBot off-road prototyping robot. Data communication with the robot platform have been ensured thanks a middleware (RTMaps) that offers a modular platform where data samples flow between functional blocks. In addition meetings were held for interacting with software developer company. Has been used the RTMaps Software Development Kit (DSK), allowing the possibility to develop and compile own components useful for realize the mobile robotic architecture: in particular functional blocks like navigator, localization, pilot and many other functions. Algorithm results were displayed using a computer station, with only functions of monitoring parameters and trajectories, by the fact that robot decisions must be taken autonomously.
Robot navigation and localization imply also the uses of optimization algorithms (deterministic and stochastic) to achieve its tasks considering the robot kinematic constraints and perceiving the environment necessary for the total autonomy of robot mobile. Were perceived advantage from simulated annealing optimization algorithm referring to its robustness and high probability to achieve an optimal solution previous suitable values parametrization. Demo's results of navigation and localization algorithm will be displayed into last chapters thanks RTMaps and Gnuplot graphic interface. Will be demonstrated the ability of the robot to perform planned trajectories, moving safely and bypassing obstacles thanks the implemented algorithms. This thesis work, performed under the national ANR research program Proteus, will be integrated into the robotic development team of Prisme laboratory. Allowing this last one results of the new theory of navigation. During the internship the candidate has attended to projects Proteus and Protech (a tele-ecography laboratory project) meetings. He has participated also to a national conference: “Control Architectures of Robots” (CAR Nancy,2012) and a meeting at Dassault premises for the Proteus project for the WifiBot demonnstration.

1.1 PRISME laboratory

The thesis activity was held to PRISME laboratory (Pluridisciplinaire de Recherche en Ingénierie des Systèmes, Mécanique, Energétique – Multidisciplinary Research in Systems, Mechanics and Energy) situated in Bourges (France). The PRISME laboratory (formerly known as the Laboratory of Vision and Robotics funded in 1988), is part of Orleans research laboratories. Currently gather about 80 research fellows. The vocation of the PRISME Institute is multidisciplinary in the Engineering Sciences, and spans a broad spectrum of disciplinary fields including engine combustion, energy,

aerodynamics, equipment mechanics, signal and image processing, automation and robotics.

1.1.a Research fields

One of the project-team (i.e. Robotic Interactive System – SRI) of the PRISME institute leads research in autonomous and tele-operated and medical robotics. The PROSIT project is a light body-mounted robot (Fig. 1.a) to perform tele-ecography: the robot placed on the patient is able to position and orient the probe faithfully reproducing the actions of remote doctor which manage a false reproducing probe.

Mobile robotic is a major topic in SRI team PRISME Institute. This last years of work lead to provide a multi-level robot control architecture which merged various degrees of tele-operation and autonomy. Nowadays, it works on a model based predictive control for mobile robots navigation, mainly dedicated to CyCab (Fig. 1.b) and WifiBot platform (Cap. 1.2.c).
Other robot models have been performed at the beginning by applied research in robotic field, mainly with BA System Company in 1996 to design a wireless guided indoor industrial mobile robot. Afterwords addressed the problem of unknown but structured environment, without absolute sensors. These works lead to validation on an indoor mobile robot (RAOUL) using rotating laser range telemeters, able to evolve autonomously in unknown environment. In 2001, has been designed and implemented a mechanical motion system allowing a ground vehicle to perform omnidirectional movements (ROMNI). Until today to apply research as stated above to evolve a multi-level robot control architecture.

Fig. 1: a) Tele-operated and b,c) mobiles robots.

1.2 PROTEUS Project overview

PROTEUS (“Plateforme pour la Robotique Organisant les Transferts Entre Utilisateurs et Scientifiques”) goal is to establish and organizing interactions between academic world and industrial partners of the robotic community in order providing suitable tools and models. To help a more easily transfer of knowledge act also to identify potential problem in the industry. Many actors are included in this project such as:

Fig. 2: Proteus project partners.

The PROTEUS project is funded by the french national research agency ANR (Agence Nationale de la Recherche) in the framework of the 2009 ARPEGE (Systèmes Embarqués et Grandes Infrastructures) call for proposal. This is a four year project started in november 2009.

One of main proposed scope of PROTEUS project is provide a complete Toolset including simulation environment and a middleware. Where simulator and middleware compatibility is ensured by the use of a ROS bus for communications allowing a total reversibility. The Proteus overview work is schematized in Fig. 3:

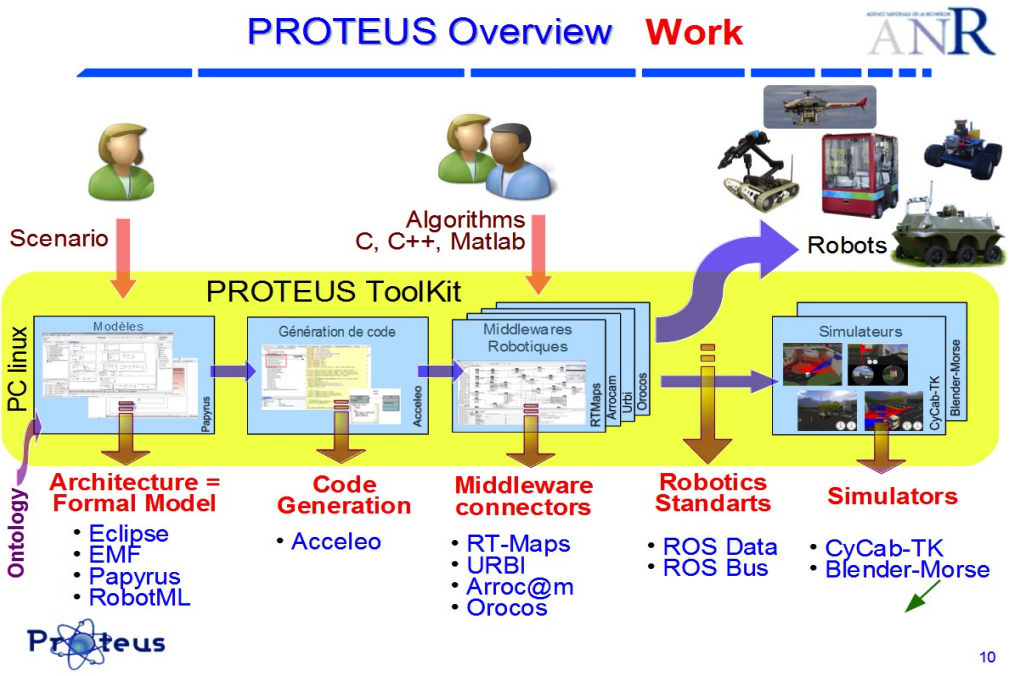

Fig. 3: Proteus overview work.

1.2.a PROTEUS Robot Youth Challenge (RYC)

The Robot Youth Challenge is proposed by the PROTEUS project consortium (http://anr-proteus.fr/). It is organized by the PRISME laboratory, and over this last one, regrouping other four research institutes (Blaise-Pascal, GREYC, INRIA, LIP6) and eight industrial partners (Dassault, ECA, Gostai, Intempora, Thales, CEA, ONERA, Effidence).
The RYC considers the problem of autonomous motion (outdoor) exploration and object searching in unknown environment. The main mobile robotics topics (Fig. 4) are:

- Autonomous navigation
- Obstacle avoidance
- Mapping
- Visual recognition

which they must be met to carry out the trials of the challenge. The challenge is divided in three event each one with different goals. They consist in operation of exploration, finding, sharing/organizing with other robots, and come back in an initial position; computed in a certain prefixed time. In order to minimize the time, in the last event, is permitted promoting data sharing between the robots that they will cooperate during the challenge.

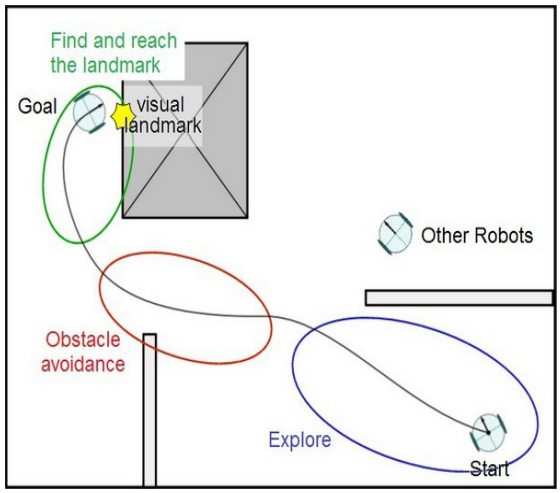

Fig. 4: Robot Youth Challenge scenario.

1.2.b Introduction to ToolKit PROTEUS

PROTEUS combined several tools (Fig. 3) in order to provide a complete uniform tool-chain for robotic development: the project has developed a toolset for the design of a complete robotics application, from its architecture modeling (with PAPYRUS), to the deployment on real robotics system (WifiBot, Pahlavi, Air Trooper, Res sac, Camel eon) or software simulation. All this through several middleware (RTMaps, Ur bi, Carrot, Siroccos). Most of them are open-source software.
The toolkit is structured in three part: meta-modeling & code generation, robotics

middleware, and simulation engines.

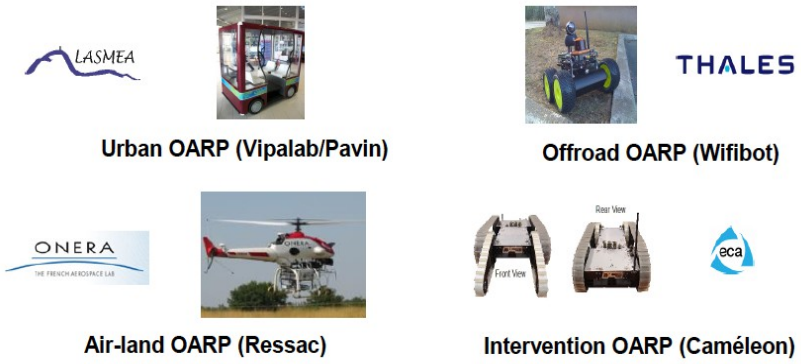

Fig. 5: Urban OARPS (Open-Access Robotic Platform.) robots types.

Meta-modeling & Code generation:
This part aim using software development environment like Papyrus, TOM, ALF to modeling and organize the structure of the robotics platform.
Papyrus provides diagram editors: with the use of connections is possible create a link between main functional blocks (e.g. navigator, actuators, database) and device like sensors (e.g. telemeter laser, IMU). The final structure appears in the form of a skeleton. TOM is a language extension designed to manipulate tree structures and XML documents. It provides pattern matching facilities to inspect objects and retrieve values. In the PROTEUS toolkit, Papyrus is used to design control architectures which are then generated on the different middleware supporting the project.

The middleware

This part aim to introduce the middleware. It was used consistently for implementing the robot platform because offer an important tools supports. On PROTEUS project RYC three middleware are available:

- Carrot (from the Effidence company)
- RTMaps (from the Intempora company)
- URBI (from the Gostai company)

In computing system, middleware is a set of programs which mediate between several applications and software components. Can be described as “software glue”, his function is to mediate interaction between the parts of an application, or between applications.
Picture 6 shows how RTMaps middleware (Real Time, Multistory, Advanced Prototyping Software), prove to be a modular platform where data samples flow between functional blocks (called components, details more over). Data flowing can be of any type as video, audio, bytes stream, CAN frames, matrices, vectors of integers or floats, text, and so on. More, for example, RTMaps is a multi-threaded architecture which allows the use of multiple asynchronous sensors within the same application i.e. “on the flow”, each data sample being retrieved at its own frequency. Data fusion algorithms can be developed thanks to the real time capabilities of the software, which

permit supports “any” type and quantity of sensors and actuators. RTMaps has also a Software Development Kit (SDK) which is a set of C++ libraries, header files, wizards, which allow the possibility to develop and compile own components useful for realize the mobile robotic architecture. It provides also a graphic interface for monitoring, display and recording data:

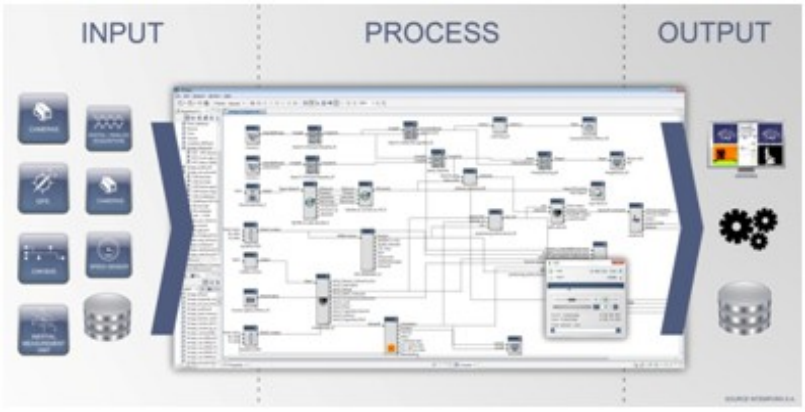

Fig. 6: RTMaps structure.

Middleware covers an important task simplify view of a system which a high set of interfaces and it cover also a wide range of software systems, including distributed objects and components, message-oriented communication, and mobile application support [1]. A system is organized as a set of parts, or components. Each of this one fulfill a function that should be consist in a service. The specification of the service is often associated to system functional property. It included the following aspects:

- *Availability*. The availability of a service quantifies the ratio of services ready to use. Is a statistical measure penalized by failures.
- *Performance*. This quality covers several aspects. Essential for real-time applications where attention is give e.g. to processing speed.
- *Security*. Security cover an important role to guarantee the respect of the rule from users, and access right control.

Multilevel architecture:
Middleware makes easier for software developers to perform communication and input/output. A *layered architecture* consists to decomposing a complex system into layers which includes libraries that provide services (i.e. data storage, screen display, multimedia, web browsing and much more). This layer organization provides guidelines for decomposing a complex system into parts allowing a high level of service to users and a high level of abstraction to developers by masking the heterogeneity and the distribution of the underlying hardware and operating systems, and by hiding low-level programming details [1]. The interface provided by each level may be viewed as set of functions defining a library, called Application Programming Interface (API).
Frameworks:
Software frameworks allow to software developers to reuse working code [2]. A software framework is a program skeleton aim to solve a family of related problems. According to well-defined rules the skeleton should be adapted or directly reused. Patterns, frameworks and middleware play a complementary role for improving the process of design, building and documenting applications increasingly complex of

today.

Objects:
Object in programming, mean of structuring computing system, is a software representation of a real-world entity like a person, a robot , a bank account, etc. that should be viewed as an association of a state and a set of procedures (or methods). The object model has the following properties:

- Encapsulation. The only way of accessing to object's state is through its interface, no part of the state is visible from outside the object. An interface comprises a set of methods (procedures) and attributes (values that may be read and written).
- Classes and instances. A class is a generic description common to a set of object (the instances of the class). The instances of a class have the same interfaces, and their state has the same structure; they differ from their value only.
- Inheritance. A class that derive from another one class preview specialization defining additional methods, attributes, or by redefining existing methods. Is said to inherit (or extend) from a class.
- Polymorphism. Polymorphism is the ability for a method to accept parameters of different types, and to have different behavior for each of these types.

Components:
A compositional architecture define the organization of a software system as an assembly of components, connectors, and composition rules.

- A *component* perform a specific function, can be assembled with other components, and provided interfaces: the only way to use a component.
- A *connector* is a device for assembling several components together in order to create a configuration.
- *Composition rules* specify the allowed ways of assembling a configuration out of components and connectors.

The Simulator:

Motivations: Autonomous robotic mobile often operate in a open and dynamic environments (which should be totally or in partially unknown), time and dynamics play a major role, and last but not least establish interactions with humans. It's easy to understand the importance function of a simulator platform where field trials are associated with high risk to the survivability of the system (e.g. aerial vehicles, and maritime) and should leads damage and risk to people. The use of a simulator as a previous step may be useful to evaluate algorithms, techniques, and verify their robustness; also positive side effects about time and cost savings, that simulator typically provides in research and robotic industry. On the other hand, it's difficult to replace the experiences of working on real robots in real environments with real-world sensors errors and unpredictable dynamics. But simulators certainly provide a convenient means for all the advantages mentioned above and by the fact it play an important role in risk reduction; and also should be useful to development of contingency management plans.
Below in picture 7 display the simulator software structure. Blender, MORSE, ROS, and RTMaps are mainly softwares used un PROTEUS Project.

Fig. 7: Blender-Morse structure simulator.

Blender [3] is a free and open-source modeling and rendering application that enables the creation of a diverse range of 2D and 3D contents. The source code is available under the GNU GPL license. Blender provides a board spectrum of modeling, texturing, lighting, animation and video post-processing functionality to satisfy the main purpose: the creation of computer generation images and animations.
The most advantage of Blender is the high level of graphical detail that can be achieved in real time, thanks to the advanced modeling of meshes, and effects such as texturing, lighting and shaders. For example when simulating robotic vision, visual aspect is important , since the images captured in the virtual world can be realistic enough to be processed with the same algorithms as real images.
Blender permits also an interactive simulation thanks the Game Engine (GE) mode [4]. It permits with a flexible graphical interface (called the Logic Bricks) to script behavior to objects in the scene, and define variables (called Logic properties) associated with the same objects. Thought dedicated API permits the iteration between Blender world and Python scripts (i.e. a Logic Bricks, over mentioned) or by additional modules that can be programmed in C/C++. Fig. 8 shows a Blender window performing a WifiBot robot model included by their equipment sensors.

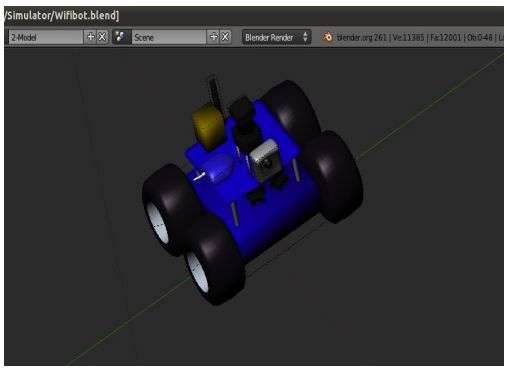

Fig. 8: Blender WifiBot robot model.

MORSE is a open source robotics simulator; projected by LAAS [5] Robotics Research Group in Toulouse, France. MORSE is based on the Blender 3D graphics program and it relies on a component-based architecture to simulate sensors, actuators and robots. It is constructed as a library of modular components that can be used to build any kind of robot and test its behavior under various conditions. MORSE can simulate complex robots in real time; has also been designed to be able to handle more robots in joint simulation scenario. MORSE can be run as a distributed network of simulation nodes; each one automatically synchronizes with the others.
The simulator aims to comply with the following requirements:

- General purpose robotics simulator
- Modular reusable components
- Multi-robots
- Variable levels of realism
- Direct interface with robot software
- Middleware independent
- Distributed architecture

MORSE provides a set of standard sensors (e.g. cameras, laser scanner, GPS, odometry), actuators (e.g. speed,controllers), and robotic bases (e.g. generic 4 wheel vehicle), but new ones can be easily be added.

ROS (Robot Operating System) [6] is a software framework which provides a set of functionalities in the development of robotic controllers. To specify that is not an operating system in the traditional sense of process management and scheduling like the abbreviation suggest; it provides a structured communications layer above the host operating system.
ROS provides standards operating system services such as hardware abstraction, low-level device control, implementation of commonly-used functionality, package management, and it takes care of various low-level functions such as message-passing between the different sub-system. ROS provides libraries and tools to help software developers. It is based on a graph architecture where processing takes place in nodes that may receive, post and multiplex sensor, control, state, planning, actuator and other messages.
At this time, ROS is mainly used for research applications in fields such as autonomous robot, but is intended also to ease the integration into industrial applications.

Overall Architecture:
MORSE is built on top of the Blender software: relies in a composition of Blender files to build simulated scenes. Each MORSE component consists into two files: a Python and a Blender file [4].The first one define an object class for the component type, with its state variables, data, and logical behavior (methods). The Blender file specifies physical properties of the object in the simulated world like material, color, surface.
There are different kind of components in MORSE. Mainly used in robotics simulations are: sensors, actuators, robots, scenes, and modifiers.

- Sensors recover data from the simulated world, emulating the functionality of the real sensors.
- Actuators aims to execute actions to the associated components. In particular

actuators move robots in function by given parameters type, in function at various levels of abstraction of the simulation. For example in path following, a low abstraction giving the robot commands like angular velocities for each wheel, higher abstraction simulation, using a direct destination coordinate (Fig. 9).

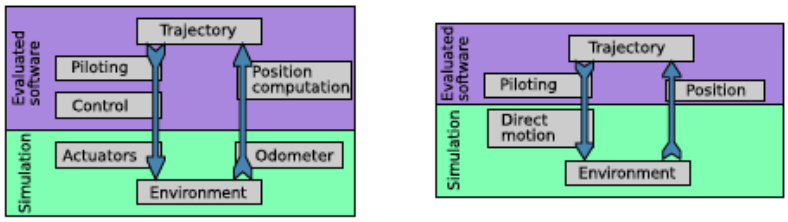

Fig. 9: Simulation of a trajectory following process at two different abstraction levels. On the left: low abstraction simulation giving directly actuators commands. On the right: higher abstraction simulation.

- Robots are the platforms where sensors and actuators are mounted.
- Scenes are the modeled environments where the robots interacts during the simulation. Can represent for example a indoor, outdoor scenes with all necessary to simulate a realistic ambient.

Fig. 10: Screenshots of MORSE simulator.

- Modifiers are function that alter data (e.g. noise functions), their function is to make realistic acquisitions from simulated sensors by the fact that these last one produce very accurate measures taken from the virtual world.

