UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA

DEPARTAMENTO DE INGENIERÍA DEL SOFTWARE E INTELIGENCIA ARTIFICIAL

TESIS DOCTORAL

Stereo vision-based perception, path planning and navigation strategies for autonomous robotic exploration

Percepción basada en visión estereoscópica, planificación de trayectorias y estrategias de navegación para exploración robótica autónoma

MEMORIA PARA OPTAR AL GRADO DE DOCTOR
PRESENTADA POR

Raúl Correal Tezanos

Directores

Gonzalo Pajares Martinsanz
José Jaime Ruz Ortiz

Madrid, 2015

UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA
Departamento de Ingeniería del Software e Inteligencia Artificial

PERCEPCIÓN BASADA EN VISIÓN ESTEREOSCÓPICA, PLANIFICACIÓN DE TRAYECTORIAS Y ESTRATEGIAS DE NAVEGACIÓN PARA EXPLORACIÓN ROBÓTICA AUTÓNOMA

Memoria para optar al grado de doctor, presentada por

Raúl Correal Tezanos

Dirigida por

Gonzalo Pajares Martinsanz
Jose Jaime Ruz Ortiz

Madrid, 2015

UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA

Departamento de Ingeniería del Software e Inteligencia Artificial

PhD thesis

STEREO VISION-BASED PERCEPTION, PATH PLANNING AND NAVIGATION STRATEGIES FOR AUTONOMOUS ROBOTIC EXPLORATION

Dissertation submitted to obtain the Ph. D. Degree by:

D. Raúl Correal Tezanos

Supervisors:

Dr. D. Gonzalo Pajares Martinsanz
Dr. D. Jose Jaime Ruz Ortiz

Madrid (Spain), 2015

ABSTRACT

This thesis is concerned with the development of a visual-based autonomous navigation strategy for robotic exploration of planetary surfaces. The collection of subsystems, modules and software presented in this work have been developed from the ground up, as most of the existing tools in this domain are proprietary of national space agencies, usually not accessible to the research community. A multi-layer modular software architecture with several hierarchical levels has been designed to host the series of algorithms that implement the autonomous navigation strategy and ensure software portability, reusability and hardware independence. This work also includes the design and implementation of a framework aimed to support the development of the navigation strategies. It is partially based in open source tools and component of the self at the reach of any researcher/institution, with adaptations and extensions. The framework provides 3D simulation capabilities and models of robotic vehicles, including mechanical design and sensors, and operational environments, emulating planetary surfaces like Mars, for analysis and validation of the developed navigation approaches and strategies at the functional level. The framework also includes debugging and monitoring capabilities.

The present thesis is composed of two main parts: in the first part it is addressed the design and development of rover's high-level autonomy capabilities, focusing in autonomous navigation, supported by a purposely designed simulation and monitoring framework. A set of field experiments, with a physical robot and real hardware, have been carried out, detailing results, algorithms' processing time and the overall system's behavior and performance. As a result, the perception system has been identified as a crucial component within the navigation strategy and, therefore, the main focus of potential system's optimizations and enhancements. As a consequence, in the second part of this work, the problem of stereo matching and features' correspondence between pair of images and high-quality 3D reconstruction of unstructured natural, rough environments is addressed. A number of matching algorithms, image processes and filters have been analyzed. It is generally assumed the intensities of corresponding points in two images of a stereo pair are equal. However, it has been verified that this assumption is often false, even though they both images are acquired from a vision system composed of two identical cameras. Consequently, an automatic expert system is proposed for automatic intensity correction in stereo pairs of images and 3D terrain reconstruction based on the novel application of image processes not applied so far to the stereo vision field. Such processes are homomorphic filtering and histogram matching. They are intended for correcting intensities of the stereo pair coordinately, adjusting one image as a function of the other. Additionally, results have been further enhanced by applying a purposely designed process directed by clusters based on the principle of spatial continuity to eliminate false positives and erroneous correspondences. A study of the effects of applying such filters to the input images, in a pre-matching and post-matching steps correspondingly, has been carried out and the performance verified favorably. The application of these processes has allowed obtaining a higher number of valid correspondences in contrast to performing the stereo matching process without applying them, achieving significant improvements in the disparity maps and, therefore, in the overall perception and 3D reconstruction processes.

Keywords: robotics, rovers, planetary exploration, mobile robots, simulation, software architecture, path planning, stereo-vision, matching, correspondence, image processing.

RESUMEN

En esta tesis se trata el desarrollo de una estrategia de navegación autónoma basada en visión artificial para exploración robótica autónoma de superficies planetarias. Se han desarrollado una serie de subsistemas, módulos y software específicos para la investigación desarrollada en este trabajo, ya que la mayoría de las herramientas existentes para este dominio son propiedad de agencias espaciales nacionales, no accesibles a la comunidad científica. Se ha diseñado una arquitectura software modular multi-capa con varios niveles jerárquicos para albergar el conjunto de algoritmos que implementan la estrategia de navegación autónoma y garantizar la portabilidad del software, su reutilización e independencia del hardware. Se incluye también el diseño de un entorno de trabajo destinado a dar soporte al desarrollo de las estrategias de navegación. Éste se basa parcialmente en herramientas de código abierto al alcance de cualquier investigador o institución, con las necesarias adaptaciones y extensiones, e incluye capacidades de simulación 3D, modelos de vehículos robóticos, sensores, y entornos operacionales, emulando superficies planetarias como Marte, para el análisis y validación a nivel funcional de las estrategias de navegación desarrolladas. Este entorno también ofrece capacidades de depuración y monitorización.

La presente tesis se compone de dos partes principales. En la primera se aborda el diseño y desarrollo de las capacidades de autonomía de alto nivel de un rover, centrándose en la navegación autónoma, con el soporte de las capacidades de simulación y monitorización del entorno de trabajo previo. Se han llevado a cabo un conjunto de experimentos de campo, con un robot y hardware real, detallándose resultados, tiempo de procesamiento de algoritmos, así como el comportamiento y rendimiento del sistema en general. Como resultado, se ha identificado al sistema de percepción como un componente crucial dentro de la estrategia de navegación y, por tanto, el foco principal de potenciales optimizaciones y mejoras del sistema. Como consecuencia, en la segunda parte de este trabajo, se afronta el problema de la correspondencia en imágenes estéreo y reconstrucción 3D de entornos naturales no estructurados. Se han analizado una serie de algoritmos de correspondencia, procesos de imagen y filtros. Generalmente se asume que las intensidades de puntos correspondientes en imágenes del mismo par estéreo es la misma. Sin embargo, se ha comprobado que esta suposición es a menudo falsa, a pesar de que ambas se adquieren con un sistema de visión compuesto de dos cámaras idénticas. En consecuencia, se propone un sistema experto para la corrección automática de intensidades en pares de imágenes estéreo y reconstrucción 3D del entorno basado en procesos de imagen no aplicados hasta ahora en el campo de la visión estéreo. Éstos son el filtrado homomórfico y la correspondencia de histogramas, que han sido diseñados para corregir intensidades coordinadamente, ajustando una imagen en función de la otra. Los resultados se han podido optimizar adicionalmente gracias al diseño de un proceso de agrupación basado en el principio de continuidad espacial para eliminar falsos positivos y correspondencias erróneas. Se han estudiado los efectos de la aplicación de dichos filtros, en etapas previas y posteriores al proceso de correspondencia, con eficiencia verificada favorablemente. Su aplicación ha permitido la obtención de un mayor número de correspondencias válidas en comparación con los resultados obtenidos sin la aplicación de los mismos, consiguiendo mejoras significativas en los mapas de disparidad y, por lo tanto, en los procesos globales de percepción y reconstrucción 3D.

Palabras clave: robótica, rovers, exploración planetaria, robots móviles, simulación, arquitectura software, planificación de trayectorias, visión estéreo, correspondencia, procesado de imágenes.

CONTENTS

ACRONYMS LIST

2D – 2 dimensions

3D – 3 dimensions

AI – Artificial Intelligence

ANW - Autonomous Navigation Workshop

API - Application Programming Interface

APXS - Alpha Proton X-Ray Spectrometer

ASTRA - Advanced Space Technologies for Robotics and Automation

BISMARC - Biologically Inspired System for Map-based Autonomous Rover Control

BM - Block-Matching

BOCM – Boletín Oficial de la Comunidad de Madrid

BOE – Boletín Oficial del Estado

CAMPOUT – Control Architecture for Multirobot Outpost

CCD – Charge-Coupled Device

CDF – Cumulative Distribution Function

CIB - Constant Image Brightness

CLARAty - Coupled Layer Architecture for Robotic Autonomy

CMU - Carnegie Mellon University

CNES - Centre National d'Etudes Spatiales

CNRS - Centre National de la Recherche Scientifique

CNSA - China National Space Administration

COTS - Components Of The Self

CPU – Central Processing Unit

DARPA - Defense Advanced Research Projects Agency

DASIA	international conference on DAta Systems In Aerospace
DEM	Digital Elevation Model
DRC	DARPA Research Challenge
DTE	Direct-to-Earth
EDRES	Environnement de Développement pour la Robotique d'Exploration Spatiale
ESA	European Space Agency
ESTEC	European Space research and TEchnology Centre
FC	Forward Control
FOV	Field Of View
FPGA	Field Programmable Gate Arrays
FRM	Functional Reference Model
GESTALT	Grid-based Estimation of Surface Traversability Applied to Local Terrain
GFE	Government Funded Equipment
GPS	Global Positioning System
GUIDE	Graphical User Interface Development Environment
HAL	Hardware Abstraction Layer
HSV	Hue Saturation Value
ICE	Internet Communication Engine
IDEA	Intelligent Distributed Execution Architecture
IDL	Interface Definition Language
IMU	Inertial Measurement Unit
ISRO	Indian Space Research Organisation
JAXA	Japanese Space Agency
JPL	Jet Propulsion Laboratory
MAM	Mobility Avionics Module
MER	Mars Exploration Rovers

MIPS - Million of Instructions Per Second

MOBC - Multi-Objective Behavior Control

MRPT - Mobile Robot Programming Toolkit

MSL - Mars Sciences Laboratory

NASA – National Aeronautics and Space Administration

NCC - Normalized Cross Correlation

NF - Nominal Feedback

NNF - Non-nominal Feedback

OBSW – OnBoard SoftWare

ORCCAD - Open Robot Controller Computer Aided Design

OS – Operating System

OSRF - Open Source Robotics Foundation

PC – Personal Computer

PROM - Programmable Read-Only Memory

RAM - Random Access Memory

RAT - Rock Abrasion Tool

RGB – Red Green Blue

ROAMS - Rover Analysis, Modeling and Simulation

ROS - Robot Operating System

RP - Robot-Procedures

RT - Robot-Task

SAD - Sum of Absolute Differences

SGBM - Semi-Global Block Matching

SIFT - Scale-Invariant Feature Transform

SIFT - Scale-Invariant Feature Transform

SLAM - Simultaneous Localization and Mapping

SRI	Stanford Research International
SSD	Sum of Squared Differences
SW	Software
TES	Thermal Emission Spectrometer

FIGURES LIST

TABLES LIST

Chapter 1

INTRODUCTION

In this chapter the antecedents of the research lines presented in this thesis and the problems identified are introduced and outlined. The motivation leading to the work described in this document as well as the goals expected to achieve are listed. The contributions made by the present thesis are stated as well as the organization and structure of the document.

1.1 Antecedents and Problems Identification

Robotic surface planetary exploration is a challenging endeavor, with critical safety requirements and severe communication constraints. A certain level of local autonomy for onboard robots is an essential feature, so that they can make their own decisions independently of ground control, minimizing the requirements of continuous support from human operators on Earth, reducing operational costs and maximizing the scientific return of the mission. Operational costs are referred to the costs derived to the mission maintenance and support once the aircraft has been launched and the robot is operating in the target planet; therefore, these costs can be reduced by increasing the level of rover's onboard autonomy. As for the scientific return of the mission, an increased level of onboard autonomy allows the robot to make its own decision on the field, reducing the dependency on human operators' decisions; it speeds up the navigation process and ultimately increases the scientific return of the mission as the rover is able to travel longer distances within the mission lifespan, minimizing the time it spends waiting for Earth resolutions and performing a higher number of experiments to collect scientific data.

There is a growing demand for autonomy capabilities for future planetary missions. Next missions are expected to be highly autonomous, where scientists on Earth will designate target destinations on the basis of images acquired and downloaded from the rover. Autonomous navigation is one of the most crucial and yet risky aspects of these operations. The vehicle has to be able to travel from one location to another to take measures, samples and perform scientific experiments. If the vehicle drives into a too steep area, runs over a rock, a hollow or a rise or a soft sand area, to name some potential hazardous situations, it may tip over, run aground, get stuck or lose traction, and that may put the whole mission at risk, as nobody can get there to help the rover get out and keep going. To achieve this, the robot must perform a series of tasks; an exploratory navigation strategy consists of an iterative perception-mapping-path planning-navigation cycle. The rover must perceive the environment making use of the appropriate sensors; create an internal representation of the robots' surroundings; and compute safe and suitable trajectories to get to the location it has been commanded.

However, one of the firsts and main difficulties researchers have to face when working in this domain is existing frameworks and tools to support research are usually proprietary to space agencies, and out of reach of most researchers. Moreover, it is indispensable to make extensive use of simulation that allow the creation of models to replicate the vehicle, sensors, terrain and operational conditions and tools that support the development of the necessary algorithms to implement a visual-based autonomous navigation approach for robotic space exploration and allow the analysis of algorithms' performance and functional validation of approaches, autonomy strategies and data monitoring. Therefore, this obstacle must be overcome somehow. In this case, the strategy followed has been designed a framework, based on the integration and adaptation of existing tools, purposely focused in this domain to support the developments and research works, and design and develop an autonomous navigation strategy.

Within autonomous navigation, perception is one of the most crucial yet challenging tasks. Stereo cameras are the preferred device for 3D environment perception in many outdoor applications. Stereoscopic vision is a mechanism to obtain depth or range data based on images. It consists of two cameras separated by a given distance so that two differing views of the same scene are obtained, similar to human binocular vision. This pair of images is the input to the matching process. By comparing both images, relative depth information is obtained, in the form of disparities, which are inversely proportional to distance to objects. The difference in position of a set of features or pixels from one image relative to the other is then computed, usually along the horizontal axis. Depth can be established by triangulation of the disparities obtained from the matching process, provided that the position of centers of projection, the focal length and the orientation of the optical axis are known. It can then be reprojected to 3D space using a perspective transformation, obtaining a set of world coordinates.

Stereo matching is a heavily researched area with a prolix published literature and a broad spectrum of heterogeneous algorithms available in diverse programming languages. However, there is no single approach that could be considered as the best one, able to solve every possible problem. Usually, each technique is suitable for a set of conditions but not for others. Therefore, as in every area of research, the first step is to analyze the state of the art to verify if there is already any solution or technique fulfilling the concrete needs and to what extent. This is, in fact, one of the main difficulties researches in the stereo vision area –as in many other areas- face, as this task is not trivial at all, given the broad range of available resources and literature, so a considerable effort has to be made to analyze it in order to determine what strategy may fit each concrete set of requirements. This difficulty is due mainly to the heterogeneous nature of the available algorithms and techniques. This heterogeneity and the lack of standardization makes the state of the art analysis, algorithms' evaluation and comparison a hard, complicated and time consuming task.

Once a technique or algorithm has been evaluated as promising to be tested for robotic navigation in natural terrains, new problems have to be usually faced when applying such techniques to real settings in contrast to its application to laboratory and controlled environments, as many published works are based on. The problem of correspondence in stereoscopic systems stems from the fact that images from cameras, although similar, show different intensity levels for the same physical entity in the 3D scene. The main reason for this feature lies in the different response from the camera sensors to the signal light from the scene and also from the different mapping of the scene over each image due to the different relative points of view of each camera. That makes necessary to devote a major research effort to correct these deviations typical of any stereo system. This problem is not yet satisfactorily solved, particularly in unstructured and uncontrolled environments. That is the main reason why literature about this topic is so broad.

This fact allows deducing that the problem concerning the low efficiency of some algorithms comes from their application to real images, which requires a global solution to this general problem. This justifies the need for a pretreatment of the images. The aim is to achieve the maximum similarity in the spectral levels of the stereo pair images, at the pixel level, which is exactly what happens in the case of the simulated images. Therefore, methods capable of correcting the differences in both images, from the radiometric point of view, are necessary.

Finally, the main purpose of the perception phase and the use of stereo vision, is to perform a 3D terrain reconstruction that serves the robot to create an internal representation of its environment and plan suitable paths and trajectories to effectively navigate from one location to another. That requires a high-quality reconstruction of the surface and the development of a system capable of performing such a task, similar to that a human expert would do in a similar situation. Therefore, the necessary knowledge has to be mapped into the system following the logical strategy the expert human applies.

The work presented in this thesis is framed in the design and development of a visual-based robotic autonomous navigation for space exploration. Although focused in this domain, it can be easily ported and equally applied to many terrestrial applications. Any application where a robot has to autonomously traverse a natural, rough terrain with not a priori information, so that the environment is unknown and it has to be explored and discovered and the robot goes making use of stereo cameras for 3D perception, will face analogous difficulties and conditions. Some examples of similar terrestrial applications are autonomous robotic patrolling in rural environments, such as fire detection in forests, or border or agrarian facilities monitoring and surveillance. Therefore, future research works and projects focused in these areas of application can also benefit from the work presented in this thesis.

1.2 Motivation and objectives

1.2.1 Motivation

The research work objective of this thesis is part of a collaboration between the Universidad Complutense of Madrid and the company TCP and Sistemas e Ingeniería in the area of planetary exploration rovers' autonomous navigation. Both entities were involved in two projects, entitled: *AUTOROVER: estudio de autonomía basada en imágenes para rovers de exploración planetaria* and *Visión estereoscópica para Auto-rover: estudio de autonomía basada en imágenes*. The former comes from the participation of the mentioned company in the public call 2259/2007 (BOCM 272 of 15/11/2007) to promote innovation of the Community of Madrid in the aerospace sector with funding from the European Regional Foundation (Reference 04-AEC0800-000035/2008). The latter corresponds to an extension of the previous program within the National Research Program in the Aerospace sector after the order PRE/998/2008 (BOE 11/04/2008) Ministry of the Presidency (reference SAE-20081093). Both projects are framed into the area of robotic autonomous navigation on the Mars surface, which main objective is the design and development of a vision-based system for autonomous navigation of planetary exploration rovers.

The research works of the mentioned projects have been carried out in the computer science school at the Complutense University of Madrid under the direction of the supervisors of this thesis.

From the point of view of its industrial utility, the developed work stems from the need raised by the European Space Agency for its next robotic mission to Mars called ExoMars, where a rover will travel across the Martian surface to search for signs of life. The primary objective is to land the rover at a site with high potential for finding well-preserved organic material, particularly from the very early history of the planet. It will collect samples with a drill and analyze them with next-generation instruments, establishing the physical and chemical properties of these samples. This future rover is intended to be highly autonomous. The rover will build a 3D model of its environment using stereo images and analyze it in order to establish scientific targets. It will then navigate autonomously to the selected target to make use its robotic arm and instruments to collect and analyze data. The intentions are to decrease by three or more the time NASA's last rovers took to plan and schedule its actions to get the samples, what would increase dramatically the scientific return of the mission.

The agency is in need of tender multiple contracts to the European industry for the design and development of the numerous systems that integrate a mission of such a complexity and these features. The works presented in this thesis are focused on developing the autonomous capabilities of the robotic vehicles, and more specifically on the navigation system, with particular emphasis on the perceptual subsystem, based on stereoscopic vision.

Moreover, from the technological point of view, motivation is prompted by the fact of trying to improve the results obtained by the stereo matching algorithms, considering the problems of its application to real settings and images, which involves some complexities and issues not taken into account in many related published works, performed in controlled environments or limited to synthetically generated images.

1.2.2 Objectives

In view of the considerations expressed in the preceding paragraphs, the following research objectives are proposed:

1. Conduct a study and analysis of the autonomy capabilities for planetary exploration vehicles, with special emphasis in autonomous navigation. Understand the current state of the art in this domain and in the available technologies and tools related to the development of these research topics.

2. Design and development of a framework that supports the development of autonomy capabilities, navigation strategies, algorithms and its functional validation, given the lack

of availability and accessibility to equivalent environments, which has typically been under the purview of national space agencies.

3. Creation of the necessary simulation models to replicate the vehicle, sensors, terrain and operational conditions, including aspects such as vehicle kinematics and dynamics, contact forces, gravity, or friction to analyze the interaction of the rover with the environment with the appropriate level of fidelity. The goal is to emulate the whole operation process of the rover on a planet surface.

4. Design and development of a visual-based autonomous navigation system for robotic space exploration so that a vehicle is able to travel from one location to another to take measures, samples and perform scientific experiments. The rover's onboard autonomy entails the set of algorithms that shall be embedded on the robot to provide this with the necessary intelligence and capabilities to make its own decision, dealing with unknown environments and unexpected situations, minimizing the dependency on ground control operators.

5. Design of a modular and scalable software architecture to structure and organize the previously developed set of algorithms so that each subsystem is independent from one another and are able to interact with each other to achieve the desired high-level autonomy capabilities.

6. Conduct a set of experiments, both using the simulation capabilities of the framework and carry out a field testing campaign with a physical robot and real hardware, in order to analyze and validate the autonomy capabilities of the developed autonomous navigation system and the usefulness of the framework to support such developments and testing.

7. Analyze the results obtained from the experiments and the performance of the navigation system.

8. Validate the designed navigation strategy, software architecture and developed algorithms.

9. Identify the potential areas for improvement and develop the necessary tools to assist in the identification of such areas. Make these tools available to the research community whenever possible.

10. Develop the methods and techniques previously identified as areas for improvement within the autonomous navigation strategy; develop the necessary tools to assist in the creation of this methods and technologies.

11. Identify future lines of research.

12. Dissemination of the obtained results, knowledge, findings and developments through scientific publications or other mechanisms like software release whenever possible.

1.3 Contributions

From the proposed objectives and considering the previously highlighted aspects, the aim is to solve the considered problems and provide the scientific community with a set of solution strategies and tools that can also be spread to other problems of similar nature. The contributions made by this research are summarized in the following points:

1. A study and analysis of the autonomy capabilities for planetary exploration vehicles have been conducted, with special emphasis in autonomous navigation. The current state of the art in this domain and in the available technologies and tools related to the development of these research topics has been also analyzed. This work is described in chapter 2 of this thesis.

2. A framework that supports the development of the autonomy algorithms, including the navigation strategies and its functional validation, has been designed and developed, supported by some third party tools and packages. This work has been motivated by the lack of availability and accessibility to equivalent environments, which has typically been under the purview of national space agencies. This work is described in chapter 3 of this thesis and in Correal and Pajares (2011a) and Correal et al. (2014b).

3. Simulation models to replicate a robotic space exploration vehicle, sensors, terrain and operational conditions, including aspects such as vehicle kinematics and dynamics, contact forces, gravity, or friction have been created, emulating the whole operation process of the rover on a planet surface. This work is described in chapter 3 of this thesis and in Correal and Pajares (2011a) and Correal et al. (2014b).

4. A visual-based autonomous navigation system for robotic space exploration has been designed and developed from the ground up, supported by some third party libraries. It allows a robotic vehicle to travel autonomously from one location to another, taking measures, samples and performing scientific experiments. The rover's onboard autonomy entails the set of algorithms that shall be embedded on the robot to provide this with the necessary intelligence and capabilities to make its own decision, dealing with unknown environments and unexpected situations, minimizing the dependency on ground control operators. It includes de design and development of several exploratory navigation strategies to autonomously compute trajectories in planetary, distant environments. This work is described in chapter 4 of this thesis, based on Correal and Pajares (2011b) and Correal et al. (2014b).

5. A modular, scalable and layer-based software architecture has been designed. It is crucial to structure and organize the set of algorithms that comprises the rover's onboard intelligence, so that each subsystem is independent from each other and are able to interact with each other to achieve the desired high-level autonomy capabilities. This work is described in chapter 4 of this thesis and in Correal and Pajares (2011b) and Correal et al. (2014b).

6. A set of experiments have been conducted, using the simulation capabilities of the framework and a real robot and devices, validating both the autonomy capabilities of the navigation system and the role of the framework to support such developments and testing. This work is described in chapter 4 of this thesis and in Correal and Pajares (2011b) and Correal et al. (2014b).

7. Some areas for improvement have been identified, mainly related to the perception subsystem. Several methods and techniques have been developed to improve these

results. One of such techniques is *histogram matching*. It normalizes the images of the stereo pair adjusting the color distribution of one image with respect to the other. This technique enhances the contrast of images using cumulative distribution functions to transform the intensity values of an image, or the values in the colormap of an indexed image, so that the histogram of the output image approximately matches the histogram of the other image, approximating both illumination components. This process, when applied to the input images previous to the stereo matching process led to achieve an important improvement in the results within the perception phase. This work is described in chapter 6 of this thesis and in Correal et al. (2014a).

8. Another significant contribution to the improvement of the perception phase within a robotic explorer has been the application to the input images of a technique called *homomorphic filtering*; which eliminates the illumination component of an image while preserving reflectance. When applied to the input images prior to the stereo matching process it led to achieve an important improvement of the results obtained from the perception subsystem. This work is described in chapter 6 of this thesis and in Correal et al. (2013).

9. A third technique, designed, developed and integrated within the autonomous navigation system, led to an improvement of the robot's perception phase is *clustering filter*. This process, based on the philosophy of clusters, is applied after the computation of disparities to eliminate correspondence errors, taking out false positives. It consists in grouping pixels in connected components based on the principle of spatial continuity. The result of the process is a list of clusters that can be analyzed to be kept or filtered out. This process, when applied to the map of disparities computed after the stereo matching process led to achieve an important improvement in the results within the perception phase. This work is described in chapter 6 of this thesis and in Correal et al. (2013).

10. An open-source project has been initiated as part of this thesis, consisting in a testbed aim to assists the stereo vision algorithms state of the art's analysis and aid researches to analyze, evaluate and compare stereo matching methods. It integrates a series of heterogeneous algorithms, and some pre and post filtering techniques, adapting and

standardizing their interfaces so they can work together and combine processes to obtain improved results in comparison to applying each process separately. This testbed has been put at the service of the research community and downloaded over one thousand times up to now, denoting a strong interest. Several new contributions have been received so far from the community and the next version of the testbed is currently under development. This work is described in chapter 5 of this thesis.

11. An automatic expert systems for image correction and terrain reconstruction in stereo vision applications has been proposed and developed. It is based on three stages where the main underlying idea is the successive application of automatic image processes, mapping the expert knowledge. This expert system is able to automatically adjust the intensities of the input stereo pair. It also proves the constant image brightness (CIB) assumption is often erroneous. This work is described in chapter 6 of this thesis, based on Correal et al. (2014a).

12. Conclusions and promising future research lines have been identified and detailed in chapter 7 of this thesis.

13. As it can be inferred from the above paragraphs, results, knowledge and findings have been disseminated through scientific publications. Some software has been released, in the form of an open-source project, as indicated before. The list of published papers is enumerated next, indicating their main contributions.

1.4 Dissemination of Results

Dissemination of results, to date, is summarized in the following works:

1. Odwyer and Correal (2008). "Experiences in Producing a Preliminary Navigation OBSW Prototype for the Exomars Rover Based on EDRES". This paper has been published in the proceedings of the ESA Workshop on Advanced Space Technologies for Robotics and Automation (ASTRA), 2008, ESA/ESTEC Noordwijk (The Netherlands). ExoMars is a European led exploration mission to Mars including a surface rover. The ExoMars

navigation flight software will have to meet ESA (European Space Agency) mission requirements and as such will be subject to strict development methods and standards. Typically, the software development process for OBSW starts with no code, an engineering design and a software requirements set then iterates with testing until a flight software is available and validated. In this case, in the B1 phase and before a mature SW (Software) design has been elaborated, preliminary OBSW (OnBoard SoftWare) has been produced. This task facilitated the assessment of previously developed code modules and algorithms based on (but not limited to) EDRES (Environnement de Développement pour la Robotique d'Exploration Spatiale / Space Exploration Robotics Development Environment). EDRES (Maurette and Rastel, 2002) is the result of thirteen years of software development for autonomous rovers at CNES (2014). It consists of a collection of algorithms, applications and tools covering the majority of the functions required for autonomous movement generation and execution for exploration rovers. EDRES addresses in a very comprehensive way the need to design, simulate and evaluate complex strategies and architectures related to rover design and software development. It contains a rich set of resources useful for the software development of any given rover. EDRES has been built to serve as a workshop for the creation of new algorithms, tools or applications. This paper focuses on the assessment of the flight suitability of heritage code developed by CNES, encompassing its ability to run on the flight processor, its performance and its flight-worthiness in terms of issues such as compliance with flight software standards. It has involved porting the developed SW from a PC and Linux environment to a more flight representative processor, a LEON2-Pender board with RTEMS. An evaluation of the performance of the navigation SW running on the flight representative processor has been also done. This paper details the experiences and lessons learnt during the porting process and the subsequent study to analyze the performance of the navigation SW within the scope of the early phases of the ExoMars rover project. This work is introduced in chapter 1 of this thesis.

2. Correal and Pajares (2010). "Framework for Simulation and Rover' Visual-Based Autonomous Navigation in Natural Terrains". This paper has been published in the proceedings of the 7th Workshop RoboCity2030-II, Madrid, Spain. It presents the design

and implementation of a framework aimed to support simulation and development of natural terrain autonomous navigation approaches, such as planetary exploration rovers. Most of the tools in this domain are proprietary of national space agencies usually not accessible to the research community. In this paper, we address the feasibility of developing high-level autonomy strategies, such as autonomous navigation, for analysis and validation at the functional level, based in open source tools, at the reach of any researcher/institution. We also present stereovision-based perception algorithms, path planning and navigation approaches developed onto this framework and how rover, sensors and terrain models have been created within this environment to support these studies. The integration of these tools under a simulated domain makes the main finding of this paper. This work is developed and described in detail in chapters 3 and 4 of this thesis.

3. Correal and Pajares (2011a). "Modeling, simulation and onboard autonomy software for robotic exploration on planetary environments". This paper has been published in the proceedings of the International Conference on DAta Systems In Aerospace (DASIA) 2011, Malta. It presents the design and implementation of a framework aimed to support development of planetary exploration rovers autonomy approaches. Most of the tools in this domain are proprietary of national space agencies and commonly not accessible to the research community. In this paper, we present a framework based on open source tools with some adaptations/extensions, at the reach of any researcher/institution. Vehicle, sensors and terrain models have been created to simulate the operational environment. Visual-based autonomous navigation capabilities have been developed; analysis and functional level validation of approaches and strategies have been carried out supported by this framework. This work is developed and described in detail in chapters 3 and 4 of this thesis.

4. Correal and Pajares (2011b). "Onboard Autonomous Navigation Architecture for a Planetary Surface Exploration Rover and Functional Validation Using Open-Source Tools". This paper has been published in the proceedings of the ESA International Conference on Advanced Space Technologies in Robotics and Automation, 2011. In the planetary domain there exist severe communication constraints linking planets like Mars

and the ground station on Earth, such as signals delays and communication windows. This makes critical the necessity of having onboard local autonomy, allowing deployed vehicles making its own decisions independently of ground control, maximizing the scientific return of the mission and reducing operational costs and risks. This paper outlines a visual-based autonomous navigation approach and software architecture for exploration rovers' onboard autonomy in planetary environments. A framework with simulation and monitoring capabilities is presented, developed to support this research, allowing analysis of performance and behaviors to evaluate feasibility of strategies and early functional validation of approaches. Some of these capabilities are partially supported by COTS (Components Of The Self) and open-source packages. This work is developed and described in detail in chapter 4 of this thesis.

5. Correal et al. (2013). "Mejora del Proceso de Correspondencia en Imágenes Estereoscópicas Mediante Filtrado Homomórfico y Agrupaciones de Disparidad". This paper has been published in Revista Iberoamericana de Automática e Informática Industrial, vol. 10, issue 2, 178-184. In this work the problem of the matching process in stereo images of terrains obtained with a Videre STH-DCSG 9mm stereoscopic system is addressed. A number of matching algorithms, which are part of the overall stereoscopic process whose purpose is to perform a 3D reconstruction for autonomous navigation of robots in unstructured natural environments, are used. First, a study of the effects of applying homomorphic filtering on the input images as a pre-matching step is performed. By that filtering a significant improvement in the disparity map is achieved, obtaining a higher number of true correspondences in contrast to perform the process without filtering. After that, the resulting disparity map is again filtered using a process directed by clusters and based on the principle of spatial continuity to eliminate false positives and erroneous correspondences. Both filtered constitute the main contribution of the work. This work is developed and described in detail in chapter 6 of this thesis.

6. Correal et al. (2014a). "Automatic Expert System for 3D Terrain Reconstruction Based on Stereo Vision and Histogram Matching". This paper has been published in Expert Systems with Applications. 41, pp. 2043-205. It proposes an automatic expert system for 3D terrain reconstruction and automatic intensity correction in stereo pairs of images

based on histogram matching. Different applications in robotics, particularly those based on autonomous navigation in rough and natural environments, require a high-quality reconstruction of the surface. The stereo vision system is designed with a defined geometry and installed onboard a mobile robot, together with other sensors such as an Inertial Measurement Unit (IMU), necessary for sensor fusion. It is generally assumed the intensities of corresponding points in two images of a stereo pair are equal. However, this assumption is often false, even though they are acquired from a vision system composed of two identical cameras. We have also found this issue in our dataset. Because of the above undesired effects the stereo matching process is significantly affected, as many correspondence algorithms are very sensitive to these deviations in the brightness pattern, resulting in an inaccurate terrain reconstruction. The proposed expert system exploits the human knowledge which is mapped into three modules based on image processing techniques. The first one is intended for correcting intensities of the stereo pair coordinately, adjusting one as a function of the other. The second one is based in computing disparity, obtaining a set of correspondences. The last one computes a reconstruction of the terrain by reprojecting the computed points to 2D and applying a series of geometrical transformations. The performance of this method is verified favorably. This work is developed and described in detail in chapter 6 of this thesis.

7. Correal et al. (2014b). “Autonomy for Ground-level Robotic Space Exploration: Framework, Simulation, Architecture, Algorithms and Experiments". This paper has been published in ROBOTICA Journal. June 2014, pp. 1–32. Robotic surface planetary exploration is a challenging endeavor, with critical safety requirements and severe communication constraints. Autonomous navigation is one of the most crucial and yet risky aspects of these operations. Therefore, a certain level of local autonomy for onboard robots is an essential feature, so that they can make their own decisions independently of ground control, reducing operational costs and maximizing the scientific return of the mission. In addition, existing tools to support research in this domain are usually proprietary to space agencies, and out of reach of most researchers. This paper presents a framework developed to support research in this field, a modular onboard software architecture design and a series of algorithms that implement a visual-based autonomous navigation approach for robotic space exploration. It allows analysis

of algorithms' performance and functional validation of approaches and autonomy strategies, data monitoring and the creation of simulation models to replicate the vehicle, sensors, terrain and operational conditions. The framework and algorithms are partly supported by open-source packages and tools. A set of experiments and field testing, with a physical robot and hardware, are described as well, detailing results and algorithms' processing time, which experience an incremented of one order of magnitude when executed in space-certified like hardware, with constrained resources, in comparison to using general purpose hardware. This work is developed and described in detail in chapters 3 and 4 of this thesis.

1.5 Contents

This dissertation is structured in chapters. These chapters are organized according to the natural order of the research works carried out, the distribution of which is given below:

1. First chapter. *Introduction*, the objectives of the thesis are considered and studied, as well as motivations and introduction of proposals, as set forth in the preceding sections.

2. Second chapter. *Robotic Space Exploration. Autonomy*, gives an overview of the robotic planetary exploration domain. Introduces the topic of autonomy, focusing on its implications to the robotics area in general and more concretely to the space robots. An overview of the rover's operations and activities performed on a target planet is given, along with the autonomy approaches commonly followed for this kind of missions.

3. Third chapter. *Software Development Support Framework*, gives an overview of the difficulties faced by researchers in the robotic planetary exploration domain and how the lack of facilities, vehicles and resources represent a challenge hard to overcome. A framework to support research in this domain is proposed, along with its requirements and design.

4. Fourth chapter. *Autonomy for Planetary Exploration Rovers: Architecture and Navigation*, outlines the design of a multilayer and modular onboard software architecture for the

rover's control and a series of algorithms that implement a visual-based autonomous navigation strategy for robotic space exploration. A set of experiments are described, including results and algorithms' processing time details.

5. Fifth chapter. *The Perception Phase within the Autonomous Navigation Process. A Testbed for Stereo Vision Algorithms*, details the design of a testbed that aims to centralize and standardize the broad range and heterogeneity of stereo matching algorithms, focusing in the application of stereo-based methods to real situations. This work has led the creation of an open-source project, already made available to the research community, and counting with one thousand downloads to date.

6. Sixth chapter. *Automatic Expert System for 3d Scene Reconstruction Based on Enhanced Stereo Vision by the Application of Novel Image Filters*, addresses the problem of computation of correspondences in pairs of images obtained with a real stereoscopic system and the necessity of the application of methods to correct the differences between both images of the pair. The effects of applying techniques such as homomorphic filter and histogram matching on the input images as a pre-matching step are detailed as well as the development of a novel filtering process, directed by clusters and based on the principle of spatial continuity, performed on the resulting disparity map to eliminate erroneous correspondences. Also an automatic expert system for 3D terrain reconstruction and automatic intensity correction in stereo pairs of images based on the previously introduced techniques is proposed. This expert system exploits the human knowledge, which is mapped into three modules, based on these image processing techniques.

