Università degli Studi di Padova
Universitat Politècnica de Catalunya

Department of Information Engineering
Escola Tècnica Superior d’Enginyeria Industrial de Barcelona

Master Degree in Automation Engineering

Approaching path planning in dynamic environment

Home university supervisor
Angelo Cenedese

Host university supervisor
Alberto Sanfeliu Cortés

Master Candidate
Marta Galvan

Academic Year 2018/2019
I dedicate this thesis to my parents, because they gave me the opportunity to develop all the work living abroad, during a personal and academic life experience which taught me a lot, especially about myself, and that I’ll bring with me forever. In particular, I have to say thanks to my mom, for having always supported me during all the years of studies.

As last, I thank all the friends that this long journey to the destination made me meet, from the initial years until the months spent in Spain, because they helped me with the studies, but even more they taught me a lot about interpersonal relationships.
Abstract

This work, entitled "Approaching path planning in dynamic environment", has been carried out in the research center "Instituto de Robòtica i Informàtica industrial" (I.R.I.) in cooperation with "Universitat Politècnica de Catalunya" (U.P.C. - E.T.S.E.I.B.), during the mobility period of Erasmus between U.P.C. and Università degli Studi di Padova.

The project aims at the implementation on a mobile robot of a path planning algorithm for approaching a person. The method allows the robot, that is moving in a dynamic environment, to approach the person in a natural human-like manner. Furthermore, the procedure finds out a feasible path for the robot between a set of candidate ones, computed using splines. Moreover, to keep the computation of the trajectories constantly updated depending on the entities present in the environment, the planner is run in real-time.

For having the robot able to move in dynamic environments avoiding static obstacles and other people, the Extended Social Force Model [4], has been combined with the G2-splines, and social forces have been also included in the definition of the cost related to a path for the best trajectory evaluation. Furthermore, for the cost, a multi-objective optimization [6] has been used and changed to adapt to our case in order to take into account different interesting criteria concerning distance, orientation and avoidance of static and dynamic obstacles.

Many simulations have been run during the realization of the algorithm to test its quality, and real experiments have been carried out with Tibi robot. Real tests demonstrated the validity of the procedure and that the obtained behavior for the robot is socially acceptable for approaching a human.

Keywords: G2-spline, Social Force Model, Multi-Objective Cost, mobile and social robotics, robot-human approaching.
Contents

Abstract v

List of figures ix

Listing of acronyms xi

1 Introduction 1
  1.1 State of art .................................................. 2
       1.1.1 Mobile robots in dynamic environments ............. 2
       1.1.2 Planning and navigating with mobile robots ...... 2
       1.1.3 Robot approaching people ........................... 4
  1.2 Problem description and objective .......................... 8
  1.3 Outline of the thesis ....................................... 8

2 Theoretical background and new contributions 9
  2.1 AKP approaching planner ................................... 10
  2.2 Extended Social Force Model ............................... 15
  2.3 Path generation .............................................. 18
  2.4 Multi-cost definition ....................................... 20
  2.5 Robot overview .............................................. 22
       2.5.1 Robot model ........................................... 23
       2.5.2 Forward kynematics .................................. 25
       2.5.3 Trajectory tracking control ......................... 26

3 Hardware and Software description 29
  3.1 Hardware of Tibi ............................................ 30
       3.1.1 Horizontal sensors ................................... 30
       3.1.2 Eye cameras (Bumblebee2) ............................ 31
       3.1.3 LADYBUG2 ............................................ 32
       3.1.4 Segway (RMP 200) ..................................... 32
  3.2 Software ..................................................... 33
       3.2.1 Robot Operating System (ROS) ......................... 34
       3.2.2 ROS nodes that the planner uses ..................... 37
Listing of figures

2.1 Overview of the planning scheme ........................................ 11
2.2 Example of a $G^2$-spline path connecting $P_A$ and $P_B$ states  ......................................................... 19
2.3 Differential drive robot model ........................................ 24
3.1 Tibi and Dabo robots ....................................................... 30
3.2 Location of sensors in Tibi and Dabo ................................ 31
3.3 Range scan of the horizontal laser ................................... 32
3.4 Bumblebee2 cameras ...................................................... 32
3.5 LADYBUG2 ................................................................. 33
3.6 Segway RPM 200 ............................................................ 33
3.7 RViz 3D graphical visualization environment for ROS with the computed paths ......................................................... 36
3.8 Dynamic-reconfigure with the planner ............................... 37
3.9 The purple area indicates the planner, in green the laser people detector and filter, in blue the tracker, in red the odometry node. Next to the planner the RViz node for visualization. The other names are topics that leave or enter in the nodes, according with the direction of the arrow .......................... 38
3.10 Laser leg detection .......................................................... 39
3.11 People detected by the tracker and identified by a number .......................... 40
4.1 Robot and person states representation ............................... 43
4.2 Examples of robot final positions computation .................. 44
4.3 Examples of paths computed with $G^2$-splines algorithm ........... 45
4.4 Examples of best paths selection in different situations .............. 48
5.1 RViz visualization of the simulation with: $\left(x_p, y_p, \theta_p\right) = (0, -3, 0^\circ), (x_r, y_r) = (-2, 10.5), res = 0.2m$ ........................................ 50
5.2 Simulation with: $\left(x_p, y_p, \theta_p\right) = (0, -3, 0^\circ), (x_r, y_r) = (-2, 10.5), index = \{4, 1\}, dist = 3m, res = 0.2m$ .......................... 52
5.3 Simulation with: $\left(x_p, y_p, \theta_p\right) = (0, -3, 0^\circ), (x_r, y_r) = (-2, 10.5), index = \{4, 1\}, dist = 3m, res = 0.6m$ .......................... 52
5.4 Simulation with: $\left(x_p, y_p, \theta_p\right) = (0, -3, 0^\circ), (x_r, y_r) = (-2, 10.5), index = \{4, 1\}, dist = 3m, res = 0.8m$ .......................... 52
5.5 Simulation with: $\left(x_p, y_p, \theta_p\right) = (0, -3, 230^\circ), (x_r, y_r) = (-3, 6.5), index = \{10, 4, 1\}, dist = \{6m, 2m\}, res = 0.2m$ .......................... 53

ix
5.6 Simulation with: \((x_p, y_p, \theta_p) = (0, -3, 230^\circ), (x_r, y_r) = (-3, 6.5), index = \{10, 7, 3\}, dist = \{6m, 2m\}, res = 0.2m \) ........................... 54

5.7 Simulation with: \((x_p, y_p, \theta_p) = (0, -3, -40^\circ), (x_r, y_r) = (-3, 6.5), index = \{10, 7, 3\}, dist = \{6m, 2m\}, res = 0.2m, goal\_tolerance = 0.3 \) ........................... 55

5.8 Simulation with: \((x_p, y_p, \theta_p) = (0, -3, -40^\circ), (x_r, y_r) = (-3, 6.5), index = \{10, 7, 3\}, dist = \{6m, 2m\}, res = 0.2m, goal\_tolerance = 0.6 \) ........................... 57

5.9 Simulation with static obstacles in the environment: \((x_p, y_p, \theta_p) = (0, -3, 0^\circ), (x_r, y_r) = (-3, 6.5), \{\alpha, \gamma, \delta\} = \{1.0, 1.0, 1.0\}, \{A_m, B_m\} = \{5.0, 0.4\}, v\_break = 1.0 \) ........................... 59

5.10 Simulation with static obstacles in the environment: \((x_p, y_p, \theta_p) = (0, -3, 0^\circ), (x_r, y_r) = (-3, 6.5), \{\alpha, \gamma, \delta\} = \{1.2, 1.0, 1.0\}, \{A_m, B_m\} = \{4.5, 0.4\}, v\_break = 0.6 \) ........................... 60

5.11 Simulation with static obstacles in the environment: \((x_p, y_p, \theta_p) = (0, -3, 160^\circ), (x_r, y_r) = (-3, 6.5), \{\alpha, \gamma, \delta\} = \{1.2, 1.0, 1.0\}, \{A_m, B_m\} = \{4.5, 0.4\}, v\_break = 0.6 \) ........................... 61

5.12 Simulation with forces included in the cost computation: \((x_p, y_p, \theta_p) = (0, -3, 0^\circ), (x_r, y_r) = (-3, 6.5) \) ........................... 62

5.13 Simulation with forces included in the cost computation: \((x_p, y_p, \theta_p) = (0, -3, 160^\circ), (x_r, y_r) = (-3, 6.5) \) ........................... 64

5.14 Simulation with static obstacles and person: \((x_{ap}, y_{ap}, \theta_{ap}) = (0, -3, 0^\circ), (x_r, y_r) = (-3, 6.5), (x_p, y_p) = (0, 5), \{\alpha, \gamma, \delta\} = \{1.2, 1.4, 1.0\} \) ........................... 65

5.15 F.M.E. scenario ........................... 66

5.16 Real experiment without obstacles and person orientation \(\theta_{ap} = 90^\circ \) ........................... 67

5.17 Real experiment without obstacles and person orientation \(\theta_{ap} = 0^\circ \) ........................... 67

5.18 Real experiment without obstacles and person orientation \(\theta_{ap} = 180^\circ \) ........................... 68

5.19 Real experiment with static obstacles and person orientation \(\theta_{ap} = 90^\circ \) ........................... 69

5.20 Real experiment with static obstacles and person orientation \(\theta_{ap} = 0^\circ \) ........................... 69

5.21 Real experiment with static obstacles and person orientation \(\theta_{ap} = 180^\circ \) ........................... 70

7.1 Wheel linear velocities block ........................... 75

7.2 Pose derivation block ........................... 76

7.3 All robot system ........................... 77
### Listing of acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AKP</td>
<td>Anticipative Kinodynamic Planner</td>
</tr>
<tr>
<td>ESFM</td>
<td>Extended Social Force Model</td>
</tr>
<tr>
<td>ID</td>
<td>Identifier of one person, it is a number</td>
</tr>
<tr>
<td>ROS</td>
<td>Robot Operating System, software used to program robots</td>
</tr>
<tr>
<td>Node</td>
<td>An executable program, basic single unit in ROS</td>
</tr>
<tr>
<td>Publisher</td>
<td>ROS node that publishes information, for example: laser, coordinate points</td>
</tr>
<tr>
<td>Subscriber</td>
<td>ROS node that gets information published by another node</td>
</tr>
<tr>
<td>Rviz</td>
<td>3D Graphic visualization environment of ROS</td>
</tr>
<tr>
<td>Rosbag</td>
<td>File where data is stored. This data can be the outputs of the sensors of a robotic system, while we are performing an experiment. These files are used to reproduce later the same experiment in a computer simulation without using the robot to test the implemented programs</td>
</tr>
<tr>
<td>Markers</td>
<td>Geometric shapes represented in the Rviz such as spheres, arrows, cubes, etc. These serve to show the output variables of a node</td>
</tr>
<tr>
<td>Papers</td>
<td>Relatively short article intended for publication in scientific journals of scientific research</td>
</tr>
<tr>
<td>Tag</td>
<td>Black and white geometric figure, which serves in vision to detect some different points or positions. In our case, the tag serves to differentiate a specific person by the utilization of a vision detector. The vision detector sends a detection in the place that detects the tag</td>
</tr>
<tr>
<td>IRI</td>
<td>Instituto de Robòtica i Informàtica industrial</td>
</tr>
<tr>
<td>FME</td>
<td>Facultat de Matemàtiques i Estadística</td>
</tr>
</tbody>
</table>
The goal of this work is the realization of a real-time path planning for a mobile robot to approach a human, in dynamic environments. For this purpose, the work has been developed starting from the path computation using G2-Splines and then realizing a real-time planner that suits to our case.

The main challenging task is to obtain at each launch of the planner a feasible path for the robot in a dynamic environment, which is always changing configuration, by the moment that there are both static and dynamic obstacles in the scene.

First a basic algorithm, only for obtaining first objectives, useful for the complete realization, has been realized and tested in simulation using MATLAB; for this purpose the kinematics of the robot has been implemented using Simulink.

The following step has been the translation of the existing code to C++ language and ROS, and the improvement of the algorithm, in order to have an algorithm that could be simulated and run in the Tibi robot of the I.R.I.

After a basic case with no other entities in the environment except from the robot and the person to approach, static obstacles and people have been added to obtain a typical urban environment, and the final algorithm has been tested first in simulation, and then on the robot.
1.1 State of art

The following is a brief insight of the state of the art of the fields related to path planning and control of mobile robots in dynamic environments. Also some material on robot approaching people has been analyzed. In particular, previous approaching methods are exposed here, in order to present the new contributes carried by our methodology.

A small introduction on these topic and the exposition of previous works and concepts is necessary for a better comprehension of the proposed work.

1.1.1 Mobile robots in dynamic environments

The idea of having robots in social environments, then sharing urban areas with people, has carried out the development of several topics such as navigating skills for the mobile robots and approaching with humans. The accompany task has been considered too, because of situations like following people, learning objects and places to future interactions, helping aging people, accompanying a person to some place while walking in side-by-side formation, or accompanying groups of people. A great interest has grown in the deployment of service robots in urban environments. However, the complexity of these scenarios is notable, since there exist multiple interactions between pedestrians, static obstacles, and robots, and the impact produced by the deployment of service robots is of vital importance for the acceptance of robots in crowded environments. Therefore, robots navigation in urban environments is a complex task, because the robot has to navigate in a safe and natural human-like manner, and it has to adapt its behavior to avoid collisions with obstacles while not disturbing other people in the environment. Also, the realization of this behavior requires several robotic cognitive functions as perception, prediction, navigation and human-robot interaction.

1.1.2 Planning and navigating with mobile robots

During the last years approaches for robot navigation in crowded urban environments where people and objects are moving simultaneously while a robot is navigating have been proposed. Avoiding moving obstacles motivates the use of a robotic planner satisfying both dynamic and nonholonomic constraints, also referred as kinodynamic constraints.