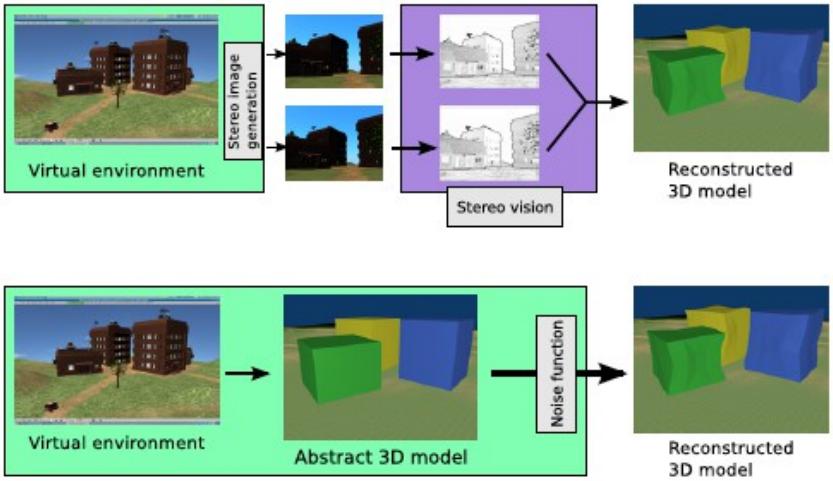

Fig. 11: Simulation of a 3D perception scene at different abstraction levels.

Important considerations are necessary about the integration of Morse components with middleware by the fact that this last one enable communication and data sharing between components. A highly coupled of an element with a given middleware make difficult to reuse it in a different environment. For this reason components should be designed to be middleware-independent [4] thanks a software package used at LAAS: GenoM 3 [7], a tool generating software modules that can be compiled with any middleware. All this permit to use various middleware such as YARP [8], ROS [6], Pocolibs [9] and other offering an high level of integration with middleware.

An other important aspect of MORSE is designed to interact directly with the software under test, without the need of software modifications. This philosophy takes after "Hardware-in-the-Loop" simulations. The evaluated components (the same in the target hardware) interact with the simulator with the same protocols than the ones interacts with real sensors and actuators of the robot [10].

Fig. 12 display how data flow between components (sensor and actuator) of the simulator using *hooks* to share data with external applications. *Hook* is a mechanism implemented in Python which consist adding at runtime (thanks the dynamic nature of Python) methods to the component instances. These methods use the data of MORSE components to elaborate it and then sent in the format required by the corresponding middleware.

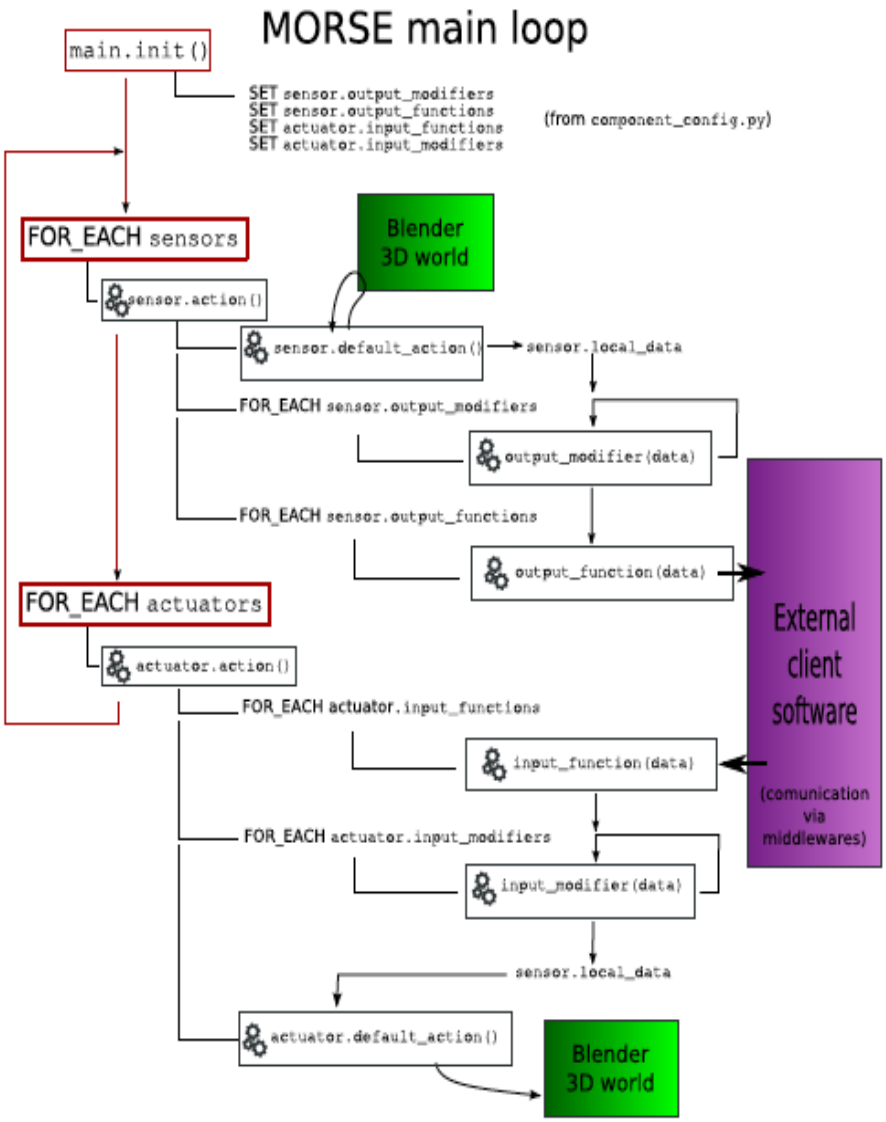

Fig. 12: Simulator structure.

1.2.c WifiBot mobile robot equipment

The WifiBot is a off-road robot model [11] which lends as a multi-purpose platform running in Linux or Windows embedded. Some reasons why this kind of robot leads this choice is for to make simple yet useful and affordable robotics. This platform should suit fulfill for flexibility and open modular architecture, fully programmable, low cost, small size, low weight, integration within Wi-Fi network, great for multi-robot applications. The base system is composed by four wheel drive chassis controllable using RS232, four infrared sensors, a Intel Atom D510 duo core running in Linux Ubuntu, installed on a 4 Gigabyte compact flash, a free Wi-fi access point, a IP-Camera, GPS, IMU, and a telemeter laser. Picture 13 shows the robot model before mentioned, well equipped with inertial measurement units and telemeter-laser on the top of the platform. From telemeter laser and IMU, are based the navigation and localization method presented in this thesis. The follow part of this chapter treat only a brief

introduction of telemeter and IMU sensors and how data are treated and displayed from middleware.

Fig. 13: WifiBot model.

Telemeter-laser. A telemeter laser is a two-dimensional (2-D) range finders scanner for measuring distances around the sensor. The WifiBot is equipped with a Hokuyo model UTM-30LX. This system is adapted for high moving speed robots thanks his longer range (30m and 270° scanner range) and fast response (25ms/scan).

Fig. 14: Hokuyo telemeter-laser UTM-30LX model.

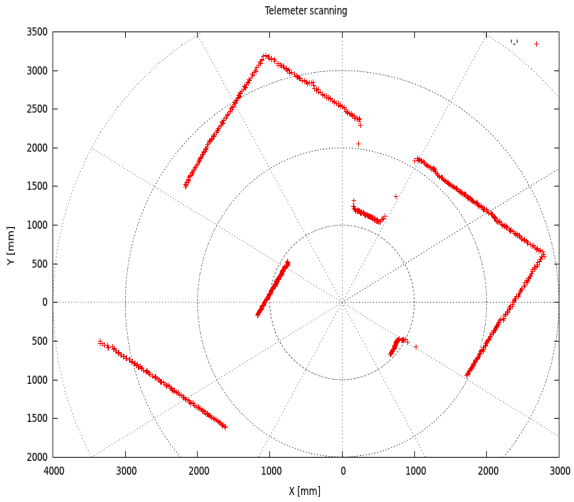

Fig. 15: Telemeter-laser acquisition.

This kind of sensor lends itself to become a suitable sensor for detect free space ahead of the vehicle, localization and map building due to their accuracy (0.1 to 10m: ±30mm, 10 to 30m: ±50mm). The angular resolution is 0,25°.

IMU package. An inertial measurement unit (IMU) is a device that estimate the relative position, velocity and acceleration of a vehicle in motion, utilizing measurement systems such as gyroscopes and accelerometers [12]. The IMU device equipped into the robot is the model VN-100 (Vectornav), see Fig. 16, six degree-of-freedom system estimate of the pose of the vehicle: position (*x,y,z*) and orientation (*yow, pitch, roll*).

Fig. 16: IMU (Vectornav VN-100).

This IMU uses three orthogonal gyroscopes and three orthogonal accelerometers. The gyroscope data ω is integrated to obtain an estimate of vehicle orientation θ. At the same time, three accelerometers are used to estimate the instantaneous vehicle acceleration α. This data is then transformed by the current orientation of the vehicle relative to the gravity, this last one should be extracted from the measure. As shown in Fig. 17, the resulting acceleration is then integrated to obtain the vehicle velocity and then integrated again to obtain the position.

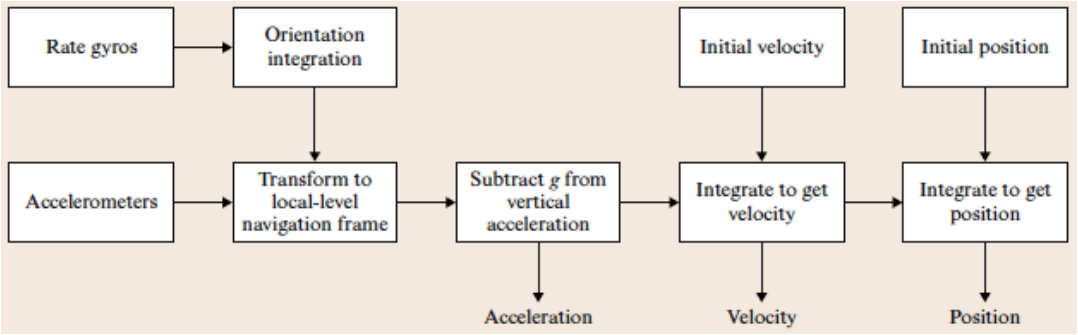

Fig. 17: IMU block diagram [12].

2 Robot control architecture

2.1 Introduction to architectures of control

In an autonomous robot mobile, behaviors are closely depending by the robot's effectiveness and robustness ability to carry out tasks in different conditions of ill-known environments. This leads to adopt a well-structured architecture of control. Ronald Arkin in [13] define a robotic architecture in the following way:

> *"Robotic architecture is the discipline devoted to the design of highly specific and individual robots from a collection of common software building blocks."*

The architecture consists of functional modules every one with a specific function, for example a function should covers robot localization, it represents environment based on sensor information, analyze the representation using knowledge thanks databases, plan actions and execute planned actions etc. An integrated architecture allows a robot mobile to plan its tasks, taking into account of temporal and domain constraints. The objectives are to perform actions and controlling their execution in real-time, in order to make reactive the programmed machine from possible events. Capacities to achieve tasks and to react to events are determined from robotic system organization. A robot's control structure should have the following properties: *programmability, autonomy and adaptability, reactivity, consistent behavior, robustness, extensibility* [14].

- *Programmability* is a property that describes the ability of a robot to achieve multiple tasks at an abstract level.
- *Autonomy and adaptability* intended as the ability to carry out robot actions and to refine or modify the task and its own behaviors according to the current goal.
- *Reactivity* because the robot taking into account of non planned events to execute its goals and takes care own safety.
- *Consistent behavior* by the fact that robot has to react to events consistently to objectives of it task.
- *Robustness*: the control architecture should be able to exploit the redundancy of the processing function.
- *Extensibility* in order for integration of new functions and definition of new tasks in easily way.

The control architectures should be classified in three categories in function by the way sensory data is processed and distributed through the system: hierarchical, reactive, and hybrid controls [15]. Each one control architecture consists by the relationship between robotics primitives (functions): sense, plan, and act.

2.1.a Hierarchical architecture

In a hierarchical control, communication and control occurs in a predictable and determined manner, flowing up and down. The typical hierarchical structure is shown in Fig 18 ([13]). This type of architecture is characterized by a clearly identifiable subdivision of functionality where the robot senses the world, plans the next action, and then acts. In the hierarchical control the ACT input is always the result of a PLAN function, and the input of this last one is always the direct output from SENSE function. When the action is chosen by PLAN, it will be execute. If during an action's execution, a new event occurs (like an mobile obstacle or dangerous situation), a new different action should be immediately elaborated. A new SENSE/PLAN/ACT cycle introduce a delay that should be generate a collision.

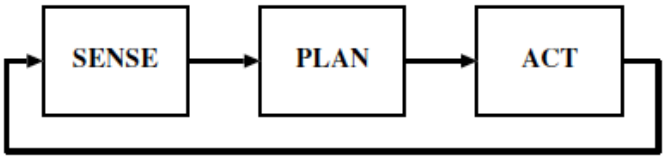

Fig. 18: Hierarchical architecture.

2.1.b Reactive architecture

The reactive control was a reaction from hierarchical control where the key aspects were design by Brooks [16]. In [13] R. Arkin define a reactive system:

"A reactive robotic system tightly couples perception to action without the use of intervening abstract representations or time history."

If in the first case the input to an ACT is an result of PLAN output, here the input to an ACT is always the direct result of a SENSE function. Coexist a direct link between sensors and effectors, in this direct way, a fast execution time is obtained. An reactive system react directly to the world as it is sensed. This architecture is presented as a whole of reactive behaviors, which operate simultaneously and controls the robot without the internal model. Generally these architectures are dedicated for multiple actions like displacement towards a goal, obstacle avoidance, aleatory displacement, following a wall etc. The priorities given to actions to be performed are fixed, this do not allow a good flexibility. But in the other hand it has the both advantages to be simply and easy to implement.

Fig. 19: Reactive control [13].

2.1.c Hybrid architecture

A control that adopt an intermediate solution between hierarchic and reactive control assume the characteristic of Hybrid control. The goal of this structure is to maintain the property of responsiveness, robustness, and flexibility of purely reactive system. But in the same time to provide a decision phase that permit a dynamic control system reconfiguration: providing a reactive control system based on available world knowledge.

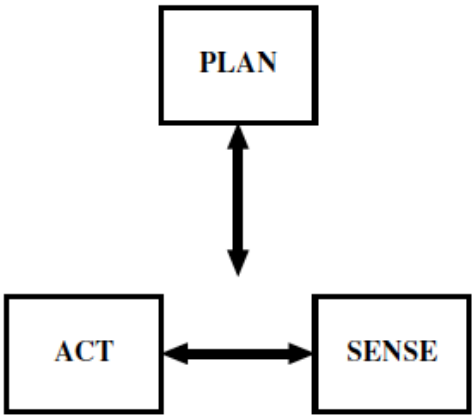

Fig. 20: Hybrid control [13].

2.2 *PRISME hierarchical control architecture*

Often, the control architectures are specific and designed for dedicated robot. They represent a closed architecture, and modifies by the user are not permitted. The need to modify and guaranty a level of reconfiguration of the robot system has led in last years the formalization of new robot's control architectures. A generic control has been conceive and then developed by PRISME laboratory [17] to aim formalize a modular robot control architecture. In particular for autonomous and tele-operated area. The objective is propose a framework architecture with property of integration with middleware and hardwares, assuming characteristic of adaptability in a opening areas of robotic. Without neglecting important aspect such as keep up to supply autonomy, reliable, and robustness. This architecture proposed by Mourioux, Novales, and Josserand resides a hierarchical controller and modularity concept at the basis. It's was used the concept originally developed by R. Brooks [16] and which appears in architectures such as “LAAS Architecture for Autonomous System” [14]. The

architecture is organized similarly to the Open System Interconnection communication system. Well knows like ISO/OSI model (defined by ISO: International Organization for Standardization) shown in figure 21.

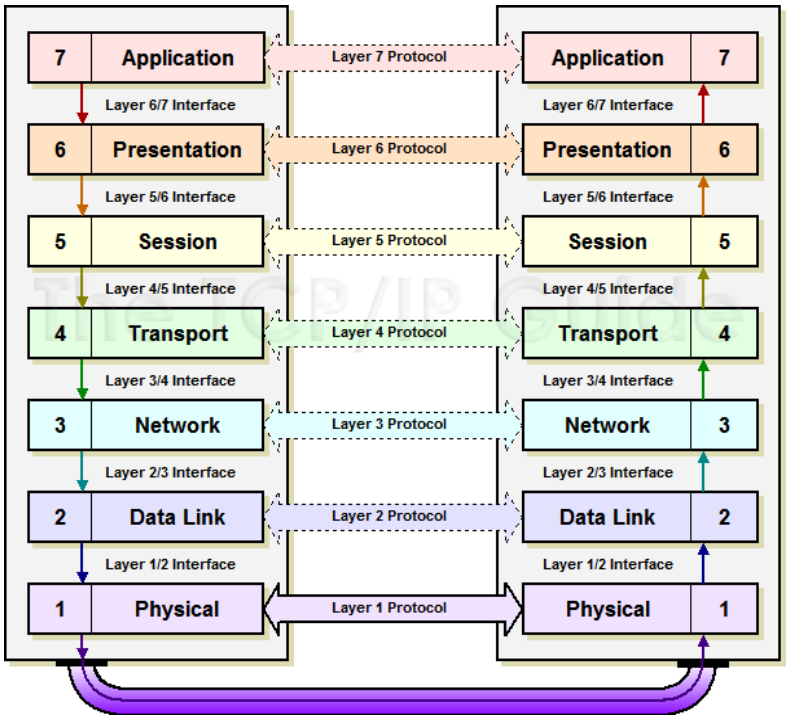

Fig. 21: ISO/OSI model.

For transpose the ISO/OSI model into a robot architecture, the operative part of the robot (A.M.S. Articulated Mechanical Structure) are taken into account. The structure is mainly composed by two part. One part where a upward flow conveys information of perception. The second one consist by a downward flow toward operative part will be used to the control features. At each level, two entities interact by means of a transmission protocol. In the communication architectures protocols enabled an entity in one host to interact with a corresponding entity at the same layer in another host. Data must necessarily pass through all layer surrounding. Because each entity interact directly only with the layer immediately beneath it.

However, unlike the communication model, in the presented robot's control architecture, data can either pass directly through entities that implement their functionality in the same architecture layer. Much more, data can either go through the different levels for processing. In this way, instead to have a single path of information, a multiple pathways is obtained [17]. So a multi feedback loop, each one associated with one of the architectural level, converging towards the robot machine. The lower-level loops associated to “reactive” part (nearly with the articulated mechanical structure, and associated to a low abstraction), are faster than loop of higher levels corresponding to “deliberative” part of the architecture.

The formalization of this robot architecture dedicated to autonomous or tele-operated scopes is presented in [17], trough a graphical representation.

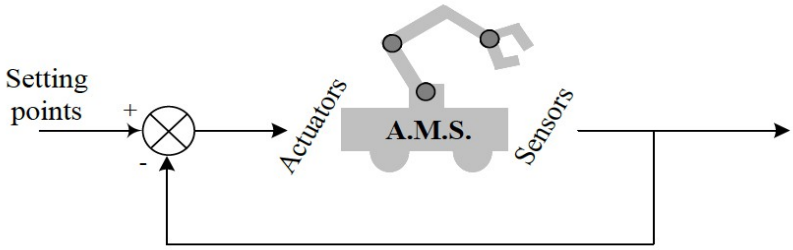

Fig. 22: Classical control loop drawing.

The control is focused by a transposition of a conventional control loop (Fig. 22), made-up by three elements: the articulated mechanical structure (A.M.S.), the *perception* part (sensors and their management), and the *decision* part which should be a PID controller for a low level.

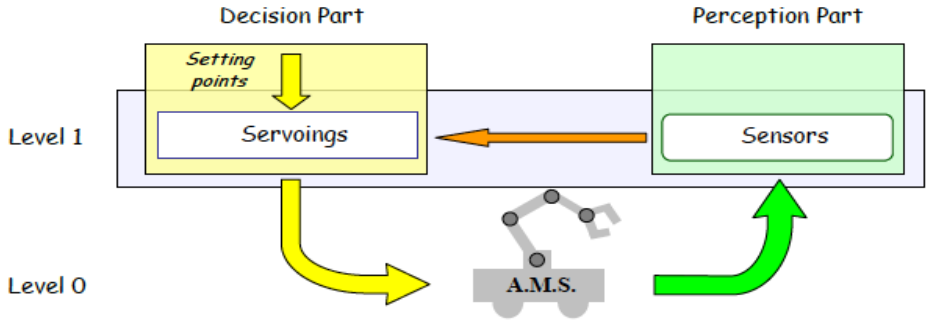

Fig. 23: Control loop transposition to architecture robot structure (Level 0 and 1).

While data flows organization, remains the same such as in a conventional control, the control loop is disposed in two levels (Fig. 23). In the Level 0 is located the system to control or rather the A.M.S. . The controls and actuators, and the sensors are located at the Level 1 respectively in decision and perception part.
To complete the architecture in order to give a more articulated structure (adapt in robot autonomous field), more levels are added for both *perception* and *decision* part. In this way a ordered control loops are obtained for each level, every one with a specific application field (i.e servoings, path planning, navigation etc.).
In the specific robotic autonomous mobile application, has been used a proposed 5 levels architecture how is shown in Fig. 24 . The levels are surmounted by the two complementary parts appointed first: the *perception* and *decision*, one ascending and one descending.
Each part is also cut transversely into the levels such as specified previously. In Fig. 24 extension for robot tele-operation appear also. In robot autonomous mobile only function of supervisor are permitted, so parallel loops are not considered.