7. Seventh chapter. *Conclusions and Future Work*, outlines the general lines followed, making an overall assessment of the work done. Prospective research lines are suggested and future improvements and extensions of the proposed methods are proposed.

Chapter 2

ROBOTIC SPACE EXPLORATION. AUTONOMY

This chapter gives an overview of the robotic planetary exploration domain, stating the past and current world-wide interest in this domain by national space agencies and the research community by summarizing the set of missions accomplished so far, as well as some scheduled in the short and mid-terms. An introduction on the topic of autonomy is also included, focusing on its implications to the robotics area in general and more concretely to the space robots, indicating how it serves the whole mission and its necessity and benefits. Finally, an overview of the rover's operations and activities performed on a target planet is given, along with the autonomy approaches commonly followed for this kind of missions. The subsystems constituting an autonomous navigation system are introduced, indicating the present work is mainly focused in the perception stage. This chapter pretends to serve as an introduction to frame the context and scope of the work introduced as well as the justification of the research lines presented.

2.1 Background

Planetary exploration has always fascinated mankind. Over the last decades, a number of missions have been developed with the purpose to explore the outer space and gather data about the climatology, composition or conditions of our neighbor planets and surroundings. Some of these consist of probes, satellites or even surface modules. In the last years, there has been an increasing interest in celestial body and planetary exploration using mobile robots. Last space operations have done an extensive use of robotics systems so far and it is foreseeable that will be the case in the future, as there is a number of missions scheduled for the next years.

So far, it has been possible for the first time in the history of humanity to drive a vehicle on the surface of Mars at the same time than performing scientific experiments and gathering crucial information about the planet. The closest pictures ever have been taken from the actual surface of the red planet, rocks and soil composition have been analyzed, meteorology information has been obtained and invaluable data have been collected. All of that would not have been possible without deploying (semi) autonomous robotic vehicles onto the surface.

The first successful robotic mission to Mars was the Pathfinder (Stone, 1996; Mishkin et al., 1998), from NASA (2014), launched in 1996. It included a lightweight wheeled robot to show the viability of sending a load of scientific instruments to another planet with a simple system at a reduced cost. It travelled approximately 100 meters in total, never more than 12 m from the Pathfinder station, during its 83 sols -Martian days- of operation.

The MER (Mars Exploration Rovers) mission (Crisp et al., 2003) by NASA, deployed two twin rovers, Spirit and Opportunity, on opposite sides of Mars in January 3 and January 24, 2004, with the primary goal of searching for clues of past water activity on Mars, characterizing a wide range of rocks and soils. The mission, originally scheduled for 3 months duration, was an unprecedented success. At the time of this writing, it has been in operation for more than 10 years, with more than 40 Km distance travelled and hundreds of thousands of Mars' images sent back to Earth. Next, the NASA Phoenix spacecraft landed on the surface of Mars in 2008 (Smith et al., 2008). It was not a mobile robot, but it included a number of scientific instruments that allowed scientists to make a historic discovery, confirming the existence of water ice near the surface beside the Lander.

The latest NASA robotic mission to Mars is the MSL (Mars Sciences Laboratory) (Grotzinger et al., 2012). A car-sized robot –Curiosity- was deployed on Mars in Aug. 2012. Curiosity is about twice as long (about 3 meters, 10 feet) and five times as heavy (900 Kg, 2000 pounds) as the previous twin Mars Exploration Rovers. It is capable of rolling over obstacles up to 65 centimeters high (25 inches) and travel up to 200 meters (660 feet) per day on Martian terrain. The main scientific goals of the MSL mission are to help determine whether Mars could ever have supported life, determining the role of water, and to study the climate and geology of Mars. The mission will also help prepare for future human exploration.

Besides the current economic circumstances and budget constraints, there is still a patent world-wide interest in this kind of mission. Both ESA (2014) and NASA have several missions scheduled to continue the exploration of our solar system planets and beyond. There is a great indecision on possible missions. However after the discovery of water in Mars, found by the Phoenix mission, it may impulse the schedule and development of new missions to Mars, and maybe other planets, to deepen into this fact and get new evidences. The ultimate goal seems to be the future human exploration of Mars and returning to the Moon.

Establishing if life ever existed on Mars is one of the outstanding scientific questions of our time. NASA Mars exploration strategy is known as "follow the water". Water is a key because almost everywhere we find water on Earth, we find life. To address this important goal, the ESA has established the *ExoMars programme* to investigate the Martian environment and demonstrate new technologies, paving the way for a future Mars sample return mission in the 2020's. The ExoMars program consists of two phases: the first, an Orbiter plus an Entry, Descent and Landing Demonstrator Module, will be launched in 2016; and the other, which features a mobile robot, will be launched in 2018. Both missions will be carried out in cooperation with Roscosmos (2014), the Russian Federal Space Agency. The ExoMars rover will travel across the Martian surface to search for signs of life. The primary objective is to land the rover at a site with high potential for finding well-preserved organic material, particularly from the very early history of the planet. It will collect samples with a drill and analyze them with next-generation instruments, establishing the physical and chemical properties of these samples. This future rover is intended to be highly autonomous. The rover will build a 3D model of its environment using stereo images and analyze them in order to establish scientific targets. It will then navigate autonomously to the selected target and use its robotic arm and instruments to collect and analyze data. The intentions are to decrease by 3 or more the time NASA's last rovers took to plan and schedule its actions to get the samples, what would increase dramatically the scientific return of the mission.

The Astrobiology Field Laboratory Mission is expected to be launched by NASA in 2016 to Mars (Beegle et al., 2007). It's intended for this mission to last one Martian year, with a possible extension of another year. It will study the chemistry associated with life, looking for components such as nitrogen and carbon. It will have a high level of autonomy, being able to perform fully autonomous long-distance navigation, avoiding hazards, instrument placement and science investigations.

The MoonNext mission is expected to be launched by ESA in 2018, with destination to the Moon. It will include a rover which will perform geochemical and seismic activity measures, as well as some biological experiments. Given the "short" distance from Earth to Moon, the rover can be tele-operated, so there are less needs for autonomy than in the case of Mars. The landing process is intended to be autonomous and avoid hazards in order to reach precisely the desired landing site.

There are also other planned missions such as *Mars Sample Return*, which objective is to reach Mars, collect some samples, leave Mars and get those samples to Earth to be analyzed in full detail. Both NASA and ESA are interested in this project, so it could finally end up being a multi-billion international mission. There is still no clear date for launching this mission, although NASA expects it being possible before 2030.

Besides NASA and ESA, other national space agencies have shown interest in robotic space exploration. The China National Space Administration (CNSA) has developed a space program in several phases. The first spacecraft of the program, the Chang'e 1 lunar orbiter, was launched on October 2007. A second orbiter, Chang'e 2, was launched on 1 October 2010. Chang'e 3, which includes a lander and rover, was launched on December 2013 and successfully soft-landed on the Moon. It will be followed by a sample return mission, Chang'e 5 scheduled for 2017. Chandrayaan-1 was India's first unmanned lunar probe. It was launched by the Indian Space Research Organisation (ISRO, 2014) in October 2008, and operated until August 2009. The mission included a lunar orbiter and an impactor. A second phase is planned to be launched by 2017, including a wheeled rover that will move on the lunar surface and will pick up soil or rock samples for on-site chemical analysis. The Japanese Space Agency (JAXA, 2014) has planned also a multi-phase mission to the moon. The first spacecraft of the program, an unmanned lunar orbiter, was launched on September, 2007. SELENE-2, Japan's first lunar lander and rover, is expected to be launched in 2017. It will consist from one large lander and a small-sized rover, about 100 Kg., with some penetrators. The program also includes a lunar sample return mission (SELENE-3), a mission to Mars, in order to collect data for future manned expeditions.

2.2 Autonomy

There are several definitions for autonomy. Depending on the concrete technology or application field, the definition of autonomy can be slightly different. The more general definition of autonomy can be "the self-government of an individual". This concept can be applied to many cases and situations. It can be applied to people, situations, systems or even robotics for example. In robotics the concept of autonomy is applied to the situations when a system –robot– is able to make some decisions by itself without human intervention. This autonomous decision making is supported by different techniques, like algorithms or artificial intelligence.

Also, sometimes the differences between a robot and a machine are not clear. There can be many ways to define what a robot is. Prof. Mataric gives this definition; "A robot is an autonomous system which exists in the physical world, can sense its environment, and can act on it to achieve some goals" (Mataric, 2007). Deepening in Prof. Mataric definition, a robot must be autonomous and acts according to its own decision. It must exists in the real world and therefore deal with the physics laws. Must be able to get information about itself and its surrounding environment through its sensors and act and change or affect in some way this environment using its actuators (wheels or arms among others). Following the previous definition, a car for instance is not a robot. A car is physical and has sensors and actuators, but it doesn't make any decision by itself nor has any goal. As another example, a robot simulation in the computer is not a robot either. It can make decision, but it doesn't exist in the real world.

Autonomy can be also though as the amount of intervention required for controlling a robot. It can be placed teleoperation in one extreme of the line and full autonomy on the other. In teleoperation constant interaction is needed, a person is remotely controlling the robot. When full autonomy is achieved, the human interaction is minimized or even eliminated from the equation. The autonomy level measures the percentage of time that the robot is carrying out its task on its own; the amount of intervention required measures the percentage of time that a human operator must be controlling the robot. These two measures sum to 100%. Teleoperated robots are fully controlled by a human operator (Yanco and Drury, 2004).

Generally a hybrid approach it's been the most widely used and successful one so far. Depending on the concrete application and scenario, different percentages and weight to each approach must be given. The operators may be in the need to override the robot's decisions because any hazard or unexpected change, or the robot may need to take control if a communication loss happens. This has been called adjustable autonomy, making possible to adjust the levels to concrete situations and develop mixed approaches.

2.3 Rover's Operation

Robotic planetary exploration is a very challenging domain. Once a robotic vehicle has been deployed on the targeted planet surface, it has to deal with unexpected situations and non-predictable local conditions, ideally without any or minimum dependency on ground control. The constraints on communications, with signal delays up to 40 min from Earth to Mars, and the restricted availability of satellites orbiting the target planets, limiting communications to 10-minutes periods twice a day in the case of Mars, makes teleoperation an unfeasible approach. This makes imperative the necessity of a certain level of autonomy. Otherwise, the rover would require continuous support from human operators, slowing down the mission pace, given the communications constraints, and raising maintenance costs. A higher level of autonomy would reduce the operational burden and cost, freeing more resources for scientific exploration.

In terms of autonomy, mobile robotic planetary missions achieve planetary surface exploration using a human-robot system that operates in a semi-autonomous fashion (Biesiadecki et al., 2005; Leger et al., 2005; Maimone et al., 2006a). That is, it incorporates remote planning, command sequencing and visualization of rover activity sequences and related data products by an Earth-based science-engineering team. The rovers perform autonomous execution of a daily sequence of commands, developed on ground control every day by human experts and sent to the rovers as a to-do list for that day.

Among the multiple operations a rover shall perform, navigation is one of the most critical. It implies the capacity of the robot to travel from one location in the planet to another by itself performing scientific experiments, avoiding obstacles and hazards such as tipping over, falling in cracks or getting stuck in soft sand, that may put the whole mission at risk. Autonomous navigation is commanded by specifying explicit surface coordinates to be reached using onboard sensing, perception, motion planning and execution. Mission operators provide a global path plan in the form of a series of waypoints that is executed and evaluated onboard the rover. Often, the goal waypoint at the furthest extent of a long traverse is beyond the reliable field of view of the rovers' mast-mounted navigation cameras, in which case the rover is commanded to drive itself –autonomously- into areas never before seen in images sent to Earth (at least not at sufficient resolution to locate potential obstacles in advance).

In general, an autonomous navigation system consists of three main subsystems: perception, path planning and path execution. The perception subsystem receives inputs from the onboard sensors, in the case of planetary rovers usually utilizing passive imaging sensors such as stereo cameras, producing local area terrain maps containing elevation data and detected hazards. The path planning subsystem allows the robot to navigate through obstacles, and finding the path in order to reach the target without collision. It is often decomposed into path planning and trajectory planning. Path planning is to generate a collision free path in an environment with obstacles and optimize it with respect to some criteria. Trajectory planning is to schedule the movement of a mobile robot along the planned path.

The path execution subsystem receives a path to execute from the path planning subsystem and controls the vehicle's motion. In this stage, the robot must monitor its motions to track the path and verify that it is moving according to the plan until the goal is reached. It must also monitor the environment to ensure that no unexpected obstacle blocks its movements. The focus for this work is the perception subsystem.

In order to design, develop and test an autonomous navigation system for a planetary rover, intense research work on earth is required prior to any launching. And that requires the appropriate infrastructure and facilities, which is the focus of the next chapter.

Chapter 3

SOFTWARE DEVELOPMENT SUPPORT FRAMEWORK

This chapter gives an overview of the difficulties faced when research works in the robotic planetary exploration domain is undertaken. Existing tools to support research in this domain are usually proprietary to space agencies, and out of reach of most researchers. This lack of facilities, vehicles and resources represent a challenge hard to overcome for many research groups. Even though in the case of teams having access to the necessary resources, there are many chances they will not be available until later phases of the mission. A framework to support research in this domain is proposed, along with its requirements and design. The framework has been developed from the ground up, as there is not any equivalent framework available to be used, by integrating existing resources and tools with new developed ones. The creation of the framework has been partly supported by open-source packages. It allows analysis of algorithms' performance and functional validation of approaches and autonomy strategies, data monitoring and the creation of simulation models to replicate the vehicle, sensors, terrain and operational conditions. Despite this framework is not part of the research objectives of this work, it has been crucial to support the developments and achievements presented in this thesis. It is a necessary resource that constitutes a contribution itself. The work described in this chapter is based on Correal and Pajares (2011a) and Correal et al. (2014b).

3.1 Background

The study of autonomy for robotic space exploration implies algorithms and hardware/software interface definitions cannot be done exclusively at the theoretical level. To develop, test and validate strategies and approaches before launching, the operational settings shall be replicated with as much fidelity as possible. Field testing for missions to Mars is usually performed at locations such as Rio Tinto or Tenerife (Spain), where terrain and soil characteristics are similar to the red planet. However, the necessary infrastructure and high cost to take the team and equipment to those locations means that testing can only be carried out sporadically or at last phases. Some institutions, like NASA or ESA, build an indoor/outdoor Mars-like terrain and have replicas of the robotic vehicles to do extensive in-house testing. However, it still requires too many resources, usually out of reach to researchers from other kind of institutions working in this domain.

An affordable alternative to test and evaluate autonomy approaches is simulation. Widely used in many other disciplines, it allows virtual replication of the operational settings, where algorithms can be developed and tested. The purpose is not to eliminate field testing, which is absolutely necessary, but to make possible and affordable the development and functional validation of autonomy approaches when a physical vehicle is not available, or not yet, and the software is not mature enough to test with the real hardware, testing feasibility and performance at initial phases. It reduces the dependability on the physical devices and limits the expensive field testing to last phases for final validation. It also makes possible the development of algorithms and functional validation by researchers from a given institution, leaving field testing and final validation to other teams at other institution with access to the necessary facilities and resources.

Nevertheless, simulation has its limitations. A complete and detailed emulation of some features such as meteorological phenomena, physical and mechanical forces or sophisticated terramechanics may be complex and hard to achieve (Thueer et al., 2007; Iagnemma et al., 2011; Smith et al., 2013; Zhou et al., 2014). However, it is important to note that depending on the scope of the research to be supported by the framework just the necessary aspects and features with the appropriate level of fidelity have to be modeled. It is not within the scope of this work to focus on aspects such as mechanical or electrical design, dynamics, forces or weather conditions on the planet. In case of this work, the objective is to support the design and development of high-level autonomous navigation approaches, for analysis of strategies and validation of algorithms at the functional level. The key requirement is closing the control loop perception-action-perception, executing multiple steps of sensors and image acquisition, path planning and navigation.

To make use of a framework that provides these capabilities, three main approaches can be considered: 1) have access to an already existing framework purposely developed to support research in this specific domain; 2) develop a tailored framework from the ground up specifically designed to include the necessary features and meet the requirements or 3) create a framework integrating and adapting existing tools and components, each one providing part of the required functionality.

Using already existing tools purposely developed to support space robotic exploration research is undoubtedly the best choice. Since they are already developed and validated, it relieves the burden of creating such a complex framework, and allows the focus to be on autonomy research, which is the main purpose of this work. There are just a handful of these frameworks. ROAMS (Rover Analysis, Modeling and Simulation) (Yen et al., 1999; Jain et al., 2003) from NASA/JPL (Jet Propulsion Laboratory), includes high-fidelity models for rover kinematics and dynamics, terrains, wheel-soil contact, power, sensors and actuators like cameras, inertial units and motors. The VIPER system (Edwards et al., 2001), from NASA, is a high-fidelity virtual reality environment. It models 3D terrains and onboard cameras to simulate environment perception, forward kinematics and environment physics. LiveInventor (Neveu and Shirley, 2003) also from NASA Ames Research Center, is an interactive development environment for robot autonomy which integrates a physically-based simulation library with a 3D rendering environment. It includes kinematics modeling, a physics simulation library, an embedded scheme interpreter and a distributed communication system. In Europe, EDRES (Maurette and Rastel, 2002; Odwyer and Correal, 2008) by CNES allows creating terrains and rover models, simulating sub-systems like locomotion, localization, perception and control. It includes libraries for computer vision processes, autonomous navigation algorithms and interfaces with rover hardware and devices. 3DROV (Poulakis et al., 2008), developed by TRASYS Space for ESA, allows creating rover models, sensor and scientific instruments, including mechanical, power and thermal subsystems, as well as planetary environment models including ephemeris, terrain and atmospheric conditions, a generic onboard controller and a ground control station module. The main problem with these environments is they are proprietary to the institutions and national space agencies and not publicly available to the research community unless through direct contract. There was an initiative from NASA/JPL to release ROAMS, but unfortunately it was never accomplished.

Regarding the second approach of developing a tailored framework from the ground up, the main advantage is that it guarantees that all requirements will be met by adjusting to each concrete necessity and projected feature. However, given the inherent complexity of building such a framework and the many technologies involved, the effort to create such a framework may be greater than the one to develop autonomy algorithms, which is the actual purpose, thus deflecting from the objective of this work.

Therefore, the approach followed in this work has been the integration of already available software packages and tools, each one supplying part of the functionality, building up a framework providing equivalent capabilities to the space agencies' ones, inaccessible to the research community. These components must be adapted and interfaced to work together. It represents a balanced approach between development effort and obtained capabilities. The design of such a framework to support development and validation of autonomy algorithms for space robotic exploration is presented next.

3.2 Framework Structure and Design

The main purpose of the framework is to support the development of autonomy algorithms and its functional validation. A crucial capability of the framework must be the simulation, so that it is capable of providing the algorithms the necessary data from the emulated sensor, as well as executing the resulting actuation commands in the simulated environmental and operational conditions. These algorithms shall be developed with the support of the framework but with complete independence on it nor the simulator. This way, the transition from the simulation to the real hardware will be smooth; once in the target platform, the algorithms will receive data from real sensors instead from simulated ones, remaining them unaware of this change or needing just small adaptations, depending on the level of fidelity of the simulated models.

The framework has been designed as a composition of subsystems with interfaces between them, where functions like simulation of operational conditions, sensors data production, control algorithms execution, software debugging, data visualization or communications are independent from each other. The conceptual architecture design of the framework is shown in Figure 1. It consists of three main subsystems, namely: 1) Simulation Environment, which emulates the rover hardware and sensors, kinematics and dynamics, terrain and environment models, physical forces and interactions, 2) Control Center, for data monitoring, control and visualization and 3) Onboard Software, which implements the rover intelligence, decision making capabilities and algorithms for autonomous navigation.

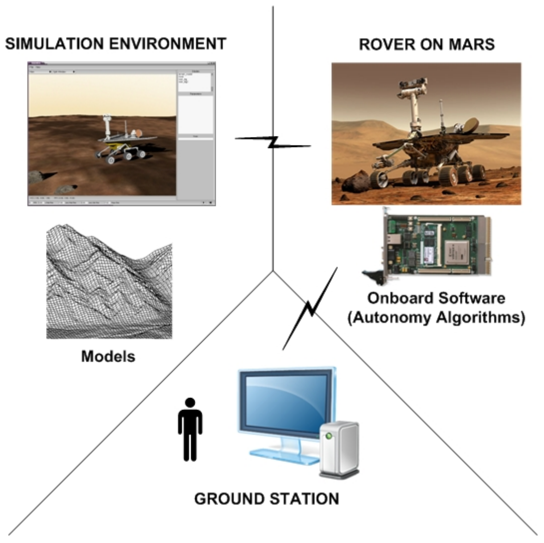


Figure 1. Framework conceptual architecture and subsystems

The system is based on the integration of available tools and components. Figure 2 shows some of the technologies, software packages and tools used to create each subsystem. Some of them are in turn built up of other components; just the ones directly used for this work will be mentioned and discussed within this document, although shown in the figure for complementary information. The higher development efforts have been put in the onboard software and control station subsystems, both developed from the ground up, with the support of some third-party libraries, as well as the models –rover and terrain- for the simulator.

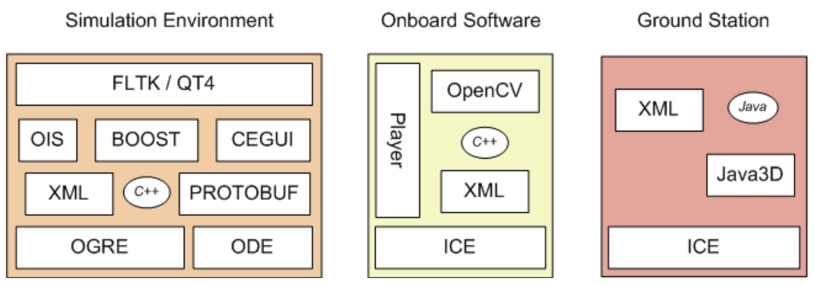


Figure 2. Subsystems of the framework, tools and technologies used on its development

The main advantages of this modular design are reuse and maintainability, so that any subsystem or module can be independently updated or replaced. For instance, the control station could be replaced by another visualization or control software in the future, just complying with or adapting the communications and interface to the system, while the autonomy algorithms and the simulator remains unaltered. Same happens with the simulator, which could be replaced or upgraded, being it transparent to other modules. And above all, the control algorithms and autonomous capabilities, which are the main focus of this work and research interests, can be developed with independence to the simulation or monitoring and visualization subsystems.

XML is used to structure files to store system configuration parameters for the different subsystems. Some sample configuration parameters are rocks' size, pose and orientation, rover' wheels size, chassis measures, joints, torques or gains for the simulation subsystem or camera's field of view and focal length, resolution, stereo base, planning distance, navigation distance or security thresholds for the navigation control software. Many of these configurable parameters are cited along the text.

In order to use the framework, each subsystem includes an API (Application Programming Interface) specifying a set of functions that accomplish specific tasks that allow interacting with specific components. For instance, the robot can communicate with the control center to send its telemetry and status data for monitoring purposes using its API. Also, from its onboard control software the robot is able of getting readings from sensors or commanding actuators as if they were physically connected by making use of the simulator API and the communications and abstraction layers. More details on the architecture are given in following sections.

3.2.1 Simulation

The main purpose of the simulation environment is to replicate the operational conditions, closing the control loop. The necessary data to feed the control algorithms is obtained from the plant, which has to be modeled; it then computes an output to be sent back to the plant to simulate the effects it produces. In this case, the plant is a simulated rover, terrain and its interactions, and the controller is the rover onboard software and autonomy algorithms. Inputs to the controller are sensor readings obtained from the plant –stereo camera images, IMU (Inertial Measurement Unit) data, etc. and outputs computed trajectories, sent to the rover decomposed in lower level motor commands to generate motion.

The space rovers' simulation domain requires some concrete capabilities not supported by many simulation frameworks. The following is a list of main requirements for this framework:

1) Capability of creating 3D models and simulate the 3D space. It implies creating models of rough terrains with rocks, craters, cracks and slope areas to emulate a Mars-like environment, and the capability of creating complex 3D robot models.
2) Simulation shall include aspects such as vehicle kinematics and dynamics, contact forces, gravity, or friction to analyze the interaction of the rover with the environment.
3) Capability of including or creating models of the vehicle's sensors and actuators, such as stereo cameras, inertial measurement units and motors.

The goal is that the whole operation process of the rover can be modeled; meaning having a rover on a Mars-like terrain capable of taking images from its cameras, as well as other measures from its sensors, and process them to build a map of the environment so it can make its own decision and compute suitable trajectories to travel from one location to another.

There are also some other non-functional requirements of principal importance for this framework such as versatility, flexibility, support, maintainability, extensibility, maturity, reliability, learning effort, platform, interfaces, source code openness, developers/users community size and activity and third party available tools. For instance, source code openness is one of the crucial aspects. Not having access to the source code prevents us from having the complete control of what is going on below the surface as well as the inability to carry out any necessary adaptations and extensions to the original software. Moreover in the space robots field, which is a very critical domain where all processes have to be under complete control and access of the development engineers. Besides, as opposite to open-source form, the classic close code industrial approach creates a complete dependability on the owner company; in case that company decides to suspend the project development or support for any strategic reasons, the work would be critically affected, as there is no access to its sources to make the necessary changes or extensions to continue the efforts and research lines.

As introduced before, the framework has been built employing available software packages and tools. To achieve the mentioned requirements and capabilities, several candidates have been analyzed such as Matlab/Simulink (Timothy, 2010), an environment for multi domain simulation and Model-Based Design for dynamic and embedded systems, Microsoft Robotics Developer Studio, a Windows-based environment to create robotics applications using Microsoft Visual Programming Language, Webots from Cyberbotics (Michel, 2004), a development environment used to model, program and simulate mobile robots, Player/Stage/Gazebo (Gerkey et al., 2003), a set of software tools for robot and sensor applications, DynaMechs (McMillan, 2003), a Multibody Dynamic Simulation Library, Vortex Simulation Toolkit (CMLabs, 2003), a development platform for physically accurate modeling of ground vehicles, terrain, soil, machines, robots, and other real-world objects, Darwin2k (Leger, 1999), a free, open source dynamic simulation and automated design synthesis package for robotics, RobotSim from Cogmation robotics (Cogmation, 2010), a 3D physics enabled robotic simulator enabling robot and environment modeling, Simbad (Hugues and Bredeche, 2006), an open source Java3d robot simulator for scientific and educational purposes, and the Mobile Robot Programming Toolkit (MRPT) (Blanco, 2010), an open source C++ library to design and implement algorithms related to Simultaneous Localization and Mapping (SLAM), computer vision and motion planning or CARMEN (Montemerlo et al., 2003), an open-source collection of software to provide basic navigation primitives for mobile robot control which includes 2D simulation capabilities. Although a detailed discussion on the evaluation of this packages is beyond the scope of this thesis, and more details can be found in Correal et al. (2011a) or Staranowicz and Mariottini (2011), a comparison between Gazebo, selected as the simulation capabilities provider for this framework, and some other popular simulation tools is indicated next as a function of the functional and non-functional requirements stated previously.

Webots (Michel, 2004) is a development environment used to model, program and simulate mobile robots. It has 3D simulation capabilities, including rigid body dynamics simulation. One of its main strengths is the fact that it is developed by a company, which, as opposite to some collaborative development projects, ensures current maintainability and support. Some of the main drawbacks of Webots are the fact that it is not for free, although it supports an educational version, and it is not open-source. Not having access to the source code is the main drawback of this tool for the reasons introduced before: there is no control of what is going on below the surface, the simulator cannot be adapted or extended by ourselves and there is a complete dependability on the owner company and the continuity of the product.

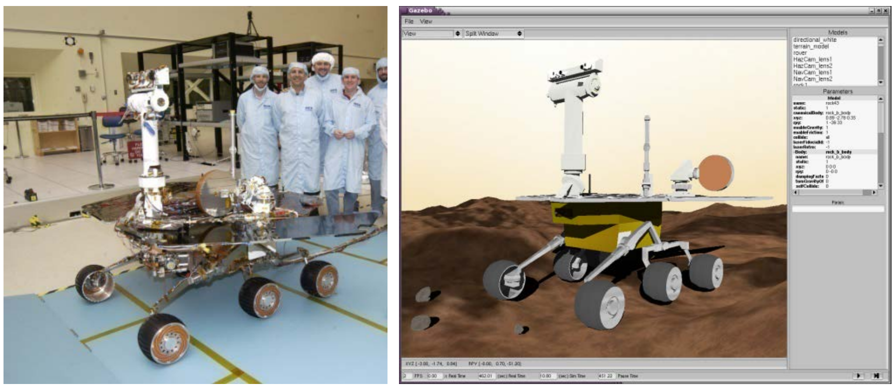
Carmen (Montemerlo et al., 2003) was a popular open-source collection of software started at CMU (Carnegie Mellon University) several years ago. It provides some simulation capabilities and basic navigation primitives for mobile robot control. If offers interfaces to some general purpose mobile platforms, typically used for academic research, such as ATRV, B21R, Pioneer I and II, Nomadic, etc. As introduced before, the capability to create 3D simulations, including models of terrains, and simulate perception in the 3D space using sensors such as cameras to capture images from within the environment is mandatory for this domain. However, CARMEN just provides basic 2D simulation capabilities; it does not allow to create complex 3D robot models like a Martian rover, as shown in Figure 3, or modeling 3D rough terrains with rocks, craters, cracks, slope areas, etc. simulating a Mars-like environments, as shown in Figure 4. 3D simulation capabilities to create and operate with such models are indispensable for this research domain and to support these developments. Moreover, CARMEN last release was done in 2008; it is a discontinued project with no activity for 6 years already; it has no maintainability, support nor active users' community nowadays. Therefore, CARMEN does not include the specified capabilities for this framework nor meets the requirements summarized at the beginning of this section.

Maybe the most popular framework nowadays for robotics is ROS (Robot Operating System) (Quigley et al., 2009). ROS is an open-source, meta-operating system for robots. It provides some services expected from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. ROS simulation capabilities are supported by Gazebo (Koenig and Howard, 2004). Gazebo is actually the simulation provider selected for this framework. Therefore, ROS and this framework share the same simulation capabilities. However, Gazebo is not interfaced through ROS; instead, the stand-alone version of Gazebo has been employed. The reason is ROS communications framework, although extremely useful for many projects and configurations, introduces the overhead of the messaging system, based on a publish/subscribe paradigm, which can accumulate hundreds of processes in complex systems. This is not a distributed system; its architecture is designed as independent subsystems and modules, but they all are in the same machine, as can be seen in Figure 6. Therefore, this overhead and delays due to the use of TCP communications and the management of publishers and subscribers by the ROS core is unnecessary for us.

Based on the above considerations, Gazebo (Koenig and Howard, 2004) has been selected as the simulation capabilities provider for this framework. It is an open-source multi-robot simulator for outdoor environments that allows 3D real-time scene rendering and simulation of rigid-body physics, necessary for rover navigation and control. Gazebo is a very well-known tool, having a broad, growing and very active community of users and developers. It has been out for more than 10 years, being widely used by the robotics community, so that it is a quite mature package. New versions, capabilities and bug fixes are continuously released. Gazebo has lately experienced a colossal development effort and sophistication process by OSRF (the Open Source Robotics Foundation), as it was recently selected by DARPA (Defense Advanced Research Projects Agency) as the GFE (Government Funded Equipment) robotics simulator for its last challenge, DRC (DARPA Research Challenge). It is available at no cost and its source code is open, meaning it is possible to perform any necessary modifications and extensions to adapt it to the concrete needs and requirements. Therefore, Gazebo meets the set of functional requirements and the criteria established for this framework stated at the beginning of this chapter.

Using Gazebo, a Mars-like terrain, including mounds, depressions, craters and rocks, has been created. Different texture maps can be used to cover the mesh, increasing the level of realism. Additional objects, such as rocks, can be added to the scene by creating its 3D model –mesh, and specifying attributes such as texture, position, orientation and size. Illumination conditions can be controlled -light direction and intensity, introducing shadows in the environment to create effects such as morning or evening light, Figure 4 shows two simulated Mars-like terrains with different rock distributions and illumination conditions.

A model of a planetary robot has been created based on the NASA's MER rover. Figure 3 shows the MER rover built by NASA/JPL and a simulated model of this rover created for Gazebo within the framework. Each piece of the rover is modeled independently, establishing links between parts and creating a complete solid body. Mobile elements, such as wheels or servomotors, are linked to other bodies by joints, where torque, controller gains, limits and direction of rotation can be established. It has boogie suspension, six independent wheels, four of them steerable, an IMU, a front low stereo camera for hazards detection and another one on top of a mast for navigation on a pan&tilt unit with two degrees of freedom.



a) b)

Figure 3. a) MER rover (courtesy of NASA/JPL) and b) its model created for Gazebo within the framework

Gazebo also includes models of some common sensors and devices, such IMUs, servo motors or stereo cameras, with configurable parameters such as field of view, resolution or stereo base. The 3D rendering engine incorporated within the simulator allows taking images of the environment from the rover's onboard cameras. Figure 4(a) shows an image of a rough terrain obtained from the framework taken with a 45º FOV (Field Of View) simulated stereo camera and (b) an image of a softer terrain with a different rock distribution and illumination conditions taken from a 75º FOV stereo camera model. Images are fed to the rover controller, which uses them to produce maps of the environment and compute suitable trajectories to be sent back to the simulator for execution. It makes possible closing the control loop, enabling the rover moving safely through the terrain so that autonomous navigation developments and approaches can be analyzed.

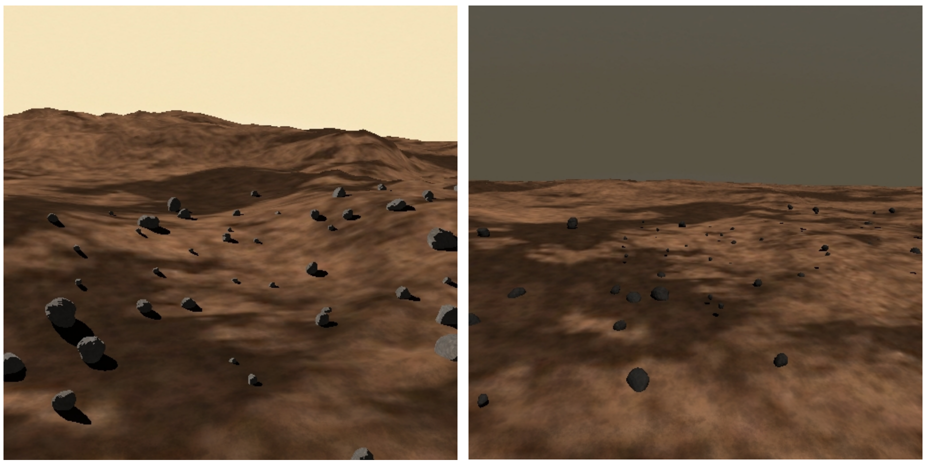


Figure 4. Simulated Mars-like terrains with different illumination conditions

Another important feature included within the Gazebo simulator is a dynamics engine which computes rigid-body physics, modeling the rover-terrain interactions and object collisions. If the rover is directed towards an obstacle, a collision will occur, blocking the rover's way and preventing it from moving forward in case the obstacle is heavier than the force the wheel's motors can produce.

As the main purpose of the framework is to be used as a software development support and research platform, easy and quick configuration of the system is crucial. The simulator allows setting environment-related values such as terrain relief and complexity, soil texture, light direction and intensity or gravity by a series of configuration files where parameters can be easily set. Model related features such as rocks' size, pose and orientation, rover' wheels size, chassis measures, joints, torques, gains, camera lenses' field of view and resolution among others can also be configured.

The alteration of any of these parameters have an important impact on the internal rover's software computations and external behavior, since the analysis, measure and characterization of these aspects are one of the main purposes of this framework. This parameterization allows experimenting with different mechanical designs and hardware characteristics and configurations as well as with algorithms behaviors and adaptations.

The simulator has some limitations though. It does not allow analyzing aspects such as meteorological influence, advanced mechanics and locomotion. Sophisticated wheel-soil contact forces cannot be modeled nor phenomena such as slipping or sinking. More information on rover slip management and prediction can be found in (Yen, 2008; Ward and Iagnemma, 2008; Iagnemma and Ward, 2009). However, it is important to emphasize that those are not the focus of this research work. The aim is to develop high-level autonomous navigation strategies, analyze their performance and validate them at the functional level. Central issues in the development of such strategies are the design and analysis of perception and stereo vision algorithms, particularly how they process the images and produce different maps and terrain representations. Map building, path planning, control algorithms, onboard decision-making techniques or sensor fusion techniques are within the focus of this research.

3.2.2 Control Center

The control center is a purposely developed graphical interface application to visualize data and measures. It receives rover's telemetry data and monitors its behavior and performance. Communication with the rest of the subsystems is by TCP/IP connections, both for receiving data and sending commands. The control center has two working modes: operation and debug. In the operation mode activity plans are sent to the rover, representing its daily activities. It includes tasks such as scientific instruments usage, resource management, a perception plan or sequences of preprogrammed moves for blind navigation, established by human operators. Currently, only autonomous navigation commands are sent, designating a concrete target location, a series of waypoints relative to the rover position or a direction to follow.

