Health, demographic change and wellbeing
Personalising health and care: Advancing active and healthy ageing
H2020-PHC-19-2014
Research and Innovation Action

Deliverable D5.4

Activity Planning

Deliverable due date: 2017-05	**Actual submission date: Thursday 1st June, 2017**
Start date of project: February 1, 2015	**Duration: 42 months**
Lead beneficiary for this deliverable: UNITN	**Revision: 1.0**
Authors: Paolo Bevilacqua, Luigi Palopoli, Marco Frego	
Internal reviewer: Daniele Fontanelli	

The research leading to these results has received funding from the European Union's H2020 Research and Innovation Programme - Societal Challenge 1 (DG CONNECT/H) under grant agreement n° 643644.

Dissemination Level		
PU	Public	
CO	Confidential, only for members of the consortium (including the Commission Services)	**X**

Contents

Executive Summary

This deliverable presents the technology developed by the ACANTO consortium to solve the problem of translating a recommended activity (produced through the technology produced in WP4) into an executable plan. An executable plan is for us a sequence of curves that joins a set of desired locations and that can be actually followed by a FriWalk using the guidance solutions developed in WP6.
The problem tackled here has two distinct facets. One is to plan a sequence of high level actions that achieves the goals of the recommended activity and respects the preferences encoded in the user profile. On the same level of importance is that the high level have a realistic match with a lower level sequence of actions (i.e. following curves), which can be executed by the user with her FriWalk and with her physical limitations. As discussed in the deliverable, our approach is based on an abstraction/refinement technique. Abstraction is used to characterise high level actions with "physical parameters" considering a wide range of possible scenarios. Refinement is used to produce a motion plan that refines the high level actions.
This final deliverable regarding the Activity Planner presents various improvements and updates from the preliminary version (D5.3). From the perspective of the motion planning, we present a possible improvement based on the deterministic search for a sub–optimal solution path, through the construction of a visibility graph. The resulting path is used as an initial guess for the stochastic search algorithm. Regarding the high level reasoning, we illustrate how we intend to model the possible tasks, preferences and constraints, and the possibility to adopt a different strategy to find a valid plan, based on the translation of the high level model to a Mixed Integer Linear Program. Finally, we provide a detailed example on how the original model can be extended to support the generation of a plan considering more users.

Chapter 1

Introduction

Activity Planning plays a very important role in ACANTO as it represents the trait-d'union between the conception of an Activity Recommendation and its execution. This is easy to see in the logical diagram in Figure 1.1. Once an activity has been generated and the user (or a group of users) commit to its execution, it has to be translated into an executable plan. This step entails:

1. Finding the best way to achieve the goals contained in an activity, given a description of the environment (with its opportunities and constraints) and the preferences contained in the users profile,

2. Generating a motion plan that can be smoothly executed by the users using their robotic walkers (FriWalks).

The former activity requires a good amount of high level reasoning. It uses a high level representation of entities and actions and it takes decisions that can be executed in typical (or worst case) operating conditions. The latter activity, on the contrary, takes into direct consideration dynamic equations and physical limitations of the vehicle and it generates a plan that can be executed given the current estimated conditions. The plan is amenable to changes and corrections as required by the situation encountered along the path, and this falls under the responsibility of the Reactive Planner.
Just to make a quick example (others will follow in the next chapter), suppose the recommendation system issues a recommendation such as "visit to the science museum". Browsing the catalogue and considering a user with a keen interest on "mammals", the planner will select a sequence of targets that will likely maximise the user's satisfaction. The total duration of the visit will have to be within the boundaries dictated by the physical conditions of the user. In the same way, other constraints (such as having a bathroom always in easy reach) will have to be enforced. It could be that some of the constraints conflict. In this case the system will use priorities to make a choice. Interestingly, all these choices will be taken using a high level of abstraction in order to effectively reason about options and opportunities. However, all abstract choices will have to be representative of feasible concrete actions. For instance suppose that moving from place A to B is an action associated with a duration between $T_{\min}$ and $T_{\max}$, then it means that many "low-level" scenarios have been considered and that in all of them the system could plan a concrete motion from A to B with a duration in the interval $[T_{\min}, T_{\max}]$. The confidence of this interval can be *hard*, meaning that for all scenarios that can be fancied the property holds true, or it can be soft, meaning that the property is satisfied with an acceptable probability (confidence). Once a decision is taken on a sequence of high level actions, the plan will have to be refined into an actual motion plan, i.e., a sequence of curves that the walker can follow and that are compatible with the current situation of the environment. As long as this specific situation falls within the scenarios considered in the definition of the high level actions, the planner will surely find a feasible solution.
In the next chapter we will offer a more complete description of the overall planning framework and of its different components. The document is organised as follows. Chapter 2 describes the general framework

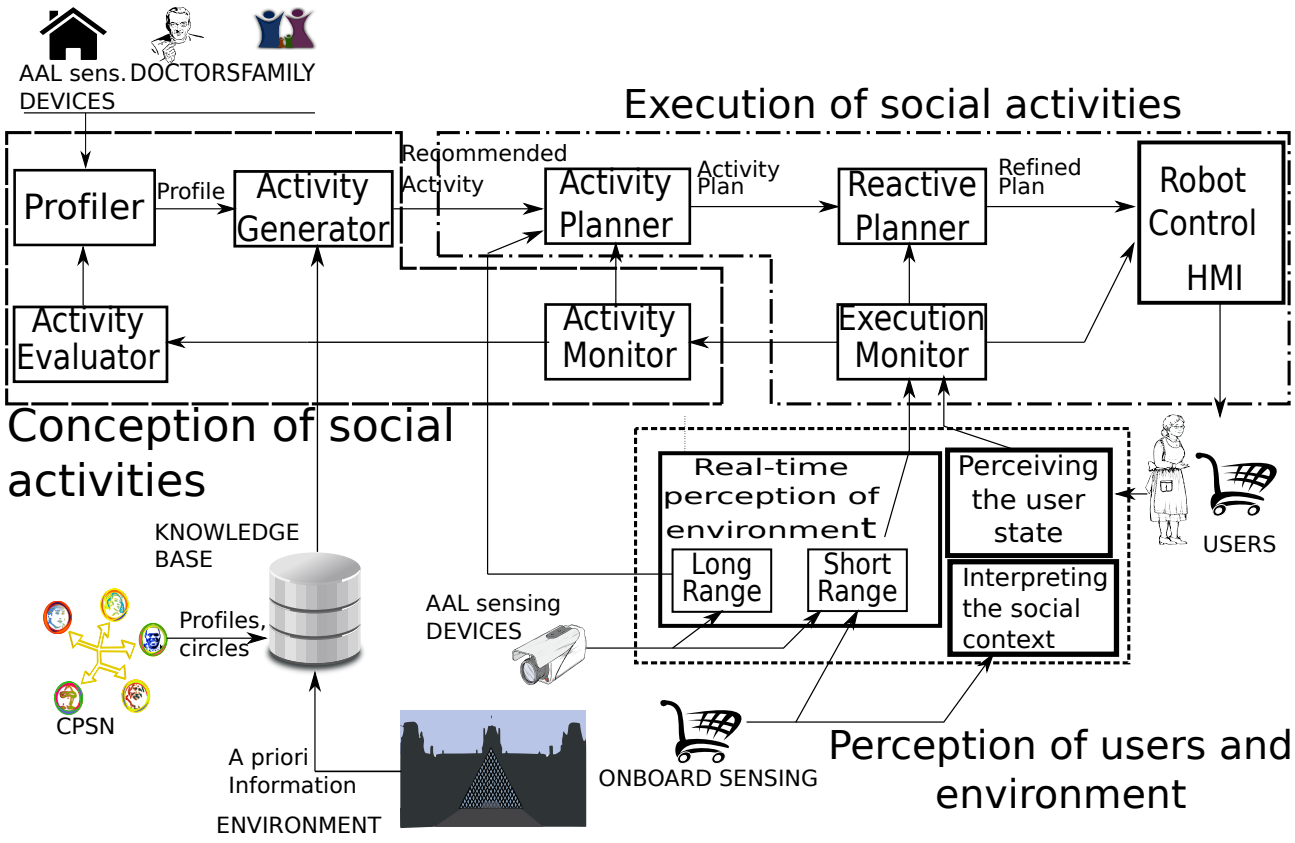

Figure 1.1: Logical diagram representing ACANTO's main components and their interconnections

composing the Activity Planner. Chapter 3 presents the motion planner, and describes the models and cost functions adopted to synthesize comfortable trajectories from the perspective of the user. Chapter 4 illustrates the high-level planning component, and how it is used in combination with the motion planner to produce optimal plans that can be executed by the FriWalk. Finally, the conclusions summarize all the key points of the Activity Planner framework, and present some open issues and possible future extensions.

Chapter 2

Framework

The objective of the Activity Planner is the synthesis of an executable plan, that can be carried out by the FriWalk. The plan is generated based on the suggestions and recommendations coming from the Activity Generator, and must take into account information and knowledge regarding the specific profile of the user. The user profile defines preferences and constraints specific to each person. The plan should try to maximize the satisfaction of the user by choosing activities and actions based on his preferences, but at the same time must respect all the imposed hard constraints (for example specific medical indications).
The idea behind the framework composing the Activity Planner is to generate and use data obtained through various simulations to dynamically produce a realistic model for the high-level instance of the planning problem (expressed with a language of the PDDL family). A diagram illustrating the structure of the Activity Planner is shown in figure 2.1.
A geometric representation of the environment defining the locations of the various points of interest and including information about the layout of the building and of the static obstacles (like walls) is used as a base model for all the simulations. This static map is randomly filled with dynamic elements like crowds, or zones to avoid (repulsive regions) or to walk on (attractive regions), producing different scenarios. This dynamic elements, different for each simulation, are generated according to some probability distribution based on statistical observations on the real environment. During each simulation, the motion planner is run to find optimal paths to move between each pair of connected points of interest. Each of the produced optimal paths must avoid collisions with static obstacles, and should optimize a cost function based on the user comfort, which depends both on the geometry of the followed trajectory, and on rewards and penalties accumulated when traversing attractive or repulsive zones.
By analysing the data extracted from these simulations, our system computes statistics regarding the amount of time and distance required to move from one point of interest to another. The parameters associated to the elementary actions are stored for each scenario (e.g. level of crowdedness, environmental conditions, etc.) in the KnowledgeBase. The frequency of each scenario is weighted when computing the aggregate value of the parameters during the planning phase. This data is also integrated with the real time observations of the sensing system, if available. This information is then injected into the model used to express the high-level instance of the planning problem. The obtained instance can thus be solved by an automated planner, to produce a sequence of high-level tasks and actions compatible with user needs and requirements. In this way, the produced plan will be valid for most of the possible situations that are likely to occur, and only for a small amount of unlikely scenarios could the plan become unfeasible. Should this happen, the reactive planner comes into the picture producing the necessary changes.
When the high-level plan is finally transformed into an executable low-level plan, actions corresponding to a motion between two different locations have to be refined by invoking the motion planner. The net result is an optimal path based on the current status of the environment, according to the current attractiveness/repulsiveness (crowdedness) of the different zones of the map. Due to the realistic simulations performed before

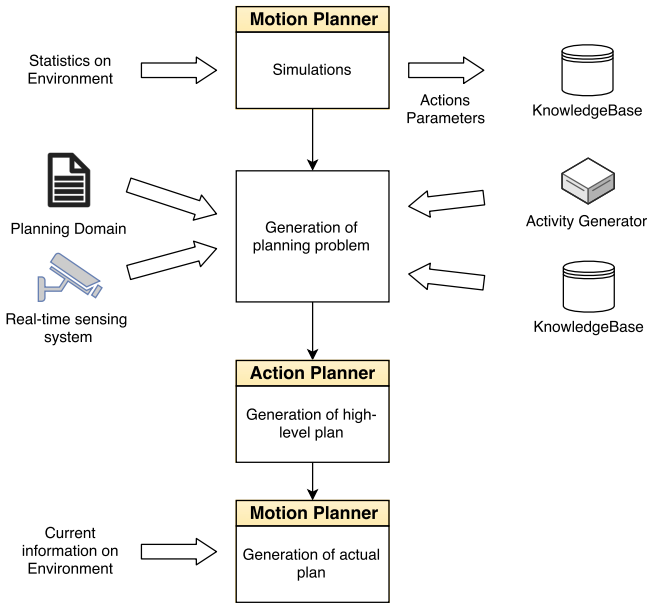

Figure 2.1: Diagram illustrating the structure of the Activity Planner

the synthesis of the high-level plan, for almost all the possible situations the produced trajectory will comply with the corresponding high-level task. Thus, the components supporting execution and monitoring will be able to deal with many different scenarios, without the need for an intervention of the automated planner. Only in the case of some unlikely and exceptional situations, will the plan become unfeasible, and a new high-level plan will be computed to address the contingencies encountered on the path. The percentage of uncovered cases depends on the percentile of considered situations when the duration of a high level action is estimated by simulation. Considering worst case scenarios (or conservative approximations) permits the production of plans valid for all the possible conditions. The price to pay is that many actions will be ruled out, even if they are very rewarding, because of unlikely catastrophic events. Worst case planning could therefore be extremely conservative and, ultimately, inconvenient. On the other hand, if our choice of the percentile is too "liberal", we could have frequent cases in which a plan has to be radically modified during its execution, which diminish the confidence of the user in the system. So a trade-off between the amount of contemplated scenarios and the amount of different admissible plans must be sought.

