Reactive Navigation in Outdoor Environments

Simon Lacroix, Sara Fleury, Hassan Haddad, Maher Khatib,
Félix Ingrand, Gérard Bauzil, Matthieu Herrb,
Christian Lemaire and Raja Chatila
LAAS-CNRS
7, Avenue du Colonel Roche
F-31077 Toulouse Cedex 4
France

Abstract

The paper presents an approach to reactive navigation in cross-country terrains. The approach relies on a particular probabilistic obstacle detection procedure, that describes the area perceived by a pair of stereo cameras as a set of polygonal cells. To generate the motion commands on the basis of this terrain description, we present some improvements and adaptations to the classical potential fields technique. The algorithms have been implemented within an instance of a generic software architecture, and experimented with the robot Lama.

1 Introduction

Among the wide range of real-world applications for intelligent machines, intervention mobile robots that have to perform tasks in ill-known remote environments has become more and more considered. Several applications of such robots have a potentially big impact (civil security, demining, scientific exploration, field survey), and some of these applications are very demanding for roboticians. Indeed, when communication constraints voids the possibility to efficiently teleoperate the machine (bandwidth, delays and communication windows – typical constraints for planetary rovers for instance), a very high level of autonomy is required. Topics studied within this kind of applications cover the whole range of mobile robotics: data acquisition, terrain modeling, robot localization, path generation and execution, *etc.*

Effective research and development in the area of field robotics trace back to the early eighties, with the ALV project. Since then, several universities and institutes accomplished important achievements, *e.g.* CMU (Ambler, Navlab, UGV, Nomad, Rattler, ...), JPL (Robby and the various Rockies), and MIT. Autonomous navigation is of course the most addressed topic, as it is a basic functionality any intervention robot should be endowed with.

At LAAS, we have addressed the various problems raised by autonomous outdoor navigation [19, 4, 3]. According to a general "economy of means" principle due to limitations of on-board processing capacities, memory and energy, and to achieve an efficient behavior, we have defined an *adaptive approach* to the long range navigation problem: depending on the nature of the terrain to traverse, the robot *adapts* its behavior, by selecting the appropriate perception, trajectory generation and execution control functionalities. Basically, two kinds of navigation modes are considered within this approach:

- **Reflex modes**: when the environment is "easy", *i.e.* essentially flat and lightly cluttered, the robot can efficiently move just on he basis of informations provided by "obstacle detector" sensors: no global environment representation is required, and the motion commands are generated at the pace of data acquisition.

- **Planned modes**: reflex modes become inefficient in more complex environments, in which the robot can be trapped in dead-ends for instance: in such cases, trajectory planners that reason on a model of the environment are required. Depending on the difficulty of the terrain,

a 2D planner using a transversable/non-transversable description, or a 3D planner using an elevation map to check stability and collision constraints have to be activated.

We focus in this paper on a reflex mode[1] (or reactive mode) we have developed and experimented on the robot Lama (figure 1). The principle of this mode is sketched in figure 2: a pair of stereo images is acquired, and a correlation procedure produces a dense 3D point image which is projected onto a grid. The probability for each grid cell to correspond to an obstacle is estimated by a Bayesian classifier. The goal is provided by a visual tracker, and the elementary motion commands are finally generated by a potential algorithm. This sequence is repeated as long as the goal is not reached, at a rate that is determined by the time necessary to process the data: when available, a new local map (or a new goal position) simply replaces the former one. The local maps are not fused into a global model, and the robot position is only defined relatively to the goal by the goal tracker: *no dead-reckoning is required.*

Figure 1: *The robot Lama: a six-wheeled marsokhod type robot, equipped with a pair of stereo cameras mounted on a pan and tilt platform (see section 5.1).*

[1]More details on the general approach and the planned modes can be found in previously published papers [4, 9].

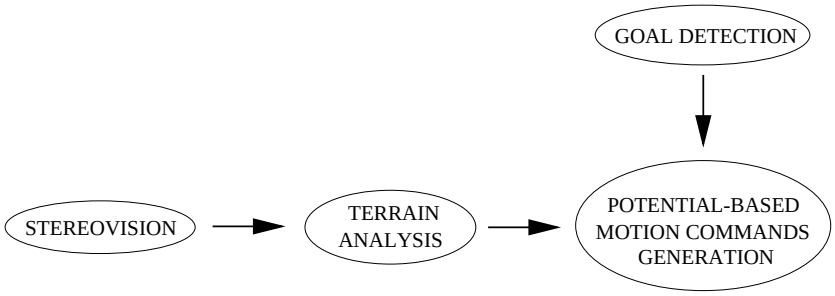

Figure 2: *Principle of the reactive motion approach.*

This mode is to be applied on relatively fair terrains, scattered with few isolated obstacles, and is used to autonomously reach a goal on sight. In our approach to long range navigation, this goal is selected by a high level navigation planner when the environment configuration does not require more precise modeling and motion planning; but the mode can also be considered as a basic functionality to execute sensor-based motion, or to teleoperate the robot: in such a case, the user would define the goal by clicking in an image acquired by the robot for instance.

The paper is organized as follows: the next section presents the perception functionalities that produce the two inputs to the motion generator, *i.e.* the local obstacle map and the goal position. Section 3 is devoted to the core of the system: it presents some improvements and adaptations to the classical potential field technique, that ensure safe and smooth robot motions within the local map. The software architecture within which are embedded and controlled these functionalities is briefly described in section 4. Section 5 presents some experimental results obtained with Lama, and discusses our approach. Proper references to existing contributions are made along the paper.

2 Perception functionalities

All the exteroceptive sensory data required by the reflex navigation mode are provided by the stereo cameras. We briefly describe here the way a pair of stereo images is processed in order to provide 3D informations (a "3D image"), how such an image is analyzed to detect the obstacles on the terrain, and how a visible goal can be tracked along a sequence of images.

2.1 Stereovision

We have implemented a "classical" stereovision correlation algorithm [18, 5], sketched in figure 3. After an optional sub-sampling, images are rectified and their distortion is corrected (using a quadratic radial distortion model) so that the epipolar line of every pixel (i, j) of one image corresponds to the line with the same index (i) on the other image: this allows for several optimization during the correlation phase. Pixel matches are then found using a ZNCC correlation criteria [18] computed on a 11×11 window, and incorrect matches are discarded thanks to *(i)* a reverse correlation and to *(ii)* two thresholds applied on the correlation score curve (value of the highest score, difference between this score and the second highest peak in the curve). A parabolic interpolation is performed to get a sub-pixel disparity estimate.

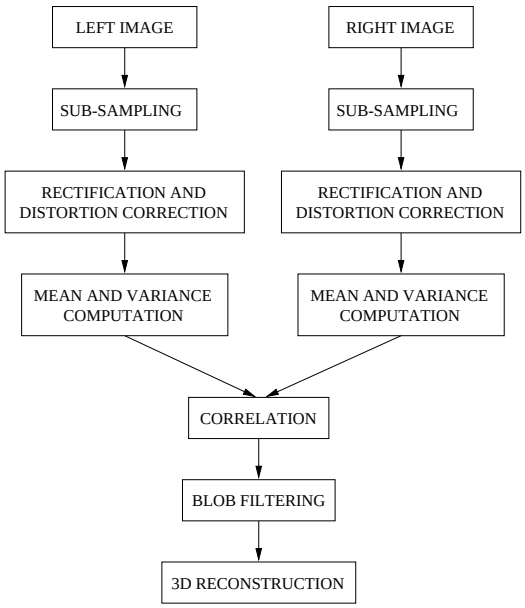

Figure 3: *The various steps of the stereo-correlation algorithm.*

We found the ZNCC correlation criteria extremely efficient: although we use cheap video cameras and low quality lenses (see section 5.1), no noise filtering of the images is required, and it was very easy to determine the values of the two thresholds so that the algorithm produces dense disparity images with very few false matches (which are eventually discarded with a simple blob filter). However, note that there are matching errors we could not

avoid, that occur at the border of two regions of very different intensity values located at different depths (which often happens when perceiving obstacles). Such errors lead to a growing of the obstacle shape in the disparity image, which depends on the size of the correlation window; to our knowledge, no stereo correlation algorithm performed on fixed size windows are able to get rid of such errors, but fortunately, this is not a problem when it comes to avoid obstacles.

Figure 4: *Pixel-based stereovision. From left to right: a stereoscopic image pair acquired with Lama, the disparity image produced by the correlation algorithm, and a top view of the corresponding 3D image.*

To get quantitative informations on the precision of the computed disparity (and therefore on the coordinates of the 3D points), we studied a set of 100 images acquired from the same robot position. As in [21], it appeared that the distribution of the standard deviation on the disparity estimate can be well approximated by a Gaussian. Not surprisingly, we noticed a strong correlation between the shape of the correlation peak and the computed standard deviation: the sharper the peak, the better the precision on the estimated disparity. This correlation allows to estimate *on line* a standard deviation on the disparity, and therefore on the 3D coordinates of the points. Although such considerations are not very useful in the context of this article, where 3D images are only qualitatively analyzed, let's note that with the cameras of Lama, a mean error model on the depth estimate z is well approximated by $\sigma_z = \alpha_z z^2$, where $\alpha_z = 10^{-3}$.