The main efforts so far have been put in debugging capabilities, where the control center receives real-time data from the rover as it navigates. This is not possible in operation mode, where communications constraints limit connections to short periods a day. However, in debugging mode, communication is permanently established in order to facilitate data monitoring and analysis. This way the rover's activities are monitored at all times, which is crucial for understanding what the robot is doing and what the onboard algorithms are computing.

Data received are stored in log files and kept within the control center for historical purposes and later analysis. The data stored includes periodical rovers' status, alarms and messages received asynchronally from the robot whenever an event occurs, data from sensors (such as IMU or encoders), images from cameras, intermediate results from the stereo process for debugging purposes, final, merged and intermediate maps generated, candidate trajectories evaluated for path planning, intermediate products in the mapping and path planning process for debugging purposes, localization estimation and multitude of traces along the processes for debugging purposes. All these data are stored with a timestamp in operational log files.

When using the control center in debugging mode, a replica of the terrain model used in the simulator is also shown within the graphical interface. Figure 5 shows the debugging view of the control center. A rover model is displayed at the location received by telemetry on a 3D terrain reconstruction. In debugging mode, communications are real-time, meaning the rover is monitored online, and the operator can follow in real-time where the rover is and the activities it is performing. In this mode, latency, bandwidth constraints or satellite passes are not simulated, as this mode is envisioned for developing and debugging purposes. In the lower part the captured images from the onboard cameras are displayed, as well as the computed maps. Three different trajectories are shown on the terrain displayed in the control center as the rover navigates: 1) the one computed by the rover, 2) the one actually executed (green trajectory in Figure 5) and 3) the one the rover perceives it has followed from its sensors' readings (red trajectory in Figure 5).

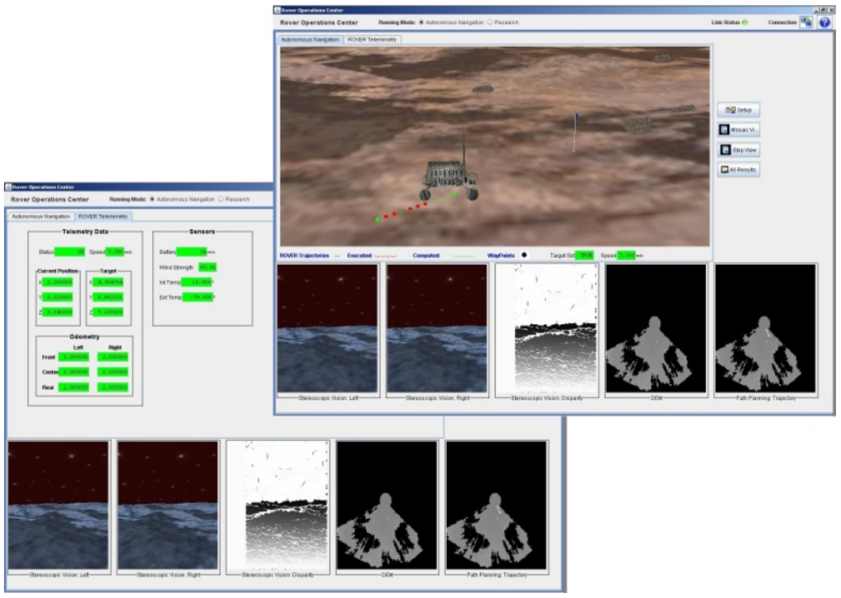


Figure 5. Debugging view of the control center showing captured images, computed maps and planned and executed trajectories on the terrain

Communication between the rover and the simulator is done by establishing a TCP/IP connection and invoking a series of functions, an API, provided by Gazebo. The simulator acts as a server and the autonomy software as a client. In real settings, instead of with the simulator, the robot control software connects with the sensors by physical ports and busses. In the following chapter it is described how the control software architecture includes a HAL (Hardware Abstraction Layer) that guarantees independence between the autonomy algorithms and the simulator, so that it is transparent whether data comes from simulated sensors or from real devices.

Communication between the rover and the Ground Station is supported by ICE (Henning, 2010), a high performance and efficient open-source object-oriented toolkit for distributed applications. It has its own IDL (Interface Definition Language), defining the data exchanged between systems. This middleware eases the communications protocol between both subsystems, especially in debugging mode, where the control station is permanently receiving data. In a real setting configuration, there is constrained radio satellite-satellite communication. The Hardware Abstraction Layer isolates the high-level autonomy algorithms from the low-level communication means, making it transparent whether TCP/IP or radio connections are used. As the focus of this work is the autonomous navigation approaches, features such as the satellite's latency, delays or bandwidth constraints have not been modeled, as they are considered out of scope at this stage, although they might be implemented in the future.

The integration of the different software components together to build up a system has been achieved intra and inter-system. Within a given subsystem, intra-system cohesion among components has been achieved developing wrapper classes around each package/library's native API and creating the necessary pieces to connect them and provide a homogeneous interface among components. Inter-system integration has been achieved creating communication interfaces, using both middleware and TCP/IP communications, following the client/server paradigm and creating a distributed system of independent modules.

The main component of the system, the rover's onboard software and autonomy algorithms, is described in detail in the next chapter.

Chapter 4

AUTONOMY FOR PLANETARY EXPLORATION ROVERS: ARCHITECTURE AND NAVIGATION

Robotic surface planetary exploration is a challenging endeavor, with critical safety requirements and severe communication constraints. Within a mission, autonomous navigation is one of the most crucial and yet risky aspects of these operations. Therefore, a certain level of local autonomy for onboard robots is an essential feature, so that they can make their own decisions independently of ground control, reducing operational costs and maximizing the scientific return of the mission. This chapter outlines two main issues: a) the design of a multilayer and modular onboard software architecture for the rover's control and b) a series of algorithms that implement a visual-based autonomous navigation strategy for robotic space exploration. These strategies and algorithms have been implemented from the ground up, partially supported by open-source packages and libraries. A set of experiments using the simulation capabilities of the framework introduced in the previous chapter, as well as a set of field testing experiences using a physical robot and real hardware are described. Results and algorithms' processing time are detailed. An increment of one order of magnitude has been experienced when the algorithms are executed in space-certified like hardware, with constrained resources, in comparison to using general purpose hardware. The work described in this chapter is based on Correal and Pajares (2011b) and Correal et al. (2014b).

4.1 Background

Onboard autonomy entails the set of procedures that shall be embedded on the robot to provide this with the necessary intelligence and capabilities to make its own decision, to deal with unknown environments and unexpected situations and minimize the dependency on ground control operators. The vehicle has to be able to travel from one location to another to take measures, samples and perform scientific experiments. Mobility is one of the most critical capabilities for an autonomous space exploration vehicle; if the vehicle drives into a too steep area, runs over a rock, a hollow, a rise or a soft sand area, to name some potential hazardous situations, it may tip over, run aground, get stuck or lose traction, and that may put the whole mission at risk, as nobody can get there to help the rover get out and keep going.

As stated before, given the communication constraints (delays, communication windows), a planetary exploration robot cannot be remote controlled in most of the cases. Therefore, alternatives have been developed in order to have the robot doing its job. The most common techniques are directed driving, where a detailed human-made plan is sent to the rover and is executed, and autonomous driving, where a set of goals and objectives are sent to the robot and it decides how to reach them by itself. There are benefits and drawbacks when using each of those techniques. In this chapter they are analyzed from the point of view of resources usage, perception, planning, execution and adaptation.

Regarding the resources usage, autonomous driving requires a more intense computation and sensor usage. This causes higher power consumption of CPU and onboard electronics and a longer execution time to reach the same goal than using directed driving. From the point of view of human resources, planning requires longer when done by humans operator than autonomously, in order to choose the appropriate waypoints when designing a blind drive for the rover, given the safety responsibility is placed on the operator. However, given the adaptability and learning human capabilities, this planning time can be dramatically decreased after some operations experience. That adaptability is also an important factor when facing new situations like new terrain types. While a human can adapt to it rapidly, an on-board software update involves a long development and testing process (Biesiadecki et al., 2005).

When planning a path to the goal based on vision, the perception capabilities of a human operator are far more developed than computer vision techniques, what makes the planning process much faster and reliable, given it's much easier to identify different terrain hazards. A computer vision system is limited in this sense, although the most common hazards like rocks and steps can be reliably detected with current technology. The human operator typically chose waypoints to avoid the most hazardous areas, thus taking advantage of the perceptual strengths of humans. However, if the goal of the rover is far away, the human operators are unable to plan the whole path from the actual position, based on the imagery sent back from the rover. It's needed to advance towards the goal and get new perceptions as it get closer. Thus, although human planning offers more reliability, it's also limited on the length of the path that can be built. The rover can be good detecting nearby hazards, but without a global planner there is a risk to get into dead-ends. A global and dynamic path planning is a clear necessity.

During the trajectory execution, when driving in directed mode, the rover has limited ability to deal with errors or uncertainty, like slippage. It must assume it hasn't deviate too far from the planned path so it won't encounter any hazard. The plan must specify alternatives and halting criteria. However, while driving autonomously the rover can respond online to accumulate errors or unexpected hazards.

Both driving methods, directed and autonomous, are complementary. The strongest point of the rover is the ability to close the loop and assess terrain not visible in the imagery available to the operators while planning the drive. On the other hand humans have a much more powerful perceptual abilities and adaptability to new situation and quick learning. A good selection of the right technique depending on the situation can lead to better overall performance.

The NASA's MER mission has been the most successful and relevant robotic mission to Mars so far. In the execution of this mission, a daily plan is elaborated from ground control and uploaded to the rover. Given the high cost and the criticality of planetary robotic missions, rovers have typically been operated in a very conservative way; in case the vehicle encounters a situation it cannot deal with, it just halts and waits for further instructions. Certain onboard autonomy is allowed only under concrete conditions of minimum risk; the most used method was to plan the first steps of a long traverse in a directed way, and let the rover continue autonomously until available time is exhausted or the goal is reached.

Given its importance and crucial role this mission has played in the history of space exploration, an overview of the mission and the autonomy capabilities of the rovers are given next as an analysis of the state of the art in this domain.

4.1.1 MER Mission

In January, 2004, the Mars Exploration Rover (MER) mission landed two rovers, Spirit and Opportunity, on the surface of Mars. The mission was scheduled for a 3-months period duration; however, one of the vehicles is still operating on the Martian surface, 10 years later. During that time period, the onboard surface mobility intelligence was used successfully to cover a distance of more than 40 km traversed combined by both rovers. It's expected the likely end of mission is insufficient power due to dust build up on the solar arrays.

4.1.1.1 Rover Description

The MER rovers were big and heavy robots. They are comprised of a 1.2 m high body and 168 Kg. Its chassis is of the type Rocker-bogey with 6 steering wheels of 25 cm of diameter each. The top speed is 5 cm/s. It communicates with ground control via an orbiter or Direct-to-Earth (DTE) antenna. During nominal operation, the rover has a power requirement of 100 W. It is battery powered, fed by a 1.3 m2 solar array, able to deliver a peak of power of 140 W. Regarding its processing capabilities, the onboard computer is an RAD6000 CPU, at 20 Mhz. with VxWorks. 3 Mb of PROM, 128 Mb of RAM and 256 Mb flash memory.

Its payload was comprised of a Pancam to provide high spatial resolution on the morphology of the landing site, a mast-mounted NavCam pair, front and rear HazCam pairs, a SunCam, which is a sun sensor used to determine global bearing, IMU and encoders for odometry and navigational purposes. About its scientific instruments they include among others an Alpha Proton X-Ray Spectrometer (APXS), Rock Abrasion Tool (RAT) for removing surface dust and weathering, a microscopic imager, a Mössbauer spectrometer to determine the properties of iron bearing materials and a Mini-Thermal Emission Spectrometer (Mini-TES) to obtain mineralogical information for rocks and soils surrounding the rover capable of detecting silicates, carbonates, sulphates, phosphates, oxides, and hydroxides.

4.1.1.2 Rover Autonomy

Several autonomous navigation capabilities were employed in space for the first time in this mission (Maimone et al., 2006a). Due to communication constraints, the MER rovers are commanded once per Martian day (with occasional nighttime communications for science activity). A commands sequence detailing the activities (images, arm positioning, etc.) is sent in the morning. At the end of the sol, the rovers send the data and images collected during that day to ground control, through the Mars orbiters. That data, representing the last known state of the rover at the termination of the sol activity, are then used for the operation team to plan the next set of science and mobility activities to be sent to the rover the next day. Those unsupervised plans last approximately for four hours. Once the rovers receive the plan, they will have to execute it on their own, without further instructions from Earth.

The MER operations approach is a form of adjustable autonomy via mixed modes of manual and autonomous commands for sequencing the rover as introduced before. The autonomy in this system is mainly associated with navigation rather than science or mission planning and scheduling. Autonomous navigation is commanded by specifying explicit surface coordinates to be reached using onboard sensing, perception, motion planning and execution. Mission operators provide a global path plan in the form of a series of waypoints that is executed and evaluated onboard the rover. If the ultimate goal is beyond the reliable field of view of the rovers' mast-mounted stereo navigation cameras data available on ground, the rover is commanded to drive blindly for the known part of the way and then switches to autonomous navigation to drive for whatever time remains in the day into areas never seen before in images sent to Earth. Blind drives can be several tens of meters. About 25% of the distance traversed on Mars by the rovers is driven autonomously, detecting and/or avoiding obstacles and processing images for position updates using Visual Odometry. So far the longest contiguous autonomous drive made by Spirit was 78 meters. And the longest made by Opportunity was 280 meters.

MER rovers use stereo image processing to gather geometric information about its surrounding terrain (Maimone et al., 2007). This is done by automatically matching and triangulating pixels from a pair of stereo-rectified images to generate 3D points representing the terrain. This technology was already used in the previous Pathfinder mission. However these rovers count with much more powerful CPUs than before that allow more complex and accurate computer vision algorithms.

The surface navigation system performs terrain hazard detection based on the images coming from the stereo vision system and analyzes the rover surroundings geometrically. It then builds a map representing traversable areas within the local terrain. It measures small patches of terrain to build and maintain a grid-based rover-centered map to determine where is safe for it to traverse through. Either the hazcams or the navcams can be used for autonomous navigation. Hazcams were designed with a wide field of view and narrow baseline in order to see more than the full width of the rover a short distance ahead, at most 3-4 meters. This is an important feature for obstacle avoidance and turning operations safety. The navcams, on the contrary, were designed with a narrower FOV and a wider baseline, so they can see further in order to verify traversability (Maimone et al., 2006b).

Next, a path planning algorithm calculates the best way to achieve the desired goal given the previously built map while avoiding hazards and obstacles. The algorithm developed to accomplish this procedure is called Grid-based Estimation of Surface Traversability Applied to Local Terrain (GESTALT) system (Biesiadecki and Maimone, 2006). It uses the previously created map, divided into cells of approximately 20 cm, to determine how safe the rover would be at each point in terms of tilt, roughness and step hazards.

Several driving modes were developed for this system: 1) *Directed Drive* executes a planned course without any onboard compensation for position or attitude drift. 2) *Terrain Assessment* drive looks for geometric hazards (e.g., rocks, ditches), but it does not measure any slip. 3) *Local Path Selection* drive corrects for heading changes and anything measured by Visual Odometry and *Terrain Assessment*, instead of just blindly following a *Directed Drive*. Visual Odometry updates the rover's position but it does not check for obstacles (Maimone et al., 2007). The selection of the mobility and navigation approach for a given traverse plan is determined by the engineers and mission managers based on the allocated driving time, amount of terrain visible, known hazards and level of uncertainty in rover position.

These three capabilities are available in any combination, providing even greater benefits of autonomy and higher level capabilities. For example, *Terrain Assessment* alone provides a useful safe or not safe indication, but combined with *Path Selection* enables fully autonomous driving around obstacles. Adding Visual Odometry would make possible to navigate around obstacles in slippery areas as well.

The *Path Selection* capability gives the rover autonomous decision and control authority to select its next drive direction, in contrast to *Directed Drive*, in which it follows a single predefined path. With *Path Selection* enabled, candidate motion paths are projected onto the traversability map and a weighted evaluation of the constituent grid cells is assigned to each path (Maimone et al., 2006a). To select from among multiple paths, waypoint path evaluations are assigned to all possible candidate paths according to how effectively each path would drive the rover toward its goal point. This is merged with the obstacle path evaluations when *Terrain Assessment* is enabled. Thus, many possible paths through the grid cell are evaluated for safety. These path-based safety evaluations, together with the goal location and current steering direction, are used to make the final selection of the next navigation step to execute.

The navigation step size is configurable by human operators at anywhere from 35 cm to 1 m or more depending on a variety of operational factors, including terrain difficulty and overall distance goals for the day, but nominally 50 cm, arc length is used. This distance is configurable and limited by the quality and extent of stereo range data, and conservatism of flight controllers responsible for rover mission operations. The rover drives a fixed distance along the selected path before stopping to acquire new images for the next driving cycle.

Typical computing time per cycle of this algorithm is around 70 seconds. On the MER flight processor, the stereo images processing algorithms can take 24 to 30 seconds per image pair to compute. During normal operations, at most 75 percent of the CPU time is available for autonomy software, but telemetry processing can sometimes reduce that number even more.

While the rover is driving, its peak speed is 5 cm/sec, but it is typically operated at less than that (3.75 cm/sec) for power reasons. With computing time, the median net driving speed was about 0.6 cm/sec (Maimone et al., 2006b).

The main restriction on the actual use of combinations of MER autonomous capabilities is processing time. The relatively slow speed of its space-qualified CPU, an architecture that prevents full benefits of the processor cache from being realized, and limited development time led to a system with very capable autonomy that can take minutes to process a single set of images.

In order to navigate from one location to another, a global path plan is provided to the rover in the form of a series of waypoints leading to the goal location. Each waypoint is reached through incremental execution of navigation steps decided by cycles of *Terrain Assessment* and *Path Selection*. This process is repeated until the goal location is reached. Waypoints and goals have associated position tolerances, i.e., radial distances within which the waypoint or goal is considered reached. This capability is encapsulated in a single rover command called Goto Waypoint, which directs the rover to drive until the estimated position is within a specified tolerance of its commanded goal location, or until a specified timeout period expires.

As the rover traverses and in order to perform successfully autonomous navigation, the onboard system must estimate and update its position and orientation. The orientation and attitude estimation is done thanks to the IMU, which gives high quality results. The position/location estimation is generally achieved via wheel odometry. However, in challenging and rough terrain this technique has some limitations. Due to slippage, sinkage and sliding factors, the wheel odometry can accumulate errors, leading to an increasing position uncertainty. It is necessary higher levels of sensing, perception, and autonomy to compensate for such errors. The system that MER rovers use for corrections in those situations is Visual Odometry.

The Visual Odometry, or ego motion estimation (Olson et al., 2000), capability enables the rover to update its current position and/or attitude by comparing locations of features found in stereo image pairs correlating and matching them, both in 2D pixel coordinates and 3D world coordinates. A maximum likelihood estimator is applied to the computed 3D offsets between the features to produce the final motion estimate. This process takes about 160 sec, and it only converges successfully if the imaged terrain has a sufficient number of detectable features. The previously described Goto Waypoint command can be configured to restrict autonomously-commanded motions to ensure sufficient overlap between successive images (nominally 60%).

During the operation time on Mars, there have been several MER software updates from Earth that have increased their capabilities and autonomy even further. The version used during the first stage of the mission had a very conservative approach regarding autonomy. It operated in a quite slow fashion due to it required imaging and terrain processing after each motion. It also checked for step obstacles within rover-sized disc, what made the rovers unable to autonomously drive over small mounds.

There has been an unprecedented enormous amount of engineering data generated by this mission, very relevant to the field of robotics, both Earth-based and planetary. The MER mission has been the first using techniques like stereo vision, local map-based obstacle avoidance, and visual odometry for autonomous rover navigation in a planetary exploration mission. Its results have contributed substantially to the success of the mission and paved the way for increased levels of autonomy in future projects. For next mission, it's foreseen the ability to navigate farther and safer. With the expected increased processing power of the onboard computers, next generation will benefit of more sophisticated approaches and algorithms.

4.2 Software Architecture Design

The development of intelligent capabilities for robotic systems is a challenging task. It involves the use of different technologies and the interaction between components in real-time, as well as the management of the uncertainty arisen from the interaction of the robot with its environment. Thus a robot will face different situations during the course of a mission and must react to perceived events by changing its behavior according to corrective actions.

A robotic system includes various subsystems coming from various fields of science and technology like mechanical engineering, automatic control, data processing and computer science. One of the main challenges designing complex systems is also the integration of low-level functional modules with high-level decisional ones. The software for navigation and control embedded on the robotic explorers has to be organized into a given architecture. The design and the integration of such diverse components require a consistent architecture which altogether guarantees the integrity of the complete system.

A control architecture provides a structured approach for design, specification, implementation and validation of a complex control system and its subsystems. It usually defines a paradigm or philosophy for structuring the problem and imposes constraints that guide the way the control problem can be solved. The goal of the control architecture is to organize coherently all subsystems so that the global system behaves in an efficient and reliable way to match the end-user's requirements.

Robot control architectures can be broadly characterized as deliberative (based on planning), reactive (direct link between sensing and actuation), or a hybrid approach. One of the most well-known robot control architecture is the subsumption architecture, introduced by Brooks (1986), where layers of control are built to let the robot operate at increasing levels of competence. This conventional design has been widely used in many robotic developments.

Typical robot and autonomy architectures are comprised of three levels - Functional, Executive, and Planning levels. Some architectures emphasize one area over the others and thus became more dominant in that domain (Nesnas et al., 2003). While some emphasize the planning aspects of the system (Firby, 1989; Estlin et al., 1999a), others emphasized the executive (Borrelly et al., 1998; Simmons and Apfelbaum, 1998), and others focus on the functional aspects of the system (Pardo-Castellote et al., 1998). There have also been efforts that aimed at blurring the distinction between the planning and executive layers (Fisher et al., 1997). Other architectures did not explicitly follow this typical breakdown. Some focused on particular paradigms such as fuzzy-logic based implementations (Konolige et al., 1997a) or behavior-based (Brooks, 1986; Arkin, 1989). There has also been considerable effort developing architectures that address multiple and cooperating robots scenarios (Parker, 1995; Estlin et al., 1999b; Werger and Mataric, 2001). Some other general robotic architectures can be found in different works (Huntsberger et al., 1998; Kurien et al., 1998; Martin et al., 1994; Martin-Alvarez, 1999; Washington et al., 1999; Charmeau and Bensana, 2005; Estlin et al., 2007).

Some adaptations of the layered approach have been also applied to the planetary rovers' control domain. That is the case of CLARAty (Coupled Layer Architecture for Robotic Autonomy) (Volpe et al., 2001; Nesnas et al., 2003), developed in a collaborative effort among several institutions such as California Institute of Technology's, Jet Propulsion Laboratory, Ames Research Center and Carnegie Mellon University among others; the LAAS (LAAS Architecture for Autonomous Systems) architecture (Ghallab et al., 2001; Ingrand et al., 2007) developed by LAAS/CNRS and supported by ESA and CNES; The ESA's Functional Reference Model (FRM) (Visentin, 2007); the ORCCAD (Open Robot Controller Computer Aided Design) architecture (Simon et al., 2006); the CNES' ANW (Autonomous Navigation Workshop) simulator architecture (Glette, 2004); the Control Architecture for Multirobot Outpost (CAMPOUT) (Pirjanian et al., 2000; Pirjanian et al., 2001; Aghazarian et al., 2004); or IDEA (Intelligent Distributed Execution Architecture) developed also by NASA (Muscettola et al., 2002).

The architecture presented in this chapter aims to control autonomous space robots from the low level control loop up to the mission planning and task execution. It is a domain-specific robotic architecture that develops a framework for generic and reusable robotic components that can be adapted to a number of heterogeneous robot platforms. The architecture remains fairly general in order to properly design, easily integrate, test and validate a complex autonomous system.

It has been designed with three main objectives: 1) reduce the effort to develop custom robotic infrastructure for every project or research endeavor; 2) facilitate the integration of new technologies onto existing systems; and 3) operate heterogeneous rovers with different physical capabilities and hardware architectures –both real and simulated- under a common framework.

The proposed architecture is based on a layered paradigm, with several hierarchical levels; some key levels crucial to ensure the software portability, reuse and hardware independence have been included besides the typical three levels previously introduced. The main objective is that most of the software developed can be ported to different robots with minimum adaptations, just by configuring parameters according to mission requirements; this is especially important for the upper levels, that shall remain unaffected even though some lower-level modules may change. It also provides a framework which aims to simplify the integration of new technologies and enable the comparison of various elements.

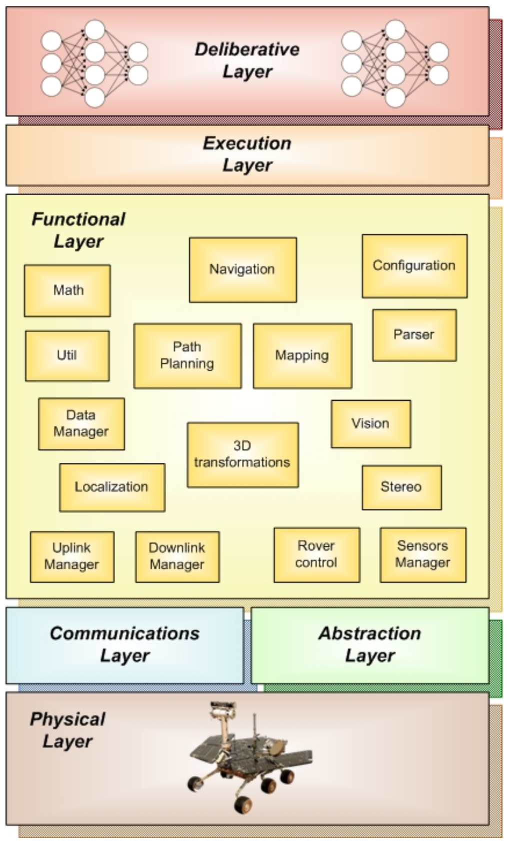


Figure 6. Onboard software layered architecture

A scheme of the designed architecture can be seen in Figure 6, indicating the layers the architecture is composed of, as well as the modules that make each layer up. The software has been implemented in C++, so that each module is generally a separated class.

This architecture consists of several different layers (physical, communication, abstraction, functional, execution and deliberative). The lowest layer is the *physical* level, where devices are electronically connected using the appropriate ports and buses. Next, *communication* and abstraction layers are placed on top. The former provides means of contacting with ground station, either using TCP/IP in debugging mode or satellite radio signals in operation mode.

The *abstraction* layer defines the various generalities of the system to adapt the general components to a variety of real or simulated specific devices; it encapsulates the access to underlying hardware and devices by means of predefined interfaces, hiding to the high-level software the concrete characteristics of the specific hardware. This layer facilitates maintainability and adaptability; in case a device has to be replaced by another one from a different manufacturer (e.g. camera), or just the module containing the device driver has to be updated. This way the autonomy architecture can be easily adapted to different vehicles, whether it is the MER rover model, MSL, Exomars or any other. The abstraction layer is supported by the open-source project Player (Gerkey et al., 2003), which provides a network interface to a variety of robot and sensor hardware, allowing writing control programs for a wide spectrum of robots. It has one of the most active users and developers communities. It later evolved into ROS (Quigley et al., 2009).

The *functional* layer includes all the basic built-in robot action and perception capabilities and provides the basic system functionality; it is where the actual autonomy algorithms lie. It does object-oriented system decomposition, so that this layer is made up of individual modules to achieve reusable and extendible components in order to provide algorithms for low and mid-level autonomy. These modules have a high level of cohesion and low interdependency, interconnected by establishing interfaces, though the use of APIs specifying a set of functions that accomplish specific tasks, to interact between them. As introduced before, this modularity facilitates maintainability, scalability and extensibility, where modules can be adapted, modified or even replaced in a transparent way to the rest of the modules, as long as it complies with the interface. For instance, the module that computes a height map is independent from the one that generates the 3D points cloud, which at the same time may come from a laser device, a stereo vision process or any other process or device; or a given stereo vision algorithms can be replaced by another one which better suit the needs, meet new requirements or just for performance analysis and comparison between algorithms.

A leading design driver has been modularity, as the software is used as a research platform more than certifiable flight software at this point. Each level is decomposed in independent modules with defined interfaces among them. Each module, or subsystem, encapsulates concrete functionality. Communications among modules, within the same layer or between adjacent layers, is done by invoking the set of functions defined in the interface of each module.

The *execution* layer is the interface between the *deliberative* and the *functional* levels. A light plan execution mechanism coordinates the execution of actions. It controls and coordinates dynamically the execution of the functions distributed in the modules according to the task requirements and the current state. Currently, a sequential control flow instantiates the corresponding modules in a sense-plan-act paradigm. Modules in the *functional* layer are activated by requests sent by this level, according to the task to be executed. They send data and reports upon completion to be used by other modules.

On top, the highest-level *deliberative* layer provides the system's high-level autonomy, which reasons and makes decisions about global resources and mission constraints; it controls the overall rover activities. The role of this decisional level is to create onboard actions plans considering mission constraints, time and available resources, monitoring operations, checking for plan deviation, dealing with unexpected problems, evaluating risks and generating contingency actions to adapt the plan whenever necessary. The decision layer receives information by querying the *functional* layer for predicted resource usage, state updates, and model information. This decision layer serves also as a monitor to the execution of the *functional* layer behavior, which can be interrupted and preempted depending on mission priorities and constraints. Currently, plans consist of lists of target locations sent to the rover from ground control. Our research has been focused on the functional layer.

4.3 Autonomous Navigation Strategy

There is a growing demand for autonomy capabilities for future planetary missions. Next missions are expected to be highly autonomous, where scientists on Earth will designate target destinations on the basis of images acquired and downloaded from the rover. In the case of the future ESA Exomars mission, it has been established that the rover must travel approximately 100 m per sol, a Martian day.

ESA currently has the necessity to develop its own infrastructure and equipment for robotic explorers, and researchers from academia and industry have to be ready to respond to the incoming calls to provide the necessary technology. The onboard intelligence for the explorer vehicle is currently under analysis (Joudrier and Elfving, 2009). Different alternatives have been studied; the author of this thesis was part of a team that participated in the evaluation of a software developed by CNES, the French Aerospace Agency, called EDRES, as a potential flight software candidate for the future ESA's rover, learning some important lessons from its analysis and evaluation (Odwyer and Correal, 2008).

This section is devoted to the design and development of an autonomous navigation strategy for robotic planetary exploration. The objective is to create the necessary infrastructure and software for a navigation strategy to be used as a baseline for prospective missions, such as the forthcoming ESA's mission Exomars, so it can be adapted, modified or extended according to the concrete needs, requirements and constraints of the mission.

The emphasis has been put in the autonomous navigation capabilities of the rover. Each time the robot moves the whole mission is put at risk. The navigation process is initiated when a daily plan containing scheduled actions, or a navigation command when working in debugging mode, is received from ground control. The plan consists on a series of locations the rover shall visit and actions to perform scientific experiments and take measures with onboard instruments. The safety and integrity of the vehicle directly depends on the path the rover decides to follow. Therefore safe trajectories must be computed to guarantee avoidance of obstacles and prevent hazardous situations such as tipping over, getting stuck or losing traction.

To achieve this, the robot must perform a series of tasks. The navigation system initiates a perception-mapping-path planning-navigation iterative cycle, see Figure 7: 1) *perception*: the rover must perceive the environment making use of the appropriate sensors, getting the necessary data; this is usually done by using stereo cameras and sensors such as IMUs and encoders; 2) *mapping*: an internal representation of the external world must be created, usually in the form of a map representing the surroundings of the robot; 3) *path planning*: the map created in the previous step is then used to compute safe and suitable trajectories that will be executed in the subsequent step; 4) *navigation*: so the robot is able to get to the location it has been commanded to following the path it has computed as a function of the perceptions it made of its environment. Each of these tasks is accomplished by the appropriate subsystem, described in following sections in detail.

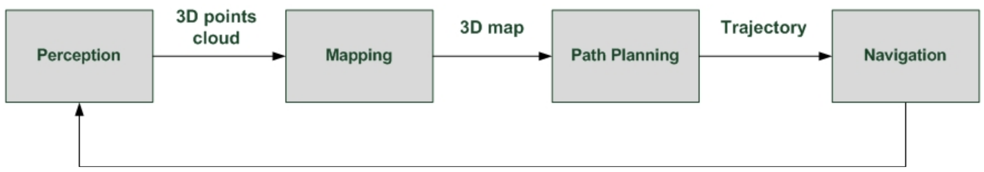


Figure 7. Diagram of the rover’s autonomous navigation processes

Locations may be more or less apart from each other and a complete route to the commanded location cannot be compute in most cases, so several navigation cycles may be necessary to reach a given destination, decomposing it in waypoints, one per cycle. In each cycle, the rover plans and navigates a certain distance, according to the configured planning and navigation distances, described below.

The navigation strategy presented in this section is somehow similar to the NASA/JPL approach created for the MER (Crisp et al., 2003) mission, which rovers have been operating on Mars for more than 10 years and have travelled above 40km. However, it is important to emphasize all these subsystems, modules and software introduced has been developed from the ground up. Even though agencies like NASA have a leading role in robotic space exploration, with years of experience and a number of successful missions accomplished, the technology and tools developed so far by these agencies -frameworks, simulators, tools, libraries or software- is proprietary to each institution/agency, not available to other institutions or to the research community. Therefore, every piece of software has been started from a blank page, as there are not any previous libraries or software from these national agencies or associated research group that may have been released to start form. Actually, the navigation strategy presented in this chapter introduces important differences and contributions with respect to the NASA/JPL's approach; they are indicated when describing the approach in the corresponding section.

All these modules have been entirely developed using C++ language, some pieces of pseudo-code have been included for clarification; although some functionality, mainly the stereo vision related part, partially relied on the use of the OpenCV library (Bradski and Kaehler, 2008). This work has been also supported by the framework introduced in the previous chapter, purposely created to assist in the development of this navigation strategy; it has facilitated the testing, functional analysis and validation in the initial phases, without the need of an actual Martian rover and mars-like terrain, reducing the costs and speeding up the development process, as well as paving the way and smoothing the transition for field testing and final validation.

4.3.1 Perception Subsystem

Perception is a crucial step in the navigation process, see Figure 7. The effectiveness of the perception process has a strong impact on the subsequent map creation, terrain assessment and path computation processes, which directly affects the rover's safety. A pair of cameras -a stereo system- mounted on the robot, usually in a mast to get a higher view, is used to obtain 3D information from the environment. They are arranged horizontally separated by a given distance to obtain two differing views of a scene, in a manner similar to human binocular vision. By comparing both images, relative depth information can be obtained, in the form of disparities, which are inversely proportional to the distance to objects. The concept of disparity and how it is computed is explained in more detail in section 5.1 Stereoscopic Vision.

A sequential process has been designed and implemented to be included within the perception subsystem of the rover. This is the process embedded in the module identified as *vision* in the scheme of Figure 6. It consists on the next steps and processes: 1) calibration, 2) image rectification, 3) pre-processing, 4) matching, 5) post-processing and 6) triangulation, see Figure 8. As indicated previously, the development of the algorithms included in this subsystem has been partially supported by OpenCV (Bradski and Kaehler, 2008), a library of programming functions for real time computer vision, very well-known and widely used by the research community.

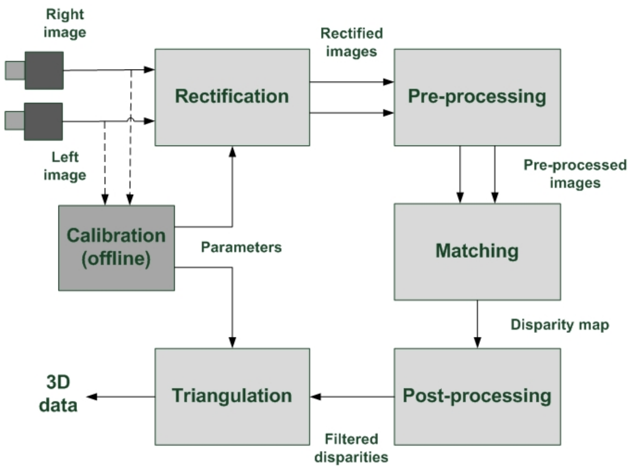


Figure 8. Diagram of the rover's stereo vision system

A previous offline calibration process obtains the intrinsic and extrinsic parameters of the cameras like focal length, principal point or lens' distortion model in case these are not provided by the manufacturer or just to experimentally check the values. Calibration is important for accuracy in 3D reconstruction. These values will be necessary later, in the triangulation phase to obtain depth data. Calibrating stereo cameras is usually dealt with by calibrating each camera independently and then applying geometric transformation of the external parameters to find out the geometry of the stereo setting. The method followed for calibrating the camera is the one proposed by Tsai (1986). The method is based on the knowledge of the position of some points in the world and the correspondent projections on the image. It required the camera to be pointed to a calibration grid, usually a chessboard-like pattern, in different positions and orientations, see Figure 9. The first step is to solve a linear equation to find the extrinsic parameters R, T, being R and T the rotation and translation matrices of the cameras with respect a reference system, and then perform a nonlinear optimization process to obtain the origin of the coordinates in the reference system. The procedure can be iterated in order to improve accuracy.

