

Autonomous Rover Navigation in Partially Known Terrain

Pierre Allard Sébastien Gemme Régent L'Archevêque Brian Moore¹ Erick Dupuis

Canadian Space Agency,

6767 Route de l'Aéroport, St-Hubert, Québec, Canada, J3Y 8Y9

pierre.allard@space.gc.ca

Keywords Artificial Intelligence, Space Autonomy,
Space Robotics, Mobile Robotics, i-SAIRAS

Abstract

This paper describes the work done in autonomous operation of mobile robot at the Canadian Space Agency. The *Mobile Robotic Test bed* (MRT) components are presented. The autonomy software, implemented using a behaviour-based approach, is presented. The peripheral sub-systems used by the reactive engine, such as vision and path planner are also described.

1. Introduction

So far, all landed planetary exploration missions have used little autonomy. Pathfinder relied heavily on operator interaction to plan its sequence of operations every day and was used mostly in a teach-and-playback mode. Similarly, the Mars Exploration Rovers (MER), although they have more autonomy capability, will likely still be operated with heavy operator involvement. It is widely accepted that, eventually, planetary exploration missions will require more autonomy to maximize the number of operations that can be conducted per sol and thus the science return on investment. Missions beyond the horizon of the current planning process may require autonomous navigation over large distances with relatively little operator interaction or may involve fleets of robots performing operations in a cooperative manner.

The project described here implemented a rudimentary autonomous navigation capability on a

laboratory mobile robot. The on-board autonomy software was implemented based on a reactive inference engine that was originally developed for space manipulation tasks [1]. The capabilities that were implemented included the ability for a mobile robot to localize itself, to plan a path to a final destination, to detect obstacles and react to anomalies in the environment.

2. Overview

The scenario implemented on the MRT involves autonomous navigation in partially known terrain. At the start of the scenario, the robot is placed inside a work area. The work area contains boxes that serve as obstacles. The robot knows the position of most of these obstacles, but does not know its position in the terrain.

Using the *Ground Control Station* (GCS), an operator send a single high-level command asking the robot to go to a position inside the terrain. Upon reception of the command, the robot finds its current position using a vision system, plans a path to its commanded destination, and starts a traverse.

If an obstacle is encountered during the traverse, the robot stops and confirms its odometry-based position using its vision system. The position of the obstacle in the terrain coordinates is computed and the obstacle added to the representation of the terrain. A new path is then planned around the obstacle, and the traverse resumes.

The autonomous operation of the robot is implemented using a behaviour-based approach that decomposes high-level tasks into simpler sub-tasks.

¹ Opal-RT Technologies on secondment to Canadian Space Agency

The behaviours, which are managed by a reactive engine, access the robot systems to get sensory inputs, control the action of the robot, control its peripherals, and access its processing subsystems such as the vision system and path planner. The reactive engine receives high-level commands from the GCS, and transmits back telemetry to allow the operator to follow the robot operations.

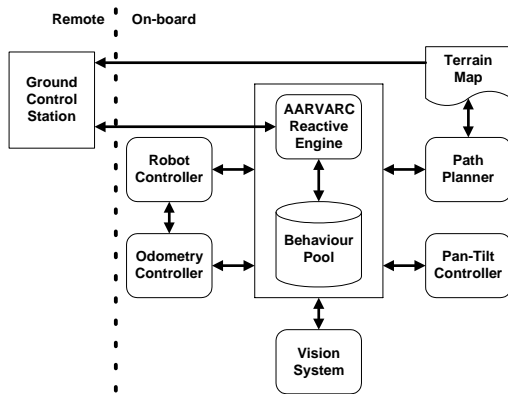


Figure 1. MRT Overall Architecture.

2.1 MRT Environment

The MRT facilities include three main elements: a mobile robot, an indoor terrain with obstacles, and a Ground Control Station.