2.2 Terrain classification

The difficulty of representing outdoor environments comes essentially from the diversity of their geometrical and physical nature: such environments are not intrinsically structured, as compared to indoor environments, where

simple geometric primitives match the reality. The problem of obstacle detection on the basis of 3D data on natural fair terrains has been frequently addressed [21, 2, 10, 23, 11]. To our knowledge, all the existing contributions come to a segmentation procedure, be it on the disparity image, on the coordinates of the 3D points or on the estimate of their normals. Moreover, all these techniques produce a *binary* description of the environment, in terms of traversable and non-traversable areas. As a consequence, in order to ensure robot safety, one must force the number of non-detections to be as close as possible to zero, which often induces false alarms, and thus "over-constrains" the problem.

We present here a classification procedure that produces a probabilistically labeled polygonal map. It relies on a specific discretization of the perceived area, that defines a *cell image*. Attributes are computed for each cell, and are used to label the cells thanks to a supervised Bayesian classifier: a probability for each cell to correspond to an obstacle is estimated, which allows to consider costs based on a *risk* to plan paths or to generate motion commands. The discretization being imposed, this procedure is essentially an *identification* process, and does not require any threshold determination (a tedious problem with segmentation algorithms).

2.2.1 Cells definition

The sensors that produce 3D points, be it a laser range finder or a stereovision correlation algorithm, generally have a regular scanning rate[2] within the *sensor frame*. But when perceiving a flat ground with such sensors, the resolution decreases dramatically with the distance to the sensor. This fundamental point lead us to choose a discretization of the perceived zone as presented in figure 5, instead of a Cartesian one such as in [10]. This discretization corresponds to the central projection on a virtual horizontal ground of a regular (Cartesian) discretization in the sensor frame.

The fundamental property of this discretization is the "conservation of density" : on a perfectly flat ground corresponding to the reference plane, the number of points that belong to a cell – *i.e.* whose vertical projection is within cell borders – is equal to a constant *nominal density*, defined by the discretization rates. On the other hand, a cell covering an obstacle area contains much more points than the nominal density. The number of points

[2]The angular scanning rate of a camera is actually not exactly constant.

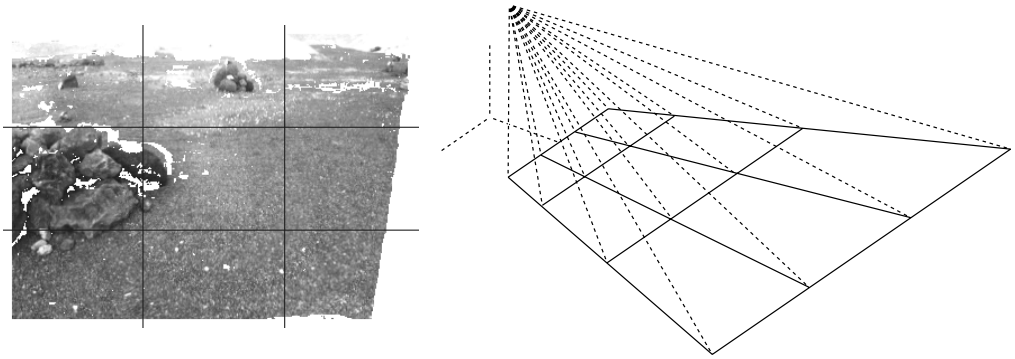

Figure 5: *Discretizations of a 3D stereo image. Left: regular Cartesian discretization in the sensor frame (only the correlated pixels are shown). Right: its projection on the ground (the actual discretization is much finer)*

in a cell is then an important characteristic of the nature of the perceived zone covered by this cell.

Other features are used to identify a cell: the elevation standard deviation and the maximum elevation difference computed on the cell points give an information on the "flatness" of the cell; the mean normal vector and the variances on its coordinates are useful to determine if the cell covers a regular slope or has an irregular surface. The cell distance to the sensor is also an important feature: there is actually no correlation between this distance and the cell's class, but all the former features strongly depend on it. The introduction of this distance feature comes to implicitly taking into account the sensor's uncertainty and resolution properties in the classification process.

2.2.2 Cell classification

A supervised Bayesian classification procedure is used to label each cell. During an off-line learning phase, a human prototypes a set of cell images: for each cell, the corresponding terrain class for the considered robot is determined. These informations associated with the feature vector compose the prototypes data base. On line, once 3D data are acquired and discretized, the feature vector F is computed for each cell. Bayes theorem is then applied to determine the partial probabilities $P(C_i|F)$ for a cell to correspond to each of the M terrain classes $\{C_1, \ldots, C_M\}$:

$$P(C_i \mid F) = \frac{P(F \mid C_i)P(C_i)}{P(F)} = \frac{P(F \mid C_i)P(C_i)}{\sum_{i=1}^{n} P(F \mid C_i)P(C_i)},$$

where $P(F \mid C_i)$ are the probability density functions computed thanks to a nearest neighbor technique applied in the feature space filled with the prototyped data base, and $P(C_i)$ are the *a priori* probabilities to perceive an area that belongs to the class C_i.

This classification procedure is very fast and robust: we tested it on hundreds of images, using various prototype data bases. For the situations considered in this paper, we only consider two terrain classes, *i.e.* flat and obstacle. Figure 6 shows some classification results with images acquired by Lama: note that the result quality decreases with the distance to the sensor, which is not surprising. The *entropy* of a cell is an estimate of the confidence of the classification results.

There are several extensions to the method, not considered for the application context of this article: the discretization is dynamically controlled to allow a finer description, the classification results can be combined with a terrain physical nature classifier using texture or color attributes, and the partial probabilities of a cell to belong to a terrain class allow to perform a fusion procedure of several classified cell images.

2.3 Visual goal tracking

Keeping track of a goal in a sequence of outdoor images is not difficult: indeed, simple correlation techniques have been widely studied and do the job quite well when the environment is textured enough [27]. The only difficulty is to be able to select in an image the areas on which the tracker will not drift. Pixel selection on the basis of the Harris detector, on some texture attributes [26] or more simply on the grey level standard deviation, is sufficient for that. Of course, the determination of the thresholds on these various criteria has to be done considering the tracking algorithm, which calls for some statistical trials.

Work concerning the goal tracker is still under way (in our current experiments, the goal is given as absolute Cartesian coordinates – section 5.1); we are considering three different tracking modes:

- The goal is initially tracked in a sequence of images. We implemented a simple pixel tracker based on the ZNCC correlation criteria computed

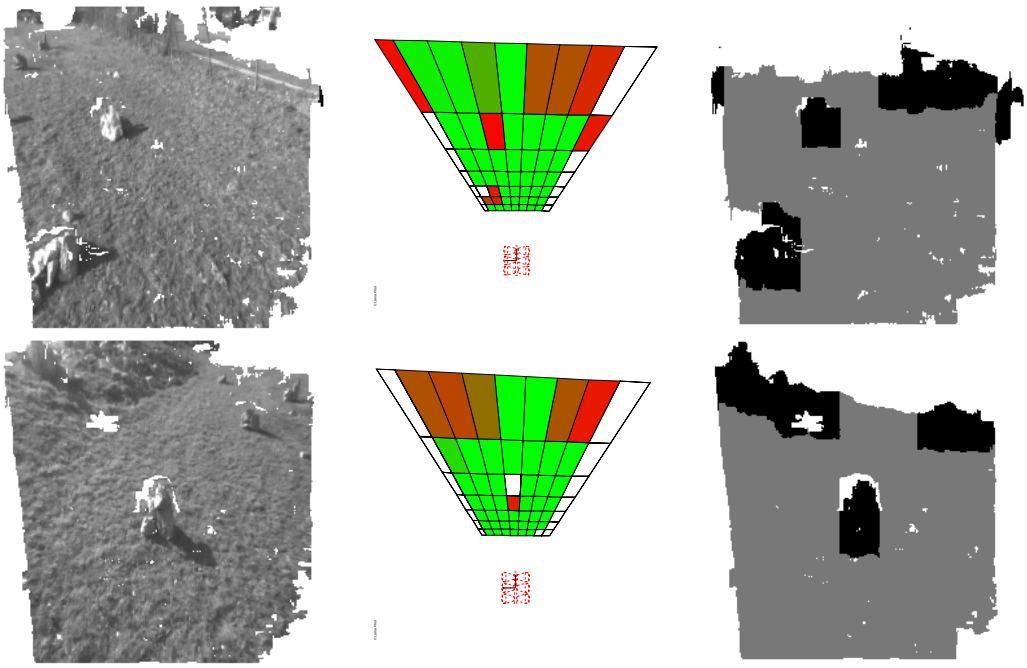

Figure 6: *Examples of classification with Lama images. From left to right: left image of the stereovision pair, partial probabilities of the defined cells to be an obstacle (represented as gray levels), and reprojection of the cells in the sensor frame, after the application of a symmetric decision function. Note that there are some empty cells, and that the ones covering areas farther than* $15m$ *are discarded*

on 15×15 pixels, the template update is done by interpolating the current image around the sub-pixel coordinates of the correlation peak (figure 7). During this mode, the resulting goal that is given to the motion generator is only a direction.

- Once the robot moved a while, the goal Cartesian coordinates can be roughly derived using dead-reckoning and the evolution of its direction in the images.

- Finally, the goal distance to the robot is more precisely estimated as the robot approaches it, thanks to its 3D coordinates that are now produced by the stereovision algorithm.