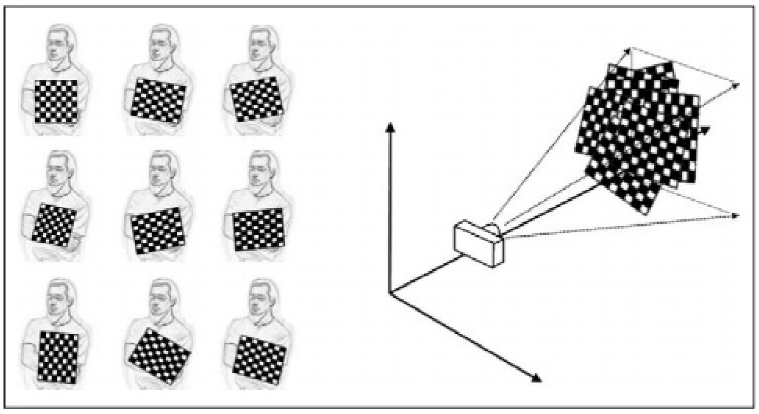


Figure 9. Scheme presenting the calibration process using a calibration object named chessboard

The *image rectification* step ensures both images are vertically aligned, so that the different features mapped in the scene coincide horizontally -lines of the right and left images -with each other. Images shall be previously rectified, see Figure 10. This procedure optimizes the pixel matching process and save computing time during the depth determination process, by forcing the potential correspondence of a given pixel to be found within the same line on the other image, called the epipolar line. When working in simulation with synthetic images, this rectification process is unnecessary as they don’t have distortions caused by camera lenses or misalignment, unless such distortions are specifically introduced.

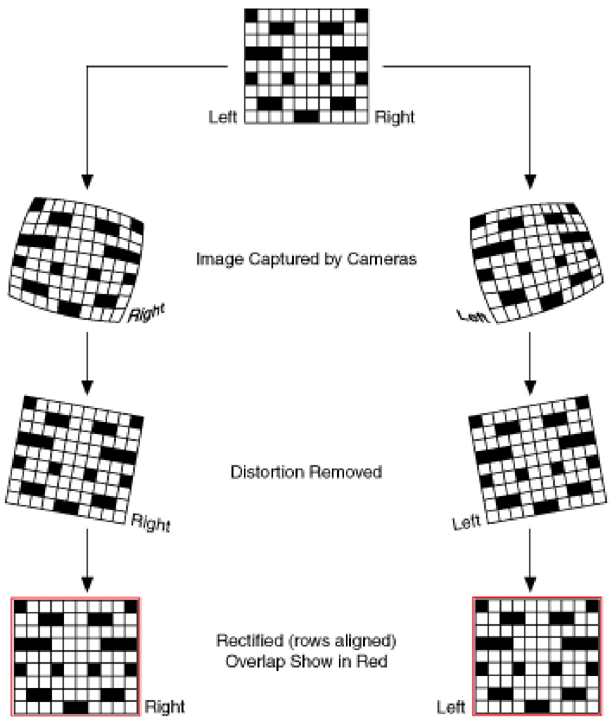


Figure 10. Scheme presenting the rectification process (National, 2013)

In a previous pre-processing procedure, the input images can be normalized to reduce lighting differences and to enhance image texture (Bradski and Kaehler, 2008). This has been done by running a window of a given size -5-by-5, 7-by-7 (default), until 21-by-21 (maximum)- over the image. The center pixel I_c under the window is replaced by min[max(I_c – I , – I_{cap}), I_{cap}], where I is the average value in the window and I_{cap} is a positive numeric limit whose value can be established -30 by default. It can be also done by any other normalization method, Laplacian of Gaussian for instance, which runs a peak detector over a smoothed version of the image.

Pixels from both images are then matched. As a result, a list of disparities, indicating the difference in position of each feature in one image (right) with respect to the other (left), is obtained. Those disparities are then transformed to distances –depth- by triangulation, explained later in more detail in section 5.1 Stereoscopic Vision.
Initially, the block matching (BM) algorithm (Konolige et al., 1997b) has been selected to process these images, given its efficiency and efficacy, as shown in the results obtained from the experiments, and detailed below. This is the matching algorithm embedded in the

module identified as *stereo* in the scheme of Figure 6. It is a dense method, trying to maximize the number of correspondences matching each pixel of the pair, which subsequently is used to create a depth map. This algorithm finds only strongly matching (high-texture) points between the two images. Thus, in a highly textured scene such as might occur outdoors in a forest, every pixel might have computed depth. In a very low-textured scene, such as an indoor hallway, very few points might register depth. Moreover, this algorithm is currently implemented as part of the standard and widely-used OpenCV library (Bradski and Kaehler, 2008).

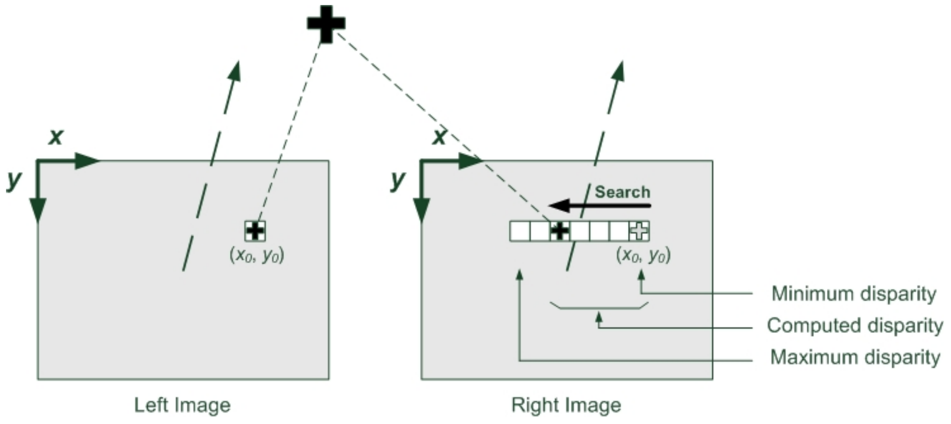


Figure 11. A right-image match of a left-image feature must occur on the same row and at the same coordinate point or to the left, between the minimum and maximum disparity search range

Correspondence is computed by a sliding a small windows applying SAD (Sum of Absolute Difference) to find matching points between the left and right stereo undistorted, rectified images. For each feature in the left image, the corresponding row in the right image is searched for a best match. After rectification, each row is an epipolar line, so the matching location in the right image must be along the same row (same y-coordinate) as in the left image; this matching location can be found if the feature has enough texture to be detectable and if it is not occluded in the right camera's view. If a pixel's coordinate for a given feature in the left image is (x_0, y_0) then, for a horizontal frontal stereo system arrangement with parallel optical axes, the match (if any) must be found on the same row and at, or to the left of, x_0; see Figure 11. With such geometry, x_0 is at zero disparity and larger disparities are to the left. Large disparities represent closer distances. For cameras that are angled toward each other, the match may occur at negative disparities (to the right of x_0). The search is bounded by a range, established by a minimum (usually 0) and maximum feasible disparity values. The computed disparity will be a value within such range. Reducing the number of disparities to be searched can help to cut down computation time by limiting the length of a search for a matching point along an epipolar line.

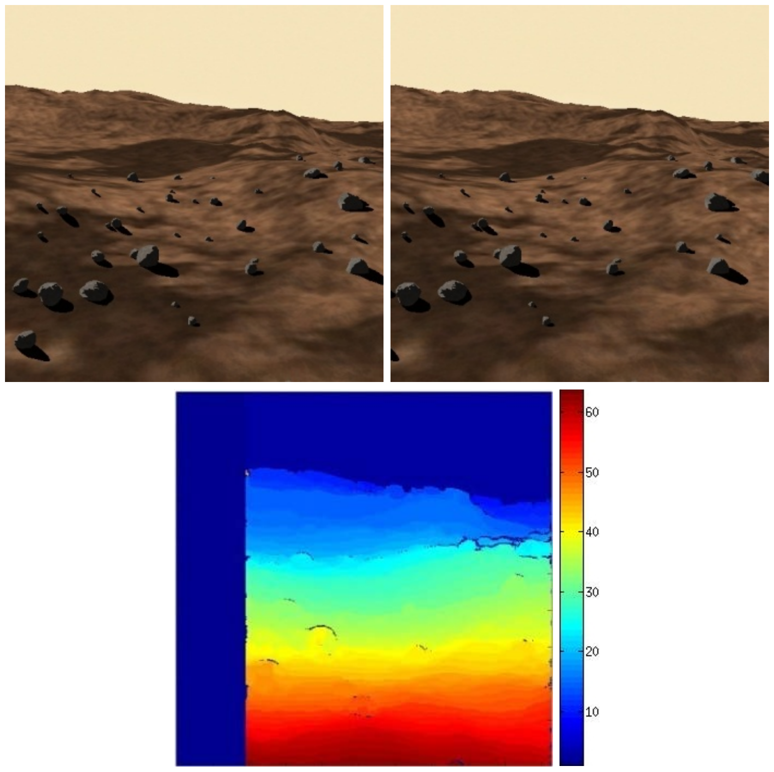


Figure 12. Disparities computation for a pair of images obtained from the simulator

The block matching algorithm (Konolige et al., 1997b) has been selected at this point as pretty good results are obtained when processing synthetic images generated from the simulator. Figure 12 shows a pair of stereo images (left and right) captured from the robot's onboard cameras, in the simulated environment, and the computed map of disparities using this algorithm. However, given the modularity and flexibility in the design of the software architecture, the algorithm embedded into this module, or any other, could be easily replaced by another one in case a new approach or version is to be employed or the algorithm it not appropriate anymore, in case the nature of the input images or the operational conditions change. This actually happened when the system was ported from the simulated to the real world and images with different characteristics were employed, as described in section 4.4.2 Field Testing, later in this chapter.

After correspondences computation, a *post-filtering* phase eliminates false matches, performing processes such as uniqueness check, texture evaluation and speckle filtering, overcoming random noise and avoiding speckle -local regions of large and small disparities. The uniqueness check process filters out wrong matches by checking the computed correspondence value in relation to a configurable uniqueness ratio (e.g.: 12), so that (*match value – minimum match value*) / *minimum match value* < *uniqueness ratio*. Regarding texture evaluation, to make sure there is enough texture to overcome random noise during matching, a limit is established on the SAD window response such that no match is considered whose response is below an established texture threshold (e.g.: 12). As for speckle filtering, block-based matching has problems near the boundaries of objects because the matching window catches the foreground on one side and the background on the other side. This results in a local region of large and small disparities called speckle. To prevent these borderline matches, a speckle detector can be set over a speckle window (ranging in size from 5-by-5 up to 21-by-21). As long as the minimum and maximum detected disparities are within an established speckle range, the match is allowed.

Figure 13 shows the effect of this post-filtering phase applied to the disparity map obtained using the block matching algorithm with a pair of images obtained from a simulated environment. In Figure 13(c) it can be appreciated some erroneously computed correspondences in the left-bottom corner. They actually do not correspond to any area of the terrain to be mapped, but to part of the robot's chassis seen from the cameras, that should not be taken into account when performing a 3D terrain reconstruction. As a result, these errors are removed, Figure 13(d).

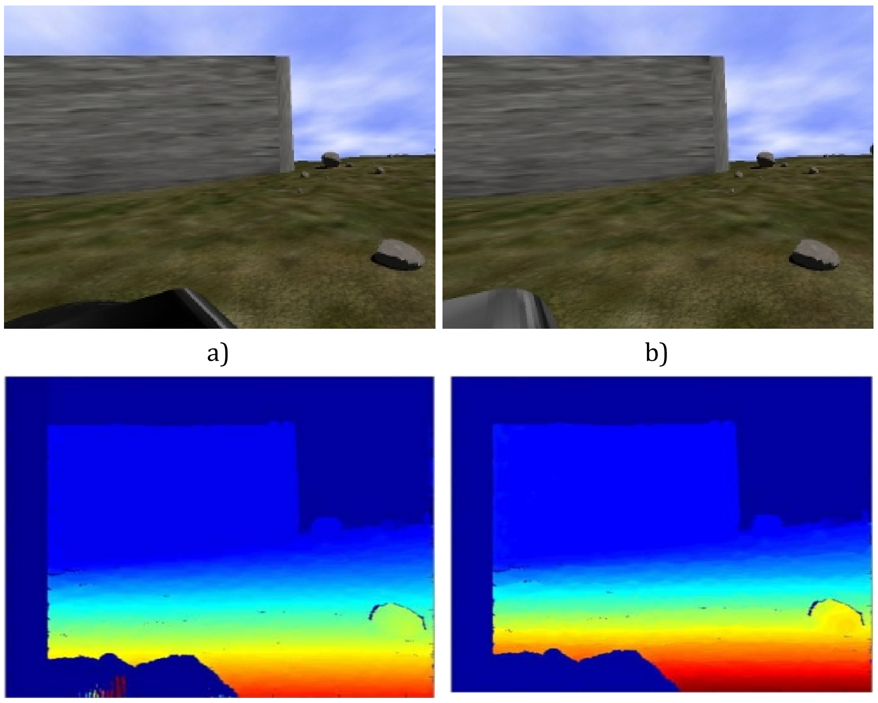


Figure 13. Disparity filtering based on clusters, a) before and b) after the process

In the last step, knowing the geometric arrangement of the stereo-based system, depth data is derived from the obtained disparities by triangulation. Each pixel is translated into depth, getting Z values, which represents the third dimension in the scene. The result is a 3D points cloud representing the environment from the camera point of view. Then, a series of geometric transformations, taking into account the rover pose obtained from an onboard IMU and the pan tilt head position, translate these points to the rover reference system for 3D terrain reconstruction. More details about this process can be found in sections 5.1 Stereoscopic Vision and 6.6.2 Automatic Image Processing Modules.

Algorithms included in this subsystem have been designed to be parametric and easily configurable. A configuration file allows establishing parameters and values so that many different configurations can be quickly set to test and evaluate its impact and performance on the overall navigation process. Some parameters for the perception process are camera's resolution, focal length, horizontal and vertical fields of view, offset between perceptions as well as parameters for preprocessing and normalizing the images, the matching process and filtering the obtained disparities in the post-processing phase.

4.3.2 Mapping of the Environment

The result of the stereo process is a disparity image, representing correspondences –matched pixels and features- in both images of the pair. This image is then reprojected to 3D space. This process transforms a single-channel disparity map to a 3-channel image representing a 3D surface. That is, for each pixel (x, y) and the corresponding computed disparity d=disparity(x, y), it computes:

$$\begin{pmatrix} X \\ Y \\ Z \\ W \end{pmatrix} = Q * \begin{pmatrix} x \\ y \\ disparity(x,y) \\ 1 \end{pmatrix} \tag{1}$$

where Q is a 4x4 perspective transformation matrix in the form:

$$Q = \begin{pmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 0 & f \\ 0 & 0 & -1/b & (C_x - C'_x)/b \end{pmatrix} \tag{2}$$

where f is the camera’s focal length, b is the stereo base –separation between both cameras of the stereo pair- and C_x and C'_x are the principal point x coordinate in the left and right image respectively. The 3D coordinates are then (X/W, Y/W, Z/W). As a result, a cloud of 3D points is obtained, usually containing tens of thousands of points, one for each correspondence found.

However, a list of 3D points is not an appropriate method of representing the environment for later terrain assessment and trajectory computation. A more appropriate representation is a height map, similar to the digital elevation model (DEM) obtained with instrumentation onboard aerial platforms. In our context, this map is to be represented as a rover-centered top-view grid of squares with height data representing the terrain's surface, including any obstacle. This is the process embedded in the module identified as *mapping* in the scheme of Figure 6 and Figure 7.

To build this map as a function of the set of 3D points derived from the imagery captured from the onboard cameras, the cloud of 3D points resulted from the application of equation (1), which are expressed with respect to the camera's coordinates system, has to be transformed into coordinates in the rover's reference system, see Figure 14.

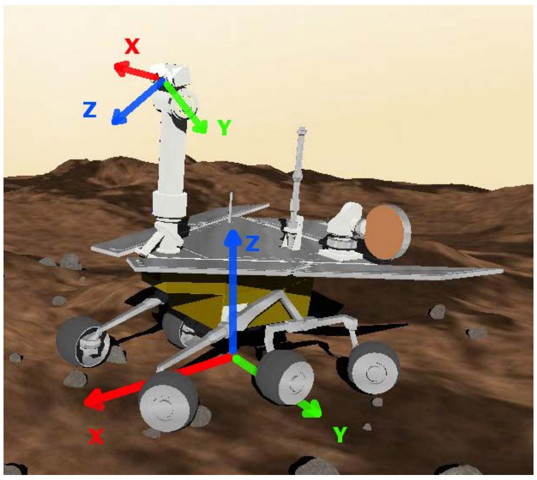


Figure 14. Camera's and rover's reference systems

This transformation has to take into account the rover's attitude, obtained from the onboard IMU, and camera position –pan/tilt- when images were captured. It is done by:

$$P_R = P_C \ * P * R \tag{3}$$

Where P_R is the 3D point transformed to the rover's reference system, P_C is a 3D point expressed with respect to the camera's reference system, R is a transformation matrix to match axes from both reference systems, detailed below, and P is the perspective matrix, computed as the composition of rotations about the three axes plus a translational vector:

$$ROT = R_X * R_Y * R_Z \tag{4}$$

where:

$$R_X = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(r_p) & -\sin(r_p) & 0 \\ 0 & \sin(r_p) & \cos(r_p) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{5}$$

$$R_Y = \begin{pmatrix} \cos(r_y + c_p) & 0 & \sin(r_y + c_p) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(r_y + c_p) & 0 & \cos(r_y + c_p) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{6}$$

$$R_Z = \begin{pmatrix} \cos(r_r) & -\sin(r_r) & 0 & 0 \\ \sin(r_r) & \cos(r_r) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{7}$$

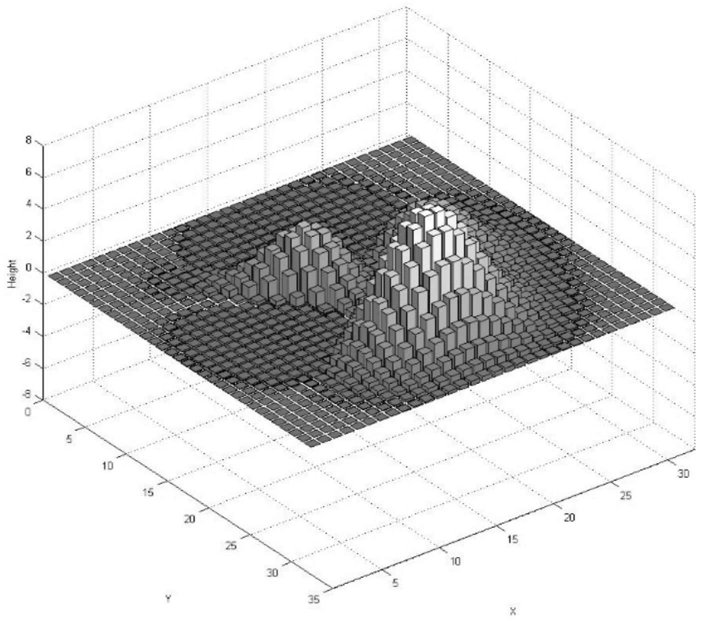
being r_r, r_p, r_y the robot's roll, pitch and yaw angels respectively and c_p, c_t the camera's pan and tilt angles respectively when the images where captured. The fixed translation from the camera's to the rover's reference system origin is also included into the perspective matrix:

$$P = \begin{pmatrix} & ROT & & \\ & & & \\ T_X & T_Y & T_Z & 1 \end{pmatrix} \tag{8}$$

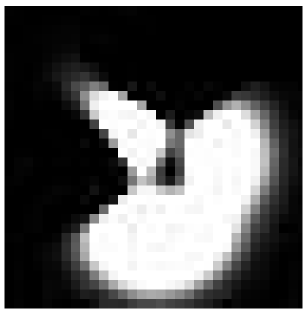
A last transformation is required to match the camera's reference system axes with the rover's, see Figure 14:

$$R = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ -1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \tag{9}$$

At this point, the height map representing the terrain can be built. This height map, a grid, initially empty, is filled with information derived from the transformed 3D points cloud, see equation (3). For each 3D point (x, y, z) in the cloud, its corresponding location in the map –grid- is computed as a function of (x, y) and the map's resolution, and then the corresponding cell is assigned the height value –z- of the point, see Algorithm 1. This height map can be thought of as a chessboard, where each square –cell- encodes a numeric value, representing the height/elevation of that terrain's portion, Figure 15. It actually is a discretization of the terrain for a convenient representation.



a)



b)

Figure 15. Discretization of the terrain for internal representation: a) a grid where each cell encodes the terrain' elevation in that area, b) usually represented as a gray-scaled image

When building a height map from the cloud of 3D points, it may happen there might be different height values computed for the same grid cell. This is especially true for the closest areas. The smaller the resolution of the map -less number of larger cells- the more points are used to compute the height associated to every single cell. On the other hand, for farther areas there is usually not enough information to fill every cell, resulting on empty areas in the grid. This is also the case for parts of the scene behind rocks or elevations, creating occluded areas. In cases when several computed height values belong to the same cell, different strategies to fuse data have been proposed to solve those ambiguities. These strategies are: 1) compute the average height from the set of data -3D points- belonging to the same cell in the map; 2) keep just the last value, replacing the stored height value by data from the 3D point currently being processed in case it provides information to the same cell; 3) keep the highest value, consisting in taking the maximum height value from the set of 3D points contributing with data for each given cell, 4) take the lowest height value, equivalent to the previous strategy but keeping the minimum height value, or 5) keep the largest magnitude, consisting in keeping for each cell in the map the largest magnitude from among the set of height values of the 3D points contributing to the same cell, either positive -elevation- or negative –depression. The latter is the default strategy, which is the most conservative one. This is the way we deal with uncertainty, by computing an average height value from the set of points that geographically belongs to the same cell according to the established data fusion strategy introduced previously.

```
For each point P in the 3D points cloud
      Determine the cell (x,y) the point P belongs to
      Z = height value from point P
            If (x,y) are within map's limits
                  If Map(x,y) is empty or criterion == KEEP_LAST
                       Map(x,y) = Z
                  Else if criterion == AVERAGE
                       Add Z to the temporal sum for Map(x,y)
                       n(x,y) = n(x,y)+1  // to later compute average
                  Else if criterion == HIGHEST and Z > Map(x,y)
                       Map(x,y) = Z
                  Else if criterion == LOWEST and Z < Map(x,y)
                       Map(x,y) = Z
                  Else if criterion == MAXMIN and abs(Z) > abs(Map(x,y))
                       Map(x,y) = Z
```

Algorithm 1. Pseudo-code for the height map building process

Figure 16(a) shows the result of building a map from the 3D points cloud following the Algorithm 1. In this figure, each pixel corresponds to a map's cell. As indicated before, these are top-view rover-centered maps, representing the terrain shape storing height data on each cell; the color encodes the actual height value. From a close-up view, it can be seen as the example shown in Figure 15, which is a 32x32 grid; but these height maps can be in the order of 2000x2000 to represent an area of 10x10 m with a 5 cm resolution. It can be appreciated there are many gaps and areas with no information (black areas). These void areas have a significant influence and a negative impact in the subsequent path planning process; gaps will be taken as unknown areas and therefore any candidate path going though, or close, to these areas will be discarded for security reasons, as the path planning subsystem is usually configured very conservatively.

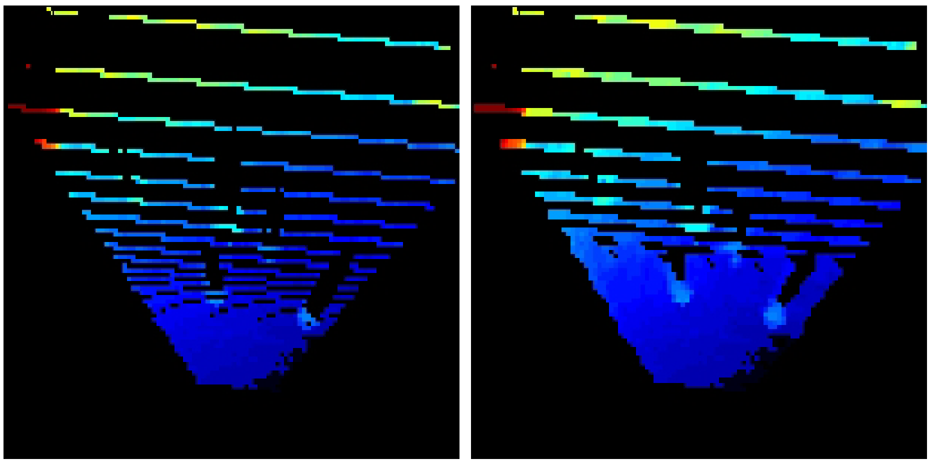


Figure 16. a) Height map built from a single perception and b) final map after the interpolation process

To minimize its impact, a post map building linear interpolation process has been designed. It tries to compute estimated height values for some of the empty portions of the map, see Figure 17. It is based on cell's neighboring values. The process, which pseudo-code is shown in Algorithm 2, consists in the next: for a given empty cell in the map, check if its neighboring cells have height data; it can be configured to check the four closest or the eight closest neighbor cells. Data from surrounding cells are summed up and averaged to compute a mean height value provided at least a minimum number of neighbor cells, configured by the operator, have valid values to contribute to the height estimate for this given cell. In such a case, the cell is updated with the computed value. This process can be repeated a number of times on the same map, also configured by the operator, achieving a growing-area or blank-shrinking effect. The proposed approach is configured to take into account the 8 closest neighbors' values, with at least 2 of them containing valid values to obtain an estimation, and repeat the process 3 times.

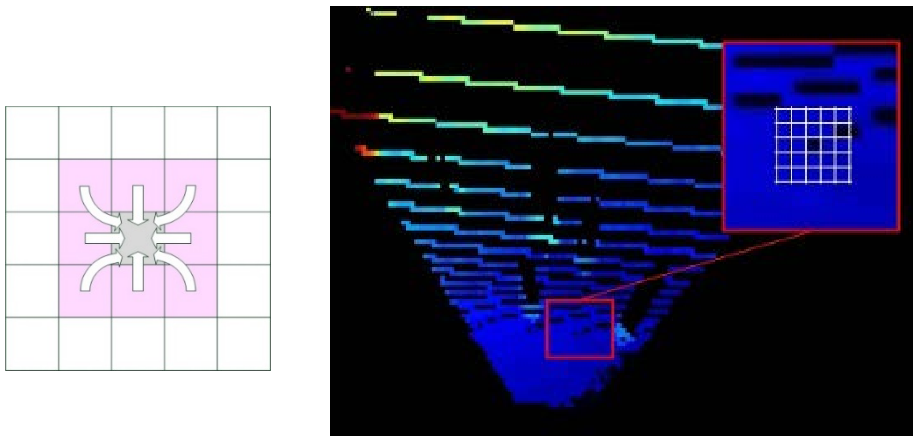


Figure 17. Interpolation process to fill gaps in the map based on neighboring cells' values

As a result of the interpolation process, a more completed map is obtained, shown in Figure 16(b). It can be appreciated some gaps have disappeared; they have been filled out with height information from neighboring areas. This is especially true for the area closest to the robot, which are the most crucial portion of the map as the candidate paths leave from the robot's current location. In these figures just a portion of the whole local map is shown, the part where there is data available, corresponding to the terrain in front of the robot appearing in the images just captured from the cameras in the previous perception stage. In these maps, the rover is located in the cone's vertex of the mapped area; in the bottom-center of the images shown in Figure 16. Now, the path planning process performed on this interpolated map will have more chances to produce a larger number of candidate paths to be evaluated, and therefore the whole process will be more effective, as the finally selected path will be chosen from a broader set of candidates.

```
For the number of times $N$ configured (default $N$ = 3)
    For each cell $c$ in the grid map
        For each neighbor $n_x$ of $c$ (default $x$ = 1..8)
            If $n_x$ has valid data
                estimatedValue = estimatedValue + value of $n_x$
                numNeigbors + 1
        if  numNeigbors > Min. number of neighbors configured (default 2)
            c = estimatedValue / numNeigbors
```

Algorithm 2. Pseudo-code for the height map interpolation process

As indicated previously, maps are built from the set of 3D points obtained as a result of the perception process. Cameras have a given horizontal and vertical field of view, so just a portion of the environment can be observed from the rover point of view. For some cameras, the data derived from the observable portion of the terrain may produce a too narrow map that may be inadequate to compute suitable trajectories to reach a given location, as large portions of the map will remain with no information, marked as unknown areas; any candidate path partially traversing an unknown area will be automatically discarded. A wider map can be obtained by performing more than one perception at each navigation cycle. As the stereo camera is mounted on a pan tilt head with two degrees of freedom, a configurable offset between perceptions can be established. A certain overlapping between images from different perceptions is desirable to avoid leaving unseen areas in between.

The next example can illustrate how this perception strategy works: consider having a camera with 90º HFOV (Horizontal Field of View) for instance; if the perception strategy is configured so that the rover performs two perceptions at each navigation cycle pointing the camera with an offset of ±35º from the frontal line for each perception, almost a complete view of the front terrain would be obtained. In the case of the previous example, a 160º view of the frontal area is obtained and therefore can be mapped. It is sketched in Figure 18. Yellow lines represent 0º and ±90º. The blue area is the portion of the terrain captured when moving the camera 35º left from the front line; in that case a view of the area comprised between 80º and -10º is obtained. The red area is the portion of the terrain captured when moving the camera -35º right from the front line; in that case a view of the area comprised between 10º and -80º is obtained. To sum up, a 160º view of the terrain can be captured for subsequent mapping with a 20º area of overlapping –the purple area- to avoid gaps and unknown areas in between.

Therefore, more than one perception is typically performed on each navigation cycle to build a local map. It is important to differentiate between local and global maps. A *local map* is the one produced by the rover as a result of the perception process at each navigation cycle – comprising one or more pair of images, as indicated before, see Figure 18. As a result, the portion of the terrain visible from the rover’s camera from the robot’s current location is mapped. On the other hand, the *global map* is the map of the area maintained by the rover, built from the integration of the set of local maps created in previous navigation cycles.

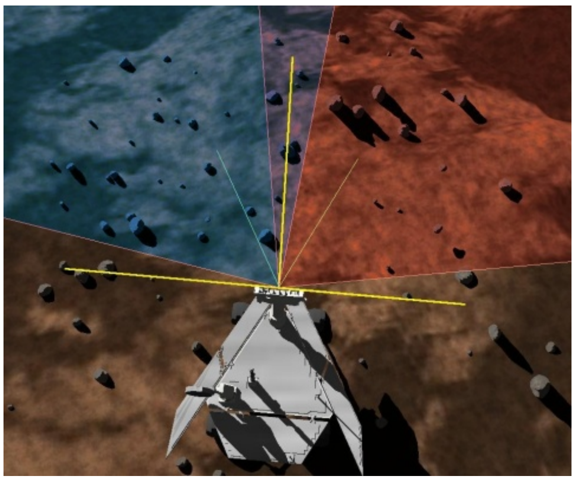


Figure 18. Area captured by two perceptions (red and blue) with a certain overlapping (purple region). Green lines represent direction of perception

Just a relative portion of the mapped environment around the rover's current position is kept in the map, forgetting old and distant data from farther regions the rover has previously traversed. This is because two main reasons: 1) memory constraints, a rover, as any embedded system, will not usually have enough space to store everything; maps can become gigantic after weeks or months of navigation. Therefore, just a given area around the rover is maintained, discarding information from farther areas; and 2) the rover follows an exploratory approach, discovering new regions as it navigates; usually the rover will be always moving forward, and rarely going back to areas it has already traversed. Therefore, the maps of these far regions already traversed can be forgotten -erased from the rover internal memory-, although they can be sent to Ground Control to be kept.

A trade-off between the global map's size and resolution must be made. The larger the area to be kept, the more memory space will be needed to store the map. The same happens with the map's resolution -the number of cells in the grid map-, representing the squared size of the terrain chunk represented by each cell. These values are configurable; nominally the area maintained in the global map is 10x10 m with a 5 cm resolution -cells' size.

As indicated previously, the global map is built from the integration of the set of local maps created as the rover moves forward and new areas are discovered. Therefore, the map has to be updated with information from new perceptions; new data is merged with previous available information. To accomplish this task, a merge maps process has been designed and included within the module identified as *mapping* in the scheme of Figure 6. It is described in Algorithm 3. *mapSrc* represents the new local map while *mapDest* represents the global map.

```
mapSrc  // source map to be merged
mapDest // source map to be merged and resulting merged map

For each cell (x,y) in mapDest
    If criterion == SRC_PRTY and mapSrc(x,y) is NOT empty
        mapDest(x,y) = mapSrc(x,y)
    Else if criterion == DEST_PRTY AND mapDest(x,y) is empty
        mapDest(x,y) = mapSrc(x,y)
    Else if criterion == SAME_PRTY
        If criterion == AVERAGE
            If mapDest(x,y) is empty
                mapDest(x,y) = mapSrc(x,y)
            else if mapSrc(x,y) is NOT empty
                mapDest(x,y) = ( mapSrc(x,y) + mapDest(x,y) / 2.0 )
        Else if criterion == HIGHEST
            If mapDest(x,y) is empty
                mapDest(x,y) = mapSrc(x,y)
            else if mapSrc(x,y) is NOT empty AND mapSrc(x,y) > mapDest(x,y)
                mapDest(x,y) = mapSrc(x,y)
        Else if criterion == LOWEST
            If mapDest(x,y) is empty
                mapDest(x,y) = mapSrc(x,y)
            else if mapSrc(x,y) is NOT empty AND mapSrc(x,y) < mapDest(x,y)
                mapDest(x,y) = mapSrc(x,y)
        Else if criterion == MAXMIN
            If mapDest(x,y) is empty
                mapDest(x,y) = mapSrc(x,y)
            else if mapSrc(x,y) is NOT empty AND abs(mapSrc(x,y)) > abs(mapDest(x,y))
                mapDest(x,y) = mapSrc(x,y)
```

Algorithm 3. Pseudo-code for the height maps merge process

When merging data from two maps, it may happen there might be different height data for the same area or grid cell. In such cases, different merging strategies have been implemented to solve those ambiguities. They are: 1) source map priority, consisting in giving priority to the new computed map over the previously stored information; in this case, taking the global map as starting point, data contained in the newly created local map is copied onto the previous global map; 2) destination map priority, consisting in giving priority to the previously stored map over the new computed; in this case, taking the global map as starting point, data from the newly created local map will be used just to fill void areas in the previous global map; 3) same priority, consisting on, instead giving priority to the old or new data, compute a new height value as a function of both. In this case, several approaches can be followed such as computing the average height value, take the highest or lowest value or take the largest magnitude, either positive -elevation- or negative –depression. This is also a way to deal with uncertainty, computing height value from more than one source according to the established merging strategy introduced previously. The first one is the default strategy, prioritizing the more recent computed data over the previously stored one. Figure 19(a) shows a local map built from two single merged perceptions and in Figure 19(b) it can be seen an updated global map integrated after several navigation cycles.

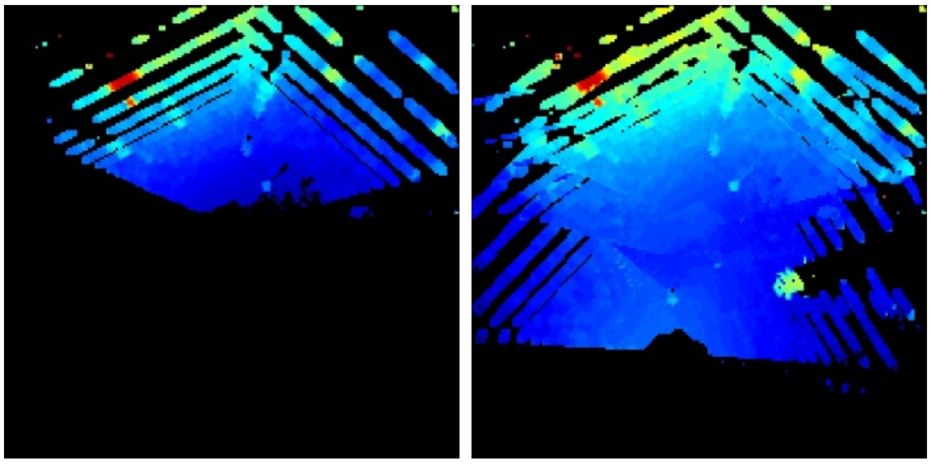

a) b)

Figure 19. a) Merged height map resulted from two single perceptions from the same location and b) integrated height map after several perceptions from different locations

As maps are always stored as rover-centered top-view grids, when updating the global map with a new local map, it has to be taken into account the rover's displacement from the previous navigation cycle. The related translation has to be computed to perform the necessary transformations onto the global map so that both maps, global and local, are centered at the same location prior to be merged. Therefore, the current rover's pose is estimated, as detailed in 4.3.4 Navigation section, and position and orientation deltas computed and applied to re-center the global map.

Values such as map's size, resolution, interpolation parameters or merging data strategy can be configured by the operator according to the mission requirements and constraints with the proposed flexible architecture.

4.3.3 Path Planning

Once the environment has been perceived and an internal representation has been built, the rover shall compute paths to safely drive from one location to another, see Figure 7. Path planning is a heavily researched area, where many techniques and strategies have been already published. Most approaches typically establish an initial and final point within the map and a trajectory between them is computed, relying upon a correct and complete map. However, many of these approaches and search algorithms cannot be applied to the problem of robotic navigation in planetary environments, as there are a number of constraints that do not normally occur in conventional problems, or not all at the same time. Some of these factors are: 1) There is not a priori information about the operational environment; there are no maps from Mars or any other planets, at least with the necessary resolution required for surface rover navigation; 2) A map has to be progressively built once the rover is on the target planet, from in-situ progressive perceptions; 3) the map built at each point is not complete, there may exists unknown areas with no information; 4) the target location may be beyond the current map's limits, in areas not seen yet by the robot; 5) the world cannot be represented as binary, free or occupied space, each cell have a range of possible values representing height data, where traversability has to be evaluated as a function of the rover characteristics and mission constraints; 6) the robot is not a point, so several cells, or a given portion of the map, are involved when computing rover safety at a given location, depending on the resolution of the map created; 7) the rover is non-holonomic and has certain mobility constraints, depending on the mechanical design, restricting the set of trajectories it can perform and 8) there is not any accurate and global localization system such as GPS on Earth, so the rover has to deal with position uncertainty.