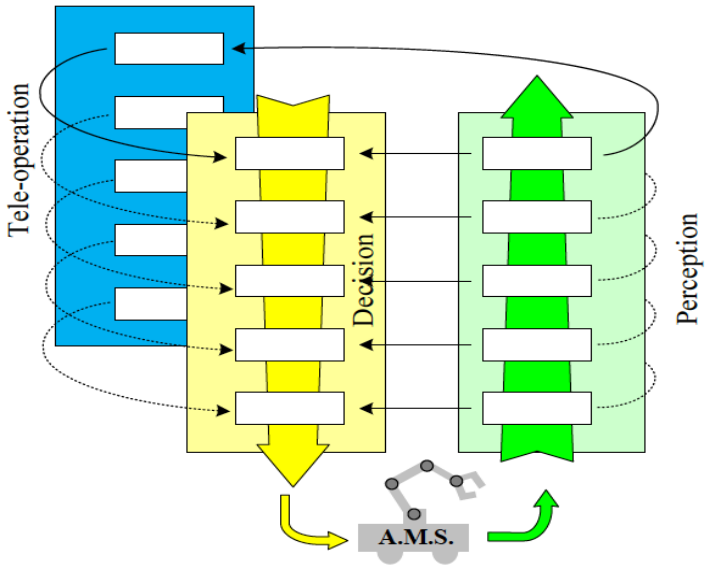

Fig. 24: 5 levels architecture design, including tele-operation extension.

In Fig. 25 is presented a particular of the architecture. In this way the "cluster" is defined as the whole of all modules of the same level. In the specific the Cluster "Sensors" in the perception part, aggregate all the sensor, conditioning modules and their management; and the Cluster "Servoing" of the decision part, includes the joint servo modules.

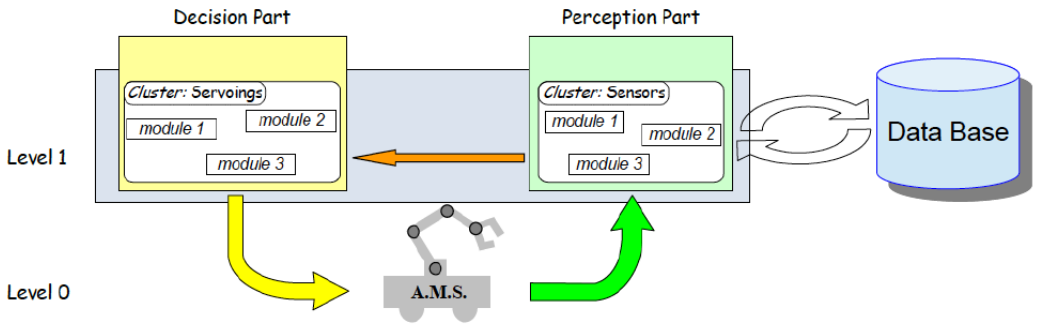

Fig. 25: Architecture particular level.

A set of rules are defined in order to obtain all the potentiality from the architecture; to quote contents from [17]:

- *Data rules*
  R1 – In perception part: data mount from level 1 to 5 in the sense of the complexity treatment. Any data of a level is available to all higher levels.
  R2 – In decision part: the data come down from level 5 to 1: all data of a level is available at all lower levels.
  R3 – In tele-operation part: the data does not flow between the levels of this part.
  R4 – Data can pass transversely from one level of the perception part to the

same level of the decision part.
R5 – Data can pass transversely from one level of perception part to the same level in the tele-operation part.
R6 – Data can pass transversely from one level of tele-operation part to the lower part of the decision part.

- *Rules for "clusters"*
  Clusters can contain multiple modules, each corresponding to a specific feature. Modules can discuss them and can use all available data at this level (in this cluster).

Architecture Levels:

This part aim allowing more detailed description of single parts of the architecture, performed for autonomous robot mobile like the CyCab or WifiBot models (Fig. 26).

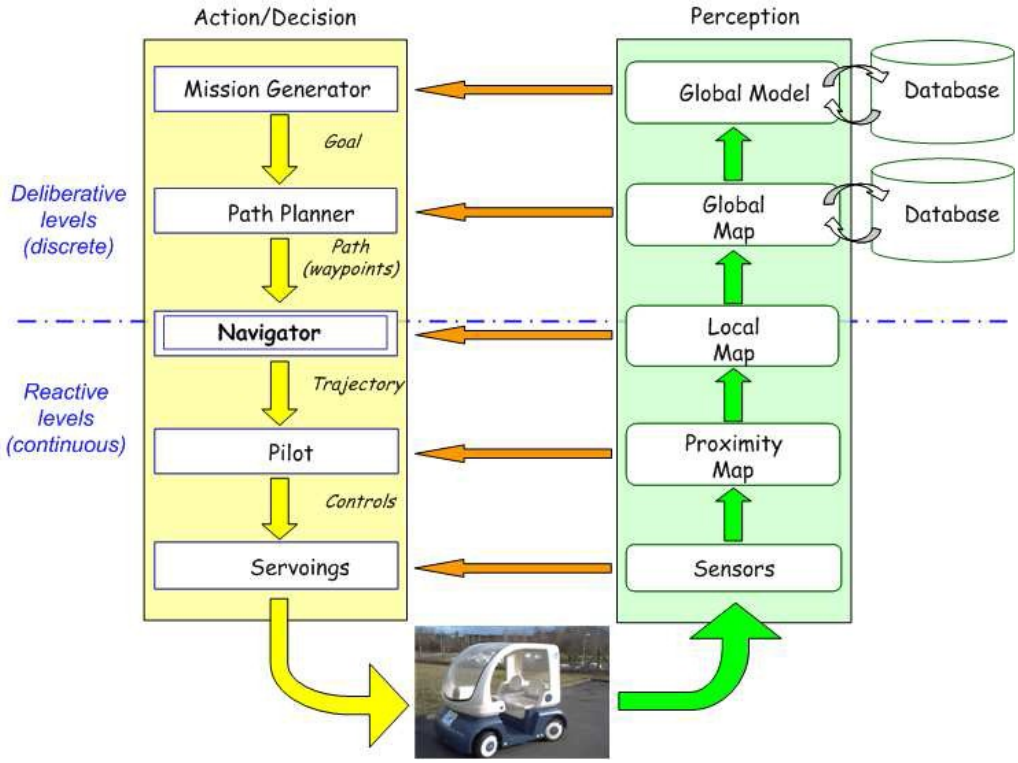

Fig. 26: PROTEUS project architecture structure.

How mentioned above the architecture is composed in five levels and in more functional blocks (by de fact the architecture assume a modular structure), each one with a specific task.
The Perception part is composed by several blocks such as *sensors*, *proximity map*, *local map*, *global map*, and *global model* (Fig. 26).
The perception blocks (Fig. 27) receive information from sensors. All or part of datas received, are transmitted to the related decision part, which placed at the same level. As well as the same datas could sent to the upper levels of the same perception part. Thus, level after level, raw data are enriched thanks successive treatments in function of confrontation with new and old information stored in memory. It improve the robustness of information, and to produce maps.

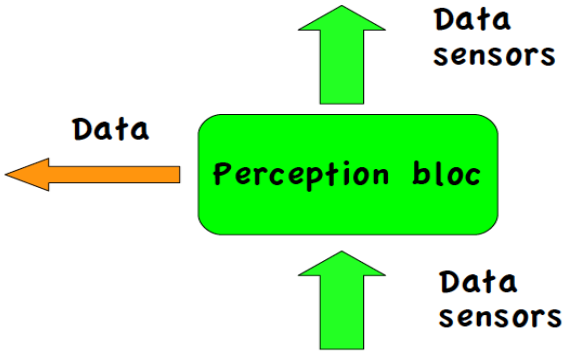

Fig. 27: Perception block.

The *action/decision* part incorporates the *mission generator*, p*ath* p*lanner*, n*avigator*, *pilot*, and *servoings*.
The *decision block* (Fig. 28) allow in output decisions relative an order received in input and in function of information received by the relative *perception block* at the same level.

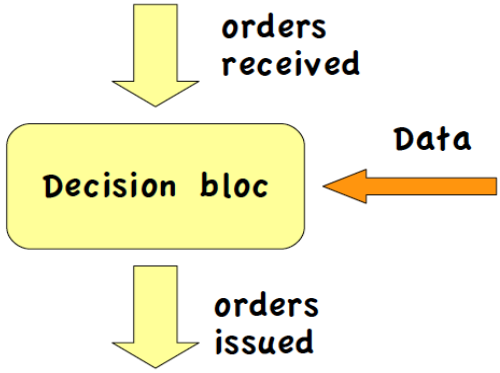

Fig. 28: Decision block.

Level 0. This level is the physical part of the robot: mechanical articulated structure, actuators to move the robot, and all that is part of the robot.

Level 1. Is the lower control loop of the architecture and is both short and fast. Is dedicated for example to treatment of joint PID feedback control loop. The part perception contains the sensors, and informations they deliver are used by the block of the decision part for the control of each servo motor.

Level 2. The second level is the driver. The *decision part* receive a path to be followed in according to the obstacles, from the upper block: the *navigator*. Or better, in addition, the *pilot* can also have the capacity to make some decisions as obstacle avoidance in case of a sudden obstacle and emergency. This module uses a method based on deformable virtual zones [18].
The elaborated informations are sent to the servos motors, for example in form of angular speed reference for each wheel (depending by the robot structure).
The *perception part* provide to the *pilot* raw data from sensor useful for obstacle detection and determining in this way free spaces.

Level 3. The Level three is the *navigator. I*s the mainly part treated in this thesis work. The *perception* part provide a model of the environment , generally in the form of map, named *local map.* This map is constantly updated with new informations from external sensors which are processed by the lower perception blocks. The *perception* part use the *local map* to generate a path for the robot, allowing it to move between obstacles in the best way as possible. And passing trough points of interest provided by the upper, the *planner*. The path generated takes also into account of the kinematic constrains of the robot.
The *navigator* symbolize the layer of the autonomous robot's architecture where the reactive levels mentioned so far, stand out from the deliberative levels. Or rather the reactive part by their nature handle information that need an immediate processing (continuous way). Differently, the proposed navigator consist in a predictive control, where the best path to follow is provided by the navigator relating a determined journey time of the robot. So the upper levels, including the navigator precisely, manage information with a certain deadline (discrete way).

Level 4. The level 4 is the *path planner block (scheduler).* Part of this decisional block is provide to *navigator* a series of preferred waypoints. It mission consist also to placing in the optimal way the waypoints to achieve the goal of the mission elaborated by the upper level. The perception part consist in a *global map*, but generally use maps present in the database, it will merge with the current information as *local map* in order to update the model of the environment.

Level 5. The 5th and last level is the *mission generator*. Its mission is provide to the *scheduler* the goal of the robot mission, such as a final destination, objectives to be carried out during the course, identify anything harmful, make field surveys, and so on.

Finally, this architecture can also integrates with teleoperation via a third part such as displayed if Fig. 24 . This part is divided into layers as the same way as the other blocks, which can involve tele-operated mode at any level. But in robot autonomous only in supervisory function mode.

3 The Pilot

3.1 Introduction

In autonomous robotic mobile, abilities to recognize and avoid obstacles are inevitably important tasks. The obstacle avoidance method proposed aims to recognize and avoid obstacles. The *pilot* is located into lower level of the *action*/*decision* part of robot's control architecture. It receives in input datas from navigation block such as commands related to a optimal trajectory, it receives also from perception part, info related to proximity map like infrared sensors and telemeter-laser. It delivers commands to *servoings block* to drive the robot into environment.
It covers problems of reactive behaviours of mobile manipulators evolving in dynamic and unknown environments [18]. The scope of pilot formulation is allow fast control laws. The pilot can be seen as a efficient low level algorithm for controlling motions to avoid unscheduled obstacles collisions. The mainly object of pilot is have the ability to react when unscheduled events occur, allowing property of artificial reflex actions to mobile robots. The DVZ (Deformable Virtual Zone) method is proposed to resolve the problem through the reflex action theory conceived by R. Zapata. This method is only reported from theoretical point of view and the environment acquisition method is explained in chapter 3.3 .

3.2 The DVZ (Deformable Virtual Zone)

The PROTEUS robot architecture uses a Deformable Virtual Zone principle as mentioned before for obstacle avoiding. It defines a safety zone around the vehicle, in which the presence of an obstacle induces an “intrusion of information”. The overall algorithm is combined with a guidance solution which path following control design relies on Lyapunov theory. The method embeds the path following requirements in a desired intrusion information function, which steers the vehicle to the desired path while the DVZ is virtually keeping a minimal contact with the obstacle, implicitly bypassing it [19].

3.3 Intrusion algorithm evaluating

The DVZ principle, where a rigid body (the robot) evolving in an unknown environment, is supposed to be surrounded by a shape. It's geometry depends by the state of the robot. The main issue is to define a risk zone surrounding the robot as a DVZ depending on the robot/environment interaction, see Fig. 29. The system reaction drive the robot velocities, angular and linear, in function of deformation calculated. The risk zone considered is an elliptic shape in order to obtain a polar DVZ. The risk zone is expressed in function of linear velocities; in particular the geometry depends on the square of this one in according with the kinetic energy of the robot in movement.

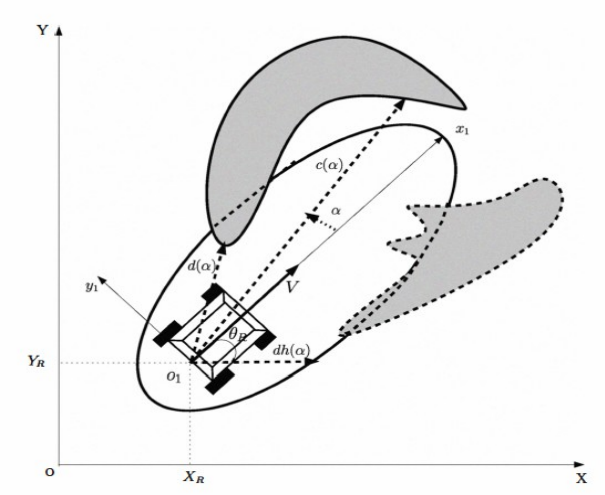

Fig. 29: Obstacle pose problem.

Fig. 30 shows a local map acquisition by telemeter-laser. Subsequent picture shows the elliptic deformable virtual zone in function on different robot speed: respectively 6 rad/s and 9 rad/s.

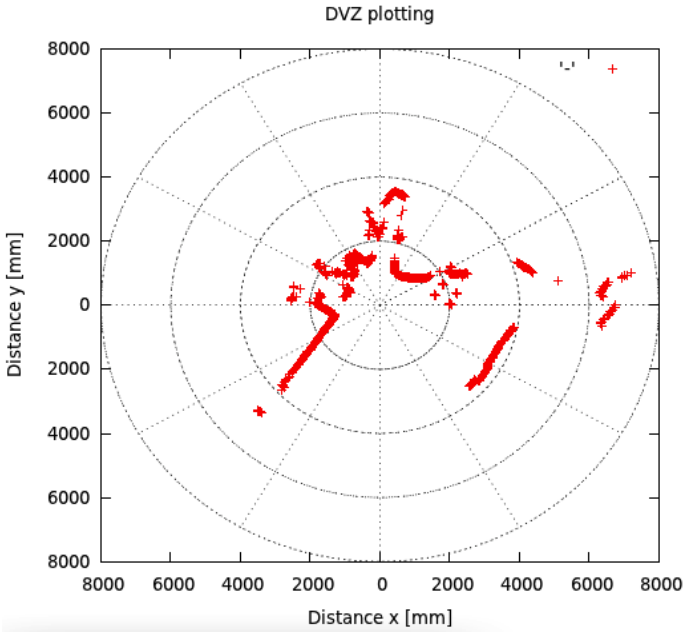

Fig. 30: Local map acquisition from telemeter-laser.

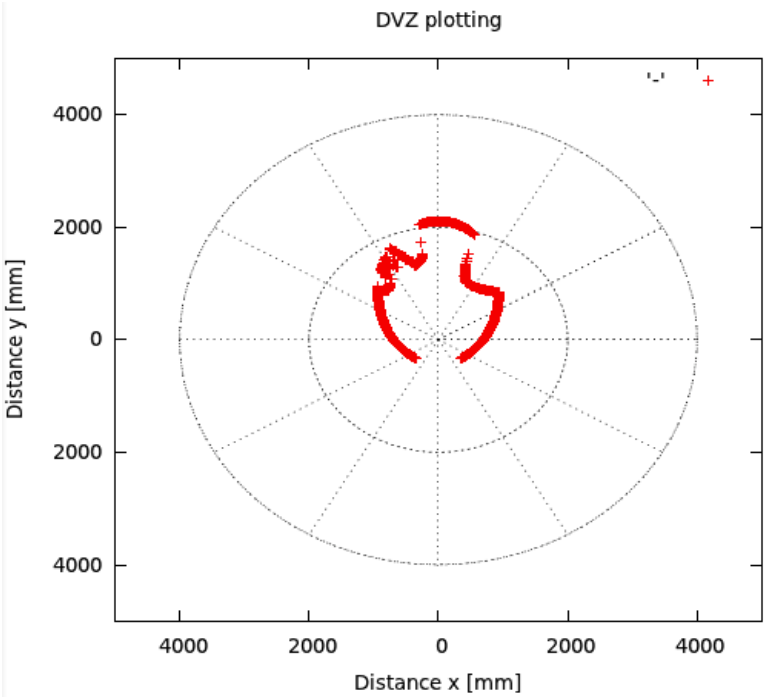

Fig. 31: DVZ @ 6 rad/s.

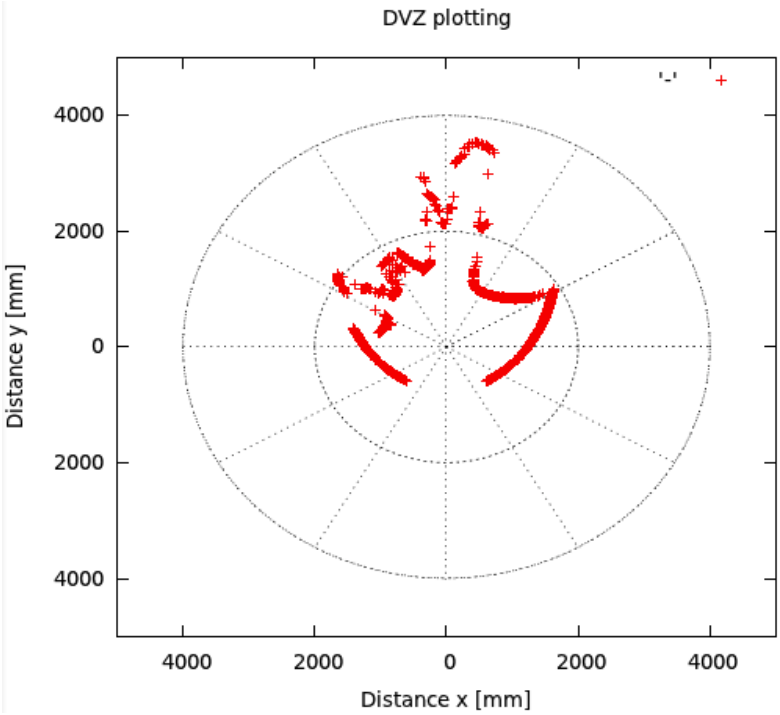

Fig. 32: DVZ @ 9 rad/s.

4 WifiBot mobile localization

4.1 Localization objective

In order to autonomously navigate and perform useful tasks, a mobile robot needs to know its exact position and orientation. Robot localization is thus a key problem in providing autonomous capabilities to a mobile robot.
Localization is a technique used by robots and autonomous vehicle for keeping track of their current location, necessary to the navigator for evaluating trajectories projected into environment from actual robot position. The goal of localization is to define the current absolute position (x,y) into cartesian space displacement. The relative orientation is taken into account by the gyroscope into IMU module.

4.2 Algorithm description

For determine the position of the robot on the map using the rangefinder, the goal is a minimization of difference between two types of values mentioned before: real values from actual robot position (x_{real},y_{real}), and the identified virtual values calculated from rangefinder given from a certain position of test (x_{test},y_{test}). To achieve this the localization algorithm is composed mainly in two part, the first will calculate values, rangefinder virtual position and the second is to minimize the difference between the values of rangefinders.

The principle of Localization algorithm is based on following steeps:

- Acquisition of a prefixed measures number displaced into telemeter-laser range.

$$\Xi_{measured} = \{d_1, d_2, \ldots, d_n\} \quad (1)$$

  Where d_i is the distance measured by telemeter-laser related to i-th position into telemeter range [-135°:135°] where 0° is signed by front direction of the robot. The i-th position should be chosen dynamically by an algorithm evaluating best condition of measure (for example in direction of fixes obstacle or laser forming an angle with the surface to measure nearest to a perpendicular angle).

- Virtual distances calculation of a virtual robot position from a internal environment model (e.g. from global map data base).

$$\Xi_{virtual} = \{v_1, v_2, \ldots, v_n\} \quad (2)$$

  Where v_i is the virtual distance measured from an algorithm respect a position given in input by the optimization algorithm explained in chapter X. The i-th index related to position respect robot orientation are the same angle of real distance measured from telemeter.

- Define a cost function Z in relation with $\Xi_{measured}$ and $\Xi_{virtual}$.

$$Z = \sum_{i=0}^{n} |d_i - v_i| \tag{3}$$

- Minimize the const function thanks the optimization algorithm. The minimum cost function correspond to robot position into environment.

4.2.a Gradient optimization algorithm

The method uses the negative of the gradient vector as a direction for minimization. The algorithm start from an initial trial point X_1 (for example a fixed initial position or the last computed position from step before) and iteratively move along the steepest descent directions until the optimum point is found.

The gradient of a function is a n-dimension vector defined as:

$$\underset{nx1}{\nabla f} = \begin{Bmatrix} \partial f / \partial x_1 \\ \vdots \\ \partial f / \partial x_n \end{Bmatrix} \tag{4}$$

The evaluation of the gradient (4) required the computation of partial derivatives. The function is differentiable at all the points, but the calculation of the components of the gradient, is either impractical. So has been used the forward finite-difference formula:

$$\left.\frac{\partial f}{\partial x_i}\right|_{X_m} \simeq \frac{f(X_m + \Delta_{X_i} u_i) - f(X_m)}{\Delta_{X_i}} \qquad i = 1,2,\ldots,n \tag{5}$$

Where u_i is a n-dimensional vector whose i-th component has a value of 1, and all other components have a value of zero. Δ_{Xi} is a scalar quantity chosen with some care which represent the increment.