The mobile robot is a commercially available unit: a *Pioneer P2-AT* from *ActiveMedia Robotics*. It is a four-wheel, skid-steered robot equipped with a sonar array located on the periphery of the robot deck. The control of the wheels and the acquisition of the odometry and sonar's data are supported by an embedded micro controller built-in the robot. Communication with the micro controller is done through a RS-232 link.

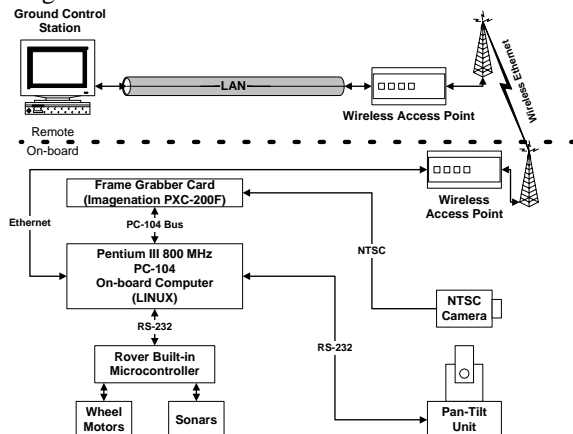


Figure 2. MRT Hardware Architecture.

The robot is equipped with a PC-104 Pentium III 800 MHz computer running LINUX operating system. The computer is equipped with a PXC-200F frame grabber card from *Imagination*. A NTSC colour camera connected to the frame grabber provides imaging capability to the robot. The camera is mounted on a pan-tilt unit controlled using an RS-232 connection.



Figure 3. The MRT robot traversing the terrain.

The mobile robot is used indoors in a work area that is 4 meters wide by 8 meters long. Square boxes distributed in the work area are used as obstacles, and their positions are precisely known. Luminous beacons, made using standard fluorescent lamps, are placed on three corners of the work area. The robot uses these beacons to triangulate its position.

The GCS is used to monitor the current robot status and telemetry information. The GCS GUI runs on a standard PC under Windows 2000. The communication with the robot is achieved through the LAN via a wireless Ethernet connection.



Figure 4. The GCS GUI with terrain map showing actual vs planned trajectory.

The GCS GUI is implemented using JAVA for cross-platform compatibility. The GUI allows the user to connect to the robot reactive engine to allow for commands to be sent and telemetry to be received. Sockets are used for both the command and telemetry streams. The GUI provides a visual display of the terrain map on which the robot planned and actual trajectories are plotted dynamically. The terrain map is updated every time the Path Planner adds an obstacle. Additional data, such as sonar readings and battery level are also displayed.

2.2 On-Board Autonomy

The on-board autonomy software is based on the *Amorphous Architecture for Variable Autonomy Robot Control (AArVARC)* [2]: a reactive engine based on Hierarchical Task Network (HTN) concept that allows the implementation of local and distributed autonomy. AArVARC can support applications with wildly varying levels of autonomy. For example, on the International Space Station, operations of the Mobile Servicing System will be conducted partially from the ground to free-up astronauts to conduct more science experiments. In this case, the autonomy requirements are relatively low since the environment is known and static, and the communication delays are relatively short.

In contrast, for a rover or manipulator on a Mars exploration mission, the operator will be located on Earth and will interact with the robot only once or

twice every day. The communication delays in this case will be on the order of 20-40 minutes. Such an application will therefore require a very high level of autonomy: the robot being sent command scripts for up to 12 hours of operations at a time.

AArVARC uses different behaviours to specify the actions to be taken according to the events and the environment changes. It is thus possible to create a generic pool of behaviours. Each behaviour may implement completely a complex task or may invoke, spawn or request existing behaviours to achieve its goal. By reducing the information transferred between the GCS and the remote system, this approach does not require communication link with high bandwidth. The AArVARC engine receives high level commands from the GCS, seeks for the proper behaviour and decomposes it into lower level behaviours (Figure 5). Once execution is completed, the engine reports the status of the system to the GCS. Notification of malfunctions may be handled by the behaviours themselves or may be reported to the operator for human intervention.