In this problem domain, an exploratory approach must be followed where the rover partially perceives the environment to build a map of the visible areas as it travels around, computing trajectories to target locations, computing stretch to intermediate waypoints within the portion of the environment discovered so far. The path planning approach described in this section has been designed and developed specifically for such purpose from the ground up. This is the strategy embedded in the module identified as *path planning* in the scheme of Figure 6.

The rover is typically commanded to a given location, so it is to be sent to a particular point in the world. This location is specified metrically, in length units. The path planning strategy is basically a set of routines that decide the next best direction for the rover to move, as a function of the portion of the world already seen, data from sensors and a desired waypoint goal. The available information about the terrain is evaluated to determine all possible moves the robot may perform. The developed path planning strategy consists on the next set of steps, see Figure 20:

1) The configuration –stored onboard the robot and/or established by ground operators- is read to obtain parameters necessary for the path planning process such as type and number of candidate paths to compute, beam width or planning distance among others. Details about these parameters are given below. This step is actually done just the first time the system is initiated; parameter values are updated as the appropriate command is received from ground control.
2) A set of candidate motion paths, or a more complex curve –spline, are projected onto the height map. This process takes into account the set of parameters taken from the previous step.
3) The algorithm chooses one of the candidates among the set of paths; the one that will best help to reach the designated goal location.
4) The list of cells traversed for a given candidate path is determined.

5) The set of cell traversed by each path is evaluated to assess rover's safety along each given path.
6) The path is considered as safe/unsafe as a result of the evaluation of the constituent grid cells. In case the path is unsafe for the rover, the algorithm goes back to the step 3) to choose another candidate in an iterative process.
7) The parameters or equations describing the selected trajectory is then sent to the low-level controller, and the rover is commanded to move. The *navigation* subsystem is in charge of this task, explained in detail in the subsequent section.

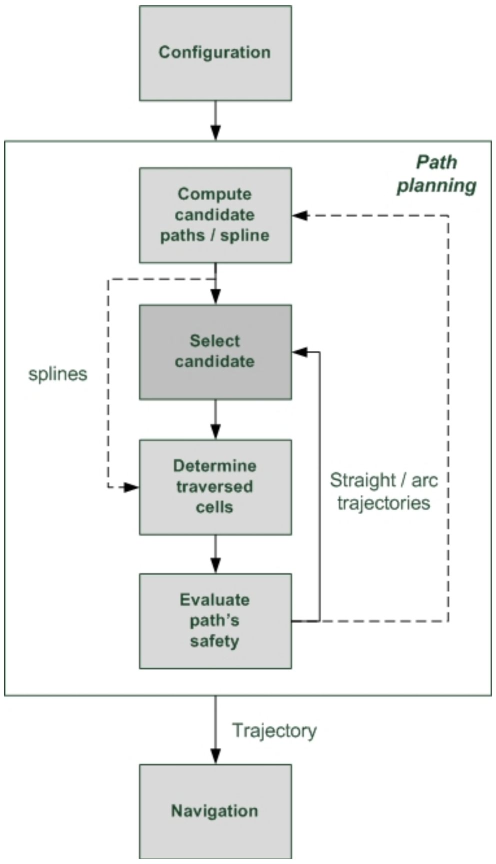
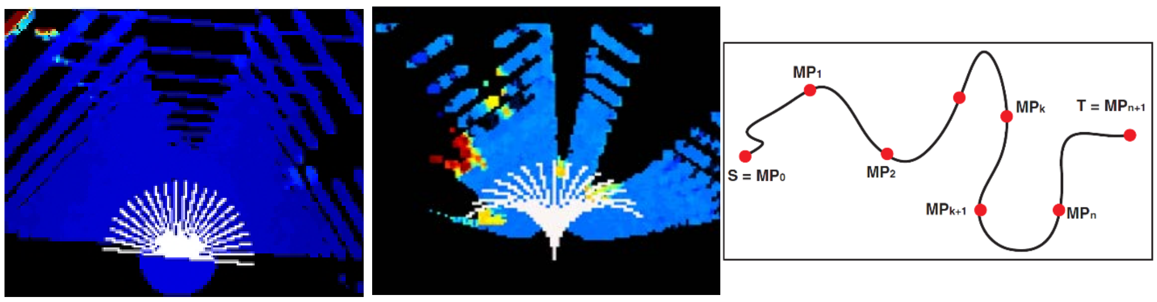


Figure 20. Path planning process within the rover's autonomous navigation strategy

4.3.3.1 Compute Candidate Paths

Three different strategies to compute candidate paths following an exploratory approach have been developed and included within the rover's navigation system: 1) straight paths; 2) arcs; or 3) splines, see Figure 21. The system's operators configure the type of paths to be computed by the rover as a function of the environment's complexity, rover mechanical characteristics and mission's safety constrains.



a) b) c)

Figure 21. a) A set of candidate straight paths or b) arcs projected onto the height map of the area and an example of spline (Magid et al., 2006)

The robot plans the next move according to the designated type of path. Every candidate path has its origin on the current rover location and ends on a given point as a function of the type of path to be computed (straight/arc), planning distance and other parameters such as number of candidate paths to be computed or beam width.

Straight paths are trajectories forming a straight line from the current robot location to a calculated final point. Arcs describe a trajectory with a given constant curvature from the current robot location to a calculated final point. Splines are mathematical representations to build complex curves constructed in the way they pass through a sequence of points, called control points.

In case the system is configured to compute spline trajectories, there will not be multiple candidate paths, just one. Therefore, the processes' flow is a bit different in this case. In Figure 20, the dotted line shows the flow in case of computing a spline; the *select candidate* step is skipped. If the path described by the spline is not safe, the path is modified in an iterative process and adjusted according to different strategies, detailed below. The solid line denotes the processes' flow in case of computing multiple straight candidate paths or arcs.

4.3.3.1.1 Straight Paths

In case the robot is configured to compute straight paths, the algorithm starts by computing the Euclidean distance from the current location to the designated goal, to check if the rover has already reached its goal, or at least a point within a configurable tolerance perimeter around it. If so, the navigation cycle has completed and the traverse will terminate successfully. In any other case, a set of equidistant candidate motion paths are projected onto the height map. The heading for each path is calculated as a function of the number of candidate paths and the beam's width, see Algorithm 4. In Figure 21(a) it can be seen 23 candidate paths with 180º beam width; it implies each path will be approximately be 8.2º separated from each other.

```
distanceToGoal = euclideanDistance(currentLocation, goalLocation)
angleToGoal = computeAngle(currentLocation, goalLocation)

pathHeadingDelta = BEAM_WIDTH / (NUMBER_OF_PATHS-1)

For each $p_i$ in NUMBER_OF_PATHS
      pathHeading = currentRobotOrientation - BEAM_WIDTH/2 + i*pathHeadingDelta
      pathDeviation = atan2(sin(pathHeading-angleToGoal), cos(pathHeading-angleToGoal))

      If distanceToGoal > PLANNING_DISTANCE
            pathLength = PLANNING_DISTANCE
      Else
            pathLength = distanceToGoal

      determineSetOfCells(pathType, pathHeading, pathLength)
```

Algorithm 4. Pseudo-code for the computation of straight candidate paths within the path planning process

Next, for each candidate path, its heading and deviation from the goal location is computed. Each of these paths are defined by its heading and length; they will all have the same length and will differ in their heading. The path's length is determined by the *planning distance* configuration parameter. This value is usually in the order or 1-5 meters. In case the remaining distance from the current location to the goal is shorter than the configured *planning distance*, this remaining distance will be used as maximum length to compute the set of candidate paths, as it is enough to reach the goal.

4.3.3.1.2 Arcs

The type of candidate paths can be also established to be arcs. An arc is a portion of the circumference of a circle. In Figure 22, the arc is the blue part of the circle, going from point A to B. Strictly speaking, an arc could be a portion of some other curved shape, such as an ellipse, but it almost always refers to a circle. To avoid all possible mistake, it is sometimes called a circular arc.

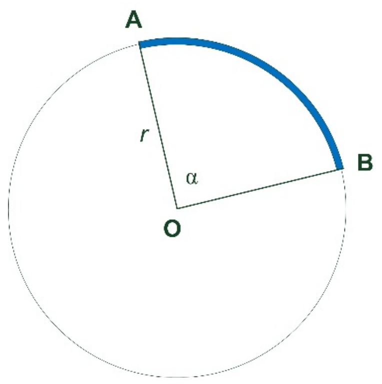


Figure 22. An arc AB is defined by its length, angle (α) and radius (r)

Arcs are defined by its length and angle. The angle α of an arc is defined as the smaller angle formed by the two lines created from points A and B to the center of the circumference. The length of an arc is the distance along the curved line forming the arc between points A and B. It would be measured in distance units, such as meters. It is longer than the straight line distance between its endpoints. The formula to compute the arc length is shown below:

$$\text{arc length} = r\,\alpha \tag{10}$$

where r is the radius of the *arc* and α is the angle.

In case of computing candidate paths as arcs, the algorithm starts, as in the previous case, by computing the Euclidean distance from the current location to the designated goal, to check if the rover has already reached it, or at least a point within the tolerance perimeter. If so, the navigation cycle has completed and the traverse will terminate successfully. In any other case, a set of equidistant candidate motion paths are projected onto the height map. In Figure 23 it can be seen 23 candidate arcs with different beam's width. In all these examples the robot is located in the confluence of the candidate arcs, heading upwards; however, the robot could actually have any orientation, and the arcs beam will be calculated accordingly.

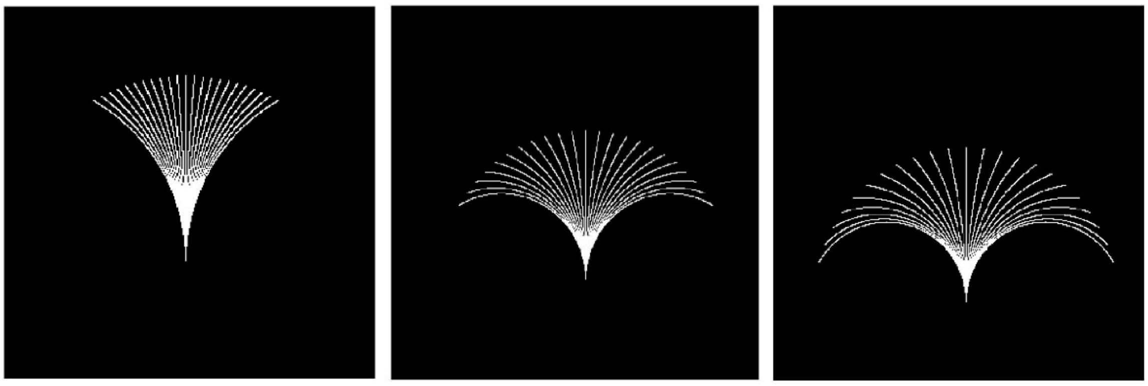

a) b) c)

Figure 23. A set of candidate arcs establishing a beam width of a) 60º, b) 120º and c) 150º

It can be observed the beam is symmetrical; therefore the method just makes the calculations for half of the paths, mirroring the computed coordinates along the axis of symmetry to create the other half paths. There have been designed and implemented two different methods to compute the set of candidate arcs: 1) *variable radius* method and 2) *equidistant ending points*.

4.3.3.1.2.1 Variable Radius

In the *variable radius* method the algorithm projects a set of arcs by computing different circumferences or varying radius between the *minimum* and *maximum exploration radius*, and according to the *planning distance* and number of candidate path parameters. Figure 24 shows a set of circumferences projected to the right of the robot; analogously, the same set of circumferences and arcs are created for the left side, in a symmetrical process. All these circumferences have their origin along the same axis, which is the axis perpendicular to the robot current heading.

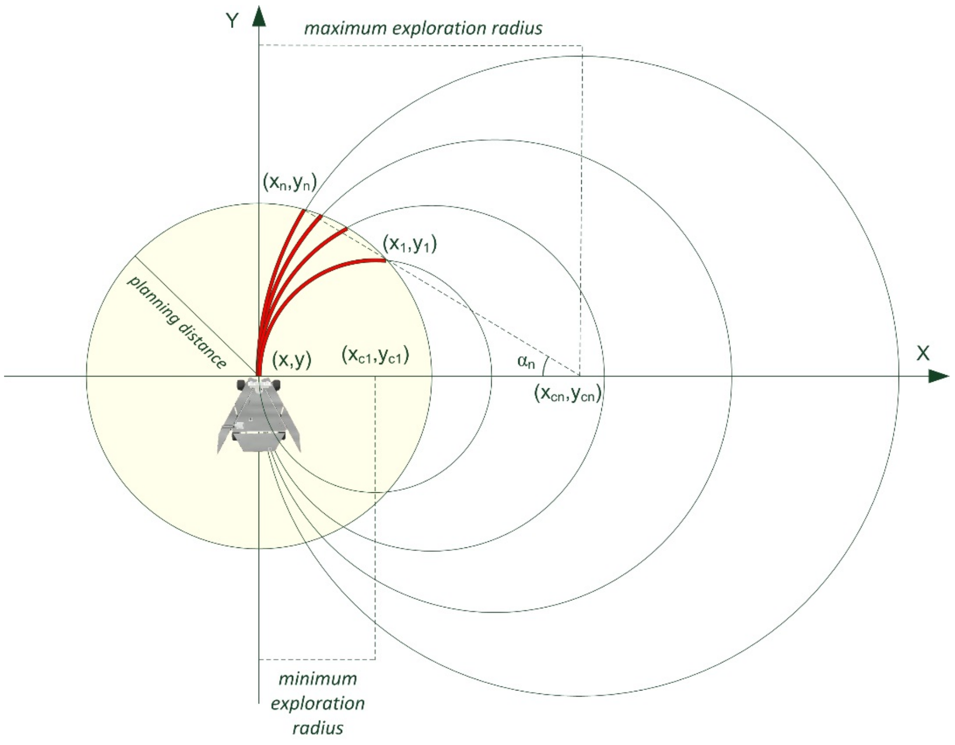


Figure 24. Generating candidate arcs with the *variable radius* method

The radius for each circumference is calculated so that they are proportionally distributed along the *X* axis between the *minimum* and *maximum exploration radius*, established by configuration, and according to the number of arcs to compute:

$$\Delta r = \frac{\text{MAX\_EXPL\_RADIUS} - \text{MIN\_EXPL\_RADIUS}}{(\text{NUMBER\_OF\_PATHS}/2) - 1} \tag{11}$$

where Δr is the radius increment between two adjacent circumferences. The algorithm determines the radius for each circumference as:

$$r_i = \text{MIN\_EXPL\_RADIUS} + (i * \Delta r) \tag{12}$$

Each candidate path, marked in red in Figure 24, is formed by the arc delimited from the current robot location (x, y) to the confluence of each of these circumferences with the circumference formed by the *planning distance* (x_i, y_i). To compute each arc *i*, it is necessary to obtain the coordinates of the origin (x_{ci}, y_{ci}) of the circumference that passes by the points (x, y) and each (x_i, y_i). All these circumferences have their origins along the axis perpendicular to the robot current heading, $y_{ci} = y$. Therefore, $(x_{ci}, y_{ci}) = (r_i, y)$.

The angle corresponding to a given arc can be computed as:

$$\alpha_i = \pi - \tan^{-1} \frac{(y_i - y_{ci})}{(x_i - x_{ci})} \tag{13}$$

The algorithm to compute the set of candidate arcs is shown in Algorithm 5.

```
(x, y) = currentLocation
distanceToGoal = euclideanDistance(currentLocation, goalLocation)

Δr = (MAX_EXPL_RADIUS – MIN_EXPL_RADIUS) / ( (NUMBER_OF_PATHS/2) – 1)

For each $p_i$ in NUMBER_OF_PATHS / 2
    $r_i$ = MIN_EXPL_RADIUS + (i * Δr)

    ($x_{ci}$, $y_{ci}$) = ($r_i$, y)

    $\alpha_i$ = π - atan2 ($y_i$-$y_{ci}$, $x_i$-$x_{ci}$)

    obtainListOfCells($x_{ci}$, $y_{ci}$, $r_i$, $\alpha_i$)
```

Algorithm 5. Pseudo-code for the computation of arc candidate paths within the path planning process following the *variable radius* method

4.3.3.1.2.2 Equidistant Ending Points

In this method for generating candidate arcs, called *equidistant ending points*, a set of points are equidistantly placed along the planning perimeter as a function of the number of candidate arcs and the beam's width, which determines the opening of the beam containing the set of arcs. This is the preferred method and the one configured by default for the rover, as it computes a set of equidistant arcs and covers more homogeneously and symmetrically the area to be explored. Figure 25 shows 15 end points, represented by the red circles, equidistantly distributed along a 100º beam.

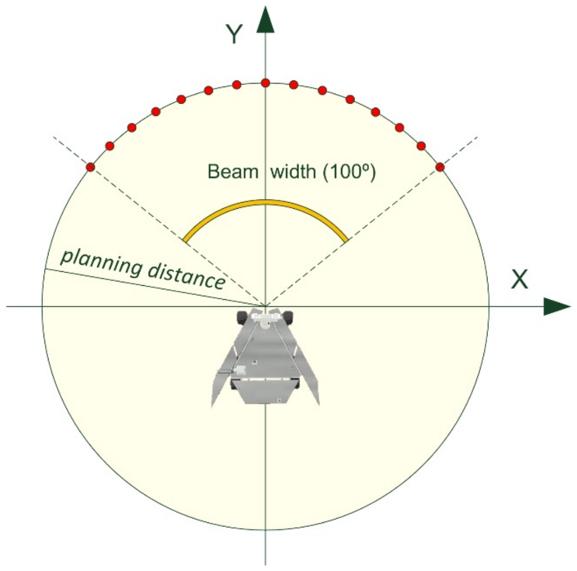


Figure 25. Generating candidate arcs with the *equidistant ending points* method

The algorithm starts by computing the set of circumferences that contain each arc from the robot current location to each of the ending points. A circumference is defined by the next equation:

$$(x - a)^2 + (y - b)^2 = r^2 \tag{14}$$

where the point (a, b) is the origin of the circumference, (x, y) is any point in the perimeter of the circumference and r is the radius.

It is important to note that the origin of those circumferences are all located along the axis perpendicular to the robot current heading (X). Therefore, the problem is formulated as to obtain the equations of the circumferences passing by two points, the current robot's location and each of the ending points located around the planning perimeter, having its origin in the *X* axe, see Figure 26.

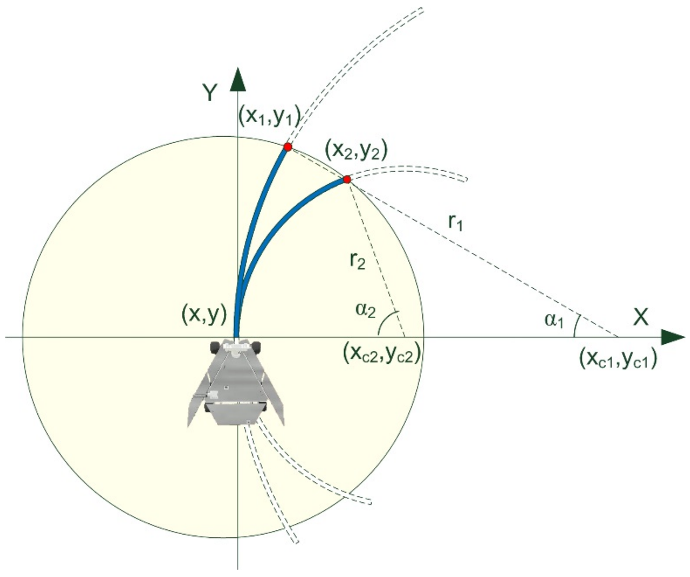


Figure 26. Geometry considerations for the *equidistant ending points* method

To compute the set of ending points (x_i, y_i), the algorithm first calculates the angle increment $(\Delta\alpha)$ to distribute the set of ending points as a function of the beam width and the number of candidate arcs, using the next equation:

$$\Delta\alpha = \frac{\text{BEAM\_WIDTH}}{\text{NUMBER\_OF\_PATHS}-1} \quad (15)$$

Now, having the angle each ending point in the planning perimeter forms with respect to the robot's current location (x, y) and orientation α, their coordinates can be computed as:

$$x_i = x + PLANNING\_DISTANCE * \cos(\alpha \pm i*\Delta\alpha) \quad (16)$$

$$y_i = y + PLANNING\_DISTANCE * \sin(\alpha \pm i*\Delta\alpha) \quad (17)$$

It is also necessary to compute the coordinates of the origins (x_{ci}, y_{ci}) for the circumferences that passes by the points (x, y) and each (x_i, y_i). As in the previous case, all these circumferences have their origins along the axis perpendicular to the robot current heading, therefore $y_{ci} = y$. To obtain x_{ci} it is enough to compute the circumference radius. Replacing (a, b) in equation (14) by (x_i, y_i) we obtain:

$$(x - x_i)^2 + (y - y_i)^2 = r^2 \tag{18}$$

Therefore, r can be calculated as:

$$r = \sqrt{(x - x_i)^2 + (y - y_i)^2} \tag{19}$$

so that $(x_{ci}, y_{ci}) = (r_i, y)$. Now, the candidate path i can be obtained as the arc formed by the circumference with origin (x_{ci}, y_{ci}) and delimited by the points (x, y) and (x_i, y_i). The angle α_i corresponding to this arc can be computed as before according to the equation (13).

The algorithm to compute the set of candidate arcs using the *equidistant ending points* method is shown in Algorithm 6.

```
$(x, y)$ = currentLocation
$\alpha$ = currentOrientation

distanceToGoal = euclideanDistance(currentLocation, goalLocation)

$\Delta\alpha$ = BEAM_WIDTH / (NUMBER_OF_PATHS-1)

For each $p_i$ in NUMBER_OF_PATHS / 2
    $\alpha_i = \alpha + (i*\Delta\alpha)$

    $x_i = x$ + PLANNING_DISTANCE * cos($\alpha_i$)
    $y_i = y$ + PLANNING_DISTANCE * sin($\alpha_i$)

    $r_i$ = sqrt( pow($x-x_i$,2) + pow($y-y_i$,2) )

    $(x_{ci}, y_{ci}) = (r_i, y)$

    $\alpha_i = \pi$ - atan2 ($y_i-y_{ci}, x_i-x_{ci}$)
    obtainListOfCells($x_{ci}$, $y_{ci}$, $r_i$, $\alpha_i$)
```

Algorithm 6. Pseudo-code for the computation of arc candidate paths within the path planning process following the *equidistant ending points* method

4.3.3.1.3 Splines

A more sophisticated strategy to compute trajectories to a given location through complicated environments has been designed and incorporated within the rover's autonomous navigation system. It is based on the calculation of splines.

Splines are piecewise polynomial functions defined through a finite set of control points, with a high degree of smoothness at the places where the polynomial pieces connect; a key property of spline functions is that they are continuous at the control points (also known as knots). Figure 27 shows an example of spline passing by a set of established control points (in red).

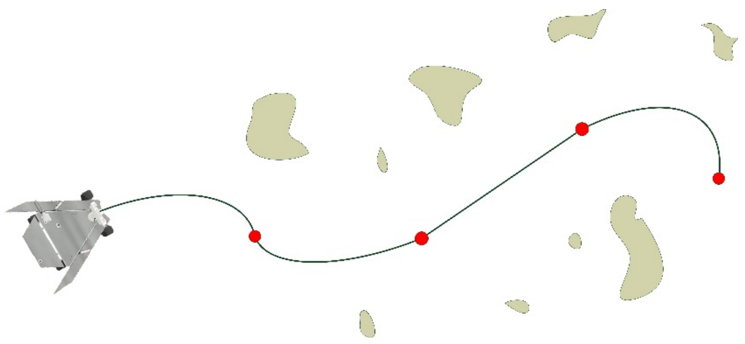


Figure 27. Example of a spline trajectory

Any spline function of degree k on a given set of knots can be expressed as a linear combination of B-splines of that degree. A B-spline (B for Basis) is a polynomial function of degree k in a variable x. It is defined over a range $t_0 \leq x \leq t_m$, $m = k+1$, where $x = t_j$ define the places where the pieces meet, knots. Each piece of the function is a polynomial of degree k between and including adjacent knots. If there are n control points, P_1, P_2,…,P_n, k must be at least 2 (linear), and can be no more than n (the number of control points / knots). The order of the curve (linear, quadratic, cubic,...) is therefore not dependent on the number of control points.

As introduced before, the rover traverses the environment in a series of navigation cycles, see Figure 7. Within the navigation process, the rover spent most of its time stopped in a given location performing computations –perception, mapping and path planning processes- rather than actually driving across the terrain. Against the more simplistic method of computing a set of fixed-length candidate straight paths or arcs, using this technique, just one variable-length candidate path is computed. The objective is to adapt the trajectory to the characteristics of the traversed terrain, being a much more flexible method than previous approaches.

This method ensures the computed trajectory drives the rover as far as possible from its current location and as close as possible to its target. Using this path planning technique the rover can goes farther on each cycle before physically stops to make a new perception. It reduces the number of cycles needed to reach a given target, implying a much more fluent navigation process, reducing the total time the robot stands still performing computations, with the consequent time saving. It increases the overall navigation process' efficiency, allowing the rover to reach farther destinations in less time. In contrast, the design and the algorithms to implement this technique are more complex and the navigability map has to be previously computed, as opposite to the straight paths or arcs methods, as detailed later in 4.3.3.4 Path's Safety Evaluation section.

This path planning method follows an iterative process, refining the trajectory from the current robot location to a given goal to find an efficient, feasible and collision free path through the unstructured environment. As these two points (start and goal) are necessary to compute a trajectory, in case the goal location is established too far, beyond the map's boundaries, a sophisticated method has been designed to find a suitable intermediate goal within the map's limits.

4.3.3.1.3.1 Establishing an Intermediate Goal

In case the goal lies beyond map's limits, the algorithm starts by initially establishing a search perimeter as a function of the map's size by computing the maximum area centered at the current's robot location (x, y) and circumscribed within the map's limits. This radius of this perimeter is actually half of the map's size. In Figure 28 it is represented by the yellow circle.

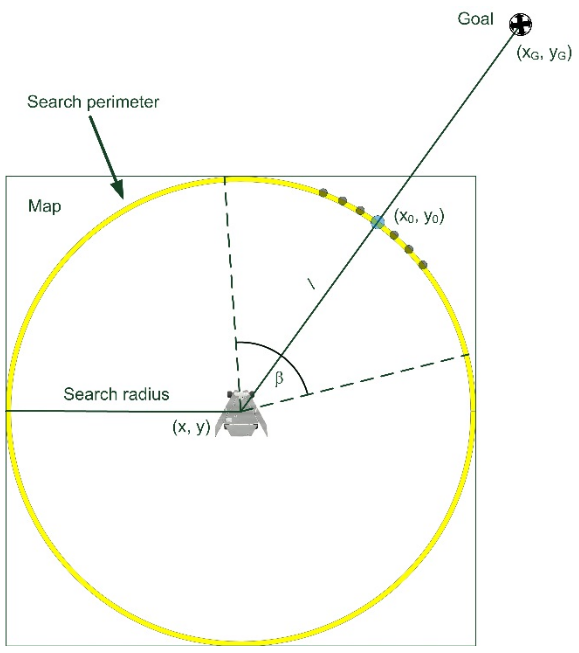


Figure 28. Example of trajectory computation using the splines method (blue) in comparison to the fixed-length candidate paths method (black)

Next, the point (x_0, y_0), where the straight line l formed from the robots' current location to the goal intersects with this search perimeter, is computed. The equation that defines a straight line l is:

$$y = mx + b \tag{20}$$

where m is the slope and b the Y-intercept (where the line intersects the Y axis). The slope m can be computed as:

$$m = \frac{y_G - y}{x_G - x} \tag{21}$$

where (x, y) is the current's robot location and (x_G, y_G) is the established goal location, both known, see Figure 28, b can be calculated as:

$$b = \frac{y}{mx} \tag{22}$$

replacing (x, y) by the current's robot location and m by the value computed from equation (21).

The equation defining a circumference can be seen in equation (14). Replacing the origin, which is the current's robot location (x, y), we obtain:

$$(x_0 - x)^2 + (y_0 - y)^2 = r^2 \tag{23}$$

where r is the known search radius and (x_0, y_0) the coordinates of the point where the search perimeter and the line l joining the robot's current location and the goals Intersect. Therefore, to obtain the (x_0, y_0) coordinates, replacing x and y in equation (23) we obtain an equation in the form:

$$x_0^2 + y_0^2 + \mathrm{D}x_0 + \mathrm{E}y_0 + \mathrm{F} = 0 \tag{24}$$

Now, from equation (20), y_0 can be expressed as:

$$y_0 = mx_0 + b \tag{25}$$

Replacing y_0 in equation (24) by the expression from equation (25), we get a regular quadratic equation. The two roots of the quadratic equation can be then obtained, x_1 and x_2. Replacing both values in equation (20), the respective y_1 and y_2 can be calculated. They represent two crossing points of line l with respect to the circumference, (x_1, y_1) and (x_2, y_2). The Euclidean distance is them calculated from each of those points to the goal location. The shortest one will determine the coordinates for (x_0, y_0).

It is important to recall when using the spline-based path planning method, a complete navigability map has to be previously computed; it implies evaluating every cell in the map to assess rover safety on each possible location and pose in the map prior to the path planning process, as opposite to the straight paths or arcs methods. The navigability evaluation process is detailed later in 4.3.3.4 Path's Safety Evaluation section

If the cell to which (x_0, y_0) belongs in the map is navigable –safe- for the rover, it is established as the intermediate goal (x_g, y_g) and the algorithm will then try to compute a suitable trajectory from the robot's current location to that intermediate goal. In case the cell is non-navigable, an iterative process is started to find an alternative point (x_i, y_i) within the search perimeter. This process computes equidistant points to the right and left of (x_0, y_0) alternately until the limits of a given search cone is reached, delimited by β, Figure 28. If any of those points belongs to a navigable cell, it is established as the intermediate goal (x_g, y_g).

The distance between those alternate points is established as a function of the map's resolution –cell's size- so that each point belongs to a different cell; this is to avoid checking the navigability of the same cell for two adjacent points, which will return the same value. To do that, the coordinates of each alternative point (x_i, y_i) are computed as:

$$x_i = x + r * \cos(\alpha \pm i * \Delta\alpha) \tag{26}$$

$$y_i = y + r * \sin(\alpha \pm i * \Delta\alpha) \tag{27}$$

where r is the established search radius, i is the ith alternative point along the search perimeter (it can be positive or negative, depending if it is located to the left of right of the crossing point (x_0, y_0)) and $\Delta\alpha$ is the angle increment computed as a function of the map's resolution –cell's size (res):

$$\Delta\alpha = \frac{res}{r} \tag{28}$$

In case all points/cells along the perimeter within the search cone α belong to non-navigable areas, the search radius r is iteratively decreased in fixed steps -measured in meters- and the process is repeated until a suitable waypoint is found or the search radius reaches the minimum established length; in that case, an error is reported indicating a safe path to the established goal cannot be computed from the robot's current location using this technique.

The algorithm to compute an intermediate goal is shown in Algorithm 7.

```
$(x, y)$ = currentLocation
$\alpha$ = currentOrientation
$(x_G, y_G)$ = goalLocation
$r$ = searchRadius

While ($r$ > MINIMUM_SEARCH_RADIUS)
    Compute circumference equation as $(x_0 - x)^2 + (y_0 - y)^2 = r^2$
    Compute line equation as $y = mx + b$
    Compute crossing points $(x_1, y_1)$ and $(x_2, y_2)$

    If (euclideanDistance($(x_1, y_1)$, $(x_G, y_G)$) < euclideanDistance($(x_2, y_2)$, $(x_G, y_G)$))
        $(x_0, y_0) = (x_1, y_1)$
    Else
        $(x_0, y_0) = (x_2, y_2)$

    col = (MAP_SIZE / 2) + ($x_0$ / MAP_RESOLUTION)
    row = (MAP_SIZE / 2) - ($y_0$ / MAP_RESOLUTION)

    If Map(col, row) is navigable
        return $(x_0, y_0)$ as $(x_g, y_g)$

    // Alternate candidate points
    $\Delta\alpha$ = MAP_RESOLUTION / $r$

    $n = \beta / \Delta\alpha$  // number of points to check along the perimeter within the cone

    For each point $i$ in $n/2$
        $x_i = x + r * \cos(\alpha \pm i*\Delta\alpha)$
        $y_i = y + r * \sin(\alpha \pm i*\Delta\alpha)$

        col = (MAP_SIZE / 2) + ($x_i$ / MAP_RESOLUTION)
        row = (MAP_SIZE / 2) - ($y_i$ / MAP_RESOLUTION)
        If Map(col, row) is navigable
            return ($x_i$, $y_i$) as ($x_g$, $y_g$)

    r = r - radiusDecrement
```

Algorithm 7. Pseudo-code for the computation of an intermediate goal within the path planning process following the spline-based method

4.3.3.1.3.2 Computing a Spline Trajectory

Once a goal location has been established, either because the commanded location is within the map or computed following the previously described strategy, the spline-based algorithm will try to find a path to it. As introduced previously, a spline $f(x)$ interpolating on the partition $x_0 < x_1 < \ldots < x_{n-1}$ is a function for which $f(x_k) = y_k$. It is a piecewise polynomial function that consists of n-l polynomials f_k defined on the ranges $[x_k, x_{k+1}]$. Furthermore, f_k are joined at x_k (k = 1,... n-2) such that f'_k and f''_k are continuous (Wolberg, 1988). An example of a spline passing through n data points, see Figure 29.

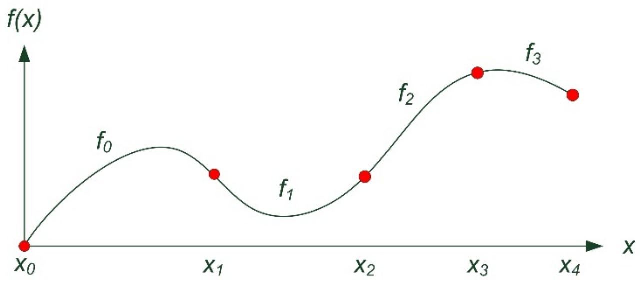


Figure 29. Spline passing through a set of defined control points (red)

Cubic splines are the most popular. The goal of cubic spline interpolation is to get an interpolation formula that is continuous in both the first and second derivatives, both within the intervals and at the interpolating nodes. This will produce a smoother interpolating function. In general, if the function we want to approximate is smooth, then cubic splines will do better than piecewise linear interpolation.

Therefore, given a set of control points (x_k, y_k), or knots, the algorithm must determine the polynomial coefficients, in the form $dx^3 + cx^2 + bx + a$, for each partition such that the resulting polynomials pass through the control points and the slope is continuous at each point. In each interval the four coefficients [a-d] are different. The k^{th} polynomial piece, f_k, is defined over the fixed interval [x_k, x_{k+l}] and has the cubic form:

$$f_k(x) = d(x-x_k)^3 + c(x-x_k)^2 + b(x-x_k) + a \tag{29}$$

The following constraints must be satisfied:

$$y_k = a \tag{30}$$

$$y_{k+1} = d\Delta x_k{}^3 + c\Delta x_k{}^2 + b\Delta x_k + a \tag{31}$$

$$y'_k = b \tag{32}$$

$$y'_{k+1} = 3d\Delta x_k{}^2 + 2c\Delta x_k + b \tag{33}$$

$$y''_k = 2c \tag{34}$$

$$y''_{k+1} = 6d\Delta x_k + 2c \tag{35}$$

where $\Delta x_k = x_{k+1} - x_k$. Therefore, solving the coefficients, it is obtained:

$$a = y_k \tag{36}$$

$$b = y'_k \tag{37}$$

$$c = \frac{1}{\Delta x_k}\left(3\frac{\Delta y_k}{\Delta x_k} - 2y'_k - y'_{k+1}\right) \tag{38}$$

$$d = \frac{1}{\Delta x_k^2}\left(-2\frac{\Delta y_k}{\Delta x_k} + y'_k - y'_{k+1}\right) \tag{39}$$

where $\Delta y_k = y_{k+1} - y_k$. The algorithm returns a set of polynomials p_k (k = 1,... n-1), where n is the number of control points. Each polynomial is defined by their coefficients [a-d]. All these polynomials together define the computed spline-based trajectory from the robot's current location (x, y) to the goal.