2.1 Example scenario

The scenario adopted within this deliverable to show realistic examples for the different components of the Activity Planner framework, is inspired from Michael's use case, discussed in deliverable 2.5. This use case gives us the possibility to define a simple but realistic scenario, and to show how different features like preferences and hard constraints can be modelled.

> Michael is a 72 year old man who lives alone in Felling, Gateshead. For the past few years, he has found mobility very difficult and he is waiting for a hip operation. Consequently, he does not get out much. He used to enjoy visiting museums and now fulfils his passion for natural history by watching documentaries on TV. He would like to be able to get out to visit the museums in Newcastle.
>
> A researcher visits Michael one day and shows him the FriTab. Michael explains to the researcher that he has mobility problems so would not be able to get out much. But the researcher explains

the FriWalk to him which is owned by several shops, galleries and museums in the area. He also explains that people on the FriTab network may be able to help him get transported to locations and events. So Michael enters his details into the system and tells it that he has mobility problems. The information on Michael's profile are also updated by his doctor, who also enters some constraints concerning the activities Michael can safely carry out.

The next day, the FriTab shows Michael that a tour is being organised at the Hancock Museum. It invites him to attend and tells him that another person attending would be willing to pick him up. The system knows that Michael enjoys natural history and that he also has mobility problems. It knows that the museum has several FriWalk devices that can help Michael. It also knows that one other attendee has a car and is willing to transport friends. Michael is hesitant but agrees to give it a try. So he tells the system that he will attend. The FriTab tells him that Jane will pick him up before the event in her car. Michael tells her his address.

At the arranged time, Jane picks Michael up and they drive to the museum. When he arrives at the museum, he is given a FriWalk which helps him to walk with the rest of the tour group. After the tour is over, the FriWalk even suggests a guided tour of its own that Michael can do alone without violating the medical prescriptions. However, Michael is tired but decides to come back and try the guided tour another day. The FriTab forwards the activity log file to the network for user profile updates.

Inspired from this use case, we define three different scenarios, based on a map representing portions of the museum with an increasing number of points of interest, and on some realistic preferences and constraints fitting Michael's profile. The map used for our experiments, depicted in figure 2.2 (needs to be updated), represents the complete museum, and contains various points of interest, regarding different topics. The solid black lines visible on the map correspond to static obstacles (walls). In figure 2.3 is shown an example of the same map, with some dynamically generated attractive (green) and repulsive (red) zones which could be used to perform some simulations. For the modelling of this example scenarios, we assume some hard constraints and some preferences for Michael, reported in table 2.1.

Table 2.1: Michael's constraints and preferences

Hard Constraints	**Preferences**
• Avoid activities requiring too much walk (< 500 m) • Avoid stairs/lift	• Level of interest for plants: 30 • Level of interest for animals: 60 • Level of interest for fungi: 15

For the sake of brevity ,for the different examples we assume to have already computed the aggregated parameters of each elementary action according to the current information of the environment yield from the sensing system.

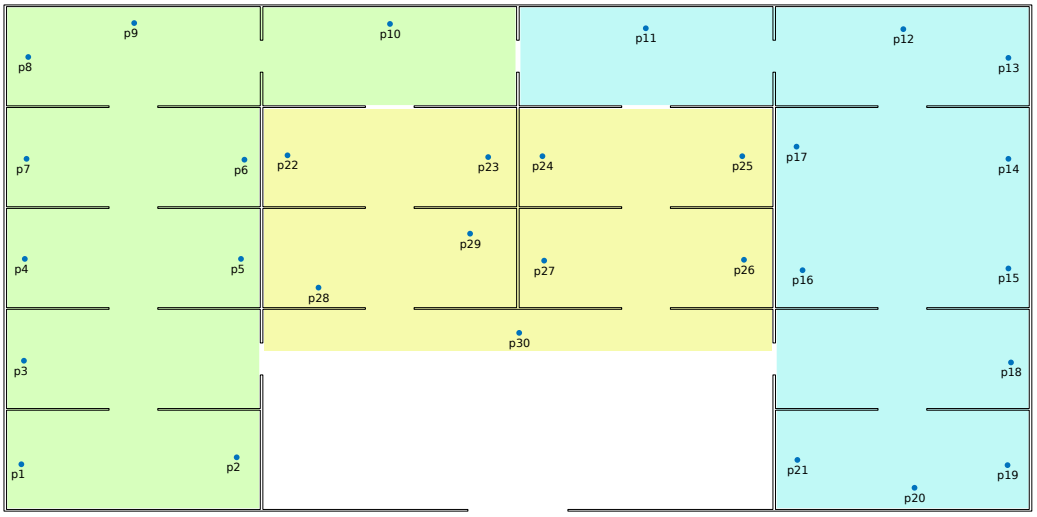

Figure 2.2: Map of the museum used for all the examples of this deliverable. The yellow zone is the section about fungi. The green zone is the section about plants. The blue zone is the section about animals.

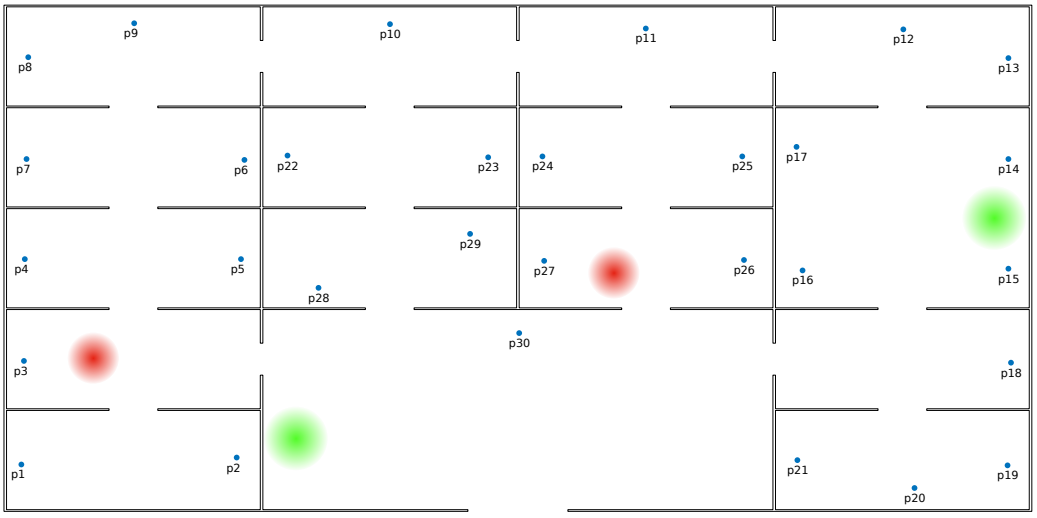

Figure 2.3: Map of the museum with an attractive (green) and a repulsive (red) zone. The attractiveness of a zone may be used to model dynamic information about the environment like for example the level of crowdedness.

Chapter 3

Motion Planning

The Motion Planner is responsible for the generation of an actual path that can be followed by the walker. Given the starting and the goal location, it produces a path connecting them, trying to minimize a certain cost function, and at the same time avoiding all the obstacles on the map.

As depicted in figure 2.1, the motion planner has two different applications within our framework. Firstly, it is used during the simulation stage to determine realistic physical parameters regarding the environment (e.g. lower and upper bounds on the time required to walk between two different locations) that are associated to the various high-level actions during the synthesis of the planning model. This preliminary step allows us to define realistic high-level actions, taking into account information about the environment and the user comfort, and to synthesize plans which are feasible in most of the possible situations.

Secondly, the motion planner is used to refine the sequence of high-level actions composing the plan into a sequence of motion primitives (paths) that can be travelled by the FriWalk. These are synthesized when the plan must be actually executed, taking into account the situation detected on the field and choosing the "best" solution in terms of user comfort.

3.1 State of the Art

Different approaches exist to solve the problem of motion planning [21]. Among them, sampling based approaches are very popular, since they provide a viable solution even for really large and high-dimensional search spaces, where deterministic techniques are too expensive in terms of computational requirements. Sampling based motion planners randomly sample points within the configuration space, searching for optimal paths connecting the given starting and goal configurations. One widespread and successful sampling based motion planning algorithm is RRT* [20]. It is an anytime algorithm, meaning that, if it is left running for more and more time, it keeps improving the produced solution. Moreover, RRT* has been shown to find an optimal solution (if one exists) with probability 1 in the limit when the running time tends to infinity.

Particularly relevant in our context is planning motion in crowded areas, which requires the availability of a good model for the interaction between agent and crowd. Agent-based crowd modelling has been studied extensively for the last two decades, mainly with attention to emergency situations such as the evacuation of a certain area, e.g. a theatre or a building [26]. Many models have been proposed, among them we recall: path planning, navigation systems, combinations of precomputed manoeuvres [2], multi-agent planning [23], crowd synthesis [18], force and potential fields, social force models, continuum fluid dynamic models.

3.2 Problem Description

Motion planning solution for a robot assisting a user requires a proper consideration of the following three points:

1. the length of the path;
2. the time required to walk to destination;
3. the comfort along the path.

The length of the path is a purely geometric fact, whereas the time required to travel on the path depends on dynamic considerations about the pedestrian; finally, the comfort index can be defined with various characteristics and can be dependent on the dynamic model or on the geometric shape of the path. Using a "global" approach, these aspects are considered together. Hence the minimisation problem J is defined, for weights w_1, w_2 and w_3, as:

$$\min J = w_1 \int_\gamma \mathrm{d}s + w_2 \int_\gamma \mathrm{d}t + w_3 \int_\gamma c(s)\,\mathrm{d}s,$$

where γ is the path, the first integral minimises the length, the second the time and the third the comfort function $c(s)$. If v is the speed profile along the path γ, the integral with respect to time $\mathrm{d}t$ and space $\mathrm{d}s$ are related with the formula $\mathrm{d}s = v\,\mathrm{d}t$, thus it is possible to reformulate the previous target functional completely in space or time.

Thus, as our motion planner should synthesize effective paths, particular care and attention must be put not only to geometric constraints, but also to external and environmental factors like for example the presence of crowds, affecting the user comfort. Indeed, depending on the density of the crowd, on the peculiar requirements of the user, and on the cost of an alternative path, the system must choose the most appropriate path, trying to avoid or minimize the crossing of penalizing zones.
In order to tackle the complexity of the resulting problem, we address two subproblems separately: a geometric subproblem that takes into account the shape of the path, length and comfort index (given by the minimum jerk) and another, different, subproblem dealing with the dynamic constraint of holding a certain speed profile v along the path (and thus considering the total time required to travel along γ).

