Velocity Tuning in Currents Using Constraint Logic Programming

Michaël Soulignac[*,**] **Patrick Taillibert**[*] **Michel Rueher**[**]

* THALES Aerospace
2 Avenue Gay Lussac
78852 Elancourt, FRANCE
{firstname.lastname}@fr.thalesgroup.com

** Nice Sophia Antipolis University
I3S/CNRS, BP 145
06903 Sophia Antipolis, FRANCE
rueher@essi.fr

Abstract

Because of its NP-hardness, motion planning among moving obstacles is commonly divided into two tasks: path planning and velocity tuning. The corresponding algorithms are very efficient but ignore weather conditions, in particular the presence of currents. However, when vehicles are small or slow, the impact of currents becomes significant and cannot be neglected. Path planning techniques have been adapted to handle currents, but it is not the case of velocity tuning. That is why we propose here a new approach, based on Constraint Logic Programming (CLP). We show that the use of CLP is both computationally efficient and flexible. It allows to easily integrate additional constraints, especially time-varying currents.

Introduction

Mobile robots are more and more used to collect data in hostile or hardly accessible areas. For physical or strategic reasons, these robots may not be able to receive directly orders from a headquarter in real-time. Thus, they have to embed their own motion planner. Because the environment is often changing or unknown, this planner has to be very reactive.

Motion planning is yet a complex task, answering to two questions simultaneously: where should the robot be, and when? It is known to be a NP-hard problem (Canny 1988). That is to say, the computation time grows exponentially with the number of obstacles.

To guarantee a reasonable response time, motion planning is commonly divided into two simpler tasks: (1) a *path planning* task, dealing with the question *where*, and (2) a *velocity tuning* task, dealing with *when*.

Algorithms associated to these two tasks are generally based on simple assumptions. For instance, obstacles are often modeled as polygonal-shaped entities, moving at constant velocity. Data about weather, in particular about (air or water) currents, are usually ignored.

However, in the case of Unmanned Air Vehicles (UAVs) or Autonomous Underwater Vehicles (AUVs), which may be small or slow, the impact of currents is significant. So, ignoring currents can lead to incorrect or incomplete planners. Such planners may return a physically infeasible path, or no path at all, even if a valid path exists.

Some extensions have been developed in the field of path planning, but currents remain neglected during velocity tuning.

That is why we propose here a new velocity tuning approach, based on Constraint Logic Programming (CLP). Our experimental results show that this approach is computationally efficient. Moreover, it offers a flexible framework, allowing to easily integrate other constraints, such as time-varying currents or temporal constraints.

This paper is organized as follows. Section I recalls the existing planning methods. Section II formalizes the problem of velocity tuning in presence of currents. Section III introduces our modeling of this problem in terms of a Constraint Satisfaction Problem (CSP) on finite domains. Section IV proposes examples of additional constraints. Finally, section V provides some experimental results, obtained on real wind charts.

I. Motion planning in currents

The decomposition of motion planning into path planning and velocity tuning tasks was first introduced in (Kant & Zucker 1986). This decomposition is widely used in robotics because both tasks can be done in a polynomial time.

However, it has to be noticed that it is source of incompleteness: the path planning phase may generate a path which is unsolvable in the velocity tuning phase.

1. Path planning

Path planning methods consist in finding a curve between a start point A and a goal point B, avoiding static obstacles O^i (generally polygonal-shaped). They can be divided into four categories: (1) decomposition methods, (2) potential fields methods, (3) probabilistic methods, and (4) metaheuristics.

Graph decomposition methods (fig. 1a) are based on a discretization of the environment into elementary entities (generally cells or line segments). These entities (plus A and B) are then modeled as nodes of a graph G. The initial -i.e. concrete- path planning problem is thus reformulated into an abstract one: find the shortest path from node A to node B in G. To do this, classical search techniques are applied, such as the well-known A^* algorithm (Nilsson 1969) or one of its numerous variants.

Potential field methods (fig. 1b) (Khatib 1986) consider the robot as a particle under the influence of a potential field U, obtained by adding two types of elementary fields: (a) an attractive field U_{att}, associated to B and (b) repulsive fields U^i_{rep}, associated to obstacles O^i. The point B corresponds to the global minimum of the function U. The path between A and B can thus be computed by applying gradient descent techniques in U values, starting from A.

Probabilistic methods (fig. 1c) (LaValle 1998) are based on a random sampling of the environment. These methods are a particular case of decomposition methods: random samples are used as elementary entities, linked to their close neighbors, and modeled by a graph. Probabilistic RoadMap (PRM) and Rapid Random Trees (RRT) are the most famous methods in this category.

