Acta Futura 5 (2012) 9-16
DOI: 10.2420/AF05.2012.9

Acta Futura

Modular Design for Planetary Rover Autonomous Navigation Software using ROS

Yang Gao[*], Renato Samperio, Karin Shala, and Ye Cheng

Surrey Space Centre, University of Surrey, Guildford, Surrey, UK, GU2 7XH

Abstract. This paper presents a modular design concept of autonomous navigation software for planetary rovers. The software covers major navigation functions such as autonomous localisation and mapping, visual rock detection, and path planning. The proposed design includes a generic data pipeline which produces a sequence of data products based on sensory raw data. To effectively and efficiently integrate the various design elements, Robot Operating System (known as ROS) is used as the middleware framework to implement the generic data pipeline and synthesize various navigation functions in terms of ROS nodes. The paper also presents test results of the proposed software implemented within the Surrey Rover Autonomous Software and Hardware Testbed (SMART) based on real and artificial data.

1 Introduction

For planetary rovers, autonomous navigation is desirable since responsive remote operation is not possible due to communication latency. Design of onboard autonomous navigation software is therefore important to effective operation of the rover. Advanced navigation software should allow the rover to autonomously build maps of its environment, identify its own location within the map, and be able to plan an optimal path for a given target. These functions involve processing sensory data from different sources, at different frequencies and qualities. Traditionally the data processing is designed independently within each navigation function and not shared between functions [11]. This can cause duplication of data products or prevent useful data product to be shared by more than one function. This paper addresses this issue by proposing a generic data pipeline process that links all the key navigation functions such as localization, mapping, and path planning, etc. These functions are effectively integrated using strategy pattern design (see [1]) into software modules for encapsulating interchangeable algorithms.

As navigation software frequently embraces new algorithms and techniques, it is also useful to employ a generic and modular framework at the software architectural level so that the system can be easily reconfigured to add or remove algorithms without dramatic change of the software architecture. This is in line with the general concept of common data pipeline mentioned above. Thanks to latest development of Robot Operating System (ROS) [15] for terrestrial robotics, a common software framework can be designed in a similar manner and hence demonstrated based on ROS. In this paper, algorithms for three major navigation functions are implemented in software modules or ROS

[*]Corresponding author. E-mail: yang.gao@surrey.ac.uk

nodes, which are also used to demonstrate the generic data pipeline products for rover environment modelling.

Section 2 of the paper outlines the challenges for designing rover autonomous navigation software and literature work. Section 3 presents the proposed design method to achieve standard data pipeline and module design for the navigation software. In Section 4 implementation using SMART is described including examples of experimental results obtained in laboratory conditions. Finally, Section 5 gives a conclusion and suggests possible future work.

2 Related Work

For a planetary rover, the mission is normally a goal to reach a specific location in its surrounding area where subsequent in-situ scientific investigation and sample collection is required, as outlined in [17]. Autonomous rover navigation often makes use of waypoint navigation that integrates different sources of sensory data. This is a procedure that guides the rover movement by defining a sequence of waypoints as a route to accomplish. The rover navigation system has always been divided into various functions or tasks essential to the navigation process. Typically, the functions are for path planning, rock detection, localisation, mapping and mission control for goal designation [11]. Each function requires particular processing of the raw sensor data for further real-time decision-making [10, 9, 3].

Most rover navigation combines autonomous functions with sporadic human inputs from ground control as described in [7]. In most cases human intervention is limited and delayed. Reliable autonomous navigation is highly desirable but it also means that the rover needs to rely on exhaustive methods for analysing terrain conditions to generate a safe route and motion plan [14]. In recent work [4], long-range navigation strategies have been introduced to rovers which condense visual information into representative data structures. Work by [20] focuses on prototyping new rover navigation software for long distance traversal with application to future missions such as Exomars, making use of experience from terrestrial applications.

The work presented in this paper focuses on navigation software design that facilitates interactions between individual functions. Also, modular real-time processes are described by communicating standardised data products, as in [2]. The data products derived from camera [19] and/or LIDAR scans provide models of the rover environment. The model built on exchangeable data products are integrated into different software modules using ROS open source framework [15]. As a result, the proposed software design offers a generic approach for implementing navigation system and a data processing pipeline.

3 Proposed data pipeline & modular design

Figure 1 illustrates the proposed data pipeline from sensor calibration (for acquiring raw data) to higher level data products. The pipeline standardise the sequence of data processing so that generic data products can be generated systematically to serve multiply navigation functions, such as localization, mapping, obstacle detection and path planning. Some data products can be related to environment models for the rover (such as keypoints and maps). Within the same level of product, data fusion could have been performed if more than one type of sensory data is used such as using Data Fusion process model described in [6].