[2] proposes a social aware reactive planning in human environments, motivated by the fact that a lot of methods have been developed to allow robots to navigate around people, but none of them explicitly account for the pre-established social conventions that people
use when moving around each other. First, a graph map with a set of destinations that completely describe the navigation environment is constructed. Then, a robot navigation algorithm is introduced, based on the so-called social-forces model, and capable of navigating in crowded environments in an acceptable social way, because mainly driven by the social-forces centered at the robot.

A proactive navigation approach with respect to the environment, in the sense that the robot calculates the reaction produced by its actions and provides the minimum impact on nearby pedestrians, has been proposed in [3]. The planner integrates seamlessly planning and prediction, and calculates a complete motion prediction of the scene for each robot propagation. While most approaches separate planning and prediction, indeed, here a simple method to jointly account predictions and planning by considering a union state of people and robots has been presented. This approach, then, integrates the search of a path avoiding obstacles as well as provides the inputs required to execute that trajectory considering kinodynamic constraints, instead of decoupling the problem into a search in space and a posterior optimization of the path satisfying the restrictions. Human motion prediction is achieved through the application of geometrical based predictors that infer human motion intentions and afterwards predict human motion in a continuous space, according to the Extended Social Force Model (ESFM), presented in [4].

The approach relies on a global planner, which provides a valid path to the goal unobstructed by static obstacles. The algorithm calculates, for each iteration, a path to a goal avoiding moving obstacles like pedestrians on the scene. The first output action is executed, and in the next iteration, a new plan is calculated, and a new action is executed. This approach permits a fast adaptation to changing environments, especially if the prediction estimation changes drastically. Moreover, the algorithm is implemented in real time to provide an adaptable local planning. Additionally, this approach shows proactive traits since the robot tends to initiate change rather than reacting to events. The trajectory calculated minimizes the amount of social work produced by the robot navigation at the same time that minimizes its navigation work and distance to goal.

Some limitations, such as the fact that the navigation algorithm is highly conditioned by the learning environment used, and then the lack of a more general approach for different kinds of scenarios, have been overcome in [6], where the advantages of the AKP have been maintained, but new contributions to the approach have been introduced. Finding a cost
function to correctly characterize robot navigation among people is not an easy problem to
solve; to tackle it a multi-objective cost function has been proposed, with the optimization of
different independent criteria. Indeed, a direct weighted-sum of costs is not able to provide a
general solution for different scenarios, while, after a normalization and a correct weighting,
this new cost function is able to provide a set of parameters valid for many scenarios.

Then, the construction of the planner tree by introducing a cost-to-go function, based on
the multi-objective costs to calculate distances between states, instead of an Euclidean-based
distance, is exposed. This cost function, has then been used to find the nearest vertex in the
tree $T$.

Moreover, the importance of the different cost functions in navigation algorithms have
been hugely discussed and a method to systematically reduce the complexity associated to
consider multiple criteria while performing robot navigation has been successfully intro-
duced, considering collision avoidance, minimization of the impact to pedestrians, efficiency
in reaching the goal, etc. In order to speed up the calculations, a steering heuristic to connect
states has been used, and, at the same time, a randomness factor to the steering function has
been introduced to obtain more distinct paths. The Extended Social Force Model (ESFM)
[4] has been used as the steering method. The AKP predicts people’s positions with respect
to time and calculates a set of paths, taking into account time restrictions, since dynamical
obstacles change their positions over time. Among the set of calculated paths, the minimum
cost path is chosen.

1.1.3 Robot approaching people

Finding social robots living together with humans in urban areas is a challenging task for the
future. To achieve that, robots have to deal with several situations, and develop different
skills. In particular, robots that navigate toward humans and initiate interactions have to
follow some social rules and patterns. The social implications of navigation paths and the
expectations they produce, indeed, are challenges that people handle regularly when moving
around each other in their everyday lives. These expectations of others’ behaviors are based
on non-verbal (or occasionally verbal) communication and knowledge of social rules and pat-
terns. If a person thinks the robot is coming too close, might react badly. That’s why a robot
following social rules constraints would make its behavior appear more predictable and thus
more comfortable to the people around it.

In [12], for the path planning and approaching of the robot, the main idea is to leverage
people’s customary use of social rules and patterns to help robots follow them. They collected data on human navigation around other humans, from which to create a model that can be used by a robot’s path planner to create paths that appear more socially acceptable. The key factor, in determining a person’s approach path to another human, has been assumed to be the direction of the other person’s attention. Therefore, several situations are possible: the person being approached could stand with the front of the torso, the right side, or the back toward the person approaching, and have their head facing forward, to the left, or to the right in each of the body poses.

Non-verbal communication and knowledge of social rules have also been studied in [13]. The ability to convey intentions non-verbally for a robot is considered beneficial for interaction initiation with humans, and, thus, for mutual collision avoidance and the reduction of interferences. Here, and in [14], specific trajectory features that provide the required natural movements and support intention conveyance are examined, such as smooth trajectory shapes, specified approach speed, appropriate human-robot distance, positioning in the field of view and the behavior for human-like dynamic obstacle avoidance.

The conducted studies revealed that trajectories with human-like features appear significantly more natural than the version without these specific features, enhancing non-verbal interaction initiation, and conveying an intention more quickly. Trajectories are generated basing on optimal control models with according constraints, and they must thereby externalize their purpose quickly to the desired interaction partner.

[17] concerns the ability to infer intentions and predict actions, in order to enable coordinating of one’s own actions with those of another human. Assuming that studying human-human interaction provides valuable insights allowing to implement mutual intention recognition and action prediction in robotic systems, it stated that the robots should act sufficiently predictable to enable the humans to attribute goals and predict motion trajectories. Then, after analyzing how well a human recognizes the goal of another person entering the room, and whether this ability is deteriorated by concealing gaze direction of the other person, the same experiment has been repeated by replacing the approaching person with a wheeled robot. On average, the distance at which subjects predicted the goal of the approaching agent was approximately 4 m, and depended on subject and goal position, but not on the type of agent. However, goal attribution showed a considerable proportion of errors for the robot (19%), much less for a human with hidden gaze direction (6%), and almost none for a
human with visible gaze (1%), concluding that the visibility of the gaze direction of the approaching person plays an important role in goal recognition.

As already explained, intercepting a moving person to interact with it is a challenging task for a robot; this has to anticipate the movement of the moving person and intercept it, while it has to avoid static and dynamic obstacles, such as other pedestrians in the environment. In addition, other people and obstacles can occlude the detection and tracking of people, thus, the robot has to deal with it. Finally, the robot needs to perform well several complex tasks as anticipation, deal with uncertainties, human-robot interaction, perception, prediction and autonomous navigation. When people interact in near distance, the proxemic rules must be used. Edward Hall coined the term proxemics for the studies of man’s use of the amount of space they occupy in daily life. According to him, one’s body is surrounded by ellipse-shaped bubbles. Each of these bubbles is appropriate for different social interactions. One of these zones, the personal space zone, acts as a virtual buffer zone around our body. When this buffer zone is invaded, people compensate for this intimate contact, by non-verbal or verbal compensation behaviors, such as stepping away, or limiting eye contact. All works agree that the proxemics rules between person and robot are similar to the rules between people.

A method to learn how to approach a specific person comfortably, without invading the personal space, is exposed in [15]; this study is based on an on-line learning algorithm that aims at generating a personalized approaching trajectory for each person in the considered environment. Indeed, since each person has a different personal space and behaves differently when another person approaches, for a service robot is not easy to approach a user in a socially acceptable manner without causing any discomfort. Moreover, this personal comfort field is unknown. Hence, in the technique exposed, the robot needs to explore regions nearby the user to search for a more comfortable approaching trajectory; encountering the same user multiple times, the robot learns user’s specific personal comfort space from user’s reactions, and approach the user more comfortably on the next encounter while avoiding uncomfortable encounters. In the end, an optimal personalized trajectory for the robot is found out gradually.

[16], instead, focuses on the question of the appropriateness of the robot’s approach behavior in different cultures. Not the study of robots approaching single people is carried out, but the case of small groups of people from different cultures is analyzed, trying to find out if a robot requires different spatial behavior depending upon the cultural background of its
users. Specifically, the optimal approach and placement position for a robot which is seeking to gain the attention of a small group of people is identified. Moreover, an on-line survey, distributed in China, U.S. and Argentina, shows that Chinese participants believed closer approaches were appropriate compared to their U.S. counterparts. Argentinian participants more closely resembled the ratings of the U.S. participants.

In [7] an adaptive side-by-side human-robot companion approach for navigation in urban dynamic environments, based on the Anticipative Kinodynamic Planning (AKP), is presented. The adaptive means that the robot is capable of adjusting its motion to the behavior of the person being accompanied, and the objective is to optimize in real time the path performed by the pair human-robot, by modifying dynamically the angle and distance between both throughout different locations of the path.

Moreover, a new cost function (denominated companion cost) for finding the best planned path that takes into account the cost of the geometrical configuration between the human and the robot is defined; the best path is obtained by minimizing all the cost functions in a RRT planner. The Extended Social Force Model (ESFM) has been modified too, in order to include the new force that maintains the side-by-side configuration while navigating. These new cost and new force are computed taking into account that the robot always has to adapt to the person’s trajectory decisions, and everything is done on-line and in real-time.

A step further has been made in [8], where the previous approach has been modified to allow the group to intercept a moving person, and to interact with it. The new model includes a framework to calculate a moving goal taken into account the movement of the person, the movement of the group and the best path to go through the obstacles of the environment. Then, the moving goal, which is recalculated in each iteration, has been included in ESFM, because now the robot has to deal with the changes of the environment and the two people (the companion and the approached) trajectory decisions.

The same aim of making robots capable of approaching, and the engaging task with a human-like behavior, has been developed in [9]. Here, the best encounter point between the human-robot group and the approached person has been computed using a gradient descent method, taking into account all people predictions. Then, in the encounter point the robot modifies its position to achieve an engagement with both people. The method reformulates the ESFM to include a dynamic goal, and the optimal destination minimizes
the time, the traveled distance, the effort to accomplish the companion task and the effort to avoid obstacles and pedestrians, working in real time.

1.2 Problem description and objective

The idea of having robots in an everyday life, and so in dynamic environments in which normal people spend their time is a challenging task, already faced. The still opened issue is having a robot behavior as much natural human-like as possible, in order to make it approach a person, looking at his direction. Therefore, the goal consists also in having a robot that doesn’t create disturbances to people in the environment, who have to walk and interact with it.

First, we obtain the candidate paths implementing an algorithm based on G2-splines computation. Then, the best path to reach the goal while avoiding collisions with all the obstacles, static and dynamic, is chosen by using a new cost function that takes into account the geometrical characteristics of the path, such as the length and the curvatures changing, but also the forces involved depending on the surrounding environment. Precisely, the best path is obtained by minimizing a cost function defined taking into account different criteria, and including the social forces present in the ESFM. The computation is done on-line and in real-time, in order to keep the environment constantly updated and to have always a feasible path for the robot.

1.3 Outline of the thesis

Chapter 2 presents the state of the art of the relevant subjects related with this project, specifically the previous works related to the planner, and the path computation algorithm used in the autonomous driving system. Other tools used for the development of the project are presented and a brief theoretical background on the robot modeling and control. Moreover, new contributions are exposed. In section 3 software and hardware used for running simulations and real experiments with the robot are presented; section 4 describes the implementation of a basic version of the algorithm in MATLAB, exposing also the first limitations found out, while the complete procedure is exposed in chapter 5. Here, the behavior of the planner in a wide variety of scenarios are presented and discussed. The real test results are reported. In the end, after the final conclusions and future possible works, in section 6, an appendix reports some details on implementation.
2

Theoretical background and new contributions

The previous contributions used in this work are exposed in this chapter together with new improvements and modifications made for our purpose, the approach. In the chapter of the simulations it will be clearer how we arrived to the final version of the algorithm reported in this chapter, but here for the general comprehension we limited to expose how the whole procedure works, using some parts already existing and adding our changes and contributions to have an algorithm suited to our case.

Since the main objective for a mobile robot is to achieve transportation from one point to another, once a route planning is generated, it is needed to provide the robot with a safe and collision-free path, while taking into account their dynamics and their manoeuvre capabilities in the presence of obstacles. This main idea will be at the base of the present work, in which a motion planning algorithm for a mobile robot in dynamic environments has been realized.

First of all the entire final version of the planning procedure for the approach is exposed, deriving from, as already said, both existing parts of algorithm and some improvements or modifications added to achieve the goal. Then the Social Force Model used and the background on splines are reported with their details. The new cost definition follows, and in the end a brief resume on robot model and control is reported.
2.1 AKP approaching planner

The AKP (Anticipative Kinodynamic real-time Planner) is a motion planner presented in [3] and implemented in [1] for autonomous on-road driving, and here modified and applied to a mobile robot for approaching a person. AKP planner can deal with usual driving manoeuvres, such as adapting the velocity to the road shape, follow lanes and also deal with obstacles. However, its biggest potential is its anticipative behavior with respect to other moving vehicles, or dynamic entities in our case. This reason makes it very interesting for navigating in urban scenarios, plenty of complex situations.

The AKP in [1] is thought to be working in a whole autonomous driving system, where the output of the planner is a path, that is a geometric trajectory that is fed to a low level controller that follows it precisely, and a velocity profile that the vehicle must follow. For this project, only the path generation task is considered.

Moreover, the planner in the autonomous driving system works discretizing the environment, choosing then several endpoints, which are state configurations. In our case, the only considered endpoint is the final goal position.

In addition, AKP planner is designed to be relaunched at high frequency, in order to be used as a real-time motion planner and to assure a high level of reactivity, and also our planner for the approaching works in real-time.