Metaheuristics refer to a class of algorithms which simulate natural processes (fig. 1d) (Zhao & Yan 2005). The three main metaheuristics applied to path planning are: (a) genetic algorithms, inspired by the theory of evolution proposed by Darwin; (b) particle swarm optimization, inspired by social relationships of bird flocking or fish schooling; (c) ant colony optimization, inspired by the behavior of ants in finding paths from the colony to food.

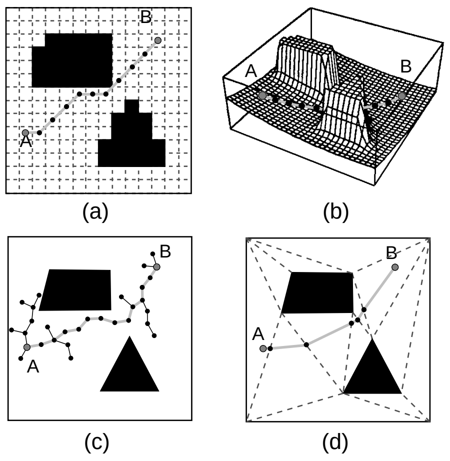

Figure 1: Paths (in light grey) obtained by the following methods: (a) A^* algorithm on regular cells; (b) potential fields; (c) RRT; (d) particle swarm optimization.

All these methods have two common characteristics: (1) the cost $\tau(M, N)$ between two points M and N represents the Euclidean distance $d(M, N)$ and (2) the computed path is made up of successive line segments. This last property is the base of our modeling, described in section II.

However, in presence of currents, the fastest path is not necessary the shortest. To illustrate, let us consider a swirl: the fastest way to link A and B is more circle-shaped than linear.

In this context, new cost functions have been proposed, to make a compromise between following the currents and minimizing the traveled distance (Garau, Alvarez, & Oliver 2005)(Petres *et al.* 2007).

2. Velocity tuning

The existing velocity tuning approaches generally work in a 2-D space-time. The first dimension $l \in [0, L]$ (where L is the length of the path) represents the curvilinear abscissa on the path. The second one, $t \in [0, T]$ (where T is the maximal arrival time), the elapsed time since departure. In this space-time:

- Each point of the path is represented by a column. In particular, start and goal points are represented by the extreme left and right columns.
- Each moving obstacle O^i generates a set of *forbidden surfaces* S^i (often only one). These surfaces contains all couples (l, t) leading to a collision between the robot and O^i. For instance, in figure 2b, the abscissa $l = 10$ is forbidden between $t = 10$ and $t = 15$.

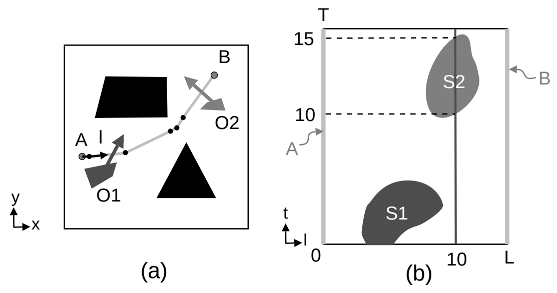

Figure 2: (a) path of fig. 1d, adding two moving obstacles; (b) the corresponding 2-D space-time.

Once the space-time is built, the initial velocity tuning problem can be reformulated into a path planning problem in this space-time. However, this space-time has specific constraints, notably due to time monotony or velocity bounds. Therefore, specific methods have been applied, like: (1) adapted decomposition methods, (2) B-spline optimization, and (3) the broken lines algorithm.

As explained before, decomposition methods (figure 3a) divide the space-time into elementary entities and apply graph search techniques. Since a lot of paths are temporally equivalent (they arrive at the same time), an appropriate cost is necessary. For instance, (Ju, Liu, & Hwang 2002) used a composite cost function balancing the arrival time and the velocity variations.

B-spline optimization techniques (figure 3b) consist in representing the optimal trajectory in the space-time by a B-spline function (Borrow 1988), parameterized by some control points K^i. Graphically, the points K^i locally attracts the curve of the B-spline. Their position is computed in order to minimize the mean travel time, using interior point techniques.

The broken lines algorithm (figure 3c) (Soulignac & Taillibert 2006) tries to link A and B using a unique velocity, i.e. a unique line segment in the space-time. At each intersection of the line with a surface S^i, a velocity variation is introduced, by "breaking" this line into two parts. To sum up, this algorithm tries first to arrive as earlier as possible, and then to minimize velocity variations.

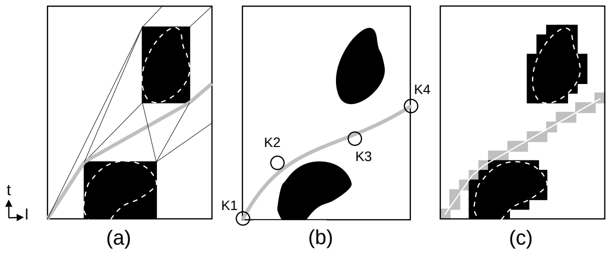

Figure 3: Paths (in light grey) obtained in the space-time of fig. 2 by the following methods: (a) visibility graph; (b) B-spline optimization with 4 control points; (c) broken lines algorithm.