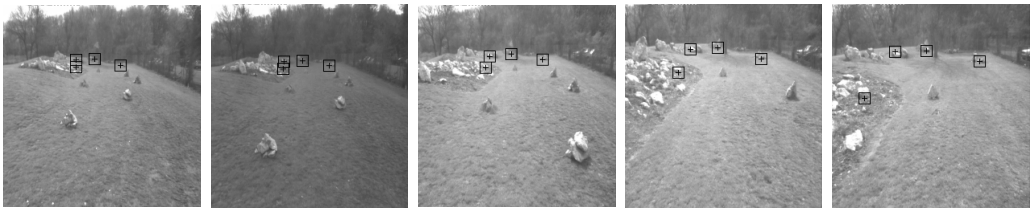

Figure 7: *An illustration of the pixel tracking algorithm: four goals located at approximately 50 meters are selected on the leftmost image, and are tracked as the robot moves (only one image out of twenty is displayed: the actual tracking has been performed on 100 images). Thanks to the ZNCC correlation criteria, illumination variation do not affect the track.*

3 Obstacle avoidance

The artificial potential field method originally proposed in [17] is one of the most widely used techniques for mobile robot local collision avoidance: it is simple, quite easy to implement, and very well adapted for real-time motion control. We present the developments we implemented to improve such a technique and to adapt it with the robot Lama, using cell images as the environment description.

3.1 Potential-based motion

The principle of artificial potential field motion generation is extremely simple: the robot motion is derived from the application of *(i)* an attractive force generated by the goal and of *(ii)* repulsive forces generated by the obstacles. The artificial potential function defined at the robot position X is of the form:

$$U(X) = U_g(X) + U_r(X)$$

where $U_g(X)$ is the attractive potential produced by the goal at X, and $U_r(X)$ the repulsive potential induced by the obstacles at X. The resultant force F is then:

$$F(X) = F_g(X) + F_r(X)$$

where $F(X) = -\vec{\nabla}\ U(X)$. F_g is the attractive force which guides the robot to the goal and F_r is the repulsion induced by the obstacles. Once

computed, the force $F(X)$ is then simply transformed into a robot motion command.

The difficulty with such methods relies in the definition of the various potentials: simple physics-based functions for instance (such as in a gravity field) do not work quite well for robots that have kinematic constraints (non-holonomy). The following sections present some improvements to classical potential field techniques adapted to the robot and to the environment description provided by the cell images.

3.2 Attractive potential

The attractive potential such as proposed in [17] induces an attractive force whose intensity is a linear function of the distance between the robot and the goal. With such a potential, the robot behavior with respect to the obstacles strongly depends on its distance to the goal: for instance, the robot tends to dangerously approach the obstacles when it is far from the goal. To cope with this, we chose to use the attractive potential function we originally proposed in [15]. This function has a quadratic behavior at the goal neighborhood and an asymptotic linear behavior away from it:

$$U_g = K_g \sqrt{d_g^2 + R^2}$$

where d_g denotes the distance to the goal, K_g is a positive gain constant, and R is a constant. The force deduced from this potential is thus:

$$F_g = -K_g \frac{d}{\sqrt{{d_g}^2 + R^2}} \frac{\partial d_g}{\partial X}.$$

where

$$R = d_m \sqrt{\frac{1}{F_g^{*2}} - 1},$$

is a function of a user-defined limit distance d_m and a maximum attractive force F_g^*. Thus, F_g is asymptotically constant for distances greater than d_m, and tends to zero as d_g tends to zero (figure 8). Figure 9 presents the result of the application of this force[3].

[3]Note that all the results of robot motions presented in this section correspond to long

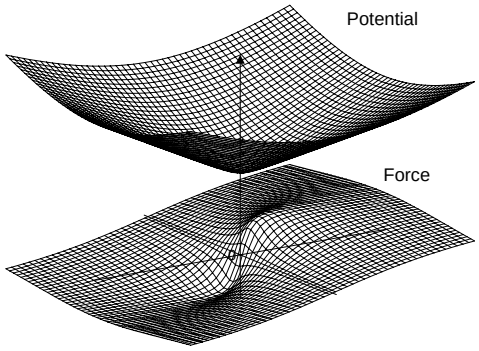

Figure 8: *Shape of the chosen attractive potential and corresponding force*

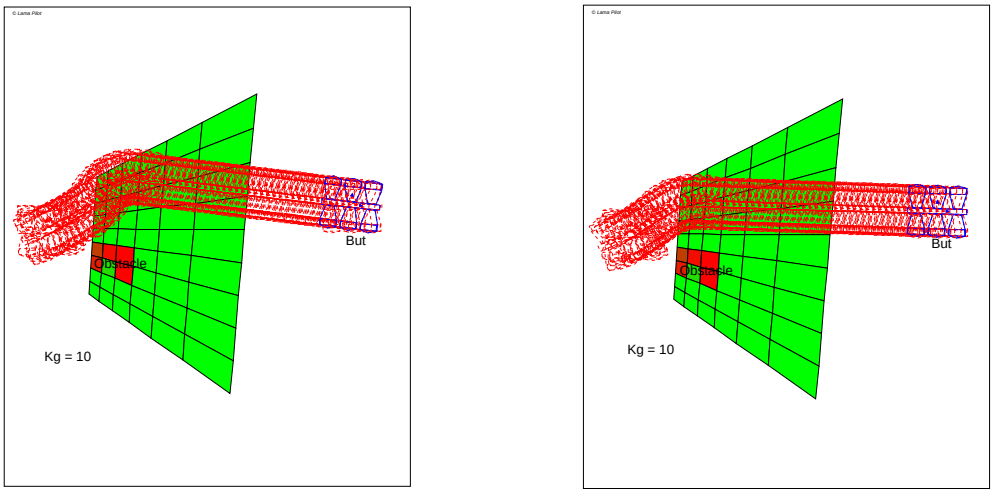

Figure 9: *Illustration of the influence of the attractive potential: on the left, with the attractive force presented in [17], the robot gets very close to the obstacle and is therefore strongly pushed away; whereas on the right, with the proposed attractive potential, it begins to avoid it earlier (a same repulsive potential is used for both situations).*

traverses of one local map: this is for illustrative purpose, in the real experiments, a local map has a short life duration, and the robot travels less than half a meter within the same map.

3.3 Repulsive potential

Most of the proposed repulsive potential functions in the literature only depend on the distance to the obstacles. A major drawback of such potentials is that obstacle segments have an influence on the robot even if the robot is moving in a direction parallel to them. This can lead to irregular motions, especially in our case where the environment description is polygonal.

We use the *rotational potential* approach, which we initially introduced in [16]. In this approach (as in [8]), some parameters describing the geometrical situation of the robot with respect to the obstacles are taken into account to define the repulsive function. The resultant repulsive potential we consider is a linear combination of two repulsive potentials $p_1(X)$ and $p_2(X)$:

$$U^{\alpha}(X) = \begin{cases} \alpha^2 p_2(X) + p_1(X) & \text{if } d < d_0, \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

where d_0 is a influence limit distance over which the obstacle has no influence on the robot motion, and $\alpha = \sin(\Delta\theta)$, $\Delta\theta$ is the relative angle between the robot direction and the nearest cell segment (figure 10). The functions p_1 and p_2 are defined as follows:

$$\begin{aligned} p_1(X) &= \tfrac{1}{2} K_1 \left(\tfrac{1}{d(X)} - \tfrac{1}{d_0} \right)^2 \\ p_2(X) &= \tfrac{1}{2} K_2 \left(d(X) - d_0 \right)^2 \end{aligned}$$

$p_1(X)$ has a quadratic behavior and $p_2(X)$ a linear one. The aim of p_1, which dominates in the neighborhood of the obstacle, is to guarantee non-collision ($p_1(X) \rightarrow \infty$ when $d(X) \rightarrow 0$). p_2 having a linear behavior, it dominates in the region near the influence limit distance boundary and acts to smoothly pre-avoid the obstacle considering its relative orientation expressed by the parameter α (figure 10).

Figure 11 illustrates the advantage of the introduction of the parameter α in the definition of the repulsive potential.

3.4 Adaptation to the cell images

A cell image gives the probability for each cell $[i, j]$ to correspond to an obstacle, but for security reasons, we divide the cells into two categories:

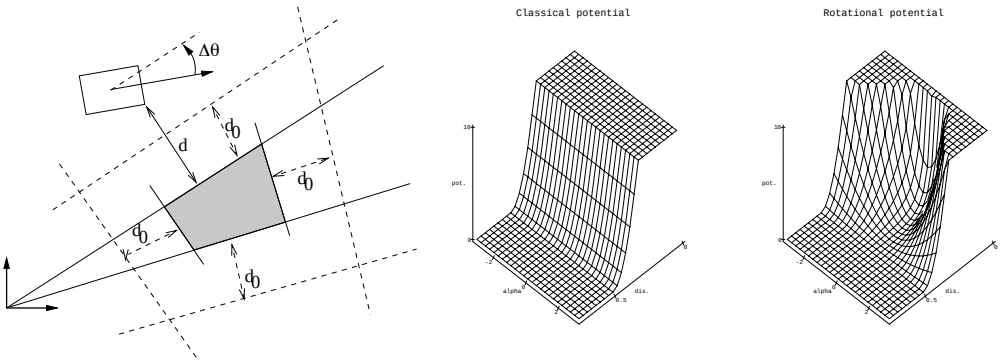

Figure 10: *The rotational potential. From left to right: parameters involved in the obstacle repulsive force computation, "classical" repulsive potential, and resultant potential* $U^{\alpha} = \alpha^2 p_2 + p_1$.

