Autonomous Rover Navigation on Unknown Terrains Demonstrations in the Space Museum "Cité de l'Espace" at Toulouse

Simon Lacroix, Anthony Mallet, David Bonnafous
Grard Bauzil, Sara Fleury, Matthieu Herrb, and Raja Chatila
LAAS/CNRS
7, av. du Colonel Roche
F-31077 Toulouse Cedex 4
France

Abstract

Autonomous long range navigation in partially known planetary-like terrain is on open challenge for robotics. Navigating several hundreds of meters without any human intervention requires the robot to be able to build various representations of its environment, to plan and execute trajectories according to the kind of terrain traversed, to localize itself as it moves, and to schedule, start, control and interrupt these various activities. In this paper, we briefly describe some functionalities that are currently being integrated on board the Marsokhod model robot Lama at LAAS/CNRS. We then focus on the necessity to integrate various instances of the perception and decision functionalities, and on the difficulties raised by this integration.

1 Introduction

To foster ambitious exploration missions, future planetary rovers will have to fulfill tasks described at a high abstraction level, such as `''reach the top of that hill''` or `''explore this area''`. This calls for the ability to navigate for several hundreds of meters, dealing with various and complex situations, without any operator intervention. Such an ability is still quite an open challenge: it requires the *integration* and *control* of a wide variety of autonomous processes, ranging from the lowest level servoings to the highest level decisions, considering time and resource constraints.

We are convinced that no simple autonomy concept can lead to the development of robots able to tackle such complex tasks: we believe in the efficiency of *deliberative* approaches [4], that are able to plan and control a variety of processes. Following such a paradigm and according to a general economy of means principle, we want the robot to autonomously *adapt* its decisions and behavior to the environment and to the task it has to achieve [5]. This requires the development of:

- Various methods to implement each of the perception, decision and action functionalities, adapted to given contexts;
- An architecture that allowsfor the integration of these methods, in which deliberative and reactive processes can coexist;
- Specific decision-making processes, that dynamically select the appropriate decision, perception and action processes among the ones the robot is endowed with.

In this paper, we present the current state of development of the robot Lama, an experimental platform within which our developments related to autonomous long range navigation are integrated and tested. We especially focus on the necessity to integrate various implementations of each of the main functionalities required by autonomous navigation (*i.e.* environment modeling, localization, path and trajectory generation). After a brief description of Lama and its equipment, the rest of the paper is split in two parts: the first part briefly presents the main functionalities required by long range navigation we currently consider (terrain modeling, path and trajectory planning, rover localization), while the second part insists on the problems raised by the *integration* of these functionalities.

2 The Robot Lama

Lama is a 6-wheels Marsokhod model chassis [10] that has been totally equipped at LAAS[1]. The chassis is composed of three pairs of independently driven wheels, mounted on axes that can roll relatively to one another, thus giving the robot high obstacle traversability capacities. Lama is $1.20m$ wide, its length varies from $1.60m$ to $2.20m$, depending on the axes configuration ($1.90m$ in its "nominal" configuration), and weighs approximately $160kg$. Each motor is driven by a servo-control board, and its maximal speed is $0.17m.s^{-1}$. Lama is equipped with the following sensors:

- Each wheel is equipped with a high resolution optical encoder, allowing fine speed control and odometry;
- Five potentiometers provide the chassis configuration;
- A 2 axes inclinometer provides robot attitude, a magnetic fluxgate compass and a optical fiber gyrometer provide robot orientation and rotational speed;
- A first stereo bench on top of a pan and tilt unit, is mounted on a $1.80m$ mast rigidly tied to the middle axis. This bench has a horizontal field of view of approximately $60°$, and is mainly dedicated to goal and landmarks tracking;
- A second stereo bench, also supported by a PTU, is mounted upon the front axis, at a $0.80m$ elevation. It has a horizontal field of view of approximately $90°$, and is mainly dedicated to terrain modeling in front of the robot;
- A differential carrier-phase GPS receiver[2] is used to qualify the localization algorithms.

All the computing equipment is in a VME rack mounted on the rear axis of the robot. The rack contains four CPU's (two PowerPc and two 68040) operated by the real-time OS VxWorks. The 68040 are in charge of the data acquisitions (except the camera images) and of the locomotion and PTU control, whereas all the environment modeling and planning functionalities are run on the PowerPc's.

The terrain on which we test the navigation algorithms is approximately 100 meters long by 50 wide. It contains a variety of terrain types, ranging from flat obstacle-free areas to rough areas, including gentle and steep slopes, rocks, gravel, trees and bushes.

Part A: Navigation functionalities

[1] Lama is currently lent to LAAS by Alcatel Space Industries.
[2] currently lent to LAAS by CNES.