All these methods neglect the influence of currents. This is acceptable in presence of weak currents, since trajectory tracking techniques such as (Park, Deyst, & How 2004) remain applicable to dynamically adjust the robot's velocity.

However, when currents become strong, the robot is neither guaranteed to stay on its path, nor to respect the time line computed by the velocity tuning algorithm. That is why we propose a new approach, based on CLP techniques.

II. Problem Statement

1. Informal description

A punctual robot is moving on a pre-computed path $\mathcal{P}$ from a start site A to a goal site B, in a planar environment containing moving obstacles and currents, with a bounded velocity.

It has to minimize its arrival time at B, with respect to the following constraints: (1) obstacle avoidance and (2) currents handling. Data about obstacles and currents are known in advance.

2. Formalization

The environment is modeled by a 2-D Euclidean space E, with a frame of reference $R = (0, x, y)$. In R, the coordinates of a vector $\vec{u}$ are denoted (u_x, u_y) and its modulus u.

The path $\mathcal{P}$ is defined by a list V of n viapoints, denoted V^i. Each viapoint V^i is situated on $\mathcal{P}$ at curvilinear abscissa l^i. Two successive viapoints (V^i and V^{i+1}) are linked by a line segment. In other terms, $\mathcal{P}$ is made up of successive line segments, which is the result of all path planning methods presented before.

Note that $\mathcal{P}$ is obtained by using adapted cost functions, incorporating the influence of currents (otherwise the velocity tuning would be meaningless).

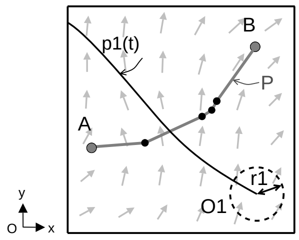

Figure 4: A velocity tuning problem with currents.

Each moving obstacle O^i is a disk of radius r^i. This disk corresponds to a punctual mobile surrounded by a circular safety zone. The position of the mobile -i.e. the center of the disk- is given at every time t by p^i. Note that contrary to most approaches, there is no restriction on the function p^i.

Finally, the current can be seen as a 2-D vector field $\overrightarrow{c}$, known either by measurement or forecasting. Thus, the data about $\overrightarrow{c}$ are, by nature, discontinuous, i.e. defined on the nodes of a mesh (not necessary regular), called *current nodes*. The mean distance between current nodes may correspond to the resolution of measures or the precision of the forecast model.

The robot's velocity vector relative to the frame R (ground speed) is denoted $\overrightarrow{v}$, and its velocity vector relative to the current $\overrightarrow{c}$ (current speed) is denoted $\overrightarrow{w}$.

It is important to understand that $\overrightarrow{w}$ only depends on the engine command, whereas $\overrightarrow{v}$ is impacted by the current $\overrightarrow{c}$. Indeed, applying the velocity composition law, the quantities $\overrightarrow{v}$, $\overrightarrow{c}$ and $\overrightarrow{w}$ are linked by the following relation:

$$\overrightarrow{v} = \overrightarrow{w} + \overrightarrow{c} \tag{1}$$

Our problem consists in finding a timing function σ:

$$\sigma : M \in \mathcal{P} \mapsto t \in [0, T] \tag{2}$$

minimizing the arrival time $t_B = \sigma(B)$, with respect to the following constraints:

1. maximal velocity: the modulus of the robot's velocity relative to the current, denoted w, is smaller than w_{max}. Note that the bound w_{max} only depends on the robot's engine capabilities;
2. obstacles avoidance: the robot has to avoid a set of m moving obstacles;
3. currents handling: the robot has to take into account disturbances due to the field $\vec{c}$.

The quantity T is called *time horizon*. It materializes the maximal arrival date to B. This upper bound may be due to the embedded energy or visibility conditions.

III. Velocity tuning using CLP

Velocity tuning using CLP consists in two steps: (1) defining the constraints describing the velocity tuning problem and (2) solving the corresponding CSP, with the adequate search strategies.

1. Data representation

The constraints above are defined on finite domains. Therefore, the initial data about the environment are reformulated using an appropriate representation.

Time representation

The interval $[0, T]$ is discretized using a constant step ε. The value ε depends on the context. In our applications, $[0, T]$ contains less than 1000 time steps. For instance, $T = 2$ hours and $\varepsilon = 10$ seconds leads to 720 time steps.

Currents representation

As we explained before, the current is known in a finite number of points, called current nodes, obtained by measurement or forecasting. Since current nodes already include an error, we think that it is meaningless to finely interpolate the value of the current between these nodes.