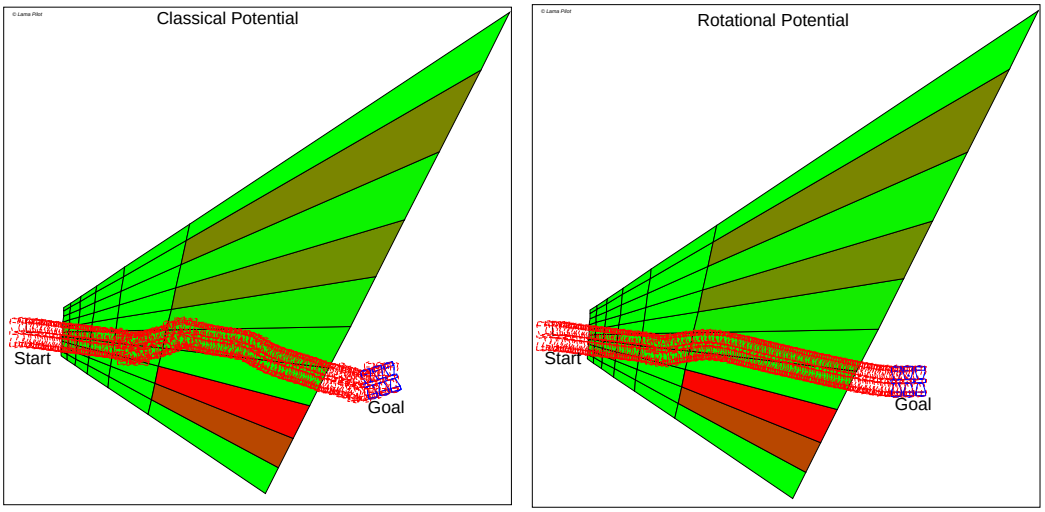

Figure 11: *"Go to goal" with classical repulsive potential* $U = p_1$ *(left), and with the rotational potential* $U^{\alpha} = \alpha^2 p_1 + p_2$ *(right). Only the two rightmost gray cells are considered as obstacles.*

- **obstacle cells:** are the non classified cells (no data) or the cells with $P_{Obst} \geq \mathcal{S}^o$ where $\mathcal{S}^o$ is a fixed threshold. The repulsive potential function associated to such cells is the rotational potential defined in equation 1;

- **traversable cells:** are the classified cells with $P_{Obst} < \mathcal{S}^o$. For a traversable cell $[i, j]$, the repulsive potential function is then given by:

$$U_r[i,j](X) = \begin{cases} \alpha^2[i,j] P_{Obst}[i,j]\; p_t(X) & \text{if } d < d_0 \\ 0 & \text{otherwise.} \end{cases}$$

where $p_t(X) = K_t \frac{1}{2} (d(X) - d_0)^2$. The force issued from this potential is linear and has a limit intensity as d tends to zero. This intensity is also modulated by the parameter P_{Obst}, the gain of this potential, which represents the cell traversability danger.

As a consequence, the robot may avoid the cells with $P_{Obst} < \mathcal{S}^o$, or traverse them if no better solution exists. The choice of K_t must insure that the traversability of these cells is still possible when the robot is subject to forces coming from obstacle cells.

The resultant potential U, generated by the cell image and the goal at position X is the following:

$$U(X) = \sum_{i,j} (U_r[i,j](X)) + U_g(X)$$

Note that $U_r[i,j] = 0$ if there is an intersection between the robot and the cell $[i,j]$: indeed, the robot must not be influenced by a cell within which it lies. Figure 12 shows the resultant potential for the a typical cell image.

3.5 Dynamic adaptation of the influence limit distance

The limit distance d_0 that defines the influence range of a cell is an extremely important parameter: using a rather large value of this distance would avoid getting the robot too close to the obstacles, but unfortunately, it would constrain too much the robot motion if there are many obstacles around. It is difficult to tune this parameter to ensure safe motions on one hand, and to guaranty that the robot finds his way trough the obstacles and does not get trapped in local minima on the other hand.

To solve this problem, we defined a *dynamic* influence limit distance:

$$D_0(\phi) = d_m f(\phi) = d_m \left(\frac{1 + e^{(\cos(\phi))}}{1 + e} \right)$$

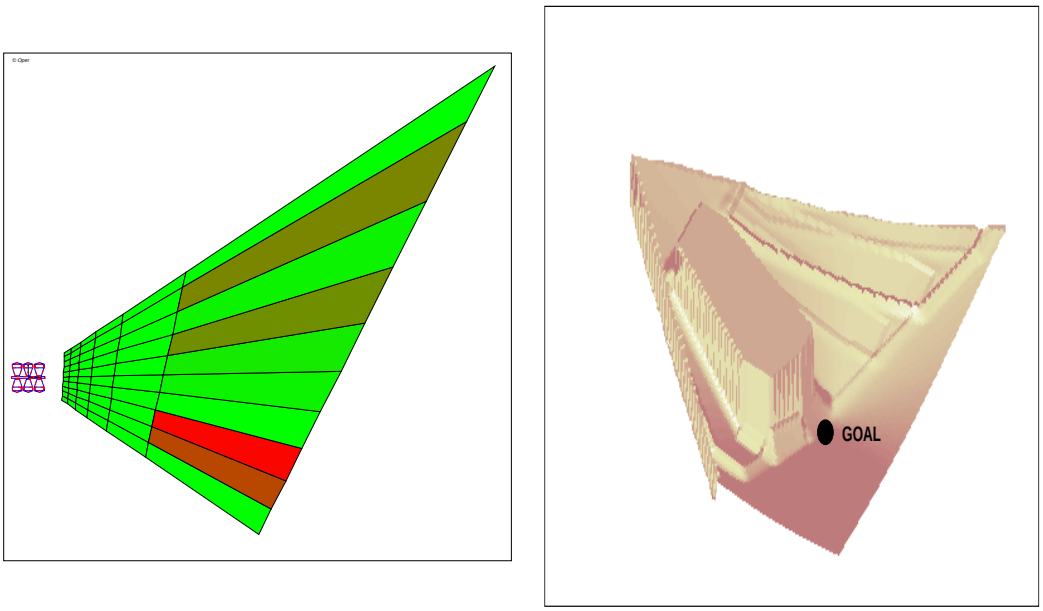

Figure 12: *A cell image and the corresponding potential field computed for all the possible positions, with a constant orientation angle.*

where d_m is the security distance corresponding to d_0 in the static case, and ϕ is the angle between the robot motion direction and the heading angle of the robot closest point to an obstacle (figure 13). With such a definition of the weighting function $f(\phi)$, the obstacle influence on the robot depends on its relative position to the robot: the limit distance $d_0(\phi)$ is maximal for $\phi = 0$ (the obstacle is in front of the robot), and is minimal for $\phi = \pi$ (the obstacle is behind the robot).

Figure 14 illustrates the advantage of the adaptation of the limit distance, in the case where the robot moves between two obstacles. Note also that using a static limit distance, the robot can hardly reach its goal because of an important repulsive force generated by an obstacle close to the goal. Using $d_0(\phi)$, the robot motion is smoother between the obstacles, and the influence of an obstacle close to the goal is reduced considerably.

3.6 Dealing with a sequence of images

From the point of view of obstacle avoidance, the difficulty of dealing with a sequence of images arises from the fact that the processing time t_{acq} needed

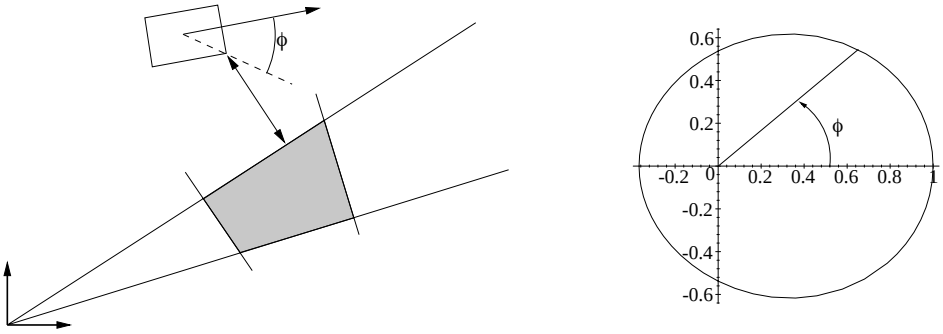

Figure 13: *The angle ϕ used to define the dynamic limit distance, and the weighting function $f(\phi)$*

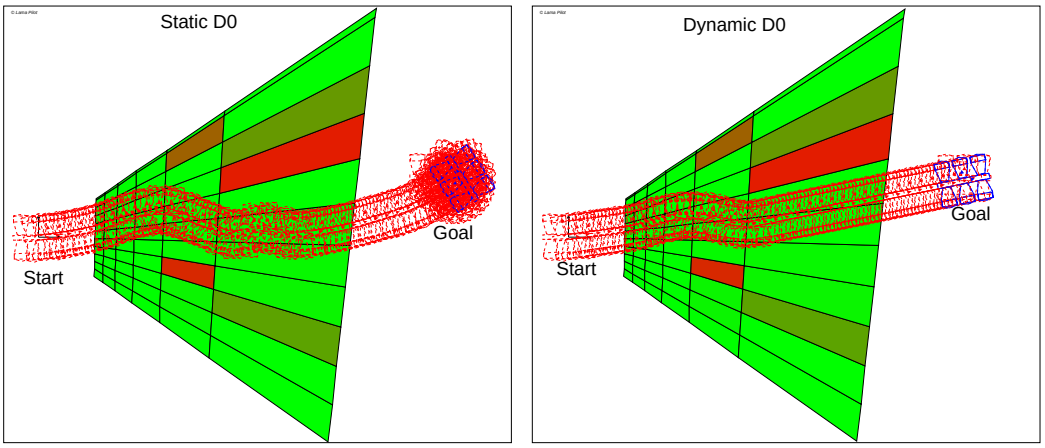

Figure 14: *"Go to goal" with a static parameter d_0 (left) and with the dynamic parameter $d_0(\phi)$*

to obtain a cell image is not negligible with respect to the robot speed.