Before introducing the details of our planning algorithm for approaching, an assumption has to be highlighted: a global planner providing a valid global path to the goal unobstructed by static obstacles is a prior requirement, and our planner only provides, at each iteration, a locally valid path which minimizes perturbances and disturbances to the other entities on the scene. The whole planning procedure is resumed in the following figure:
2. THEORETICAL BACKGROUND AND NEW CONTRIBUTIONS

2.1. AKP approaching planner

Figure 2.1: Overview of the planning scheme

The algorithm for the approach calculates for each iteration some candidate paths to reach a goal and the correspondent best path, after a cost evaluation; an output action is executed, and in the next iteration, a new plan is calculated, and a new action is then executed, and so on. This approach permits a fast adaptation to changing environments, and the avoidance of obstacles, since it considers all the social forces present in the environment at each iteration, then the solution obtained is kinodynamic. Moreover, the planner uses prediction information about other entities in the environment.

Considering, for planning purposes, both robots and people as moving in a two-dimensional space, which represents the urban environment, here the state-space formulation is exposed.

Let $\mathcal{X}$ denote the workspace, and $x \in \mathcal{X}$ describe the position $x = [x, y]^T$ in a two dimensional space, for both moving objects, people and robots.

The configuration space $\mathcal{C}_z$ is defined as a configuration $q_z \in \mathcal{C}_z$, where $z$ denotes different configuration spaces for people and robots.

Since a kinodynamic treatment of the planning scheme is taken into account, the phase space $\mathcal{S}_z$ that only considers the first order derivative is defined, where $s'_z \in \mathcal{S}_z$ is given by $s'_z = [q_z, \dot{q}_z]^T$. In addition, strong time constraints are present: object movements alter the outcome of the planning calculations, and one should consider the augmented phase space $\mathcal{S}T_z = \mathcal{S}_z \times \text{time}$, where $\text{time} \in \mathbb{R}^+$, and $s_z \in \mathcal{S}T_z$ is a state $s_z = [q_z, \dot{q}_z, t]^T$ at time $t$. In general, the planning problem will be addressed using the state $s_z \in \mathcal{S}T_z$. Finally, an action $u_z \in \mathcal{U}_z$ modifies the states $s_z$, as it will be clear later.
People are treated as free moving particles, and therefore, no orientation is required. Accordingly, the configuration space is equal to the workspace $C_P = \mathcal{X}$ and the person’s phase space $ST_P$ is described as $s_p \in ST_P$, where $s_p = [x, y, v_x, v_y, t]^T$.

The input space $U_P$ for the person’s action is $u_p \in U_P$, which are linear accelerations $u_p = [a_x, a_y]^T$. The kinodynamic model describing the person’s motion is constrained by the following differential equations:

$$\dot{s}_p = dc(s_p, u_p) = \begin{bmatrix} v_x \\ v_y \\ a_x \\ a_y \\ 1 \end{bmatrix}$$ (2.1)

The robot model is considered characterized by a unicycle robot model. Thereby, non-holonomic constraints appear in the robotic dynamic model due to the rolling contacts between the rigid bodies. The phase state of the robot $ST_R$ is described as $s_r \in ST_R$, where $s_r = [x, y, \theta]^T$. The robot action space is defined as $u_r \in U_R$, where $u_r = [\omega, a_v]^T$ are the translation and rotation accelerations. Then, the resultant differential constraints are:

$$\dot{s}_r = dc(s_r, u_r) = \begin{bmatrix} v \cos \theta \\ v \sin \theta \\ \omega \\ a_v \\ a_w \\ 1 \end{bmatrix}$$ (2.2)

The joint state-space $ST$ consists of $ST = ST_R \times \bigcup ST_P$, which considers the robot phase space $ST_R$ and the union of every person’s phase space $ST_P$. Correspondingly, the joint state $s \in ST$ is defined as $s = [s_r, s_{p_1}, \ldots, s_{p_N}]$.

The variable time $t = t(s)$ is equal to all the states that $s$ consists of. The robot state will be $s_r(s) \in ST_R$ and the person $ith$ state $s_{p_i}(s) \in ST_P$.

The algorithm, summarized in 2.1, is now explained in details; at each iteration, a plan is computed and the output action $u_r(t)$ is executed. The input $s_{init} \in ST$ is the initial
Algorithm 2.1 AKP approaching($s_{ini}, \Delta t$)

1: Initialize $\mathcal{T}(\mathcal{V}, \mathcal{E}) \leftarrow \{\emptyset\}$
2: $\mathcal{V} \leftarrow s_{ini}$
3: $\{q_r^{goal}\} = \text{scene\_prediction()}$ \hfill ($N$ is the number of possible final positions)
4: $\{q_r^{goal}\} = \text{final\_destinations}(s_{ini}, N)$
5: for $j = 1$ to $N$ do
6:  $\{x\} = \text{splines\_generation}(s_{ini}, q_r^{goal,j})$
7:  for $i = 1$ to $K$ do \hfill ($K$ is the length of the spline)
8:  $\text{step}_{goal} \leftarrow x_{i+1}$
9:  $u_j = \text{calculate\_edge}(s_{ini}, \text{step}_{goal}, i, \text{dist}_{col})$ \hfill ($\text{dist}_{col}$ defines the radius of the circular area in which to consider the social forces)
10: end for
11: $J_j = \text{path\_cost\_computation}(\{x\}, u_{r,j})$ \hfill ($u_{r,j}$ is defined inside $u_j$)
12: end for
13: $\{[q_r^{goal}, J_{best}]\} = \text{best\_path\_selection}(\{J_1, \ldots, J_N\})$ \hfill ($q_r^{goal}$ is the chosen final desired robot goal)
14: $\{[\text{step}_{goal}, \text{index}]\} = \text{path\_optimization}(q_r^{goal})$
15: $u = \text{calculate\_edge}(s_{ini}, \text{step}_{goal}, \text{index}, \text{dist}_{col})$
16: $s_{new} = \text{robot\_propagation}(s_{ini}, u)$ \hfill ($u_{r}$ is defined inside $u$)
17: $s_{new} = \text{orientation\_adjusting}(s_{new})$
18: $\mathcal{V} \leftarrow \{s_{new}, J_{best}\}$
19: $\mathcal{E} \leftarrow \{u\}$
20: return $\text{branch}(\mathcal{T})$

state, containing the information of the robot state plus all people’s states considered on the scene, included the approached person. The algorithm builds a tree $\mathcal{T}(\mathcal{V}, \mathcal{E})$ consisting only of a branch, selected minimizing an adequate cost. The edge $\mathcal{E}$ represents the robot control inputs $u_r \in U_R$, and the vertex $\mathcal{V}$ consists of the joint state $s \in ST$ and the accumulated cost $J \in \mathbb{R}$ to reach that vertex.

The implementation of some parts of the algorithm will be discussed later, here the main procedure and the edge computation and propagation of the robot functions, already existing from previous works, are exposed for the comprehension.

After the computation of the candidate paths, using the splines algorithm, for each of them all the forces in the environment, affecting the robot along the path taken into account, have been calculated: to this purpose the calculate_edge() function has been used, considering the forces affecting each step of the path, from the actual position of the robot to the next sample of the path, fixed as stepgoal in Line 9 of Alg. 2.1. Then, after computing the cost for each path, and selecting the minimum one, another procedure,
path_optimization(), exposed later, is used to get the new value for the step_goal which is used in the second call of calculate_edge() in Line 15. This time the function is used to get again the input for the robot, in order to propagate it using the computed forces, but the calculation has been made not considering each sample of the path, but only some ones, which specific index is expressed by the output of the optimization of the path procedure. Finally, the orientation of the robot has been adjusted to have it perfectly looking at the approached person. All the details will be explained in Chapter 5.

Despite that the joint state takes into account all people, we only select the input actions for the robot platform, then, in the final edge the robot control input \( u_r \in U_r \) is executed. For this purpose, the resultant robot force \( f_r \) is computed by making use of the ESFM model, introduced in the following paragraph. This force is then transformed into an acceleration, according to:

\[
a_v = f_x \cos \theta_r + f_y \sin \theta_r
\]

\[
a_v = -f_x \sin \theta_r + f_y \cos \theta_r
\]

finding then the robot action \( u_r = [a_v, a_\omega] \). The goal of the robot is the final \( q_{goal} \) (Line 13 in Alg.2.1), defined after selecting the best path between the candidates; at the same time the robot reacts to the environment obstacles, that is, it takes into account the states of the nearby people \( s_{p1}, \ldots, s_{pN} \), included using scene_prediction(). Here the resume of the edge computation algorithm:

**Algorithm 2.2 Calculate edge**

1: function \text{Calculate edge}(s_{ini}, q_{goal}, \text{plan_index}, \text{dist_col})
2: \quad f_{goal} = \text{FORCE_Goal}(s_{ini}, q_{goal})
3: \quad f_{people} = \text{FORCE_PEOPLE_INTERACTIONS}(s_{ini}, \text{plan_index}, \text{dist_col})
4: \quad f_{obs} = \text{FORCE_OBSTACLES_INTERACTIONS}(s_{ini}, \text{plan_index}, \text{dist_col})
5: \quad \text{if } f_{obs_{x}} > f_{obs_{x}}^{\text{max}} \text{ then} \quad \text{end if}
6: \quad f_{obs_{x}} = f_{obs_{x}}^{\text{max}} \quad \triangleright \text{To limit the sum of the forces due to obstacles in x}
7: \quad \text{if } f_{obs_{y}} > f_{obs_{y}}^{\text{max}} \text{ then} \quad \text{end if}
8: \quad f_{obs_{y}} = f_{obs_{y}}^{\text{max}} \quad \triangleright \text{To limit the sum of the forces due to obstacles in y}
9: \quad f_{tot} = \alpha f_{goal} + \gamma f_{people} + \delta f_{obs} \quad \triangleright \text{To weight the single forces components}
10: \quad \text{return } u(\text{plan_{index}}, f_{tot}, f_{goal}, f_{people}, f_{obs})
11: \text{end function}
In Lines 5 and 8, two parameters are introduced for the maximum value of both the forces in \( x \) and \( y \) related to the obstacles; as it will be clearer in the discussion of the simulations, these will be fundamental to limit the sum of all the forces brought by all the obstacles present in the environment, in order to avoid to stop the robot, or have an unnatural behavior of it, due to the big amount of repulsive forces.

Once the forces are computed, the propagation of the robot (Line 16 of Alg. 2.1) provides the update of the robot state, giving as output the state propagation \( s_{\text{new}} \) from the initial state \( s_{\text{ini}} \in S^T \). The resume is showed in Alg. 2.3. First, the 2D forces are converted into robot forces with nonholonomic constraints and used for getting the acceleration components; then the velocities are updated and used to get the final position of the robot in the new state.

Algorithm 2.3 Robot propagation

```plaintext
1: function Robot propagation(\( s_{\text{ini}} \), \( u_T \), \( \Delta t \))
2: \( s_{\text{ini}} = [x_{t_i}, y_{t_i}, \theta_{t_i}, v_{t_i}, \omega_{t_i}, t_{i}]^T \)
3: for \( t = \Delta t, 2\Delta t, \ldots, t_n \)
4: \( a_{vt_i} = f_x \cos(\theta_{t_i}) + f_y \sin(\theta_{t_i}) \)
5: \( a_{\omega t_i} = -f_x \sin(\theta_{t_i}) + f_y \cos(\theta_{t_i}) \)
6: \( v_{t+1} = v_t + a_{vt_i} \cdot \Delta t \)
7: \( \omega_{t+1} = \omega_t + a_{\omega t_i} \cdot \Delta t \)
8: \( x_{t+1} = x_t + v_t \cos(\theta_{t_i}) \cdot \Delta t + a_{vt_i} \cos(\theta_{t_i}) \cdot \Delta t^2 / 2 \)
9: \( y_{t+1} = y_t + v_t \sin(\theta_{t_i}) \cdot \Delta t + a_{vt_i} \sin(\theta_{t_i}) \cdot \Delta t^2 / 2 \)
10: \( \theta_{t+1} = \theta_t + \omega_t \cdot \Delta t + a_{\omega t_i} \cdot \Delta t^2 / 2 \)
11: end for
12: return \( s_{\text{new}} = [x_{t_n}, y_{t_n}, \theta_{t_n}, v_{t_n}, \omega_{t_n}, t_n]^T \)
13: end function
```

2.2 Extended Social Force Model

In order to achieve a model capable of represent the interactions between pedestrians and robots the Social Force Model (SFM) introduced by Helbing, [5], has been used and extended for navigation purposes, [3], and in particular for obtaining a version which is suited for our robot Tibi, since it provides a realistic model describing human and human-robot interactions in typical social environments. The main idea is that the motion of pedestrians can be described as if they would be subject to ”social forces”; these ”forces” are not directly exerted by the pedestrians’ personal environment, but they are a measure for the internal mo-
tivations of the individuals to perform certain actions, or movements. Basing on the social forces concept, a powerful scheme for robot’s human-awareness navigation can be derived.

The Extended Social Force Model (ESFM) considers humans and robots as free particles in a 2D space, abiding the laws of Newtonian mechanics.

The ESFM uses attractors and repulsors in the continuous space. Formally, it is assumed that a pedestrian \( n \) tries to move at a certain desired speed \( v^0_n(q^{\text{goal}}_n) \) in a desired direction \( e_n \), i.e., with desired velocity vector \( \mathbf{v}^0_n(q^{\text{goal}}_n) = v^0_n(q^{\text{goal}}_n) e_n \).

Assuming that the pedestrian tries to adapt his or her velocity within a relaxation time \( k^{-1} \), the attraction force to reach the goal \( f^{\text{goal}}_n \) is given by:

\[
f^{\text{goal}}_n(q^{\text{goal}}_n) = k(\mathbf{v}^0_n(q^{\text{goal}}_n) - \mathbf{v}_n) \quad (2.5)
\]

where \( \mathbf{v}_n \) is the current velocity.