Therefore, we propose the concept of *Elementary Current Area* (ECA). An ECA is an polygonal region of the environment, in which the current is homogeneous. Each ECA contains a unique current node. The value of this node is extended to the whole area.

ECAs are computed by building the Voronoï diagram (Fortune 1986) around the current nodes. This diagram is made up of line segments which are equidistant to the nodes. It is illustrated in figure 5, for uniform and non-uniform distributions.

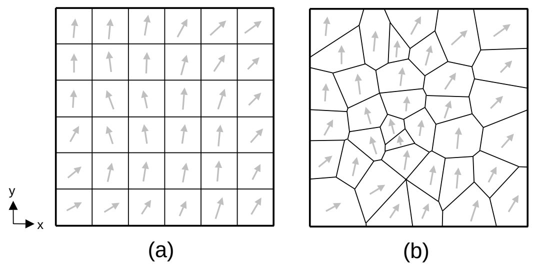

Figure 5: Illustration of ECAs for two distributions of current nodes (grey arrows): (a) uniform and (b) non-uniform.

Artificial viapoints

Artificial viapoints are additional viapoints guaranteeing that the current is constant between two successive viapoints. They are obtained by intersecting the path $\mathcal{P}$ and the borders of ECAs. Since both $\mathcal{P}$ and borders are made up of line segments, these intersections can be computed easily.

The initial list V of viapoints is thus enlarged into V', containing $n' > n$ elements. The current between two successive viapoints V^i and V^{i+1} is denoted $\overrightarrow{c^i}$.

2. Constraints definition

In this part, we show how the velocity tuning problem can be described thanks to two types of constraints: (a) constraints related to currents and (b) constraints related to moving obstacles avoidance.

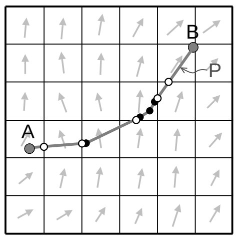

Figure 6: Artificial viapoints (white dots) obtained for the Voronoï diagram of fig. 5a. These viapoints are added to the initial viapoints (black dots).

Note that the currents are constant in time here (time-varying currents are considered in section IV).

a. Constraints related to currents

Let us consider the straight line move $\overrightarrow{d^i}$, between the viapoints $V^i = (x^i, y^i)$ and $V^{i+1} = (x^{i+1}, y^{i+1})$. For this move, we define:

- $\overrightarrow{c^i}$ the velocity of the current
- $\overrightarrow{v^i}$ the robot's velocity relative to the frame R
- $\overrightarrow{w^i}$ the robot's velocity relative to $\overrightarrow{c^i}$

As explained in equation 1, $\overrightarrow{v^i}$ and $\overrightarrow{w^i}$ are linked by $\overrightarrow{v^i} = \overrightarrow{w^i} + \overrightarrow{c^i}$. Moreover, since we want to impose the move $\overrightarrow{d^i}$ to the robot, $\overrightarrow{v^i}$ and $\overrightarrow{d^i}$ are collinear.

Thus, if we denote C^i the result of translating V^i by vector $\overrightarrow{c^i}$, we can build the vector $\overrightarrow{v^i}$ by intersecting:

- The line $\mathcal{L}^i$, of direction vector $\overrightarrow{d^i}$
- The circle $\mathcal{C}^i$, of center C^i and radius w^i

If I_1^i and I_2^i are the intersections obtained[1] (possibly confounded), $\overrightarrow{v^i}$ can be either the vector $\overrightarrow{v_1^i} = \overrightarrow{V^i I_1^i}$ or $\overrightarrow{v_2^i} = \overrightarrow{V^i I_2^i}$. This is illustrated in figure 7.

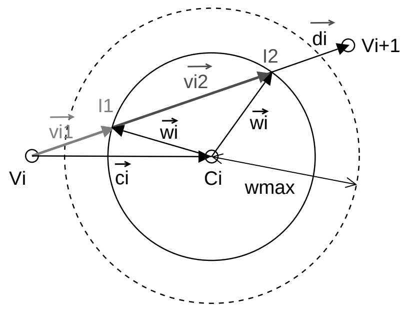

Figure 7: Different possibilities for $\overrightarrow{v^i}$, for $w^i < w_{max}$.

[1]Note that we are sure that at least one intersection exists, because the path $\mathcal{P}$ is supposed to be entirely feasible.

The radius $w^i = w_{max}$ allows to compute the minimal and maximal modulus for $\overrightarrow{v^i}$, denoted v^i_{min} and v^i_{max}:

$$\begin{array}{c} v^i_{min} = \min(v^i_1, v^i_2) \\ v^i_{max} = \max(v^i_1, v^i_2) \end{array} \quad (3)$$

If $\overrightarrow{v^i_j}$ and $\overrightarrow{d^i}$ are not in the same direction, the robot is not moving toward the next viapoint V^{i+1}, but at the opposite (backward move). In this case, to force a forward move, the modulus v^i_j is replaced by 0 in equation 3.