Let us assume that at instant t of image acquisition, the robot is in a cell image V_{t-1} (figure 15). From t until the availability of the next cell image V_t, the robot is bounded in V_{t-1}, since it is the only area on which it has informations. At the time t of the new image acquisition, only the boundaries of the future cell image V_t are available. Therefore, the robot motion executed during the processing time t_{acq} must constrain the robot to reach the area $V_{t-1} \cap V_t$.

This constraint has an important implication: the time t_{acq} and the robot

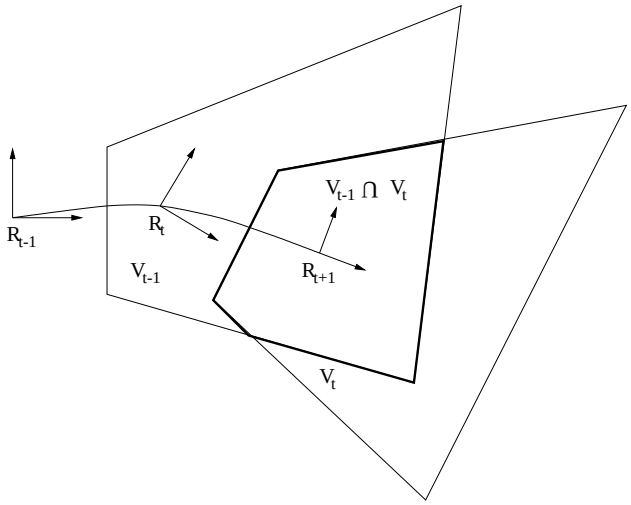

Figure 15: *Linking an image sequence*

speed being given, one must make sure that at any time there is enough free space in front of the robot such that the intersection between the current cell image and the next one is still accessible. This constrains the robot to avoid sharp turns and get close to obstacles, which is made by tuning propoerly the various gain parameters in the repulsive potential definition.

3.7 Conclusions

The main difficulty with potential-based motion generation is to trigger the various parameters of the force definition. These parameters must be set so that the robot executes avoiding maneuvers safely, *i.e.* without approaching the obstacles too much, while avoiding to get trapped in local minima. If most of the "classical" potential functions can easily be tuned to satisfy these requirements in simple environments, it becomes much more difficult when the environments gets more cluttered. Thanks to the various improvements we have proposed (definition of the attractive force, introduction of the geometric robot configuration with respect to an obstacle to define the potential and the influence limit distance), we could easily tune the various parameters so that the robot executes safe and smooth maneuvers. These parameters are the following:

- For the attractive potential, the gain K_g, the limit distance d_m and the

maximum attractive force F_g^*;

- For the repulsive potential, the limit distance d_0 and the three gains K_1, K_2 and K_t.

Figure 16 shows various trajectory examples, all executed with the same parameter set. Of course, we still can not guaranty that no local minima occur: it is one of the attribution of the decision level (next section) to detect and solve them.

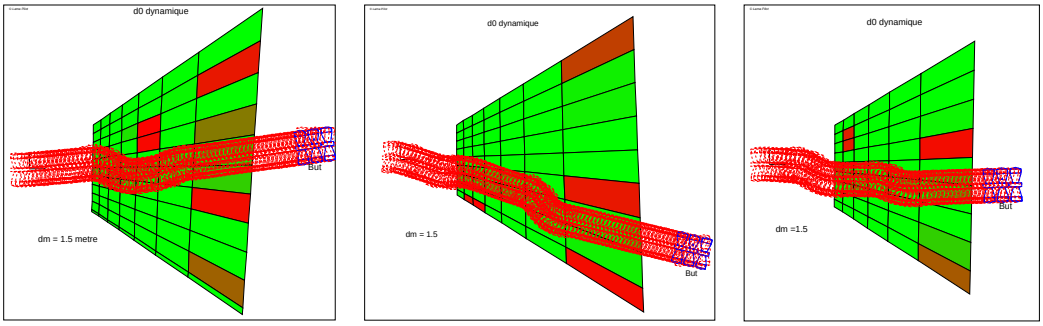

Figure 16: *Some trajectories resulting from the application of the various improvements.*

Finally, to test our algorithms on realistic image sequences, we have completed a simulation system on Unix platforms (figure 17): a ray-tracing algorithm run on a digital elevation map produces 3D points images, at the resolution of the cameras that equips Lama. The classification procedure being rather qualitative, its results on simulated 3D images are similar to those on real images. This simulation has been helpful to tune the various parameters of the obstacle avoidance module.

4 Software architecture

The presented navigation capacities have been integrated onboard the robot. The software organization, or architecture, that we have adopted is not specific to this application. It is an implementation of a generic architecture elaborated to design full autonomous robots [1].

Only a subset of this architecture is really necessary to demonstrate the reactive navigation capacity – which in our viewpoint is *one* among other needed capacity of an autonomous robot. This approach can take advantage

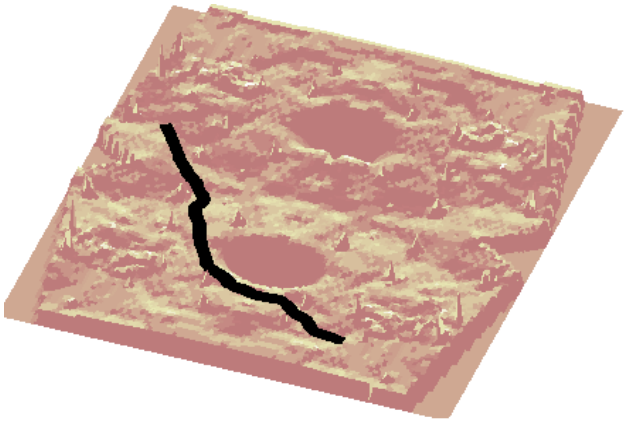

Figure 17: A complete simulation result

of a background of concepts and tools (PRS [13, 12], GenoM [6, 7]) for the architecture design, and above all it will allow to extend the robot capacities.

After a brief overview of the generic architecture, the different components of the Lama architecture instantiation are presented.

4.1 The generic architecture

The generic architecture has three hierarchical levels [1], having different temporal constraints and manipulating different data representations. From bottom up, the levels are:

- **A functional level.** This level includes all the basic built-in robot action and perception capacities. These processing functions and control loops (image processing, motion control, *etc.*) are encapsulated into controllable communicating modules (section 4.2). In order to accomplish a task, the modules are controlled by the next level.
- **An execution control level,** or Executive. Just above the functional level, it controls and coordinates dynamically the execution of the functions, distributed in the modules, according to the task requirements specified by the higher level (section 4.3.1).
- **A decision level.** This higher level includes the capacities of producing the task plan and supervising its execution, while being at the same time reactive to events from the underlying levels.

However, as pointed out in [1], not all instances of this generic architecture use all the levels and components presented. Indeed, in the application considered in this paper, we did not deploy any plan supervisor, nor mission planner. The resulted software architecture for the presented experiment is shown on figure 18 on page 24.

Nevertheless, the overall application has been built following this generic architecture and, if necessary, can easily be extended toward a more complete instance, including a high level planner, as well as a more specific execution control component.

4.2 Functional Level

The Functional Level embeds a set of elementary robot actions implementing processing functions and task-oriented servo-loops. In particular, it integrates the platform-dependent basic functionalities to control the sensors and effectors (cameras, odometry, drive-wheels, *etc.*); but also the reactive navigation capacities presented in this paper: images analysis (section 2) and obstacle avoidance (section 3).

The functions are embedded into *modules* [7] where they are run using internal synchronous or asynchronous task-processes. The state of the robot, its execution context and its perception of the environment are entirely apprehended through the modules. A module has the responsibility for the physical resources (sensors or effectors for the low level modules) or the logical resources (shared data: a map, a position, *etc*) that it controls: it must respect the time constraints imposed by the controlled resources and must guarantee the integrity of the resources, or at least detect and signal any problem that may occur. For instance the module that runs the obstacle avoidance procedure must also be able to detect "local minima".

However a module has not a global view of the robot task and can not decide to start or to stop a treatment. This operation is controlled, directly or indirectly, by the decision level. Thus, each module offers a set of predefined services that are parameterized and/or activated asynchronously through a non-blocking client/server protocol: a relevant *request*, that may include input parameters, applies to every service of each module. For instance, the acquisition of a pair of stereo images is a service offered by a module `CAMERAS`. A module may also send a request to an other module (a module does not know *a priori* its clients).

Once the service is over, the module returns a *reply* to its client that

includes an *execution report* which qualifies how the service has ended (`local-minima-error`, ...). All the possible execution reports for a given service are defined *a priori* and allow the client to plan adequate recovery procedures. These client/server relationships are established dynamically (upon reception of a request) and they implement the *control flow* of the functional level.

The data produced by the modules (a pair of stereo images, a map of obstacle cells, a robot position estimation, *etc*) are exported into shared memory structures named *posters* (the circles in figure 18 on the next page). A poster is readable by any element of the architecture, but its contents can be modified only by the module that owns it. Each poster is identified by its name. This identification permits to easily *redirect the data flow*: the request associated to a service that needs an input data (produced by another service) has a parameter to specify the poster identifier.

The following presentation of the Lama functional level will give the occasion to illustrate this module organization.