3.3 Geometric subproblem

The geometric component of the cost function adopted to optimize the comfort of the user should minimize the total amount of jerk (corresponding to variations of the curvature) and the length of the path. Within our system, the generated paths are thus composed by splines of clothoid curves minimizing the jerk while keeping the total length as short as possible [5, 10].

3.3.1 Clothoids

This subsection contains some brief remarks about clothoids and the adopted notation.

Definition 3.3.1 (Clothoid)**.** A clothoid is a parametric plane curve $(x(s), y(s))$, where s is the arclength, with the property that the curvature $\kappa(s)$ is a linear function of s, i.e $\kappa(s) = \kappa' s + \kappa_0$.

It has been proved since Euler that a clothoid arc is represented via the Fresnel Integrals, and is characterised by six real parameters: (x_0, y_0), the initial point, θ_0, the initial angle, κ_0, κ_0' the curvatures and the length L. The space coordinates, angle and curvature at arclength s on a clothoid are given by:

$$x(s) = x_0 + \int_0^s \cos\left(\frac{1}{2}\kappa_0' t^2 + \kappa_0 t + \theta_0\right) \mathrm{d}t, \qquad \theta(s) = \frac{1}{2}\kappa_0' s^2 + \kappa_0 s + \theta_0,$$
$$y(s) = y_0 + \int_0^s \sin\left(\frac{1}{2}\kappa_0' t^2 + \kappa_0 t + \theta_0\right) \mathrm{d}t, \qquad \kappa(s) = \kappa' s + \kappa_0.$$

3.3.2 Kinematic model

Clothoids have been chosen as motion primitives because, as shown in [9], they represent time-optimal trajectories for the kinematic model:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\bar{\delta}} \end{bmatrix} = \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \\ \bar{\delta} \\ 0 \end{bmatrix} v + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \bar{\omega} \tag{3.1}$$

where θ is the orientation of the vehicle, v is the forward velocity, $\bar{\delta}$ the steering angle and $\bar{\omega}$ its velocity. In [1] is shown as the optimal solution of model 3.1 provides a good approximation for human trajectories, based on an high number of observations of human motions in real environments. In addition, the generated trajectories must be feasible for our FriWalk, that can be represented by the nonholonomic model 3.1.
Indeed, our FriWalk platform can be represented using the kinematic model of an unicycle-like vehicle, as

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \omega \end{bmatrix} \tag{3.2}$$

where v and ω are respectively the forward and the angular velocities. However, the trajectory followed by the FriWalk should have a continuous curvature and minimize the jerk to maximize the user comfort [15, 7]. Thus, the model described by the equations 3.2 is not adequate, and needs to be extended to the model described by the equations 3.1 to comply with these constraints.
Therefore, the adoption of clothoids as motion primitives allows our system to produce feasible, comfortable paths with continuous curvature. Moreover, with the adoption of semi-analytic solutions like the ones proposed in [5], the required computations are extremely fast. The use of motion primitives having analytical or semi-analytical solutions, which are extremely fast and efficient to compute, is of fundamental importance for the motion planner, that may need to generate an high number of trial trajectories to find an optimal path.

3.3.3 Approach: Stochastic search

The motion planning algorithm adopted within our framework is a variation of the traditional RRT*, called Informed-RRT* (I-RRT*) [12]. Algorithm 1 displays the pseudocode showing the main steps of this algorithm. During the initial phase, while a solution has not been found, it works identically to RRT*, by sampling new points within the free configuration space, and connecting them to the closest nodes within the search tree (if the subpath connecting them is collision free). When a node is added to the tree, the cost to reach neighbour nodes already in the tree passing through this new node is computed. For all the neighbours for which the cost of the new path is lower than the current cost, a 'rewiring' operation is performed, and the new node is set as their parent. Differently from traditional RRT*, when a solution has been found I-RRT* does not sample all the possible regions of the free space anymore, but it samples possible candidates only within an elliptical region of

the configuration space for which an heuristically calculated total-cost is lower than the current optimal cost. In figure 3.1 is shown an example of a search tree produced by the I-RRT* algorithm. After a sequence of points connecting the start and the goal locations has been found, a collision-free spline of clothoids passing through these points and minimizing the jerk and length is returned as the final trajectory [6]. The detection of collisions between a clothoid and generic static obstacles is performed by first decomposing each clothoid into a sequence of enclosing triangles. At the beginning of the execution, also the boundaries of the obstacles are decomposed into a sequence of enclosing triangles, which are organized into an efficient, hierarchical axis-aligned bounding boxes tree data-structure. Thus, to check whether a clothoid is in collision with some of the obstacles, it is sufficient to check whether any of the triangles composing the clothoid is in collision with any of the triangles describing the boundaries of the obstacles. More details on the implementation of the motion planner can be found in [7].
An example of an optimal clothoid spline interpolating the points produced by the I-RRT* algorithm, avoiding all the static obstacles and minimizing the total jerk and length is shown in figure 3.2.

Visibility Graph

To improve the performance of the motion planner, an initial sub–optimal solution is provided to the RRT* algorithm. This solution is found deterministically by constructing a visibility graph, where the nodes represent points of the Euclidean plane corresponding to all the vertices of the polygons representing the obstacles, together with the initial and the final goal positions of the robot. The visibility graph considering only the obstacles can be constructed once from the map of the environment, and dynamically updated with the new start and goal location for each invocation of the motion planner. Moreover, since the locations of the points of interest on the map are in general static or changing with a low frequency, also these nodes can be inserted in the visibility graph associated to the current map. For each pair of nodes, the segment connecting them represents an edge of the graph if it does not pass through any obstacle.
The visibility graph can be constructed and updated efficiently, using the algorithm proposed by Lee [22, 4]. Since the shortest path from the start to the goal node on the visibility graph is optimal when the adopted cost metric is the total length, the waypoints composing this path can be inserted as an initial solution when running the randomized algorithm. This approach allows us to overcome some limitations occurring when adopting a solution based only on RRT* (mainly the bug trap, i.e. the difficulty and slowness to find solutions passing through small openings, like doorways). The stochastic search, then, is used to update the initial path, adopting more complex cost functions, accounting also for the user comfort and for the different attractiveness of different areas of the map. An example of visibility graph is shown in figure 3.3.

3.4 Walking in the crowd

To obtain sensible physical parameters regarding the cost/duration of high-level actions from the simulations, it is of fundamental importance to estimate sensible values for the time required to walk a given path, accounting also for the crowdedness of the different zones. Indeed, an user should take more time to traverse unfavourable and packed zones, depending on the specific density of crowdedness. In addition, the walking time can be used as a cost function to evaluate the “goodness” of a specific solution. In this way, the optimal solution in terms of user comfort will be sought as a trade-off between the length of the path and the avoidance of unpleasant areas.
Within our solution, the user is assumed to walk at an almost constant speed for each time interval, which will have higher values when the path is obstacle-free, and lower values for example when crossing unfavourable areas. We consider herein only fixed attractive and repulsive regions, that are modelled with radial symmetry like a central force (e.g. gravitational field). As an example, we can model the effect of an obstacle (or of a

Algorithm 1: Generation of a spline of clothoids from a given starting position to a given goal

Data: $mapName$, $start$, $goal$
Result: Smooth path from start to goal
function *GeneratePath*
  $map \leftarrow processMap(mapName)$
  $tree \leftarrow initTree(start, goal)$
  $ellipse \leftarrow initEllipse(start, goal)$
  while (**not** $termConditions$) **do**
    $samplePos \leftarrow sampleNextPosEllipse(map, ellipse)$
    $parentFound \leftarrow findParent(samplePos, map, tree, parent, minCost)$
    if $parentFound$ **then**
      $newNode \leftarrow insertNewNode(samplePos, tree, parent, minCost)$
      $rewireNeighbours(newNode, map, tree)$
      if $goalInRange(samplePos, goal, map)$ **then**
        $connectGoal(node, goalNode, tree)$
      end
      $updateEllipse(ellipse)$
    end
    $updateTermConditions(termConditions)$
  end
  $path \leftarrow interpolateClothoidPath(tree)$
  return $path$
end

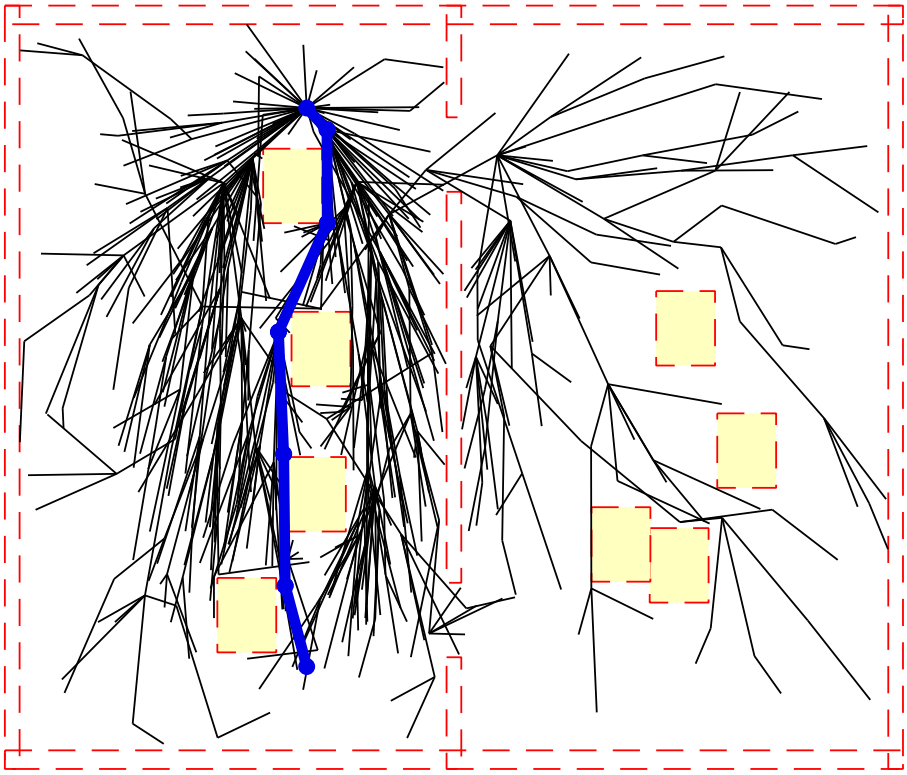

Figure 3.1: Example of a possible search tree produced by the I-RRT* algorithm

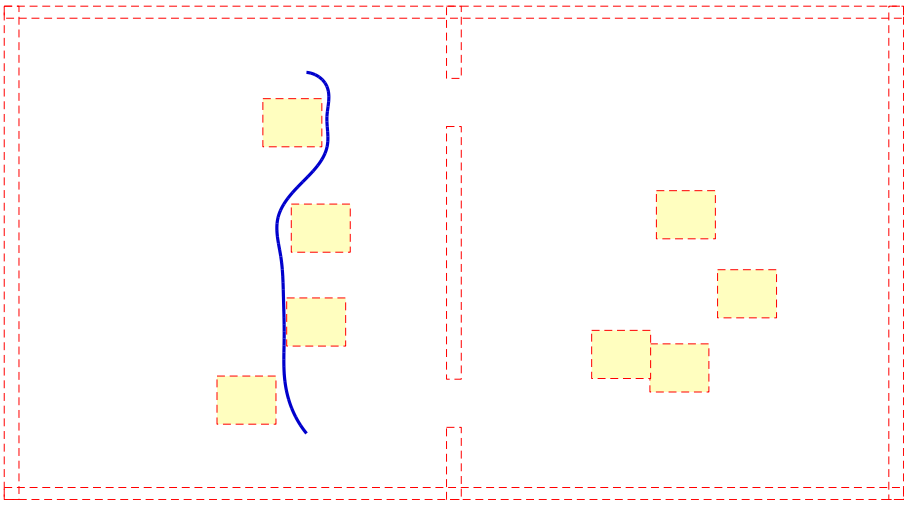

Figure 3.2: Example of an optimized clothoid spline interpolating the path produced by the I-RRT* algorithm