These results allow us to describe the robot's cinematic in presence of currents:

$$\forall i \in [1, n'] : \quad t^i \in [0, T] \qquad (D_{ti})$$

$$v^i \in [v^i_{min}, v^i_{max}] \qquad (D_{vi})$$

$$t^i = t^{i-1} + d^i / v^i \qquad (C_{ti,vi})$$

Note that the quantities d^i, v^i_{min} and v^i_{max} are known and constant:

- The distance d^i is deduced from position of viapoints V^i and V^{i+1},
- The velocity bounds v^i_{min} and v^i_{max} are computed using equation 3.

Therefore, the only variables in the above equations are t^i and v^i (both scalar).

b. Constraints related to moving obstacles avoidance

As explained in section I.2, moving obstacles can be represented in a 2-D space-time (l, t), which l represents the curvilinear abscissa on the path $\mathcal{P}$, and t the elapsed time since departure.

In this space-time, each moving obstacle O^j generates a set of *forbidden surfaces* S^j, containing all forbidden couples (l, t), leading to a collision between the robot and O^j.

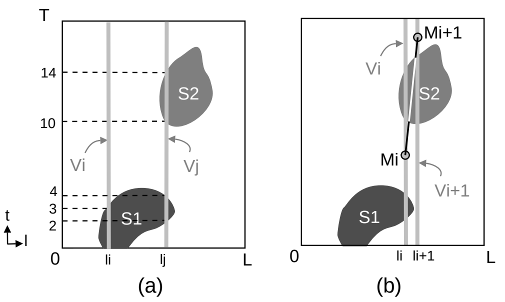

Figure 8: (a) forbidden times for two viapoints V^i and V^j: $F^i = [0, 3]$ and $F^j = [2, 4] \cup [10, 14]$; (b) impossible move between two successive viapoints M^i and M^{i+1}: $M^i \notin F^i$ and $M^{i+1} \notin F^{i+1}$, but $[M^i, M^{i+1}]$ intersects the forbidden surface S^2.

Therefore, for each viapoint V^i, we can define the interval of *forbidden times*, denoted F^i. This interval has the following meaning: if $t^i \in F^i$, then the robot collides a moving obstacle at viapoint V^i. F^i is computed by intersecting all surfaces S^j with the line $l = l^i$.

As shown in figure 8a, F^i is an union of subintervals $F^i_1 \cup F^i_2 \cup ... \cup F^i_{s^i}$, where s^i denotes the number of intersected surfaces.

A first idea to model obstacle avoidance would consist in using the simple constraint:

$$\forall i : \ t^i \notin F^i \qquad (4)$$

However, this constraint is too weak to avoid collisions in all cases.

To illustrate this point, let us consider two successive viapoints V^i and V^{i+1}. In the space-time, the visit of these viapoints is symbolized by two points M^i and M^{i+1}. Even if both points respect equation 4, it is not necessary the case for all intermediate points lying on the line segment $[M^i, M^{i+1}]$. Indeed, this line segment can intersect some forbidden surfaces, as shown in figure 8b.

This problem appears when a forbidden surface is by-passed by one side at point M^i and by the other side at point M^{i+1}. In the example of figure 8b, M^i is above the surface S_2, whereas M^{i+1} is below, which leads to an intersection.

A simple way to avoid this situation is to force all the points of the space-time to be on the same side of each forbidden surfaces. This is modeled by the following constraints:

$$\forall i \in [1, n'] : \quad F^i = [\underline{t_1}, \overline{t_1}] \cup ... \cup [\underline{t_{s^i}}, \overline{t_{s^i}}]$$

$$\forall j \in [1, s^i] : \quad \begin{array}{ll} b_j \in \{0, 1\} & (D_{bj}) \\ t^i \geq \overline{t_j} - T \cdot (1 - b_j) & (C^1_{ti,bj}) \\ t^i \leq \underline{t_j} + T \cdot b_j & (C^2_{ti,bj}) \end{array}$$

The binary variables b_j allow to represent how the forbidden surface S_j is by-passed. Indeed, $b_j = 1$ if the point M^i is above S_j, else $b_j = 0$.

Since these variables b_j are shared by all points M^i, they are forced to by-pass forbidden surfaces in the same way. Combining the variables b_j and T allows to avoid the use of reification techniques: if one constraint is true, the other is naturally disabled (since $\forall i : \ t^i \leq T$).

2. CSP solving

a. CSP formulation

A CSP is commonly described as a triplet (X, D, C), where:

- $X = \cup\{x^i\}$ is a set of variables,
- $D = \Pi\ D^i$ is a set of domains associated to X (D^i represents the domain of x^i),
- $C = \cup\{C^i\}$ is a set of constraints on elements of X.