In this specific design, data processing goes through three broad stages in order to generate a map for rover navigation. It begins with pre-processing of raw sensor data (or data enhancement) and the sensor calibration to remove systematic and random noise. Afterwards, the enhanced data is redefined in its size and format according to relevant navigation function. The data product of the second stage transforms raw data into features describing rover environment. The last stage makes use of space projections in 2D/3D to generate distinctive map(s) for each navigation function. The final data product is therefore a map or model of the rover environment. The map is constantly updated.

The strategy pattern design methodology described in [1] is used to integrate the data pipeline with different navigation functions. There are three patterns involved in the proposed design (see Figure 2):

- **Strategy**: Also known as policy, is a class that contains a definition of the stages of data processing.
- **Concrete Strategy**: These are elements that implement specific methods for data control according to autonomous navigation functions.
- **Context**: In this use case, the class context contains a ROS node for its interaction and communication with messages/services within other strategies.

Effectively, a general strategy is used to represent the proposed data pipeline process. Three concrete

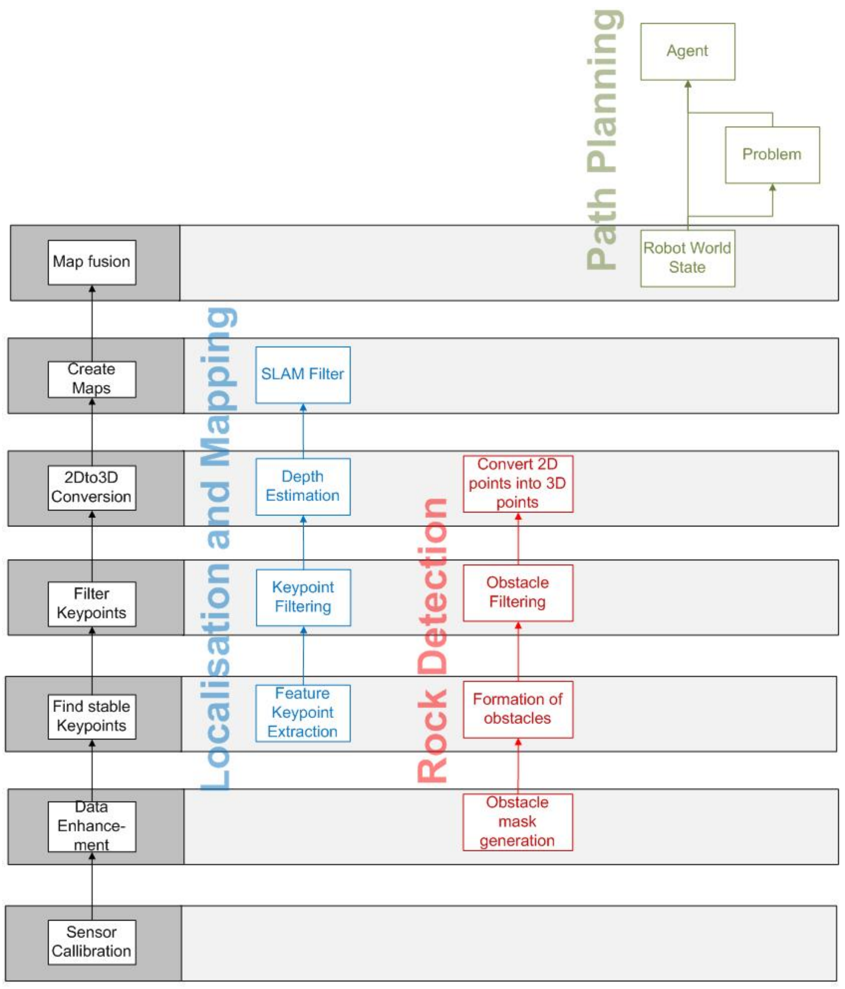

Figure 1. *A data pipeline process of rover navigation with relations to three navigation functions*

strategies are defined with their individual specifications and requirements to represent three different navigation functions and also to encapsulate relevant algorithms in a changing context. To standardise such concrete strategies, the input and output of data products are described as system states. As a result, the pattern definition is realised according to the requirements of independently processed functions. The overall process produce in each step a data product within the pipeline and will generate an environment model of the rover.