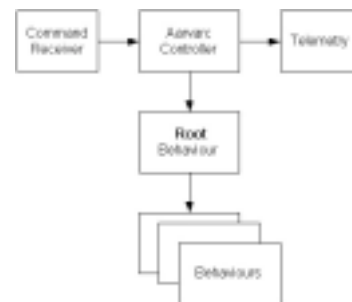


Figure 5. AArVARC Architecture.

The current release of MRT autonomy uses a set of 11 behaviours. As shown in Figure 6, the highest-level behaviour, named *GoToLocation*, spawns in sequence 4 Behaviours. *LocateSelf* seeks beacons using camera mounted on a pan-tilt unit. Then a trajectory is requested to *PlanTrajectory*. Once determined, *FollowTrajectory* invokes *TurnOnTheSpot* and *MoveInLine* sequentially in order to move along that trajectory. *DetectObstacle* and *TrackBeacon* are spawned in parallel to prevent collision with unknown obstacles and to track one beacon to ease future

localization if an obstacle is detected. Each of them has limited capability to handle malfunctions such as no path available, pan-tilt problem, beacon not detected, etc.

By design, the AArVARC architecture allows invocation of behaviours at any level in the tree. This feature, coupled with the dynamic behaviour loading capability that allows the behaviour pool to be modified on-line, is useful to recover from handled malfunctions. It is also helpful during the development of new behaviours.



Figure 6. MRT Behaviours Tree.

2.3 Vision

Localization is achieved using a vision approach. Why vision? It seemed to be the natural approach to identify objects at known position used to triangulate the robot position. In the current implementation, a method that filters colors and finds patterns in the resulting image is used. While simple, this method is very sensitive to light variations, and calibration of the color matching algorithm had to be done almost every time the system was used.

The vision system extracts beacons, which show up as vertical segmented lines. Three different beacons have been installed on three of the four corners of the test terrain, and are made of fluorescent lamps mounted on a vertical post. The beacons have one, two and three segments that are used to uniquely identify each beacon.

The vision system is implemented in Java and is using the JAI (Java Advanced Imaging) API developed by Sun Microsystems for low level image processing. The vision system acquires imagery from the NTSC

camera using the frame grabber. A color filter is then applied to the raw image to remove unwanted background scenery. This filter produces a binary image where all pixels that do not match, within a given threshold, the beacon reference color, are set to black, while the matches are set to white. A vertical convolution filter is then applied to the binary image. This additional filtering reduces the noise that might appear in the image, such as spots of colors that are not part of a beacon but are falling within the color's threshold.

The next step involves grouping the remaining pixels into blobs. The blob construction is achieved using a standard *flood fill* algorithm with a threshold based on inter-pixel distance. [3]. A database of all the blobs in the image is built in the process.

Using the blobs database, the number of blobs vertically aligned (with a tolerance on the vertical position of their centroid) is counted to identify the beacons. In the figure presented below, two vertically aligned blobs (circled) are identified as a beacon (beacon #2). The smaller white spots are not considered blobs since they don't contain enough pixels. Once the beacon is detected, its angular position from the center of the field of view the camera is computed (the camera has a field of view of about 50 degrees) and returned to the reactive engine. The computation of the angular position from the pixel frame position is based on experimental data, and produces results with a precision of 0.5 degrees.

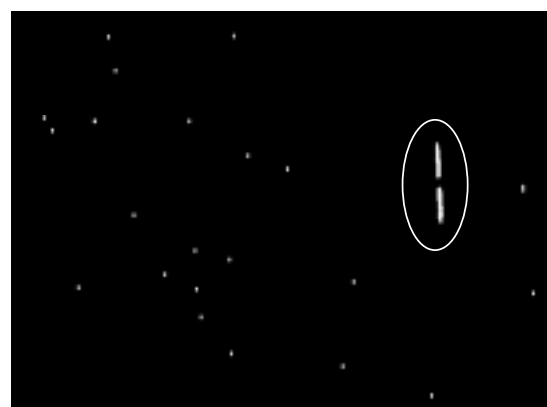


Figure 7. Filtered image.