For better results has been used the central finite difference formula to find the approximate partial derivative:

$$\left.\frac{\partial f}{\partial x_i}\right|_{X_m} \simeq \frac{f(X_m + \Delta_{X_i} u_i) - f(X_m - \Delta_{X_i} u_i)}{2 \cdot \Delta_{X_i}} \qquad i = 1,2,\ldots,n \tag{6}$$

The steepest descent method can be summarized by the following steps [20]:

1. Start with an arbitrary initial point X_1 (e.g. prefixed initial position if first iteration or detected position from last algorithm iteration). Set to i=1 the iteration number.
2. Find the search direction S_i defined as:

$$S_i = -\nabla f_i = -\nabla f_{(X_i)} \tag{7}$$

3. Determine the next point X_{i+1} from point X_i and step length λ :

$$X_{i+1} = X_i + \lambda_i \cdot S_i = X_i - \lambda_i \cdot \nabla f_i \tag{8}$$

4. Test the new point X_{i+1} for optimality. If this last one is optimum, stop the process. Otherwise go to step 5.
5. Set the new iteration number i=i+1 and go to step 2.

For better results was used the displacements given by odometry between two localizations to enhance the accuracy of the initialization of the algorithm: initialization position = previous computed position + displacements computed from odometry between two execution of localization process.

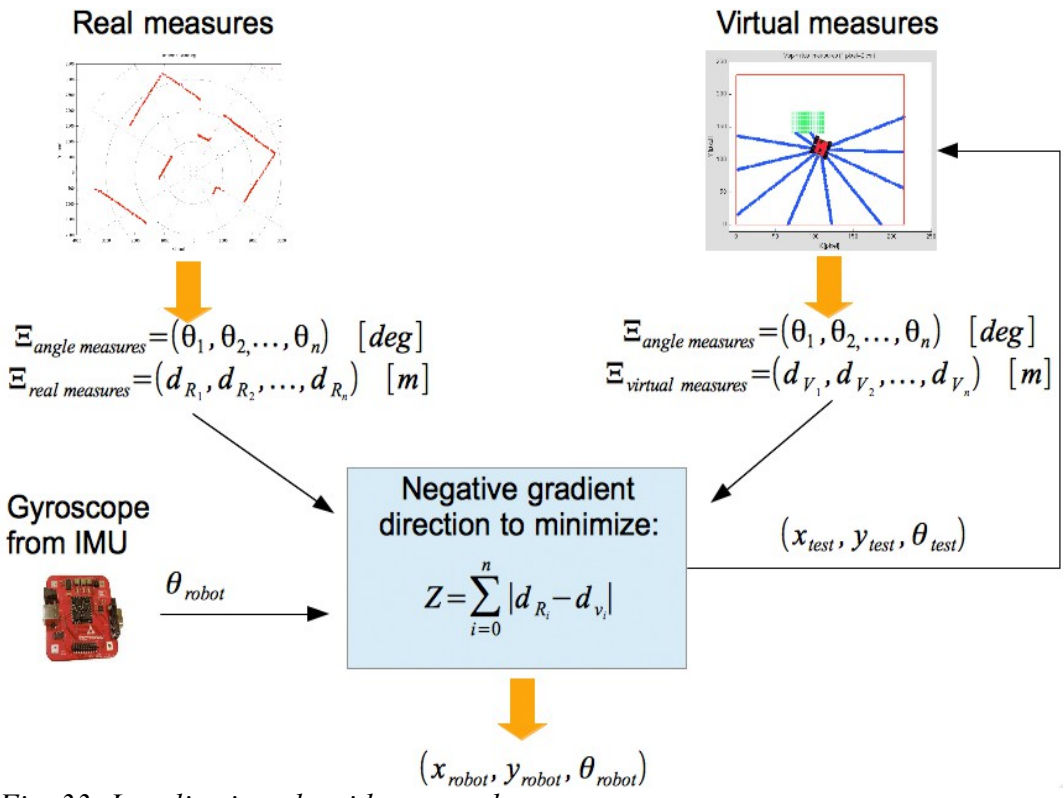

Fig. 33: Localization algorithm procedure.

4.2.b Algorithm results

Pictures 34,35,36 show from different point of view the representation of cost function into local map robot environment. The darkness blue color mash identify the minimum function value (local minimum) or better the robot coordinate displacement.

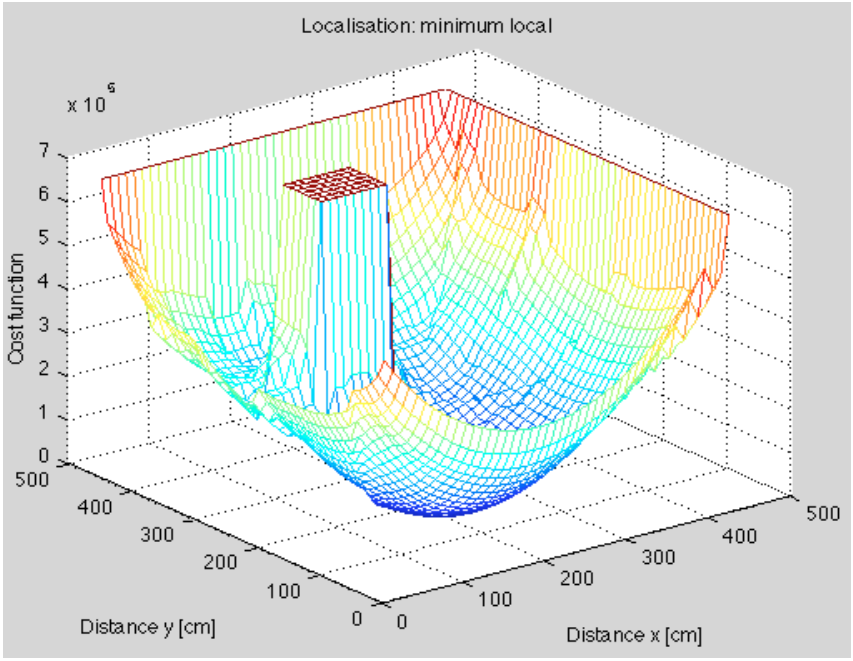

Fig. 34: Cost function plotting 1.

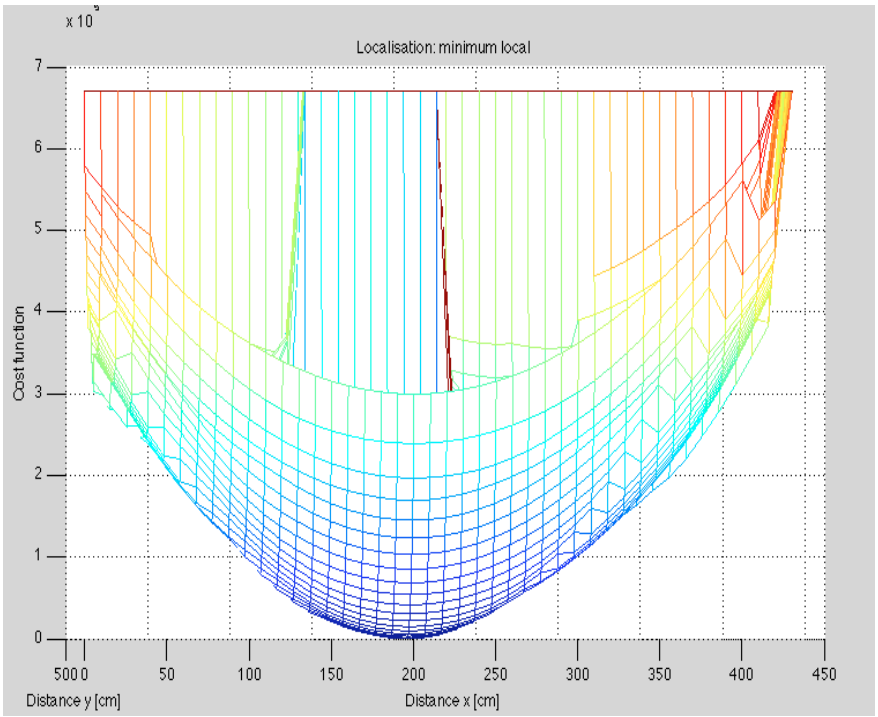

Fig. 35: Cost function plotting 2.

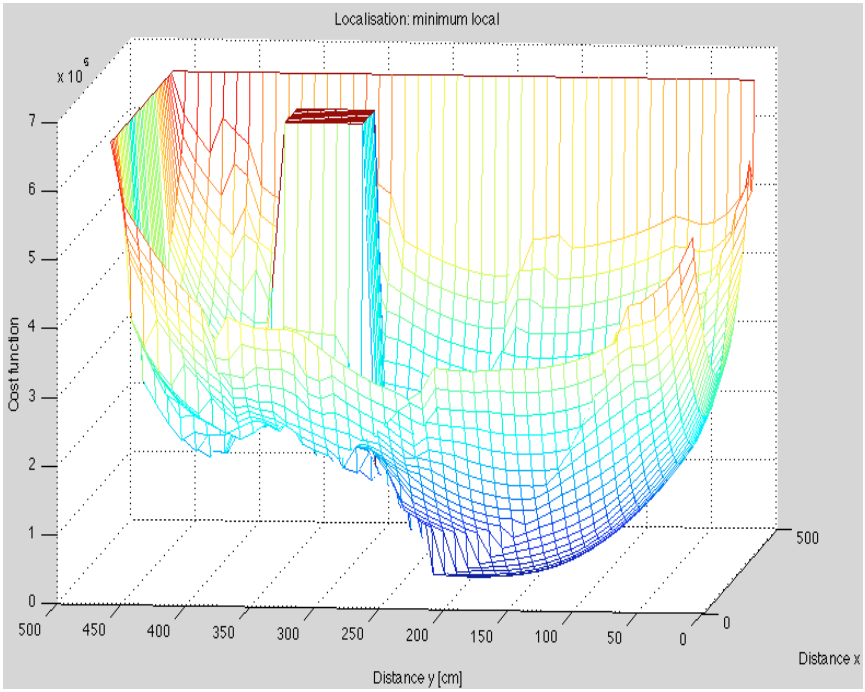

Fig. 36: Cost function plotting 3.

Pictures before shows that local minima problems can occur, potentially leading the localization algorithm to make errors. In order to prevent this to occur, it is important to initialize the localization algorithm as close as possible from the real robot position, and this leaded us to make the robot always begin its navigation at a known position (in order to initialize correctly the localization algorithm and keep it efficient iteratively once the robot begin to move). Practically, was measured localization accuracy inferior to 3%, which was judged sufficient enough for the navigation.
For to validate the accuracy between results from localization algorithm (red lines) and real robot position into the environment (blu lines) were made measures:

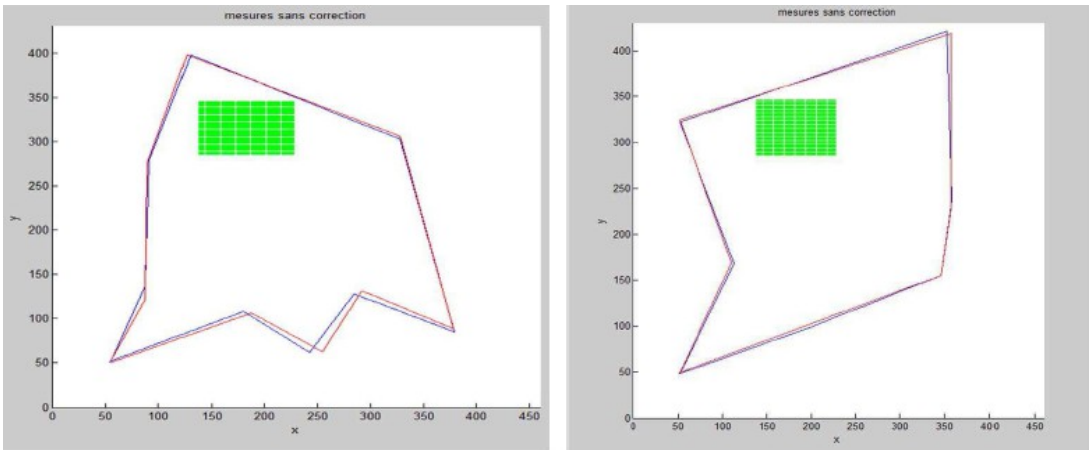

Fig. 37: Results measured from localization algorithm (red lines) between real robot positioning (blu lines).

Fig. 38 shows a simple representation of internal model map alias of the real environment of robot. For localization and navigation has been assumed a known environment. The architecture part which build up a map within a unknown

environment is not thesis argument. External rectangular shape (defined by continuous line) represent the border of arena. A rectangular obstacle is located inside. A function compute the distances from a trial position and orientation (x,y,θ_{IMU}) and defined angular given as arguments.

Expression (9) report the algorithm virtual measures from a set of defined angles in degree (10) related to the detection range of the telemeter-laser [-135°:135°]. These results was compared with real measures and considering an accuracy of 0.6% from telemeter-laser measures, has been obtained 3.6% accuracy results for final robot position.

$$\Xi_{virtual}=(57,83,221,225,295,246,233,279,247,217,239)\quad [cm] \tag{9}$$

$$\Xi_{angle}=(-135,-108,-81,-54,-27,0,27,54,81,108,135)\quad [deg] \tag{10}$$

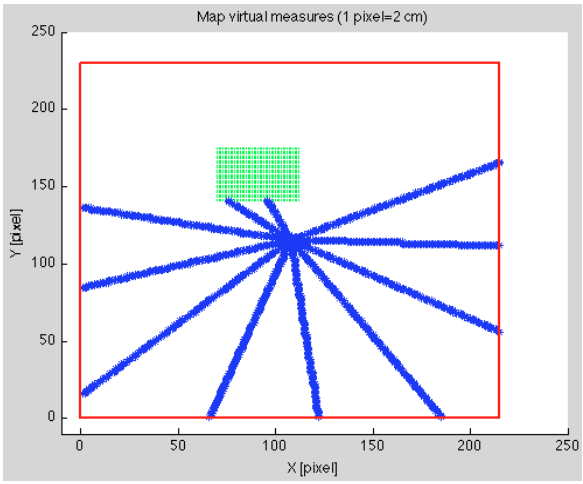

Fig. 38: Virtual distances computation.

Picture (39) shows a MatLab execution of localization algorithm. The green point identify the initial point of the robot. A second random point (red color) is chosen closeness and the minimum local direction is designed from segments connecting the two points. Picture (40) shows the cost function value at each optimization algorithm computation. A convergent solution has been reached.

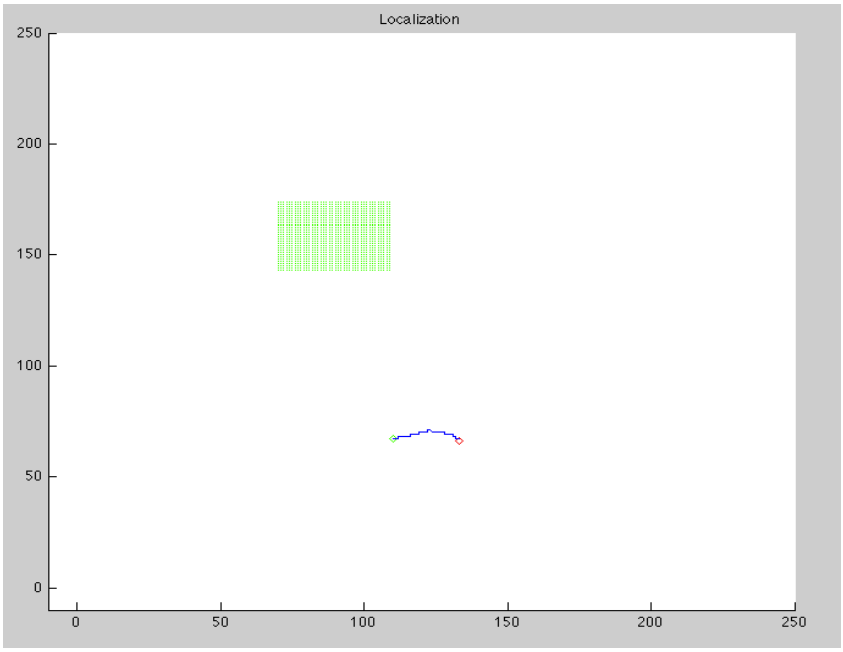

Fig. 39: MatLab localization algorithm evaluating

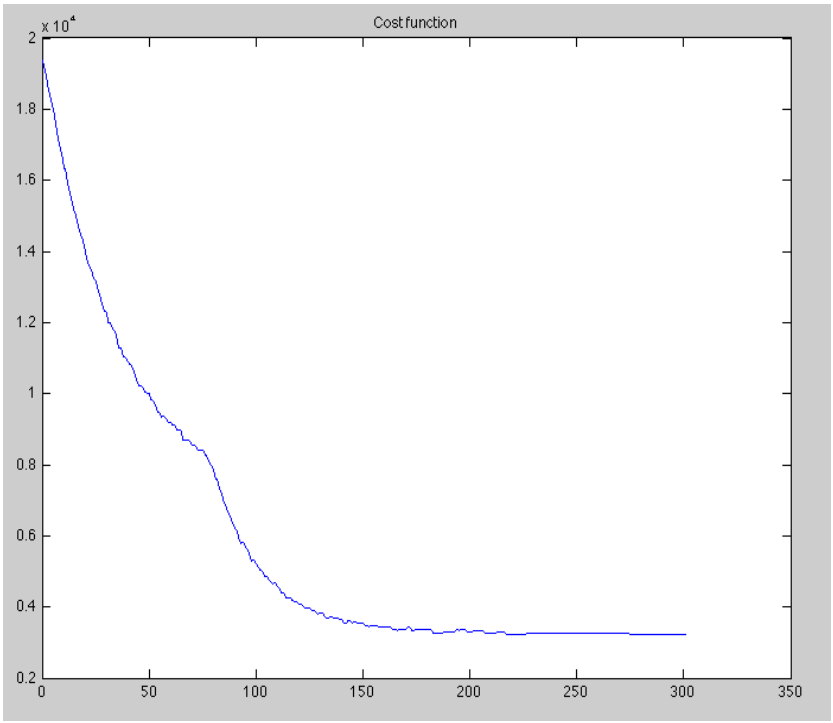

Fig. 40: MatLab cost function assumption.

5 Classification of navigation methods

5.1 *Introduction to navigation*

The ability of navigation for an autonomous robot mobile is one of the most important aspects. The navigation task is drive the robot in the environment doing evolution in the best possible way. This is possible determining it own position in the environment, and follow the best possible trajectory to achieve own tasks depending from the application field. Typical behaviors is for example planning a path towards some goal location (or rather, what called path planning), adapting own velocity to different circumstances such as the traffic road or more in general integrate the constrains in order to ensure that the robot is able to perform physically the trajectories requested. The robot must be free also to takes decisions in real-time on the basis of its current perceptions to carry out displacing in the environment. The *navigator* is a key point for an autonomous robot mobile, it mades the connection between the movement capabilities of the robot and the environment.

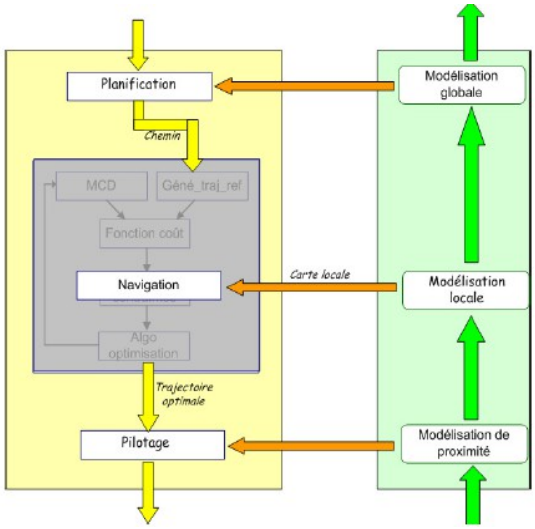

Fig. 41: Navigation block into robot's control architecture.

In the robot's control architecture, the *navigator* receive multi inputs. From *path planner block* receives a path, which can be in different forms more or less complete: a series of waypoints coordinates, may be contain additional information such as the orientation angle, speed of the robot or also auxiliary information such as a waiting time to stop in that point.
Several methods of navigation have been presented in the literature, where has been introduced many different choices in the design process. As part of the navigation of robot autonomous mobile, three main approach can be defined. Methods without explicit path (such as Artificial Potential Fields (APFs), Neural Networks, and Fuzzy logic); and methods of "trajectory tracking". This last one allows to the robot "the best possible" commands to follow a given reference path knowing kinematic constrains. In

this method a further division leads to consider methods of navigation based on the robot invert kinematic model (IKM) and method using direct kinematic model (DKM), described in chapter 5.4 .

5.2 *Classical methods by force filed-based*

The first methods consists in force field-based models. Obstacles can be considered such as repellents barriers and the robot attractive for itself so that it seeks to get closer. Planning or anticipate the trajectory of the robot is not necessary within these methods because robot's displacement is conditioned instant by instant from collected information on it local environment.

5.2.a APFs (Artificial Potential Fields)

Artificial Potential Fields. APFs is a first type of approach to solve the problem of robot navigation. The obstacles generate a repulsive potential field for robot and intensity is inversely proportional to the distance to the obstacles. The field is minimal to the point that the robot must reach, the idea is to move the robot in the gradient direction of the strongest negative potential.
APFs often represented a good solution to achieve a fast and reactive response to a dynamically changing environment, it has also the advantage to be simple to implement and not required high computational capacity. However, it has been widely demonstrated that they suffer from unavoidable drawbacks [21]. In particular, since the law of motion of the robot is basically determined by descending the gradient of the potential field, it is very likely for the robot to get trapped into a local minimum. And only invoking for example a planner, the problem can be solved. Yet more harmful, the presence of movings obstacles and sensor noise, significant deviations from the original path can lead to a deadlock configuration from which it is harder to escape [22].

