Taking Into Account Geometric Constraints for Task-oriented Motion Planning

Julien Guitton
ONERA - DCSD
2 avenue Edouard Belin,
31055 Toulouse, France
julien.guitton@onera.fr

Jean-Loup Farges
ONERA - DCSD
2 avenue Edouard Belin,
31055 Toulouse, France
jean-loup.farges@onera.fr

Abstract

Planning a mission for a mobile robot involves the use of a symbolic and a geometric planner. The gap between their internal representation of the environment is an open issue and current researches are conducted without unified formalisms and algorithms. In this paper, we propose to extend the classical planning formalism in order to model actions with geometric preconditions and we propose, develop and test a constraint satisfaction method that aims at defining a destination attitude for motion planning.

Introduction

To accomplish its assigned mission, a mobile robot has to act and move in the environment. Mission planning for mobile robotics involves different reasoning: a symbolic reasoning aiming at choosing the action the robot will have to undertake to achieve the mission, and a geometric reasoning in order to compute the motions of the robot and, more globally, to compute its travels in the environment.

Task planners become more and more efficient and are able to solve complex problems (Hoffmann et al. 2006). Nevertheless, their internal representation of the environment does not allow to deal with geometric information. Coupling a symbolic task planner with a specialized geometric reasoner, as for instance, a motion planner is often the only realistic way to solve robotics problems.

In conventional robotic architectures, as three-layered architectures (Alami et al. 1998), the task planner and the motion planner are decoupled: First a symbolic action plan is found and, then, this plan is refined using geometric information. In this kind of architecture, each layer has its own description of the environment and actions, and the translation of computed results between layers is not easy. Moreover, if one motion needed to achieve an action is not possible, the entire plan has to be invalidated and recomputed.

Another type of architecture includes geometric preprocessing which produces a graph: nodes are computed from the current objectives and geometrical constraints on actions, and arcs may be computed using a motion planning method. The main drawback of this approach is that geometric computations are performed even for actions that will not be selected to belong to the final plan. This kind of architecture is not generic but applied to specific problems (Chanthery, Barbier, and Farges 2005; Baltié et al. 2008).

The idea of conceiving hybrid planning architecture was first proposed by (Kambhampati et al. 1993). More recently, other hybrid planning architectures have been developed (Guitton, Farges, and Chatila 2008; Cambon, Alami, and Gravot 2009) in which symbolic and geometric reasonings are closely related. Figure 1 presents the hybrid planning architecture proposed in (Guitton, Farges, and Chatila 2008). In this planning architecture an interface dispatches parts of the planning problem to a task planner and a path planner. Then, during its planning process, the task planner produces motion requests to the path planner. The answers to the requests and the advices given by the path planner allow an efficient symbolic plan search. Indeed, on one hand, request and answer mechanism enables backtracking not only at the task level but also at the path level and, on the other hand, advices allow a reordering of tasks according to geometric aspects. At the end of the search, the interface gathers actions from the task planner and paths from the path planner to produce a global plan.

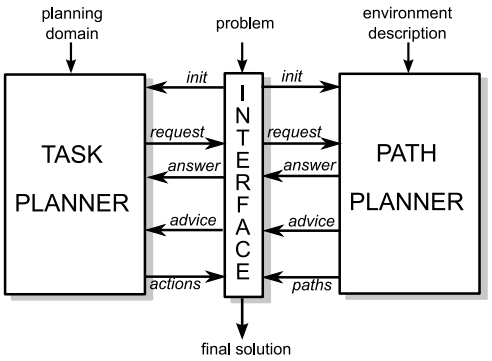

Figure 1: Example of a hybrid planning architecture.

Even with the use of these hybrid planning architectures, the link between task planning and motion planning can be difficult to handle. Some research efforts have been done to improve this link. In (Lamare and Ghallab 1998), the link between the generic planner and an itinerary planner is

done using special symbolic attributes. However, the specialized planner builds its work on an accessibility graph and does not integrate reasoning about the geometric constraints of the problem. The other approaches dealing with the link between task planning and geometric reasoning are mainly approaches for one kind of specific problems (Guéré and Alami 2001; Zacharias, Borst, and Hirzinger 2006). In (Cambon, Alami, and Gravot 2009), a planning architecture is presented, in which the definition of states of the world contains symbolic and geometric information. These data are defined by the use of a set of types and specific predicates that allow to establish a link between symbolic planning and manipulation planning. None of these works propose a clear and reusable way to specify geometric constraints for an action achievement as well as a generic method to handle these constraints.