A person wants to keep his desired velocity through the steering force, \( f^{\text{goal}}_n \), but is also influenced by other pedestrians \( p_j \), \( f_{n,j}^{\text{int}} \), by obstacles, \( f_{n,o}^{\text{int}} \), or by robots, \( f_{n,r}^{\text{int}} \).

The resulting force \( f_n \) governing the trajectory described by the target \( p_n \), is equal to:

\[
f_n = f^{\text{goal}}_n(q^{\text{goal}}_n) + \sum_{j \in P \setminus n} f_{n,j}^{\text{int}} + \sum_{o \in O} f_{n,o}^{\text{int}} + \sum_{r \in R} f_{n,r}^{\text{int}} \quad (2.6)
\]

where, \( P \) is the set of people moving in the environment where the human interacts, \( O \) is the set of obstacles, and \( R \) the set of robots.

Each repulsive force prevents the entity from walking along their intended direction, and, moreover, each interaction force is modeled as:

\[
f_{n,z}^{\text{int}} = A_z e^{(d_z - d_{n,z})/B_z} \frac{d_{n,z}}{d_{n,z}} w(\varphi_{n,z}, \lambda_{n,z}) \quad (2.7)
\]

where \( z \in P \cup O \cup R \) is either a person, an object of the environment, or a robot. \( A_z \) and \( B_z \) denote respectively strength and range of the interaction force. In order to calculate the Euclidean distance between \( p_n \) and the entity \( z \), humans and objects are assumed to be of circular shape with radii \( r_n \) and \( r_z \). Then, \( d_z \) is the sum of the radii of the pedestrian and an entity, and \( d_{n,z} \equiv r_n - r_z \). Moreover, given the limited field of view of humans, influences might not be isotropic. This justifies the presence of a scaling factor, \( w(\varphi_{n,z}, \lambda_{n,z}) \), which is
an anisotropic factor depending on \( \varphi_{n,z} \) between \( v_n \) and \( d_{n,z} \):

\[
w(\varphi_{n,z}, \lambda_{n,z}) = \left( \lambda_{n,z} + (1 - \lambda_{n,z}) \frac{1 + \cos(\varphi_{n,z})}{2} \right)
\]

(2.8)

where \( \lambda_{n,z} \) defines the strength of the anisotropic factor,

\[
\cos(\varphi_{n,z}) = -n_{n,z} \cdot e_r
\]

(2.9)

The term \( n_{n,z} \) is the normalized vector pointing from \( z \) to person \( p_n \) which describes the direction of the force.

All the parameters \( \{k, A_z, B_z, \lambda_{n,z}, d_{n,z}\} \) are defined depending on the nature of the object, and interaction.

For what concerns the robot, nonholonomic constraints reduce its platform mobility, although it has full reachability in \( C_R \). The resultant robot force is given by:

\[
f_r = \alpha f_{r,\text{goal}} + \gamma \sum_{j \in P} f_{r,j}^{\text{int}} + \delta \sum_{o \in O} f_{r,o}^{\text{int}}
\]

(2.10)

where the repulsive effects from the influences of other people and obstacles are described by an interaction force, given by the sum of forces each person or obstacle introduces in the environment. These are modeled as:

\[
f_{r,j}^{\text{int}} = A_{r,p} e^{(d_{r,p} - d_{r,j})/B_{r,p}} w(\varphi_{r,j}, \lambda_{r,p})
\]

(2.11)

\[
f_{r,o}^{\text{int}} = A_{r,o} e^{(d_{r,o} - d_{r,o})/B_{r,o}} w(\varphi_{r,o}, \lambda_{r,o})
\]

(2.12)

using the specific parameters \( \{A_{r,p}, B_{r,p}, \lambda_{r,p}, d_{r,p}\} \) \( \{A_{r,o}, B_{r,o}, \lambda_{r,o}, d_{r,o}\} \) defined depending on the interaction. Moreover, the force to the target’s destination is inferred by:

\[
f_{r,\text{dest}}^{\text{goal}} = k_\tau (v_r^0 - v_r)
\]

(2.13)

In equation (2.10) some parameters \( \{\alpha, \gamma, \delta\} \) compare, but these will be discussed later in the chapter dedicated to the simulations.
2.3 Path generation

Some of the most popular existing motion planning algorithms are based on the main idea of discretizing somehow the space and search for the best solution; for example, the most popular techniques used for on-road autonomous driving are the lattice planers, where the planner uses a limited horizon both in terms of time and space, and the search space contains a certain geometric curve. Lattice planners need to generate candidate geometric paths that will be followed by a low level control system. In addition, kinodynamic constraints are mandatory to achieve good control performance and also to assure that paths can be exactly followed. More specifically, these constraints are related with the geometric continuity ($G^m$) of the path.

The most popular used techniques in path generation use arcs, polynomial spirals and splines. They differ in the number of parameters and degrees of freedom, and consequently, in the level of complexity.

Generating a path from one state to another and satisfying the mentioned kinodynamic constraints is not an easy problem, and some solutions can be very complex and may need a numerical optimization algorithm. The complexity of the generation derives also from the road shape; for this reason, a very useful property for path generation algorithms is the correlation between shape and parameters: if the parameters of the path correspond to its shape on an intuitive level.

In our case, the paths generated by the algorithm are based on $G^2$-splines, used as mathematical tool. This type of geometric splines presents very nice properties, as it will be clearer later, even if it requests to find some parameters, which might be adjusted to have the best possible trajectory.

In order to build a general path $p(u)$, understood as a path with arbitrary defined starting and ending states, the quintic $G^2$-splines offers flexibility and very good properties. They are geometric polynomials of 5th order, with second order geometric continuity ($G^2$), so that the curvature $\kappa$ is continuous. The needed information to generate a path is only related to the initial and final states, and it’s showed in figure 2.2.
2. THEORETICAL BACKGROUND AND NEW CONTRIBUTIONS

2.3. Path generation

Figure 2.2: Example of a $G^2$-spline path connecting $P_A$ and $P_B$ states

Then, considering the initial state $X_A = [x_A, y_A, \theta_A, \kappa_A]$, as well as the ending state $X_B = [x_B, y_B, \theta_B, \kappa_B]$, the equations to define this type of splines are the following:

$$
p(u) = \begin{bmatrix} x(u) \\ y(u) \end{bmatrix} = \begin{bmatrix} x_0 + x_1 u + x_2 u^2 + x_3 u^3 + x_4 u^4 + x_5 u^5 \\ y_0 + y_1 u + y_2 u^2 + y_3 u^3 + y_4 u^4 + y_5 u^5 \end{bmatrix}
$$

where $u \in [0, 1]$ and:

$$
x_0 = x_A \\
x_1 = \eta_1 \cos \theta_A \\
x_2 = \frac{1}{2}(\eta_1 \cos \theta_A - \eta_2^2 \kappa_A \sin \theta_A) \\
x_3 = 10(x_B - x_A) - (6\eta_1 + 2\eta_2) \cos \theta_A \\
x_4 = -(4\eta_2 - \frac{1}{2}\eta_3) \cos \theta_B + \frac{3}{2}\eta_3^2 \kappa_A \sin \theta_A - \frac{3}{2}\eta_3^2 \kappa_B \sin \theta_B \\
x_5 = 6(x_B - x_A) - (3\eta_1 + \frac{3}{2}\eta_2) \cos \theta_A \\
\eta_2 = \frac{1}{2}(\eta_3 \sin \theta_B - \eta_2^2 \kappa_A \cos \theta_A) \\
\eta_3 = 10(y_B - y_A) - (6\eta_1 + 2\eta_2) \sin \theta_A \\
\eta_4 = -(4\eta_2 - \frac{1}{2}\eta_3) \sin \theta_B + \frac{3}{2}\eta_3^2 \kappa_A \cos \theta_A + \frac{3}{2}\eta_3^2 \kappa_B \cos \theta_B \\
\eta_5 = 6(y_B - y_A) - (3\eta_1 + \frac{3}{2}\eta_2) \sin \theta_A \\

The resulting spline depends on some parameters that are free, and only affect the spline shape. These are all included in the vector:

$$
\eta = [\eta_1, \eta_2, \eta_3, \eta_4]
$$

In order to generate any kind of path it is necessary to find certain values for the elements of
this vector. Solutions can be obtained through a numerical optimization, but to get a more
general one, without a numerical optimization process and valid for any situation, one can
discuss the behavior of the path while changing the values.

To have a symmetrical behavior, \( \eta \) parameters need to be \( \eta_1 = \eta_2 \) and \( \eta_3 = -\eta_4 \), so only
two parameters must be tuned: \( \eta_{12} = \eta_1 = \eta_2 \) and \( \eta_{34} = \eta_3 = -\eta_4 \).
\( \eta_{34} \in (-\infty, +\infty) \), affects the curvature changes, and it has been set equal to 0, in order to
have a smooth and balanced trajectory. \( \eta_{12} \in (0, +\infty) \), and it generally forces \( \theta \) and \( \kappa \) to
stay close to the initial and final values; the smoothest trajectory is reached when \( \eta_{12} \) is close
to the trajectory length, in meters. In order to converge into the best possible trajectory, a
simple iterative method is performed: firstly, the Euclidean distance between the initial and
ending points is taken as the initialization value for \( \eta_{12} \). Then, the trajectory is computed and
the total length of this new trajectory is used as the new \( \eta_{12} \) to compute the next iteration.
After few iterations, 3 or 4 depending on the case, all the computed trajectories converge into
the smoothest one. This optimization process has the same results as the numerical optimiza-
tion, which only minimizes the curvature variability \((\min_{\eta} \frac{d\kappa}{du})\).

Moreover, an important characteristic of the resulting spline is its curvature \( \kappa(u) \), which
can be computed using the following equation:

\[
\kappa(u) = \frac{\dot{x}(u)\ddot{y}(u) - \dot{y}(u)\ddot{x}(u)}{(\dot{x}(u)^2 + \dot{y}(u)^2)^{3/2}}
\]

It is important to underline that this algorithm can approximate any kind of path shape,
preserving a smooth curvature continuity and minimizing curvature variability. This implies
kinematic feasibility for a vehicle, or robot, following it. As well, this algorithm generates the
candidate paths only from the geometrical information of the initial and final points, and this
means that these splines can follow a possible road shape very easy and intuitively.

2.4 Multi-cost definition

In the planning procedure resume it has been highlighted that a cost is computed for each
path in order to choose the best one as the one related to the minimum cost.

Finding a cost function to correctly characterize robot navigation among people is not
an easy task, and to deal with the minimization problem a Multi-Objective Optimization
has been proposed in [6], and utilized and modified here, in order to choose the best path
considering different independent criteria. Indeed, since in dynamic environments scenarios are several, and there are multiple objectives to be minimized, instead of a single-objective cost, a multi-objective optimization suits better.

The aim is to obtain a navigation algorithm that considers different cost functions: for each step of the trajectory, different costs related to different criteria are computed; then after a normalization of these, they are summed in order to get a final cost related to a specified criteria for the entire path. The details will be explained in Chapter 4, but here the general cost definitions are presented.

The first considered cost is the one to reach a goal position. This is defined along the path, then, as the difference between the actual position of the robot and the one in the next sample of the path:

$$J_{d,i}(S) = ||x_{r, i}^{i+1} - x_{r, i}^{i}||^2$$ (2.14)

where the final cost represents the accumulated square distance to the final desired goal location. Then, the orientation cost, expressing the difference between the orientation in the next sample of the path and the current one, is derived as:

$$J_{or,i}(S) = ||\theta_{r, i}^{i+1} - \theta_{r, i}^{i}||^2$$ (2.15)

representing the accumulated difference of orientation to the final desired goal orientation.

Additionally, from the results of the simulations we decided to include, as it will be explained in Chapter 5, some costs strictly related to the environment surrounding the robot, that then consider the social forces. Therefore, the cost associated to the pedestrians interacting with the robot at each step, \(J_{p,i}\), is defined as:

$$J_{p,i}(U) = \sum_j ||f_{r,p,j}^{int}||^2$$ (2.16)

where the inputs \(u_{p,j}\) are due to the people influence to the robot, while it walks towards his goal, and then related to the interaction forces, as already pointed out. The cost produced by nearby obstacles at each step \(J_{o,i}\) is defined as:

$$J_{obs,i}(U) = \sum_j ||f_{r,obs,j}||^2$$ (2.17)

where, again, the perturbations to the robot as a result of nearby obstacles \(u_{obs,j}\) are consid-
ered to avoid collisions.

Since the main purpose is to combine these cost functions into a well-posed function, multi-objective optimization techniques are used to solve the problem of finding the best solution path. Each cost related to each step is normalized, according to:

\[ J_{k,i}(X) = erf \left( \frac{x - \mu_x}{\sigma_x} \right) \]  

where \( k \) and \( i \) mean that this is valid for all the costs for each step.

The different cost functions \( J_d(S), J_{or}(S), J_p(U), J_{obs}(U) \) for a path are derived as the accumulated costs throughout the path, where each cost has been correctly normalized and combined according to the \( w \) parameters:

\[ J_d(S) = \sum_i w_{d,i} J_{d,i}(S) \]  

\[ J_{or}(S) = \sum_i w_{or,i} J_{or,i}(S) \]  

\[ J_p(U) = \sum_i w_{p,i} J_{p,i}(U) \]  

\[ J_{obs}(U) = \sum_i w_{obs,i} J_{obs,i}(U) \]  

and, in the end, the final cost for the single path has been computed as:

\[ J_{path}(S, U) = J_d(S) + J_{or}(S) + J_p(U) + J_{obs}(U) \]

An interesting consequence of the proposed normalized weight-sum is that the navigation algorithm presents similar performance for very different scenarios, meaning that all the simulations will be carried out without changing the values of the parameters \( w = [w_d, w_{or}, w_p, w_{obs}] \) that have been derived according to a learning procedure exposed in [6].

2.5 Robot overview

This section introduces an overview of the robot with its kinematics analysis, and the behavior control, in order to better understand the implementation in MATLAB and Simulink,
reported in Chapter 4 and 7 respectively.