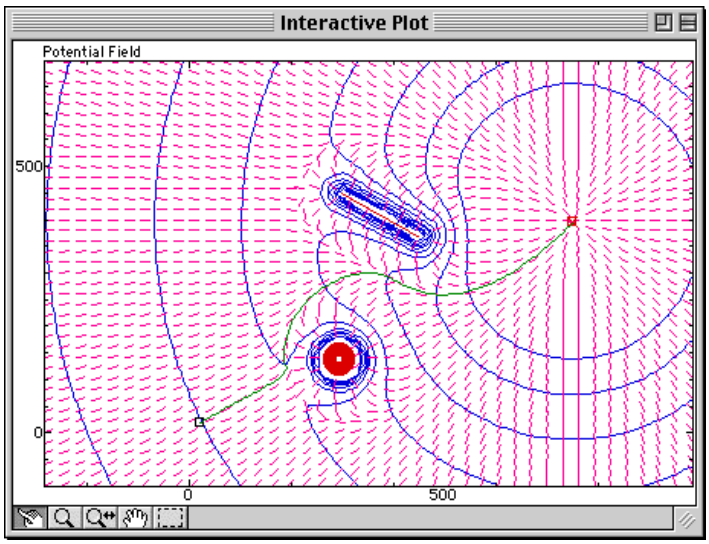

Fig. 42: Potential Field method.

5.2.b ANN (Artificial Neural Network)

Artificial Neural Network (ANN). This method is a biologically inspired techniques, based on nonlinear mapping functions of several variables. A neuron is an electrically excitable cell that processes and transmits information through electrical and chemical signals. A chemical signal occurs via a synapse, a specialized connection with other cells [23]. In Artificial Neural Networks the design principle is the many-to-one mapping between inputs of the system (where each input is assigned a different weight called synapse weight); if the amount of weighted signals of the different inputs exceeds a threshold, the output takes a positive value (neuron fires). Such relation can be represented as:

$$y = f(\omega_0 u_0 + \omega_1 u_1 + \ldots + \omega_n u_n) = f(\sum_{i=0}^{n} \omega_i u_i) \tag{11}$$

where:
$\omega_0 u_i$: array of weighted input values.
y : desired output of the system.

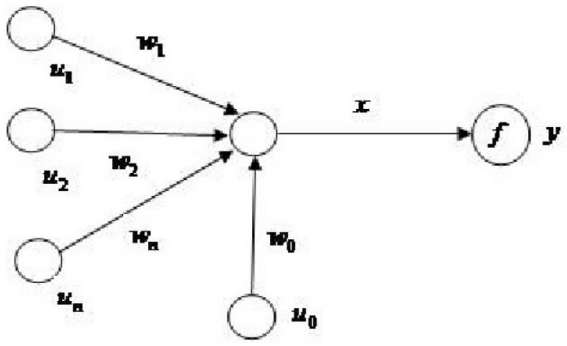

Fig. 43: Neural network schematic diagram.

Fig. 43 shows the schematic diagram of a simple neural network. Depending on the structure of the network, a neural network may comprise one or more input an only one output, it consists of different layer of neurons. The first layers is associated to detection, related in robotic to the information supplied by the sensors of the robot. The upper layers correspond to layers of interpretation, which allow to the network an interesting property, so called learning procedure. Is essentially an algorithm that makes it possible to find parameters (adjust the various synaptic weights in Fig. 43) such that to minimize the error; which means that the function matches given input-output values. The parameter are typically obtained recursively by giving both an input value and the desired output value to the function.

5.2.c Fuzzy Logic

Fuzzy Logic. Fuzzy Logic should be defined as a method for logical reasoning, (based on probabilistic logic) that is approximate rather than fixed and precise. The advantage is their simplicity, but differ from the human way reasoning. Objective is achieved by

introducing linguistic variables and associating them with membership functions. So, in contrast with traditional logic where a binary sets may have only two values: true or false; Fuzzy Logic may have a truth value which ranges between 0 and 1. In this way this method extend the concept of partial truth: the truth values varying between a range of completely true and completely false; managed by specific functions when linguistic variables are used. Quantify the concept of proximity of an obstacle, Fuzzy Logic involve concept of “close enough” or “too far” instead of binary definitions such as “obstacle” or “non obstacle”. A block diagram of a fuzzy PD controller is shown in Fig. 44. The principle of a controller based on fuzzy logic is divided in three phases. The first is a process called fuzzification where the measured values of the linguistic variables, the control error 'e' and the time derivative of the error 'ce', are converted into linguistic values. In essence, fuzzification process converts continuous values of the linguistic variables into a discrete collection of linguistic values such as Negative Big (NB), Positive Zero (PZ), etc.. The second one use a control strategy expressed in terms of a function, defined in terms of “if-then” rules, that maps linguistic variables to a linguistic values. A final phase is called “defuzzification”, the inferred output of each function, representing the linguistic value, is aggregated. The linguistic set representing the control that is then mapped into a real number in order to send command 'u' to the actuators.

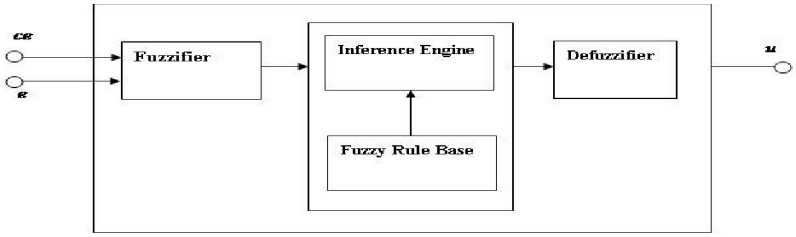

Fig. 44: Block diagram of the fuzzy PD controller [24].

Fuzzy Logic such as APFs are subject to local minimum problems, in robot navigation the robot can remain trapped in dead ends and other technique are needed to overcome this problem. An other problem in autonomous navigation is the need to cope with the large amount of uncertainty that is inherent of natural environments [24].

5.3 Method by IKM

A second type of approach to solve problems of navigation consist to consider the kinematic robot model. In particular generally require to determine an inverse kinematic model (IKM) or its invert dynamic model in such a way to calculate the commands to be sent to the robot (joint space), knowing the path that the robot has to follow (in the Cartesian space). While for most industrial and mobile robots the determination of the direct kinematic model is relatively simple and always leads to equations in closed form; inverse kinematic problem is more complex solution of the previous. Not always the equations can be written in closed form, and sometimes the problem is reduced to solve non-linear equations in several unknowns that many possible solutions.
The determination of this model is often the focal points of navigation method, in fact the goal is to get the best possible convergences of the robot to the reference trajectory.

5.3.a Non-holonomic systems

A method that ensures the robot to follow an accurate path is through his inverse kinematic model. Or rather this model allows to calculate directly the commands to be sent to the robot (joint space) knowing the path reference (in Cartesian space). Determination of such model is often the focal point of some type of method which goal is to get the best possible convergence of the robot to the reference trajectory.

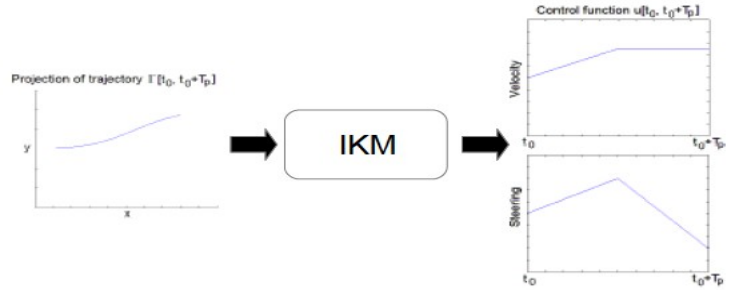

Fig. 45: IKM commands generation.

The problem occurs in mobile robots by the fact that almost are non-holonomic. This mean that an analytic model cannot be defined and in some situations is impossible to generate a trajectory following a desired path.
Non-holonomic vehicles are systems for which the number of controllable states is larger than the number of d.o.f. (degree of freedom). A non-holonomic system mean that some states cannot be instantaneously controlled, but could be controlled via the execution of maneuvers. A typical example should be illustrated by the classical car parking problem. This type of vehicle cannot move instantaneously sideways, force the conductor to do a series of maneuvers in function by the circumstances and their capacity.
Generally, all wheeled mechanical systems or provided by rolling parts, are non-holonomic systems if associated with the wheels property of "rolling without slipping".
Some approaches resolve this problem by using flat outputs [25] or transverse functions [26].

5.3.b Function plat

Flat outputs. The flatness property is useful for both the analysis of the controller synthesis and for non-linear dynamical system. It is particularly advantageous in the context of tele-operation where the path is provided by the user, or for solving trajectory planning problems and asymptotical setpoints following control of a robots fleet where trajectory is determined by the leader. The system theory of flatness was introduced in 1995 by M. Fliess [25] that extends the notion of Controllability from linear system to non-linear dynamical systems introducing a special type of feedback called endogenous.
A non linear system:

$$\dot{x}(t) = f(x(t), u(t)) \qquad x \in \mathbb{R}^n, \quad u \in \mathbb{R}^m \tag{12}$$

where:

$$f(0,0) = 0$$

$$rank\left(\frac{\partial f(0,0)}{\partial u}\right) = m$$

has the property of differential flatness if there is a set of measurable variables y (the outputs flat), such that the state x and the input u can be expressed as real-analytic functions of the components of $y=(y_1, \ldots, y_m)$ and of a finite number of their derivates:

$$\begin{aligned} x &= A(y, \dot{y}, \ldots, y^{(\alpha)}) \\ u &= B(y, \dot{y}, \ldots, y^{(\beta)}) \end{aligned} \tag{13}$$

The variables state of the robot (linear and angular speed) are expressed in terms of flat outputs (x and y position and their derivates parameterized by time). The problem became to reaches the desired state respecting the time of flat output and according with the constraints set.
The fact that the system is flat can prove the convergence of trajectories to the desired trajectory.
Advantage of writing the system in this form, is that it allows to prove the convergence of the system to desired path. The downside is that there is no systematic method for determining the flatness of a system.

5.3.c Transverse functions

The Transverse Function approach [26] is a method developed for treating the problem of control laws stabilization of reference trajectories. This method found particular applications in holonomic systems. For non-holonomic systems, several difficulties occur: from a well known result in nonlinear control theory, Brockett's theorem, shows that for non-holonomic vehicles, asymptotic stabilization of fixed points is not possible with smooth feedback laws. The linearized system is not controllable in proximity of this points. The objective of the transverse function approach provide feedback laws that achieve practical stabilization of any reference trajectory, in particular asymptotically converges to an arbitrary small neighborhood of reference trajectory. The transverse functions approach offer a practice solution to asymptotic problem of a trajectory reference [27] [28] based on additional frequency variables to control input. These variables relies on the existence of a function f satisfying transversality conditions. These functions allows a set of benchmark parter to the vehicle in a close vicinity that are slave to the reference path [27].
These two approaches that can solve problems arising from using IKM are penalized by

- high computation cost.
- determination of a specific robot's model, which is not trivial and non-systematic.
- limitation is the formalization of inequality constraints (joint limits, obstacles, etc.) inside IK.

5.3.d Visual method

Visual method for autonomous navigation. Other methods for monitoring trajectories are based on the recognition images along the way taken by a camera or a telemeter laser. The idea is to drive the robot during a manual recording phase along the desired trajectory. Once the database is created, the next phase of autonomous trajectory tracking can be achieved comparing at each sampling instant the image taking by the robot with the correspondent on the database. By the correlations between the two images the robot commands are calculated to lead it following the trajectory reference [29]. This method has large application in transportation systems in indoor and urban areas. For this last one an efficient visual memory management and small computational cost are required to ensure real-time navigation specially in large-scale outdoor situation [30].

5.4 Method by DKM (Direct Kinematic Model)

An other alternative family of navigation is introduced in this part and with more details will be treated in the next chapter. This method takes advantages of the fact that it is based on an element always defined in the robot mobile: the Direct Kinematic Model (DKM). This last one for almost all robot mobile assumes a simple form and also is easy to determine analytically. These methods consist to generate a trajectory based on the direct kinematic model. Afterwards a criterion will be applied to choose the one that results in the expected behavior.

Appendix A refers to DKM formalization of a differential wheeled robot such as the WifiBot model. Considering a control function $u[t_0, t_0+T_p]$, where the time Tp is a prediction horizon time, injecting this function into the kinematic model, the evolutions of the robot in the cartesian space $\dot{X}=(\dot{x}, \dot{y}, \dot{\theta})^T$ is obtained.

By integration of this evolution in the prediction time, the trajectory $\Gamma[t_0, t_0+T_p]$ corresponding the control function $u[t_0, t_0+T_p]$ is generated from robot's initial position $X(t_0)=(x(t_0), y(t_0), \theta(t_0))$. See Fig. 46.

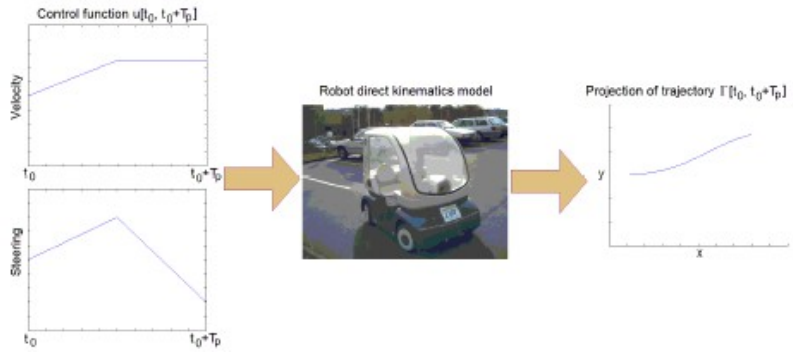

Fig. 46: Trajectory generation based on DKM.

6 Navigation by DKM

6.1 Navigation by DKM (Direct Kinematic Model)

This chapter presents methods of navigation based on direct kinematic model of the robot. Technique particularly suitable for robot mobile with strong conceptual constraints such as non-holonomic and low degree of mobility. All trajectories from the direct model assure a feasible path of the robot by the fact that cinematic constrains are taken into account and also these trajectories can be projected into a representation of the environment of the robot. In this way it is possible to select the optimal path according to defined criteria, which allows the robot to a given position avoiding obstacles. The idea is looking which movements the robot is able to perform choosing the most suitable at the present situation knowing the current configuration (position and orientation), current kinematic state, and environment. So several parameters must be integrated to make sure that navigator allowing the ability to predict a trajectory achieved by the robot in a given horizon time.

6.2 A classical "Model Generation"

A method "Motion Generation" has been developed by Bonnafous [31] that determines safe motions for an articulated rover on rough terrains. Uses the principle of projection trajectory achievable by the robot in a pattern of the local environment, and finally thanks a risk criterion (e.g. related to obstacles collision), associated to each path generated. The path performing the best result according with assigned mission, is chosen. This method for streamline the computational work by the fact it use a large set of trajectories; these are predefined and stored in a database. So this method does not use the kinematic model directly bringing a limitation to this method: adaptability to all type of robots is not straightforward.
A robot LAMA (LAAS) application of this methods is showed in Fig. 47, where the set of trajectories in the form of arcs and circle are projected into the middle space.

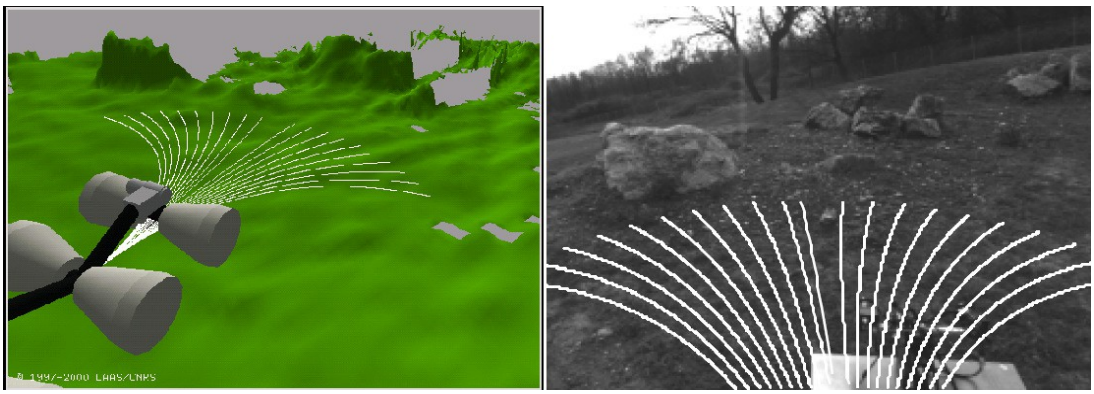

Fig. 47: Motion Generation method apply to LAMA robot (LAAS).

6.3 *Escape line method*

The “escape lines” method, developed by C. Novales [32], is a improvement of the precedent method. This one differs because use directly at run time the robot kinematic model to generate a set of trajectories considering the kinematic robot constrains and also the time horizon. The final trajectory selected is one that is more in the reference.

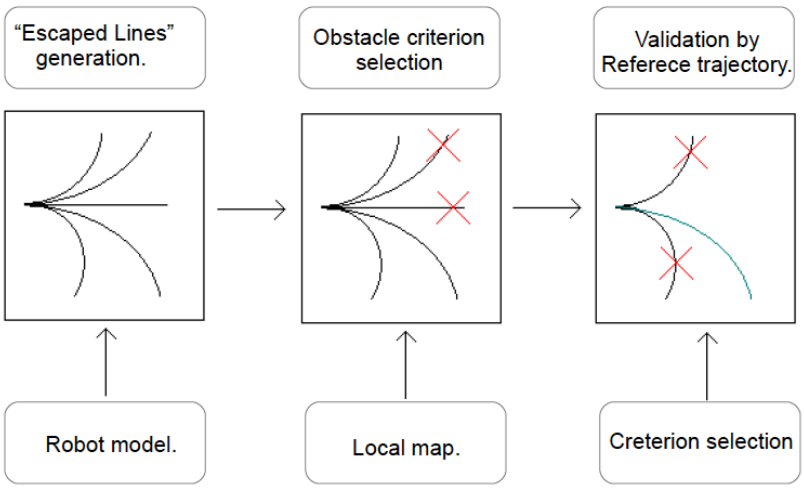

Fig. 48: Escape Lines principle.

This method is composed by different stages (see also Fig. 48):
Step 1: Trajectory generating. The first part of the method generates trajectories from commands eligible by the kinematic model of the robot (i.e. its possibilities of movement), respecting the kinematic constrains, actuators saturation constraints (e.g. maximum speed), dynamics constrains (e.g. acceleration and deceleration), and the temp horizon.

Step 2: Elimination of non-free “escape lines”. Compare all trajectory in the out space with the map of the obstacles known the relative distance from that and the current position of the robot. So the objective is to delete trajectories which projection into local map intersect or pass too close obstacles.

Step 3: Best trajectory selection. The last part choose a free trajectory that achieve a minimal cost by the proposed trajectory of reference. The related trajectory commands are then sent to the robot to achieve the desired movements.

Advantage of this method derive by the use of direct kinematic model which always exist and is simple to determine. This method is easily adaptable to most type of robots mobile and navigation methods.
Disadvantage of this method reside by the fact that hardware limitations implies a limit of trajectory projection into the environment (Fig. 49). So the set of trajectories eligible by the robot is obtained by discretization of commands acceptable by the robot. A trade off is needed between the number of trajectories generated and the accuracy reference path. Although it tests different trajectories over a prediction horizon, it is still subject to local minima problems which may occur for example when the robot is in front of a U-

shaped obstacle. Avoiding minima local is possible providing to navigator well-posed way-points from the planner.

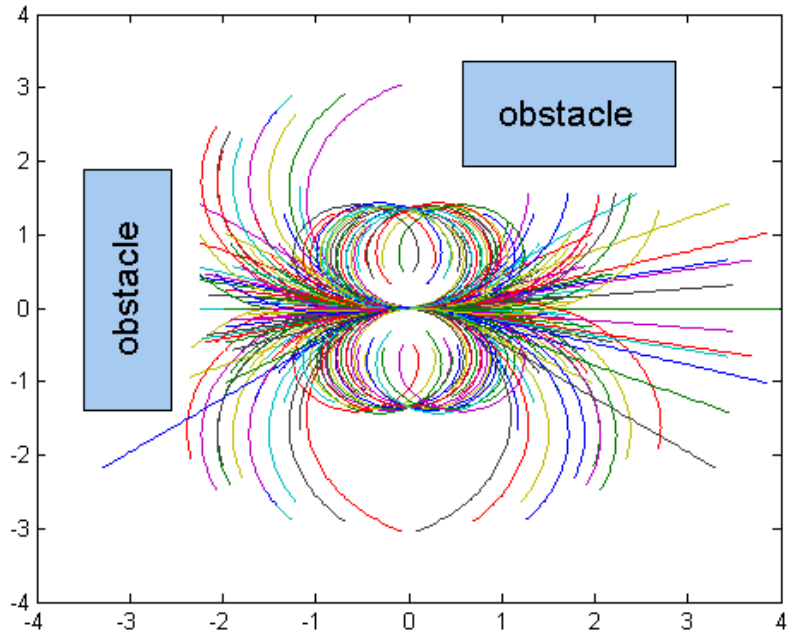

Fig. 49: Map free-trajectory projection.

This method has been applied to a CyCab robot mobile: a vehicle type with four steering wheels. Simulation of navigation results are shown in Fig. 50 where the trajectory in continuous line and the perpendicular segments indicate the sampling frequency of the method in a 40x40 m^2 area. The reds crosses are the waypoints.

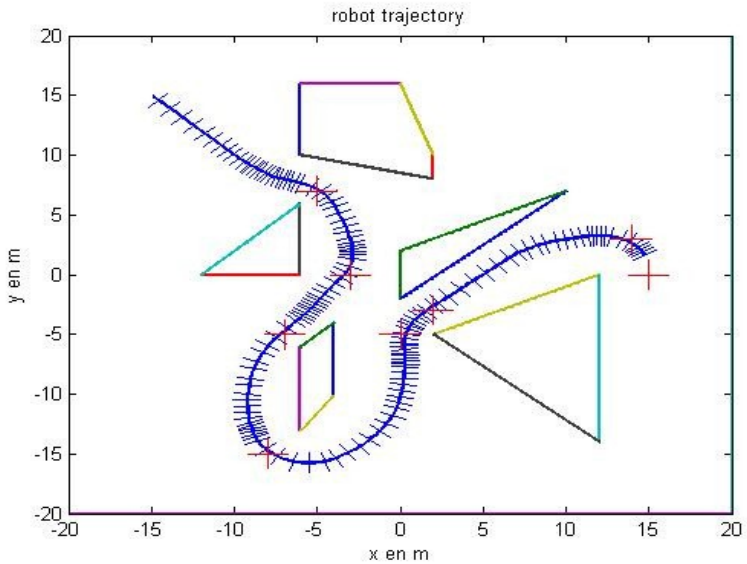

Fig. 50: "Escape lines" navigation method for a vehicle robot mobile (CyCab).