In this paper, we place ourselves in the context of the planning architecture described in the figure 1 and we present a method allowing to take into account geometric constraints defined at the symbolic action specification level. These geometric constraints aim at defining the necessary motions for the achievement of actions. The proposed method is based on the translation of constraints into a set of mathematical penalty functions and the use of an unconstrained non-linear programming method to solve them. The goal of this work is to go one step toward the possibility for a planning domain designer to express any kind of geometric constraints to model an action.

In a first part, we present the language used to define geometric constraints at the symbolic level. Then, we specify the mathematical representation of these constraints as well as the algorithm allowing to satisfy them. Finally, through an example of mission planning for a mobile robot, we present some results obtained with this model.

Expressing geometric constraints at the symbolic planning level

The achievement of an action may require a specific attitude of the robot, *eg.* a specific position, heading... This attitude can be accurately defined or refer to a subset of the environment. A way to specify a particular attitude is to use geometric constraints between the robot state variables and data from the environment objects (obstacles, waypoints, targets...).

In order to take into account these specific attitudes, we propose to extend the planning operators description to express geometric preconditions. These preconditions are intended to the motion planner.

Augmenting the operators description

A task planning domain, written in the PDDL language (Fox and Long 2003), is a set of planning operators. An operator is defined as a set a preconditions specifying the necessary conditions to achieve the corresponding action, and a set of effects expressing changes in the state of the world resulting from the action achievement. Planning operators can be defined in the same way for the HTN approach (Nau et al. 2003).

We suggest to improve the planning operator description with a new kind of preconditions: the geometric preconditions. Thereby, in order to be selected, an action will have to respect a set of symbolic preconditions and a set of geometric preconditions. Definition 1 presents this new formalism.

Definition 1 (Augmented planning operator)
An augmented planning operator A is defined as a tuple :
$\langle\ head(A), symb\_pre(A), geom\_pre(A), behav(A)$ *eff(A), ref_geom_eff(A)* $\rangle$
where:

- $head(A)$ *is the planning operator header (name);*
- $symb\_pre(A)$ *is the set of symbolic preconditions;*
- $geom\_pre(A)$ *is the set of geometric preconditions;*
- $behav(A)$ *is the set of geometric constraints on behavior during the action;*
- *eff(A) is the set of symbolic effects;*
- *ref_geom_eff(A) is the reference to geometric effects.*

This paper focuses on geometric preconditions. Geometric preconditions consist of statements linking together the symbolic variables and the corresponding elements known by the motion planner, followed by:

- commands of direct assignment of the components of the robot state vector; or
- constraints on the relative attitude that the robot has to meet.

The purpose of direct assignment is mainly the management of the initial state of the robot. Figures 2 and 3 illustrate these two kinds of geometric preconditions.

```
;; Operator !init_rover_attitude
  (Operator (!init_rover_attitude ?r ?o)
;; symbolic preconditions
   ( (location ?o)(not(initialized ?r)) )
;; geometric preconditions
   ((agent ?r)(object ?o)
    (setProperty(?r.x,?o.x))
    (setProperty(?r.y,?o.y))
    (setProperty(?r.heading,0))
;; symbolic effects
   ( (initialized ?r) )
  )
```

Figure 2: Example of a planning operator description using direct assignments.

In these two examples, statements are declared with the predicates `(agent ?r)` and `(object ?o)`. The first predicate indicates the robot which will perform the action, *ie.* the robot for which a path is required. This predicate is not useful in a mono-agent context but appears in the statements for future multi-agent uses. The second statement indicates that the variable `?o` must correspond to an element of the list of the motion planner known objects. `agent` et `object` are keywords that aim at specifying if the symbolic variable corresponds to a mobile robot or a static object of the environment.

```
;; Operator !take_photo
  (Operator (!take_photo ?r ?o ?c)
;; symbolic preconditions
   ( (rover ?r)(available ?r)
     (location ?o)(need_photo_of ?o)
     (has_camera ?r ?c) )
;; geometric preconditions
   ((agent ?r)(object ?o)
    (distance(?r, ?o) >= 50)
    (distance(?r, ?o) <= 100)
    (rel_angle(?r,?o) = cos-and-sin(pi/2))
;; symbolic effects
   ( (has_photo_of ?r ?o) )
  )
```

Figure 3: Example of a planning operator description using geometric constraints.

<precondition>	::	(<statement>* <element>*)
<element>	::	<command> | <constraint>
<statement>	::	(<concept> <var_C>)
<command>	::	c_ident(<arg>$_1$,...,<arg>$_n$)
<constraint>	::	<elt_c> `comparator` <elt_c>
<concept>	::	*agent* | *object*
<C_property>	::	<var_C>.<parameter>
<var_C>	::	`variable`
<parameter>	::	*robot parameter*
<elt_c>	::	<function> | <C_property> | `constant`
<function>	::	f_ident(<arg>$_1$,...,<arg>$_n$)
<f_ident>	::	*function header*
<c_ident>	::	*command header*
<arg>	::	<elt_c>

Table 1: Formal description of the geometric preconditions syntax.