This modular software design is realized using ROS which is an open source midware or meta-OS for robots (http://ros.sourceforge.net, [5, 15]). Interfaces for data transitions are asynchronously communicated by publishing existent data in particular data flows, following data architecture defined in ROS. The data products are generated by interfaces of each transition step defined for autonomous functions that are implemented according to the general policy. The implementation of these interfaces standardises the data product transitions and facilitates the organisation of information in a pipelined process. Each concrete policy is also implemented as a ROS node. A ROS node is defined by a base class definition in which the general policy is implemented. Finally, the pattern context is placed in a ROS node that makes the publishing of available information through a pipeline process. The overall software is conformed by independently executed ROS nodes that generate

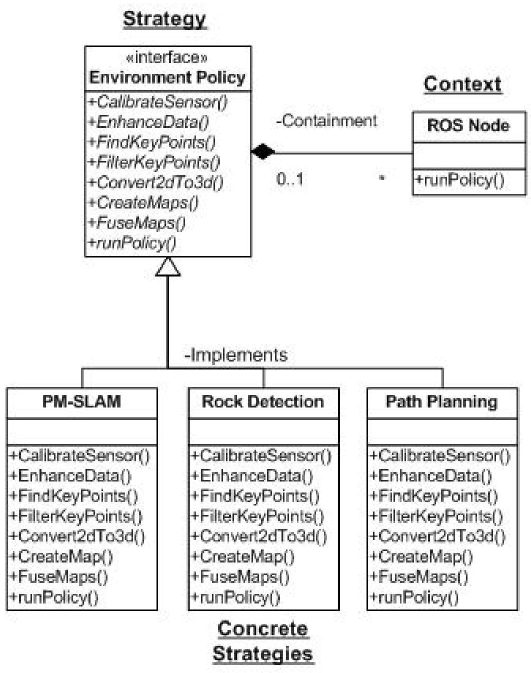

FIGURE 2. *System overview of general strategy (upper left block), concrete strategies (lower blocks) and context (upper right block).*

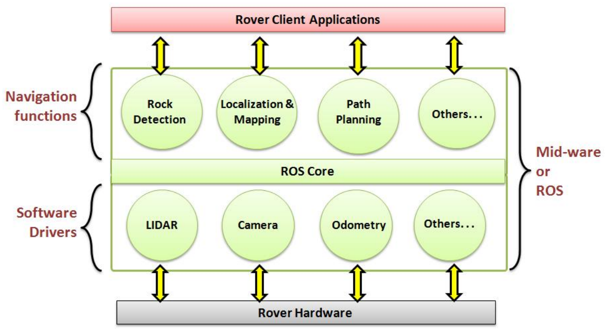

FIGURE 3. *Software system overview of SMART*

generic data products as topics. However, the content of each topic is specified by each navigation function.

4 System implementation & testing

The proposed software design is implemented within the Surrey Rover Autonomous Software and Hardware Testbed (SMART). Figure 3 shows its software overview. The SMART implements major navigation functions with atuation and sensory functions all as software modules on-board its rover platform.

4.1 Localisation and Mapping

Within SMART, this module uses a SLAM (Simultaneous Localisation and Mapping) approach that provides a reactive calculation of the rover position and heading while building a sparse map of visual features using an information filter [18]. The module uses SURF features [8] as visual key points and translates them to 3D landmarks once they are filtered from noise and it obtains values of their uncertainty. Figure 4 illustrates how the SURF feature points (in green) are extracted from the images producing 3D landmarks (in red) through matching and triangulation. Figure 5 presents an example output representing a sparse 3D map of the environment and the corresponding rover trajectory.

4.2 Rock Detection

This robot function detects rocks within the rover's field of view by a supervised learning process using image data (see details in [16, 13]). Rocks are detected by finding changes in intensity and size from input examples and learning a binary classification for the pixels. A linear Support Vector Machine (SVM) classifier is used to locate rock features in image data by running a binary classification of pixels. A set of rock features is subsequently interconnected through a standard breadth-first search algorithm. This process offers a mask to the input image with rock candidates. Each individual group of candidate pixels from the rock masks is checked for correspondence to a rock. This procedure is realised through heuristic methods that operate on the shape and the texture of the candidate rock's image region. The generated data product is a list of rocks from a candidate rock mask. Finally, the centre point of each detected rock is transformed from image-view $2D_R$ coordinates to $3D_W$ world coordinate system. Since the depth information is not available from the 2D image, it is assumed that a set of observable stable 3D key points and their locations in the image is given (i.e. by invoking a *Depth Estimation* component of the Localisation and Mapping module).