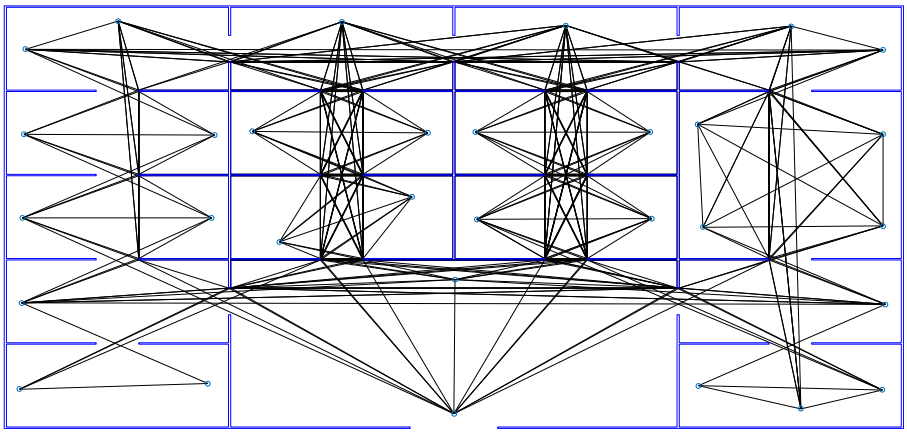

Figure 3.3: Example of visibility graph

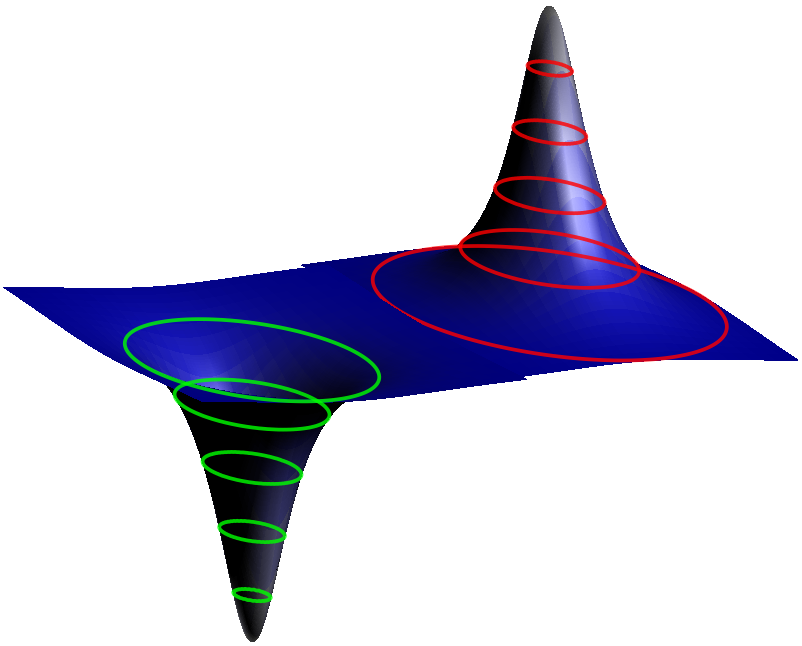

Figure 3.4: Example of the adopted force function: in green level sets for an attractive zone, in red for a repulsive zone.

desired point) by a field

$$\boldsymbol{F} = (F_x, F_y) = \frac{\sigma\alpha}{r^2+\beta}\hat{\boldsymbol{r}}, \tag{3.3}$$

where $\alpha, \beta > 0$ are the parameters of the field, $\boldsymbol{r}$ is the distance of the particle that moves in the field from the center of attraction/repulsion $\boldsymbol{A} = (A_x, A_y)^T$, $\hat{\boldsymbol{r}}$ is the corresponding versor (unit vector), and of course we have $r^2 = (x - A_x)^2 + (y - A_y)^2$, $\sigma = \pm 1$ is the sign that specifies whether the effect is attractive or repulsive. This function should not be thought of as a proper "force". Rather, it can be used to specify the "viscosity" (or the stimulus) experienced by the user. For instance if we encounter a crowd, this function expresses the "friction" experienced by the user, which is related to the density of people.
An important feature of our approach (as described in the previous section) is that once we determine the terminal and the intermediate points (e.g., by using RRT*) the solution of the optimal path is extremely quick, being based on semianalytical or closed form solutions of the optimal control problem. If we want to retain the same advantage also when planning the motion across crowded areas, it is key to set up the problem as the solution of a simple linear ODE (as shown in a previous work of Frego et al. in the context of motion planning for cars [10]). For simplicity we will modify only the coefficient of the drag (Reynolds constant) according to a density field $f(x, y) = ||\boldsymbol{F}||$ of the area in which we are travelling similar to the one shown in Equation (3.3). We suppose the effect of the density function to be active inside a circle of radius r around the point $\boldsymbol{A}$, and $f = 0$ outside. The instantaneous coefficient c will be then equal to $c = f(x(s), y(s))$ at each abscissa s, where $(x(s), y(s))$ is a point on the clothoid. It is not possible to find a closed form of the ODE for the velocity over a general function f, hence we have to sample it in opportune areas and consider a piecewise constant approximation of f for c. If the density function f is constant inside the circle, then the computation becomes easy to handle. We assume that f has the property of being constant for points at the same distance from the center of the circle, see for example, figure 3.4. It is convenient to sample f choosing a suitable partition of the interval of the possible coefficients c in $[f_{\min}, f_{\max}]$. In this way, the intervals of the original clothoid arc are found by intersecting the clothoid with the concentric circles around A.

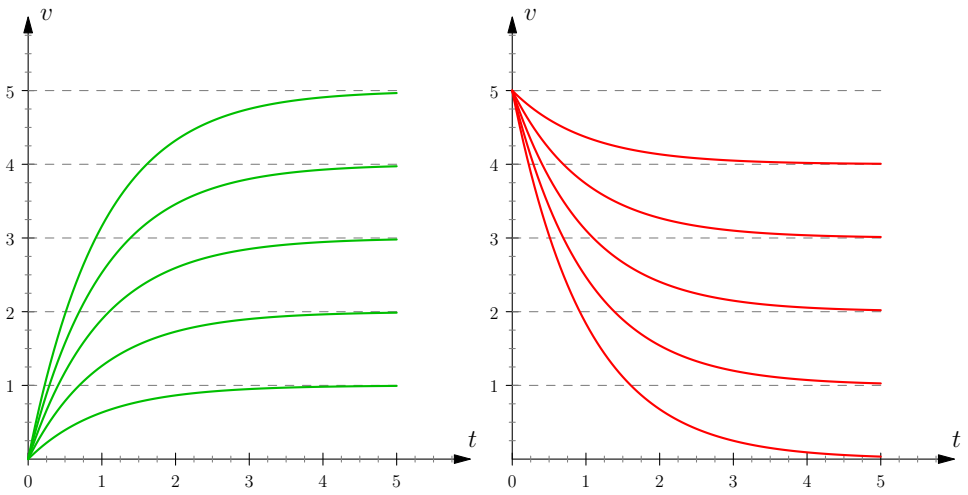

Figure 3.5: Example of the adopted velocity function: left, the case of increasing velocity ($v_0 < v_\infty$), right the case of decreasing speed ($v_0 > v_\infty$). v_∞ is the asymptotically constant velocity.

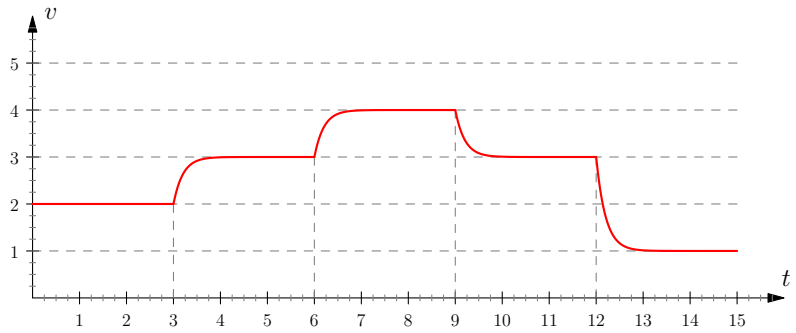

Figure 3.6: Example of a possible speed profile when walking in different conditions, the transitions connecting two different zones of constant velocities have a steep coefficient that can be modelled with the parameter a of (3.4). The higher a the steeper the transition.

The speed profile we adopt for the user motion simulates a piecewise constant function but with continuous transitions between different velocities. To model this behaviour we use a classic linear ODE with the characteristic of being monotone with a horizontal asymptote. The ODE for this is

$$\dot{v}(t) = -av(t) + b, \qquad a, b \in \mathbb{R},\ a \geq 1,\ b > 0,$$

the initial condition to solve the associated Cauchy Problem is $v(t_0) = v_0$, then the closed form solution is

$$v(t) = \frac{b}{a} + (v_0 - b/a)e^{-a(t-t_0)} = v_\infty + (v_0 - v_\infty)e^{-a(t-t_0)}, \tag{3.4}$$

where $v_\infty = b/a$ is the asymptotic velocity. The behaviour of the function is plotted in Figure 3.5 for the two cases $v_0 < v_\infty$ (v is monotone increasing) and $v_0 > v_\infty$ (v is monotone decreasing), the case $v_0 = v_\infty$ is the case of v constant. The composition of a path with different velocities because of different scenarios produces a speed profile like the one in Figure 3.6. To use it successfully, we can find parameters a and b such that the transition is steep enough and the asymptotic velocity v_∞ is the desired velocity. This can be obtained easily by choosing a suitable a and then set $b = av_\infty$. Therefore it is convenient to directly consider a sampling of constant velocities in concentric circles around the centres of attraction/repulsion of the force field, see figure

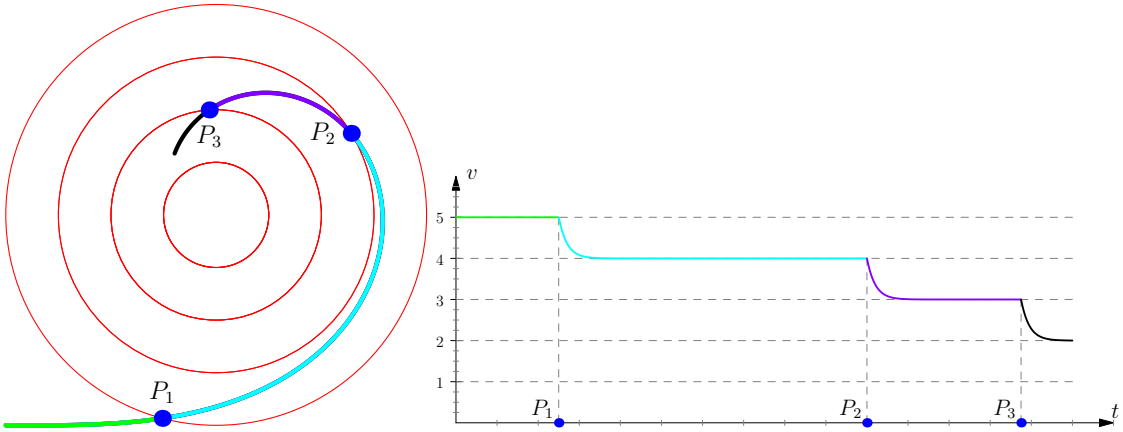

Figure 3.7: Intersection of a clothoid with a (repulsive) force, on the right the corresponding decreasing speed profile when the path approaches the center of repulsion.