4.2.1 Lama Functional Level

The functional level for Lama is currently composed of ten modules: the navigation capacities are integrated in four distinct modules; the basic capacities to control sensors and actuators are integrated in four other modules; and finally, two modules are used for teleoperation purposes.

These modules are represented on figure 18. They can be virtually classified into three groups:

1. The hardware devices control modules. At the bottom of the functional level we find the four basic modules that control the hardware devices of the robot:

- the module `COM` controls the radio communication with the operator station. It offers two services: one to send messages and one to receive messages (this second one is similar to a monitoring: the final reply is sent back on the reception of a message).

- the module `CAMERAS` controls the two cameras. It offers several services to acquire mono (with any camera) or stereo images. The resulted images are stored in a poster.

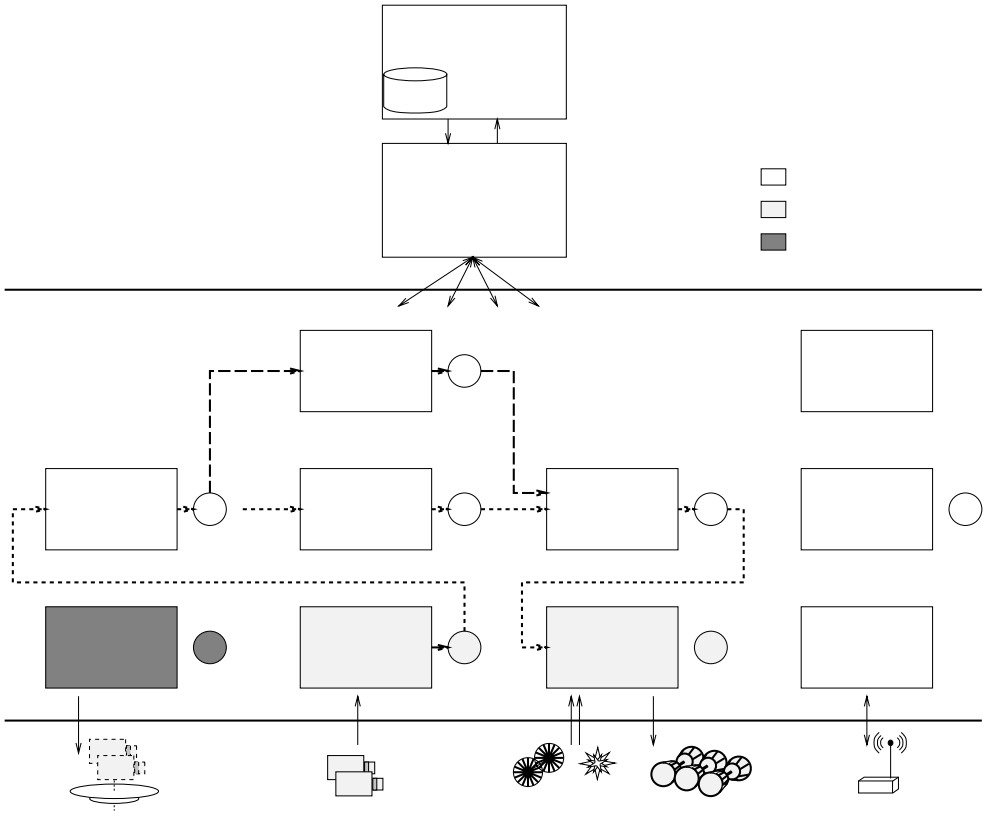

Figure 18: *The software architecture for reactive navigation. The dotted arrows at the functional level represent a possible instance of the data flow between the modules, established by the decision level, during the execution of a reactive navigation loop. For each module, the main services are indicated.*

- the module `PLATFORM` controls the pan-tilt platform that supports the cameras. It maintains on a poster the current configuration of the platform and offers services to execute rotations to a given orientation, or along a given trajectory (dynamic speed profiles) for tracking purpose.

- the module `LOCOMOTION` controls the proprioceptive motion sensors (odometry, compass, inclinometers) and the wheel motors. It has two main purposes: (i) using the proprioceptive sensors it maintains on a poster the current configuration of the robot at a 25 milliseconds rate; (ii) it looks after the servo-control of the robot on a given position, or along a given trajectory. The position or the speed references are

read periodically on a poster which identifier is specified within the request parameters. In this application, this poster could belong to two different modules: `AVOID` or `PILOT` has shown bellow.

These modules depend only on the physical robot and fit with almost all possible applications.

2. The reactive navigation modules. The navigation functionalities presented above are organized in four modules:

- the module `CORREL` embeds the stereo correlation functions. It produces a 3D image from a pair of stereo images read from the `CAMERAS` poster in this application.
- the module `CLASSIF` integrates the terrain classification functions. It produces a cell map from a 3D image. This image is currently produced by the module `CORREL`, but it could as well use a 3D image provided by a 3D laser module.
- the module `AVOID` integrates the local avoidance functions. From a map of obstacles and the current positions of the robot and the goal in this map, it updates dynamically in a poster the reference position that the robot (*i.e.,* the module `LOCOMOTION`) has to track: `AVOID` is a client of the module `LOCOMOTION`.
- the module `GOALFIND` integrates the goal identification and location functions. It produces a goal position. Currently this module is nearly an empty shell: the initial estimation of the goal position is directly copied in its goal data poster.

3. The teleoperation modules. Beside the autonomous reactive navigation mode the robot can be controlled directly by an operator in a teleoperated mode which uses two more modules:

- the module `TV` down-loads the last images acquired by `CAMERAS`. It allows the operator to visualize the environment "seen" by the robot and to remotely control the robot.

- the module `PILOT` allows to execute basic trajectories (translations and rotations). The reference positions along the required trajectory are sent to the module `LOCOMOTION`.

The arrows on figure 18 on page 24 represent the *data flow* (and not the control flow) during a complete the execution of the nominal loop presented on the introduction.

The sequencing is ensured by the decision level.

4.2.2 Integration Methodology

All the modules present standard interfaces, a standard structure and a standard behavior. In fact they are built according to a common generic template [7]. This model allows to integrate both synchronous and asynchronous treatments. Moreover, modules are automatically generated using a generator of module named GenoM [7].

Only the specific parts of a given module have to be described. This description is based on a simple declarative language, associated with GenoM, which mainly consists in declaring the services offered by the module, their parameters, their execution reports, their temporal characteristics, their posters and some others parameters.

This description is analyzed by GenoM which instantiates the common template and compiles the resulting code files. The new module can then be directly integrated in the architecture (on Unix or VxWorks operating system). The algorithms linked to each service are incrementally included and finalized.

GenoM produces also the client libraries to communicate with the module and an interactive test program to evaluate the module.

4.3 Execution Control Level and Decision Level

Although the execution control level and decision level have distinct functionalities, and should be running while satisfying different temporal constraints, we present them together in this paper as they were programmed using the same tool: PRS. Their presence in the system allow the robot programmer to build a more complex system by adding or combining other navigation methods (provided the proper functional modules are available).

4.3.1 The Execution Control Level

The Execution Control Level (or Executive) is a purely reactive system, with no planning capability. It receives from the decision level the sequences of actions to be executed. It selects, parameterizes and synchronizes dynamically the adequate services of the functional level, *i.e.* the requests to be sent to the modules.

The selection and the instantiation of the request parameters depend on the task and on the current state of the system. This state is maintained by the Executive, according to the ongoing services and to the output of previous processing (*i.e.* according to requests sent and to replies returned). Replies may trigger requests delayed by the Executive. A report must be returned systematically to the decision level (*e.g.* `robot-localized`, `target-not-found`, `trajectory-computed`), to enable plan supervision and selection of next actions. The instantiation of the request parameters consists generally in redirecting data previously produced by other services through the posters (*e.g.* a 3D map computed by the module `CORREL` which has to be classified).

The executive interacts with the modules using the client/server protocol and it integrates the communication rules mentioned previously (requests, replies and posters).

For instance, let's consider the main service of the module `AVOID`: the service `AVOID-TRACK` that produces the reference positions for the robot to reach its goal while avoiding obstacle cells.

The execution control consists in sending the corresponding request, waiting for the final reply, and, as far as possible, handling the failures (that do not break down the task plan).

In our application, two failures are considered by this procedure: a local minimum can be detected by the module `AVOID`, and servo-controller may auto-shutdown on overvoltage (detected via the module `LOCOMOTION` which informs its client: the module `AVOID`).

For the first case, the executive simply tries to restart the service, expecting that a path will be found in a new obstacle cells map. After a given number of unsuccessful retries the failure is sent back up to the supervisor.

The second case may occur in two situations: a temporary slipping of a wheel, or a fault of the servos which needs a specific recovery procedure. To detect these situations, the executive checks for the state of the servos during a given time. It restarts the service if they work correctly or informs the supervisor after the timeout.

4.3.2 The Decision Level

The Decision level is in charge of all the processes that require anticipation and some global knowledge of the task and the execution context: it embeds the deliberative capacities of planning and decision-making, and should remain reactive to incoming events which inform how the plan execution is going on.

The activity of the supervisor consists in monitoring the execution of the plan by performing situation detection and assessment and by taking appropriate decisions in real time.

As mentioned earlier, we have not currently integrated any high level planner in this application (see [1] for examples of applications using high level planners). Besides the plan, the supervisor makes use of a set of *situation-driven procedures* embedded in a database.

Finally, the supervisor is in charge of receiving the task or the goal from the operator. They are either sent to the planner for producing the sequence of actions achieving them (if any), or directly executed if they correspond to procedures that are already present in the system (case of this application).

Figure 19 on the next page presents the main PRS procedure which is in charge of the proper sequencing of the various data acquisitions/processings and goal tracking.