Figure 1: *The robot Lama on the experimentation site*

3 Environment Modeling

Perceiving and modeling the environment is of course a key capacity for the development of autonomous navigation. Environment models are actually required for several different functionalities: to plan paths, trajectories and perception tasks (section 4), to localize the robot (section 5), and also to servo the execution of trajectories. There is no "universal" terrain model that contains all the necessary informations for these various processes. It is more direct and easier to build different representations adapted to their use. The environment model is then *multi-layered and heterogeneous*, and perception is *multi-purpose*: several modeling processes coexist in the system, each dedicated to the building of specific representations.

3.1 Qualitative Model

We developed a method that produces a description of the terrain in term of *navigability classes*, on the basis of stereovision data [7]. Most of the existing contributions to produce similar terrain models come to a data segmentation procedure (*e.g.* [15, 9]), that produce a *binary* description of the environment, in terms of traversable and non-traversable areas. Our method is a classification procedure that produces a probabilistically labeled polygonal map, close to an occupancy grid representation. It is an *identification* process, and does not require any threshold determination (a tedious problem with segmentation algorithms).

Our method relies on a specific discretisation of the perceived area, that defines a *cell image*. The discretisation corresponds to the central projection on a virtual horizontal ground of a regular (Cartesian) discretisation in the sensor frame (figure 2). It "respects" the sensors characteristics: the cell's resolution decreases with the distance according to the decrease of the data resolution.

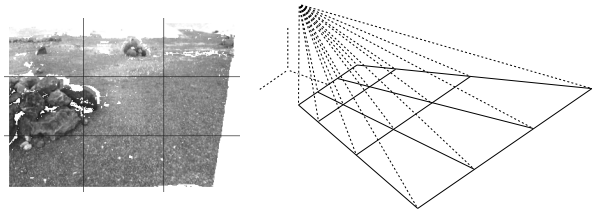

Figure 2: *Discretisation of a 3D stereo image. Left: regular Cartesian discretisation in the sensor frame; right: its projection on the ground (the actual discretisation is much finer)*

Features 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 a pre-defined traversability class is estimated. Figure 3 shows a classification results, with two terrain classes considered (flat and obstacle). There are several extensions to the method: the discretisation can be dynamically controlled to allow a finer description, and the classification results can be combined with a terrain physical nature classifier using texture or color attributes. One of its great advantages is that thanks to the probabilistic description, local maps perceived from different viewpoints can be very easily merged into a global description. The produced terrain model can be either used to generate elementary motions on rather obstacle-clear terrains (section 4.1), or to reason at the path level (section 4.3).

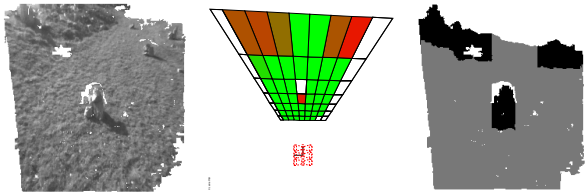

Figure 3: *An example of classification result. From left to right: image from stereo, partial probabilities of the cells to be an obstacle (represented as gray levels), and reprojection of the cells in the sensor frame, after the application of a symmetric decision function*

3.2 Digital Elevation Map

Digital elevation maps, *i.e.* ground elevations computed on a regular Cartesian grid, are a very common way to model rough terrains [11, 2]. Although there has been several contributions to this problem, we think that it has still not been addressed in very satisfactory way: the main difficulty comes from the uncertainties on the 3D input data, that can be fairly well estimated, but hardly propagated throughout the computations and represented in the grid structure.

However, a quite realistic model can be easily built by computing the mean elevation of the data points on the grid cells, using only the points that are provided with precise coordinates. With our stereovision algorithm for instance, 3D points whose depth is below $10m$ can be used to build a realistic $0.1 \times 0.1m$ cell digital elevation map. Provided the robot is localized with a precision of the order of the cell size, data acquired from several view-points can be merged into a global map (figure 4). This model is then used to detect landmarks (section 3.3) and to generate elementary trajectories (section 4.2).

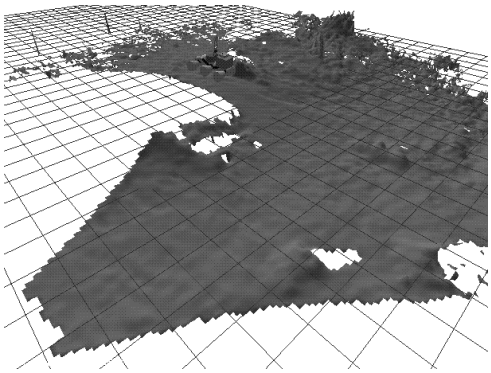

Figure 4: *A digital elevation map built by Lama during a 20 meter run using 50 stereovision pairs. The displayed grid is* $1 \times 1m$*, the actual map grid is* $0.1 \times 0.1m$*.*

3.3 Finding Landmarks

An efficient way to localize a rover is to rely on particular elements present in the environment, referred to as *landmarks* (section 5.3). A landmark should have the following properties: *(i)* it must be easy to identify, so that landmark association in different images can be performed; *(ii)* it must be "geometrically rich enough" to allow the refinement of all the parameters of an initial position estimation; *(iii)* its perceived geometric attributes must be as stable as possible, *i.e.* independent to viewpoint changes. The first requirement can be relaxed when a sufficiently good initial position estimation is available (which is the case we consider in section 5): matching landmarks extracted from different images is then simply done by comparing their estimated position. The second property can be bypassed when several landmarks are perceived. Local peaks, such as the summit of obstacles, satisfy the third property, are often numerous in planetary-like environments and can be quite easily extracted: we therefore decided to extract such features as landmarks.