A mobile robot is a platform with a large mobility within its environment (air, land, underwater) since it is not fixed to one physical location. It has a potential application in industrial and domestic applications, but accurate designing and control of mobile robot is not a simple task, by the moment that it is essentially time-variant and the operation parameters related to robot, environment and road conditions are always varying. Therefore, the mobile robot, as whole including controller, should be designed to make the system robust and adaptive, improving the system on both dynamic and steady state performances.

One of the simplest and most used structures in mobile robotics applications are the two-wheel differential drive mobile robots, reported in Figure 2.3.

The model for motion generation of differential-drive mobile robots takes into account both the robot kinematic and dynamic constraints. The kinematic part concerns the study of the mathematics of motion without considering the forces that affect it, and deals with the geometric relationships that govern the system, with the relationship between control parameters and behavior of a system in state space. Kinematics is, basically, the most study of how mechanical systems behave. Moreover, in mobile robotics, we need to understand the mechanical behavior of the robot, in order to both design appropriate mobile robots for tasks and understand how to create control software for an instance of mobile robot hardware.

A mobile robot’s controllability defines possible paths and trajectories in its workspace (that defines the range of possible poses that the mobile robot can achieve in its environment). Robot’s dynamics places additional constraints on workspace and trajectory, due to mass and force considerations.

Since measuring a mobile robot’s position precisely is an extremely challenging task, and there is no direct way to measure a mobile robot’s position instantaneously, the process of understanding the motions of a robot begins with the process of describing the contribution each wheel provides to the motion, and then each constraint that the wheel imposes on the robot’s motion.

2.5.1 Robot model

The robot consists of a chassis with two drive wheels mounted on a common axis, and each wheel can independently being driven either forward or backward. The expression of robot motion in a global reference frame as well as the robot’s local reference frame is now introduced and used to construct a simple forward kinematic model.
of motion, describing how the robot, as a whole, moves as a function of its geometry and individual wheel behavior.

The total dimensionality of the robot chassis on the plane is three, two for position and one for orientation along the vertical axis, which is orthogonal to the plane as shown in Figure 2.3. Then the position of the central point P along the axis in the global reference frame is specified by coordinates \( x \) and \( y \), and the angular difference between the global and local reference frames \( \theta \). Therefore the vector representing the pose of the robot with respect to the global reference frame is expressed as:

\[
q_I = [x \ y \ \theta]^T
\]

To describe robot motion in terms of component motions, it will be necessary to map motion along the axes of the global reference frame to motion along the axes of the robot’s local reference frame. The mapping is accomplished with the following formula:

\[
\dot{q}_R = R(\theta)\dot{q}_I \tag{2.24}
\]
where it is used the orthogonal rotation matrix:

\[
R(\theta) = \begin{bmatrix}
\cos \theta & \sin \theta & 0 \\
-\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

which is a function of the current pose of the robot, since it depends on \( \theta \), and which expresses the motion in the global reference frame \( X_I, Y_I \) in terms of the local reference frame \( X_R, Y_R \).

### 2.5.2 Forward KINEMATICS

The kinematics of a mobile robot concerns the study of how the robot moves given its geometry and the speeds of its wheels. Given their radii \( r \), a point \( P \) centered between the two drive wheels is at a distance \( L/2 \) from each wheel. Therefore, given \( r, L/2, \theta \) and the spinning speed of each wheel, \( v_R \) and \( v_L \), a forward kinematic model would predict the robot’s overall speed in the global reference frame:

\[
\dot{\mathbf{q}}_I = [\dot{x} \quad \dot{y} \quad \dot{\theta}]^T
\]

By varying the velocities of the two wheels, we can vary the trajectories that the robot takes.

From Equation 2.24 it is possible to compute the robot’s motion in the global reference frame from motion in its local reference frame:

\[
\dot{\mathbf{q}}_I = R(\theta)^{-1}\dot{\mathbf{q}}_R \tag{2.25}
\]

Therefore, considering independently the contribution of each of the two wheels in the local reference frame and the following relations:

\[
v = \frac{v_R + v_L}{2}
\]

\[
\omega = \frac{\omega_R + \omega_L}{L}
\]

\[
v_R = w_R \cdot r
\]

\[
v_L = w_L \cdot r
\]
for the right wheel the forward spin results in counter-clockwise rotation at point \( P \), so the rotation velocity \( \omega_R \) at \( P \) is computed by:

\[
v_R = \omega(r + L/2)
\]  

(2.26)

The same calculation applies to the left wheel, with the exception that forward spin results in clockwise rotation at point \( P \), then:

\[
v_L = \omega(r - L/2)
\]  

(2.27)

where we remind that \( v_R, v_L \) are the right and left wheel velocities along the ground. Therefore, specified the linear and angular velocities of the robot, \( v \) and \( \omega \):

\[
\omega_R = \frac{v + L/2 \cdot \omega}{r}
\]  

(2.28)

\[
\omega_L = \frac{v - L/2 \cdot \omega}{r}
\]  

(2.29)

And rewriting in a matrix formulation:

\[
\dot{q} = \begin{bmatrix}
\dot{x} \\
\dot{y} \\
\dot{\theta}
\end{bmatrix} = \begin{bmatrix}
-sin \theta & 0 \\
cos \theta & 0 \\
0 & 1
\end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}
\]  

(2.30)

one can substitute \( v \) and \( \omega \) obtaining the final Jacobian for the differential drive:

\[
\dot{q} = \begin{bmatrix}
\dot{x} \\
\dot{y} \\
\dot{\theta}
\end{bmatrix} = \begin{bmatrix}
-(r \sin \theta)/2 & -(r \sin \theta)/2 \\
(r \cos \theta)/2 & (r \cos \theta)/2 \\
-r/L & r/L
\end{bmatrix} \begin{bmatrix} \omega_L \\ \omega_R \end{bmatrix}
\]  

(2.31)

### 2.5.3 Trajectory tracking control

Giving the linear and angular velocities to the robot, it is supposed that this will follow a precise path, computed according to the algorithm introduced in chapter ???. Here, the path is defined as some locus on the \( x-y \) plane, therefore consider the problem of moving the robot along the path. We can consider the robot as aiming at a reference point \((x^*, y^*)\) which
is moving.

The main idea is to control the robot velocity to be proportional to its distance from the
goal, considering also that we wish to maintain a given distance \( d^* \) behind the pursuit point. Given the actual position coordinates \( x \) and \( y \) of the robot, and the aimed point, the error defined as:

\[
e = \sqrt{(x^* - x)^2 + (y^* - y)^2 - d^*}
\]  \hspace{1cm} (2.32)

is then the input of a PI controller for the regulation of the velocity:

\[
v = K_v \cdot e + K_i \int e dt
\]  \hspace{1cm} (2.33)

The objective angle is not assigned, but is set according to:

\[
\theta^* = \text{atan2}(y^* - y, x^* - x)
\]  \hspace{1cm} (2.34)

and controlled by a proportional controller:

\[
\gamma = K_h (\theta^* \ominus \theta)
\]  \hspace{1cm} (2.35)

Note that the operator \( \ominus \) computes the angular difference to keep the angle between \([ -\pi, \pi ]\).

In Appendix 7 the overall system implementation in Simulink is reported, and in particular the details of the wheel velocities and pose computation blocks.
3

Hardware and Software description

This chapter contains a brief explanation of the software and hardware used. In this hardware section, it is explained the hardware used in the Tibi (and Dabo) robots, figure 3.1, but only Tibi will be used for this work.

The software section has a brief explanation of the ROS (Robot Operating System), software which is used in our robots, and a brief explanation of some ROS nodes.

As social robots, Tibi is meant to interact with people using the following elements: a touchscreen, speaker, movable arms and head, and LED illuminated face expressions. Power is supplied by two sets of batteries, one for the Segway platform and one for the computers and sensors, giving about five hours of full working autonomy. Two on-board computers (Intel Core 2 Quad CPU @ 2.66 and 3.00 GHz with 4 GB RAM) manage all the running processes and sensor signals. An external laptop (Intel Core i5-2430M @ 2.40 and 3.00 GHz with 4 GB RAM) is used for external monitoring. As Operating System Ubuntu (first version 12.04, and 14.04) is used; and as middle ware the Robot Operating System (ROS) [Quigley et al., 2009] is used, a software environment for robot system integration that provides a useful and large set of libraries and tools.
3.1 Hardware of Tibi

For the experiments, we used the mobile service robot Tibi, which was created during the URUS project [Sanfeliu et al., 2010, Trulls et al., 2011] to work in urban pedestrian areas, and to interact with people. It has a two-wheeled Segway RMP200 platform as base, which can work as an inverted pendulum in constant balancing, can rotate on the spot (nonholonomic), and it has wheel encoders, providing odometry, and inclinometers, providing pitch and roll data. Although that, for the experiments two additional wheels in front and rear the platform are incorporated to maintain the platform always vertical.

The necessary sensors are: two horizontal laser scans, the eye cameras, or the LADYBUG2, and the robot odometry in the segway.

3.1.1 Horizontal sensors

The goal of the sensors is to interact with people mainly through detection and tracking, and to interact with the environment through navigation, mapping, and collision avoidance.

To perceive the environment, Tibi is equipped with two (Front: UTM-30LX, Rear: UTM-30LX-EW) laser range sensors used to detect obstacles and people, giving scans over a local horizontal plane at 40 cm above the ground, facing forward and backward. One is located in the front, and the other one in the rear of the robot, as is showed in figures 3.2a and 3.2b. These sensors facilitate location and mapping through wall detection, allowing navigation and obstacle avoidance, as well as the detection of people. The laser UTM-30LX-EW is an area laser scan sensor with a wavelength of $\lambda = 905$ nm and with security class 1. The laser
allows the scanning of a semi circle of 270°, divided in steps of 0.25° (360 degrees/1, 440 steps) of resolution, this allows to obtain a resolution in the detection of 1 mm respect to the real position of it, in distances between 0.1 m and 30 m. Also, the laser can detect up until 60 m but with less accuracy. The minimum detectable amplitude of the laser at 10 m of distance is of 130 mm. The scan, reported in figure 3.3, has a velocity of 25 msec/scan (Motor speed: 2400 rpm). The laser returns the distance of each detected obstacle between its scan range. The diameter of the laser beam is 400 mm at the 30 m distance. The laser scan precision was of ±30 mm inside the range from 0.1 m to 10 m, and of ±50 mm inside the range from 10 m to 30 m. We can find the data sheets of these sensors in [20] and [21].

3.1.2 Eye cameras (Bumblebeez)

As vision sensor Tibi uses a Bumblebee 2 stereo camera located in the eyes. The camera will serve to identify the required images to a specific person. The most relevant data for the image detector is: the images have 640 × 480 pixels. The camera uses the color RGB compression
format and the frame rate is the maximum that the camera can give us, 7.5 FPS. Furthermore, the image processing is made by the camera itself. We can find the data sheet of the sensor in [22].

![Figure 3.3: Range scan of the horizontal laser](image)

3.1.3 LADYBUG2

The Ladybug2 has six 0.8 MP cameras which enable video perception of 360°. Its six cameras are closely-packed Sony 1024 × 768 CCDs placed within 20 mm of each other. With high quality 2.4 mm micro-lenses enabled to collect video from more than 75% of the full sphere. We can obtain videos at 30 FPS. We can have multiple image output formats: JPEG, BMP, PNG, TIFF and more. Also we can have multiple video output formats: AVI, MP4 and FLV. We can obtain more information about this software in the official page [23]. With this camera we obtain a vision detection in all of the 360° and with it we can fuse with the laser detection, and obtain a more robust track of any person in the robot’s environment.

3.1.4 SEGWAY (RMP 200)

The two-wheeled Segway, which use Tibi, provides great mobility. This Seqway allows them to move freely through the same places as people. This Segway also provides us with some in-
formation about the movement of the robot. This information is sent by the status message of the Segway, which is sent each 0.01 sec. And in the case of the odometry, we only want to know the the displacement from the before robot position to the new robot position and the velocity of this displacement, both data is obtained from the two wheels of the Segway. This displacement and velocity is used to obtain the position odometry and the linear velocity of the odometry. Furthermore, we want to know the three angles of orientation: roll, pitch and yaw, to obtain the angular velocity of the odometry. We can find the user guide of the Segway in [25] and some specifications in [24].

3.2 Software

Throughout the implementation of the planning algorithm, MATLAB, C++, and ROS were used. First, the adequate behavior of the equations and a basic version of the algorithm are tested using MATLAB. Then, the algorithm has been improved in C++, making the libraries that the node uses. Finally, the algorithm performed in C++ has been encapsulated in the ROS node, so it can be interpreted and used by the robot. For security, before the direct test of the code in the robot, we simulate the robot configuration in the experiments using
rosbags, our node, the RViz visualization tool. Each of these tools will be better explained later. The last step is the real experiments with the robot, to validate the appropriate behavior of the algorithm. Given the widespread knowledge of Matlab and C++, the focus in the remainder of this chapter will be given to ROS (Robot Operating System), since this tool is being used more and more in the context of robotic investigation.

3.2.1 Robot Operating System (ROS)

ROS is an open-source Software that can run in any operative system and robotic hard-ware. ROS provides libraries and tools to develop applications for robots. The system is based on a modular concept, that consists in dividing a complex task in simpler and reduced sub-tasks, these subtasks are encapsulated in executable files called nodes, which communicate amongst each other to generate the final behavior of the robot. Each node has a unique name that distinguishes it from the rest of the existent nodes. The nodes can communicate with one another through three different communication procedures: publishing or subscribing it to a topic, providing or using one service, or using actions. One robot can have many independent nodes working in a cooperative manner with the global goal to achieve a complex behavior of the robot. With this modularity we obtain an important advantage that is a easy error correction, due to the fact that we can localize and resolve easily the errors inside small functions, reducing the complexity in comparison with the monolithic codes. Also, ROS is used since it allows the sharing of information between all kind of nodes, and it is possible to use previously developed processes and driver can easily be encapsulated in the nodes. Furthermore, it offers powerful tools for debugging, which saves time and allows the programmer to correct any errors found. Also, it offers the RViz that is a visualization tool which allows to observe the behavior that has been implemented in the programs before these are tried on the actual robot, this also saves time.