6.4 *Predictive navigation based on DKM*

A new navigation method for mobile robots navigation is proposed in this chapter. It aim to improve the precedent "escape lines" method providing a more degree of robustness to local minima problems, which is the strong point of this method. Proposed by Nicolas Morette (Lab. Prisme) is a direct kinematic model based, using a predictive control approach, see thesis: [33]. To allow a better exploitation of the robot kinematics performances, the problem to research the optimal trajectory is shifted in a continuous parameters space. An algorithm of optimization is used to achieve this scope bypassing obstacles and in particular avoid incurring in unwanted local minima. The model of the system is used to predict its evolutions over a time horizon after a trajectory criteria evaluating explained in the next paragraphs.

In the Chapter 6.2 and 6.3 have been presented navigation methods (by DKM) which characteristic is to perform in a discrete space: the number of trajectories generated is limited due to limitations previously described, thus the choice of the best trajectory cannot be optimal [34] and this hit in a uncertainty in the controlled system. In order to overcome the limitation previously mentioned, this new method of navigation contribute to propose a continuous formalism based on the successfully tested model predictive control (MPC). Predictive control is an advanced control method that uses a model control system to predict its behavior in the future. Then at each sampling time, the optimal sequence of commands calculated, is applied to the system. So the problem system to perform a optimal trajectory has to predict its changes over a given time period [T_P], and during the interval [t_0 , $t_0 + T_P$] the commands sent to the robot. The problem is set as an optimization problem under constraints where the error to minimize is between the reference path and the expected path of the system. The problem of navigation mobile can be addressed as a constrained optimization problem, considering the obstacles and the robot kinematic constraints to perform the minimization commands to give at the robot.
This method requires a reference trajectory, by the fact that only way-points list are provided by Path-Planner block (a way-points is a data set containing coordinate in the map, speed, and preferably direction that the robot has to achieve in that point).
The proposed method of Navigation is composed by several steps. Firstly a geometric curve from his position (given by the localization block) and intersecting way-points is generated. Without consider environment informations (e.g. obstacles), or other robot constraints which are considered in next steps.
At this point, the Navigator has all the tools to generate by commands input to the kinematic model a trajectory which is closer to the reference. The reference trajectory is obtained adapting the geometrical path to some robot constraints. For example is necessary to assure a reference trajectory distance according to the capable robot speed and prediction time. An optimization algorithm (Simulated Annealing Cap. 6.4.e) evaluating a minimization cost function trajectory; according with a number of constraints such as minimum/maximum speed (commands *u*), obstacles (by an integration of a local map of the environment) , prediction time (T_P), and the state of the robot by the fact permitting a real time problem solution. The optimization problem solution (commands *u:* left and right speed for example, related to a prediction time trajectory design) is computed periodically according to the sampling time (T_S).

6.4.a DKM trajectories

A trajectory is achievable from the robot if it respects the possible movements of the robot by the fact that always robots mobile are non-holonomic systems. Despite this, using the direct kinematic model and a possible robot speed value, is possible to predict a trajectory that the robot can perform in a given prediction time horizon.

The Reference curve ξ_{ref} . Computing the reference curve is one of firstly action carry out from the navigator. The objective is connecting the current state (absolute position into the map, orientation and velocity) of the robot $X_0=(x_{0,}\, y_{0,}\, \theta_{0,}\, v_0)^T$ between a desired state $X_d=(x_d\,, y_d\,, \theta_d\,, v_d)^T$ corresponding to the next waypoint to reach; then respecting position $(x_d\,, y_d)$, orientation θ_d , and linear speed v_d provided by the *path planner* block. The reference curve is a purely geometric curve, which is not related to the time, but may take into account certain geometric constraints related to the mechanical structure of the robot such as size and turning radius. To obtain a continuous ξ_{ref} curve is necessary maintain continuity with their derivatives. Typically polynomial curves are used as Bézier or Beta-splines. In the implementation has been used Bezier curve reference (from the French engineer Pierre Bézier) given by following form:

$$\begin{cases} x_{(t)}=(1-t)^3 x_1+3t(1-t)^2 x_2+3t^2(1-t)x_3+t^3 x_4 \\ y_{(t)}=(1-t)^3 y_1+3t(1-t)^2 y_2+3t^2(1-t)y_3+t^3 y_4 \end{cases} \quad t\in[0,1] \tag{14}$$

This type of curve is defined by a set of control points, in this case only four are needed: $P_1(x_{1,}\, y_1), P_2(x_{2,}\, y_2), P_3(x_{3,}\, y_3), P_4(x_{4,}\, y_4)$.

$$P_1:\begin{cases} x_1=x_0 \\ y_1=y_d \end{cases} \qquad P_4:\begin{cases} x_4=x_d \\ y_4=y_d \end{cases}$$

$$P_2:\begin{cases} x_2=x_1+\frac{1}{3}v_0\cos(\theta_0) \\ y_2=y_1+\frac{1}{3}v_0\sin(\theta_0) \end{cases} \qquad P_3:\begin{cases} x_3=x_4-\frac{1}{3}v_0\cos(\theta_d) \\ y_3=y_4-\frac{1}{3}v_0\sin(\theta_d) \end{cases} \tag{15}$$

Fig 51 shows a general bezier curve using cubic equations (14). The vectors $\overline{P_1P_2}$ and $\overline{P_3P_4}$ are tangent to the curve. The thus far found reference curve is fully geometric and independent from robot's kinematics.

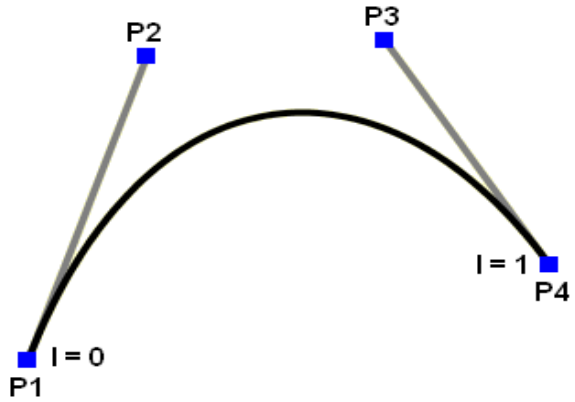

Fig. 51: Bezier curve and control points.

The reference trajectory. The curve thus far obtained, can not catch directly the role of reference trajectory because is not related to time. To determine a reference trajectory for the robot, is necessary to associate the reference robot speed to the reference curve ξ_{ref} . Multiplying the speed reference by the prediction horizon time T_P , the real distance reasonably by the robot that go along the reference curve is obtained. The choice of reference speed value (corresponding to a realistic moving speed of the robot) has to be based on the type of terrain, motor speed limitation and so on.
Knowing the distance from Bezier curve ξ_{ref} and the walkable real distance from the robot, introducing a temporal parameterization on the reference curve ξ_{ref} is now possible to obtain a reference trajectory.
Denoted $D \in \mathbb{R}$ the total length of ξ_{ref} , the robot is able to cover a real distance: $d_{real} = v_{max} * T_P$ during T_P . Is now possible to perform a time parametrization of ξ_{ref} . So two cases are considered: $d_{real} \leqslant D$ and $d_{real} > D$. In this chapter only formulation principle is presented, Cap. 7 treat practical implementation solution.

- $d_{real} \leqslant D$ This condition is verified when during T_P the robot cannot cover the totality of the proposed curve. In this case, the reference trajectory corresponds to a reduced part of the geometric curve. This situation is shown in Fig. 52. So the equation of the reference trajectory is computed with the respect to the initial and final constraints. The time parametrization is obtained considering the new coefficient into the Bezier equations :

$$t' = \frac{d_{real}}{D} * t \quad , \quad t \in [0, T_P]$$

$$\begin{cases} x_r(t') = (1-t')^3 x_1 + 3t'(1-t')^2 x_2 + 3t'^2(1-t') x_3 + t'^3 x_4 \\ y_r(t') = (1-t')^3 y_1 + 3t(1-t')^2 y_2 + 3t'^2(1-t') y_3 + t'^3 y_4 \end{cases} \tag{16}$$

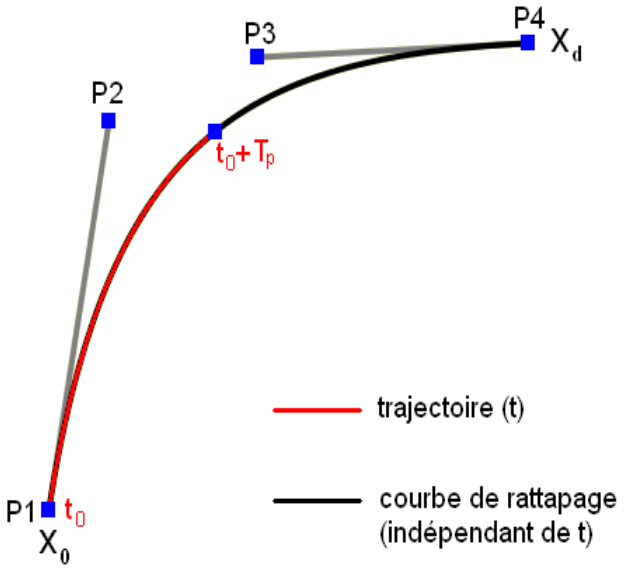

Fig. 52: Trajectory reference obtained by reducing reference curve.

- $d_{real} > D$ In this case the robot is able to reach the next waypoint during the time prediction horizon. So the next waypoint provided by the path planner list has to be considered and the next one Bezier calculated. The amount of new Bezier will be integrated to first one by distance difference. Fig. 53 shows clearly this procedure:

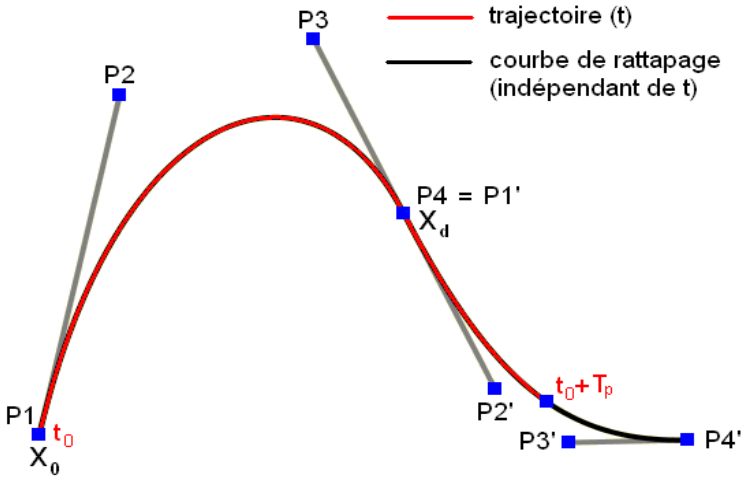

Fig. 53: Trajectory reference by Bezier curves composition.

A note about, the trajectory reference which does not takes into account of the local environment and robot's physics constraints. Adapt the reference speed could be made through information provided by the mission planner, from ground state, and dependency on congestion of the route [33].

6.4.b Local Map

The *local map* should be defined as a model of the environment, elaborated from *perception part* of robot's control architecture. It consists from all informations that sensors capture in a specific instant. The *local map* is constantly updated with new informations from external sensors which are processed by the lower perception blocks. The maximum range of the *local map* corresponding to the maximum measuring range of the sensor. An example is shown in Fig. 54 where WifiBot camera imaging is captured and overlying the related telemeter-laser measure in the environment. A local map should be also represented by a telemeter laser range where measures are described as vectors of distance and showed in a radial disposition l(see in Fig. 55).

Fig. 54: Camera imaging capture and telemeter laser measures.

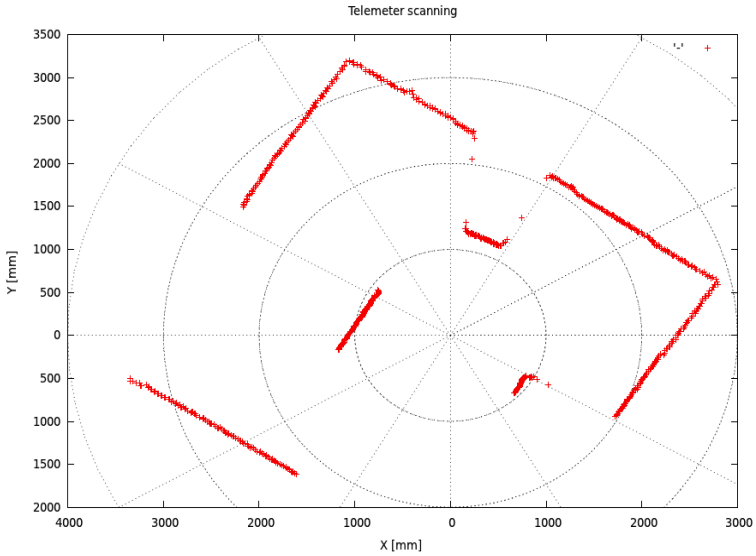

Fig. 55: Telemeter area scanning.

6.4.c Robot Commands

How mentioned in the top of this chapter, the navigation problem can be set as an optimization problem under constraints. The scope is to minimize a cost function Z. The variables on which plays to minimize this criterion are the commands. The *u* control function is a vector of *m* components, each one corresponding to one of the robot's actuators:

$$u_{(t)}=(\dot{q}_{1,}\dot{q}_{2,}\ldots,\dot{q}_{m})^{T} \quad ; \quad t\in[t_{0,}t_{0}+T_{P}] \tag{17}$$

where $\dot{q}_i$ is the articular velocity of the i_{th} actuator.
In the case of a differential wheeled robot such as WifiBot robot mobile, should be defined the angular left and right speed [rad/s] :

$$u_{(t)}=(\omega_{L(t)},\omega_{R(t)})^{T} \quad ;u\in\mathbb{R}^{[2x1]} \quad , \quad t\in[t_{0,}t_{0}+T_{P}] \tag{18}$$

and for a CyCab robot vehicle $u_{(t)}$ is defined as linear speed [rad/s] and steering component:

$$u_{(t)}=(v_{(t)},\xi_{(t)})^{T} \quad ;u\in\mathbb{R}^{[2x1]} \quad , \quad t\in[t_{0,}t_{0}+T_{P}] \tag{19}$$

Giving the commands into kinematic model input, the robot displacement vector in the Cartesian space is obtained. Or rather, the trajectory in the time horizon T_P is defined as: $\Gamma[t_{0,}t_{0}+T_{P}]$.
For the WifiBot a class of control family is defined by four parameters $P=(p_{1,}p_{2,}p_{3,}p_{4})$:

$$\begin{cases}\begin{cases}\omega_{L(t)}=\omega_{L(t_0)}+p_1 & t\in[t_{0,}t_{0}+\dfrac{T_P}{2}]\\ \omega_{R(t)}=\omega_{R(t_0)}+p_2\end{cases}\\ \begin{cases}\omega_{L(t)}=\omega_{L}(t_0)+p_3 & t\in[t_{0}+\dfrac{T_P}{2},t_{0}+T_P]\\ \omega_{R(t)}=\omega_{R}(t_0)+p_4\end{cases}\end{cases} \tag{20}$$

Before giving commands into to the kinematic model, constraints of acceleration and deceleration are considered into news commands, allowing by ramps generation a gradual velocity variation in time (where cceleration values depends from robot level battery). The motion laws considered are trapezoidal motion profiles:

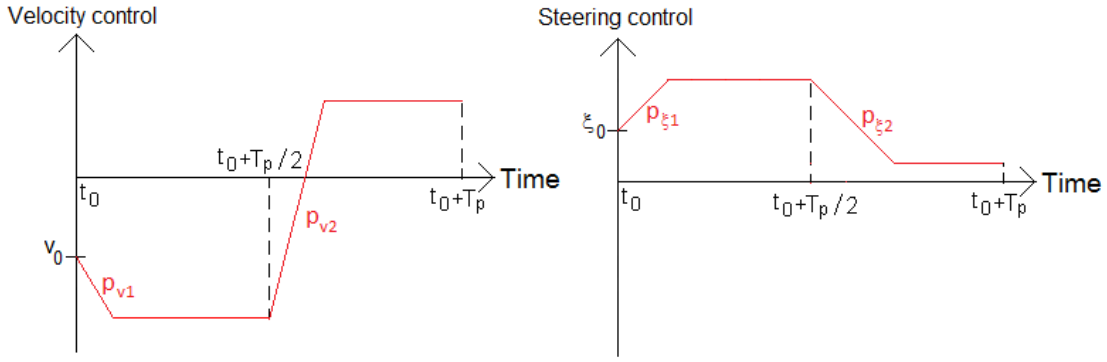

Fig. 56: CyCab motion law commands.

The use of four parameters are necessary to provide a by-pass trajectory generation. Only two parameters they can only generates a linear trajectory or arc in the same direction, as is showed in Fig. 49 . This implies a obstacle motion limitation during the horizon time prediction. To by-passing obstacles a veer is needed, translated into commands mean the use of four parameters. In this way four parameters permit to execute a robot change deviation during the time prediction, enabling the generation of by-pass trajectories.
The space of admissible solution is infinite dimensional, in fact an infinite number of admissible parameterized control function *u* can be defined on the prediction horizon time. Hence, performing a continuous variation on the four parameters, thanks the use of the robot kinematic model will be generated an infinite number of admissible trajectories. This makes the search for optimal solution very complex.

6.4.d Cost function structure

From Cap. 6.4.a follows that the non-holonomy constraints are implicit in the formulation of the problem. For against, kinematics constraints such as actuator saturations, must be clearly formulated by the way to achieve a realizable trajectory. So the cost function has also to take into account the environment: informations from local map database. So the optimal trajectory corresponds to a trajectory $\Gamma[t_0, t_0+T_P]$ which is the nearest to the reference trajectory, where robot and environment constraints are included. The returned cost function value is the means to evaluate the trajectory into admissible space of solutions. The cost function Z is structured in three part how shown in Fig. 57:

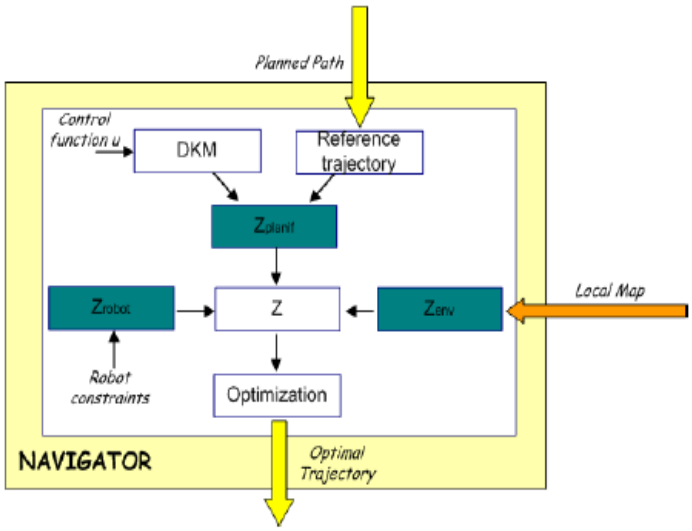

Fig. 57: Cost function structure.

The cost function should be defined as follows equation:

$$Z = Z_{planif} + Z_{robot} + Z_{env} \tag{21}$$

where:

- Z_{planif} quantifies the difference between the reference trajectory and the evaluated paths of the robot; the cost is composed by a first stage cost function and a terminal cost function. The two function are based on computation of the quadratic distances point by point of evaluated trajectories.
- Z_{robot} quantifies the constraints inherent to the robot such as actuators saturations and dynamics constraints. In a wheeled mobile robots, they are usually related to the wheel's maximum rotation speed, the maximum acceleration and deceleration.
- Z_{env} quantifies constraints of the environment, penalty due from fixed or mobile obstacles. It is assumed that the local map environment is known as a on-line refreshed occupancy grid [35] where to each square of the grid is assigned a boolean value indicating the occupancy.
  The cost is calculated projecting the trajectory to evaluate into the grid, and a zero cost identify a free trajectory from obstacles. One can note that the proposed DKM predictive control method is independent from the map representation type, for example Bayesian occupancy grid or a potential field map can be integrated.

6.4.e Optimization algorithms comparison

This chapter aims to illustrate motivations that leads to adopt the navigator's algorithm optimization. A critical obstacle situation is illustrated in Fig. 58, or rather the robot encounters an U-trap shape obstacle. The first simulation, performed by Nicolas Morette [34], a deterministic algorithm has been implemented.

- The deterministic algorithm is trapped in this local minimum in the parameters space (Fig. 58 - top left and bottom left) and cannot find the four parameters $P=(p_1, p_2, p_3, p_4)$ corresponding to the global minimum of cost function Z (expression 21).
- The stochastic algorithm manages to find the parameters regardless the initial settings (see Fig. 58 - top right and bottom right) and is not subjected by local minimums.

The success of the deterministic algorithm, to find the global minimum, highly depends from parameter initialization. The simulated annealing algorithm provides less accurate results but they are constantly near to the global minimum.