Several authors presented 3D data segmentation procedures to extract salient objects, and our first attempts were based on a similar principle [3]. However, such techniques are efficient only in simple cases, *i.e.* in scenes were sparse rocks lie on a very flat terrain, but rather fragile on rough or highly cluttered terrains for instance. To robustly detect such local peaks, we are currently investigating a technique that relies on the computation of similarity scores between a digital elevation map area and a pre-defined 3D peak-like pattern (a paraboloid for instance), at various scales. First results are encouraging (figure 5), and the detected landmark could be used to feed a position estimation technique (section 5.3).

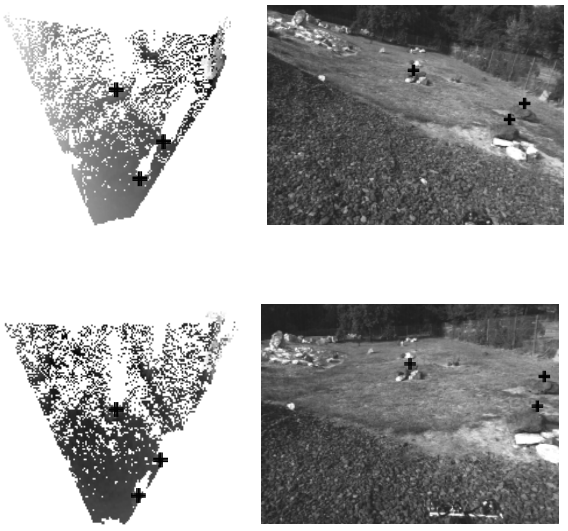

Figure 5: *Landmarks (black + signs) detected on the locally built digital elevation maps (left), and reprojected in the camera frame (right). Three meters separate the two image acquisitions, and landmarks are from 3 to 10 meters away.*

4 Trajectory generation

Natural terrains being unstructured, specific trajectory generation algorithms have to be developed. A generic trajectory planner able to deal with any situation should take into account all the constraints, such as rover stability, rover body collisions with the ground, kinematic and even dynamic constraints. The difficulty of the problem calls for high time-consuming algorithms, which would actually be quite inefficient in situations where much simpler techniques are applicable. We therefore think it is worth to endow the rover with various trajectory generation algorithms, dedicated the kind of terrain to traverse. Section 8 describes how they are actively started and controlled.

4.1 On easy terrains

On easy terrains, *i.e.* rather flat and lightly cluttered, dead-ends are very unlikely to occur. Therefore, the robot can efficiently move just on a basis of a goal to reach[3], and of a terrain model that exhibits non-traversable areas, using techniques that evaluate elementary motions [7].

To generate motions in such terrains, we use an algorithm that evaluates circle arcs on the basis of the global qualitative probabilistic model. Every cycle, the algorithm is run on an updated terrain model. It consists in evaluating the interest (in terms of reaching the goal) and the risk (in terms of terrain traversability) of a set of circle arcs (figure 6). The risk of an arc is defined in terms of the probability to encounter an obstacle; arcs whose risk is bigger than a chosen threshold are discarded, and the arc that maximizes the interest/risk ratio is chosen.

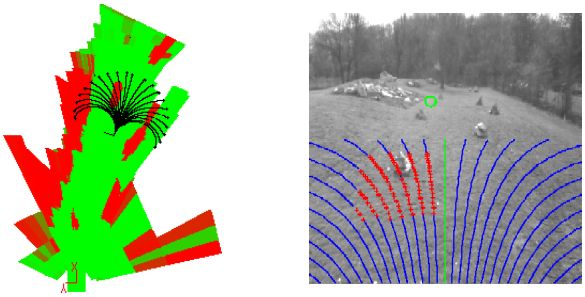

Figure 6: *A set of circle arcs to evaluate on the global probabilistic model (left), and reprojection of the arcs in the current camera view (right)*

4.2 On rough terrains

On uneven terrain, the notion of obstacle clearly depends on the capacity of the locomotion system to overcome terrain irregularities, and on specific constraints acting on the placement of the robot over the terrain. These constraints are the stability and collision constraints, plus, if the chassis is articulated, the configuration constraints (figure 7). To evaluate such constraints, the probabilistic qualitative model is not anymore sufficient: the digital elevation map is required.