As we already presented in the previous paragraph, there are three communication protocols that the nodes use to communicate in ROS:

Topics: These are data busses that are used by nodes to exchange messages among them. The nodes can be of two different types: Publishers or Subscribers. The publishers generate data of one topic, for example the nodes that corresponds to the encapsulated drivers of the sensors are publishers that publish messages that contain the sensed values. The subscribers are nodes that are subscribed to the topics that publish another node. All the nodes can be at
the same time publishers and subscribers of different topics, also we can also have multiple publishers and subscribers of the same topic. The nodes can be subscribed or can published topics in anonymous form, therefore, the production of information is independent of its use. In general, nodes do not know with whom they are communicating. The unit that performs communication and knows whether nodes are published or subscribed to a topic is the roscore.

**Services:** These allows communication between nodes that have requests and answers. One service is defined by two types of messages, one for the request and other for the answers. In these situations one node takes up the role of the client and sends the request to obtain a service, and waits until the server node sends the answer.

**Actions:** These are based on the same principle as the services, a request is sent and a response is received. The difference is that the action adds the ability to cancel the service, and therefore the nodes do not need to wait until it gets the answer. An action is defined by three messages: goal, feedback and result. The first contains the reason of the request, the second periodically provides the information of the state of the system, and the last is the result of the request.

**Rosbags:** These are data files where the publisher messages are stored, in these files we only save the information of the publisher nodes that we need to replay the real robot experiment offline in the computer. This data can be stored in topics of one node that performs a complex behavior of the robot or can be only nodes that encapsulate one driver of one sensor and return the sensed data. These files allow us to recreate real situations offline and provide the option to verify or improve our created algorithms. Therefore, the rosbags may be designated for two functions: use our node in the environment RViz of our computer to finish the debug of some errors, or record the experiments on the robot to analyze them further.

**RViz:** A 3D graphical visualization environment for ROS. This graphical visualization environment allows us to draw the sensor outputs and create markers that show the output variables of our node, we can show only the parameters that we want to see. Also, It allows us to draw a robotic model at the point where the robot is located in the experiments, as well as, several coordinate axes that allows to know where are the position of each sensor and how the sensor is oriented in the robot. Finally, we can move the camera within the graphical
environment to see different views of the simulation, so we can do a thorough check of the satisfactory behavior of our programmed node.

The procedure to check the nodes or to see the good results of the nodes consists in: when the node is completed and ready to be tested, we create an executable file where we include the commands to run all the nodes that are needed, including our created node. In this executable we include the commands to run the rosbag file previously recorded, to replay the real robot experiments. And finally we give the commands to activate the RViz.

In Fig. 3.7 the environment map used for simulations is shown. Obstacles are black, free space is light grey and unknown space is dark grey. The cells are squares of around 1m × 1m. Here, also some computed paths are visible: the darker one is the best path selected, the blue marker indicates the robot actual goal, while the blue line shows the Euclidean distance between robot and final destination, at the actual iteration of the planner. The approached person is represented in green, and an id number identifies it. The black and red points that form lines are the laser detections of the walls.

![Figure 3.7: RViz 3D graphical visualization environment for ROS with the computed paths](image)

Dynamic-reconfigure: The dynamic reconfigure is a package which provides a means to change node parameters at any time without having to restart the node. In Figure 3.8 you can see the dynamic reconfigure for the planner case, in it all the modifiable parameters of the planner can be easily changed.

Node relations for local planning

In Figure 3.9 we can see the implemented planner and the relation between this and the nodes
it uses, as well as, the topics that these nodes use to communicate. The nodes that we can see in the figure are: the laser people detector, \textit{lpd}, and the laser people map filter, \textit{lpmf}, that is a filter that removes the laser people detections in the wall when we have a map. Then, this gives its detection messages to the tracker, \textit{mht}. The tracker node processes these detections and obtains the tracks. The tracks marker messages are given to the planner implemented in this work, and its messages are finally given to the RViz, for the visualization.

### 3.2.2 ROS nodes that the planner uses

In this subsection, we explain briefly the different ROS nodes implemented by other people that the local planner uses. First, the detector node used to obtain the position of the detected people in each moment and the laser people map filter are explained. Then, the tracker used to know all the positions and covariances of all tracked people in the environment is introduced. Furthermore, we explain the node used to make the prediction of each track. This node allows the use of the linear prediction with constant velocity and other predictions, which takes into account the people behavior. Finally, we briefly explain the odometry node, which is used to obtain the robot odometry.
Figure 3.9: The purple area indicates the planner, in green the laser people detector and filter, in blue the tracker, in red the odometry node. Next to the planner the RViz node for visualization. The other names are topics that leave or enter in the nodes, according with the direction of the arrow.

**Laser people detector and map filtered**

This detector is able to detect people using the shape produced by a person’s legs, when they are detected by the horizontal laser. The laser pattern corresponding to the detected legs appears as two semicircles very close to one another. Also, in two consecutive detections people can not travel long distances. These nodes, `/tibi/lpd_front` and `/tibi/lpd_rear`, search and identify the pattern using certain features, such as: the number of points that are associated with the detection of the person, the standard deviation, the distance between one detection and the detection in the next instant, the circularity, the radius, the curvature, the average speed, etc. We can see the mentioned pattern in the Figure 3.10. The theory of this detector can be found in [26].

The messages are sent to the laser people map filtered, `/tibi/lpmf`, which is a node that gets all the laser detections and remove the ones at less distance from the wall than a specific threshold, that is usually set equal to 0.2m or similar.

Finally, messages pass to the tracker, that finds the positions and covariances of all the possible people detections. These positions are located in an x-y plane which corresponds to the floor.
3. HARDWARE AND SOFTWARE DESCRIPTION

3.2. Software

![Figure 3.10: Laser leg detection](image)

**Multi-hypothesis tracker**

The tracker is a multi-hypothesis tracker [28], in Fig. 3.9 `/tibi/mht`. To be used in planning, the tracker uses the filtered people leg detections to differentiate and track all the people in the environment. This tracker uses as the backbone of the theory the multi-hypothesis tracker approach proposed by Reid [29]. Where they developed a multi-hypothesis strategy to associate tracks with detections using a Mahalanobis distance and they propagate and update the associated tracks using a Kalman filter. The algorithm was modified to add a better control of the confirmation and deletion of the tracks. Also, the algorithm was improved by using the people’s velocities to improve the association between tracks and detections in crossing situations. Furthermore, the tracker was adapted to take into account divisions in the detection of the tracked objects by grouping them using similarities in distance and velocity.

The planner, `/tibi/move_base`, uses this tracker to know all the positions and covariances of all tracked people in the environment. Moreover, the planner can select a specific track of one person to approach this, as we do in this work.

**People prediction**

This node predicts the trajectories performed by people [11], and is contained in the planner. These predictions are made by defining and estimating the intention of people when describing trajectories in social environments. That is, given the previous track position corresponding to an existent track which the tracker is following, this node makes the prediction of this track (calculates the new position of the track with a certain covariance after a certain
3.2. Software

Figure 3.11: People detected by the tracker and identified by a number

time interval). These predicted positions are calculated taking into account two things: the interaction forces between the obstacles, other people and the robot, and the possible destination where people often go into a scene, some of them may be: doors, corridors, stairs, elevators, etc.

To make the prediction the node uses the previously corrected position, the average velocity of the track and the time interval between detections. The first time that the tracker has a new detection, it generates a new track which is sent to this node. In the next iteration, this node predicts the new track position in the current instant of time. The prediction is done in order to compare the current detections with the tracks that the tracker has previously. At this moment, this prediction is make like a constant linear velocity propagation. The tracker uses this node to obtain the predicted position of all the tracks at the current time, and the planner navigates the robot to a goal position, /tibi/move_base/current_goal.

Odometry node

This node gets the wheel velocity of the left and right wheel of the Segway status message, as well as the rotation in the roll, pitch and yaw angles. Using these values, the node calculates the linear and angular velocities and the translation in position of the Segway odometry. Then, this node publishes a message with the Segway’s odometry and this message is used in local planning.
In this chapter, the realization of the main path planning algorithm in MATLAB, together with the best trajectory selection, is presented and discussed.

Moreover, also a brief description of some main aspects of the general system considered are exposed here, in order to make the reading and comprehension of the algorithm clearer later.

However, only the basic computations have been performed in MATLAB and exposed here, because, as it will be discussed at the end of the chapter, some limitations occurred. Then, the complete implementation of the real-time planner will be presented in the following chapter, where some modifications in the C++ version of the code have been made, in order to solve the issues found in MATLAB.

4.1 General aspects

Before exposing the algorithm in its details, in the following sections, it is worth to describe the considered simulated environment, and how the elements are represented in that.

Both the robot and the approached person are considered as two circles, each one with a round area, of a certain radius, that represents their "confidence region". This is the part of the environment surrounding them that must be respected, meaning that in the approaching phase, the robot doesn’t have to invade that area, in order to have a natural approaching
phase. For our simulation, the robot radius has been set equal to 0.5m, and the person one 0.3m.

4.2 Final positions and candidate paths computation

The development of the first version of the algorithm has been made considering an empty environment, with just the robot and the person to be approached. The latter is considered static, and without turning movements of the head, then without changing the direction of the sight. The main task has been to have the computation of the candidate paths and the best path selection for the robot, from an initial state to a final one.

Later, in the C++ version, all these goals have been realized at high frequency, in order to have a real-time motion planning algorithm.

Since we’re aiming at a human-like natural behavior for the robot, it doesn’t have to arrive exactly in front of the person, but it is enough to approach him just arriving close, respecting the "confidence distance", and looking at him, also from the side, exactly like a human would do. For this reason, the robot possible final position is not a definite one, but more of a choice is possible and has been computed. More precisely, first the final position exactly in front of the person has been computed, and then other locations have been calculated as some points in the area of the direction of sight of the person, next to the previous one. For what concerns the possible final robot orientations, these are derived in such a way to have it looking at the person, even if not only from the front. Therefore, once set position and orientation for the person \((x_p, y_p, \theta_p)\), considered constant for the moment, a first computation of the final possible states for the robot has been done.

First, the final state \((x_r, y_r, \theta_r)\) exactly in front of the person and looking at this, as reported in figure 4.1, has been derived. For this purpose, \texttt{final_goal_pose()} function has been realized, in which the following equations are used:

\[
x_r = -d \cos(\pi - \theta_p) + x_p \tag{4.1}
\]

\[
y_r = d \sin(\pi - \theta_p) + y_p \tag{4.2}
\]

\[
\theta_r = \pi + \theta_p \tag{4.3}
\]

where \(d\) is the distance between robot and person centers, choosen equal to 2m.
4. BASIC ALGORITHM IN SIMULATION

4.2. Final positions and candidate paths computation

Then, other states next to this one, along the circle centered in the person and with radius equal to the mentioned distance $d$, are figured out. The function `other_goal_poses()` has been realized with this purpose. Once defined some parameters such as the number of other states we want to find, $Nposes$, and the angular difference $\Delta \gamma$ between them, it computes the final orientations for the candidates goal states in such a way to have the robot looking at the person, and starting from the orientation of this. The procedure is illustrated in algorithm 4.1: at the first iteration, from the orientation of the person, it computes first the positions for the robot on the right and left with respect to the central position given by `final_goal_pose()`, and with a displacement of $\Delta \gamma$, in Lines 4 and 7. Then, the orientations are adjusted in Line 5 and 8 to get the ones looking at the person; the same is repeated in the next iterations for a displacement of $i \cdot \Delta \gamma$, where $i$ varies, so the displacement increases.

Some examples are reported in figure 4.2: $\Delta \gamma$ is set equal to 20°, $Nposes$ to 4, and different orientation angles $\theta_p$ for the person are tried, while initially the robot is positioned in $(0, 0)$. 

![Diagram](image-url)
Algorithm 4.1 Compute other robot poses

1: function Compute other robot poses((x_p, y_p, θ_p), d, Δγ, Nposes)
2: for i = 1 to Nposes/2 do
3: \[ \theta_{right} = \theta_p - i \cdot \Delta \gamma \]
4: \[ (x_{i,r}, y_{i,r}) \leftarrow (d \cdot \cos(\theta_{right}) + x_p, d \cdot \sin(\theta_{right}) + y_p) \]
5: \[ \theta_{right} \leftarrow \theta_{right} + \pi \]
6: \[ \theta_{left} = \theta_p + i \cdot \Delta \gamma \]
7: \[ (x_{i,l}, y_{i,l}) \leftarrow (d \cdot \cos(\theta_{left}) + x_p, d \cdot \sin(\theta_{left}) + y_p) \]
8: \[ \theta_{left} \leftarrow \theta_{left} + \pi \]
9: \[ \text{return } (x_{i,r}, y_{i,r}, \theta_{right}) \text{ and } (x_{i,l}, y_{i,l}, \theta_{left}) \]
10: end for
11: end function

Figure 4.2: Examples of robot final positions computation

(a) \( x_p = 5, \ y_p = 7, \ \theta_p = 70^\circ \)  
(b) \( x_p = 5, \ y_p = 7, \ \theta_p = 135^\circ \)  
(c) \( x_p = 5, \ y_p = 7, \ \theta_p = -40^\circ \)
4. BASIC ALGORITHM IN SIMULATION

4.3 Cost computation and best trajectory evaluation

Once the final candidate positions have been set, the \texttt{G2\_spline()} algorithm is called, in order to compute a path for each final state. Some examples of possible paths are reported in figure 4.3, the details are showed in the captions.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{paths.png}
\caption{Examples of paths computed with G2-splines algorithm}
\end{figure}

4.3 Cost computation and best trajectory evaluation

Given the set of paths, the best path selection between the candidates has been realized depending on a cost, as already explained in Chapter 2. Here, however, not the final expression of the cost, eq. 2.23 has been used, since in the MATLAB version of the algorithm no forces are present. Only the cost to reach the goal, depending then on the length of the path, and the one related to the changing in the curvature, or, more precisely, to the changing of the orientation of the robot along the path, have been considered. The two costs, used in 2.14
and 2.15, are report another time here here for more clearance:

\[ J_{d,i} = ||x_{r}^{i+1} - x_{r}^{i}||^2 \]  \hspace{1cm} (4.4)

\[ J_{or,i} = ||\theta_{r}^{i+1} - \theta_{r}^{i}||^2 \]  \hspace{1cm} (4.5)

where the first computes the square distance value between the actual position of the robot and the next one along the path, and the second one expresses the difference between the orientations in the two locations. These costs have been computed for each step along the path, meaning that the path has been divided in samples, once defined a parameter called \textit{resolution} that indicates the distance, in meters, between these, and set equal to 0.4m. For the cost related to the curvature, however, not all the samples of a path have been taken in consideration in the computation, but a parameter, \textit{window size}, defining how many of them, has been set equal to:

\[ \text{window size} = \frac{5m}{t_s} \]

considering that we want to compute this cost only for the first 5m of trajectory. This choice is justified by the computational cost for the entire planning algorithm; taking in account all the samples of the path, indeed, the computation of the orientation differences would require too much time, so it could be unfeasible the realization in \( t_s \), set equal to 0.2s, that is the set time for the relaunch of the planner.