The procedure is started on the `(REACTIVE-LOOP)` invocation (sign `!` in PRS). It then starts in parallel the `AVOID-TRACK-LOOP` procedure presented above, and the sequence acquisition-correlation to produce the 3D image.

The stereo-vision loop (dashed lines loop on the left of the figure) proceeds while the obstacle cell map is constructed and passed to the module `AVOID` using the service `AVOID-GETOBSTACLECELLS` (dotted lines loop on the right of the figure).

The `SYNCHRO` node synchronizes the two loops, *i.e.* the start of a new service `CLASSIF-BUILDMAP` with the end of both services `CORREL-STEREO` and the previous `CLASSIF-BUILDMAP`. On the contrary, no synchronization is needed with the end of the service `AVOID-GETOBSTACLECELLS`: a new obstacle cell map must be considered as soon as possible, even if the reading procedure of the previous map has not yet ended.

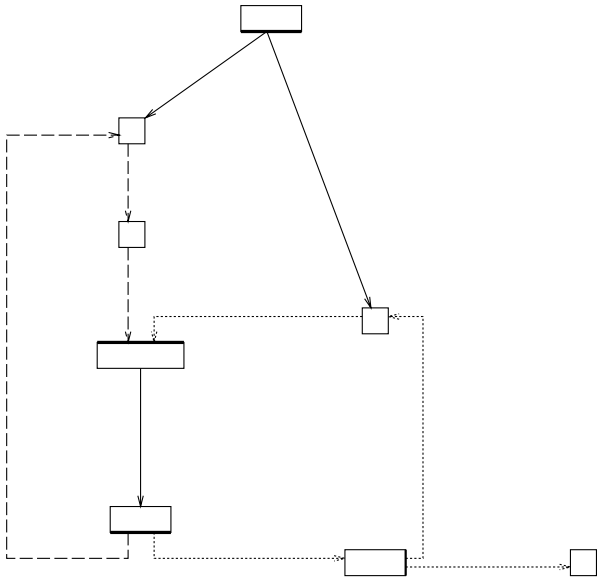

Figure 19: *An supervisor procedure: the reactive navigation loop. It is the main loop of the presented application.*

5 Discussion

5.1 Experimental results

The perception and motion generation algorithms presented in this paper have been integrated on the robot Lama (figure 1). Lama is a robot from the Marsokhod[4] family [14], owned by Alcatel and currently lent to LAAS. The chassis is composed of three pairs of independently driven wheels, mounted on axes that can roll relatively to one another. Lama is $1.20m$ wide, and its length varies from $1.60m$ to $2.20m$, depending on the axes configuration ($1.90m$ on its "nominal" configuration), and weighs approximately $160kg$. A $1.80m$ mast, that supports a pan and tilt platform, is mounted on the middle axis. Each motor is driven by a servo-control card, and the maximal speed of Lama is $0.20m.s^{-1}$.

Lama is equipped with the following sensors:

- Each wheel is equipped with an optical encoder;

[4]Marsokhod is a Russian term for Mars rover.

- Five potentiometers provide the robot axis configuration;
- A 2 axis inclinometer and a magnetic fluxgate compass provide the robot orientation
- Finally, the stereo images are provided by two synchronized Micam VHR 2000 video cameras (752×582 CCD matrix, with $4.8mm$ focal length lenses: their field of view is $68°$ horizontally and $53°$ vertically.

All the computing equipment is in a VME rack mounted on the rear axis of the robot. The rack contains a DataCube frame grabber, and three CPU's (a $200Mhz$ 604 PowerPc, and two 68040) operated by the real-time OS VxWorks. All the algorithms presented in this paper, including the decision control level, are run on board (the algorithm distribution on the CPU's is shown on figure 18: the current hardware configuration is actually too powerful).

The terrain on which we have tested the approach with Lama is an essentially flat ground, with gentle slopes (less than $5°$) and some scattered obstacle rocks and trees. We performed dozens of experiments there, changing various parameters and environmental conditions. The best results have been achieved using *all* the improvements presented in section 3. Figures 20 and 21 show two typical runs of the robot, that travels around $30m$ toward its goal.

Up to now, the goal tracker has not yet been integrated: we emulated it using dead-reckoning, the goal being specified as global Cartesian coordinates at the initialization. Note that much more cycles are usually executed during the robot motions: in the experiments shown here, the frequency has been reduced because of data downloading time. The actual rate is approximately one loop every three seconds, the stereo-vision algorithm taking $2s$ to produce a 170×250 3D image, whereas one loop took around $15s$ in the experiments shown here.

5.2 Ongoing work

Several problems are still to be handled within the context of this approach. The following ones are currently being studied:

- Integration of the visual goal tracker: if the implementation of goal tracker is not a big issue, its integration is a bit more tedious problem.

Indeed, it requires the same resource as the obstacle avoidance module (the cameras and the pan-tilt platform), and often in a contradictory way.

- Parameters determination: the various parameters have to be studied more deeply; especially, various camera resolution and field of should be evaluated, since they have an impact on the loop frequency.

- Finally, the autonomous selection of (intermediate) visual goals by a higher level navigation planner must be considered, so that this reactive navigation capacity could be integrated in our general long range navigation approach.

Most of the effective achievements in autonomous outdoor navigation deal with a context similar to this paper: the environment is essentially flat, and motion commands are issued on the basis of *local* terrain representations. Some approaches consist in a rather systematic strategy [24, 22], while others evaluate a discrete set of steering arcs [20, 25]. We think the approach presented here is an interesting contribution to the problem: to our knowledge, it is the first application of a potential field technique in outdoor environments, and it proved its capacity to generate smooth and reliable trajectories.

References

[1] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An architecture for autonomy. *International Journal of Robotics Research*, 17(4):315–337, April 1998.

[2] S. Betge-Brezetz, R. Chatila, and M.Devy. Natural scene understanding for mobile robot navigation. In *IEEE International Conference on Robotics and Automation, San Diego, California*, 1994.

[3] R. Chatila and S. Lacroix. Adaptive navigation for autonomous mobile robot. In *International Symposium on Robotics Research, Munich (Germany)*, October 1995.

[4] R. Chatila, S. Lacroix, T. Siméon, and M. Herrb. Planetary exploration by a mobile robot : Mission teleprogramming and autonomous navigation. *Autonomous Robots Journal*, 2(4):333–344, 1995.

[5] O. Faugeras, T. Vieville, E. Theron, J. Vuillemin, B. Hotz, Z. Zhang, , L. Moll, P. Bertin, H. Mathieu, P. Fua, G. Berry, and C. Proy. Real-time correlation-based stereo : algorithm, implementations and application. Technical Report RR 2013, INRIA, August 1993.

[6] S. Fleury, M. Herrb, and R. Chatila. Design of a modular architecture for autonomous robot. In *IEEE International Conference on Robotics and Automation, San Diego, California*, May 1994.

[7] S. Fleury, M. Herrb, and R. Chatila. Genom: A tool for the specification and the implementation of operating modules in a distributed robot architecture. In *International Conference on Intelligent Robots and Systems (IROS'97), Grenoble, France*, 1997.

[8] D.N. Green, J.Z. Sasiadek, and G.S. Vukovich. Path traking, obstacle avoidance and position estimation by an autonomous, wheeled planetary rover. In *IEEE International Conference on Robotics and Automation, San Diego California, (USA)*, pages p. 1300–1305, 1994.

[9] A. Hait and T. Simeon. Motion planning on rough terrain for an articulated vehicle in presence of uncertainties. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, Osaka (Japan)*, pages 1126–1133, Nov. 1996.

[10] M. Hebert. Pixel-based range processing for autonomous driving. In *IEEE International Conference on Robotics and Automation, San Diego, California (USA)*, 1994.

[11] L. Henriksen and E. Krotkov. Natural terrain hazard detection with a laser rangefinder. In *IEEE International Conference on Robotics and Automation, Albuquerque, New Mexico (USA)*, pages 968–973, April 1997.

[12] F. F. Ingrand, R. Chatila, R. Alami, and F. Robert. Prs: A high level supervision and control language for autonomous mobile robots. In *IEEE ICRA'96, St Paul, (USA)*, 1996.

[13] F. F. Ingrand, M. P. Georgeff, and A. S. Rao. An Architecture for Real-Time Reasoning and System Control. *IEEE Expert, Knowledge-Based Diagnosis in Process Engineering*, 7(6):34–44, December 1992.

[14] A. Kemurdjian, V. Gromov, V. Mishkinyuk, V. Kucherenko, and P. Sologub. Small marsokhod configuration. In *IEEE International Conference on Robotics and Automation, Nice (France)*, pages 165–168, May 1992.

[15] M. Khatib. *Sensor-based motion control for mobile robots*. PhD thesis, LAAS-CNRS, Toulouse, France, December 1996. – Ref.: 96510.

[16] M. Khatib and R. Chatila. An extended potentiel field approach for mobile robot sensor-based motions. *Intelligent Autonomous Systems (IAS'4), Karlsruhe (Germany)*, pages 490–496, 1995.

[17] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90–98, 1986.

[18] E. Krotkov, M. Hebert, and R. Simmons. Stereo perception and dead reckoning for a prototype lunar rover. *Autonomous Robots*, 2(4):313–331, Dec. 1995.

[19] S. Lacroix, R. Chatila, S. Fleury, M. Herrb, and T. Siméon. Autonomous navigation in outdoor environment : Adaptive approach and experiments. In *Proceedings of the IEEE International Conference on Robotics and Automation, San Diego, California (USA)*, May 1994.