The use of these statements allows to delegate entirely the management of the robot and the geometric data of the environment to the motion planner. The task planner only manipulates labels identifying the robots and objects. Thus, it does not need to be informed about the obstacles positions for example.

The first example (figure 2), illustrates a direct assignment command of the robot state vector components. This assignment is done through the use of the function `setProperty` whose first argument is a component of the robot state vector (*eg.* `?r.x`), and the second argument is either a numerical constant or a value determined by the access to a field of an object data structure of the motion planner (*eg.* `?o.x`).

The second example (figure 3) illustrates, following the statements, a conjunction of geometric constraints on the relative attitude that the robot has to meet in order to achieve the action `!take_photo`.

These constraints relate to the state vector components of the robot model used by the motion planner and are parameterized by the characteristics of the objects on which the task refers. They are expressed in terms of functions (*eg.* `distance` and `rel_angle`), logical comparators and numerical constants.

In the next paragraph, we present a formal description of the geometric preconditions syntax.

Formal description of geometric preconditions

Geometric preconditions expressed at the symbolic task planner level will be interpreted by the motion planner. Therefore, they have to be formulated in a language understandable by this one. Table 1 highlights the basic concepts of this language under a Bacchus-Naur Form.

A geometric precondition is a list of *statements* followed by a list of *elements*. An element is either a *command* or a *constraint*. Statements are used to inform the motion planner of the agents and objects that it will have to manipulate. They allow to link symbolic labels with geometric data. For instance, in figures 2 and 3, `(rover ?r)` and `(object ?o)` are statements. Commands aim at acting directly on the components of the agent's state vectors. For instance, in figure 2, `(setProprety(?r.x,?o.x))` is a command. Finally, constraints are used to describe the state space subsets in which the action is achievable.

These preconditions involve *concepts* that identify *agents* and *objects* jointly handled by both symbolic and motion planners. A set of properties called *concept properties* is associated to each concept. These concept properties correspond to the fields in the data structure of an agent or an object. For an agent, the concept properties are the components of its state vector.

Geometric constraints are formulated with the use of logical comparators specifying the type of the constraint (*eg.* equality or inequality constraint), numerical constants or concept properties, and element of constraint corresponding to geometric functions. These functions are translated by the motion planner into a set of mathematical functions. The translation is done by selecting the mathematical functions corresponding to the element of constraint in a library of functions and derived codes (see table 2 for examples of such functions).

Link between task and motion planning

During a search for a plan allowing to accomplish the given mission, the symbolic task planner checks symbolic preconditions and, in case of success, sends requests to the motion planner. These requests contain the geometric preconditions in which the symbolic variables are totally unified.

When the motion planner receives a direct assignment command, it sets the robot state vector to the corresponding values. This new state vector is now the current robot state vector.

When the motion planner receives a set of geometric constraints, it proceeds in two steps. First, the planner tries to satisfy the geometric constraints, *ie.* it tries to compute a destination state satisfying these constraints as well as the environment constraints. Then, it computes a path between the current state of the robot and the destination state.

In the next section, we present the computation technique used to satisfy the geometric constraints.

Solving geometric constraints

To satisfy geometric constraints, many resolution techniques can be applied as, for instance, genetic algorithms, CSP on continuous domains... We choose to represent geometric constraints in terms of penalty functions at the motion planner level. To solve these geometric constraints, the adopted method is based on a classical and robust non-linear programming technique: the gradient descent.

Formal resolution model

Searching a destination state satisfying the constraints related to the request sent by the task planner is based on the resolution of a problem of non-linear programming in which the optimization variables are the components of the robot state vector r. The criterion to optimize is composed of penalty functions translated from the unverified constraints.

The non-linear programming problem is defined as follows:

$$\min_{r} F(r) \tag{1}$$

with:

$$F(r) = \sum_{i=1}^{i=n} G_i(f_i(r)) \tag{2}$$

where:

- n is the number of mathematical functions derived from constraints;
- G_i is the penalty function associated with the i^{th} constraint;
- $f_i(r)$ is a function deducted from constraint rewriting.