3.7. Once the velocities are assigned on each clothoid subarc, we need to compute the time required to travel that piece of curve. This can be obtained integrating the equation of the speed profile (3.4) to retrieve the space travelled and then invert the relation for the total time. The computation can be done analytically, which is very important in order to use (many times) this procedure as a submodule of a master algorithm that considers a family of possible trajectories. The first step is to integrate (3.4) which gives an expression for the space:

$$s(t) = \frac{b}{a}t + \frac{b/a - v_0}{a}e^{-a(t-t_0)} = v_\infty t + \frac{v_\infty - v_0}{a}e^{-a(t-t_0)}.$$

The inverse of the function $s(T) = L$ with respect to T, i.e. the total time T required to travel a distance L on the clothoid subarc, is given in terms of the special function LambertW. The expression for T is:

$$\begin{aligned} T(L) &= \frac{aL}{b} + \frac{1}{a}\text{LambertW}\left(\frac{(av_0 - b)e^{-(a^2L)/b + at_0}}{b}\right) \\ &= \frac{L}{v_\infty} + \frac{v_\infty}{b}\text{LambertW}\left((v_0/v_\infty - 1)e^{-(aL)/v_\infty + at_0}\right). \end{aligned}$$

The LambertW function is defined as the inverse function of the problem $z = W(ze^z)$ and in general is defined over the complex numbers and is real if the argument is greater than $-e^{-1}$. The next theorem shows that the above expression for T yields always a real value for any admissible value of L.

Theorem 1. *The expression for* $T(L)$

$$T(L) = \frac{L}{v_\infty} + \frac{v_\infty}{b}\text{LambertW}\left((v_0/v_\infty - 1)e^{-(aL)/v_\infty + at_0}\right)$$

is well defined for feasible lengths $L \geq t_0 v_\infty$.

Proof. The space is a monotone increasing function for positive velocities, thus the minimum value that L can assume is given by the value of $s(t_0)$ which is equal to

$$L \geq L_{\min} := s(t_0) = t_0 v_\infty + \frac{v_\infty - v_0}{a}.$$

The argument of the LambertW is $(v_0/v_\infty - 1)e^{-(aL)/v_\infty + at_0}$ and the sign of the (positive) exponential function is given by the sign of $(v_0/v_\infty - 1)$. On an arc of decreasing speed, $v_0 \geq v_\infty$ and hence the argument of the

LambertW function is positive. On the other hand, if the speed is increasing, $v_0 < v_\infty$ and we can bound the corresponding factor in the argument of LambertW with -1. We have then to check if the exponential $e^{-(aL)/v_\infty + at_0} \leq e^{-1}$, or taking the logarithm and simplifying, if

$$a\left(\frac{L}{v_\infty} - t_0\right) \geq 1.$$

Using $a \geq 1$ we simplify the condition to $\dfrac{L}{v_\infty} - t_0 \geq 1$ which gives

$$t_0 v_\infty \leq L \leq L_{\min} = t_0 v_\infty + \frac{v_\infty - v_0}{a}.$$

Thus the function $T(L)$ is always well defined for admissible values of L and never takes complex values. □

Taking advantage of the result just shown, we can execute Algorithm 1, where the cost is no longer a combination of jerk and path, but of jerk and time to travel. The latter is found using the function $T(L)$ discussed above.

3.5 Map representation

Maps for the different activities of interest for ACANTO (e.g. museums, shopping malls, ...) are stored and internally represented using SpatialLite [11]. SpatialLite is an open source library supporting the modelling of physical and geometrical data, and allowing the users to perform queries involving space and geometry extremely efficiently. For example, it is possible to find all the geometric shapes belonging to a certain region, or all the points of interest within a certain radius from a given point. In our case, for each floor of a building, we store all the polygons representing the static obstacles (e.g. walls), and, in a different table, all the points of interest. For each PoI, we keep track of its type, associated tags (e.g. mobility constraints) and additional properties (e.g. time required to perform the associated action, cost, ...).
Moreover, special points of interest are stored to model the indoor connections, i.e. the connections between the different floors (e.g. lifts). During the formalization of the high-level activity, only the set of PoIs reachable according to the mobility constraints of the users (e.g. the possibility to use the stairs, lifts, ...) is considered.

Chapter 4

High level planning and refinement

Action planning based on PDDL allows us to model the high-level actions representing different tasks and activities that the agent (in our context the ensemble user + FriWalk) can perform. An high-level representation of this kind gives us the possibility to express in a simple and effective way user needs and preferences, and therefore to produce plans tailored on the specific profile of each user. The automated planner does not need to consider low-level details on how each action is actually implemented and carried out. For example, if the domain we are describing is about a visit to a museum, a single action could be for example $<$*move* $PoI1$ $PoI2>$ representing the task of transferring the user from point of interest 1 to point of interest 2, or $<$*visit* $PoI1>$ representing the task of seeing and visiting $PoI1$. Preferences may be used to express and model interests and hobbies of the current user, and thus to generate tasks and activities involving topics and subjects which are really compelling and suited for the user. Hard constraints may be used to model medical indications. For example they may model the need to avoid plans longer than a certain overall length, or to avoid regions which are too far away from a bathroom.

The planning problem in PDDL is described by defining a domain for the model, declaring predicates and function symbols used to represent the problem, as well as the possible actions which can be applied to evolve from the current state to possible subsequent states. The planning domain describes general operators and symbols, which are constant and shared by all instances of that particular problem. Then, for each instance of the planning problem that we want to solve, a file defining the specific instance must be generated. This file defines the "concrete" objects for the current instance of the problem (and also their type). The same file defines the initial state (in terms of the initial truth assignment of the predicates and of the numeric values taken by the functions) and all the accepted goal states (in terms of predicates that must hold true and constraints that must be respected). In addition, starting from PDDL2.1, we can define the problem as an optimization (minimization or maximization) over the values of some numerical functions [8]. Moreover, from PDDL3, also hard and soft constraints can be expressed [14]. Soft constraints represent preferences, i.e., conditions that the generated plan should respect, but which could be violated by paying some penalisation cost. On the contrary, hard constraints represent conditions which must always hold true in order for the generated plan to be accepted as a valid solution. Within our framework, specific planning instances are generated and populated by combining static information with data on the expected duration of a transition between different locations. As explained in the previous chapter, such numbers are derived solving the motion planning problem under different realistic scenarios. Each simulation considers a different realistic scenario in terms of crowdedness and other dynamic phenomena of interest affecting the specific environment. Once the problem instance has been defined and filled with all the required data, the high–level planning can be performed to find a valid solution, optimizing the given metric function.

4.1 Formalization of Activities

All the recommended social activities share the same kind of actions (at least from the perspective of the FriWalk and Activity Planner). Also the set of possible constraints and user preferences is similar for all the planned social activities. Thus, the planning domain should be the same for all the planned activities, and define the family of activities supported by our system. In particular, each planned task corresponds to a set of actions that the Activity Execution Engine must be able to perform during the execution of the activity. The different kinds of points of interest, constraints and parameters of the user and the environment are used to generate the planning instance, based on the predicates, functions and types defined in the PDDL domain.
The generated plan is composed by a sequence of points of interest to visit. During the execution of the plan, the Execution Engine should be provided with a list of tasks representing the transfers among the various PoIs, and the actions that the Walker should perform during the visit to a particular point of interest (e.g. display some information, wait for the user to enjoy a painting, ...). For each action composing a task, the relevant parameters (e.g. duration, cost, ...) should be stored.
The domain for our example scenario defines the following types, predicates and functions:

```
(:types
    ;; a Point of Interest
    Poi - object
)
```

```
(:predicates
    ;; true iff. the user is currently at PoI ?p
    (at ?p - Poi)

    ;; true iff. the PoI ?p has not been visited yet
    (to-visit ?p - Poi)

    ;; true iff. PoI ?p can be visited
    (visitable ?p - Poi)

    ;; true iff. PoI ?p2 is directly connected to PoI ?p1
    (link ?p1 ?p2 - Poi)

    ;; true iff. the user is currently not busy
    (idle)
)
```

```
(:functions
    ;; the minimum amount of distance walked so far
    (min-total-distance)

    ;; the maximum amount of distance walked so far
    (max-total-distance)

    ;; the amount of user satisfaction for the current plan
    (satisfaction)

    ;; the minimum distance between two PoIs (determined by running
    ;; many realistic simulations)
```

```
        (min-distance ?p1 ?p2 - Poi)

        ;; the maximum distance between two PoIs (determined by running
        ;; many realistic simulations)
        (max-distance ?p1 ?p2 - Poi)

        ;; the speed at which the user moves
        (speed)

        ;; the amount of time required to visit PoI ?p
        (visit-time ?p - Poi)

        ;; the level of interest of the user to a specific PoI ?p
        ;; (inferred from the user profile)
        (interest ?p - Poi)

        ;; the maximum amount of time elapsed so far
        (max-time-elapsed)
    )
```

In addition, the actions to move the user to the next PoI and visit it are defined as:

```
    ;; an action representing the transfer of the user from the current PoI
    ;; ?from to another destination PoI ?to
    (:durative-action move
      :parameters (?from - Poi ?to - Poi)
      :duration (and (>= ?duration (/ (min-distance ?from ?to) (speed)))
                     (<= ?duration (/ (max-distance ?from ?to) (speed))))
      :condition
      (and
          ;; the action can be executed if the user is actually at
          ;; PoI ?from, and the two PoI are directly connected
          (at start (in ?from))
          (at start (link ?from ?to))
          (over all (link ?from ?to))
          (at start (idle))
      )
      :effect
      (and
          ;; at the end of the execution, the user will be located at
          ;; PoI ?to, and the upper and lower bounds on the total and
          ;; partial walked distances will increase according to the
          ;; statistical values obtained from the simulations and
          ;; suitably defined for each planning instance
          (at start (not (idle)))
          (at start (not (in ?from)))
          (at end (in ?to))
          (at end (increase (max-total-distance) (max-distance ?from ?to)))
          (at end (increase (min-total-distance) (min-distance ?from ?to)))
          (at end (increase (max-partial-distance) (max-distance ?from ?to)))
          (at end (increase (min-partial-distance) (min-distance ?from ?to)))
          (at end (idle))
          (at end (increase (max-time-elapsed)
```

```
                                (/ (max-distance ?from ?to) (speed))))
    )
)

;; the actual action of visiting a specific PoI
(:durative-action visit
    :parameters (?p - Poi ?t - Topic)
    :duration (= ?duration (visit-time ?p))
    :condition
    (and
        ;; the user must be at location ?p, must be idle and
        ;; obviously he should not visit an already visited PoI
        (at start (to-visit ?p))
        (at start (idle))
        (at start (visitable ?p))
        (at start (subject ?p ?t))
        (over all (visitable ?p))
        (at start (in ?p))
        (over all (in ?p))
    )
    :effect
    (and
        ;; at the end of the action, PoI ?p is set as visited
        ;; the satisfaction of the user is increased based on
        ;; his interest on the specific topic ?t, and the
        ;; elapsed time is increased based on the time required
        ;; to visit that specific PoI
        (at start (not (idle)))
        (at end (idle))
        (at end (not (to-visit ?p)))
        (at end (increase (satisfaction) (interest ?p)))
        (at end (increase (max-time-elapsed) ?duration))
    )
)
```

The planning instance modelling our example scenario defines different objects representing the various points of interest:

```
(:objects
    pStart - Poi
    p1 - Poi
    p2 - Poi
    p3 - Poi
    ...
    pGoal - Poi
)
```