The output of rock detection function is presented in 6.

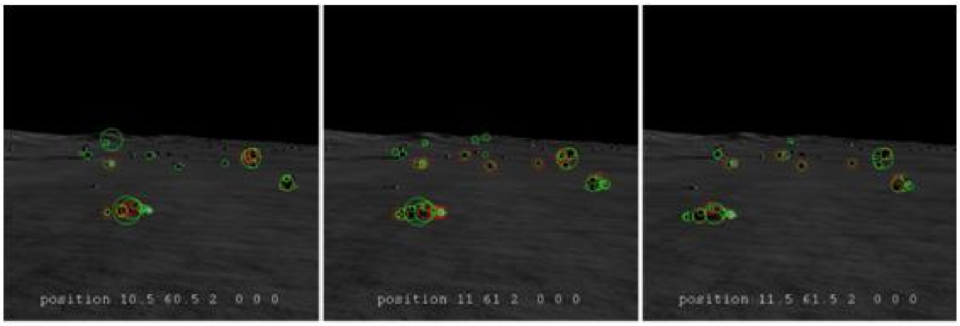

Figure 4. *Image sequence acquired from the PANGU simulator [12] with overlaid SURF key points (green) and triangulated 3D landmarks (red).*

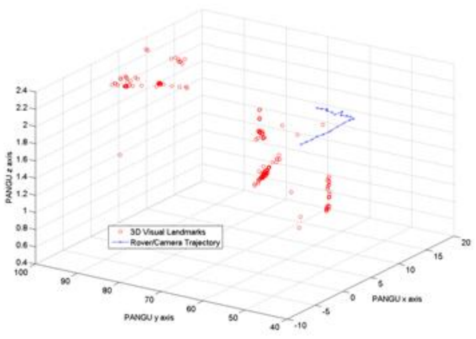

Figure 5. *Example SLAM output: sparse map of visual landmarks and the estimated rover trajectory.*

4.3 Path Planning

This module is a route generator to reach a pre-located goal in a simulated planetary environment. The route is obtained as a result of calculating the minimal cost between distances from available waypoints with a graph search algorithm. In the scope of this research, the waypoints are considered as coordinates that identify positions in robot's physical space during terrestrial navigation. In specific, this function aims to produce a route as a collection of waypoints that guide the rover to its goal by avoiding rocks and obstacles

The algorithm's objective is to estimate the robot's control actions by defining a world state according to its wheel velocities, position and available space information found in maps. This information is asynchronously provided as input by the robot functions. Furthermore, we make use of an agent to collect the information and

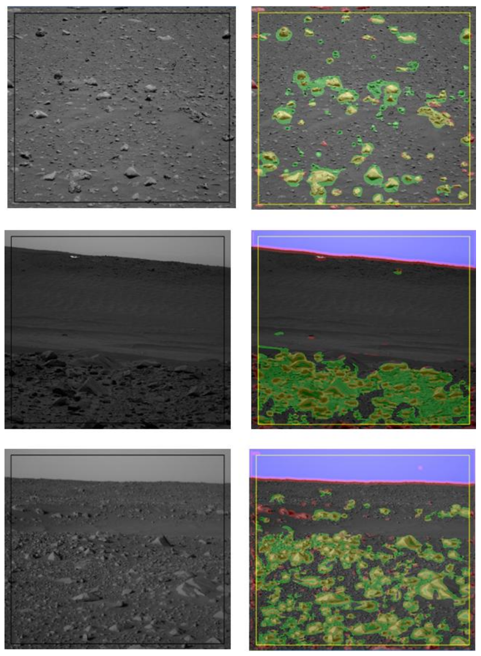

Figure 6. *Tested images showing detected rocks (in green), missed rocks (in red) and sky (in blue).*

find the actions to follow the route and produce robot mobility.

The methods' performance is measured by properties that characterise its efficiency by a) Priority buffer, b) Amount of visited cells during route generation, c) Endurance of the algorithm in total cycles for replanning and d) Route length by measuring the total steps as route sections to cover. The experiment setting up is defined