Concerning the penalty function, it is necessary to distinguish the constraints of type equality from the other constraints. When the i^{th} constraint is an equality constraint, we have :

$$G_i(X) = \frac{1}{2}X^2 \tag{3}$$

Otherwise, we have :

$$G_i(X) = \frac{1}{2}X^2\mathcal{H}(X) \tag{4}$$

where $\mathcal{H}(X)$ is the Heaviside function, taking a value of 0 for a negative X *i.e.*, if the constraint is satisfied, and 1 otherwise. The Heaviside function allows to penalize inequality constraints only when they are not satisfied.

The construction of a function $f_i(r)$ by rewriting is done by subtracting a function associated to a constraint element `elt_c` present on the right side of the comparator to the function associated to the constraint element of the left side. If the comparator is $\geq$, the sign of the constructed function is reversed.

Furthermore, some constraints may be expressed by more than one mathematical function. For example, when the received constraint involves angle computations, the model matches it to two numerical constraints. One is related to the cosine of the angle and the other to the sine.

Resolution algorithm

The minimization of the function F (equation 1) is done by gradient descent. The resulting robot state vector is:

$$r_{k+1} = r_k - \lambda_k \nabla F(r_k) \tag{5}$$

where :

- r_k is the value of the robot state at the iteration k;
- λ_k is the optimized step length. At each iteration, λ_k is initialized to 1 and divided by 2 until $F(r_{k+1}) < F(r_k)$;
- $\nabla F(r_k)$ is the gradient value of $F()$ for the robot state at the iteration k.

We have :

$$\nabla F(r) = \sum_{i=1}^{i=n} \nabla F_i(r) \nabla G_i(f_i(r)) \tag{6}$$

with two cases for $\nabla G_i(X)$:

$$\nabla G_i(X) = X \tag{7}$$

when $G_i(X)$ is expressed by the equation 3, and:

$$\nabla G_i(X) = X\mathcal{H}(X) \tag{8}$$

when $G_i(X)$ is expressed by the equation 4.

The computation of $\nabla f_i(r)$ is done by associating the derived codes of each mathematical function of the geometric constraints. Indeed, constructing the functions $f_i(r)$ from the geometric constraints involves only additions, subtractions and sign changes. The impact of these operations on the derivate function is easily computable. Moreover, the constants appearing in the constraints have a zero derivative.

Implementation in the motion planner

The formal resolution described in the previous paragraph allows, starting from the current robot state vector or from a random vector, to compute a destination attitude. The implementation is illustrated by the algorithm 1.

Algorithm 1 Geometric preconditions satisfaction

```
1:  r_0 = S; nbTries = 0
2:  noPathFound = true;
3:  while (noPathFound and nbTries < maxTries) do
4:      k = 0;
5:      while (F(r_k) > 0 and k < maxK) do
6:          compute ∇F(r_k);
7:          compute r_{k+1};
8:          k = k + 1;
9:      end while
10:     if (notInObstacle(r_{k+1})) then
11:         noPathFound = findPath(S, r_{k+1});
12:         if (noPathFound) then
13:             r_0 = Srand() ;
14:         end if
15:     else
16:         r_0 = Srand() ;
17:     end if
18:     nbTries = nbTries + 1;
19: end while
```

This algorithm tries to compute a new robot state that:

- satisfies the geometric constraints (line 5);
- is not in an obstacle (line 10);
- is reachable by the motion planner (line 11).

The algorithm starts by initializing the gradient descent with the current robot state S (line 1). If this initialization does not allow to find a solution, successive random initializations are tried (lines 12 and 15).

During a gradient descent, the algorithm computes the gradient using the equations 6, 7 and 8 (line 5), and computes a new state vector r_k using the equation 5 (line 6).

The optimization of the step length λ_k is done by setting initially the value at 1 and by dividing it by 2 until $F(r_{k+1})$ is greater than $F(r_k)$.

Experiments

After having presented the planning architecture used for the experiments as well as the model of the robot used by the motion planner, we propose to illustrate some interesting results through different examples: In a first part, we explain the computations done by the motion planner during the satisfaction of a constraint. Then, we illustrate the results obtained with this method on a mission planning in which some actions need the satisfaction of geometric constraints. Finally, we show that, in addition to specifying how to achieve an action, the use of geometric constraint and the delegation of motions to a specialized planner allow to simplify the internal representation of the state of the world in the task planner and so, alleviate its computations.

Experimental platform and model of the robot

For the experimentations presented in this section, we used an implementation of the hybrid planning architecture illustrated in the figure 1. This planning architecture is composed of a symbolic planner and a geometric reasoner. The symbolic planner is a HTN planner similar to SHOP2 (Nau et al. 2003) in which the operators are defined using the formalism defined by definition 1. The geometric reasoner is a path planner named CELL-RRT (Guitton, Farges, and Chatila 2009) which is based on the Rapidly-exploring Random Trees algorithm (LaValle and Kuffner 1999) and optimized with a cellular decomposition phase.