In the initialization section, the initial values of the numerical functions are set, together with all the predicates holding in the initial state. Geometric information like the topology of the map in terms of interconnections between the various points of interest, and the upper and lower bounds on the lengths of these links, are determined based on the results of the realistic simulations performed using the map of the environment (randomly populated with sensible attractive/repulsive zones, as described in chapter 2).

```
(:init
    ;; initially the user is located in pStart and idle
    (in pStart)
    (idle)

    ;; the geometric description of the environment is synthesized
    ;; according to the results of the simulations
    (link p1 p2)
    (= (min-distance p1 p2) 21.505800)
    (= (max-distance p1 p2) 25.067700)

    (link p1 p3)
    (= (min-distance p1 p3) 35.000000)
    (= (max-distance p1 p3) 46.640400)

    ...

    ;; all the PoIs are set as visitable locations
    (visitable p1)
    (visitable p2)
    (visitable p3)
    ...

    ;; all the PoIs are initially not visited
    (to-visit p1)
    (to-visit p2)
    (to-visit p3)
    ...

    ;; the total walked distance and the maximum
    ;; elapsed time are initially 0
    (= (min-total-distance) 0)
    (= (max-total-distance) 0)
    (= (max-time-elapsed) 0)

    ;; initially the satisfaction of the user is 0
    (= (satisfaction) 0)

    ;; the level of interest of the user for the different PoIs
    ;; must be defined, based on his profile
    (= (interest p1) 30)
    (= (interest p2) 60)
    (= (interest p3) 15)
    ...

    ;; the time required to visit each PoI
    ;; and the user speed are suitably set
    (= (visit-time p1) 60)
    (= (visit-time p2) 90)
    (= (visit-time p3) 60)
    ...
    (= (speed) 1)
)
```

The last part of the PDDL planning problem defines the goal conditions, the constraints on the final plan, and a possible metric that should be optimized during the synthesis of the plan.

```
    ;; the goal for this planning problem is to be in location pGoal,
    ;; to walk a distance of at least 30m and at most 150m, and to
    ;; last at most 500s
    (:goal
        (and
            (in pGoal)
            (>= (min-total-distance) 30)
            (<= (max-total-distance) 150)
            (<= (max-time-elapsed) 500)
        )
    )

    ;; the optimization criterion adopted for this problem
    ;; is to maximize the satisfaction of the user
    (:metric maximize
        (satisfaction)
    )
)
```

4.2 Temporal Uncertainty

The uncertainty in the duration of actions cannot be handled trivially by considering worst case scenarios in the general case. For example, when the problem involves timed initial literals, representing exogenous events changing the boolean value of some predicate at specific moments of time, or concurrent interfering actions, a simple reduction of the problem by considering only worst-case durations can produce infeasible and invalid plans. In [24], a sound and complete technique to transform PDDL planning problems with uncertain durative actions to plain temporal planning problems without uncertainty is shown. The transformed problem can thus be handled by one of the existing planners supporting temporal problems.
In the presence of exogenous and interfering events, our framework can adopt a similar technique to transform the planning problem and remove uncertainties. Additional constraints on the minimum and maximum length-/duration of a plan can then be defined by considering lower and upper bounds on the walked lengths/elapsed times, as shown in the previous listing. Upper and lower bounds on the length/duration of each $<move\ ?from\ to>$ action are determined by running many realistic simulations, as described in the previous sections, and storing the length and duration of each path connecting PoI $?from$ with PoI $?to$. The generated data can then be processed, to compute for each pair of connected points of interest an upper and a lower bound on their distance. The values for the upper and lower bounds are chosen by keeping a certain percentile of covered situations, and leaving out only a few unlikely situations.

4.3 Automated Planning

Once the PDDL planning domain and problem have been generated, the search for the most appropriate solution implementing the activity can be performed using different strategies. Different state-of-the-art open source tools are capable of solving automated planning problems (e.g. OPTIC [3], Fast Downward [16], SGPlan5 [17]). Each of them has some advantages and disadvantages in comparison with the others, in terms of performance, supported features (e.g. preferences, time-dependent costs, optimization metrics), and licensing.

On the other hand, a custom solution can be designed, considering the particular domain supported by the Activity Planner. Indeed, the problem that the Activity Planner is required to solve consists in finding a sequence of points of interest that the user should visit, according to some preferences and constraints. This problem presents many similarities with other well known problems in literature, i.e. the Orienteering Problem [27] and the Tourist Trip Planning problem [13, 25]. Various algorithms exist to solve these families of problems, providing optimal or sub–optimal solutions, and applicable to problems of different sizes. For example, optimal solutions can be found by representing the problem as a Mixed Integer Linear Programming instance to be optimized by state-of-the-art solvers, when the number of PoIs is limited. Sub–optimal solutions, on the other hand, are usually found with the adoption of custom metaheuristic search techniques (e.g. genetic or ant colony optimization algorithms).
To choose the most appropriate solution to implement our Activity Planner, we compared the performance of an automated planner (OPTIC) applied directly to the high–level PDDL problem with a MILP solver (CPLEX [19]) applied after encoding the problem as illustrated below:

```
/*********************************************************************
 *
 * OPL model and script for Activity Planning
 *
 *********************************************************************/

/*********************************************************************
 *
 * DATA
 *
 *********************************************************************/

// The values specified as "..." represent external parameters that are
// specified in an external file containing all the dynamic data

int     n    = ...; // number of POIs

int   s    = ...; // index of starting POI (corresponding to the current location)
int   g    = ...; // index of final POI (corresponding to the final location)
range   POIs = 1..n;

float maxDist = ...; // constraint on the maximum length of the plan
float maxTime = ...; // constraint on the maximum duration of the plan

// Edges
tuple         edge        {int i; int j;}
setof(edge)   Edges       = ...;
float         dist[Edges] = ...; // length of each Edge
float         time[Edges] = ...; // duration required to walk each Edge

float         score[POIs]     = ...; // score given by the user to each POI
float         visitTime[POIs] = ...; // time required to visit each POI

// Decision variables
dvar boolean x[POIs]; // x[i] true iff. POI i is part of the solution
dvar boolean y[Edges]; // y[i] true iff. Edge i is part of the solution
dvar int t[POIs]; // t[i]==n, n>0, iff. POI i is visited at time n


/*********************************************************************
 *
 * MODEL
 *
```

```
 *********************************************************************************/

// Objective -> maximize the plan score
maximize sum (i in POIs) score[i]*x[i];
subject to {

   // One edge must exit from the start node
   sum (<s,j> in Edges) y[<s,j>] == 1;

   // One edge must enter to the goal node
   sum (<j,g> in Edges) y[<j,g>] == 1;

   // One edge must enter each visited node (except s)
   forall (i in POIs: i != s)
     sum (<j,i> in Edges) y[<j,i>] == x[i];

   // One edge must leave each visited node (except g)
   forall (i in POIs: i != g)
     sum (<i,j> in Edges) y[<i,j>] == x[i];

   // The starting node has time counter 1
   t[s] == 1;

   // To avoid cycles, the time counter must increase each time
   // an edge is crossed
   forall (<i,j> in Edges)
     t[j]-t[i] >= -n*(1-y[<i,j>])+1;

   // Constraint on overall length
   sum (<i,j> in Edges) y[<i,j>]*dist[<i,j>] <= maxDist;

   // Constraint on overall duration
   (sum (<i,j> in Edges) y[<i,j>]*dist[<i,j>])
       + (sum (i in POIs) x[i]*visitTime[i]) <= maxDist;

};
```

```
/*********************************************
 * OPL Data
 *********************************************/

n = 10;
s = 1;
g = 10;

Edges = {
  <1,2>, <1,3>, <1,4>, <1,5>, <1,6>, <1,7>, <1,8>, <1,9>, <1,10>,
  <2,3>, <2,4>, <2,5>, <2,6>, <2,7>, <2,8>, <2,9>, <2,10>,
  <3,2>, <3,4>, <3,5>, <3,6>, <3,7>, <3,8>, <3,9>, <3,10>,
  <4,2>, <4,3>, <4,5>, <4,6>, <4,7>, <4,8>, <4,9>, <4,10>,
  <5,2>, <5,3>, <5,4>, <5,6>, <5,7>, <5,8>, <5,9>, <5,10>,
  <6,2>, <6,3>, <6,4>, <6,5>, <6,7>, <6,8>, <6,9>, <6,10>,
  <7,2>, <7,3>, <7,4>, <7,5>, <7,6>, <7,8>, <7,9>, <7,10>,
  <8,2>, <8,3>, <8,4>, <8,5>, <8,6>, <8,7>, <8,9>, <8,10>,
  <9,2>, <9,3>, <9,4>, <9,5>, <9,6>, <9,7>, <9,8>, <9,10>,
};


dist = [
```

	Automated planning (OPTIC)		MILP problem (Cplex)	
Number of POIs	Score	Time (s)	Score	Time (s)
5 (+2)	57.172	0.16	57.172	0.04
10 (+2)	76.999	TO	84.669	0.22
20 (+2)	54.644	TO	120.37	1.47
30 (+2)	45.592	TO	120.717	2.58

Table 4.1: Comparison of the running time and solution quality using both an automated planner (OPTIC) and a MILP solver (CPLEX) to plan an activity. The maximum allowed time (TO) is set to 90 s, and the size of the problem (number of POIs) increases from 5 to 30 (+2 because two virtual POIs must be introduced to model the initial location and the arrival location).

```
  20,  20,  20,  20,  20,  20,  20,  20,  0,
  20,  20,  20,  20,  20,  20,  20,  20,
  20,  20,  20,  20,  20,  20,  20,  20,
  20,  20,  20,  20,  20,  20,  20,  20,
  20,  20,  20,  20,  20,  20,  20,  20,
  20,  20,  20,  20,  20,  20,  20,  20,
  20,  20,  20,  20,  20,  20,  20,  20,
  20,  20,  20,  20,  20,  20,  20,  20,
  20,  20,  20,  20,  20,  20,  20,  20,
];

score = [ 0,  33 ,  36 ,  0 ,  17 ,  8 ,  26 ,  50 ,  37 ,  0 ];
```

In table 4.1 are shown the performance in terms of execution time (with a timeout of 90 seconds) and quality of the solution for problems with an increasing number of points of interest, from 5 to 30. The automated planner is able to find the optimal solution only for the smallest problem (5 POIs), while for all the others it times out before converging, achieving a really low quality in comparison with the optimal solution. The MILP solver, on the other hand, is able to find the optimal solution for all the problem instances, with a maximum running time of 2.58 s.

4.4 Plan Refinement

During the execution of the planned activity, high-level actions involving user motion are refined into an executable, low level trajectory by running the motion planner described in chapter 3, to find a path connecting the $?from$ with the $?to$ locations indicated in the corresponding $<move>$ action. The motion planner can use information about the current status of the environment, like the presence of crowded zones, to synthesize paths comfortable to follow based on the current situation.

4.5 Planning for groups

Until now we have shown the main concepts and ideas adopted to model and solve the problem of Activity Planning, considering a scenario with a single user. However, in order to achieve the social aspect of "Social" activities, the planner must support a context where an activity is carried out by more than one person. However, since the kind of possible actions, constraints and preferences remains the same, a minor extension of the original model suffices to provide support for a multi-agent context. The planning domain can be extended by introducing a new type of object to represent an Agent, named "User". In addition, a score is given by each user to each PoI. Soft constraints associated to each user (e.g. "the user prefers plans avoiding the use of lifts") are modelled as costs given to the different PoIs and connectors between pairs of PoIs:

```
;; the level of interest of an user ?u to a specific PoI ?p
;; (inferred from the user profile)
(interest ?u - User ?p - Poi)

;; the cost given by an user ?u to a specific PoI ?p
;; (inferred from the user profile)
(poi_cost ?u - User ?p - Poi)

;; the cost given by an user ?u to a link between two PoIs
;; ?p1 and ?p2 (inferred from the user profile)
(link_cost ?u - User ?p1 - Poi ?p2 - Poi)
```

The hard constraints considered by the planner are generated as the conjunction of all the hard constraints of each user. For example, the constraint on the total duration *max-global-time* is determined as:

$$\textit{max\_global\_time} = \min_{u \in \textit{Users}} \textit{max\_time\_allowed}(?u)$$

The metric function to optimize is then the weighted sum of the overall score of the visited PoIs for all the users, and the overall costs deriving from the violation of the soft-constraints:

$$\begin{aligned}\textit{metric} = w_1 \sum_{p \in \textit{Pois}} \sum_{u \in \textit{Users}} x[i]\, \textit{interest}(u,p) \;-\; w_2 \sum_{p \in \textit{Pois}} \sum_{u \in \textit{Users}} x[i]\, \textit{poi\_cost}(u,p) \\ -\; w_3 \sum_{\langle p_1,p_2\rangle \in \textit{Links}} \sum_{u \in \textit{Users}} y[\langle p_1,p_2\rangle]\, \textit{link\_cost}(p_1,p_2)\end{aligned}$$

where $x[i]$ is a Boolean indicator variable, indicating whether PoI i^{th} is part of the solution, $y[\langle p_i, p_j\rangle]$ indicates wheter the link $\langle p_i, p_j\rangle$ is part of the solution, and $w_1, w_2, w_3 \in \mathbb{R}$ are weighting factors, allowing us to give different weights to the overall interest and to the violation of soft constraints involving both PoIs and connectors.