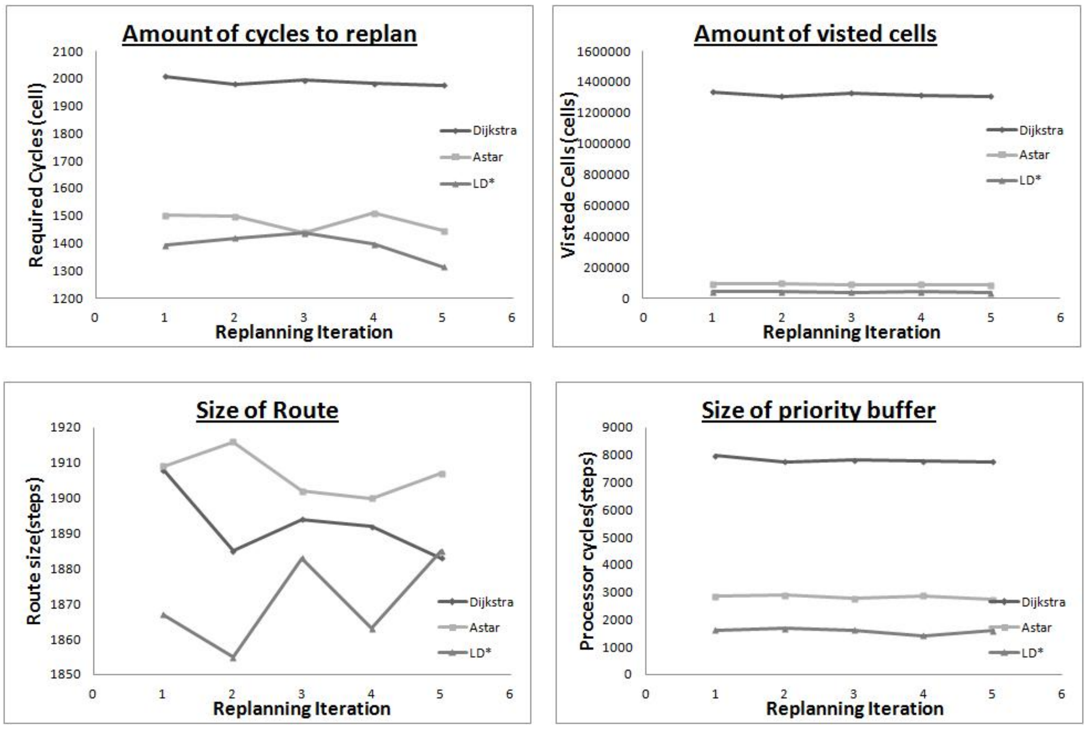

Figure 7. *Graphs for metrics of path planning: Priority buffer, amount of visited cells, amount of cycles after each re-planning total requires steps.*

by incrementally discovered maps and rock-based cost maps for generating a search space comparison among the a AStar (A*), Dijkstra and light Dstar (LD*) methods.

The methods functionality is graphically described in Figure 8. On the one hand, the functionality of Dijkstra methods shows an initialisation that requires plenty of robot's search space. Moreover, AStar (A*) and light Dstar (LD*) methods maintain a lower proportion of use of resources for path planning. On the other hand, the incremental search characteristics of Dstar (LD*) method accumulate search space size over time. The methods performance comparison is presented in Figure 7

As a result, we identify the importance of initialising the search space with as much cells as possible and to make use of re-planning with a reduce search space. In specific, we made use of A* for initial propagation and keeping cost of the shortest path (G-value) from the explored cells based on a map. Moreover, experimental testing shows that by using Light D* with a re-planning based on the potential field constructed by new and previously updated cells.

5 Conclusions & future work

The proposed design provides a generic framework to navigation software making use of data pipeline to generate environment models from independently processed navigation functions. On the one hand, software modules are built using ROS protocols such as defining concrete policies for each navigation function. ROS is the middle-ware to encapsulate different algorithms for the functions. On the other hand, the functions are also part of a pipeline process to standardise sensor data into data products. The overall performance during rover navigation describes the robot state with a collection of performance variables updated through sensory data. The software design was implemented within SMART and test results were shown based on both simulated and real data.

In the future, more integrated tests should be performed to demonstrate the system-level performance

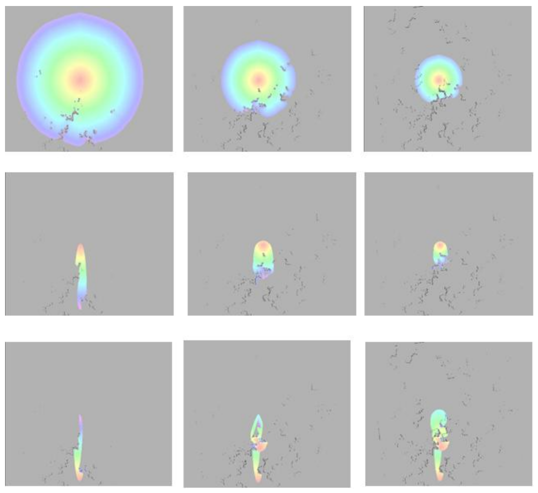

Figure 8. *Search space for different path planning algorithms: First row is Dijkstra; Second row is A* and the last line is light DStar (LD*)*

including integrated data products. The current data pipeline design only covers data from exteroceptive sensors which can be extended to include proprioceptive sensor data.