The mobile robot used to illustrate our approach is a car-like vehicle. The state of such robot can be described by three configuration variables (x, y, θ) : x and y are the Cartesian coordinates of the robot position and θ is its heading. The robot is considered to move with a constant speed and its turning angle is bounded:

$$-\phi_{max} \leq \phi \leq \phi_{max}$$

The robot kinematic model is defined as follows:

$$\begin{cases} \dot{x} &= v.cos(\theta) \\ \dot{y} &= v.sin(\theta) \\ \dot{\theta} &= \frac{v}{L} tan(\phi) \end{cases}$$

where v is the speed and L is the vehicle's axle length. Computing a robot movement corresponds to integrating this model on a time interval Δt. In order to limit the search space, changes of turning angle are modeled by a three-valued command:

$$\phi \in \{-\phi_{max}, 0, \phi_{max}\}$$

Examples of functions and associated derived codes

Table 2 presents the functions associated to the constraint elements `distance` and `rel_angle` that are involved in the geometric preconditions of the figure 3. The function `distance` is the Euclidean distance function and only uses the robot position. The function `rel_angle` aims at expressing angle constraints between the robot heading and the vector formed by the couple (robot position, object position).

In these examples of functions, we choose to define only useful derived codes. For example, the function associated to `rel_angle` has partial derivatives for the components of the robot state vector related to the position, but we define only the partial derivative for the robot heading because a change of heading can be done by rotating the robot. The derived codes not defined are equal to zero.

Example of constraint satisfaction

From the geometric preconditions:

$$\begin{cases} distance(r, o) \geq 10 \\ distance(r, o) \leq 20 \end{cases}$$

We obtain the following constraints :

$$\begin{cases} c1 = distance(r, o) - 10 \geq 0 \\ c2 = -distance(r, o) + 20 \geq 0 \end{cases}$$

So:

$$\begin{cases} f_1 = \sqrt{(x_o - x_r)^2 + (y_o - y_r)^2} - 10 \\ f_2 = -\sqrt{(x_o - x_r)^2 + (y_o - y_r)^2} + 20 \end{cases}$$

Let G_i be the penalty function used for f_1 et f_2

$$G_i(X) = \frac{1}{2} X^2 \mathcal{H}(X)$$

We obtain:

$$\begin{cases} G_1 = \frac{1}{2}(\sqrt{(x_o - x_r)^2 + (y_o - y_r)^2} - 10)^2 \mathcal{H}(f_1) \\ G_2 = \frac{-1}{2}(\sqrt{(x_o - x_r)^2 + (y_o - y_r)^2} + 20)^2 \mathcal{H}(f_2) \end{cases}$$

The global penalty function is:

$$F = \sum_{i=1}^{i=2} G_i = G_1 + G_2$$

Thus:

$$\nabla F = \begin{bmatrix} \frac{\partial G_1}{\partial f_1} * \frac{\partial f_1}{\partial x} \\ \frac{\partial G_1}{\partial f_1} * \frac{\partial f_1}{\partial y} \\ 0 \end{bmatrix} \mathcal{H}(f_1) + \begin{bmatrix} \frac{\partial G_2}{\partial f_2} * \frac{\partial f_2}{\partial x} \\ \frac{\partial G_2}{\partial f_2} * \frac{\partial f_2}{\partial y} \\ 0 \end{bmatrix} \mathcal{H}(f_2)$$

elt_c	associated functions	some derivatives
`distance(r, o)`	$f_1 = \sqrt{(x_o - x_r)^2 + (y_o - y_r)^2}$	$\frac{\partial f_1}{\partial x_r} = \frac{x_r - x_o}{\sqrt{(x_o - x_r)^2 + (y_o - y_r)^2)}}$ $\frac{\partial f_1}{\partial y_r} = \frac{y_r - y_o}{\sqrt{(x_o - x_r)^2 + (y_o - y_r)^2)}}$
`rel_angle(r, o)`	$f_1 = \frac{(x_o - x_r) * cos(\theta_r) + (y_o - y_r) * sin(\theta_r)}{\sqrt{((x_o - x_r)^2 + (y_o - y_r)^2}}$ $f_2 = \frac{(x_o - x_r) * sin(\theta_r) - (y_o - y_r) * cos(r.\theta)}{\sqrt{((x_o - x_r)^2 + (o.y - r.y)^2}}$	$\frac{\partial f_1}{\partial \theta_r} = \frac{-(x_o - x_r) * sin(\theta_r) + (y_o - y_r) * cos(\theta_r)}{\sqrt{(x_o - x_r)^2 + (y_o - y_r)^2}}$ $\frac{\partial f_2}{\partial \theta_r} = \frac{(x_o - x_r) * cos(\theta_r) + (y_o - y_r) * sin(\theta_r)}{\sqrt{(x_o - x_r)^2 + (y_o - y_r)^2}}$

Table 2: Examples of constraint elements, associated functions and derived codes.