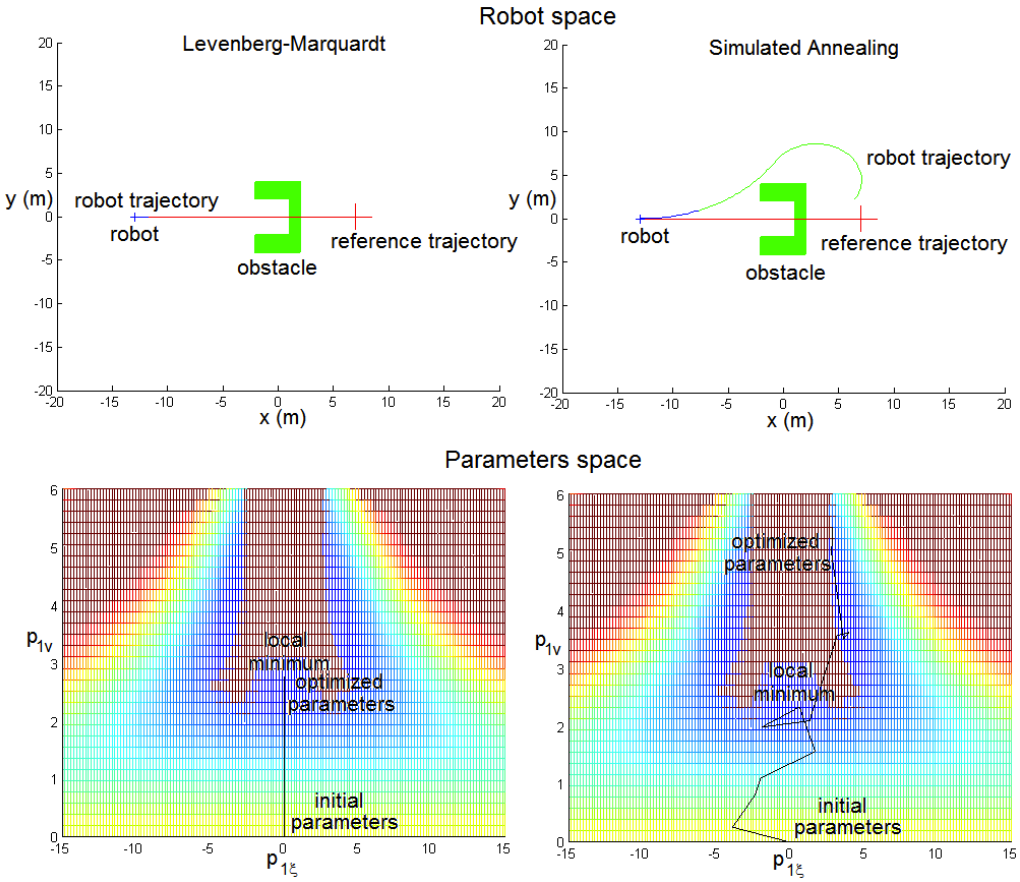

Fig. 58: Comparison between a deterministic and stochastic algorithms in presence of minimum local.

How display Fig. 58, the complete bypass trajectory is obtained from stochastic algorithm, in particular simulated annealing algorithm [20].

6.4.f **Simulated annealing optimization algorithm**

Cit. [36]: The evolution of nature led to the introduction of highly effective and power efficient biological mechanisms; result of billions of years of evolution. Failed solutions often led to the extinction of the specific species that became a fossil.

Imitating the mechanisms of nature, offers enormous potentials for the life improvement. It becomes significant to mimic biological methods, processes and systems.
In robot mobile, every change of environment (e.g. an mobile obstacle) involves to consider a new configuration of the robot and this require a new complete computation of the trajectory. It should be necessary to adopt algorithms to satisfy optimization of the motion.
Simulated annealing (SA) optimization algorithm has been used to find the optimum trajectory. The idea of SA comes from N. Metropolis in 1953 [37], which is motivated by the physic annealing process. This method can find the global minimum with a high probability and is also applicable for the solution of discrete optimization problems. The term of this optimization method derives from the process of material (metal) cooling at a slow rate, known as annealing. The method is based on the simulation of thermal annealing of critically heated solids [20]. For example, when a metal is brought into a molten state by heating it to a high temperature, the atoms move freely. The metal state attain a high energy state. As the temperature reduces, the atoms movements get restricted. The atoms tend to get ordered and forming crystals having the minimum possible internal energy. This process of crystals formation depends essentially on the cooling rate.
If the temperature of the molten metal is reduced at a very fast rate, it may not be able to achieve a crystalline state. So a polycrystalline state may be obtained having a higher state of energy than that crystalline state. Rapid cooling may introduce defect inside the material.
Thus the temperature of the heated solid (molten metal) needs to be reduced at a slow and controlled rate to ensure proper solidification with a highly ordered crystalline state that corresponds to the lowest energy state (internal energy).

The simulated annealing method aim to achieve the minimum function value in a minimization problem, simulating the process of slow cooling of molten material. The cooling phenomenon is simulated introducing a temperature-like parameter and controlling it using the concept of Boltzmann's probability distribution. This last one implies that the energy (E) of a system in thermal equilibrium at temperature T is distributed probabilistically according to the relation (22):

$$P[E]=\mathrm{e}^{-\Delta E/kT} \tag{22}$$

where *P(E)* denotes the probability of achieve the energy level E, and k is the Boltzmann's constant. At a high temperature levels, equation (22) shows that the system has nearly a uniform probability of being at any energy state. In fact the term *P(E)* tends to a unit value. However, at low temperatures, the system has a small probability of being at a high-energy state. These observations indicates that when the search process is assumed to follow Boltzmann's probability distribution, the convergence of the SA algorithm can be controlled by controlling the temperature T. The temperature at k-th

iteration T_k is given by the following cooling function:

$$T_k = \left(1 - \frac{k}{k_{max}}\right) \cdot T_0 + T_{min} \tag{23}$$

Considering the function minimization problem, let the current design point (state) be X_i , the corresponding energy state E_i of a thermodynamic system is given by function cost $f_i=f(X_i)$. So the energy Ei at state X_i is given by

$$E_i = f_i = f(X_i) \tag{24}$$

According to the Metropolis criterion, the probability of the next design point X_{i+1} depends on the difference of energy state (or cost function value) at the two design points (states) given by

$$\Delta E = E_{i+1} - E_i = f_{i+1} - f_i = f(X_{i+1}) - f(X_i) \tag{25}$$

The new state X_{i+1} can be found using the Boltzmann's probability distribution:

$$P[E_{i+1}] = min\left(1, \mathrm{e}^{-\Delta E/kT}\right) \tag{26}$$

Equation (26) return the minimum value between 1, chosen for simplicity and the exponential function. If the function value at X_{i+1} is smaller than at X_i , according with (25), $\Delta E<0$, and (26) give $P[E_{i+1}]=1$; and the point X_{i+1} is always accepted. This is a logical choice in the context of minimization of a function, reaching the smaller energy state of the system. On the other hand, when $\Delta E>0$, the function value f_{i+1} at X_{i+1} state is worse than the one at X_i state, the probability of accepting the point X_{i+1} is finite according to the Metropolis criterion.
Note that the probability of accepting the point X_{i+1} is not the same in all situations, but depends on the values of energy and temperature. If the temperature is large, the probability for design points X_{i+1} will be high. Thus at high temperatures, even worse design points X_{i+1} are likely to be accepted, thanks a high probability of equation (22). However, if the temperature is small, the probability of accepting worse state points X_{i+1} will be small. Fig. 59 illustrate a flow chart related the simulated annealing procedure.
From literature [20] this method benefits meanly by follows features:

1. the quality of final solution is not affected by the initial guesses, but worse starting designs may increase computational effort.
2. the convergence or transition characteristics are not affected by the continuity or differentiability of the functions, thanks the discrete nature of the function and constraint evaluations.
3. The convergence is also not influenced by the convexity status of the feasible space.

On the other hand this method require a detailed parametrization (e.g. the choice of the initial temperature, the temperature reduction factor called cooling rate) because they play important roles in the successful of convergences SA algorithm. All these

parameters still remain an art and generally require a trial-and-error process to find suitable values. Chapter 7 propose navigator results assuming following parametrization:

- The temperature reduction strategy (also termed the cooling schedule) is function of iteration algorithm number how shows linear equation (23).
- A random steps of 2-4% are performed in the parameter space because the new design point from actual state must be taken in the vicinity of the feasible space research. The random step unit is in rad/s according to the robot speed commands unit.
- The choice of the initial temperature has been chosen as the average value of the objective function (cost function) computed at a number of randomly selected points in the design space.

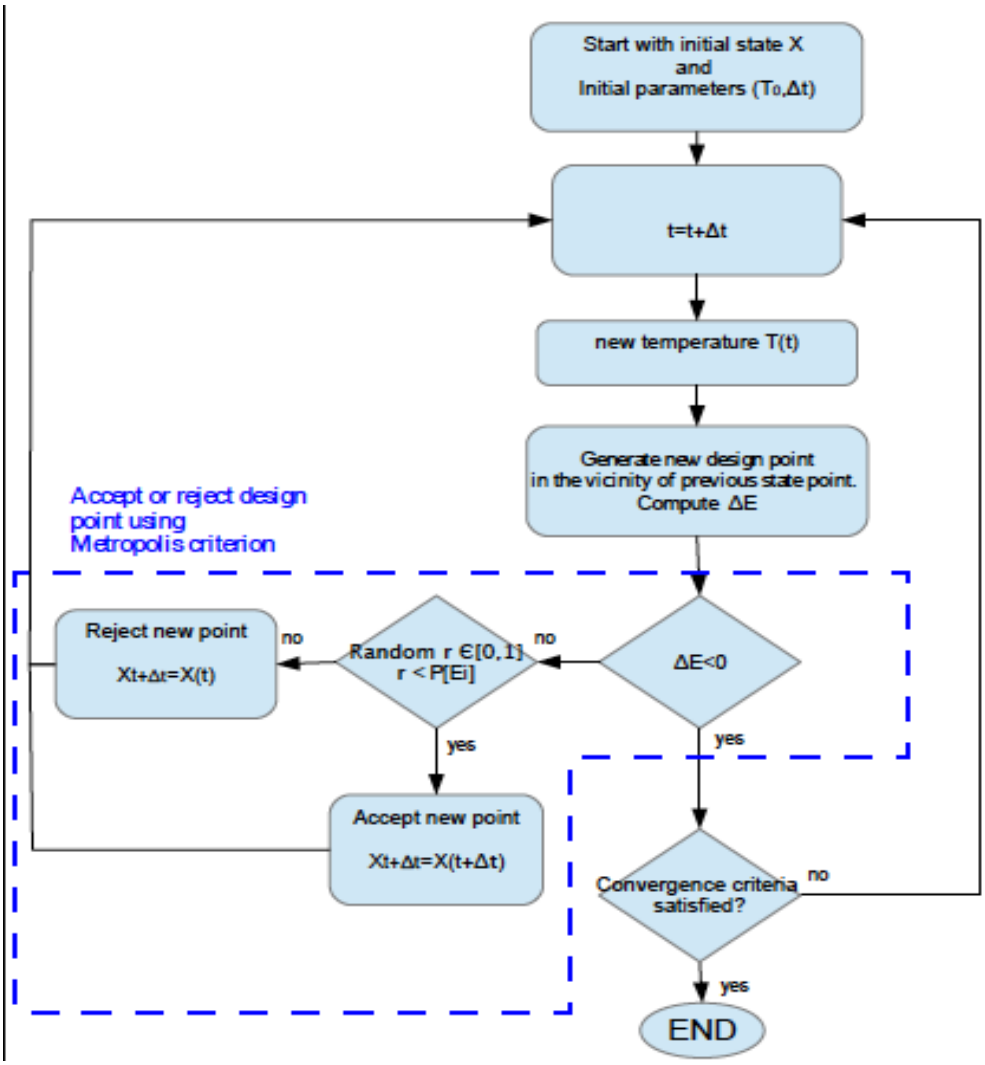

Fig. 59: Simulated annealing procedure.

7 Navigation results

7.1 Introduction

This chapter shows the predictive navigation results by direct kinematic model mentioned on chapter 6 . The navigator provides optimal trajectories according to the environment, robot constraints and the path planner block. This last block is located in the higher levels of the architecture. The path planner send to navigator a list of waypoints processed with first-come first-served service. The navigator commands continuously the robot into environment: from its actual position to the next waypoint. The trajectory, based on specific direct kinematic model of the robot under test, is computed thanks to an optimization algorithm. The optimal commands, solution of the problem, corresponding to the optimal trajectory, are then given to the pilot. So the navigator moves the robot autonomously continuously towards a destination point to point into the map.

7.2 Navigator performances

In order to test the navigation method and its adaptability to various kinematics, this method was applied to WifiBot robot model. The *path planner block* gives to the *navigator* a list of waypoints placed into the map in vector notation (x,y,θ), where x,y is the cartesian displacement into the map and θ is a possible orientation. Then the received waypoints are connected thought a Bezier curve forming a path (see Fig. 60). It is possible to note that this curve does not take into account the local environment constraints and the robot's physic constraints. To obtain the reference trajectory, or better the robot's admissible trajectory from the Bezier curve, the prediction horizon time $[t_0 , t_0+T_P]$ and the reasonable reached speed must be taken into account. A recursive algorithm permit to obtain the reference trajectory from Bezier curve and waypoints, based on effective distance covered by the robot during the fixed prediction time T_P. Fig. 60 display how the Bezier path connect a series of successive waypoints provided from path planner block. In this case the path planner block provides waypoints located around the obstacle, but this lucky condition, if not verified, do not compromise the optimal solution from navigator algorithm, which avoid obstacles anyway. Thanks a recursive algorithm, Fig. 61 display how the trajectory reference curve is obtained from Bezier curve (continuous line) consistently the robot speed constraints. Fig. 62 shows an optimal trajectory result from optimization algorithm, considering the reference trajectory and robot constraints.

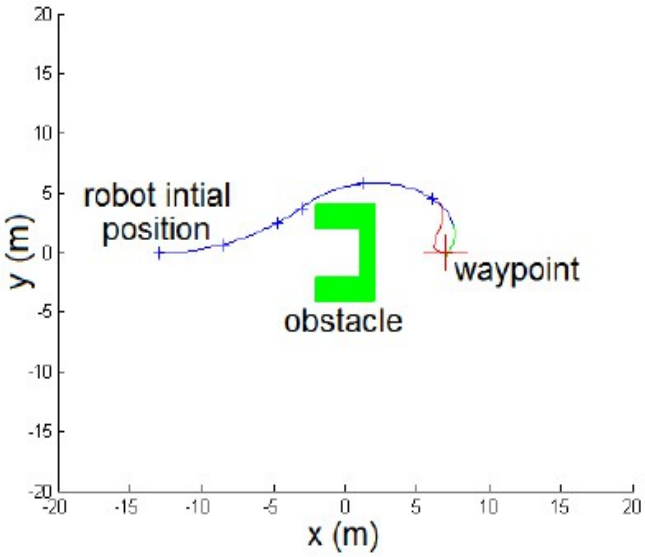

Fig. 60: Bezier curves connecting waypoints.

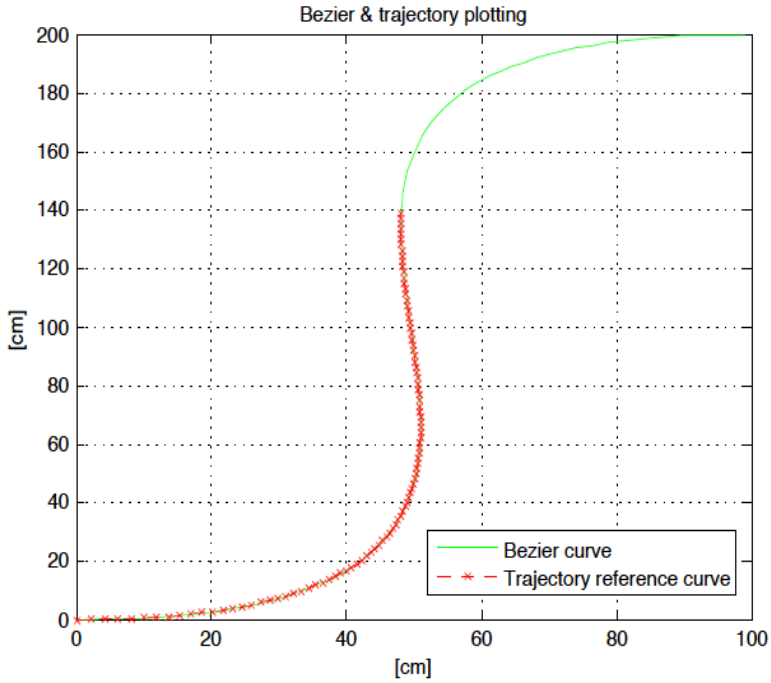

Fig. 61: Trajectory reference curve obtained from Bezier curve.

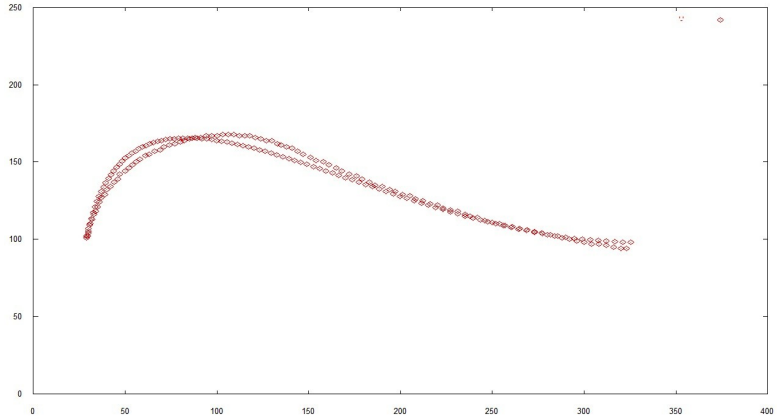

Fig. 62: Reference trajectory and optimal solution.

Following pictures show real results when the robot moves safely among the obstacles, and using specific trajectories according with its motions abilities. The AS optimization algorithm provides obstacle avoidance trajectories (and related robot commands) solutions. In first picture (Fig. 63), the virtual border margins security are takes into account by function constraints, lets to avoid robot impact.

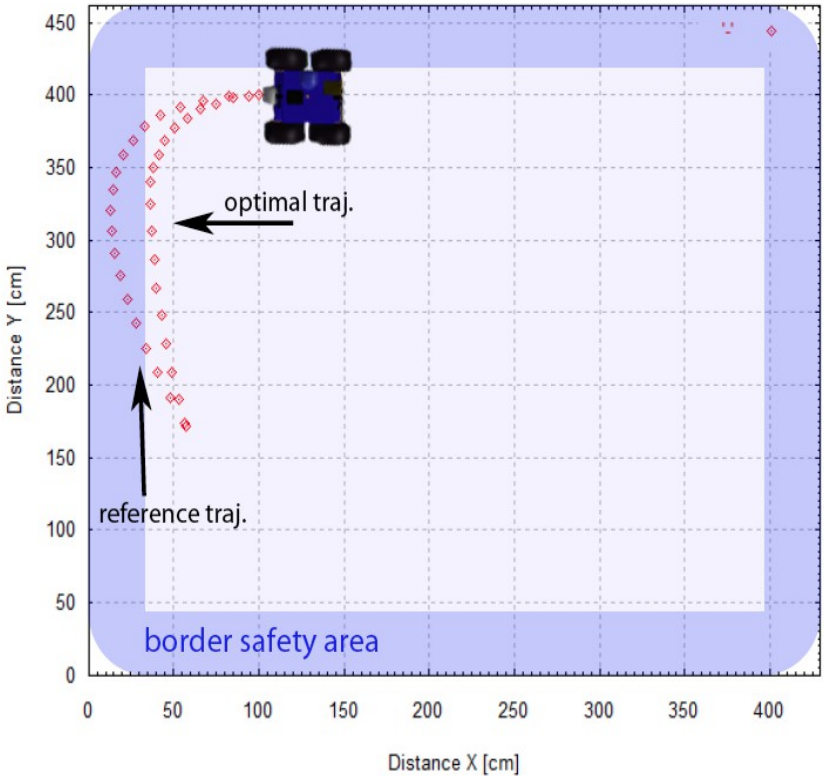

Fig. 63: Optimal trajectory considering border security margins.

Fig. 64 shows the navigation result for the WifiBot robot navigating around obstacles. Despite the fact that the reference path intercepts obstacles, the navigator provides obstacle bypassing trajectory solution. Enabling the robot to reach its various waypoints while avoiding collision. The robustness of this method ensure to find a global minimum solution, enabling the robot to diverge from the reference path when needed.