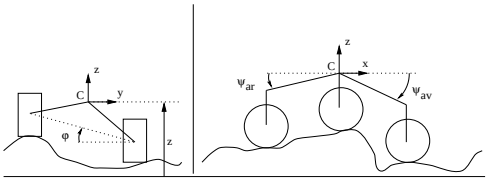

Figure 7: *The chassis internal configuration angles checked on the digital elevation map*

We developed a planner [8] that computes motions verifying such constraints by exploring a three dimensional configuration space $CS = (x, y, \theta)$ on the digital elevation map. This planner builds a graph of discrete configurations that can be reached from the initial position, by applying sequences of discrete controls.

[3] not necessarily the distant global goal, it can be a formerly selected sub-goal - see section 4.3

It is however quite time-consuming: we therefore evaluate elementary trajectories, in a way very similar to section 4.1. A set of circle arcs is produced, and for each arc, a discrete set of configurations are evaluated. Each arc is then given a *cost* that integrates the dangerousness of the successive configurations it contains, the arc to execute being the one that maximizes the interest/cost ratio.

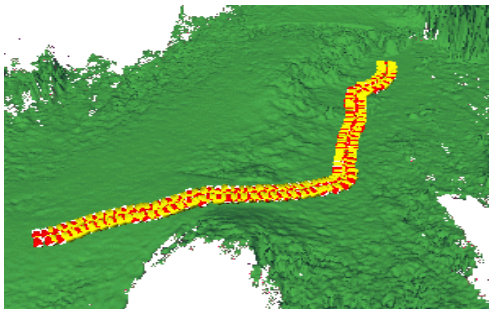

Figure 8: *A trajectory resulting from the application of the rough terrain local planner (approximately 30 cycles)*

4.3 Planning Paths

The two techniques described above only evaluate elementary *local* trajectories, and are therefore not able to efficiently deal with highly cluttered areas and dead-ends. For that purpose, we use a *path* planner, that reasons on the global qualitative model to find sub-goals and perception tasks [12].

The global qualitative model, which is built upon a bitmap structure, is segmented to produce a region map. This map defines a graph, in which a search algorithm provides an *optimal* path to reach the global goal. The "optimality" criterion takes here a crucial importance: it is a linear combination of time and consumed energy, weighted by the terrain class to cross *and the confidence of the terrain labeling*. Introducing the labeling confidence in the crossing cost of an arc amounts to *implicitly* consider the modeling capabilities of the robot: tolerating to cross obstacle areas labeled with a low confidence means that the robot is able to easily acquire informations on this area. The returned path is then analyzed, to produce a sub-goal to reach: it is the last node of the path that lies in a traversable area.

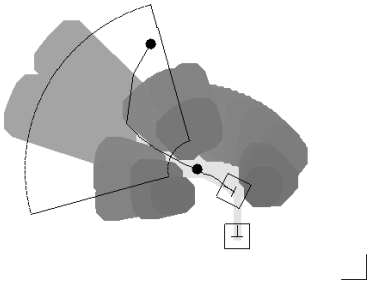

Figure 9: *A result of the navigation planner on the qualitative model: the result of the analysis of the shortest path found can be interpreted as the answer to the question "what area should be perceived to reach the goal ?"*

5 Localization

A position estimate is not only necessary to build coherent environment models, it is also required to ensure that the given mission is successfully being achieved, or to servo motions along a defined trajectory. Robot self-localization is actually one of the most important issue to tackle in autonomous navigation.

The various techniques required to compute the robot's position as it navigates range from inertial or odometry data integration to absolute localization with respect to an initial model. One can distinguish various algorithm categories: *(i) motion estimation* techniques, that integrate data at a very high pace as the robot moves (odometry, inertial navigation, visual motion estimation - sections 5.1 and 5.2), *(ii) position refinement* techniques, that rely on the matching of landmarks perceived from different positions, and *(iii) absolute localization* with respect to an initial global model of the environment. All these algorithms are complementary, and provide position estimates with different characteristics: we are convinced that an autonomous rover should be endowed with at least one instance of each category.

5.1 Odometry on Natural Terrain

Odometry on natural terrains is of course much less precise than on a perfect flat ground, but it can however bring some useful informations. We use 3D odometry with Lama by incorporating the attitude informations provided by the 2 axes inclinometer[4] to the translations measured by the encoders of the central wheels. Due to skid-steering, the angular orientation measured by the odometers is not reliable: the information provided by the integration of the gyrometer data is much better, and do not drift significantly before a few tens of minutes.

To have a quantitative idea of the precision of odometry, we gathered some statistics, using a carrier-phase DGPS as a reference. Figure 10 presents an histogram of the measured translation errors of odometry every $0.5m$. What is noticeable is the secondary peak around $0.1m$, that appeared during the traverse of a gravel area, where some longitudinal and lateral slippages occurred.