For example, let $r_0 = [20, 100, 0]$ be the initial robot state vector and $o = [50, 50]$ the position of an object, the constraint c_1 is satisfied but not c_2 :

$$distance(r_0, o) = 58.31$$

From the previous computations, we obtain:

$$r_1 = r_0 - \lambda_k \nabla F(r_0) = \begin{bmatrix} 20 + 19.71 \\ 100 - 32.85 \\ 0 \end{bmatrix} \approx \begin{bmatrix} 40 \\ 67 \\ 0 \end{bmatrix}$$

With $\lambda_k = 1$.

Now:

$$distance(r_1, o) = 19.72$$

The constraints c_1 and c_2 are both satisfied.

Planning a mission for a rover

As a mission, a rover has to take photos of 8 different objectives. Figure 4 represents the environment used for the mission. The obstacles are drawn in black and the yellow squares are the objectives. The planning problem is defined as follows :

```
(!init_rover_attitude rover0 location0)
(achieve-goals (list
    (has_photo_of rover0 location1)
    (has_photo_of rover0 location2)
    (has_photo_of rover0 location3)
    (has_photo_of rover0 location4)
    (has_photo_of rover0 location5)
    (has_photo_of rover0 location6)
    (has_photo_of rover0 location7)
    (has_photo_of rover0 location8)
))
```

The planning domain is composed of only two operators (and some methods): `!init_rover_attitude` and `!take_photo`. (figures 2 and 3). We did not write an operator of navigation because motions of the robot are geometric preconditions to the actions achievement. Such an operator would be redundant with the definition of other operators.

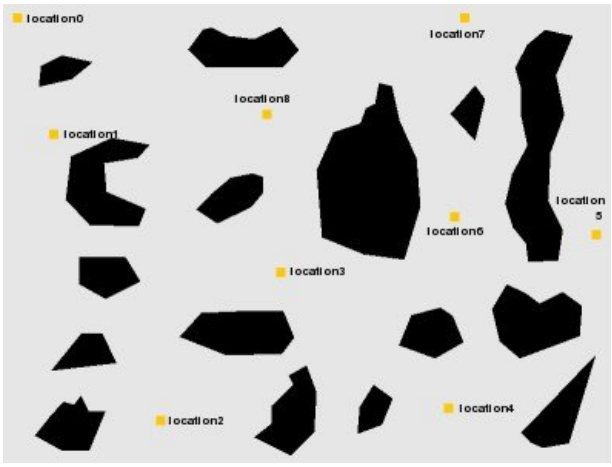

Figure 4: The environment used for the mission.

The initial state is composed of symbolic predicates describing the robot and the environment:

```
(rover rover0)(available rover0)
(camera cam_0)(has_camera rover0 cam_0)
(location location0)(location location1)
(location location2)(location location3)
(location location4)(location location5)
(location location6)(location location7)
(location location8)
```

The motion planner holds geometric knowledges corresponding to these symbolic predicates (table 3).

To achieve the mission, the robot takes 8 photos. For each photo, it places itself perpendicularly to the objective, *ie.* its lateral camera points to the objective. Figure 5 illustrates a solution found by the hybrid planning architecture. The blue dots represent the positions where the actions are undertaken. The green dot is the initial position of the robot and the red one is the final position. The black curve is the path found by the motion planner to achieve this mission.

labels	Cartesian coordinates
location0	(20, 20)
location1	(67, 171)
location2	(206, 542)
location3	(362, 350)
location4	(580, 526)
location5	(772, 301)
location6	(588, 278)
location7	(601, 20)
location8	(344, 145)

Table 3: Coordinates of objects.

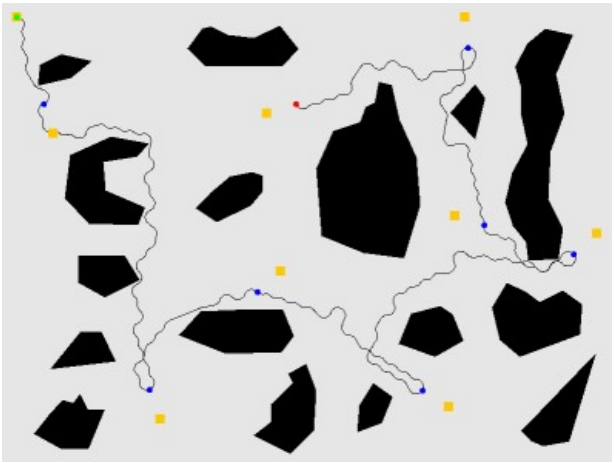

Figure 5: One possible solution found for the mission.