The algorithm to compute the polynomials defining a spline-based trajectory is shown in Algorithm 8.

```
controlPointsList  // set of k control points

For each control point k in controlPointsList
      $a_k = y_k$
      $b_k = y_k'$
      $c_k = (1 / (x_{k+1} - x_k)) * ( 3*((y_{k+1} - y_k) / (x_{k+1} - x_k)) - 2*y_k' - y_{k+1}'$
      $d_k = (1 / (x_{k+1} - x_k)^2) * ( -2*((y_{k+1} - y_k) / (x_{k+1} - x_k)) + y_k' + y_{k+1}'$

      polynomialList.add($a_k$, $b_k$, $c_k$, $d_k$)

return polynomialList
```

Algorithm 8. Pseudo-code for the computation of the polynomials that define a spline-based trajectory within the path planning process

4.3.3.1.3.3 Checking Navigability and Adapting the Trajectory

As introduced before, this path planning method follows an iterative process, refining the trajectory in incremental steps to find a safe, collision-free path to the goal through the environment. Initially a trajectory passing by just two control points, the current and goal locations, is computed; the resulting cubic spline between only two points degenerates to a straight line.

The set of cells traversed by the spline is obtained, as detailed in section 4.3.3.3.3 Splines, and its traversability evaluated, detailed in 4.3.3.4 Path's Safety Evaluation. If every cell in the map along the itinerary is evaluated as safe, the path is returned and the path planning process done. In case the path traverses trough an obstacle, non-navigable or unknown areas, the spline is manipulated to guide the path around the obstacle. This manipulation is achieved by the introduction of additional control points, detailed below, so a new spline is computed that passes through these points. This process is repeated until a completely safe path is obtained, the established maximum number of iterations is achieved (e.g.: 500) or the complexity of the trajectory computed so far is too high, meaning there have been included too many control points, established as a density value indicating the maximum number of points allowed per meter (e.g.: 3).

Three different strategies have been designed to manipulate the spline and compute additional control points in order to guide the trajectory through safe areas: 1) *Simple 3-points trajectory*; 2) *overall fitting;* and 3) *progressive fitting*.

4.3.3.1.3.3.1 Simple 3-points Trajectory

This is the first and simplest approach. The trajectory is always computed using just three points: 1) initial, 2) goal and 3) an intermediate point to avoid a given obstacle, except the first iteration where just the initial and goal points are included.

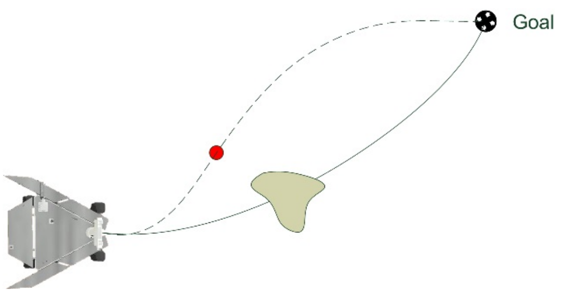


Figure 30. Computing a new control (passing) point (red) to avoid a non-navigable area along the trajectory

Following this strategy, once an initial spline-based trajectory has been computed, the process iterates trough the list of cells in the map traversed by the computed path. In case a cell is evaluated as non-navigable, the algorithm tries to find a navigable cell, close to the non-navigable one, to be used as passing (control) point. A new spline-based trajectory will be then computed so that it passes by the initial, goal and the newly computed passing point, to avoid the obstacle, non-navigable area (the brown irregular area). Figure 30 shows how a computed trajectory (solid line) passes through a non-navigable area; a new control point (red) is computed so that a new trajectory (dashed line) can be obtained to avoid that area.

To compute that new control point, the algorithm iterates around the detected non-navigable cell following a spiral pattern until a navigable cell is found or it reaches the map's limits, see Figure 31. For each candidate cell along the spiral, it is evaluated the rover safety as if it were centered at that location, according to the strategy detailed in 4.3.3.4 Path's Safety Evaluation, see Figure 41. Whenever a navigable, safe cell is found, the coordinates of its central point is calculated and introduced in the control points' list to compute a new spline-based trajectory. Any intermediate control point –cell- checked so far is stored in a list, so that it is not used again in future iterations, as the same result -an unsafe path- will be obtained.

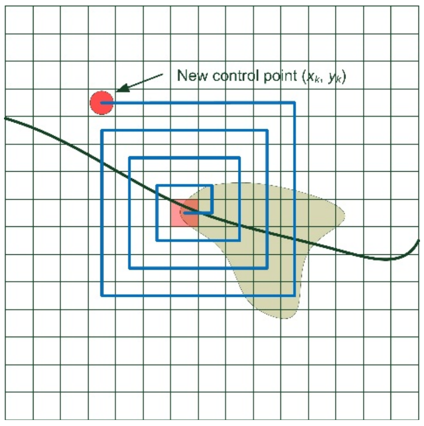


Figure 31. Iterative process following a spiral pattern to compute a new control (passing) point to avoid a non-navigable area

The process is repeated until a safe trajectory is found or the maximum number of iterations has been reached. The algorithm to compute a spline-based trajectory using the *simple 3-point trajectory* method is shown in Algorithm 9. The *computeSpline* method is described in Algorithm 8.

```
do
      controlPointsList.add(currentLocation)
      controlPointsList.add(goalLocation)
      P = computeSpline(controlPointsList)
      numIteration++

      P is safe

      For each cell c along the path P
            If c is not navigable
                  P is not safe
                  p_x = computeAlternateControlPoint(c)

                  controlPointsList.clear()
                  controlPointsList.add(p_x)
                  break   // do not check any more cells
While (P is not safe and numIteration < MAX_ITERATIONS)

If (P is safe)
      return P
Else
      return NULL
```

Algorithm 9. Pseudo-code for the computation of a spline-based trajectory using the *simple 3-point trajectory* method

The main advantage of this approach is it is simple and easier to implement. It is suitable for low-density obstacle, non-navigable areas. However, for complex environments, by using just one passing point may not be possible to compute a safe path through to the goal.

4.3.3.1.3.3.2 Overall Fitting

The main difference with respect to the previous method is the strategy for computing new control points and that the number of control –passing- points is not so limited. There is an established maximum number of control points per meter anyway.

As in the previous strategy, the process first computes an initial spline-based trajectory including the initial and goal locations. It then iterates trough the list of cells in the map traversed by the computed path. Differently than before, the algorithm detects the non-navigable areas (not cells) the trajectory goes through, and tries to find an alternative control point -navigable cell- for each. To compute new control points, the algorithm calculates the portion of the trajectory that traverses a given non-navigable area. It then determines the cell to which the central point of that portion of trajectory belongs to and iterates around, as before, following a spiral pattern until a navigable cell is found or it reaches the map's limits, see Figure 32. As before, at each candidate cell along the spiral, rover's safety is evaluated. Whenever a navigable, safe cell is found, the coordinates of its central point is calculated and introduced in the control points' list to compute a new spline-based trajectory.

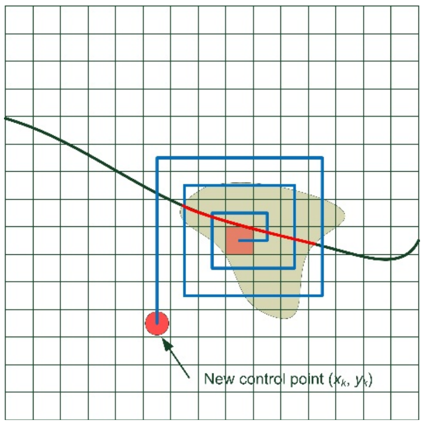


Figure 32. Iterative process following a spiral pattern to compute a new control (passing) point to avoid a non-navigable area

A new spline-based trajectory is then computed so that it passes by the initial, goal and the list of newly computed passing points, to avoid every obstacle and non-navigable areas along the route. The navigability of the trajectory is checked and the process is repeated. On each iteration, any control point along the route before the first crash point is maintained while the rest, from that crash point forward, are erased from the list. Therefore, the next computed trajectory will pass through the previous safe points plus the new computed control points from the first crash point detected on.

The process is repeated until a safe trajectory is found, the maximum number of iterations has been reached or the trajectory become too complex (a maximum number of control points have been reached). The algorithm to compute a spline-based trajectory using the *overall fitting* method is shown in Algorithm 10. The *computeSpline* method is described in Algorithm 8.

```
distanceToGoal = euclideanDistance(currentLocation, goalLocation)

controlPointsList.add(currentLocation)

do
      controlPointsList.add(goalLocation)
      P = computeSpline(controlPointsList)
      numIteration++

      P is safe

      For each cell c along the path P
            If c is not navigable
                  P is not safe
                  $p_x$ = computeAlternateControlPoint(c)
                  controlPointsList.remove(control points beyond c)
                  controlPointsList.add($p_x$)
While (P is not safe and numIteration < MAX_ITERATIONS and controlPointsList.size() <
distanceToGoal*TRAJECTORY_COMPLEXITY)

If (P is safe)
      return P
Else
      return NULL
```

Algorithm 10. Pseudo-code for the computation of a spline-based trajectory using the *overall fitting* method

The main advantage of this approach is that it may be possible in some cases to adjust the trajectory in just one shot, as it computes an intermediate point for every obstacle along the route at the same time. This may be the case in low-density obstacle areas. However, on each iteration the adaptations may cause that recursively some portions of the new trajectory pass through non-navigable areas, forcing more recalculations in the future; it is a collateral effect of computing several control points at the same time. Moreover, as the trajectory suffers from many changes along the process, it becomes more complex on each iteration, creating too sinuous and long trajectories to reach the goal, as it separates more from obstacles on each iteration, see Figure 33.

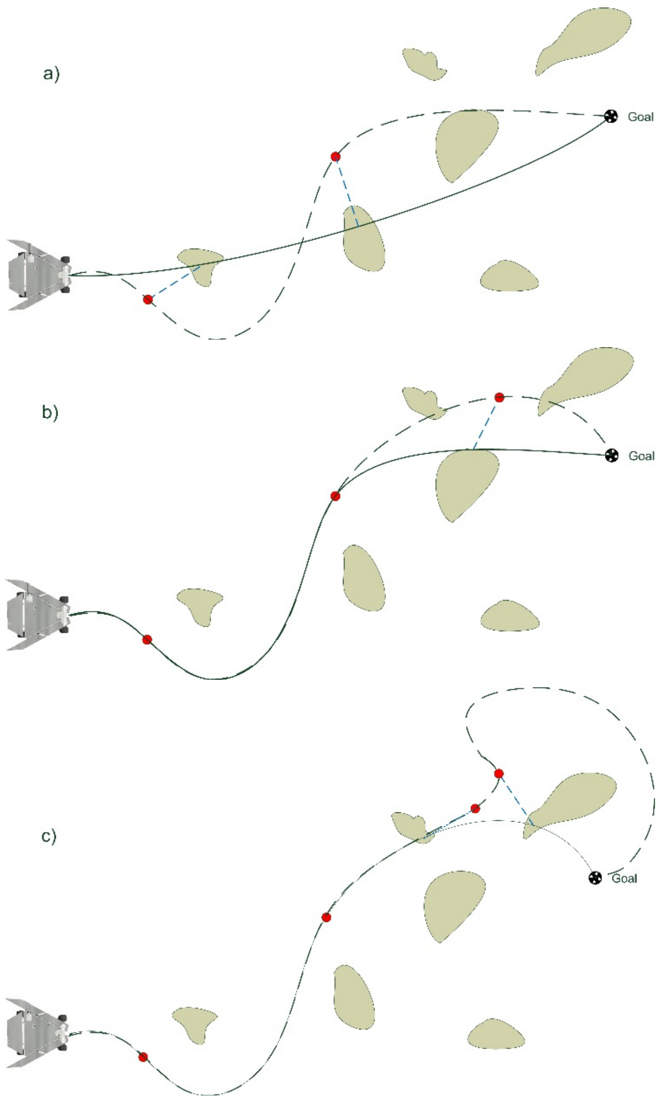
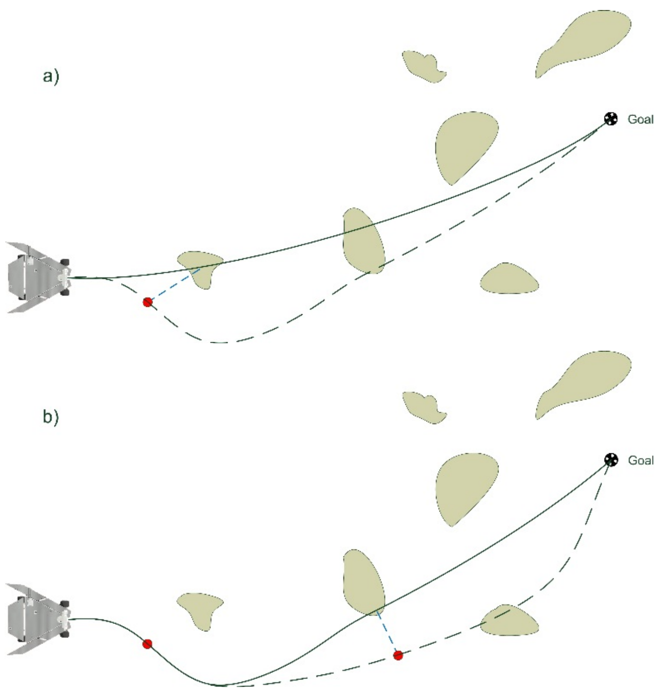


Figure 33. Several steps computing new control (passing) points (red) to avoid non-navigable areas along the trajectory following the *overall fitting* method

4.3.3.1.3.3.3 Progressive Fitting

To avoid such complexity and the computation of too sinuous and long trajectories, the progressive fitting method fits the trajectory progressively along the path. Similarly to the previous approach, this method allows multiple control points; it is also subject to an established maximum allowed number of points. The process first computes an initial spline-based trajectory including the initial and goal locations. It then iterates trough the list of cells in the map traversed by the computed path to detect non-navigable areas (not cells). However, instead computing an alternative control point for each of those areas, it fits the trajectory progressively, step by step, adjusting the curve to avoid each non-navigable area one at a time. The procedure can be thought as drawing an (elastic) line between the initial and end points and with the finger drag the line to avoid each obstacle and go around, see Figure 34. The process to find alternative control point is the same as in the previous method, see Figure 32.



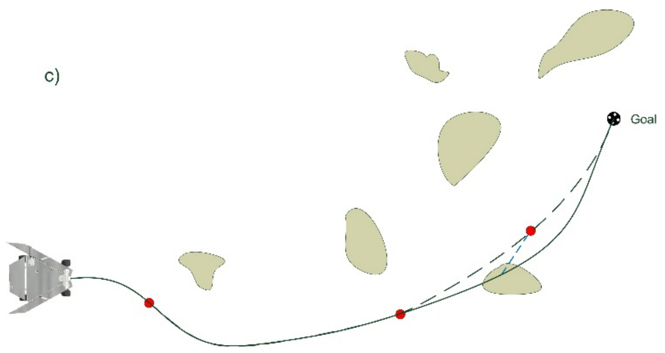


Figure 34. Several steps computing new control (passing) points (red) to avoid non-navigable areas along the trajectory following the *progressive fitting* method

Trajectories are less sinuous than using the previous strategy; they are simpler and more flexible, and also take less computation cycles to be computed. The process is repeated until a safe trajectory is found, the maximum number of iterations has been reached or the trajectory become too complex (a maximum number of control points have been reached). The algorithm to compute a spline-based trajectory using the *progressive fitting* method is shown in Algorithm 11. The *computeSpline* method is described in Algorithm 8.

```
distanceToGoal = euclideanDistance(currentLocation, goalLocation)

controlPointsList.add(currentLocation)
controlPointsList.add(goalLocation)

do
      P = computeSpline(controlPointsList)
      numIteration++

      P is safe

      For each cell c along the path P
            If c is not navigable
                  P is not safe
                  $p_x$ = computeAlternateControlPoint(c)
                  controlPointsList.add($p_x$)
                  break;
While (P is not safe and numIteration < MAX_ITERATIONS and controlPointsList.size() < distanceToGoal*TRAJECTORY_COMPLEXITY)
```

```
If (P is safe)
      return P
Else
      return NULL
```

Algorithm 11. Pseudo-code for the computation of a spline-based trajectory using the *progressive fitting* method

Regardless of the strategy followed to compute a spline-based trajectory, in case a safe path to the goal cannot be found, if the goal location was commanded by the system's operators the robot informs it hasn't been able to find a safe path to the goal through the environment and waits for new instructions. In case the goal location is an intermediate goal computed by the spline-based path planning algorithm, see section 4.3.3.1.3.1 Establishing an Intermediate Goal, a new alternative intermediate goal is computed and the process is started over. Even though this last strategy may seem a long process, it is still much faster than wait for operators' instructions, which usually causes the rover to delay its operations until next day. Meanwhile, it is worthy to try to compute al alternative path.

4.3.3.1.3.4 Smoothing the Trajectory

If the spline-based trajectory has been computed following the *overall* or *progressive fitting* approaches, it may contain a large number of passing points, waypoints, as a consequence of the several iterations performed. An optimization process has been designed in order to simplify it and try to create a smoother trajectory.

After computing a spline-based trajectory, it may happen some intermediate waypoints may be unnecessary. Initially, each waypoint along the route can just be reached from the previous waypoint. However, there are situations where a given waypoint could be reached safely from an even previous waypoint than its immediate predecessor. For instance, Figure 34(c) shows a trajectory computed after several iterations containing five waypoints: the initial and goal locations and three intermediate waypoints. A safe route can be planned from the second to the fourth waypoint, so that the one in the middle, the third one, can be safely removed, see Figure 35.

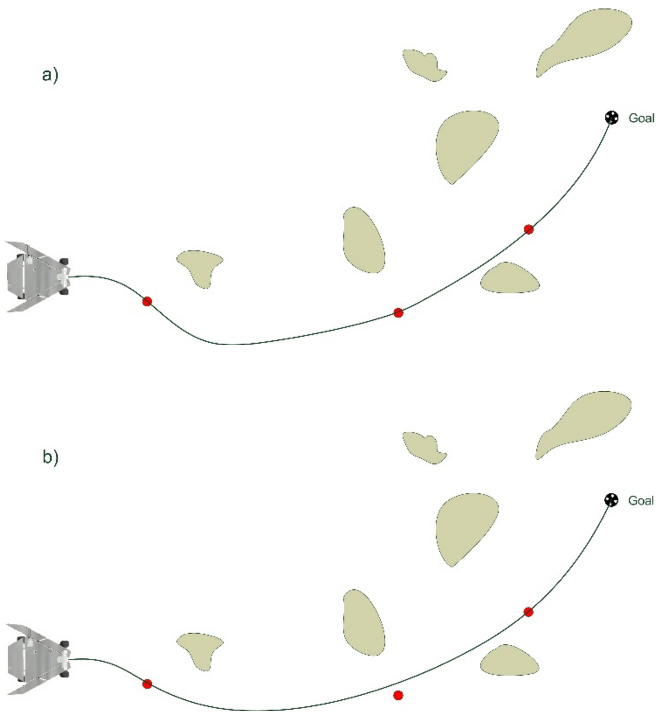


Figure 35. Some waypoints (red) can be safely removed from the trajectory

In such cases, some waypoints can be removed from the trajectory and still reach the goal safely. The fewer waypoints, the simpler and smoother the trajectory will be. This often happens in cases where several waypoints are more or less aligned. Analogously to the assumption that the shortest distance between two points is a straight line, it is also true a polynomial curve passing by two waypoints will be always equal or shorter than a curve passing by three or more waypoints.

Therefore, the smoothing procedure consists of checking the safety of alternative trajectories created by selectively removing waypoints from the original trajectory; the first and last waypoints cannot be removed. The process starts by computing the trajectory resulting from removing the second waypoint; in case the resulting trajectory is safe, it is definitely removed and the process is started all over again, as it may happen a waypoint previously marked as not removable can be removed now, as the path has changed. If the resulting trajectory is not safe, the waypoint is kept and the next one is removed to check the trajectory's safety; this process goes on until the second to last waypoint is reached. The procedure ends when no waypoints can be removed, see Algorithm 12.

```
While (not changes)

      changes = FALSE

      For each control point $p_x$ along the path
            controlPointsList.remove($p_x$)
            P = computeSpline(controlPointsList)

            P is safe

            For each cell c along the path P
                  If c is not navigable
                       P is not safe
                       break;

            If (P is not safe)
                  controlPointsList.add($p_x$)
            Else
                  changes = TRUE
```

Algorithm 12. Pseudo-code for spline-based trajectories' smoothing

The spline-based trajectories strategy represents a significant advantage with respect to the straight paths and arcs computation, and to the NASA/JPL's approach. Following this strategy, the maximum planning distance, fixed for the straight paths and arcs approaches, is dynamically set depending on the quality of the map created and how far a path to the goal, or to get closer to the goal, can be computed on each situation, see Figure 36.

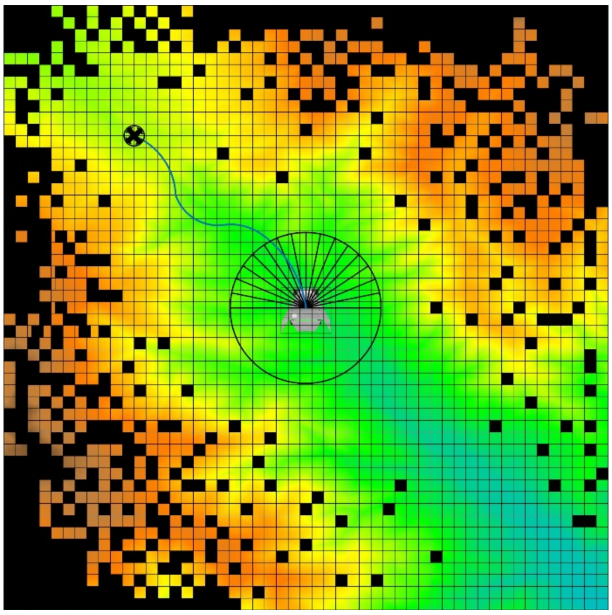


Figure 36. Example of trajectory computation using the splines method (blue) in comparison to the fixed-length candidate paths method (black)

4.3.3.2 Candidate Path Selection

The path selection capability actually gives the rover autonomous decision and control authority to select its next drive direction. In the case of straight paths or arcs, to select from among multiple candidates, each of them is examined in order according to how effectively it would drive the rover toward its goal point. The path that would lead more directly toward the goal or closer is given the highest evaluation priority.

In case the set of candidate paths are straight paths, each one has a value associated that indicates its deviation from the goal. This value is computed as a function of the path's heading with respect to the goal location from the current robot's pose, shown in Algorithm 4. The path with the minimum absolute deviation with respect to the goal location is selected first for safety evaluation, see Algorithm 13. The path selected is removed from the list of candidate paths, so that in case it is evaluated as unsafe, next time another path will be selected from the set of remaining candidates.

```
minDeviation = $P_0$.pathDeviation
x = 0

For each $P_i$ in pathsList
    If (abs($P_i$.pathDeviation > minDeviation))
        x = i
        minDeviation = $P_i$.pathDeviation

remove $P_x$ from pathsList
return $P_x$
```

Algorithm 13. Pseudo-code to select the most directed path from a set of candidate straight paths

If the candidate paths are arcs, the one which end-point is closer to the goal location is chosen as the first to be evaluated, see Algorithm 14. This value is computed as a function of the Euclidean distance from the path's end point to the goal. As before, once a path has been selected for safety evaluation, it is removed from the list of candidate paths. The strategy continues following an increasing distance order.

```
minDistance = $P_0$.distanceToGoal
x = 0

For each $P_i$ in pathsList
    If ($P_i$.distanceToGoal < minDistance)
        x = i
        minDistance = $P_i$.distanceToGoal

remove $P_x$ from pathsList
return $P_x$
```

Algorithm 14. Pseudo-code to select the closest ending path from a set of candidate arcs

In case the computed trajectory is a spline, there is not a set of candidates to select from, as there is just one candidate path. In this case, this step is skipped and the algorithm continues to the next phase, to determine the set of cells traversed by a given path.

4.3.3.3 Determine Traversed Cells

Once a path has been selected, or computed in the case of splines, the list of cells traversed by the path on the grid map has to be determined. To do this, the algorithm goes over the path by dividing it into smaller stretches. The x_i and y_i coordinates of each stretch are computed then to determine the cell in the map it belongs to. The size of these stretches is nominally a third part of a cell's size –the map's resolution.

4.3.3.3.1 Straight Paths

In the case of straight paths, defined by its heading and length, each stretch is equidistantly distributed along the path and also along the X and Y axes, see Figure 37.

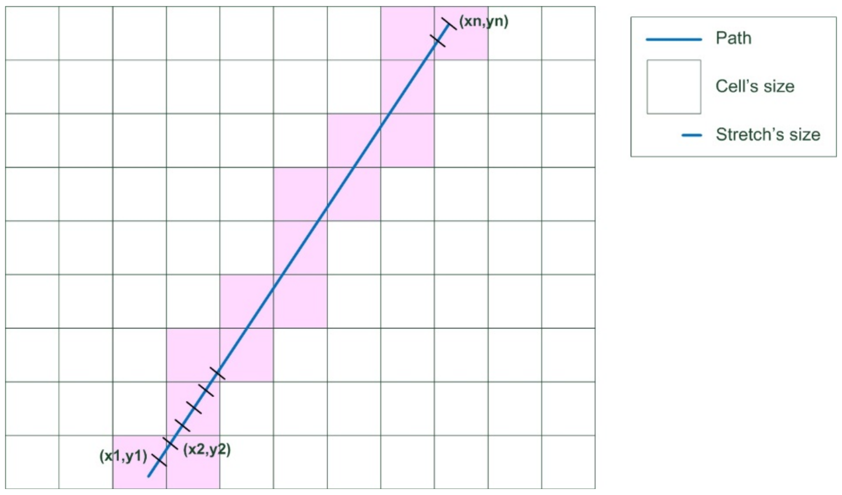


Figure 37. Determining the set of cells traversed by a given path by dividing it in smaller stretches

Therefore, a constant increment in these axes can be computed for each stretch, as:

$$\Delta x = s * cos(\alpha) \tag{40}$$

$$\Delta y = s * sin(\alpha) \tag{41}$$

where s is the stretch's length -a third of a cell- and Δx and Δy are the coordinates increment along the X and Y axes respectively.

The coordinates of each stretch j are computed as:

$$x_j = j * \Delta x \tag{42}$$

$$y_j = j * \Delta y \tag{43}$$

The cell to which each stretch's coordinates belongs to is calculated from the central point of the map, which is the current rover's location, and as a function of the cells' size, which is the map's resolution.

$$col = c_r + (x_j \,/\, res) \tag{44}$$

$$row = c_r - (y_j \,/\, res) \tag{45}$$

where c_r denotes the cell in the map where the rover is –map's center- and *res* is the map's resolution –cell's size. The algorithm to compute the set of cells traversed by a given straight is shown in Algorithm 15.

```
previousRow = 0;
previousCol = 0;
stretchLength = MAP_RESOLUTION / 3

deltaX = stretchLength * cos(pathHeading)
deltaY = stretchLength * sin(pathHeading)

For each stretch j in pathLength / stretchLength
      x = j * deltaX
      y = j * deltaY

      col = (MAP_SIZE / 2) + (x / MAP_RESOLUTION)
      row = (MAP_SIZE / 2) - (y / MAP_RESOLUTION)
```

```
    If (row != previousRow || col != previousCol)
        listOfCells.add(row,col)  // add this cell to the list
        previousRow = row;
        previousCol = col;
```

Algorithm 15. Pseudo-code for the computation of the set of cells traversed by a candidate straight path

As more than one stretch may belong to the same cell, which is actually done on purpose, and is the reason because the stretches' size is configured as one third of the cell's size, it is checked they are not repeated, so that the result is a list of the unique cells traversed by a given path on the current map.

4.3.3.3.2 Arcs

Regardless of the method employed to compute the set of candidate arcs, *variable radius* or *equidistant ending points* method, to determine the list of cells traversed by the arc on the grid map, the algorithm also divides the arc into smaller stretches, computing the *x* and *y* coordinates of each stretch, see Figure 38.

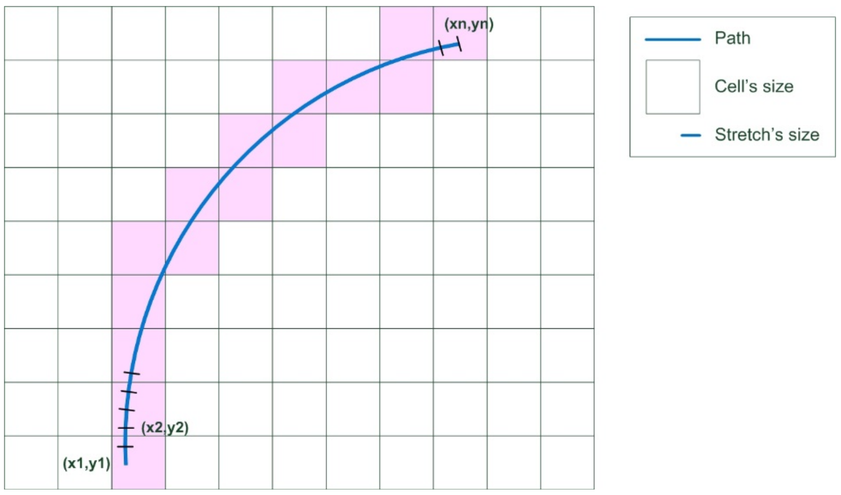


Figure 38. Determining the set of cells traversed by a given arc by dividing it in smaller stretches

The coordinates of each stretch are computed by calculating the angle increment to go over α_i, the angle defining the arc, stretch by stretch from the beginning (x, y) to the end point (x_i, y_i). This angle increment is computed by dividing the stretch size - third of a cell's size as in the case of straight paths- by the radius of the circumference, as shown in equation (10), using the equation:

$$\Delta\alpha = \frac{res/3}{r_i} \quad (46)$$

where *res* is the cell's size in the map and r_i is the computed radius for the circumference describing a given arc *i*. The coordinates for each stretch *j* are computed using the next equations:

$$x_j = x_{ci} + r * cos(\alpha + j * \Delta\alpha) \quad (47)$$

$$y_j = y + r * sin(\alpha + j * \Delta\alpha) \quad (48)$$

That computes arcs to the right of the rover. As the left arcs are symmetrical to the right ones, to obtain the coordinates of those to the left the next formula can be used to compute the *x* coordinate (the *y* coordinate is the same computed for its symmetrical arc to the right):

$$x_j = x - (x_{ci} - r * cos(j * \Delta\alpha)) \quad (49)$$

Finally, the algorithm has to determine the set of cells in the map traversed by each arc. To do that, it computes the cell which coordinates for each stretch ($x_{j,}$, y_j) corresponds to, using the equation (44) and equation (45), see Algorithm 16.

```
(x, y) = currentLocation
α = currentOrientation

(xci, yci) // center of the circumference
ri         // radius of the circumference
αi        // portion of the circumference –angle- described by the arc
```

```
initialRow = MAP_SIZE / 2   // the robot is located at the map center
InitialCol = MAP_SIZE / 2   // the robot is located at the map center
previousRow = 0;
previousCol = 0;
stretchLength = MAP_RESOLUTION / 3

$\Delta\alpha$ = stretchLength / $r_i$

For each angle increment j in $\alpha_i$ / $\Delta\alpha$
      $x_j = x_{ci} + r * cos(\alpha + j * \Delta\alpha)$
      $y_j = y + r * sin(\alpha + j * \Delta\alpha)$

      // Do this also for the symmetric arc, computed as:
      $x_j = x - ( x_{ci} - r * cos(\alpha + j * \Delta\alpha) )$

      col = (MAP_SIZE / 2) + ($x_j$ / MAP_RESOLUTION)
      row = (MAP_SIZE / 2) - ($y_j$ / MAP_RESOLUTION)

      If (row != previousRow || col != previousCol)
           listOfCells.add(row,col)  // add this cell to the list
           previousRow = row;
           previousCol = col;
```

Algorithm 16. Pseudo-code to obtain the set of cells traversed by a candidate arc

As in the previous case, more than one stretch may belong to the same cell, so it is checked they are not repeated and the result is a list of the unique cells traversed by a given path on the current map,

4.3.3.3.3 Splines

Regardless of the method employed to compute a spline-based trajectory, to determine the list of cells traversed by the trajectory on the grid map, as with previously described types of paths, the algorithm divides the trajectory into smaller stretches, computing the *x* and *y* coordinates of each stretch, see Figure 39.

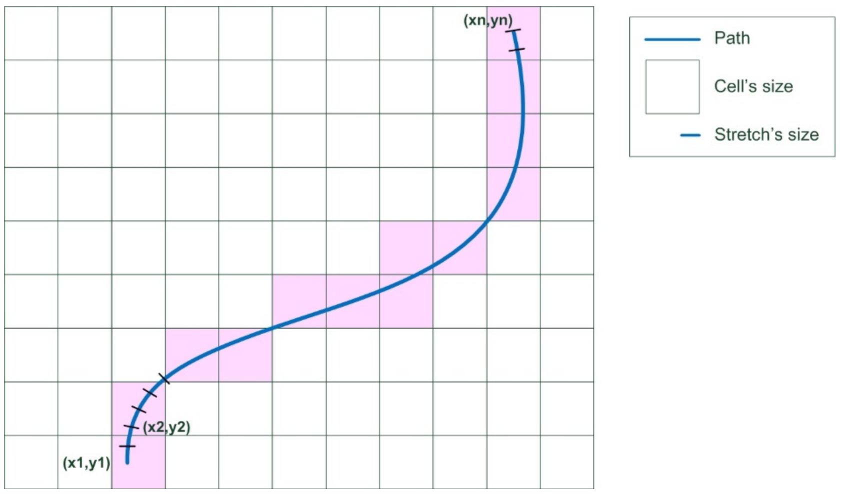


Figure 39. Determining the set of cells traversed by a given spline-based trajectory by dividing it in smaller stretches

The coordinates of each stretch are computed by calculating each interval's length to go over the trajectory's section P_x, stretch by stretch from the beginning (x_k, y_k) to the end point (x_{k+1}, y_{k+1}). The distance between stretches shall be, as with previous methods, the third of a cell's size. So, the interval's length is divided to obtain the number of stretches:

$$n = \frac{euclidean(P_k, P_{k+1})}{res/3} \tag{50}$$

where *res* is the cell's size in the map. The coordinates for each stretch *j* are computed using the next equations:

$$(x_j, y_j) = (a_k + b_k(j/n) + c_k(j/n)^2 + d_k(j/n)^3) \tag{51}$$

Finally, the algorithm determines the set of cells in the map traversed by each arc by computing for each stretch (x_j, y_j) the cell which coordinates corresponds to, using the equation (44) and equation (45), see Algorithm 17.

```
initialRow = MAP_SIZE / 2   // the robot is located at the map center
InitialCol = MAP_SIZE / 2   // the robot is located at the map center
previousRow = 0;
```

```
previousCol = 0;
stretchlength = MAP_RESOLUTION / 3

For each polynom $P_k$ in polynomialList

          length = euclideanDistance(($x_k$, $y_k$), ($x_{k+1}$, $y_{k+1}$))
          n = length / stretchlength

          For each j in n
                    ($x_j$, $y_j$)= ($a_k$ + $b_k$(j/n) + $c_k$(j/n)^2 + $d_k$(j/n)^3)

                    col = (MAP_SIZE / 2) + ($x_j$ / MAP_RESOLUTION)
                    row = (MAP_SIZE / 2) - ($y_j$ / MAP_RESOLUTION)

                    If (row != previousRow || col != previousCol)
                              listOfCells.add(row,col)  // add this cell to the list
                              previousRow = row;
                              previousCol = col;
```

Algorithm 17. Pseudo-code to obtain the set of cells traversed by a spline-based trajectory

It is also checked cell are not repeated and the result is a list of the unique cells traversed by the trajectory on the map, as more than one stretch may belong to the same cell.

4.3.3.4 Path's Safety Evaluation

For a given selected path, terrain traversability and rover integrity has to be evaluated in order to declare a path safe, and decide if the rover shall describe that trajectory and navigate through that path. Therefore, the rover has to have the ability to evaluate the shape of the nearby terrain and determine potential hazards that may hurt the rover along a path.

To assess rover's safety, the *path safety evaluation* algorithm evaluates height values of the list of cells in the map, computed in previous steps, traversed by a given candidate path. It has into account safety parameters and constraints established for the mission as well as the mechanical characteristics of the rover, like chassis measures. This is because a rover is typically larger than a cell in the map. The chassis' measures of a rover like spirit and opportunity from the MER mission (Crisp et al., 2003) are 2.3 x 1.6 m, and the typical map's resolution –cell's size- is 5 or 10 cm. Therefore, supposed the rover centered in a given cell in the map, to evaluate the potential safety of the vehicle in that position, the set of cells underneath the rover –footprint- have to be evaluated.

In Figure 40, the red cell, under the rover platform, indicates the cell in the map where the rover is centered. Green cells, the area under the wheels, define the footprint of the rover. However, given the criticality of the navigation task, and following a conservative approach, to consider a given location –cell- as safe, it has to be safe for the vehicle whatever its orientation is. To do that, rover-sized diameter circumference is computed, represented by the yellow circumference in the figure, to include the set of cells underneath the rover having this any possible orientation; it delimits the area to be evaluated to assess rover safety when it is centered at a concrete location –cell. The intuitive effect of this representation is that the appearance of an obstacle in the local map tends to grow beyond the obstacle's physical boundaries by half the vehicle width in all directions.