The two cost functions are then combined into a well-posed function, according the multi-objective optimization technique exposed in Section 2.4, and used to solve the problem of finding the best solution path. Reporting here the procedure, once computed mean and standard deviation of the costs of each path, all the step costs are normalized according to:

\[ \bar{J}_{k,i} = \omega_k \cdot erf \left( \frac{J_{k,i} - \mu_{path}}{\sigma_{path}} \right) \]  \hspace{1cm} (4.6)

where \( k \) means that this is valid for both the costs associated to distance and orientation, and \( i \) that is applied to each step cost. The final costs are derived as the accumulated costs \( J_{d} \) and \( J_{or} \) throughout the path and combined according to a weight parameter, obtaining:

\[ J = \sum_{i} \omega \cdot \bar{J}_{d,i} \]  \hspace{1cm} (4.7)
\[ J_{or} = \sum_{i} \omega_{or} \cdot J_{or,i} \]  

(4.8)

In the practice, the function `paths_steps_costs()` has been realize to compute all the costs related to all the steps in which the paths have been divided, and then the path related to the minimum cost is chosen. After some tests, in which really often the final costs are negative, the minimum one has been taken after computing the absolute values:

\[
\text{abs_length_costs} = \text{abs}(\text{length_steps_costs});
\]

\[
\text{best_path_cost} = \text{min}(\text{abs_length_costs});
\]

and the same for the orientation cost:

\[
\text{abs_or_costs} = \text{abs}(\text{or_steps_costs});
\]

\[
\text{best_path_cost} = \text{min}(\text{abs_or_costs})
\]

Once the simulation has been run in Simulink, some tests have been made to verify that the final orientation of the robot in the last sample of the path is exactly the desired one, thanks to the fact that at each step, going on through the samples, the robot orientation is adjusted to arrive at the final desired one.

Some examples of the best path selection are reported in Figure 4.4, with different person orientations. Here, the selected path is the orange one, and it is worth to remind that this computation is not made in real-time, meaning that the algorithm is launched only one time. The real-time implementation will be discussed in the following chapter, in the C++ version, where also some issues now exposed will be solved. As one can notice from Fig. 4.4c, indeed, the orange path is the one that the robot chooses and the trajectory that it performs. Here, in particular, the robot is not able to completely follow the selected path; this is due to the fact that the curvature of the path is really strong, and it seems that the robot is not able to perform a curve like that, and so to arrive to the final destination. This problem, however, will be solved in C++, where the ESFM will be implemented, and, in particular, there will be the inclusion of the social forces in the cost definition and for the robot motion.
Figure 4.4: Examples of best paths selection in different situations
Complete algorithm simulations and real experiments

In this chapter, some simulations, shown with RViz as visualization tool, are presented and discussed. In particular, for each example shown, some limitations, or modifications in the values of the parameters will be explained and overpassed, and the new results will be exposed. Sometimes, for a better exposition of the improvements, some lines of code will be reported, even if the complete algorithm has already been presented in Alg. 2.1.

As already stated before, in the C++ version of the code the algorithm has been relaunched at high frequency (precisely, every 0.2 s), and the ESFM has been included for the robot propagation. I

In the end, a brief introduction of the environment in which real experiments took place is presented, and the related conclusions.
5.1 Simulations

5.1.1 Basic case: without obstacles

First, the simplest case without obstacles is tested, in order to check if the robot follows the path selected, if the computation of the splines performs well and other aspects.

The first simulation is reported in figure 5.1, where on the left there are the candidate paths computed, and on the right the motion performed by the robot. The orientation of the person is set equal to $0^\circ$, and the algorithm computes the splines going from the initial position of the robot to the final possible goals, as shown in fig. 5.1a. From the other figure, however, it is clear that the robot is not able to follow the spline, and the trajectory performed consists in a simple straight line. This is due to the fact that the spline shape, at the beginning, is more or less equal to a straight line, then, since the algorithm is relaunched every 0.2s, the part of the path the robot performs during that interval is exactly the straight part, and in the end, after all the launches, this results in a straight line. In addition to this, the value of the path resolution, $res$, plays an important role: this, indeed, is used to set the actual goal for the robot; from the code:

$$\text{step\_goal} = \text{Sdestination}(0, \text{xpoints[index]}, \text{ypoints[index]});$$

$$\text{actual\_f} = \text{calculate\_edge\_approaching}($$

\text{current\_robot\_point, step\_goal, index, distance\_to\_obstacles\_and\_people});$$

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure5.1.png}
\caption{RViz visualization of the simulation with: $(x_p, y_p, \theta_p) = (0, -3, 0^\circ), (x_r, y_r) = (-2, 10.5)$, $res = 0.2m$}
\end{figure}
If the distance between the actual position of the robot and the best selected goal is more than 3m we set $index = 4$ meaning that the robot actual goal is the 4th sample of the computed path from the actual position, if the robot is closer to the destination, then we take the following sample, as in the simulation before. The performances with the new method are displayed in the following figures, where we tried different resolutions, in order to test which is the best value and to check the changing of trajectory performed depending on $res$.

In fig. 5.2 the simulation with $res = 0.2$ is shown. Note that we added a blue marker, visible in 5.2a, to visualize the position of the step goal for the robot. From the picture on the right we can see that the robot is not able to arrive at the final destination and the path is still too straight; this is due to the fact that, even if the path presents a bigger curvature with respect to the simulation before, the resolution is still too small for having the robot able to turn enough to arrive at the final goal position, even if we took 4 samples in the first part of the motion. For these reasons we decided to try with an higher resolution, as depicted in Fig. 5.3, where $res = 0.6$, and in Fig. 5.4, where $res = 0.8$. Both the cases are better,
5.4. Simulations

Figure 5.2: Simulation with: \((x_p, y_p, \theta_p) = (0, -3, 0^\circ), (x_r, y_r) = (-2, 10.5), index = \{4, 1\}, dist = 3m, res = 0.2m\)

Figure 5.3: Simulation with: \((x_p, y_p, \theta_p) = (0, -3, 0^\circ), (x_r, y_r) = (-2, 10.5), index = \{4, 1\}, dist = 3m, res = 0.6m\)

Figure 5.4: Simulation with: \((x_p, y_p, \theta_p) = (0, -3, 0^\circ), (x_r, y_r) = (-2, 10.5), index = \{4, 1\}, dist = 3m, res = 0.8m\)
because the robot arrives to one of the possible final goals, but the path is still not really good, meaning that the initial path computed and the one performed are strongly different. To try to get better results we run other simulations using the same technique, but this time leaving $res = 0.2$ and dividing a path in three phases, then using three values for $index$, depending on the distance between robot actual position and final one, as before.

In Figure 5.5 we took 10 samples if the robot is at least $6m$ from the final goal, and then 4 between $6m$ and $2m$ and only at the end 1 the next sample as goal. It seems that it arrives and also precisely in the final position, but the final path performed, specially in the end, is not really good, and the final orientation is completely wrong.

![Figure 5.5: Simulation with: $(x_p, y_p, \theta_p) = (0, -3, 230^\circ), (x_r, y_r) = (-3, 6.5), index = \{10, 4, 1\}, dist = \{6m, 2m\}, res = 0.2m$](image)

As visible from Fig. 5.5b we want the robot to curve more at that point, in order to arrive more close to the final goal and also perform a trajectory more similar to the original one reported in Fig. 5.5a. To this aim, we augment the $index$ value from 4 to 7, because the higher is this number, the more the actual step goal for the robot is far and the robot tries
to reach it curving more. This depends on the fact that along the path the robot adjusts his orientation of an angle computed according to:

$$\theta_r = \tan^{-1} \left( \frac{y_r^{\text{actual}} - y_r^{\text{goal}}}{x_r^{\text{actual}} - x_r^{\text{goal}}} \right)$$  \hspace{1cm} (5.1)$$

and then, the more is the difference between actual position and goal, the bigger is the angle, but the final orientation is completely wrong. Updating the angle considering each sample of the path, indeed, as it was in the MATLAB version of the code, produces a more precise result, while taking a farer goal reduces the precision, ending sometimes in arrivals with wrong orientations as in 5.5d.

The result is depicted in Figure 5.6 and, in particular, in 5.6c we can see that the robot curves more before arriving close to the goal, thanks to $\text{index} = 7$, and for the ending part we decided to take $\text{index} = 3$ to have the robot curving more to arrive in the final position.

![Simulation Diagram](image)

**Figure 5.6:** Simulation with: $(x_p, y_p, \theta_p) = (0, -3, 230^\circ), (x_r, y_r) = (-3, 6.5), \text{index} = \{10, 7, 3\}, \text{dist} = \{6m, 2m\}, \text{res} = 0.2m$
Moreover, as already stated, changing the step goal also the computed forces change; then, taking a farther goal produces a more natural behavior for the robot, having an higher speed and a more fluent motion, by the moment that acceleration and velocity depend on the actual forces.

Despite the result in 5.6d is not really bad, we tested also cases with a different orientation for the person, maintaining the same values as before. With $\theta_p = -40^\circ$ in Fig. 5.7, the final orientation is completely wrong, and the motion of the robot in the final phase, strictly next to the goal, is not natural, because it seems to try to adjust the position going on and turning back continuously.

![Figure 5.7: Simulation with](image)

(a) Splines computation  (b) Robot final state

To solve the problem, we decided to relax the final arrival condition, meaning that we changed the tolerance for considering the robot arrived or not to the goal. In the previous simulations and in Fig. 5.7 the parameter $\text{goal \_tolerance}$ was set equal to 0.3, so we tried setting it equal to 0.6, that seems to be a good value for having a relaxed condition but still arriving close enough to the desired position, and we adjusted the orientation of the robot in order to face the person with the technique exposed here.

First we defined a parameter $\text{goal \_tolerance \_turn}$, set here equal to 1.5, to check if the robot is arrived inside a region more or less closed to the goal; then, if this is verified, we define two angles:

```matlab
>> robot_angle1 = robot_angle + 0.2;
```
5.1. Simulations

```
>> robot_angle2 = robot_angle - 0.2;

We compute the square differences, diff1 and diff2, between desired final orientation and the two before, and then we decide in which direction to turn the robot:

```
>> if(diff1 < diff2){
    pose_command.w = pose_command.w + 0.2;
    final_diffAngle = diff1;
} else{
    pose_command.w = pose_command.w - 0.2;
    final_diffAngle = diff2;
}
```

In the end, we check if we are already close to the final desired orientation, and in this case we stop the robot:

```
>> if(final_diffAngle < threshold_of_difference_between_the_angles_){
    pose_command.w = 0.0;
}
```

Results are depicted in Fig. 5.8. As we can see, now the robot is perfectly looking at the approached person, since we not command it to stop but, once arrived a certain distance defined by the new tolerance parameter, we use its actual velocity to move on and we adjust the rotational one deciding the direction of turn. In this way the robot starts to adjust the orientation and arrives at the end with the desired one.
5. COMPLETE ALGORITHM SIMULATIONS AND REAL EXPERIMENTS

5.1. Simulations

Figure 5.8: Simulation with: \((x_p, y_p, \theta_p) = (0, -3, -40^\circ), (x_r, y_r) = (-3, 6.5), index = \{10, 7, 3\}, dist = \{6m, 2m\}, res = 0.2m, goal\_tolerance = 0.6\)

5.1.2 Static obstacles

The case with static obstacles has been handed trying first the case with only static obstacles, to adjust parameters and check if the robot was able to move through the environment, and then a static person has been added. As already stated, indeed, the presence of obstacles or people changes the social forces the robot feels. Moreover, while before only the \(f_{goal}\) was affecting the robot propagation, now the new forces \(f_{obs}\) and \(f_{people}\) are taken into account.

Before showing the simulations results, here some important equations are reported, in order to discuss the selection of some parameters later. We remind that the resultant force affecting the robot is given by:

\[
f_r = \alpha f_{r,goal} + \gamma \sum_{p \in P} f_{r,p}^{int} + \delta \sum_{o \in O} f_{r,o}^{int}\tag{5.2}
\]

where

\[
f_{r,goal} = k_r(v_r^0 - v_r)\tag{5.3}
\]

and the interaction forces are expressed as:

\[
f_{r,p}^{int} = A_{rp}e^{(d_{rp} - d_{r,p})/B_{rp}w}(\varphi_{rp}, \lambda_{rp})\tag{5.4}
\]

\[
f_{r,o}^{int} = A_{ro}e^{(d_{ro} - d_{r,o})/B_{ro}w}(\varphi_{ro}, \lambda_{ro})\tag{5.5}
\]

\(A_{rj}\) and \(B_{rj}\), with \(j = \{p, o\}\), reflects the strength of the force and the radius of the circular
area in which the force is considered, respectively. This means that the robot before a distance equal to $B_{r_j}$ is not influenced by the related force, and after that it feels it.

Since the parameters $\{\alpha, \gamma, \delta\}, \{A_{rp}, B_{rp}, \lambda_{rp}\}, \{A_{ro}, B_{ro}, \lambda_{ro}\}$ and $k_r$ affect the contribution given by the single forces, and then the performance of the robot, here their correct values are discussed. The previous values have been derived according to a learning procedure exposed in [4], and here we report them:

$$\begin{align*}
\text{alpha_param: } & 1.0 \\
\text{gamma_param: } & 1.0 \\
\text{delta_param: } & 1.0 \\
\text{esfm_to_person_A: } & 5.05 \\
\text{esfm_to_person_B: } & 0.91 \\
\text{esfm_to_person_lambda: } & 0.25 \\
\text{esfm_to_robot_A: } & 4.05 \\
\text{esfm_to_robot_B: } & 0.81 \\
\text{esfm_to_robot_lambda: } & 0.05 \\
\text{esfm_to_obstacle_A: } & 5.0 \\
\text{esfm_to_obstacle_B: } & 0.4 \\
\text{esfm_to_obstacle_lambda: } & 1.0 \\
\text{esfm_k: } & 2.3
\end{align*}$$

while other important parameters are:

$$\begin{align*}
\text{v_max: } & 0.9 \\
\text{w_max: } & 1.0 \\
\text{av_max: } & 0.4 \\
\text{aw_max: } & 0.9 \\
\text{av_break: } & 1.0
\end{align*}$$

In Fig. 5.9 is depicted the case with $\theta_p = 0^\circ$; note that the arrows represent the forces: the red one is the resultant force obtained from the force to the goal, indicated by a blue arrow, and the repulsive forces due to obstacles, that the robot feels especially along the path and that are represented by black arrows. Sometimes, it is visible also a purple arrow next to the
approached person, this is the force due to the robot.

We can see that the robot doesn’t crash against the obstacles, but the path performed is completely different from the one planned at the beginning, and the robot motion is not fluent, due to the repulsive forces he feels from the presence of obstacles and that stops him along the path. Then we try to relax a bit the value of the strength, $A_{ro}$, without setting it too low because the robot is arriving really close to the obstacles (Fig. 5.9c) and we don’t want it to crash. Moreover, we modify a bit also $\alpha$, passing from 1.0 to 1.2, to give more weight to the force related to the goal, so the robot can perform a trajectory going more straightly to the final destination.

To have a more fluent motion, we set a smaller value for $av_{break}$, and we add two variables that limit the total amount of impulsive force with respect to obstacles, in order to avoid to have a too high $f_{obs}$ in the case in which many obstacles are present in the environment. The two thresholds limit the $x$ and $y$ components of $f_{obs}$. The new values, then are:
5.1. Simulations

\begin{itemize}
\item \textbf{alpha\_param}: 1.2
\item \textbf{esfm\_to\_obstacle\_A}: 4.5
\item \textbf{esfm\_to\_obstacle\_B}: 0.4
\item \textbf{av\_break}: 0.6
\item \textbf{force\_obs\_max\_x}: 0.8
\item \textbf{force\_obs\_max\_y}: 0.5
\end{itemize}

Results are shown, with the same person orientation as before, in Fig. 5.10.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{simulations}
\caption{Simulation with static obstacles in the environment: \((x_p, y_p, \theta_p) = (0, -3.0^\circ), (x_r, y_r) = (-3, 6.5), \{\alpha, \gamma, \delta\} = (1.2, 1.0, 1.0), \{A_{ro}, B_{ro}\} = \{4.5, 0.4\}, v_{break} = 0.6\).}
\end{figure}

The path shape improved a lot; the robot, indeed, performs a more logical and simple trajectory, and more similar to the original plan too. However, as visible in Fig. 5.10a, what influences a lot the resultant robot motion is the planned path at the beginning, before the propagation. More precisely, the computed splines go through the obstacles, because they
are calculated from the information we have on initial and final points only, without taking into account the elements present in the environment. Another example is reported in Figure 5.11, where the path executed is really different from the initial one because some paths are not touching the obstacles but they are really close to them, and the others pass through them.

To solve this, we decided to include the social forces in the cost computation, and consequently in the best path selection. As reported in Alg. 2.1 (Line 9) in Chapter 2, the computation of the forces at that point of the scheme let us define the cost related to each path using that information, and then choose the best path depending on a cost which includes the forces too.

Reporting briefly the code, first for each path we compute:

```
>> actual_f = calculate_edge_approaching(current_robot_point, step_goal,
```
index, distance_to_obstacles_and_people);

considering as step_goal the next sample along the path, then iterating along it. Then we compute the cost for each path, including the forces:

>> single_path_cost = path_cost_computation(xpoints, ypoints, actual_f;
    paths_costs.pushback(single_path_cost);

and we select the best one, corresponding to the minimum cost:

>> best_path_index = best_path_selection(paths_costs);

In this way, the planner at the beginning selects for sure the path that is not touching obstacles, since the related cost is the minimum one because there are no forces due to entities in the environment included in the computation; then, while the robot is moving, path and costs values change, depending on obstacles, length and curvature. Two simulations are shown in Figures 5.12 and 5.13. As we can see, the first path selected is not going through obstacles, then, while the robot is moving, the forces values change and the selection of the path takes into account them. In Fig. 5.12b the robot is subjected to the repulsive force with respect to static obstacles, then in Fig. 5.12c it feels the repulsion from the person. In the end, it arrives to the destination after having avoided all the obstacles. In Figure 5.13 the behavior is the same but with a different orientation for the approached person. Note that also the final path shape is similar to the one planned at the beginning.

5.1.3 Static obstacles and person

In Fig. 5.14 a static person is added in the environment. The orientation for the approached person is set to 0°, then the robot passes between static obstacles and person, and it feels the repulsive forces due to both the elements. In Fig. 5.14b is depicted the instant in which the robot has passed close to the obstacles and it feels the repulsive force with respect to the goal person, which also feels the repulsive force due to the robot. To avoid the robot to crash against the person, after some tests, we found out that $\gamma = 1.0$ was not enough, so we decided to change it to weight more the repulsive robot force with respect to the person. Then we have:
5. COMPLETE ALGORITHM SIMULATIONS AND REAL EXPERIMENTS

5.1. Simulations

Figure 5.12: Simulation with forces included in the cost computation: $(x_p, y_p, \theta_p) = (0, -3, 0^\circ)$, $(x_r, y_r) = (-3, 6.5)$
Figure 5.13: Simulation with forces included in the cost computation: $(x_p, y_p, \theta_p) = (0, -3, 160^\circ)$, $(x_r, y_r) = (-3, 6.5)$
Figure 5.14: Simulation with static obstacles and person: $(x_{ap}, y_{ap}, \theta_{ap}) = (0, -3, 0^\circ)$,
$(x_r, y_r) = (-3, 6.5), (x_p, y_p) = (0, 5), \{\alpha, \gamma, \delta\} = \{1.2, 1.4, 1.0\}$

\begin{itemize}
  \item alpha_param: 1.2
  \item gamma_param: 1.4
  \item delta_param: 1.0
\end{itemize}

The behaviour of the robot, after all the adjustments and this final change is natural, the motion is fluent and it arrives in the final goal position with a good orientation.
5.2. Real tests

Some experiments have been carried out in the area of the Facultat de Matemàtiques i Estadística (F.M.E.), $8 \times 8\,\text{m}^2$ accessible, located next to our research institute I.R.I. The F.M.E. environment is outdoors, but partly covered by a roof, and since no obstacles were in the field, we first tested without obstacles, and then we placed several artificial ones. The area is shown in Figure 5.15.

![F.M.E. scenario](image)

5.2.1 Basic case: without obstacles

First the simple case without obstacles has been tested. We did experiments changing the orientation of the approached person. In Fig. 5.16 $\theta_{ap} = 90^\circ$. The robot is perfectly able to navigate, Fig. 5.16b, and arrive exactly next to the person facing her, as shown in Fig. 5.16c. Then the cases with $\theta_{ap} = 0^\circ$ and $\theta_{ap} = 180^\circ$ are depicted in Figures 5.17 and 5.18 respectively.
5. COMPLETE ALGORITHM SIMULATIONS AND REAL EXPERIMENTS

5.2. Real tests

Figure 5.16: Real experiment without obstacles and person orientation $\theta_{ap} = 90^\circ$

(a) Beginning  (b) During the motion

(c) Robot final state

Figure 5.17: Real experiment without obstacles and person orientation $\theta_{ap} = 0^\circ$

(a) During the motion  (b) During the motion

(c) During the motion  (d) Robot final state
5.2. Real tests

![Image](image-url)

(a) During the motion  
(b) During the motion  
(c) Robot final state

**Figure 5.18**: Real experiment without obstacles and person orientation $\theta_{ap} = 180^\circ$

5.2.2 Static obstacles

Static obstacles have been added in the environment. The same orientations as before have been set for the person and results are reported in the following pictures. The robot is always able to avoid collisions and to arrive in front of the person, even without uncertainties along the path and a natural behavior. The most difficult case is the one with $\theta_{ap} = 180^\circ$; in Fig. 5.21, indeed, all the movements of the robot along the path are reported. It is visible how the robot arrives more or less next to the person, but to adjust its orientation to face this it has to rotate a little and turn to look at the goal. However, in general, also this test seems to be acceptable.
5. COMPLETE ALGORITHM SIMULATIONS AND REAL EXPERIEMENTS

5.2. Real tests

Figure 5.19: Real experiment with static obstacles and person orientation $\theta_{ap} = 90^\circ$

Figure 5.20: Real experiment with static obstacles and person orientation $\theta_{ap} = 0^\circ$
Figure 5.21: Real experiment with static obstacles and person orientation $\theta_{ap} = 180^\circ$
6

Conclusions and future work

In this final chapter, we present the obtained conclusions of the work. Also, we evaluate the fulfilled objectives of the project. Furthermore, we present some possible future lines of work based on the open issues identified during the project.

6.1 Conclusions

In this work, we presented the whole local planner approach for Tibi. In the local planner implementation we meet all the initial fixed objectives, meaning that we obtained a planning algorithm able to get a feasible path for the robot in each instant of time. Furthermore, we tested the planner with rosbags and in the robot proving its good behavior.

The real contribution apported has been the implementation of a path planning based on G2-splines for the path computation on a mobile robot, while before this was realized only for an autonomous driving system. With the Anticipative Kinodynamic local Planner based on splines we can approach a person in a robust way. Also, we can avoid obstacles present in the environment by exploiting the social forces due to each entity. Furthermore, these forces have been used to select the best path for the robot, given a set of candidate ones.

Moreover, the approach is natural, meaning that the robot presents a human-like behavior, thanks to the fact that it arrives to the area surrounding the person and with an orientation that lets it to face the person, since we adjusted the angle of the looking direction for the
robot while arriving to the goal. Moreover, no perturbations are created by the robot.

However, even if the robot is able to navigate in an area where other entities are present, in the future we could improve the path computation. Since this computation is based only on the knowledge on initial and final robot position, indeed, the algorithm computes the candidate paths, and then we included the forces on the cost associated to each trajectory. In this way, however, some of the computed paths go through the obstacles, because the possible way for the robot doesn’t take into account the environment. For the future, we expect that including the forces related to the environment in the path computation, the shape of the trajectories will be adjusted to the scene, meaning that they will avoid directly the robot to crash against something or someone.

Running the simulations, the best possible values for the forces have been derived, but a more general method, like a learning parameters procedure, could be a future improvement for a more general algorithm. Anyway, the adaptability of the realized planner to different situations, like different orientations for the approached person and different locations for obstacles in the environment have been tested, and the robot seems to respond well.

In the experiments part, we proved the good behavior of the planner with our robot, Tibi. We saw how the robot arrives to the person in a natural manner, and performing a good trajectory, similar to the original one planned at the beginning with the spline computation.

For the future, we will test the algorithm with a moving goal, changing not only position but especially the orientation, in order to check completely the validity of the planner. The idea will be to get the looking direction of the person from the data on his velocity. Indeed, from the equation:

$$\theta_{ap} = \tan^{-1}\left(\frac{v_y}{v_x}\right) \quad (6.1)$$

we can find out the orientation to give to the robot for computing the candidate final positions and paths.

### 6.2 Future work

The future lines of research which we found are related with some directions that we could follow in the future of this investigation project. Some of the possible improvements that we can make have been already exposed and we resume them here:

- We could consider a moving and turning person to approach
6. CONCLUSIONS AND FUTURE WORK

6.2. Future work

- We could include dynamic entities in the environment perturbing the robot motion, in order to test the capability of the planner to adapt to the environment.

- We could include the Social Force Model in the computation of the possible paths for the robot, to improve the shape and adaptation to the scene.

- We could analyze the performance of the robot and find out the best values for the parameters of the SFM with a learning procedure.
Appendix

Here the details on the Simulink implementation are shown. The overall system is depicted in Fig. 7.3, and before the two blocks for the derivation of the wheel velocities and the final robot pose derivation are reported.

Figure 7.1: Wheel linear velocities block
Figure 7.2: Pose derivation block
Figure 7.3: All robot system
References


Acknowledgments

I would like to thank, Professor Alberto Sanfeliu, for giving me the opportunity to develop a project on an interesting topic as mobile robotics, and for working with a real robot, Tibi, in the I.R.I.

Also, I would like to express my specials thanks to Ely Repiso, that helped and taught me a lot of things during my permanence at I.R.I.