References

[1] A. Alexandrescu, S. Meyers, and J. Vlissides. *Modern C++ Design: Applied Generic and Design Patterns (C++ in Depth)*. Addison Wesley, 2001.

[2] B. Bauml and G. Hirzinger. When hard realtime matters: Software for complex mechatronic systems. In *Journal Robotics and Autonomous Systems*, volume 56, Issue, 2008.

[3] P. T. Furgale and T. D. Barfoot. Stereo mapping and localization for long-range path following on rough terrain. In *In Proceedings of the International Conference on Robotics and Automation (ICRA)*, Anchorage, Alaska, USA, 3-8 May 2010.

[4] P. T. Furgale and T. D. Barfoot. Visual teach and repeat for long-range rover autonomy. *Journal of Field Robotics, Special Issue: Visual Mapping and Navigation Outdoors*, 27(5):534–560, September/October 2010.

[5] W. Garage. navfn. Licensed under Creative Commons Attribution 3.0., Last edited 2010-08-19 2010. http://www.ros.org/wiki/pluginlib.

[6] D. Hall and J. Llinas. Introduction to multisensor data fusion. In *Proc. of IEEE*, volume 85, San Francisco, California, Jan 1997.

[7] D. Helmick, A. Angelova, and L. Matthies. Terrain adaptive navigation for planetary rovers. *Journal of Field Robotics*, 26(4):391–410, April 2009.

[8] B. Herbert, T. Tinne, and V. G. Luc. Surf: Speeded up robust features. In A. Leonardis, H. Bischof, and A. Pinz, editors, *Computer Vision ECCV 2006*, volume 3951 of *Lecture Notes in Computer Science*, pages 404–417. Springer Berlin / Heidelberg, 2006.

[9] E. Krotkov and R. Hoffman. Terrain mapping for a walking planetary rover. *IEEE Transactions on Robotics and Automation*, 10(6):728 – 739, December 1994.

[10] R. Li, K. Di, J. Wang, and S. He. Rock modeling and matching for autonomous long-range mars rover localization. *Journal of Field Robotics - Special Issue on Space Robotics*, 24(3), March 2007.

[11] L. Matthies, E. Gat, R. Harrison, B. Wilcox, R. Volpe, and T. Litwin. Mars microrover navigation: Performance evaluation and enhancement. *Autonomous Robots*, 2:433–440, 1995.

[12] S. Parkes, M. Dunstan, I. Martin, P. Mendham, and S. Mancuso. Planet surface simulation for testing vision-based autonomous planetary landers. In *57th International Astronautical Congress*, October 2006.

[13] M. Pham, Y. Gao, V. Hoang, and T. Cham. Fast polygonal integration and its application in extending haar-like features to improve object detection. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, San Francisco, California, Jun 2010.

[14] M. Pivtoraiko, I. Nesnas, and A. Kelly. Autonomous robot navigation using advanced motion primitives. In *IEEE Aerospace conference*, pages 1 – 7, Big Sky, MT, 7-14 March 2009.

[15] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[16] D. Sancho-Pradel and Y. Gao. A survey on terrain assessment techniques for autonomous operation of planetary robots. *Journal of British Interplanetary Society*, 2010.

[17] P. S. Schenker, T. L. Huntsberger, P. Pirjanian, E. T. Baumgartner, and E. Tunstel. Planetary rover developments supporting mars exploration, sample return and future human-robotic colonization. *Autonomous Robots*, 14(2-3):103–126, 2003.

[18] K. Shala and Y. Gao. Comparative analysis of localisation and mapping techniques for planetary rovers. In *Proceedings of The 10th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*, 2010.

[19] D. Wooden. A guide to vision-based map building. In *IEEE Robotics and Automation Magazine*, volume 13 Issue:2, pages 94 – 98, June 2006.

[20] M. Woods, A. Shaw, D. Barnes, D. Price, and D. Long. Autonomous science for an exomars rover–like mission. *Journal of Field Robotics. Special Issue: Special Issue on Space Robotics, Part II.*, 26. Issue 4:358–390, April 2009.