Future implementations will use pattern recognition,

to detect beacons from noisy image, such as different groupings of squares to identify the beacons.

Using images with a resolution of 640x480, the complete vision algorithm can be performed at 2Hz on the robot on-board computer.

2.4 Localization

Localization uses the vision system to compute the robot position in the terrain. The localization behaviour first moves the camera in the direction where the first beacon should be found, based on the robot initial position estimate. Once the first beacon is detected, the camera is re-centered on the beacon, the vision system called for a new beacon angle measurement, and the camera is re-centered on the beacon. The process is repeated until the position of the beacon is close enough to the center of the camera view, or the maximum number of iteration is reached. The process is repeated for the remaining two beacons.

If the vision system fails to detect the specified beacon in the view, the behaviour assumes that the robot initial position estimate is wrong and starts scanning the entire horizon, calling the vision system at every 45 degrees interval, until all three beacons are found.

Once the azimuth of the three beacons has been found, the position and orientation is compute using simple geometry.

2.5 Path Planning

Path planning approaches based on partial knowledge of the terrain can be classified in two categories. First, the *potential field approaches* use a method similar to the analysis of viscous fluid flow to find the shortest path between two rover configurations [4][5]. The second category is formed of the *cell decomposition approaches* in which an efficient representation of the terrain is obtained by dividing it in uniform and non-uniform cells, and then planning a path between the cells [6][7]. In the MRT Path Planner, we use a *cell decomposition approach* since any cost functions can be used as optimization criterion instead

of only the distance.

Three functionalities of the Path Planner module are called from the reactive engine: **Initialization**, **GenerateTrajectory** and **AddObstacle**. The **Initialization** is used to create a discretized and efficient representation of the terrain. The **GenerateTrajectory** uses this map to generate an optimal trajectory between two configurations of the rover. Finally, **AddObstacle** is used to update the terrain map based on the information collected by the rover when it encounters obstacles.

At initialization, a traversability map is created from a known elevation map of the terrain as shown in Figure 8. In this figure, the white areas represent obstacles and are associated a huge cost. The gray zones represents bounding boxes to prevent the rover from moving too close to the boxes and are assigned a high cost. Finally, a low cost is assigned to the black areas that are safe areas. Other criteria could be used to evaluate the traversability cost like the terrain slope, the level of confidence of the map data and the scientific interest of the area. The traversability map is then decomposed in uniform cells (safe cells and unsafe cells) and small uncertain cells using a quadtree representation as illustrated in Figure 9 [6]. A graph describing the connectivity between the quadtree nodes is then derived from the quadtree traversability map.

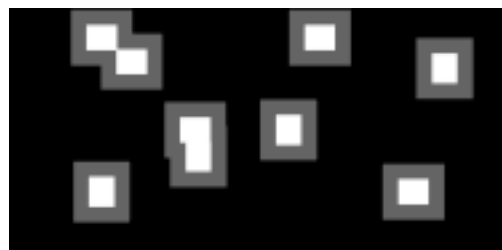


Figure 8. Traversability Map.

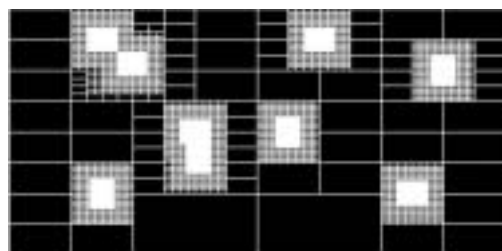


Figure 9. QuadTree Map.