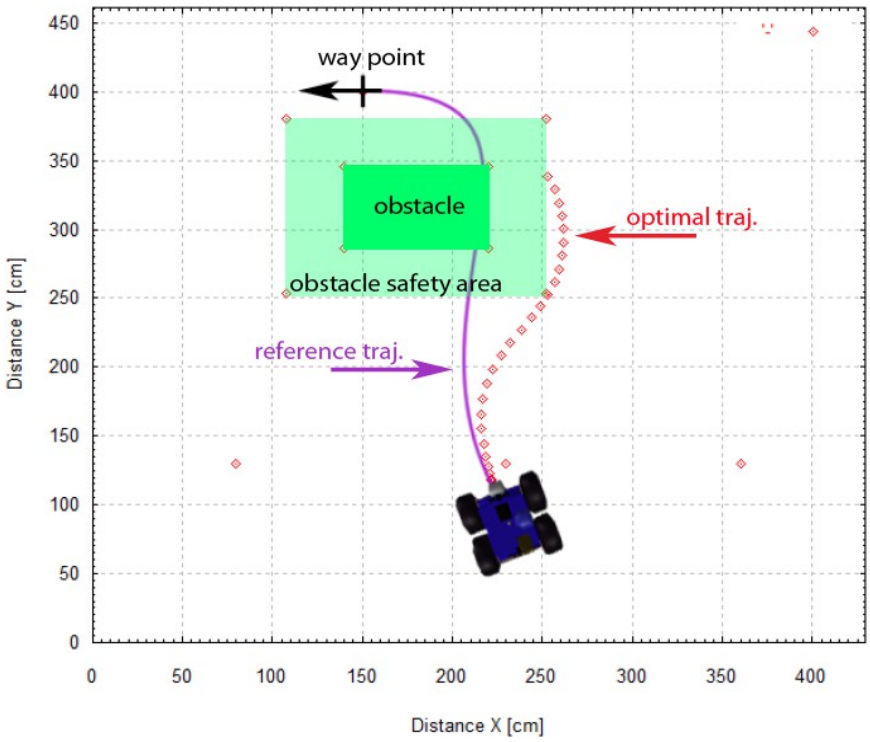

Fig. 64: Obstacle trajectory avoidance

Using an adapted temporal horizon enables the generation of trajectories that are long enough to bypass obstacles:

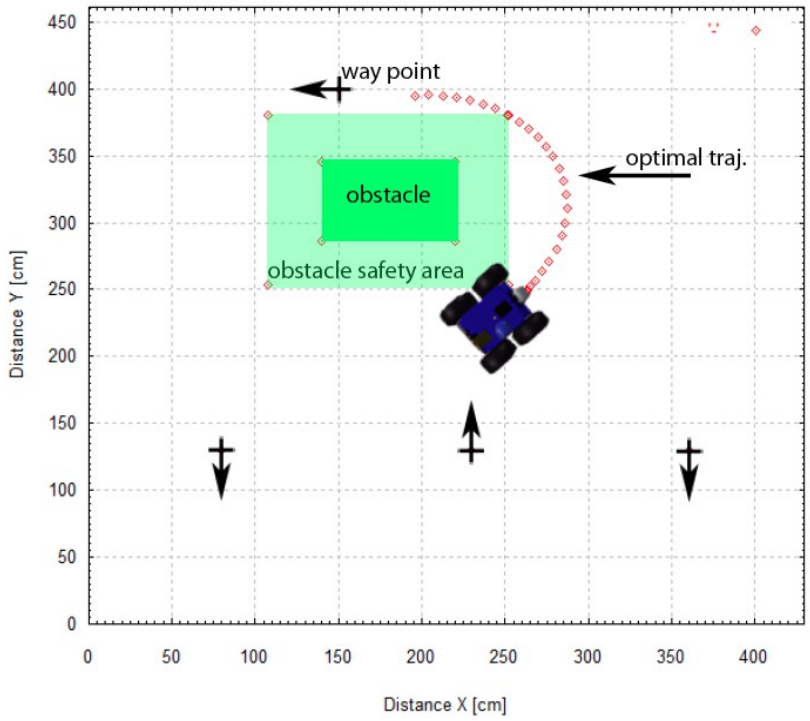

Fig. 65: Obstacle robot avoidance

Fig. 66 shows a demo where the Bezier reference trajectory (continuous track line) is computed from way-points list. Fig. 67 shows the real track reached from mobile robot during his evolution into the map. Is possible to note which the obstacle and borders are avoided thanks the optimization algorithm solutions. Care was taken the robot from impacts during the navigation thanks the safety area margins.

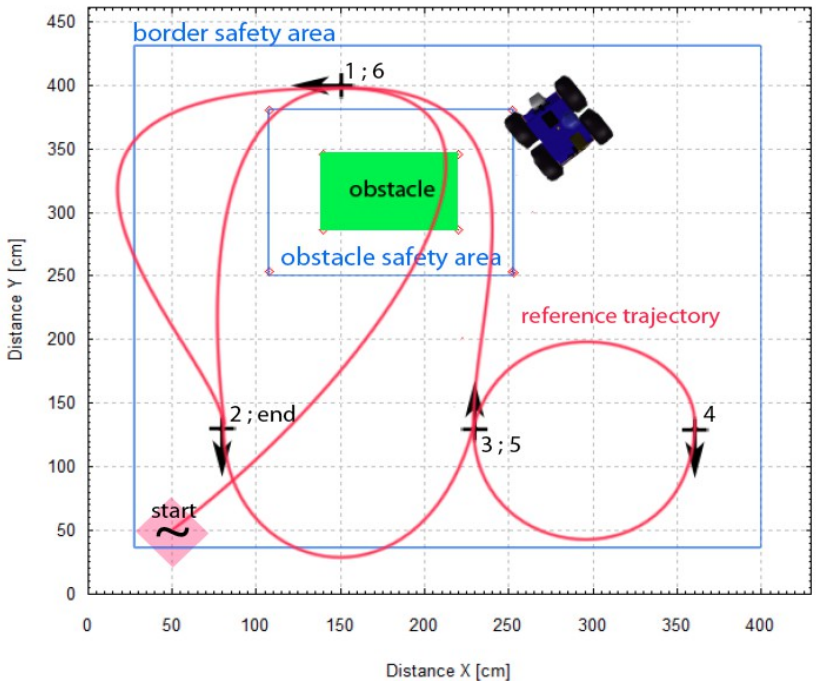

Fig. 66: Demo robot reference trajectory.

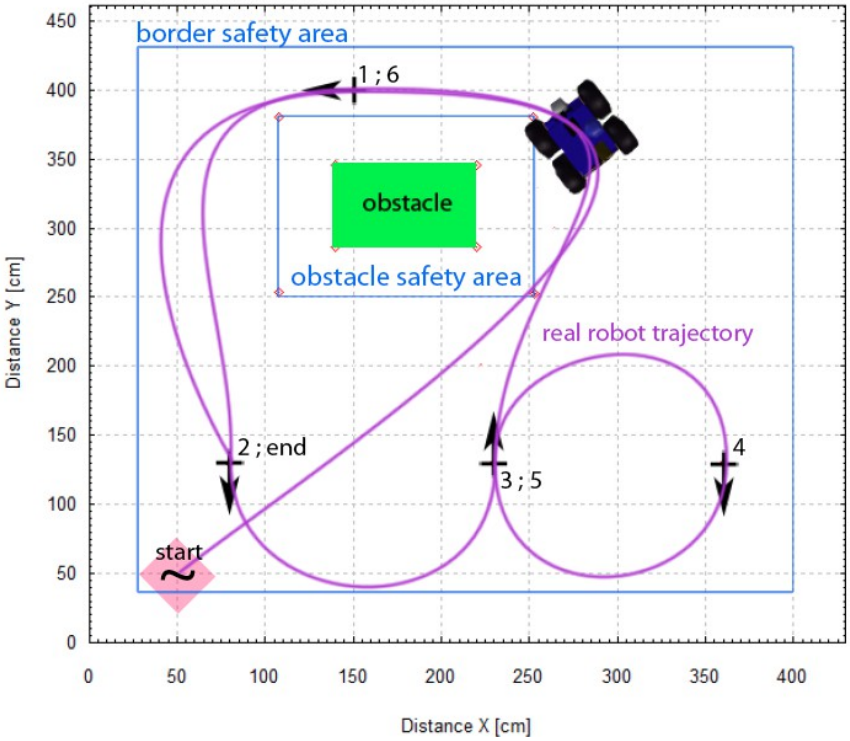

Fig. 67: Real robot reached trajectory.

7.2.a RTMaps drawing

RTMaps middleware offers a modular platform where data samples flow between functional blocks. Fig. 68 shows the graphical window interface dedicated to data acquisition from WifiBot robot sensors. In particular from the top, telemeter-laser rangefinder, IMU. From WifiBot driver board returns robot speed and odometer values: left and right side. All these datas are sent by a *SocketSender* block to a supervision station, for monitoring the robot state during navigation.

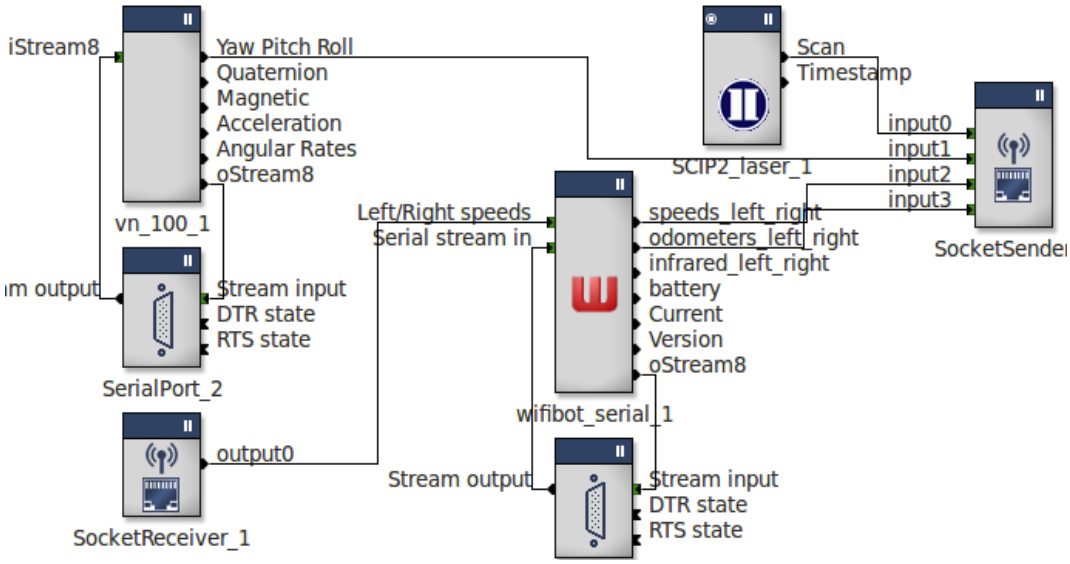

Fig. 68: RTMaps drawing, sensors data acquisition.

Fig. 69 shows the drawing connecting different functional blocks of robot architecture. In sequence, the localization block uses the telemeter-laser data, IMU and odometers values to localize the robot into the map. It gives to the navigation block the robot position (x,y,θ). Necessary from the navigation block for projecting the trajectory into the space, to find optimal commands. Then the optimal robot commands are elaborated by the pilot: the most reactive functional block of robot architecture.

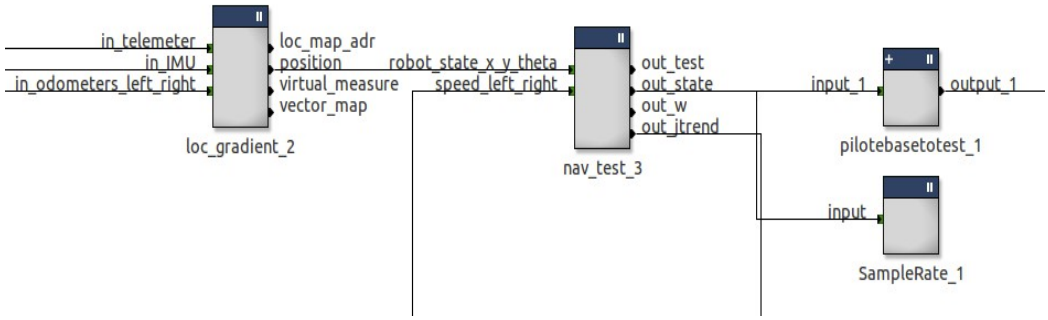

Fig. 69: Localization block, Navigator, and Pilot architecture blocks.

7.2.b Navigation conclusions

The experimental results validate this navigation method based on predictive control and trajectory projection through the direct kinematics model. Indeed, the robot was able to move safely around obstacles, with fluent trajectories between the successive waypoints, and with enough autonomy to diverge from the reference trajectory when needed. However, was noticed that the choice of the prediction horizon multiplied by the maximum velocity of the robot was a key factor to enable the navigator to find bypass trajectories. Moreover the settings of the simulated annealing algorithm are very important too, and these settings are complicated to do well (lots of parameters to set, and sometimes no systematic methods to do this).

8 Conclusions and further work

In this thesis has been implemented and tested a new generation of mobile robot navigator thanks to the predictive control approach. The contribution aims to the development of an autonomous mobile robot architecture inside the PROTEUS Project. The work was integrated in the team development of Prof. Cyril Novales and Ing. Nicolas Morette, allowing to validate the new theory of navigation.
Into robot's control architecture the navigator has proved to be a pivot between the lower reactive piloting and servoings levels (the continuous levels) and the upper deliberative path and mission planning levels (usually the discrete levels). From a deliberative point of view, practical simulations and demonstration showed that the robot is capable of avoiding obstacles, moving safely, thanks bypassing trajectories on it predictive horizon time T_P.
Almost all mobile robots are non-holonomic, and an analytic invert model cannot be defined: this means there are situations where it is impossible to generate a trajectory that follows a desired path. Otherwise considering a navigation method by direct kinematics model (DKM) leads to consider a model that can always be determined analytically, and is often simple to define for most robot's structures. Using a navigator by DKM means that any kinds of mobile robots can be easily adapted knowing the robot's motion ability precisely.
The validation of this method leads to replace an existing discrete DKM navigator (discrete because the solution is chose in a finite number of trajectories mapped into memory) towards the presented continuous domain DKM navigator. In this way, providing a continuous variation of parameters value is possible to generate an infinite number of trajectories: the robot motion abilities are more throughly exploited.
The use of stochastic optimization algorithm such as simulated annealing enables the robot to find optimal commands solution (related to obstacle bypassing trajectories) into continuous domain. This kind of algorithm provide a trajectory offering an escape from local minima; otherwise deterministic algorithm is not able to stay always nearly the global minimum. On the other hand SA algorithm require a detailed parametrization because they play important roles in the successful of convergences. Generally require a trial-and-error process to find suitable values (the global minimum) with a high probability. A choice and methodology was proposed in order to obtain an acceptable minimum solution.
The robot final tests showed how the navigation method enables the robot to correctly navigate in cluttered surroundings by avoiding static obstacles; allowing the integration of the continuous navigation by DKM into autonomous robot's control architecture.

Further work

Thanks the use of a DKM navigation method, other kind of robot (like CyCab model) should be tested to proving the flexibility and adaptability of this method.
The use of *Simulator* should be useful to test also the robustness of algorithms and robot platform in general.
With the validation of *Navigator* module, in order to implement the robot's control architecture, other methodology for *pilot* and *path-planning* modules must be simulated and tested.

APPENDIX A

DKM for a WifiBot differential drive mobile robot

The position of the robot in a plane surface is given by the vector (x, y) , which contains the Cartesian coordinates of its characteristic point *P* (geometrical center of the mechanical structure). Usually, this point is placed in the middle of the common axis of the driven wheels. As is possible to see in Fig. X, the orientation of the differential mobile robot is given by the angle θ between the vector direction of the instant linear velocity of the robot $\vec{v}$ and the local vertical axis.
The instant linear velocity of the robot $\vec{v}$ is attached and defined relative to the characteristic point *P*.
As equation (27) denotes, the instant linear velocity is a result of the linear velocities of the left driven wheel $\vec{v}_L$ and respectively of the right driven wheel $\vec{v}_R$, where two velocity vectors $\vec{v}_L$ and $\vec{v}_R$ are permanently perpendicular on the common mechanical axis of these two driven wheels.. The same equation should be expressed in terms of angular velocity known the radium wheel (28).

$$v = \frac{v_L + v_R}{2} \tag{27}$$

$$v = \frac{(\omega_L + \omega_R)}{2} \cdot r \tag{28}$$

The next two equations regrouped in (29) gives the two Cartesian components of the linear velocity:

$$\begin{aligned} v_x &= \dot{x} = v * \cos(\theta) \\ v_y &= \dot{y} = v * \sin(\theta) \end{aligned} \tag{29}$$

The robot state should be defined as a four elements vector: $\{X\} = (x, y, \theta, v)^T$ where $\{X\} \in \mathbb{R}^{[4x1]}$
The two state equations for the linear velocity components are given using equation (27) into (29):

$$\begin{aligned} \dot{x} &= \frac{v_L + v_R}{2} * \cos(\theta) \\ \dot{y} &= \frac{v_L + v_R}{2} * \sin(\theta) \end{aligned} \tag{30}$$

A third state equation representing the angular velocity of the robot, can be write as:

$$\dot{\theta} = \omega = \frac{v_L - v_R}{L} \tag{31}$$

where L represent the length of the wheel axis.
For the simulation the model is defined to following equations:

$$\begin{aligned} x_K &= x_{K-1} + T_P * \frac{v_{L_K} + v_{R_K}}{2} * \cos(\theta_K) \\ y_K &= y_{K-1} + T_P * \frac{v_{L_K} + v_{R_K}}{2} * \sin(\theta_K) \\ \theta_K &= \theta_{K-1} + T_P * \frac{v_{L_K} - v_{R_K}}{L} \end{aligned} \tag{32}$$

where Tp is the sampling time and x_k and y_k the Cartesian positions of the driven wheels in the global reference attached to the operational space.

The Kinematic model determines directly the displacement of the robot in the Cartesian reference based on derived joint coordinates. The matrix which permit to do this is the Jacobian matrix $[\Im]$.

$$\{\dot{X}\} = [\Im] * \{\dot{q}\} \tag{33}$$

where:
$\{\dot{X}\} = (\dot{x}, \dot{y}, \dot{\theta})^T$ is the robot displacement vector in the Cartesian space.
$\{\dot{q}\} = (\dot{q_1}, \dot{q_2}, \ldots, \dot{q_n})^T$ is the displacement in the joint space.

However in robotic mobile wheeled this model is not always useful, should be more useful use a simplified kinematic model which linking the robot Cartesian speed with the command that may be apply directly to the robot:

$$\{\dot{X}\} = [C(q)] * \{\dot{u}\} \tag{34}$$

where in the case of a robot mobile with differential wheels the vector $\dot{u} \in \mathbb{R}^{\{2x1\}}$ is defined by the two angular speed right and left respectively: $(\dot{u}) = (\omega_L, \omega_R)^T$.

References

[1] Krakowiak Sacha: "Middleware Architecture", .
[2] Johnson, R. E.: "Frameworks=(Components+Patterns): How frameworks compare to other object-oriented reuse techniques", 1997.
[3] Blender: http://www.blender.org/.
[4] G. Echeverria, N. Lassabe, A. Degroote, S. Lemaignan: "Modular Open Robots Simulation Engine: MORSE", .
[5] LAAS: http://www.laas.fr/.
[6] M. Quigley, B. Gerkeyy, K. Conleyy, J. Fausty, T. Footey, J. Leibsz, E. Bergery, R. Wheelery, A. Ng: "ROS: an open-source Robot Operating System", .
[7] A. Mallet, C. Pasteur, M. Herrb, S. Lemaignan, F. F. Ingrand: "Genom3: Building middleware-independent robotic components", 2010.
[8] G. Metta, P. Fitzpatrick, L. Natale: "YARP: yet another robot platform", 2006.
[9] Pocolibs: https://www.openrobots.org/wiki/pocolibs/.
[10] A. Goktogan and S. Sukkarieh: "Simulation of multi-UAV missions in a real-time distributed hardware-in-the-loop simulator", 2007.
[11] WifiBot: http://www.wifibot.com/.
[12] Bruno Siciliano, Oussama Khatib: "Handbook of Robotics", 2008
[13] R. Arkin: "Behavior-Based Robotics", 1998.
[14] Alami R., Chatila R., Fleury S., Ghallab M: "An architecture for autonomy", .
[15] D. Lyons : "Planning, reactive", .
[16] R. A. Brooks: "Intelligence without representation", 1991
[17] G. Mourioux, C. Novales, L. Josserand: "Framework for modular robot control architecture", .
[18] R. Zapata: "Reactive Bahaviours of Mobile Manipulators Based on the DVZ Approach", 2001.
[19] Lionel Lapierre, Rene Zapata and Pascal Lepinay: "Simultaneous Path Following and Obstacle Avoidance Control of a Unicycle-type Robot", 2007.
[20] Singiresu S. Rao: "Engineering Optimization",
[21] Y. Koren, J. Borenstein: "Potential field methods and their inherent limitations for mobile robot navigation", 1991
[22] A. Sgorbissa, R. Zaccaria: "Planning and obstacle avoidance in mobile robotics", 2012.
[23] IBRO: "International Brain Research Organization", .
[24] Soheil Keshmiri, Shahram Payandeh: "Mobile Robotic Agents' Motion Planning in Dynamic Environment: a Catalogue", 2009.
[25] Fliess, M., Lévine, J., Martin, P., and Rouchon: "Flatness and defect of non-linear systems : introductory theory and examples", 1995
[26] INRIA: https://www-sop.inria.fr/arobas/.
[27] Fruchard, M.: "Méthodologies pour la commande de manipulateurs mobiles non-holonomes" , 2005.
[28] Morin, P. et Samson, C.: "Trajectory tracking for non-holonomic vehicles : overview and case study",
[29] J. Courbon, Y. Mezouar, L. Eck, P. Martinet: "A generic framework for topological navigation of urban vehicle", 2009.
[30] Courbon, J., Mezouar, Y., Lequievre, L., Eck, L.: "Navigation of urban vehicle: An efficient visual memory management for large scale environments", .
[31] Bonnafous, D. Lacroix, S. Simeon, T. : "Motion generation for a rover on rough terrains", 2001.

[32] Novales, C.: "Pilotage par actions reflexes et navigation locale de robots mobiles rapides" , 1994.
[33] Nicolas Morette: " Contribution à la Navigation de robots mobiles : approche par modèle direct et commande prédictive" , 2010.
[34] N. Morette, C. Novales, L. Josserard, P. Vieyres: "Trajectory projections and direct model predictive control approach for a mobile robot's navigator.", .
[35] Fruchard, M.: "Méthodologies pour la commande de manipulateurs mobiles non-holonomes" , 2005.
[36] Yoseph Bar-Cohen : "Biomimetics: mimicking and inspired-by biology", .
[37] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller: "Equation of state calculations by fast computing machines", 1953.

List of figures

Acknowledgments

Firstly I would like to express my gratitude to Prof. Roberto Oboe for giving me the opportunity to make this very important life and work experience. And also for his helpfulness for my graduation.
I express my gratitude to all PRISME Laboratory, in particular Prof. Cyril Novales, Prof. Pierre Vieyres, Ing. Nicolas Morette, and trainees Noura Ayadi and Athmane Ayouni for their leadership, support, attention to detail, hard work that led me to advance the work with enthusiasm and interest, leading to results, bring off the target and helped enrich my experience. Thanks also for life-long education and continuing education during my permanence. I thank also Nicolas Du Lac from Intempora for his valuable suggestions.
I would also like to thank the many fellow students and friends that lead me during this important university experience.
For last but not least I thank all my family for the valuable support that they provided me through my entire life and giving me the opportunity to conclude my university studies.