The task planner sends back to the interface the following symbolic plan:

```
SOLUTION PLAN:
1:(!init_rover_attitude rover0
   location0)
2:(!take_photo rover0 location1 cam_0)
3:(!take_photo rover0 location2 cam_0)
4:(!take_photo rover0 location3 cam_0)
5:(!take_photo rover0 location4 cam_0)
6:(!take_photo rover0 location5 cam_0)
7:(!take_photo rover0 location6 cam_0)
8:(!take_photo rover0 location7 cam_0)
9:(!take_photo rover0 location8 cam_0)
```

Additional remarks

In the previous paragraph, we have seen that the new syntax for describing a planning operator alleviates the planning domain. For the same reason, the initial state can be reduced. Indeed, the domain used in the previous example can be formulated only with logical predicates if the geometric preconditions and the robot kinematic model are not taken into account. In this case, possible paths between the objectives are modeled with the predicate `can_traverse(?obj1 ?obj2)` like in the academic planning domain *the rover domain* proposed at the 3rd International Planning Competition[1].

We compare three different methods. First, symbolic approach with blind search, *i.e.* first waypoint is chosen. Then, symbolic approach with nearest-first heuristic search, *i.e.* nearest waypoint of the current robot position is chosen. Finally, an hybrid approach in which a path from one objective to the next one in computed by an external shortest path algorithm from a visibility graph: given a weighted graph $\langle V, E, l : E \rightarrow R \rangle$, find a path β from v to v' so that $\sum l(e), e \in \beta$ is minimal. The first and second methods use an HTN `navigate` method and a `goTo` operator while the third method delegates the navigation to the shortest path algorithm. The graph representing the environment is composed of 70 edges and 1126 vertices. Symbolic approaches correspond to an abstract mission plan search, *i.e.,* the task planner reasons only on logical predicates representing the environment and translated from the visibility graph. We compare the computation time spent by each method to find a solution plan as well as the the solution quality in terms of traveled distance on 15 examples of increasing complexity. In order to be able to compare the solution qualities in terms of traveled distance, we add predicates indicating distances between waypoints: `distance(?obj1 ?obj2 ?value).`

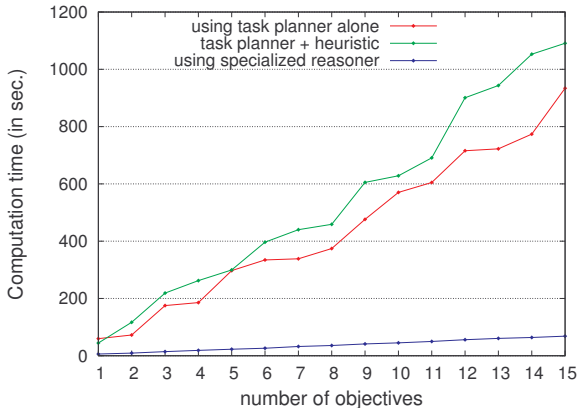

Figure 6: Comparison of the computation time

Figure 6 depicts the evolution of computation time according to the complexity (number of objectives) for the three methods. The computation time is significantly reduced when using a specialized reasoner for paths computation. The evolution of computation time when the number of goals increases is slower. Indeed, adding a new goal increases the time by, in average, 50 seconds for symbolic approaches against 4 seconds for the hybrid approach. The task planner alone or with heuristic takes so long because each time the `navigate` method is applied to build a path, a valid `can_traverse` predicate is searched for unification among a list of 70 predicates. The use of the heuristic in the task planner increases the computation time because it implies the search and ordering of all valid `can_traverse` predicates instead of only one.