Using these notations, our velocity tuning problem can be modeled by the following CSP:

- $X = \cup\{v^i, t^i, b_j\},\ i \in [1, n'],\ j \in [1, s^i]$ where n' is the number of viapoints (including artificial ones) and s^i the number of intersected forbidden surfaces by the line $l = l^i$ in the space-time,
- $D = D_{ti} \times D_{vi} \times D_{bj}$,
- $C = C_{ti,vi} \cup C^1_{ti,bj} \cup C^2_{ti,bj}$.

This CSP has the following properties:

- It contains $2n' + \max_i\{s^i\}$ variables and $n' + 2\max_i\{s^i\}$ constraints. In our applications, $n' < 50$ and $\max_i\{s^i\} < 10$.
- All constraints are linear[2].
- Variables are defined on finite domains, with the following sizes:
  - $|D_{ti}| \approx 1000$ (number of time steps)
  - $|D_{vi}| \approx 100$ (number of different velocities)
  - $|D_{bj}| = 2$ (binary variables)

b. Enumeration strategy

Since many solutions are temporally equivalent, we chose the following enumeration strategy:

- Variables ordering: b_j, then t^i, then v^i (in the decreasing order of i).
- Values ordering:
  - increasing values for b_j (to by-pass the forbidden surface by the bottom first)
  - increasing values for t^i (to determine the first valid time steps)
  - decreasing values for v^i (because $v^i \sim O(1/t^i)$)

With this strategy, we try to visit the viapoints as earlier as possible, from the last viapoint to the first viapoint.

The variables b_j allow to roughly identify a first solution, by determining by which side the forbidden surfaces are bypassed. Then, the variables t^i and v^i refine this solution. Note that the enumeration mainly concern the variables t^i, because a value of t^i imposes a value for v^i.

IV. Extension to other constraints

Modeling the velocity tuning problem as a CSP allows to easily integrate other constraints. This section gives two examples: (1) time-varying currents and (2) temporal constraints.

1. Time-varying currents

In a forecast context, values of currents are valid during a time interval ΔT, depending on the application. For instance, in maritime applications, ΔT represents a few hours.

As for ECAs, we find that it is useless to interpolate these data between two intervals. We thus consider that a time-varying current is defined by successive levels, as shown in figure 9.

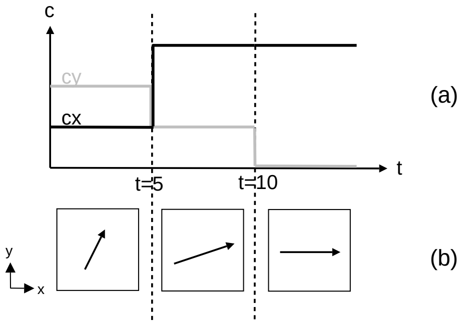

Figure 9: A time-varying current. (a) graph of c_x and c_y functions, defined by levels; (b) the corresponding velocity vector.

Let us consider a current $\overrightarrow{c^i}$, between viapoints V^i and V^{i+1}, changing k times in the interval $[0, T]$. This interval is thus split into $k+1$ subintervals: $[0, t_1]$, $[t_1, t_2]$, ..., $[t_{k-1}, t_k]$, $[t_k, T]$. In each subinterval $[t_j, t_{j+1}]$, the value of the current is constant, denoted $\overrightarrow{c^i_j}$.

The influence of this time-varying current can be modeled in our CSP by using some binary variables. Indeed, the equation (D_{vi}) is replaced by the following constraints:

$$\forall j \in [1, k-1]: \quad \begin{array}{l} b_j \in \{0, 1\} \\ t^i \geq t_j \cdot b_j \\ t^i < (1 - b_j) \cdot T + t_{j+1} \end{array} \tag{5}$$

$$\textstyle\sum_{j=1}^{k-1} b_j = 1 \tag{6}$$

$$v^i \geq \sum_{j=1}^{k-1} b_j \cdot v^i_{min,j} \tag{7}$$

$$v^i \leq \sum_{j=1}^{k-1} b_j \cdot v^i_{max,j} \tag{8}$$

The binary variables b_j allow to identify the subinterval $[t_j, t_{j+1}]$ in which lies the variable t^i. In other terms, $b_j = 1$ if and only if $t^i \in [t_j, t_{j+1}]$. This is modeled by equations 5 and 6.

Then, equations 7 and 8 allows to impose velocity bounds on v^i according to this subinterval. That is, if $t^i \in [t_j, t_{j+1}]$, then $v^i \in [v^i_{min,j}, v^i_{max,j}]$. The values of $v^i_{min,j}$ and $v^i_{max,j}$ are computed as explained in part III.1a, substituting $\overrightarrow{c^i}$ by $\overrightarrow{c^i_j}$.