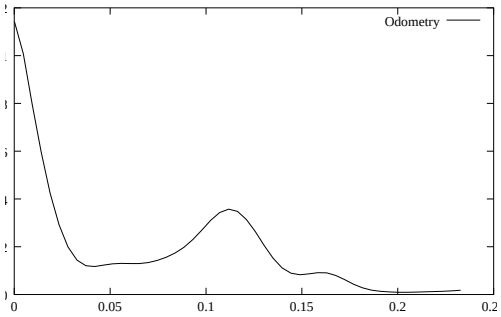

Figure 10: *Histogram of odometry errors measured every $0.5m$ steps during a 50 meter run with Lama on various kinds of ground.*

These preliminary figures show that odometry can hardly be modeled an estimator with Gaussian uncertainties: some *gross errors* actually occur quite often. We are currently investigating the possibility to analyze on line a set of proprioceptive data in order to be able to dynamically qualify the odometry, and especially to detect such errors. These data are the 6 wheel encoders, the measured currents, the two attitude parameters and the five chassis configuration parameters.

5.2 Visual motion estimation

We developed an exteroceptive position estimation technique that is able to estimate the 6 parameters of the robot displacements in any kind of environments, provided it is textured enough so that pixel-based stereovision works well: the presence of no particular landmark is required [13]. The technique computes the motion parameters between two stereo frames on

[4] after the application of a slight smoothing filter on its data.

the basis of a set of 3D point to 3D point matches, established by tracking the corresponding pixels in the image sequence acquired while the robot moves.

The principle of the approach is extremely simple, but we paid a lot of attention to the selection of the pixel to track: in order to avoid wrong correspondences, one must make sure that they can be faithfully tracked, and in order to have a precise estimation of the motion, one must choose pixels whose corresponding 3D points are known with a good accuracy. Pixel selection is done in three steps: an *a priori* selection is done on the basis of the stereo images; an empirical model of the pixel tracking algorithm is used to discard the dubious pixels during the tracking phase; and finally an outlier rejection is performed when computing an estimate of displacement between two stereo frames (*a posteriori selection*).

Figure 11 presents a set of positions visually estimated, on a $25m$ run. On this run, the algorithm gives translation estimates of about 4%. Similar precision has been obtained over several experiments, processing several hundreds of images. Work related to this algorithm is still under way, with the goal of reaching a precision on translation estimates of about 1%.

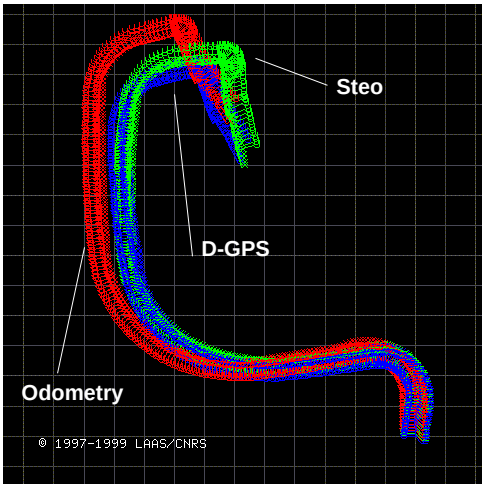

Figure 11: *Comparison of the position measured by odometry, the visual motion estimation estimator, and the DGPS reference*

5.3 Landmark Based Localization

We are currently investigating Set Theoretic approaches for landmark-based localization [14]. No statistical assumptions are made on the sensor errors: the only hypothesis is that errors are bounded in norm. Estimates of the robot and landmarks positions are derived in terms of *feasible uncertainty sets*, defined as regions in which the robot and the landmarks are guaranteed to lie, according to all the available informations.

Some simulation results using realistic bounds, using the landmarks detection algorithms presented in section 3.3 are promising. The integration of these algorithms on board Lama is currently under way.

Part B: Integration

6 A general architecture for autonomy

Our research group has been working for several years on the definition and development of a generic software and decisional architecture for autonomous machines. We briefly present here the concepts of this architecture that allows the integration of both decision-making and reactive capabilities (a detailed presentation can be found in [1]). This architecture has been successfully instantiated in multi-robot cooperation experiments, indoor mobile robotics experiments, and autonomous satellite simulations [6]. The architecture, sketched in figure 12, contains three different layers:

• *The functional level* includes all the basic robot action and perception capacities. These processing functions and control loops (image processing, obstacle avoidance, motion control, etc.) are encapsulated into controllable communicating *modules*. A module may read data exported by other modules, and output its own processing results in exported data structures. The organization of the modules is *not fixed*, their interactions depend on the task being executed and on the environment state. This is an important property that enables to achieve a flexible, reconfigurable robot behavior. Modules fit a standard structure, and are implemented thanks to a development environment, *Genom*.

Note that in order to make this level as hardware independent as possible, and hence portable from a robot to another, the functional level is interfaced with the sensors and effectors through a *logical robot level*.

• *The Executive* controls and coordinates the execution of the functions distributed in the modules according to the task requirements. It's function is to fill the gap between the decision and functional levels decision, *i.e.* between the *slow rate* logical reasoning on *symbolic* data, and the *higher bandwidth* computation on *numerical* data. It is a purely reactive system, with no planning capability. It receives from the decision level the sequences of actions to be executed, and selects, parameterizes and synchronizes dynamically the adequate functions of the functional level.