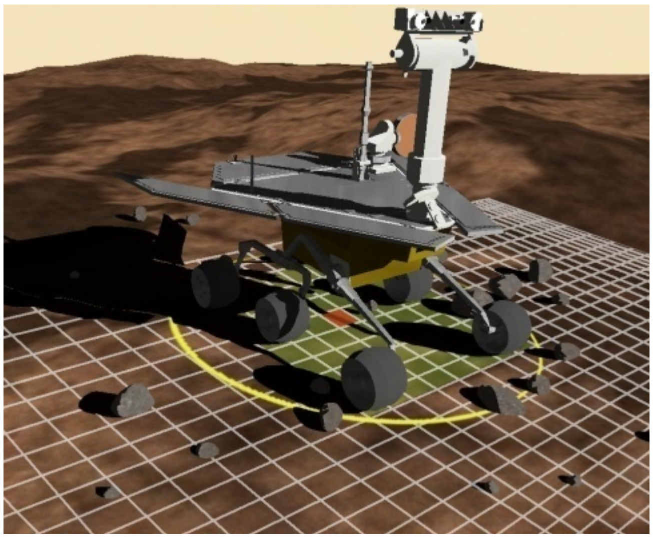


Figure 40. 3D area affected when checking rover safety

Following the same conservative approach, unknown areas are considered by default as non-navigable, so that a void cell within the delimiting area would cause to mark the central cell (red) as unsafe, and therefore to invalidate the whole candidate path. However, a parameter permits configuring the system to avoid the influence of this unknown areas in the planning process.

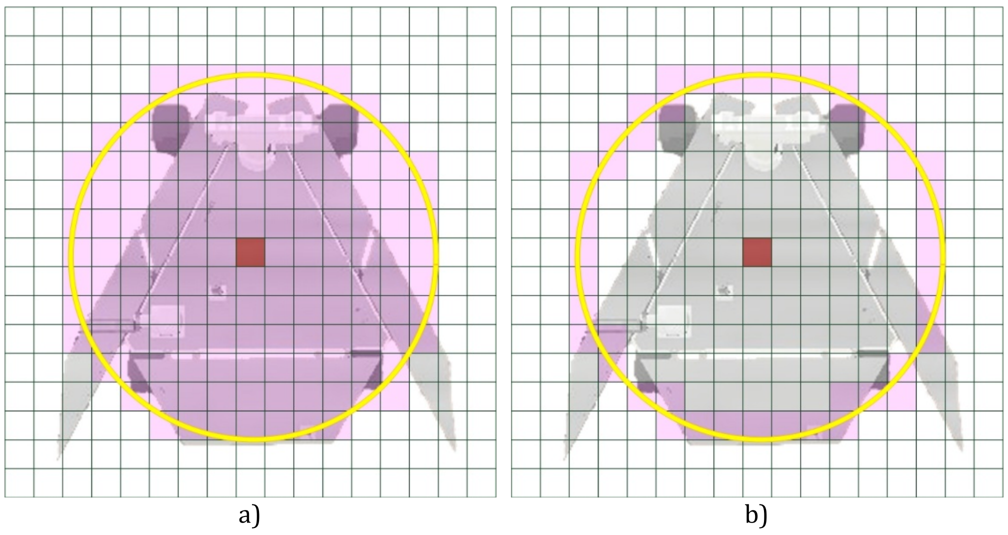


Figure 41. Set of cells considered to compute a) excessive step, excessive roughness and c) excessive tilt

To assess safety and assign a value to the corresponding cell in the map, three potential hazards are evaluated: tilt, roughness, and step.

- Excessive step: indicates if there is a too large step within the rover-sized area in the map. To determine it, the maximum height difference between any pair of adjacent cells in the designated area is computed, identifying the largest step. The set of cells considered in the determination of this value can be seen in Figure 41(a), red cells within the perimeter.
- Excessive roughness: indicates how rough or uneven the underlying terrain is within the rover-sized area in the map. To determine it, the maximum height difference among all cells within the designated area is computed. The set of cells considered in the determination of this value can be seen in Figure 41(a), red cells within the perimeter.
- Excessive tilt: indicates if the underlying terrain is too inclined so that it may cause the rover to tip over. To determine it, the slope is computed as a function of the

maximum height difference among the set of peripheral cells, where front and back wheels rest, within the designated area, which are the responsible for the rover tilt. The set of cells considered in the determination of this value can be seen in Figure 41(b), red cells along the perimeter.

Therefore, for each cell, three values are calculated, corresponding to step, roughness and tilt hazards. Algorithm 18 shows the pseudo-code to analyze each of these situations for each of the cells a given candidate path is made up.

The first step is to compute the radius of the robot in terms of cells, as a function of its chassis' measures and the resolution of the map –cell's size. During path evaluation, in case any part of this rover-sized area is beyond map's limits, the cell is marked as unsafe, returning a maximum value for risk; it will cause the whole path to be invalidated. This is a rare situation nevertheless, as every path starts at the current robot's location, which is at the center of the map, and the size of the map is usually several meters around and the planning distance is not typically configured large enough to exceed map's limits.

Then, for every cell within the rover-sized evaluation area, it is checked if any of them has an unknown value. In such a case, the maximum risk value is returned. By default, the system is conservatively configured to reject any path including cells with no information, as they cannot be evaluated and therefore the safety of the rover along that path cannot be guaranteed. However, this is a configuration parameter than can be set by the operators. In this case is where the interpolation process, detailed in *4.3.2 Mapping of the Environment* section, makes a major contribution, in an attempt to minimize the number of void cells.

In case the previous conditions are met, values for each of the potential hazards are computed for each cell as indicated before; a maximum *step* value is computed as the maximum height difference between any pair of adjacent cells in the designated area; a maximum *roughness* value is computed as the maximum height difference among all cells within the designated area; and a maximum *tilt* value is computed as the arcsin of the maximum height difference among just the set of peripheral cells divided by the rover's diameter, following the next formula:

$$\text{tilt} = \arcsin\left(\frac{maxHeight - minHeight}{\text{ROBOT\_BASELENGTH}}\right) \tag{52}$$

where *maxHeight* and *minHeight* denote the maximum and minimum height values found among the set of cells considered, see in Figure 41(b).

```
x = MAP_SIZE / 2   // the robot is located at the map center
y = MAP_SIZE / 2   // the robot is located at the map center

maxStep = 0

minRoughnessHeight = Map(x,y)
maxRoughnessHeight = Map(x,y)

minTiltHeight = Map(x,y)
maxTiltHeight = Map(x,y)

robotRadiusInCells = (ROBOT_BASELENGTH / 2) / MAP_RESOLUTION

For each cell c in listOfCells

      //check limits
      if (c-robotRadiusInCells < 0 || c+robotRadiusInCells >= MAP_SIZE)
            return MAX_VALUE

      // go over the rover-sized area horizontal and vertically
      For each i from c-robotRadiusInCells to c+robotRadiusInCells
            For each j from c-robotRadiusInCells to c+robotRadiusInCells
                  If euclideanDistance((x,y), (i,j)) <= robotRadiusInCells
                        If Map(i,j) == UNKNOWN && not UNKNOWN_OK
                              return MAX_VALUE

                        // Check excessive step hazard
                        For each k from i-1 to i+1
                              For each l from j-1 to j+1
```

```
                                If abs(Map(i,j) - Map(k,L) > maxStep)
                                    maxStep = abs(Map(i,j) - Map(k,L)

                        // Check excessive roughness hazard
                        If (Map(i,j) < minRoughnessHeight)
                            minRoughnessHeight = Map(i,j)
                        else if (Map(i,j) > maxRoughnessHeight)
                            maxRoughnessHeight = Map(i,j)

                        // Check excessive tilt hazard
                        If euclideanDistance((x,y), (i,j)) == robotRadiusInCells
                            If (Map(i,j) < minTiltHeight)
                                minTiltHeight = Map(i,j)
                            else if (Map(i,j) > maxTiltHeight)
                                maxTiltHeight = Map(i,j)

    stepHazard = maxStep
    roughnessHazard = maxRoughnessHeight - minRoughnessHeight
    tiltHazard = asin((maxTiltHeight – minTiltHeight) / ROBOT_BASELENGTH
```

Algorithm 18. Pseudo-code to check potential hazards for the set of cells forming a given candidate path

Finally, for the candidate path selected for evaluation, the computed values for the potential hazards of each cell of the path are compared with the thresholds configured by the operators of the system, indicating the maximum allowed values for each of the potential hazardous situations identified previously. In case any of them exceeds any of the thresholds, the cell is marked as unsafe, and the whole candidate path is discarded. In such a case, the second best path is chosen for evaluation and the process is repeated, see Algorithm 19.

```
If pathType == SPLINE
    return pathsList(0)
Else
    While there are paths in pathsList
        If pathType == STRAIGHT
            $P_x$ = getMostDirectedPath(pathsList)
        Else
```

```
                $P_x$ = getClosestEndingArc(pathsList)

            For each cell c in $P_x$
                If c.stepHazard > STEP_THRESHOLD ||
                   c.roughnessHazard > ROUGHNESS_THRESHOLD ||
                   c.tiltHazard > TILT_THRESHOLD)
                        safePath = FALSE

            If safePath == TRUE
                return $P_x$
```

Algorithm 19. Pseudo-code to select a path from the set of candidate paths

One of the main contribution of this navigation strategy with respect to similar approaches, as the one developed by NASA/JPL for the MER mission, is actually this path evaluation approach and the computation of navigability. The strategy presented in this section computes values and evaluates just the cells that are part of any candidate path instead computing a value for each cell in the map, except for the case of the splines approach, where the entire map shall be initially evaluated. This approach avoids spending computation power and time calculating cells' values that will never be used.

To get a sense of the computation savings following this approach, a typical map of 10x10 m and 10 cm cell's resolution contains 10,000 cells. If the system is configured to plan 23 candidate paths for instance with a 3 m planning distance, where each path is composed of 30 cells roughly, it implies the set of paths is constituted by around 700 cells at most. This number is actually lower, as many cells are shared between several paths, especially in the area closer to the rover, where candidate paths are all together, see Figure 23; this shared cells are evaluated just once. It means just 7%, at most, of the cells need to be evaluated, while 93% of the cells in the map are not part of any candidate path and will never be taken into account for the path evaluation process. Therefore, it is unnecessary to compute a traversability value for them, with the consequent computing resources and time savings. In case of increasing the resolution, the optimization achieved following this strategy is even greater; in the same previous example but with a 5 cm map's resolution, following an analogous analytical process, the percentage of cells part of any path decreases to a mere 3.5%; generalizing, this percentage decreases as resolution increases.

Moreover, the optimization in the computing resources saving goes even further. Instead evaluating the set of cells part of any candidate path, the algorithm choses the most promising path among the set of candidates. Traversability of cells affecting that candidate path, and just those cells, is then checked, assessing potential hazards. If safety constraints are met the path is declared safe and selected to be sent to the *navigation* module to be executed. Otherwise, the process is repeated with the next most promising candidate. In some cases, if the first or second path examined is actually selected, just about 0.1-0.3% of the cells will be evaluated. This also represents a major improvement and contribution with respect to other comparable approaches.

The system allows a system's operator to establish a series of parameters for this *path planning* module, known as the safety parameters, setting the maximum allowed threshold values for each of the previously indicated potential hazards –step, roughness and tilt-, to be evaluated and taken into account by the path planning process while computing trajectories. Other configurable parameters affecting the path planning module are also: number of candidate paths, type (arcs, lines, splines), planning distance, navigation distance, chassis measures, body clearance, wheel's diameter or wheels' separation among others, so that the impact altering any of these mechanical design or mission constraints parameters can be evaluated straightforward.

In case no suitable path is found among the set of candidates, several strategies has been designed to deal with this situation; the desired behavior can be established by the operator, specifying one of the following strategies: 1) report an error and wait for a new plan: this is the most conservative approach, it prevents the robot of making any decision, relying on ground-control operators; however, the rover has to wait until the next communication window to inform, and the activities will not be resumed until next, or later, day; 2) repeat the planning process increasing the number of candidate paths: the number of these paths will be multiplied by a factor of *n*, specified in the robot configuration file –and also commanded, as well as the number of attempts before announcing no suitable path was found; 3) make new perceptions to extend and update the internal map model, usually pointing at different directions than previous perceptions; this offset can be specified in the robot configuration file –and also commanded; 4) compute backwards paths to move away from the non-navigable areas and plan again: this is the most risky strategy. In case of computing backwards paths, chances are the portion of the terrain necessary for the computation has already been perceived and stored within the map. In such cases, paths can be computed without taking new images. Otherwise, the camera shall be rotated to face the rear area to take new images to update the map.

4.3.4 Navigation

The locomotion in planetary rovers is usually achieved through six wheels. Each wheel pair is suspended on an independently pivoted bogie (the articulated assembly holding the wheel drives), and can be independently steered and driven. All wheels can be individually pivoted to adjust the rover height and angle with respect to the local surface, and to create a sort of walking ability, particularly useful in soft, non-cohesive soils like dunes.

Besides *planning distance*, related to the length of the candidate paths to be projected on the map, there also exist another key parameter: *navigation distance*. It specifies how long the selected path will be actually followed. Figure 48 shows the rover stopped at a given location performing the path planning process. In Figure 42(a) the navigation strategy has been configured to compute a set of straight paths, while in Figure 42(b) it computes a set of candidate arcs. The outer circular area represents the *planning distance*, and the inner circular area represents the *navigation distance*.

Navigation distance is commonly set shorter than *planning distance* to avoid getting too close to obstacles or non-navigable areas right at the end of a given executed trajectory that may prevent the rover to move forward in the next navigation cycle. As an example, *planning distance* can be configured to 3 meters and *navigation distance* to 2 meters. In case the estimated remaining distance from the current rover's location to its goal is shorter than the *planning distance*, both the planning and the navigation distances are set to this estimated remaining distance.

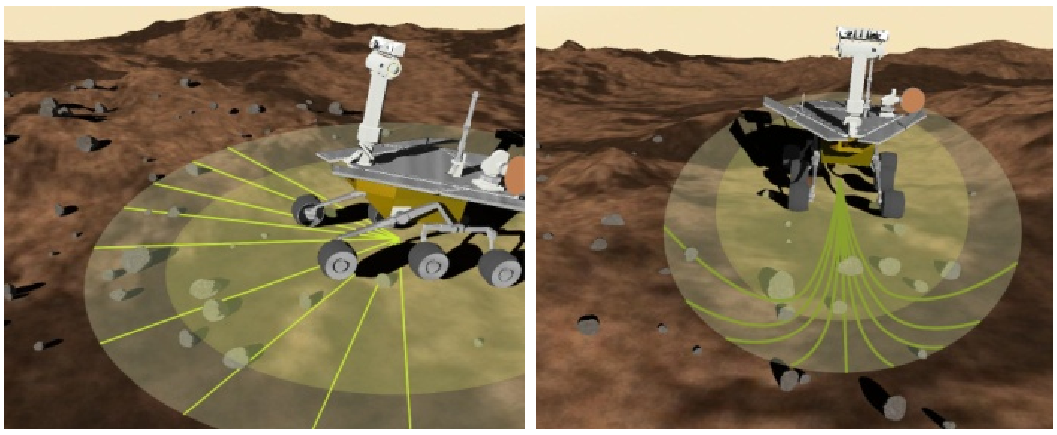


Figure 42. Rover model in a simulated terrain and path planning process with a) straight and b) arc candidate trajectories

Depending on the type of path computed –straight, arc or spline- the desired turn and navigation distance, or the equations describing the trajectory, is sent to the *navigation* subsystem, see Figure 7, so the rover is commanded to move and the low-level controller can actually drive the robot along the path. This is the process embedded in the module identified as *navigation* in the scheme of Figure 6.

In the case of straight path, the trajectory is defined by its angle and length -*navigation distance*; the vehicle must turn-in-place about the vehicle center of rotation to face the computed path's heading. Turns-in-place can be commanded using absolute or relative reference headings or specific cartesian coordinates of a location toward which to face. Arcs are defined by their angle, radius and length -*navigation distance*; in this case the rover does not need to turn-in-place, but configure the adequate velocities of each wheel to describe the computed arc according to its defining parameters. As for the splines, they are defined by the set of polynomial equations describing the trajectory; each of the equations describes a curve, where each one is concatenated with the next to create the complete trajectory.

4.3.4.1 Proportional Control

To execute the commanded trajectories, the rover implements a P (Proportional) controller, estimating its position for corrections. A proportional control system is a type of linear feedback control system. In the proportional control algorithm, the controller output is proportional to the error signal, which is the difference between the desired value (setpoint) and the actual value (process variable). In other words, the output of a proportional controller is the multiplication product of the error signal and the proportional gain. This can be mathematically expressed as:

$$P_{out} = K_p * e + p0 \tag{53}$$

where P_{out} is the output of the proportional controller; K_p is the proportional gain; e is the current error, computed as the difference between the current and desired signal values; and $p0$ is the controller output with zero error.

Two proportional controllers have been implemented for the robot navigation, to compute a desired linear and angular velocity values respectively, see Algorithm 20.

```
$(x_{ref}, y_{ref})$ = referenceLocation
$\alpha_{ref}$ = referenceOrientation

setTimer(controlFrecuency)

While ($e_v > e_{v\text{-}max}$ || $e_w > e_{w\text{-}max}$)

      $(x, y)$ = currentLocation

      $e_v$ = euclideanDistance($(x, y)$, $(x_{ref}, y_{ref})$)

      If there is any $\alpha_{ref}$
            $e_w$ = currentRobotOrientation - $\alpha_{ref}$
      Else
            $e_w$ = currentRobotOrientation - atan2($y_{ref}$ - y, $x_{ref}$ - x)
```

```
    $v = K_v * e_v$
    $w = K_w * e_w$

    setRobotSpeed(v, w)

    waitForTimer()
```

Algorithm 20. Pseudo-code to implement the proportional controllers for trajectories' execution

A waypoint has an associated position tolerance, i.e., a radial distance within which the waypoint is considered reached. This is indicated through two configuration parameters, *position tolerance error* ($e_{v\text{-}max}$) and *orientation tolerance error* ($e_{w\text{-}max}$), establishing the tolerance perimeter around the waypoint and the orientation accuracy respectively.

Maximum and minimum linear and angular velocities are established so that they cannot be exceeded, no matter the results of the control law's computations are, for security reasons, and also to avoid too slow motion when the rover is close to the desired –reference- position.

4.3.4.2 Straight Trajectories

In case the navigation trajectory is a straight path, defined by its angle (α) and length (l), the robot first turns in place to face the path's heading (α). The controller's parameters α_{ref} is set to the path's heading and (x_{ref}, y_{ref}) to the robot's current location, to complete the turn in place, and next it advances straightway for the established navigation distance, setting the controller's parameters α_{ref} to robot's current orientation and (x_{ref}, y_{ref}) to the robot's current location plus the path I's length minus the navigation distance.

4.3.4.3 Arc Trajectories

In case the navigation trajectory is an arc, defined by their angle (α), radius (r) and length (l), an algorithm divides the arc into intermediate reference points, in a similar process described above to determine what cells in the map are traversed by an arc shown in Figure

38. The coordinates of each intermediate reference points (x_{ref}, y_{ref}) are used to command the control algorithm (Algorithm 20) at regular time intervals, see Figure 43. Each blue marker denotes a reference point.

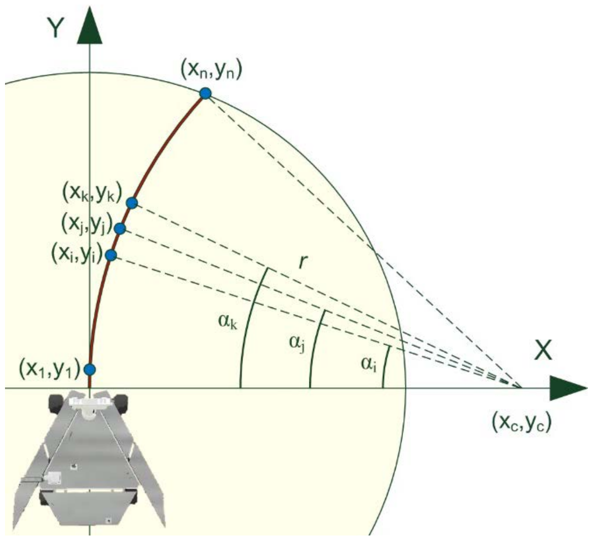


Figure 43. Computation of reference points (x_{ref}, y_{ref}) along the arc to command the control algorithm at regular time intervals

As those reference points are generated at periodical time intervals, the distance between them depends on the frequency they are generated. The location of each reference point is computed as a function of the estimated distance traveled by the rover at a given nominal speed in the time elapsed. From equation (10) it can be inferred:

$$l_{ref} = r\,\alpha_{ref} \tag{54}$$

To compute α_{ref}, and determine the coordinates of the point (x_{ref}, y_{ref}), it can be done by:

$$\alpha_{ref} = \frac{l_{ref}}{r} = \frac{v * \Delta t}{r} \tag{55}$$

where v is the robot's nominal speed, Δt is the time elapsed from the beginning of the trajectory navigation process and r is the radius that defines the arc. As a result, α_{ref} designates a concrete point in the arc (x_i, y_i), see Figure 43, with respect to the circumference of radius r and origin (x_c, y_c). The coordinates of a concrete reference point can be then computed, and established as the next waypoint to be reached and passed to the control algorithm, using the next formulas:

$$x_{ref} = x_c + r * cos(\alpha_{ref}) \quad (56)$$

$$y_{ref} = y_c + r * sin(\alpha_{ref}) \quad (57)$$

In this case there is no orientation (α_{ref}) for a given reference point to be passed to the control algorithm (Algorithm 20), just its position (x_{ref}, y_{ref}), see Algorithm 21.

```
$(x, y)$ = currentLocation
$x_{ref} = x$
$y_{ref} = y$

// compute arc's end-point
$x_n = x_c + r * cos(\alpha)$
$y_n = y_c + r * sin(\alpha)$

$t_0$ = currentTime()

setTimer(timeInterval)

// generate reference points while end-point not reached
While (euclideanDistance(($x_{ref}$, $y_{ref}$), ($x_n$, $y_n$)) > TOLERANCE_PERIMETER)

    $t$ = currentTime()

    $\alpha_{ref}$ = (NOMINAL_SPEED * $(t - t_0)$) / $r$

    $x_{ref} = x_c + r * cos(\alpha_{ref})$
    $y_{ref} = y_c + r * sin(\alpha_{ref})$

    call control algorithm to go to ($x_{ref}$, $y_{ref}$)

    waitForTimer()
```

Algorithm 21. Pseudo-code to compute intermediate waypoints -reference points- to describe an arc trajectory

4.3.4.4 Splines

In case the navigation trajectory is a spline-based trajectory, it is defined by a set of polynomial equations –intervals-, where each one describes a curve between two adjacent control points; the concatenation of all of them creates the complete trajectory. Each polynomial equation for an interval is defined by their coefficients in the form $dx^3 + cx^2 + bx + a$, see section 4.3.3.1.3.2 *Computing a Spline Trajectory*. There is actually two polynomials, one to compute the *x* coordinate and another for *y*. The variable (*x*) can be given a value in the range [0, 1], where 0 denotes the interval's starting point and 1 the interval's end point. Therefore, to compute a concrete location –coordinates- along the trajectory, it is enough to give *x* a value within the range and solve the previous equation; the *y* coordinate is computed equivalently. For instance, the coordinates of the interval's midpoint can be computed solving the equation with the value 0.5.

The length of each interval can be computed in a similar process than the one followed to determine which cells are traversed by the trajectory in the map shown in section 4.3.3.3.3 *Splines*. The interval's trajectory is divided into stretches. The total length of the interval is approximated by adding up the lengths of each of those stretches, computed as the Euclidean distance between the coordinates of two adjacent stretches, see Figure 44. In the figure, the trajectory is shown in red and stretches in yellow.

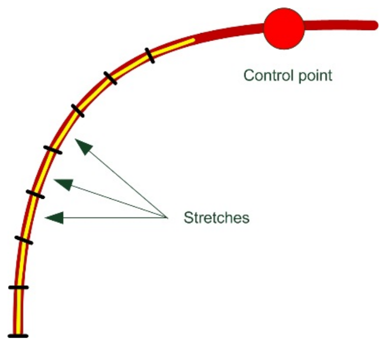


Figure 44. Approximating an interval's length by adding up the lengths of its stretches

The algorithm to compute an interval's length is shown in Algorithm 22.

```
// compute interval's start-point
x = getPolynomValue(P_k, 0)
y = getPolynomValue(P_k, 0)
length = 0

n = number of points used to approximate interval's lengths

For each j in n
    x_j = getPolynomValue(P_k, j/n)
    y_j = getPolynomValue(P_k, j/n)

    length += euclideanDistance((x_j, y_j), (x, y))

    x = x_j
    y = y_j

return length
```

Algorithm 22. Pseudo-code to calculate an interval's length for a spline-based trajectory

In order to describe a computed spline-based trajectory, the navigation algorithm divides each interval into intermediate reference points, following an analogous process to describe an arc trajectory, see section 4.3.4.3 Arc Trajectories. The coordinates of each intermediate reference points (x_{ref}, y_{ref}) are used to command the control algorithm (Algorithm 20) at regular time intervals, see Figure 45. Blue markers indicate reference points, and red markers are the control points that delimits each interval.

As before, those reference points are generated at periodical time intervals; therefore, the distance between them depends on the frequency they are generated. The location of each reference point is computed as a function of the estimated distance traveled by the rover at a given nominal speed in the time elapsed. Distance traveled can be computed by

$$d = v * \Delta t \tag{58}$$

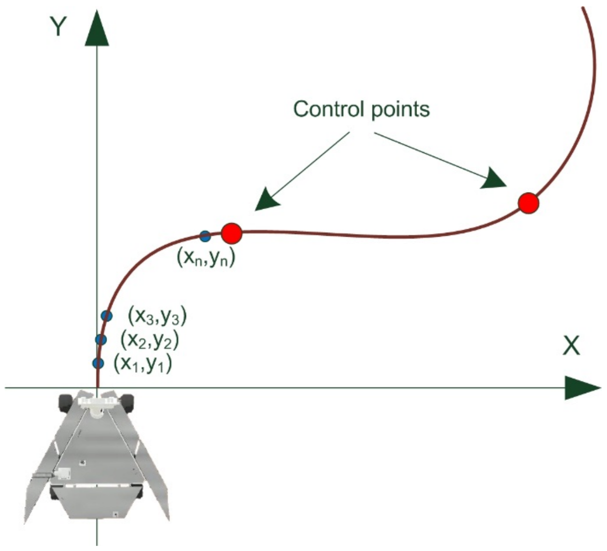


Figure 45. Computation of reference points (x_{ref}, y_{ref}) along the trajectory to command the control algorithm at regular time intervals

where v is the robot's nominal speed, Δt is the time elapsed from the beginning of a given interval's trajectory navigation process. To determine the coordinates for each reference point, knowing the distance traveled by the rover so far, it is necessary to calculate at which point, along the trajectory, the reference point shall be located. A point along a polynomial spline-based trajectory is denoted by a value in the range [0, 1]. Therefore, knowing the distance traveled and the interval's length, it can be computed as:

$$i = d \,/\, length \tag{59}$$

where i is a value in the range [0, 1], d is the traveled distance computed using equation (58) and *length* is the interval's length, computed as previously indicated, see Algorithm 22. Therefore, the coordinates of a concrete reference point can be then calculated, and established as the next waypoint to be reached to be passed to the control algorithm, using the next formulas:

$$x_{ref} = x + P_k(i) \tag{60}$$

$$y_{ref} = y + P_k(i) \tag{61}$$

where (x, y) is the rover's initial location, and $P_k(i)$ is the value of the polynomial equation P_k at the point *i*.

In this case there is no reference orientation (α_{ref}) for a given waypoint –reference point- to be passed to the control algorithm (Algorithm 20), just its position (x_{ref}, y_{ref}), see Algorithm 23.

```
(x, y) = currentLocation

For each polynom $P_k$ in polynomialList
    length = computeIntervalLenght($P_k$)

    // compute interval's end-point
    $x_k$ = x + getPolynomValue($P_k$, 1)
    $y_k$ = y + getPolynomValue($P_k$, 1)

    $t_0$ = currentTime()

    setTimer(timeInterval)

    // generate reference points while interval's end-point not reached
    do

        t = currentTime()

        i = (NOMINAL_SPEED * (t - $t_0$)) / length

        $x_{ref}$ = x + getPolynomValue($P_k$, i)
        $y_{ref}$ = y + getPolynomValue($P_k$, i)

        call control algorithm to go to ($x_{ref}$, $y_{ref}$)

        waitForTimer()

    While (euclideanDistance(($x_{ref}$, $y_{ref}$), ($x_k$, $y_k$)) > TOLERANCE_PERIMETER)
```

Algorithm 23. Pseudo-code to compute intermediate waypoints -reference points- to describe a spline-based trajectory

4.3.4.5 Position Estimation

To execute the commanded path, the rover low-level control shall estimate its position (x, y) at all times along the way, see Figure 46.

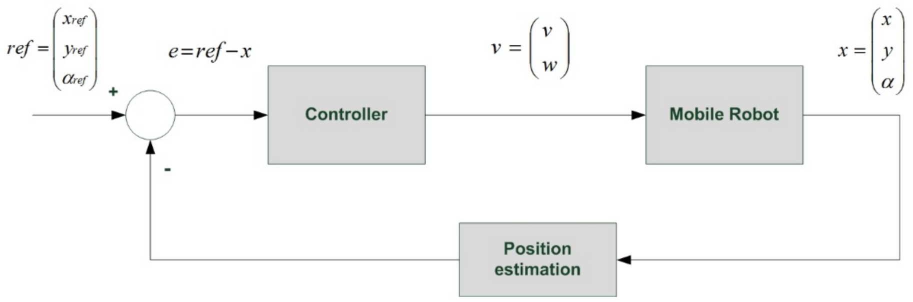

$ref = \begin{pmatrix} x_{ref} \\ y_{ref} \\ \alpha_{ref} \end{pmatrix}$
$e = ref - x$
$v = \begin{pmatrix} v \\ w \end{pmatrix}$
$x = \begin{pmatrix} x \\ y \\ \alpha \end{pmatrix}$
+
-
Controller
Mobile Robot
Position estimation

Figure 46. Rover's feedback control and position estimation

The position estimation is addressed following the differential steering model. The method for estimating the rover's location is equivalent for both the simulated 6-wheels rover, shown in Figure 3, and the 4-wheels rover used for field testing, shown in Figure 53. This model can be used to predict how a robot will respond to changes in its wheel speed and what path it will follow under various conditions. The model can also be used to calculate a robot's position in dead-reckoning or localization by odometry applications (techniques that estimate a robot's position based on distances measured with odometer devices mounted on each wheel) (Lucas, 2000). At the end of each step, sensors provide a reasonably accurate estimation of the rover's new position. The relative position and orientation of the rover can be reasonably inferred using odometry and provided as input.

It is worth emphasizing that the equations given next represent an elementary model for the motion of a robot or vehicle. They describe the robot's position and orientation as a function of the movement of its wheels, but ignore the underlying physics involved in making that motion happen. Issues such as torques and forces, friction, energy and inertia are not taken into account in this model. In technical terms, this method of describing motion is referred to as a *kinematics approach*. It ignores the causes of motion (which would be the *dynamics approach*) and focuses on the effects. In this initial and simplified approach, it is also ignores details of motors, gearing, electromagnetics, power supplies, and other engineering considerations that make wheel-based actuators possible. However, this initial approach is enough to estimate robot movement for the scope of this work. As indicated at the beginning of this thesis, the objective is to design and develop high-level autonomous navigation approaches, for analysis of strategies and validation of algorithms at the functional level and not analyzing aspects such as advanced mechanics and locomotion or sophisticated wheel-soil contact forces, which is more proper of a mechanical focus on the rover design. Moreover, regarding global positioning errors, in the case of planetary robot where an exploratory strategy based on the computation of candidates paths is followed, it is not required that the rover motion exactly matches that which was commanded, but it does assume that wherever the rover ended up, its relative position and orientation can be reasonably inferred and provided as input, as sensors are expected to provide a reasonably accurate estimate.

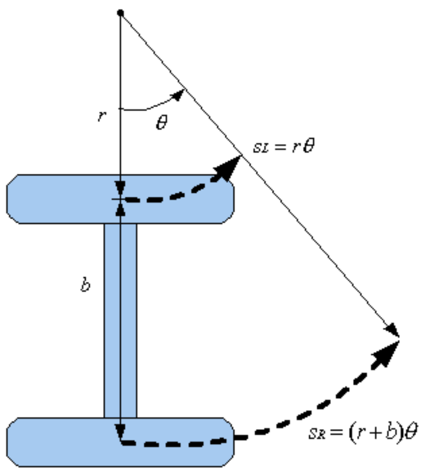


Figure 47. Path of wheels through a turn

Going back to the differential drive model, it is usually applied to a wheeled mobile robot whose movement is based on two separately driven wheels placed on either side of the robot body. If both drive wheels turn in tandem, the robot moves in a straight line. If one wheel rotates faster than the other, the robot follows a curved path inward toward the slower wheel. If the wheels turn at equal speed, but in opposite directions, the robot pivots. Thus, steering the robot is just a matter of varying the speeds of the drive wheels, see Figure 47.

SL, SR give the displacement (distance traveled) for the left and right wheels respectively, r is the turn radius for the inner (left) wheel, b is the distance between wheels (from center-to-center along the length of the axle) and θ is the angle of the turn in radians. SM is the speed at the center point on the main axle. It will be treated the axle's center point as the origin of the robot's frame of reference.

This exactly matches the model of the real robot used for field testing shown in Figure 53. As for the case of the 6-wheeled simulated model, shown in Figure 3, its mechanical configuration allows a steering robot system, where the steering wheels are turned to face the preferred direction and then the traction wheels are actuated to move the robot towards the indicated direction, like in a car. However, given the focus of this work is the high-level autonomous navigation strategy, for simplicity it has been approximated to the differential drive model, so that the same model and approach will be used for both, the simulated and the real robots.

Following Figure 47, the next relationships are observed:

$$SL = r\,\theta \tag{62}$$

$$SR = (r + b)\,\theta \tag{63}$$

$$SM = \left(\frac{r + b}{2}\right)\theta \tag{64}$$

Once established the simple geometry for the differential steering system, it is easy to develop algorithms for controlling the robot's path. In our case, as our robot's have more than one wheel per side, see Figure 48, a mean on the wheel's speed for each side of the robot is calculated to comply with the differential drive model, following the next equation:

$$v_L = \frac{v_1 + v_3 + (v_5)}{{}^{n}/_{2}} \tag{65}$$

$$v_R = \frac{v_2 + v_4 + (v_6)}{n/2} \tag{66}$$

where v_x is the velocity of wheel x, v_L and v_R are the calculated mean velocity of left and right wheel(s) respectively and *n* is the number of wheels. Velocities are expressed in meters per second.

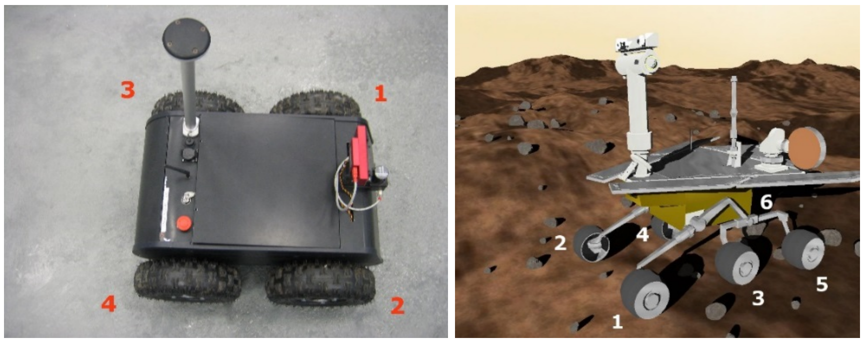


Figure 48. Mobile platforms' wheel numbering

Once computed the right and left velocities, the linear and angular velocities of the system can be derived as:

$$v = \frac{v_L + v_R}{2} \tag{67}$$

$$w = \frac{v_L - v_R}{b} \tag{68}$$

where *v* is the linear velocity of the system, expressed in meters per second, and *w* is the angular velocity of the system, expressed in radians per second. It can be then decomposed in v_x, v_y, calculating the orientation increment.

$$\Delta\theta = w * t \tag{69}$$

$$\theta = \theta + \Delta\theta \tag{70}$$

$$v_x = v * \cos(\theta) \tag{71}$$

$$v_y = v * \sin(\theta) \tag{72}$$

where Δθ is the orientation increment (delta) in a given period of time (t), which is the control cycle frequency, in our case 20 ms. θ represents the current robot's orientation, in radians, and v_x and v_y the velocities of the system in the X and Y axes. Orientation computed by this method is typically very noisy and accumulate errors very quickly; therefore, IMUs and sensor fusion strategies –kalman filter- are usually employed to compute the robot's orientation. Having these v_x and v_y velocities, the distance increment –relative position- can be calculated by:

$$\Delta p_x = v_x * t \tag{73}$$

$$\Delta p_y = v_y * t \tag{74}$$

And so the new rover's position will be:

$$\mathrm{x} = x + \Delta p_x \tag{75}$$

$$\mathrm{y} = y + \Delta p_y \tag{76}$$

There is a source of uncertainty, as the position estimation in differential-drive mobile robots suffers from systematic and non-systematic errors. The non-systematic errors are independent of the robot, caused by effects such as wheel slippage, traction loss or an imperfect floor surface. They cannot be compensated for as they are random errors.

For the systematic errors, there are two dominant sources: unequal wheel diameters (E_d) and the uncertainty about the effective wheelbase (E_b). These errors are vehicle-specific and don't usually change during a run (although different load distributions can change some systematic errors quantitatively). It is important to note that E_b has an effect only when the robot is turning, while E_d affects only straight line motion, as was shown by Borenstein and Feng (1995). E_d and E_b are dimensionless values, expressed as fractions of the nominal value.