This model is simple but rough. More precisely, it ignores current changes between two successive viapoints. Therefore, an error is potentially made on velocity bounds. This error remain negligible if the distance d^i between viapoints is small.

[2]After the change of variable $v^{i'} = 1/v^i$.

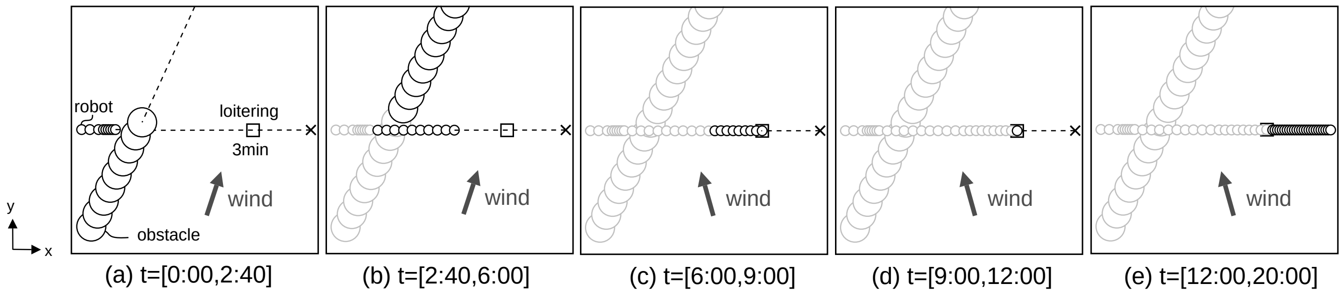

Figure 10: Complete example: (a)(b) moving obstacle avoidance, (c) effect of a current change at $t = 6\text{min}$, (d) loitering during $D = 3\text{min}$ (in black square) and (e) effect of a time window, imposing the arrival at $t = 20\text{min}$.

If it is not the case, d^i can be reduced by artificially subdividing ECAs. By this way, the size of ECAs is decreased and the number of artificial viapoints increased. Therefore, viapoints will be globally closer from each other.

2. Temporal constraints

In this section, we explain how to temporally constrain a viapoint V^i. Especially, we study two temporal constraints particularly mentioned in literature: (a) time windows and (b) loitering.

a. Time windows

A time window W^i is a couple $(\underline{w^i}, \overline{w^i})$, specifying the minimum date $\underline{w^i}$ and the maximum date $\overline{w^i}$ for the robot to visit the viapoint V^i.

In a military context, by example, time windows may correspond to strategic data, such as: "the target will be at V^i between $\underline{w^i}$ and $\overline{w^i}$".

Modeling of W^i is quite natural in our CSP, leading to the single constraint:

$$t^i \in [\underline{w^i}, \overline{w^i}]$$

b. Loitering

The concept of loitering consists in forcing the robot to wait at viapoint V^i for a given duration D^i. From a practical point of view, D^i may correspond to the minimum time required to perform a task at V^i.

Here, our goal does not consist in choosing the best value of D^i, but choosing the best beginning time t^i for the loitering task.

This choice seems to be hard, because it depends both on the moving obstacles and the current changes. However, it can be simply modeled in our CSP, replacing the constraint $(C_{ti,vi})$ by :

$$t^i = t^{i-1} + d^i/v^i + D^i \tag{9}$$

V. Experimental results

This section has two objectives: (1) illustrating our approach and (2) evaluating its performance.

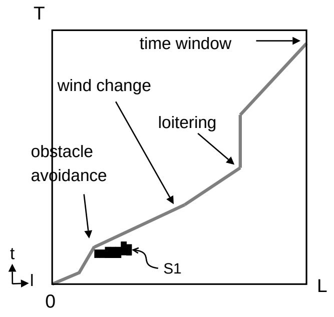

Figure 11: The space-time corresponding to fig. 10

1. Illustrative example

We illustrate here all the constraints presented before through a complete example containing: a moving obstacle, a current change, a loitering task and a time window on arrival.

In this example, simple instances of the constraints have been chosen: (1) the current is uniform on the map and (2) the moving obstacle performs a straight-line move at constant velocity.

The result obtained by our approach is depicted in figures 10 and 11. Figure 10 shows the different phases of velocity tuning in the initial environment, and figure 11 in the space-time.

2. Performance evaluation

In this part, we evaluate experimentally the impact of current changes and moving obstacles on the computation time, in the following conditions:

- **Hardware**: Our approach has been run on a $1.7Ghz$ PC with $512Mo$ of RAM, using the `clpfd` library (Carlsson, Ottosson, & Carlson 1997), provided by Sicstus.
- **Current data**: All data are issued from real wind charts, collected daily during three months on Meteo France website[3] (leading to about 90 different charts). The wind changes are simulated as follows: to simulate k wind

[3]http://www.meteofrance.com/FR/mer/carteVents.jsp

changes, the interval $[0, T]$ is divided into $k + 1$ equal subintervals. A different wind chart is used for each subinterval.