Using the quadtree map, the connectivity graph, the location of the rover obtained by the localization module and a target location, **GenerateTrajectory** finds the shortest path between the rover actual location and the target location in terms of the quadtree nodes (Figure 10). A trajectory passing through these cells is then generated using straight lines (Figure 11). Post-processing of the trajectory is then applied in order to optimize the trajectory by removing unnecessary points on the trajectory (Figure 12).

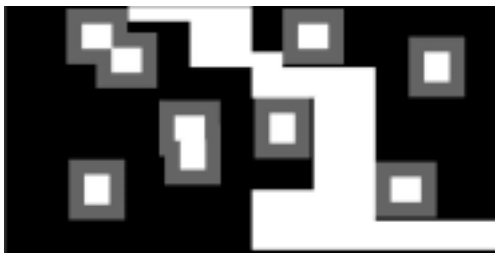


Figure 10. Quadtree nodes path.



Figure 11. Trajectory.



Figure 12: Refined trajectory

In some cases, part of the map could be incomplete or unknown. Therefore, there is a need to be able to update the traversability map when obstacles are detected. This is implemented by **AddObstacle**. For

example, if an obstacle is found, the traversability map is updated by adding a non-safe area around the obstacle position. The quadtree traversability map and the connectivity graph are also updated based on the new information (Figure 13). Therefore a new trajectory can be planned to avoid the obstacle. Using these functionalities, it is also possible to start with a blank map and create the terrain map along the rover trajectory as obstacles are found.

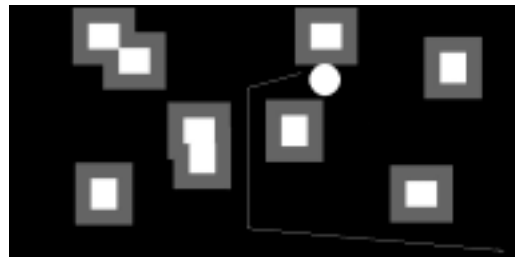


Figure 13. Map update with obstacle and trajectory update.

3. Conclusion

The project described in this paper implemented a rudimentary autonomous navigation capability on a laboratory mobile robot. The on-board autonomy software was implemented based on the CSA's AArVARC reactive inference engine. The capabilities that were implemented included the ability for a mobile robot to localize itself using a beacon-based vision system, to plan a path to a final destination, to detect obstacles using sonar arrays and to react to anomalies in the environment.

Future work includes the incorporation of much more capable 3D environment sensors (such as a lidar). This will enable the robot to map out and navigate in outdoors setting much more representative of planetary exploration environments. The capability for multiple robots to coordinate their actions in order to conduct a complex task will also be implemented using the same suite of tools.

References

- [1] Dupuis, E. and Gillett, R., "Remote Operation with Supervised Autonomy", Proceeding of 7th ESA Workshop on Advanced Space Technologies for Robotics and Automation, ASTRA 2002, Noordwijk, the Netherlands, November 2002.

[2] E. Dupuis, R. L'Archeveque, "Autonomous Robotics and Ground Operations", iSAIRAS 2003, May 2003.

[3] Foley, Van Dam, Feiner, Higes, "Computer Graphics, Principles and Practice", Addison-Wesley, 1997.

[4] Y. Hwang and N. Ahuja, "A potential field approach to path planning", IEEE Transactions on Robotics and Automation, 8(1), pages 23-32, 1992.

[5] C. Louste and A. Liégeois, "Path planning for non-holonomic vehicles: a potential viscous fluid field method", Robotica, 20, pages 291-298, 2002.

[6] A. Yahja, S. Singh and A. Stentz, "An efficient on-line path planner for outdoor mobile robots", Robotics and Autonomous Systems, 32, pages 129-143, 2000.

[7] S. Al-Hasan, G. Vachtsevanos, "Intelligent route planning for fast autonomous vehicles operating in a large natural terrain", Robotics and Autonomous Systems, 40, pages 1-24, 2002.