[20] D. Langer, J. K. Rosenblatt, and M. Hebert. An integrated system for autonomous off-road navigation. In *IEEE International Conference on Robotics and Automation, San Diego, Ca (USA)*, pages 414–419, 1994.

[21] L. Matthies. Toward stochastic modeling of obstacle detectability in passive stereo range imagery. In *IEEE Conference on Computer Vision and Pattern Recognition, Champaign, Illinois (USA)*, pages 765–768, 1992.

[22] L. Matthies, E. Gat, R. Harrisson, B. Wilcox, R. Volpe, and T. Litwin. Mars microrover navigation: Performance evaluation and enhancement. *Autonomous Robots Journal*, 2(4):291–311, 1995.

[23] L. Matthies, A. Kelly, and T. Litwin. Obstacle detection for unmanned ground vehicles: A progress report. In *International Symposium of Robotics Research, Munich (Germany)*, Oct. 1995.

[24] D. Miller, R. Desai, E. Gat, R. Ivlev, and J. Loch. Reactive navigation through rough terrain: Experimental results. In *10th National Conference on Artificial Intelligence (AAAI '88), San Jose, Ca (USA)*, pages 823–828, 1992.

[25] J. Rosenblatt and C. Thorpe. Combining multiple goals in a behavior-based architecture. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, Pittsburgh, Pa (USA)*, 1995.

[26] J. Shi and C. Tomasi. Good features to track. In *IEEE International Conference on Computer Vision and Pattern Recognition, Seattle (USA)*, pages 593–600, 1994.

[27] D. Wettergreen, H. Thomas, and M. Bualat. Initial results from vision-based control of the ames marsokhod rover. In *IEEE International Conference on Intelligent Robots and Systems, Grenoble (France)*, pages 1377–1382, Sep. 1997.

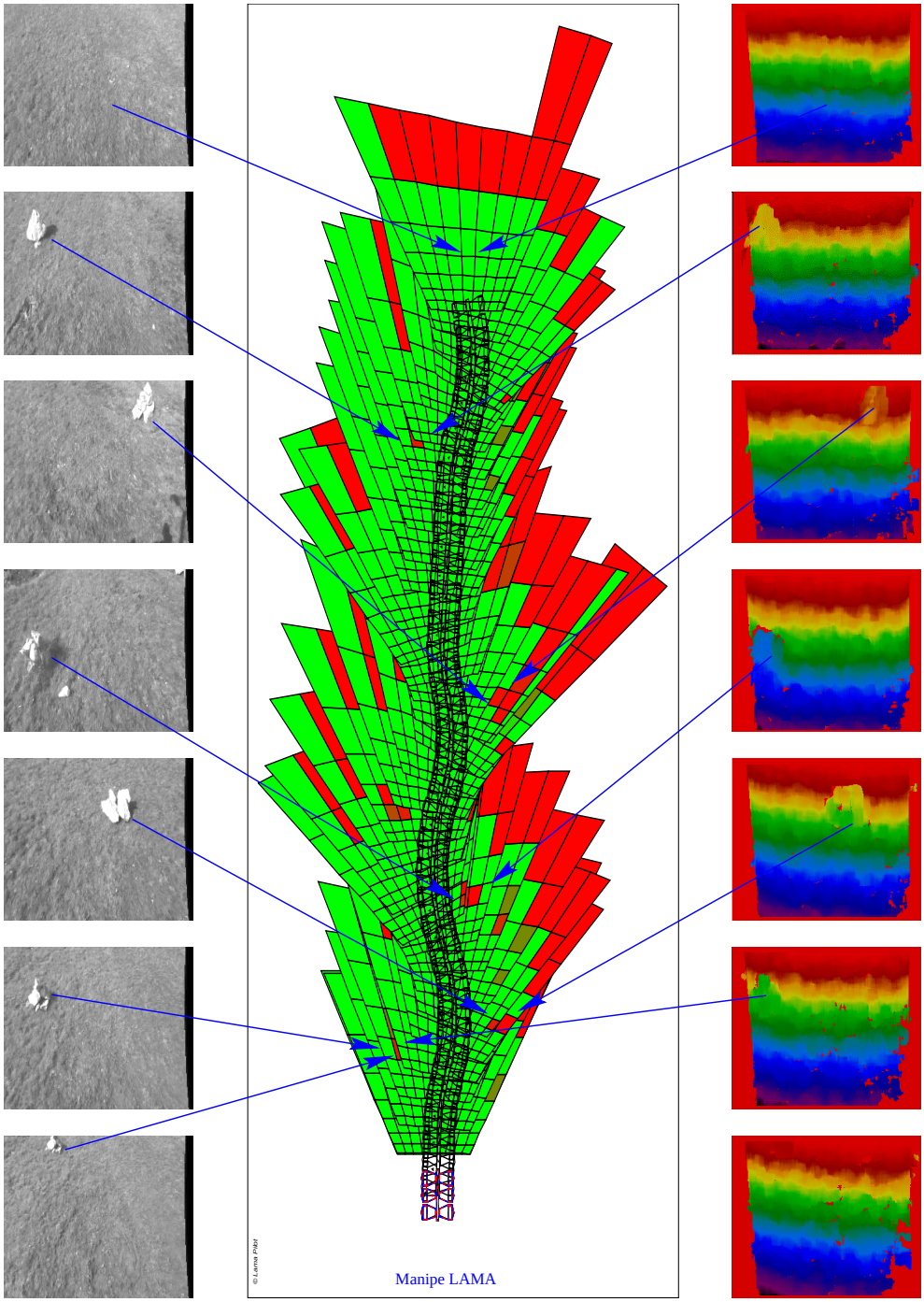

Figure 20: *A typical run with Lama*

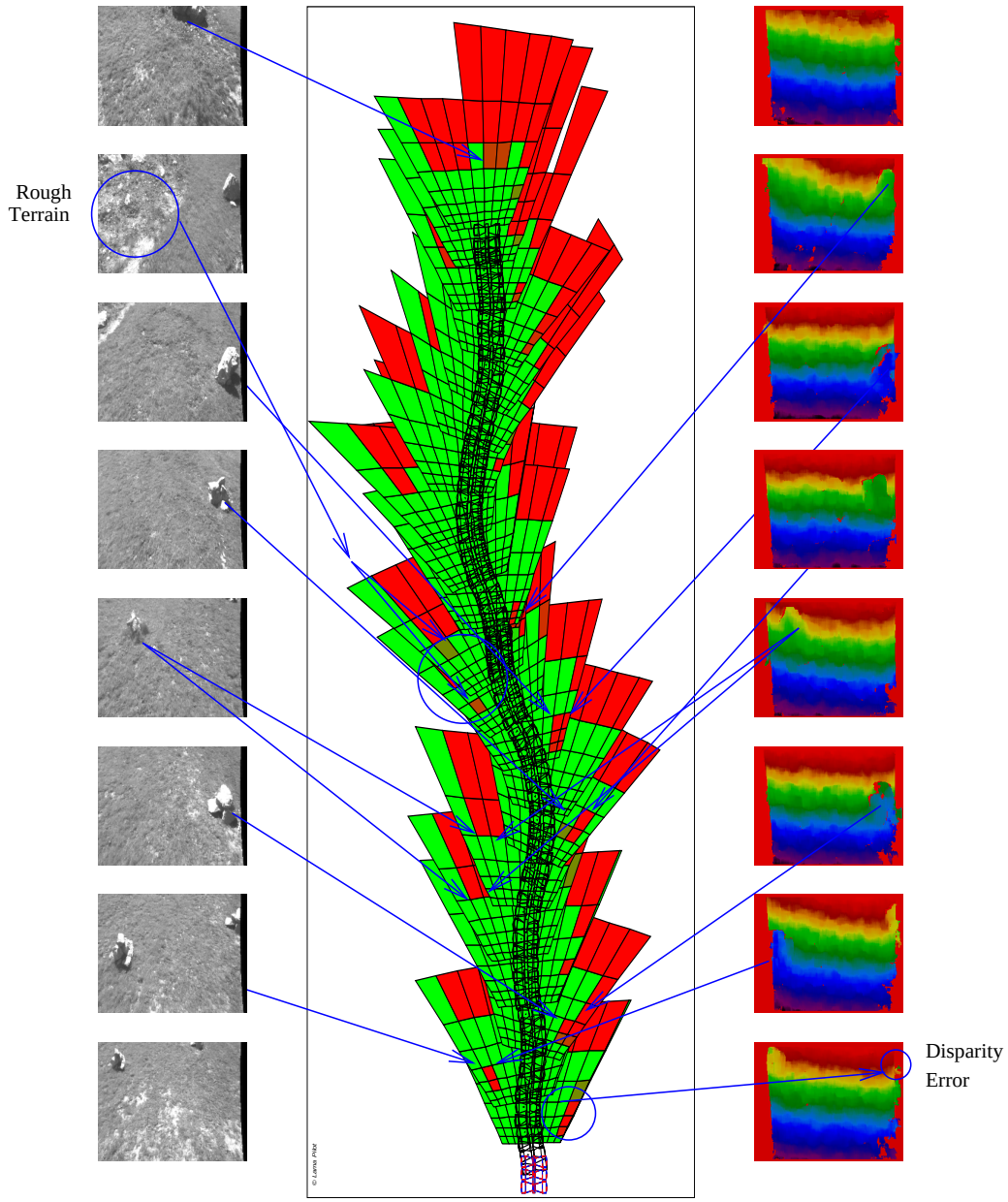

Figure 21: *A typical run with Lama*