The remainder of this section illustrates activity planning in a multi-agent context with a simple example involving three users. Figure 4.1 shows the topology of the points of interest used for this scenario. The map contains six points of interest $P_1, \ldots, P_6$, with a connection between each pair of them. In addition, a special "virtual" point is defined (P_0), to which are associated both the starting and the final location of the activity (P_S and P_G). Table 4.2 shows the worst-case distance and time required to move from each point of interest to all the others (for simplicity, in this scenario we consider the worst-case walking time equivalent to the distance). Tables 4.3 to 4.5 display the penalty given by each user to each link and point of interest (corresponding to violations of some soft constraints), and the level of interest for each PoI. Table 4.6 shows the resulting plan. It can be seen how the point of interest p_1 is reached from p_S, indeed the penalty for all the other links incoming to p_1 would be greater. Furthermore, p_2 is not visited since the violation of the constraint associated to the first user would be greater than the obtained score.

	P_1	P_2	P_3	P_4	P_5	P_6	P_G
P_S	20	20	20	20	20	20	0
P_1	-	20	20	20	20	20	20
P_2	20	-	20	20	20	20	20
P_3	20	20	-	20	20	20	20
P_4	20	20	20	-	20	20	20
P_5	20	20	20	20	-	20	20
P_6	20	20	20	20	20	-	20

Table 4.2: Worst-case distance (and time) to move from each point of interest to all the others

	P_1	P_2	P_3	P_4	P_5	P_6	P_G
P_S	0 | 0 | 0	0 | 0 | 0	0 | 0 | 0	0 | 0 | 0	0 | 0 | 0	0 | 0 | 0	0 | 0 | 0
P_1	-	0 | 5 | 0	0 | 0 | 0	5 | 5 | 0	0 | 0 | 15	5 | 0 | 0	0 | 0 | 0
P_2	5 | 0 | 0	-	0 | 0 | 0	0 | 0 | 0	0 | 0 | 0	0 | 0 | 0	0 | 0 | 0
P_3	0 | 4 | 0	0 | 0 | 0	-	0 | 0 | 0	0 | 0 | 0	0 | 0 | 0	0 | 0 | 0
P_4	9 | 0 | 0	0 | 0 | 0	0 | 0 | 0	-	0 | 0 | 0	0 | 0 | 0	0 | 0 | 0
P_5	0 | 0 | 8	0 | 0 | 0	0 | 0 | 0	0 | 0 | 0	-	0 | 0 | 0	0 | 0 | 0
P_6	0 | 5 | 0	0 | 0 | 0	0 | 0 | 0	0 | 0 | 0	0 | 0 | 0	-	0 | 0 | 0

Table 4.3: Cost of each link between pairs of points, representing the violation of some soft constraints, taking different values for each user (u_1 | u_2 | u_3)

P_0	P_1	P_2	P_3	P_4	P_5	P_6
-	5 | 5 | 8	0 | 4 | 0	3 | 3 | 5	10 | 5 | 10	5 | 0 | 5	3 | 3 | 3

Table 4.4: Score given from each user to each point of interest (u_1 | u_2 | u_3)

P_0	P_1	P_2	P_3	P_4	P_5	P_6
-	0 | 0 | 0	10 | 0 | 0	0 | 0 | 0	0 | 0 | 0	0 | 0 | 0	0 | 0 | 0

Table 4.5: Penalty given from each user to each point of interest, representing the violation of some soft constraints (u_1 | u_2 | u_3)

nextPOI p_S p_1
nextPOI p_1 p_3
nextPOI p_3 p_4
nextPOI p_4 p_5
nextPOI p_5 p_6
nextPOI p_6 p_G

Table 4.6: Plan generated by the Activity Planner

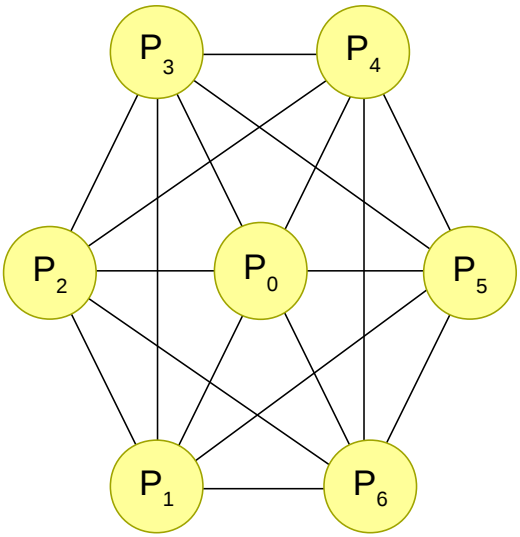

Figure 4.1: Simple multi-agent planning scenario

Chapter 5

Links to other work packages

In this section we highlight the possible links of the Activity Planner with the other work packages composing the ACANTO project.
The Activity Planner is used to expand an activity suggested by the Activity Generator (WP4) into a possible plan realizing that activity. The synthesized plan is then executed with the collaboration of the Reactive Planner (WP5) and the Activity Monitor (WP5), responsible respectively for the refinement (and adjustment) of the plan depending on the specific actual circumstances, and for monitoring the actual plan execution. The high-level plan is represented using the formalisms defined within the WP2, to share a common language to express information across all the ACANTO modules. Each of the high-level actions represent a sequence of low-level actions directly executable by the FriWalk.
Finally, the high-level action planning component of the Activity Planner could be provided within the cloud services (WP7), to relieve the local hardware from the computational burden.

Chapter 6

Conclusions

In this deliverable is presented the general framework implementing the Activity Planner. We described how abstract high-level actions are parametrized with realistic "physical parameters", determined by running many simulations over various different realistic scenarios. Once the simulations have been completed and sensible values for the parameters of the different actions have been determined, an high-level plan can be synthesized, according to the recommendations from the Activity Generator. The produced plan tries to maximize user satisfaction by fulfilling the indicated preferences, while respecting all the imposed hard constraints. When the plan must be executed, the motion planner is invoked to expand the different high-level actions into a sequence of motion primitives, generated trying to optimize user comfort by considering both geometrical and environmental constraints (like for example the crowdedness of a specific zone).
We have shown how all the features required to develop an Activity Planner capable of planning and refining the activities suggested by the Activity Generator can be implemented. Possible research directions to improve the current solution are the introduction of a direct support for probabilistic distributions for the physical parameters, to replace the current approach based on a probabilistic threshold values (considering a certain percentile of the possible outcomes). Additional future work will focus on the most effective ways to model and plan activities for groups of people, and on the translation of the planning problem to a Mixed Integer Linear Program.

Appendix A

Example scenario

In this appendix we show the complete example scenario, with the results produced during each step. The environment for our example scenario is described in chapter 2. Specifically, to keep the number of actions manageable, we consider a subset of the museum composed by ten points of interest. Since this is a simulative example, crowds occupying different zones of the map are generated randomly for each of the various simulations. In the first stage, all the simulations are run and statistics on the length/time required to move from one point of interest to another under different levels of crowdedness are produced. In figure A.1 are shown some of the possible trajectories generated during the simulation stage. For each pair of points of interest, tens of simulations are run with different crowd densities, and all the results are stored for the successive analysis phase.
In table A.1 are shown the generated upper bounds of the times to walk from the point of interest $PoI0$ to all the other points of interest.

PoI	**Upper bound**
$PoI1$	30.74
$PoI2$	29.64
$PoI3$	33.97
$PoI4$	42.06
$PoI5$	40.48
...	
$PoI10$	46.62

Table A.1: Upper bounds of the time to walk from point of interest 1 to all the other connected points of interest

The obtained upper bound values are then injected into the planning problem instance described in chapter 4. The synthesized high-level plan filled with the information and ready to be given as input to the Execution Engine is show in table A.2.
In figure A.2 are depicted the upper bounds for the completion times of the different tasks, while in figure A.3 are shown the upper bounds on the total walked distance after each task. Considering these figures it can be seen how all the constraints on the maximum elapsed time (550) and the maximum walked distance (150) are respected by the synthesized plan.
Eventually, when the high-level plan must be executed, all the <*move*> actions are expanded by the motion planner to comfortable trajectories connecting the two PoIs and avoiding the currently crowded zones. The synthesized paths can thus be followed by the FriWalk, and the activity completed by the user.

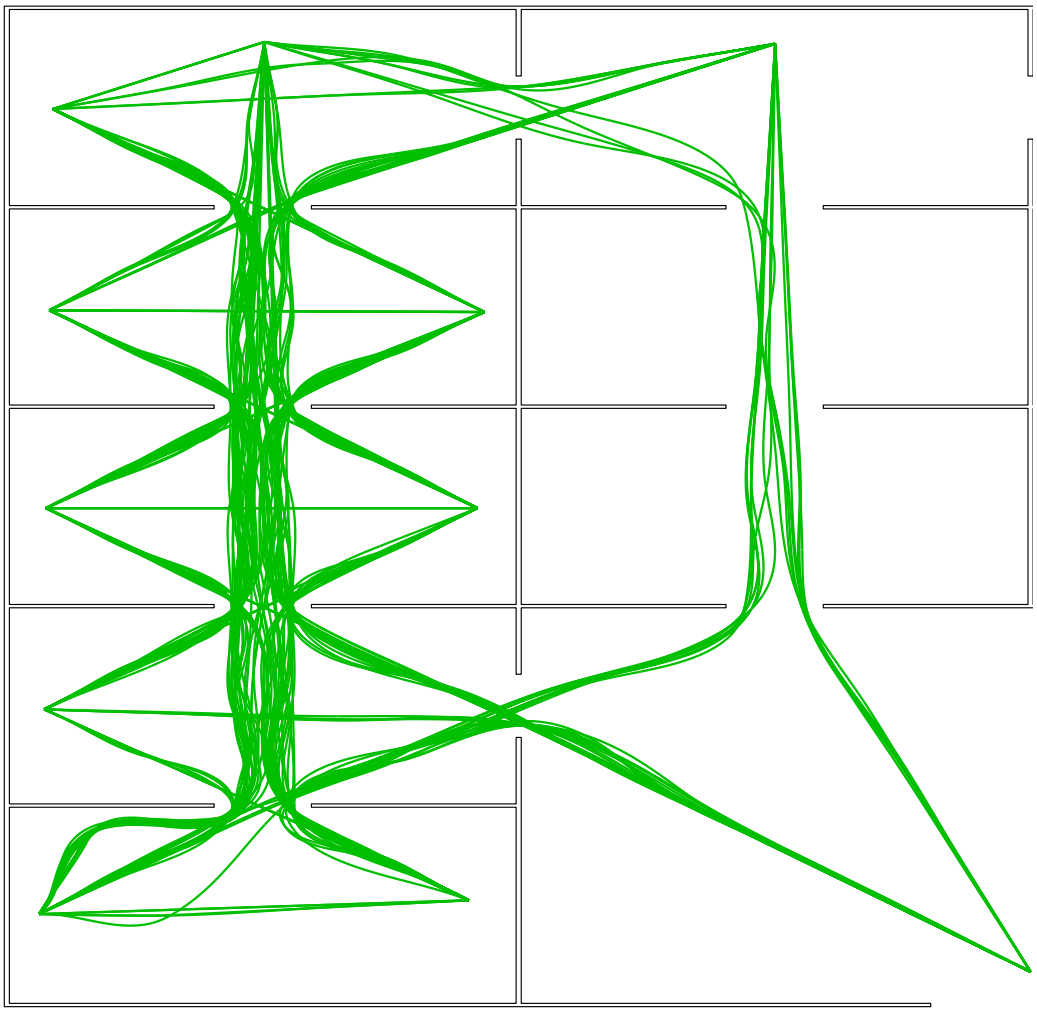

Figure A.1: Example of some possible trajectories generated during the simulation stage

nextPOI(pStart, p1)	*beforeMotion*: < activityExecutionInit > *move*: [pStart – p1] *afterMotion*: < userActivity p1 >
nextPOI(p1, p4)	*beforeMotion*: ∅ *move*: [p1 – p4] *afterMotion*: < userActivity p4 >
nextPOI(p4, p5)	*beforeMotion*: ∅ *move*: [p4 – p5] *afterMotion*: < userActivity p5 >
nextPOI(p5, p6)	*beforeMotion*: ∅ *move*: [p5 – p6] *afterMotion*: < userActivity p6 >
nextPOI(p6, pGoal)	*beforeMotion*: ∅ *move*: [p6 – pGoal] *afterMotion*: < activityExecutionTerm >