- **Moving obstacles**: As in figure 10, each moving obstacle goes across the environment by performing a straight-line move $P_1 \rightarrow P_2$ at constant velocity. This move is computed in the following way:

1. Two points P_1 and P_2 are randomly chosen on two borders of the environment, until an intersection I between the path $\mathcal{P}$ and the line segment $[P_1, P_2]$ is detected.
2. The velocity of the obstacle is chosen such that the obstacle and the robot are at the same time at point I.

The resulting computation times are provided in table 1. Each cell is the mean time obtained on 100 different environments.

Table 1: Average computation time (in ms), for m moving obstacles and k current changes .

k \ m	0	1	2	3	4	5	6
0	5	9	11	14	17	21	26
1	7	12	13	16	20	24	27
2	10	14	15	18	23	28	29
3	16	21	23	25	34	35	38
4	51	55	56	68	66	67	71
5	80	97	104	106	111	112	114
6	98	127	147	152	159	162	166

From a strictly qualitative point of view, we can observe that the global computation time remains reasonable (a few milliseconds) even in complex environments. Therefore, we think that our approach is potentially usable in on-boards planners.

A theoretical study of the time complexity could confirm these results. In particular, it could be interesting to try different enumeration strategies and evaluate their impact on computational performances.

Conclusion

In this paper, we proposed a velocity tuning approach, based on Constraint Logic Programming (CLP). At our knowledge, this approach is the first able to handle currents. Moreover, this approach is computationally efficient and flexible.

Indeed, we explained that modeling the velocity tuning problem into a Constraint Satisfaction Problem (CSP) allows to easily incorporate more complex constraints, in particular time-varying currents. Moreover, our experiments showed the velocity tuning task could be performed in a polynomial time. It means that our approach is potentially usable in on-board planners.

Further works will investigate the coordination of multiple robots sharing the same environment. In particular, we will study how additional constraints could allow the coordination of fleets of UAVs (Unmanned Air Vehicles).

Acknowledgments

The authors would like to thank Paul-Edouard Marson, Maxime Chivet, Nicolas Vidal and Katia Potiron for their careful reading of this paper.

References

Borrow, J. E. 1988. Optimal robot path planning using the minimum-time criterion. *Journal of Robotics and Automation* 4:443–450.

Canny, J. 1988. *The Complexity of Robot Motion Planning*. MIT Press.

Carlsson, M.; Ottosson, G.; and Carlson, B. 1997. An open-ended finite domain constraint solver. In *Proceedings of Programming Languages: Implementations, Logics, and Programs*.

Fortune, S. 1986. A sweepline algorithm for voronoi diagrams. In *Proceedings of the second annual symposium on Computational geometry*, 313–322.

Garau, B.; Alvarez, A.; and Oliver, G. 2005. Path planning of autonomous underwater vehicles in current fields with complex spatial variability: an a^* approach. In *Proceedings of the International Conference on Robotics and Automation*, 194–198.

Ju, M.-Y.; Liu, J.-H.; and Hwang, K.-S. 2002. Real-time velocity alteration strategy for collision-free trajectory planning of two articulated robots. *Journal of Intelligent and Robotic Systems* 33:167–186.

Kant, K., and Zucker, S. W. 1986. Toward efficient trajectory planning: the path-velocity decomposition. *The International Journal of Robotics Research* 5:72–89.

Khatib, O. 1986. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings of the International Conference on Robotics and Automation*, volume 2, 500–5005.

LaValle, S. M. 1998. Rapidly-exploring random trees: A new tool for path planning. *TR 98-11, Computer Science Dept., Iowa State Univ*.

Nilsson, N. J. 1969. A mobile automation: An application of artificial intelligence techniques. *Proceedings of the International Joint Conference on Artifical Intelligence* 509–520.

Park, S.; Deyst, J.; and How, J. 2004. A new nonlinear guidance logic for trajectory tracking. *Proceedings of the AIAA Guidance, Navigation and Control Conference*.

Petres, C.; Pailhas, Y.; Patron, P.; Petillot, Y.; Evans, J.; and Lane, D. 2007. Path planning for autonomous underwater vehicles. *Transactions on Robotics* 23:331–341.

Soulignac, M., and Taillibert, P. 2006. Fast trajectory planning for multiple site surveillance through moving obstacles and wind. In *Proceedings of the Workshop of the UK Planning and Scheduling Special Interest Group*, 25–33.

Zhao, Q., and Yan, S. 2005. Collision-free path planning for mobile robots using chaotic particle swarm optimization. In *Proceedings of the International Conference on Advances in Natural Computation*, 632–635.