• *The decision level* includes the capacities of producing the task plans and supervising their execution, while being at the same time reactive to events from the previous level. This level may be decomposed into two or more layers, based on the same conceptual design, but using different representation abstractions or different algorithmic tools, and having different temporal properties. This choice is mainly application dependent.

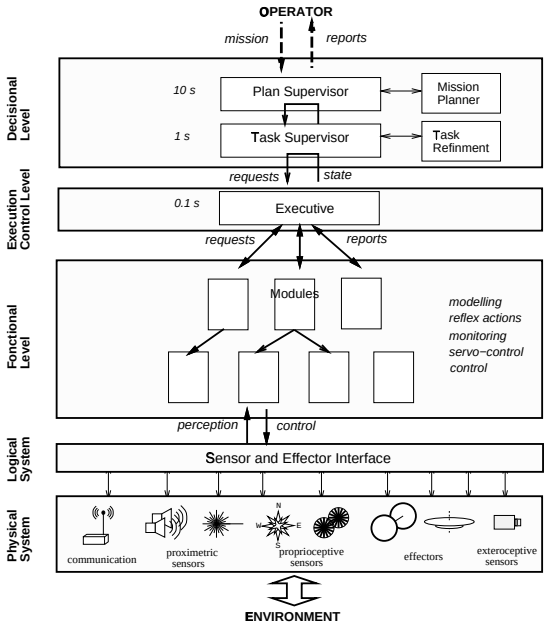

Figure 12: *LAAS architecture for robot autonomy.*

Up to now, all the algorithms are integrated as modules on board Lama, while the executive is simply implemented as scripts written in Tcl.

7 Integration of Concurrent Localization Algorithms

Some particular integration problems are related to the coexistence of several localization algorithms running in parallel on board the robot. To tackle this in a generic and reconfigurable way, we developed a particular module named PoM (position manager), that receives all the position estimates produced by the localization as inputs, and produces a single consistent position estimate as an output. PoM addresses the following issues:

• *Sensor geometrical distribution.* The sensors being distributed all over the robot, one must know their relative positions, which can dynamically change. Indeed, we want all the modules to be generic, *i.e.* to handle data without having to consider the position from which it had been acquired. This is particularly true for video images, since Lama is equipped with two orientable stereo benches: we may want to switch benches whenever required, with the minimal effort. For this purpose, we developed a framework named "InSitu" (internal situation, section 7.1) which is part of PoM.

• *Localization modules asynchronism.* The localization algorithms have individual time properties: some produce a position estimate at a regular high frequency, while others require some non-constant computation time. To be able to provide a consistent position estimate at any time, the "time-management" part of PoM has been developed (section 7.2).

• *Fusion of the various position estimates.* The fusion of various position estimates is a key issue in robot localization. This is done within the "fusion" part of PoM (discussed in section 7.2).

7.1 Internal situation

Some problems arise when the sensors of a robot are geometrically distributed. For instance, the vision-based localization module has to know the orientation of the cameras for each image it processes, whereas the digital elevation map module needs the relative and/or absolute position of the *3-D* images it is using in a predefined coordinate frame.

Distributing the geometrical informations in the modules is not satisfying: some informations are hard-coded and duplicated within the modules, and it complicates the porting of a module to another robot or another sensor. For that purpose, we use InSitu, a centralized geometrical description of a robot. It reads a configuration file upon startup, and it provides the necessary frame coordinates to any module when the robot navigates. All the data acquisition modules use this information to tag the data they produce with the necessary informations.

The configuration file is the textual description of a geometrical graph (figure 13). The nodes of the graph are frames coordinates that needs to be exported. They usually correspond to sensors locations but can also be a convenient way to split otherwise complex links. This graph is the only robot-specific part of PoM.

The links between frames are either *static* (rigid) or *dynamic* (mobile). A static link cannot change during execution and is usually related to some mechanical part of the robot. On the other hand, dynamic links, that depend on the chassis configuration or on a PTU configuration for instance, are updated continuously at a predefined frequency. Updates are made possible with the definition of link specific functions that get links parameters from the underlying hardware system. These functions are grouped together in a library associated with the configuration file. Last, the configuration file defines a *common* frame (also called *main* frame). To facilitate and homogenize inter-module data transfers, the various modules data are expressed in this frame.

To ease data manipulation and transfer among the modules, InSitu continuously exports all the frames configurations found in the configuration file with the structure shown in figure 14.