Table A.2: Sequence of actions implementing an Activity synthesized by the Activity Planner

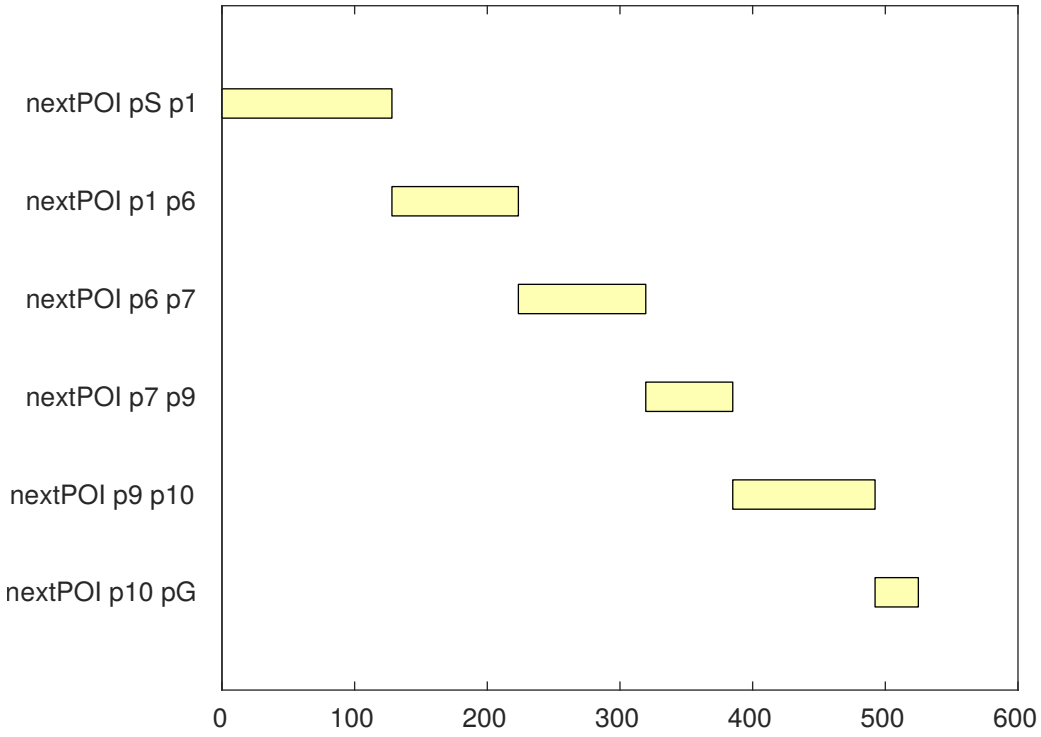

Figure A.2: Sequence of tasks composing the synthesized high-level plan. For each task is shown the maximum (worst case) starting and completion time when all the execution times correspond exactly to the upper bounds.

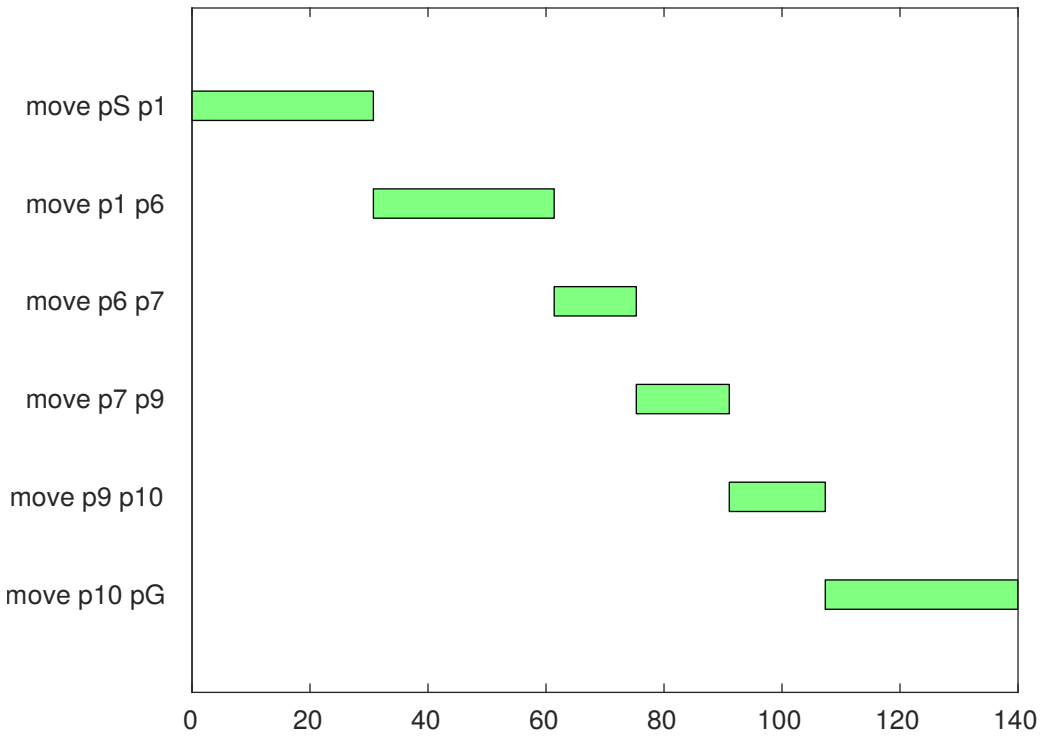

Figure A.3: Sequence of actions composing the synthesized high-level plan involving motion. For each motion action is shwon the maximum possible walked distance (worst case).

Bibliography

[1] Gustavo Arechavaleta, Jean-Paul Laumond, Halim Hicheur, and Alain Berthoz. An Optimality Principle Governing Human Walking. *IEEE Transactions on Robotics*, 24(1):5–14, Feb 2008.

[2] Maren Bennewitz and Wolfram Burgard. Finding solvable priority schemes for decoupled path planning techniques for teams of mobile robots. In *Proc. of the International Symposium on Intelligent Robotic Systems (SIRS)*, 2001.

[3] J Benton, Amanda Jane Coles, and Andrew Coles. Temporal planning with preferences and time-dependent continuous costs. In *ICAPS*, volume 77, page 78, 2012.

[4] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition, 2008.

[5] Enrico Bertolazzi and Marco Frego. G1 fitting with clothoids. *Mathematical Methods in the Applied Sciences*, 38(5):881–897, 2015.

[6] Enrico Bertolazzi and Marco Frego. Interpolating clothoid spline with curvature continuity. *Mathematical Methods in the Applied Sciences*, 2016. Submitted.

[7] Paolo Bevilacqua, Marco Frego, Enrico Bertolazzi, Daniele Fontanelli, Luigi Palopoli, and Francesco Biral. Path planning maximising human comfort for assistive robots. In *IEEE Multi-Conference on Systems and Control*. IEEE Press, 2016.

[8] Maria Fox and Derek Long. Pddl2.1: An extension to pddl for expressing temporal planning domains. *J. Artif. Intell. Res.(JAIR)*, 20:61–124, 2003.

[9] Thierry Fraichard and Alexis Scheuer. From reeds and shepp's to continuous-curvature paths. *Robotics, IEEE Transactions on*, 20(6):1025–1035, Dec 2004.

[10] Marco Frego, Enrico Bertolazzi, Francesco Biral, Daniele Fontanelli, and Luigi Palopoli. Semi-analytical minimum time solutions for a vehicle following clothoid-based trajectory subject to velocity constraints. In *Proc. of European Control Conference 2016 (ECC16)*, 2016.

[11] Alessandro Furieri. SpatiaLite. `https://www.gaia-gis.it/fossil/libspatialite/index`, 2015-09-07.

[12] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 2997–3004. IEEE Press, 2014.

[13] Damianos Gavalas, Charalampos Konstantopoulos, Konstantinos Mastakas, and Grammati Pantziou. A survey on algorithmic approaches for solving tourist trip design problems. *Journal of Heuristics*, 20(3):291–328, 2014.

[14] Alfonso Gerevini and Derek Long. Plan constraints and preferences in pddl3. *The Language of the Fifth International Planning Competition. Technical Report, Department of Electronics for Automation, University of Brescia, Italy*, 75, 2005.

[15] Shilpa Gulati, Chetan Jhurani, Benjamin Kuipers, and Raul Longoria. A framework for planning comfortable and customizable motion of an assistive mobile robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4253–4260. IEEE, 2009.

[16] Malte Helmert. The fast downward planning system. *J. Artif. Intell. Res.(JAIR)*, 26:191–246, 2006.

[17] Chih-Wei Hsu, Benjamin W Wah, Ruoyun Huang, and Yixin Chen. New features in sgplan for handling preferences and constraints in pddl3. 0. In *Proceedings of the Fifth International Planning Competition*, pages 39–42, 2006.

[18] Roger L. Hughes. A continuum theory for the flow of pedestrians. *Transportation Research Part B: Methodological*, 36(6):507 – 535, 2002.

[19] IBM. IBM ILOG CPLEX Optimization Studio. `https://www.ibm.com/software/commerce/optimization/cplex-optimizer/`, 2015-12.

[20] Sertac Karaman, Matthew R. Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. Anytime motion planning using the rrt*. In *International Conference on Robotics and Automation (ICRA)*, pages 1478–1483. IEEE Press, 2011.

[21] Steven M. LaValle. *Planning algorithms*. Cambridge University Press, 2006.

[22] Der-Tsai Lee. Proximity and reachability in the plane. Technical Report R-831, Dept. Elect. Engrg., Univ. Illinois, Urbana, IL, 1978.

[23] Tsai-Yen Li and Hsu-Chi Chou. Motion planning for a crowd of robots. In *International Conference on Robotics and Automation (ICRA)*, pages 4215–4221. IEEE Press, 2003.

[24] Andrea Micheli, Minh Do, and David E. Smith. Compiling away uncertainty in strong temporal planning with uncontrollable durations. In *Proceedings of the twenty-fourth international joint conference on artificial intelligence (IJCAI)*, pages 1631–1637. AAAI Press, 2015.

[25] Wouter Souffriau and Pieter Vansteenwegen. Tourist trip planning functionalities: State–of–the–art and future. *Current Trends in Web Engineering*, pages 474–485, 2010.

[26] Adrien Treuille, Seth Cooper, and Zoran Popović. Continuum crowds. *ACM Trans. Graph*, 25:1160–1168, 2006.

[27] Pieter Vansteenwegen, Wouter Souffriau, and Dirk Van Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research*, 209(1):1–10, 2011.