[1] http://planning.cis.strath.ac.uk/competition/

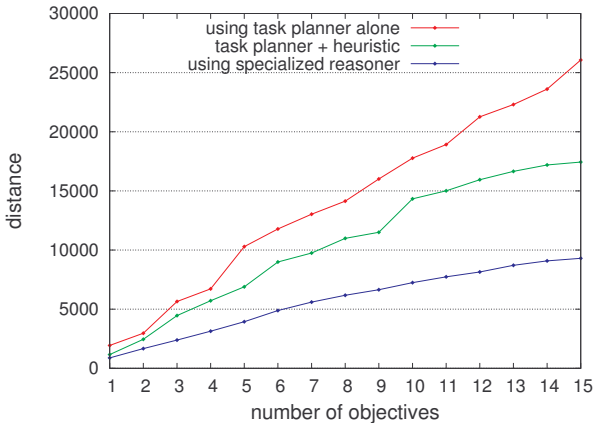

Figure 7: Comparison of the overall distance

The comparison of the overall traveled distance is presented on the figure 7. Using a geometric reasoner allows to obtain plans with a higher quality in terms of path length than classical approaches even when a heuristic to improve the overall traveled distance is used.

Conclusion

In this paper, we proposed a method allowing to take into account geometric constraints at the symbolic description of an action and to satisfy them. These constraints are formulated through the addition of a new set of preconditions, the geometric preconditions, which are sent to the motion planner through planning requests. These geometric preconditions are translated into a set of mathematical penalty functions using a library of corresponding functions and derived codes. Then, from the geometric constraints, the motion planner computes a destination state using a non-linear programming method.

We illustrated the feasibility of our approach through an example of mission planning for a mobile robot and shown that the use of hybrid planning architectures with a specialized reasoner dedicated to the paths computation allows to obtain better performances than the use of a task planner alone reasoning only on the symbolic part of the problem.

Our future investigations will involve an improved functions and derived codes database in order to plan more complex missions. The main future improvement of our work is to extend the formalism described in this paper in order to take into account not only geometric preconditions on the attitude the robot must adopt to achieve an action, but also geometric preconditions allowing to describe the behavior of the robot during the achievement of actions and proper geometric effects allowing to link different symbolic actions through reference to geometric features as, for instance, `go_back_same_location`. Another interesting research tracks is the use of the G function to evaluate already generated paths in order to detect opportunities to achieve actions earlier and to produce advices for the task planner.

References

Alami, R.; Chatila, R.; Fleury, S.; Ghallab, M.; and Ingrand, F. 1998. An architecture for autonomy. *International Journal of Robotics Research* 17:31–337.

Baltié, J.; Bensana, E.; Fabiani, P.; Farges, J.; Millet, S.; Morignot, P.; Patin, B.; Petitjean, G.; Pitois, G.; and Poncet, J. 2008. Multi-vehicle missions : architecture and algorithms for distributed online planning. In *Artificial Intelligence for Advanced Problem Solving Techniques*. Information Science Reference. 1–22.

Cambon, S.; Alami, R.; and Gravot, F. 2009. A hybrid approach to intricate motion, manipulation and task planning. *The International Journal of Robotics research* 28:104–126.

Chanthery, E.; Barbier, M.; and Farges, J.-L. 2005. Planning algorithms for autonomous aerial vehicle. In *16th IFAC World Congress*, volume 16.

Fox, M., and Long, D. 2003. PDDL 2.1: an extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:61–124.

Guéré, E., and Alami, R. 2001. Let's reduce the gap between task planning and motion planning. In *IEEE International Conference on Robotics and Automation*, 15–20.

Guitton, J.; Farges, J.-L.; and Chatila, R. 2008. A planning architecture for mobile robotics. In *1st Mediterranean Conf. on Intelligent Systems and Automation*, 162–167.

Guitton, J.; Farges, J.-L.; and Chatila, R. 2009. Cell-RRT: Decomposing the environment for better plan. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (to appear)*.

Hoffmann, J.; Edelkamp, S.; Thiébaux, S.; Englert, R.; Liporace, F.; and Trug, S. 2006. Engineering benchmarks for planning: the domains used in the deterministic part of ipc-4. *Journal of Artificial Intelligence* 26:453–541.

Kambhampati, S.; Cutkosky, M.; Tenenbaum, J.; and Lee, S. 1993. Integrating general purpose planners and specialized reasoners: case study of a hybrid planning architecture. In *IEEE Transactions on Systems, Man and Cybernetics*, volume 23, 1503–1518.

Lamare, B., and Ghallab, M. 1998. Integrating a temporal planner with a path planner for a mobile robot. In *AIPS Workshop Integrating planning, scheduling and execution in dynamic and uncertain environments*, 144–151.

LaValle, S., and Kuffner, J. 1999. Randomized kinodynamic planning. In *IEEE International Conference on Robotics and Automation*, volume 1, 473–479.

Nau, D.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. SHOP2 : An HTN planning system. *Artificial Intelligence Research* 20:380–404.

Zacharias, F.; Borst, C.; and Hirzinger, G. 2006. Bridging the gap between task planning and path planning. In *IEEE International Conference on Intelligent Robot ans Systems*, 4490–4495.