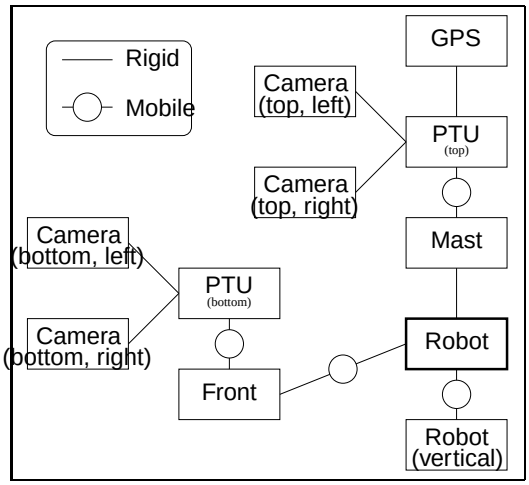

Figure 13: *The geometrical graph used on the robot* Lama. *Rectangular boxes are frames configurations that are exported in the system. The thick box (*Robot *in the figure) is the* main *frame. Solid lines are links (either rigid or mobile) that are declared in the configuration file. Each mobile link has an associated function that gets the link parameters from the underlying hardware, computes the current transformation matrix and sends it back to PoM.*

The header of exported data contains three fields: the first field is the current PoM-date expressed in ticks since boot-time. The next two fields are positions: the *Main to Origin* transformation is the current *absolute* position of the *main* frame relative to an absolute frame. The *Origin* frame is usually the position of the robot at boot-time but it can be specified anywhere. The *Main to Base* transformation is a relative position which cannot be used as such. The sole operation permitted with this frame is the composition with another *Main to Base* transformation (see section 7.2 for a detailed description). *Base* is a virtual frame that is maintained and computed by PoM.

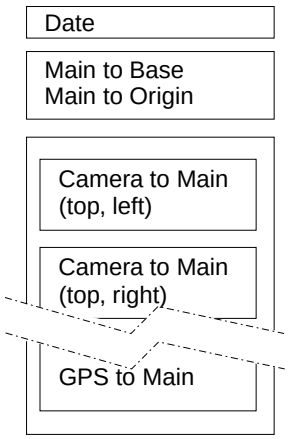

Figure 14: *Structure exported by InSitu.*

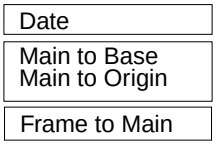

Figure 15: *Structure for data tagging.*

After the header, there is a variable number of transformations which are the current frames configuration. They are named according to the scheme "*Frame to Main*" and are the transformation matrices that map *3-D* points in the "*Frame*" frame to *3-D* points in the "*Main*" frame.

Every data acquisition module reads the InSitu frame which it relates to, and associate to its data a "tag" structure (figure 15). Thanks to this tagging, data acquisition modules only have to know the name of the frame they are connected to, which can be specified dynamically. Once tagging is done, clients using such

data do not have to care from where the data comes, since all the necessary geometrical and time information is contained in the data itself. The tag is propagated along with the data between modules, thus making inter-module data communication very flexible.

7.2 Position Management

In addition to internal frame configurations, PoM collects the various position estimators present in the system. It is the place where positional information is centralized and made available for any module that require it.

Positions computed by the various position estimators are always produced with some delay, that depends on the computation time required to produce a particular position. Thus the robot has always to deal with outdated positions. One of the role of PoM is to maintain the time consistency between the various position estimates.

Internally, there is one time chart for each position estimator handled by PoM, plus one particular chart for the *fused* position. The *fused* position is the result of the fusion of every position estimator (see section 7.2). All charts have the same length and hold every produced positions since the beginning of the chart, up to the current date. Length is variable but is computed so that we always have at least two positions of every motion estimator (figure 16).

PoM periodically polls every position estimator and look if a position have been updated. When a new position is found, it is stored in the chart of the corresponding motion estimator, at date its corresponding date ("updates" arrows in figure 16). If the chart is not long enough to store a too old date, it is enlarged as needed. Once every motion estimator has been polled, every fused position from the oldest update to the current time is marked for re-estimation.

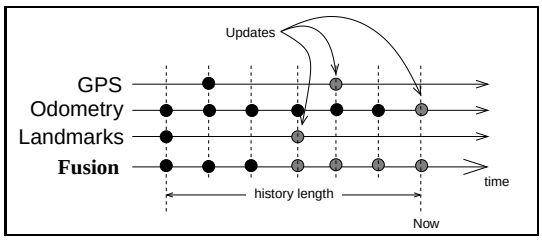

Figure 16: *Time management and internal representation of motion estimators in PoM. Position estimators shown here are only examples. Black dots show previously stored positions. Dashed dots show new upcoming positions.*

Actually, no data fusion algorithm is currently implemented within PoM: given the individual position estimator characteristics, we indeed consider that a *consistency check* (fault detection) has to be performed formerly to any fusion. Up to now, an estimate selection is performed on the basis of a confidence (real value between 0.0 and 1.0, 1.0 being the best) that is hard-coded for each position estimator.

Once the current best position is computed, PoM stores it in the *fusion* time chart (figure 16). The important thing to note is that we cannot simply store the new position as such because it might be very different from the previous position. Since some module may use the robot position continuously (such as servoing on a computed trajectory for instance), virtual jumps are not permitted. This is why we defined a *reference position estimator*, and a virtual frame named *Base*. The fused position computed by PoM is the *Robot to Origin* position. This one is exported as such and used by modules that require *absolute* position. The second exported position, *Robot to Base* is the position of the *reference estimator* and can only be used locally, *i.e.* to compute a *delta* position. PoM computes a *Base to Origin* transformation for every position estimator found in the system. This transformation is an indicator of the drift of each position estimator.

8 Navigation strategies

The integration of the various algorithms presented in the first part of the paper requires specific decisional abilities, that are currently instantiated as Tcl scripts. The following simple strategy is currently applied: the three environment models (qualitative map, digital map and landmark map) are *continuously* updated every time new data are gathered, and the two integrated localization algorithms (odometry and visual motion estimate) are also continuously running[5]. The selection of the trajectory generation algorithm is the following: given a global goal to reach, the easy terrain algorithm, which requires negligible CPU time, is applied until no feasible arcs can be found. In such cases, the rough terrain algorithm is applied. It is run until either the easy terrain algorithm succeeds again, or until no feasible arcs are found in the digital map. In the latter case (that can be assimilated to a dead end), the path planning algorithm is run, to select a sub-goal to reach. The whole strategy is then applied to reach the sub-goal, and so on.

9 Conclusion

We insisted on the fact that to efficiently achieve autonomous long range navigation, various algorithms have to be developed for each of the basic navigation functions (environment modeling, localization and motion generation). Such a paradigm eventually leads to the development of a complex integrated system, thus requiring the development of integration tools, at both the functional and decisional levels. We are convinced that such tools are the key to implement efficient autonomy on large time and space ranges.

There are however several open issues. Among these, we believe that the most important one is still localization. In particular, the system must be robust to extremely large uncertainties on the position estimates, that will eventually occur: this requires the development of landmark *recognition* abilities to tackle the data association problem, and also the development of terrain model structures that can tolerate large distortions. Note that both problems should benefit from the availability of an initial terrain map, such as provided by an orbiter, whose spatial consistency is ensured. Indeed, the development of algorithms that match locally built terrain models with such an initial map would guarantee bounds on the error of the position estimates.

References

[1] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An architecture for autonomy. *Special Issue of the International Journal of Robotics Research on Integrated Architectures for Robot Control and Programming*, 17(4):315–337, April 1998. Rapport LAAS N97352, Septembre 1997, 46p.

[2] P. Ballard and F. Vacherand. The manhattan method : A fast cartesian elevation map reconstruction from range data. In *IEEE International Conference on Robotics and Automation, San Diego, Ca. (USA)*, pages 143–148, 1994.

[3] S. Betge-Brezetz, R. Chatila, and M.Devy. Object-based modelling and localization in natural environments. In

[5] landmark-based localization integration is under way, but is should also be continuously run, while *actively* controlling the image acquisition.

IEEE International Conference on Robotics and Automation, Nagoya (Japan), pages 2920–2927, May 1995.

[4] R. Chatila. Deliberation and reactivity in autonomous mobile robots. *Robotics and Autonomous Systems*, 16(2-4):197–211, 1995.

[5] R. Chatila and S. Lacroix. A case study in machine intelligence: Adaptive autonomous space rovers. In A. Zelinsky, editor, *Field and service Robotics*, number XI in Lecture Notes in Control and Information Science. Springer, July 1998.

[6] J. Gout, S. Fleury, and H. Schindler. A new design approach of software architecture for an autonomous observation satellite. In *5th International Symposium on Artificial Intelligence, Robotics and Automation in Space, Noordwijk (The Netherlands)*, June 1999.

[7] H. Haddad, M. Khatib, S. Lacroix, and R. Chatila. Reactive navigation in outdoor environments using potential fields. In *International Conference on Robotics and Automation, Leuven (Belgium)*, pages 1232–1237, May 1998.

[8] A. Hait, T. Simeon, and M. Taix. Robust motion planning for rough terrain navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, Kyongju (Korea)*, pages 11–16, Oct. 1999.

[9] 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.

[10] 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.

[11] I.S. Kweon and T. Kanade. High-resolution terrain map from multiple sensor data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):278–292, Feb. 1992.

[12] S. Lacroix and R. Chatila. Motion and perception strategies for outdoor mobile robot navigation in unknown environments. In *4th International Symposium on Experimental Robotics, Stanford, California (USA)*, July 1995.

[13] A. Mallet, S. Lacroix, and L. Gallo. Postion estimation in outdoor environments using pixel tracking and stereovision. In *IEEE International Conference on Robotics and Automation, San Francisco, Ca (USA)*, pages 3519–3524, April 2000.

[14] M. Di Marco, A. Garulli, S. Lacroix, and A. Vicino. A set theoretic approach to the simultaneous localization and map building problem. In *39th IEEE Conference on Decision and Control, Sydney (Australia)*, Dec. 2000.

[15] 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.