Regarding E_d, most mobile robots use rubber tires to improve traction. These tires are difficult to manufacture to exactly the same diameter. Furthermore, rubber tires compress differently under asymmetric load distribution. Usually, in planetary rovers' designs metal wheels are employed instead of rubber tires, avoiding this effect. However, for any robot using this kind of tires, as the one employed for field trials, shown in Figure 53, this effect must be taken into account. It can be defined as:

$$E_d = \frac{D_R}{D_L} \tag{77}$$

where D_R and D_L are the actual wheel diameters.

Uncertainty in the effective wheelbase (E_b) is caused by the fact that rubber tires contact the floor not in one point, but rather in a contact area. The resulting uncertainty about the effective wheelbase can be on the order of 1% in some commercially available robots. This error can be defined by:

$$E_b = \frac{b_{actual}}{b_{nominal}} \tag{78}$$

where b is the wheelbase of the vehicle. Odometry can be improved generally by measuring the individual contribution of these two dominant errors sources, and then counter-acting their effect in software, as shown in (Borenstein and Feng, 1995).

There are also some other ways to deal with uncertainty in localization in this domain. These position estimation errors are commonly eliminated when the rover receives its daily activities plan from ground control. Scientists on Earth provide position and orientation corrections using data they have available, both from the rover instruments and sensors and from orbiters, resetting errors. Another technique commonly used for autonomous robot relocalization in robotic planetary exploration is visual odometry (Angelova et al., 2007; Helmick et al., 2009). It is a process for determining the position and orientation of a robot by analyzing the associated camera images, using sequential images to estimate the distance travelled. The process consists of detecting and extracting certain features and matching them across consecutive frames by correlation, estimating the camera motion from the optical flow. However, the navigation algorithm does not require that the rover motion exactly match that which was commanded, as each waypoint, or next portion of the path, is computed from the current location and the height map is updated every cycle with new data from perceptions. It first checks if the rover has already reached its goal, within some tolerance perimeter around it. If so, the navigation process has completed and the traverse will terminate successfully. Otherwise, a new waypoint and a path to it is computed and sent to the low level controller to execute the trajectory. While the rover is driving its next step, it will not use its imaging sensors to look for obstacles. Other types of safeguarding are enabled while driving, like tilt sensors and inertial measurement units, but no additional high-level terrain-based planning or sensing need to be performed.

For this *navigation* subsystem several parameters can be established by the operator such as the nominal and maximum linear and angular velocities, maximum steering angle or localization accuracy, to establish a tolerance perimeter around the waypoint to determine when it has been reached.

4.3.5 Processes and Control Flow

As introduced previously, the rover reaches a commanded location by performing subsequent navigation cycles, moving from one waypoint to the next, see Figure 7. These waypoints cannot be computed from the beginning, as there is no a priori information. Therefore, the navigation strategy follows an exploratory approach; at each cycle, the rover stops to perceive the environment and update its map, determines a new waypoint in the route and computes a trajectory to it. Figure 49 shows the rover model in a terrain in the simulator. The goal location has been set several meters away from the rover, marked with an end-of-race flag. The red path is the result of the concatenation of a series of paths, computed one at each cycle, where the end point of each one concurs with the starting point for the next. The intermediate waypoints are represented by blue flags in the figure.

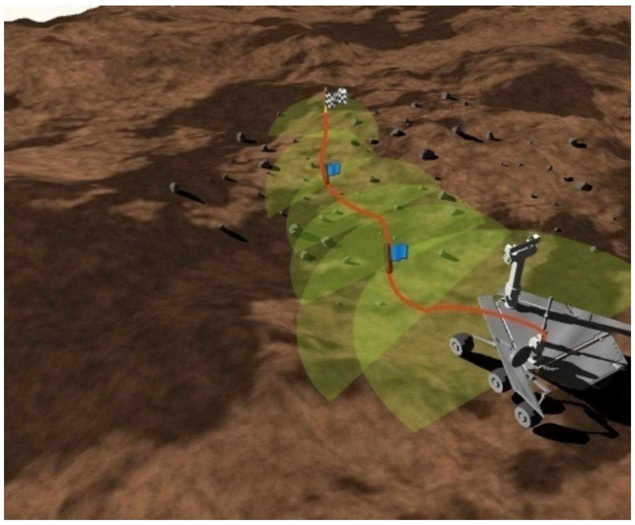


Figure 49. Several navigation cycles –waypoints- are usually necessary to reach a given location

To accomplish this, the autonomous navigation system, is composed of a set of subsystems, as seen before, and each one comprises a series of processes. Figure 50 shows these subsystems, which are mainly: configuration, robot, communications, vision, mapping, path planning and navigation, as well as the data flow among them. The *configuration* subsystem includes modules to parse configuration files with mission and operation parameters, which can be updated anytime by the operator. These parameters are stored in an internal data structure and used throughout the system by all modules. The *robot* subsystem includes functions to manage the robot's hardware, devices, sensors and actuators, creating a hardware-independent interface to the rest of subsystems and modules. The *communications* subsystem contains the downlink and uplink modules to manage the rover interactions with ground control, receiving activity plans and commands and sending back data from sensors, messages, alarms, status information and computation results. The modules within the *vision* subsystem make the system's perception up, processing the acquired images, executing the stereo process, producing a cloud of 3D points and performing the necessary system reference transformations. The *mapping* subsystem manages the rover's internal map of the world created from perceptions; it includes features such as building a map, updating it with new data when available, merging maps and evaluating certain areas within the map to identify safe or non-navigable areas for the rover. Modules within the *path planning* subsystem receive updated maps and compute suitable trajectories to given locations as a function of the configuration parameters, meeting mission criteria and safety constraints. In this module, the estimation of the rover's location and orientation are computed from onboard sensors data, such as encoders or inertial measurement units. The *navigation* subsystem coordinates the system's autonomous navigation capabilities. It acts as a central manager where other processes are instantiated from as required, organizing the obtained results and data flows. Once a suitable trajectory has been computed, the execute path module decomposes it in the appropriate motion commands to be sent to the rover's low level controller.

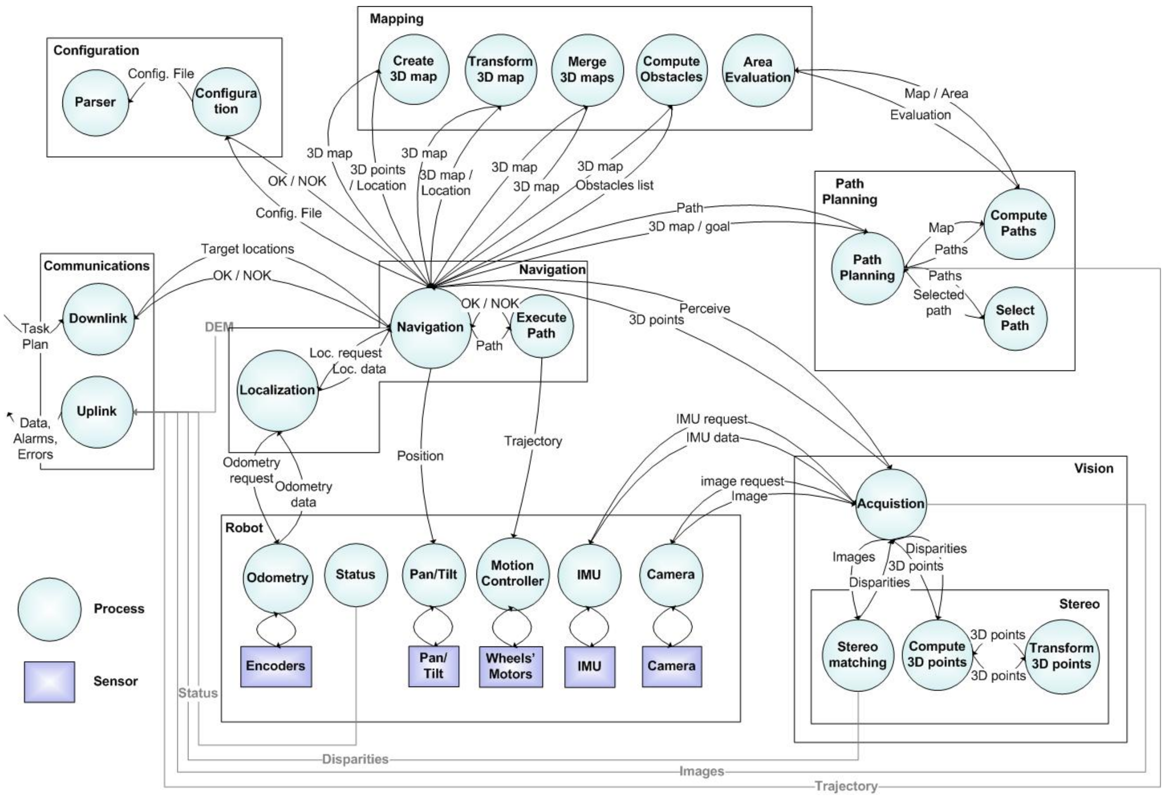


Figure 50. Autonomous navigation processes and control flow

4.4 Testing, Experiments and Validation

4.4.1 Testing and Validation of Algorithms

A visual-based autonomous navigation strategy is a complex software system, comprised of large amounts of code and algorithms performing computations interacting among them. Before embedding the software into a physical robotic platform, it is advisable testing the approaches and algorithms and validate them, to avoid damaging the hardware, usually quite expensive, which would increase the costs and delay the project planning. The framework introduced in this chapter and its simulation capabilities have been exhaustively used for this purpose. A large set of tests have been performed both during development and in final test campaigns.

In this chapter results, measures of performance and computing times are presented, obtained from a concrete test and settings; pretending to serve as an example to illustrate the system performance. Equivalent results have been obtained for the complete set of tests performed, according to the parameters established in each case. The scenario consists on a Mars-like terrain model, as the one shown in Figure 4, and a robotic vehicle model, based on the NASA MER mission rover, shown in Figure 3. Some of the most relevant system parameters have been configured as follows:

- Stereo camera resolution: 640x480
- Stereobase (separation): 9 cm
- Camera focal distance: 3 mm
- Camera field of view: 94º (horizontal), 69º (vertical)
- Two perceptions per cycle, ± 35º each from the front line
- Stored terrain map's size: 10x10 m
- Terrain map's resolution: 5 cm (grid cell's size)
- Mapping interpolation method: average of the 8 nearest-neighbors, with at least 2 valid values
- Path planning method: straight paths

- Number of candidate paths: 23
- Planning distance: 1.5 m
- Navigation distance: 1 m
- Maximum step hazard allowed: 8.5 cm
- Maximum roughness hazard allowed: 13 cm
- Maximum tilt hazard allowed: 20º
- Nominal speed: 5 cm/s

These parameters values have been configured in a very conservative way, as in real mission scenarios; a low nominal speed has been set as well as short planning and navigation distances, forcing the rover making perceptions more frequently to update the map and planning short paths, maximizing safety. In case the rover traverses a flat and low-risk terrain, the operators may set less conservative parameters.

Having the rover in a random location, a target position 4.8 m ahead and 2.3 m right, is commanded to the robot from the control station. The autonomous navigation process described previously, sketched in Figure 50, is initiated. It starts by perceiving the environment, then it updates its map and computes a trajectory. Using the control center in debugging mode, telemetry is continually received in real-time, so that the rover location is known at all times. Figure 51 shows how the robot performs 6 navigation cycles to reach the commanded target location and the sequence of intermediate waypoints and paths to them it computes from the starting to the goal location, as well as the distances traveled between each pair of waypoints.

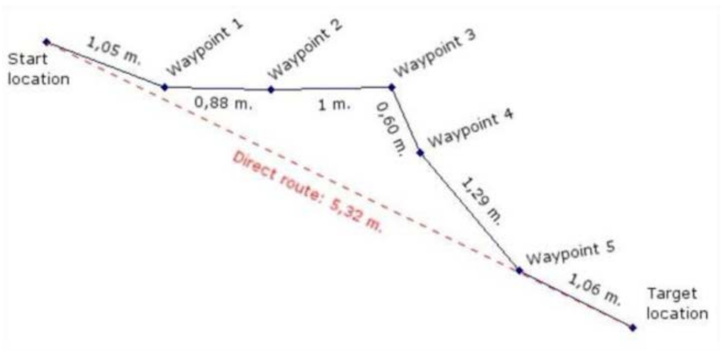


Figure 51. Rover route composed of several waypoints

The straight distance to the goal location is 5.32 m. However, terrain roughness, rocks and obstacles prevent the rover to go directly, computing suitable trajectories according to safety criteria and mission parameters established. The rover performs six navigation cycles, travelling a total distance of 5.88 m in 9:29 min. Table 1 shows the navigation process timing during this test, measured in seconds. *Computation time* accounts for the period of time the rover is stopped performing internal calculations, in seconds. *Navigation time* measures the period the rover is actually moving through the terrain, in seconds. *Turn* indicates how much the rover rotates, in radians, to face the selected path direction on each cycle and how long it takes. And *distance* accounts for the actual distance traveled on each cycle, in meters.

Table 1. Navigation process timing

Cycle	**Total time (s)**	**Comp. time (s)**	**Nav. time (s)**	**Turn (rad)**	**Distance (m)**
1	85	52	33	0.29 (9s)	1.05
2	79	46	33	-0.43 (12s)	0.88
3	68	47	21	0 (0 s)	1.00
4	118	62	55	1.29 (40s)	0.60
5	105	63	41	-0.43 (12s)	1.29
6	114	78	35	-0.43 (12s)	1.06
	9:29	5:48	3:38		5.88

The computation time is usually higher than navigation time. The rover drives short distances, taking not very long to navigate. An influencing factor that increases the computation time is also the time required to move the pan/tilt, pointing the camera towards the right direction previous the perception process. Two perceptions are performed each cycle; first, the camera is pointed to the left a given offset from the central position to take a pair or images. Then, the camera is moved to the right to take another pair of images, and finally it is left in the central position. Just those three moves take approximately 28-30 seconds out of the computation time each navigation cycle. There is also another interesting aspect influencing the computation time. As the algorithm evaluates candidate paths in order starting with the more directed one to the target, the more paths are evaluated the more the computation time increases. That can be seen in cases like cycle 4; where the selected path is 1.29 radians from the current orientation, indicating many paths have been evaluated. Concretely, seven paths were evaluated in cycle 4 in contrast with just one path in cycle 3.

Regarding navigation time, the robot is configured to drive at a nominal speed of 5 cm/s. This is a low speed, making an efficient use of energy, which is a limited resource in robotic planetary explorer, usually obtaining it from solar panels, and a conservative approach, ensuring the robot can frequently check its inertial measurement unit as it drives to guarantee its safety. Also, in this experiment the candidate paths are computed as straight trajectories; the rover makes on-site turns to face the selected path before actually following it, Figure 52 shows a sequence of moves through the environment; four navigation cycles are shown. The planning area configured for the path planning module is represented by the yellow circle, establishing the maximum length for the set of candidate paths to be computed and evaluated on each cycle; then the selected path is followed for a given *navigation distance*, represented by the red trail in the figure.

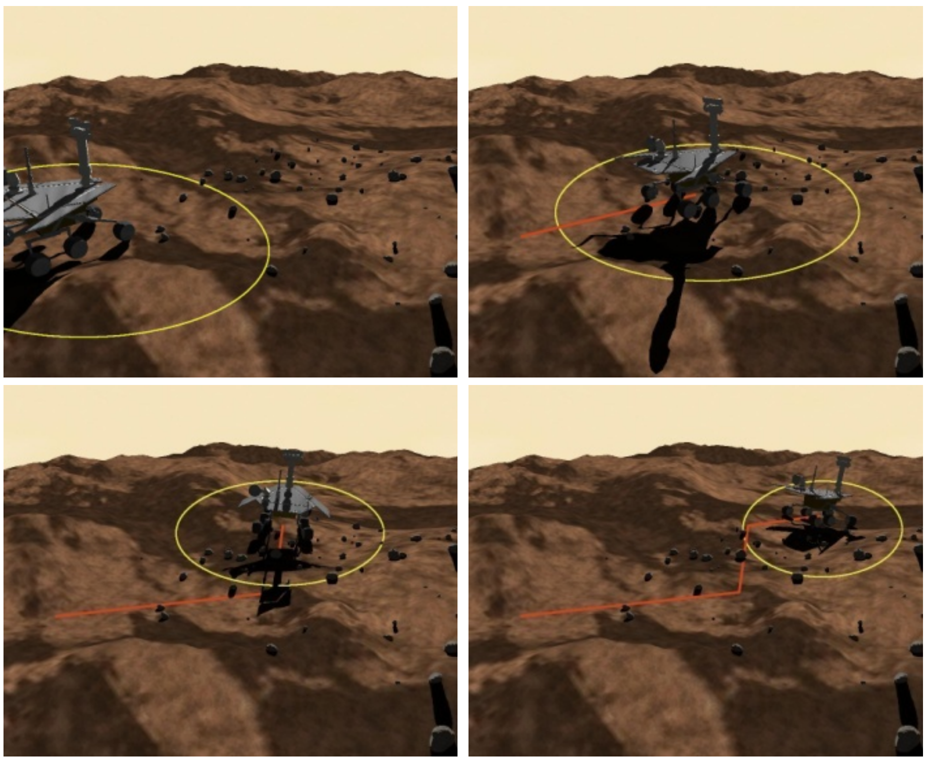


Figure 52. Multiple waypoints path, showing the planning area (yellow) and followed trajectory (red)

When the path planning module is configured to compute straight candidate paths, the longer the turns to face the selected path the longer time it takes. That is the case in cycle 4, where the rover takes 40 out of the 55 seconds of the navigation time to turn 1.29 rad.

Straight trajectories computation is a simple strategy, and the least computational resources demanding one, but a very interesting approach to be used for comparison with others methods such as arcs or the more sophisticated splines. The distance driven each cycle is not always the same, as it depends on the localization estimation from odometry and the tolerance perimeter established around the waypoint to determine when it has been reached.

Besides times per navigation cycle, accurate measures on several processes' computing demands have been obtained from the framework. As it runs on a desktop PC, which is not a real-time system, and these processes share the processor with other tasks executed in the system, CPU clock tics have been counted for each process, shown in Table 2. In the space exploration domain only certified hardware can be employed, where resources are considerably constrained; so it must be expected processes will take longer.

Table 2. Computing time on a PC Intel Core2 1.86 Ghz.

Function		Computing time (ms)
Stereo matching		410-480
	Disparity filtering	90-130
Computing 3D points		430-470
	Reprojection	210-320
Height map construction		70-90
Height map interpolation		< 10
Height map update		< 10
Merge height maps		< 10
Path planning process		40-50

The actual CPU computing time at each navigation cycle, the time the rover is busy with calculations, excluding camera moves, is 2-3 seconds. It can be seen that the heaviest processes are the computer vision related ones. The entire perception process takes between 360 to 600 ms. The *disparity filtering* process is part of the stereo matching algorithm. The *stereo matching* process is very dependent of the images' resolution, as the stereo algorithm has to deal with every pixel in both images looking for matches, many of them several times depending on the established search window. In this case 640x480 images, 5x5 windows and a maximum disparity value of 64 pixels have been used. *Reprojection* is part of the 3D points computation, including perspective transformations, where the points list is transformed to the rover reference system. Operations with height maps depend on map's size and resolution; for these experiments a 10x10 meters map with 5 cm resolution has been configured.

At this point, the algorithms have evolved through an iterative code refinement process and thoughtful testing, simulation experiments have been carried out and measures and performance data have been obtained from the framework; the software can be considered mature enough to be tested with real hardware in a physical platform.

4.4.2 Field Testing

The ultimate step in the validation process is field testing, where strategies and procedures are tested under real settings, analyzing measures and data obtained to assess its performance and correctness. This section describes the characteristics and configuration of the mobile platform used for testing and experiments, time performance analysis of embedded algorithms and path planning and navigation accuracy details.

4.4.2.1 Mobile Robotic Platform

The navigation system has been ported to a mobile robotic platform; the Movirobotics MR7, shown in Figure 53, along with its simulated model created for the framework. It is a powerful 4-wheeled all-terrain rover with 60 kg payload and maximum speed of 1.8 m/s. It weighs 28 kg, has a clearance of 11cm and 30cm diameter wheels. It is not steerable, turning by differential drive. Although its design does not exactly match a typical Martian rover, it is still an adequate platform to test the navigation strategy. Regarding sensors, the platform has been equipped with a Videre stereo camera STH-DCSG 9mm (base-line), a pan/tilt unit, an inertial measurement unit and wheel encoders. It counts with an onboard PC104, AMD LX800 processor.

In order to verify and validate the system and to demonstrate the flexibility of the framework and the utility of the simulation subsystem, a correlation between the virtual and the hardware tests have been accomplished by importing the design of the MR7 rover in the framework and modeling all its subsystems, as shown in Figure 53. Equivalent tests have been carried out, to compare results obtained using the simulation tool and models, analogous to the ones presented in the previous chapter (Table 2), and results obtained using the real hardware in a physical environment (Table 3).

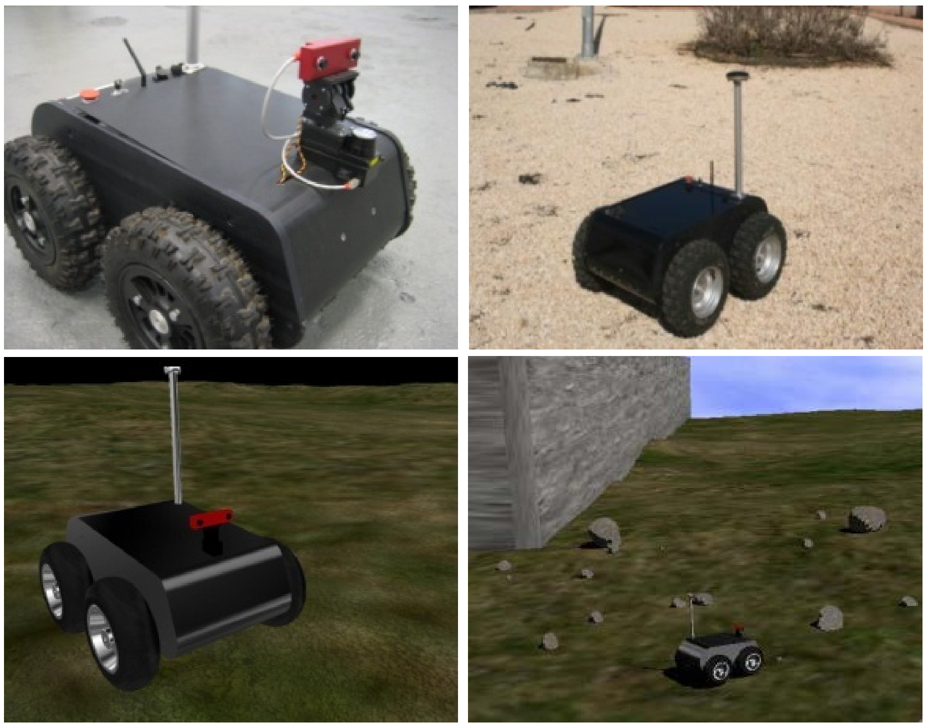


Figure 53. Mobile platform used for field testing and its equivalent simulated model in the framework

4.4.2.2 Time Performance Analysis of the Navigation Strategy

The porting process has consisted on four main steps: 1) adaptation of the hardware abstraction layer; 2) replication of the runtime environment in the onboard computer, to execute the navigation system in the robot; 3) system and parameters configuration and 4) algorithms' adaptations to the real operational conditions, filling the gap between the simulated and the real world.

Adapting the hardware abstraction layer implies including and developing the necessary drivers to manage the concrete underlying hardware and robot's devices so that the porting is a transparent process to the higher-level software, which remains unaware. For the replication of the runtime environment, as the development host and the robot onboard PC have both an equivalent operative system, based on Linux, the porting process has been greatly simplified, just taking the binaries and the right version of the required dynamic libraries and dependencies. In case they were different systems, the source code shall be specifically adapted and recompiled to use the libraries and services provided by the target platform. Once the navigation system has been ported to the target platform, the robot mechanical description –body and wheels' size, clearance, camera fixing position, and other parameters have to be adjusted, as well as mission, operational and safety constrains, including perception, mapping, path planning and navigation parameters.

Finally, although a given algorithm may work reasonably well in simulation, it may fail or produce incorrect results under real conditions and may need to be adapted. The most critical points of the system are the processes handling data coming from sensors, which are the interfaces with the external world. Unless purposely designed, simulated sensors tend to deliver perfect, error-free data; that is almost never the case when working with real devices. In this case, the main and more complex source of data is the stereo camera; the stereo vision process is critical as perception is the entry point to the internal rover computations. It determines the quality of the data obtained from the environment, used to create and update the world map, which directly influences the path planning process and ultimately the whole rover navigation capabilities; once the pair of images has been obtained and processed, the subsequent processes perform mostly derivations and transformations of this input, so it is a critical piece the rest of the system depends on.

Images from real stereo vision systems can be slightly misaligned, due to the visual system mechanical configuration, as well as differences between images of the pair usually appear due to the different characteristics of each optic, illumination effects, focus and shutter settings on each lens. None of these effects are typically present in synthetic images. Figure 66 shows a pair of stereo-images taken with the onboard camera, where some illumination differences can be appreciated between them. It can be seen the algorithm finds a small number of correspondences, insufficient for a quality 3D terrain reconstruction and mapping. After some tests with the physical robot and devices, it was determined some adaptations to the stereo matching algorithm were necessary to handle these issues. Details on the analysis, algorithms' adaptations carried out and obtained results can be found in (Correal et al., 2013), and will be addressed more deeply in following chapters. In advance, it can be said these adaptations have consisted basically in using a different matching algorithm, the semi-global block-matching, and the implementation of a series of processes to manipulate the input images and filter the obtained correspondences, such as the homomorphic filtering, histogram matching and clustering filter, described in detail in chapter 6. The results can be observed in Figure 66(d), where a much higher number of matches are obtained for the same images. These results are systematically extrapolated to the whole set of images obtained during the field testing campaign. This was actually the main issue encountered when ported the system from the simulated to the real world. The subsequent processes involved in the rover's navigation, like map building/merging or path planning, needed no further adaptations other than adapting the configuration parameters according to the real platform and operational conditions such as chassis measures, planning and navigation distances, security thresholds or camera's focal length, resolution and stereo base, as they depend on the data obtained from the perception module.

Table 3. Computing time on a PC104 AMD LX800, 500 MHz.

Function	Computing time
Stereo matching	29.25-29.45 s
Disparity filtering	1.02-1.16 s
Computing 3D points	3.66-3.85 s
Reprojection	2.15-2.95 ms
Height map construction	460-560 ms
Height map interpolation	120-130 ms
Height map updating	< 10 ms
Merge height maps	< 10 ms
Path planning process	440-610 ms

Table 3 shows processes' computing time obtained from the set of field testing experiments executed on the robot's onboard processor. Same perception, mapping and navigation parameters have been used in simulation tests. It can be observed that processes take about one order of magnitude higher than when executed on the development PC, shown in Table 2. It is due to the constrained processing capabilities of the robot's onboard computer. However, it is comparable to the hardware used for planetary missions. As introduced before, in the space domain only certified hardware can be employed onboard spacecrafts, where resources are considerably constrained compared to common desktop computers; it is actually like using hardware from a decade ago or so.

The stereo process is the most demanding one; it is actually a bottleneck of the system. Besides the limited computing resources, processing real images makes finding matches harder, given the influence of illumination, optics' differences, misalignments, lack of homogeneity and texture richness, increasing the processing time in contrast to using synthetic generated images. To give some context to these numbers, they can be compared with the ones obtained from the NASA MER mission, focusing in the stereo correlation process, which is the most demanding and critical one. A simple stereo correspondence approach, a block matching algorithm based on sum of absolute differences with 7x7 windows, was employed by NASA. They downsample images to 256x256 and the algorithm takes about 30 seconds to compute; while the same algorithm runs at 30 Hz for 320×240 disparity maps on a 1.4 GHz Pentium M in other applications (Matthies et al., 2007). This is the same approach initially implemented to process simulated images, described in section 4.3.1 *Perception Subsystem*, later discarded in favor of a more sophisticated algorithm, the semi-global block-matching, Figure 66(d). In this case, a processing time close to 30 seconds was obtained in the onboard robot CPU, shown in Table 3. Within this time a series of filters purposely developed to enhance the results, both before and after the correspondence process, are also executed (Correal et al., 2013). Besides the higher number of correspondences obtained because of the algorithm itself, images are not downsampled, using the full 640x480 size, obtaining even more matches. 256x256 images can produce 65,536 potential matches, while 640x480 images contain 307,200 pixels. It results in a much more complete and correct 3D terrain reconstruction, increasing mission safety and having an important impact on the subsequent path planning process.

To put in relation the processing times obtained from the robot used in the field experiments and check the applicability of the navigation strategy to real operational settings, Table 4 shows computing power of several CPUs and boards used in these experiments as well as processors used in previous and prospected planetary mission. MIPS (Million of Instructions Per Second) are used as a measure for comparison. Algorithms were initially tested in simulation in a desktop computer, a PC Intel Core2 1.86 GHz. processor (1 Gb. RAM, 4 Mb. cache size, 1066 MHz. bus speed), obtaining the measures shown in Table 2. The MR7 rover used in field testing experiments, shown in Figure 53, includes a PM-LX 800 board (1 Gb. RAM, 128 Kb. cache size, 400 MT/s memory bus speed), obtaining the processing times shown in Table 3. The robot is currently being updated to use a PCM-3363 board with a higher computing power; preliminary tests show the stereo matching algorithm that takes about 30 s in the current onboard computer, is executed in this new board in around 1 second.

Table 4. CPUs' computing power employed in experiments and space missions

Robot/Board	CPU	MIPS
MER Rover	BAE RAD6000 (up to 25MHz)	35
Curiosity Rover	BAE RAD750 (up to 200MHz)	400
MR7 Rover	AMD LX800 500 MHz	1000
PCM-3363	Atom D525 Dual Core 1.8 GHz	4500
Exomars Rover	???	>1000

Increasingly higher computing capabilities are expected for future space missions. For the next Exomars mission the rover's onboard computer requirements is currently being analyzed; no concrete data is yet available but Thales Alenia Space, the main contractor, indicated a significant computational performance will be required, suggesting onboard computer power of at least 1000 MIPS. NASA just announced plans for a robust multi-year Mars program, including a new robotic science rover based on the Mars Science Laboratory (MSL) architecture set to launch in 2020; no data are yet available regarding computing power requirements.

There are alternatives to increase computing power. A promising line is exploiting flight qualified field programmable gate arrays (FPGAs) as computing elements. To increase speed further, the most time-consuming vision functions, like the SAD algorithm for stereo vision, can be moved into the FPGA logic. The implementation can been designed to be highly parallel and pipelined. It considerably reduces the computation time while improving the quality of disparity data allowing developing more sophisticated approaches and algorithms that can reduce noise, improving performance at occluding boundaries, reducing pixel locking, and attempting to learn predicting slippage ahead of the vehicle by regressing past slippage experience against the visual appearance of the terrain. Actually, JPL is developing a new flight computing system, called the Mobility Avionics Module (MAM), around the Xilinx Virtex-II Pro FPGA as the main computing element (Matthies et al., 2007). This FPGA includes PowerPC 405 (PPC405) processor hard cores that can be clocked at up to 300 MHz; the rest of the FPGA logic can be clocked at approximately 100 MHz. Initial tests show the entire stereo process, together with running rectification, pre-filtering and triangulation on the PPC405 takes 250 ms/frame. This is a vast speed-up over the MER flight processor capabilities.

4.4.2.3 Path Planning and Navigation Performance

In a set of 40 field testing experiments carried out using the mobile platform shown in Figure 53, the rover was placed at different random locations within the environment, containing rocks and obstacles. For each test, the rover is commanded a target position, as a relative displacement from its current location. In these experiments displacements are between 10-15 m ahead and ±5 m right/left. On each test, the rover tries to reach the commanded target position exploring the environment and computing suitable sub-trajectories, choosing among a set of candidate paths on each cycle. The process is the one described in section 4.3.4 *Navigation*, where each cycle consists in: a) perception; b) mapping updating; c) path planning; d) navigation to an intermediate target point. Thus, a set of cycles are required before the final position is reached. Regarding the results, both the global positioning error and the errors at each cycle have been analyzed. Figure 54 shows the global position errors for the set of experiments. Results obtained indicate the rover reaches its final target location with an average positioning error of 0.84 m and a standard deviation of 0.07 m. No orientation error is computed as the rover is commanded to reach a certain position, but without any specific orientation. Regarding the errors measured along the path, the average positioning error on each cycle is 0.08 m with a standard deviation of 0.007 m and an orientation error of 0.07 rad with a 0.006 rad of deviation (with a configured navigation distance of 1 m). These errors have been computed comparing rover positioning with ground-truth data, by measuring locations and rover's position using a laser telemeter, which has 1 mm. of accuracy.

Regarding global positioning errors, sensors are expected to provide a reasonably accurate estimate of the rover's new position. Following the same approach JPL did for their Mars rovers (Goldberg et al., 2002), an exploratory strategy based on the computation of candidates paths does not require that the rover motion exactly match that which was commanded, but it does assume that wherever the rover ended up, its relative position and orientation can be reasonably inferred and provided as input. Based on the results above and this reasoning, it can be inferred the navigation approach performs as expected, which allows validating the strategy and the proposed framework.

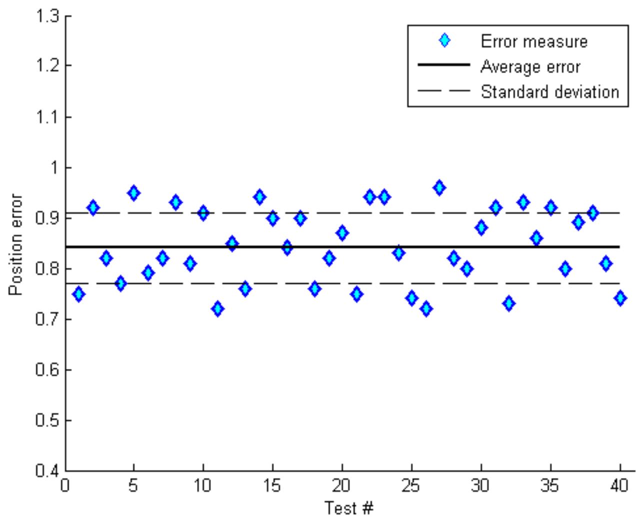


Figure 54. Positioning error at target location from navigation tests, average error and standard deviation

Chapter 5

THE PERCEPTION PHASE WITHIN THE AUTONOMOUS NAVIGATION PROCESS. A TESTBED FOR STEREO VISION ALGORITHMS

The perception phase within a robot's autonomous navigation system is a crucial step. It greatly influences the subsequent mapping and path planning phases, and therefore the whole navigation process as well as its safety and effectiveness. In the case of this work, perception is based on a stereo vision system. Some problems have been experienced working with this system; specifically noted when porting the algorithms from the simulated to the real world, as introduced in the *Field Testing* section in the previous chapter. This fact is detailed, as well as the characteristics of a tool specifically designed and built to address this problem: a testbed that aims to centralize and standardize the broad range and heterogeneity of existing stereo matching approaches, focusing in its application to real situations. It allows for configuring and executing algorithms, as well as comparing results, in a fast, easy and friendly setting. Algorithms can be combined so that a series of processes can be chained and executed consecutively, using the output of a process as input for other; additional filtering and image processing techniques have been included within the testbed for this purpose, such as homomorphic filtering, histogram matching and clustering filter, explained in detail in subsequent chapters. The testbed has been conceived as a collaborative and incremental open-source project, where its code is accessible and modifiable, with the objective of receiving contributions and release future versions to include new algorithms and features. It is actually available online for the research community; it counts a thousand downloads already. The usage of the testbed and its utility and application to a real problem is detailed in the form of use cases and experiments; it is illustrated how the testbed aided in the problem of porting the system from simulated to real settings, demonstrating its utility and usability. It has been crucial to support the developments and achievements presented in this thesis; a necessary resource that constitutes a contribution itself.

5.1 Stereoscopic Vision

In the domain of robotic autonomous navigation in natural, rough terrain and planetary rovers for space exploration, one of the most important features is 3D perception and terrain reconstruction for path planning and navigation.

Machine vision is an excellent sensor, widely used for a multitude of different applications. Concretely, stereoscopic vision has been widely used in many autonomous navigation approaches and space missions so far for 3D scene reconstruction. It is a mechanism to obtain depth or range data based on images. This process is similar to human binocular vision and our intuitive perception of depth, where the farther the objects are in the scene the less their position change when closing our eyes alternately. A similar principle happens in stereo vision: objects lying more far away correspondingly have a small difference, or disparity, between the images of the stereo pair.

According to Barnard and Fishler (1982) or Cochran and Medioni (1992) the classical problem of stereo analysis consists mainly of the following steps: image acquisition, camera modeling, feature extraction, image matching and depth determination. Of these, the matching process is the key step, described below.

To do that, the system consists of two cameras separated by a given distance, base-line (b), see Figure 55, so that two differing views of a scene are obtained, similar to human binocular vision. This pair of images is the input to the matching process; it is the process of identifying features in both images and compute the difference in position of that set of features or pixels from one image relative to the other, usually along the horizontal axis, obtaining a set of correspondences or disparities. Disparity is defined as the subtraction, from the left image to the right image, of the 2D coordinates of corresponding points in image space.

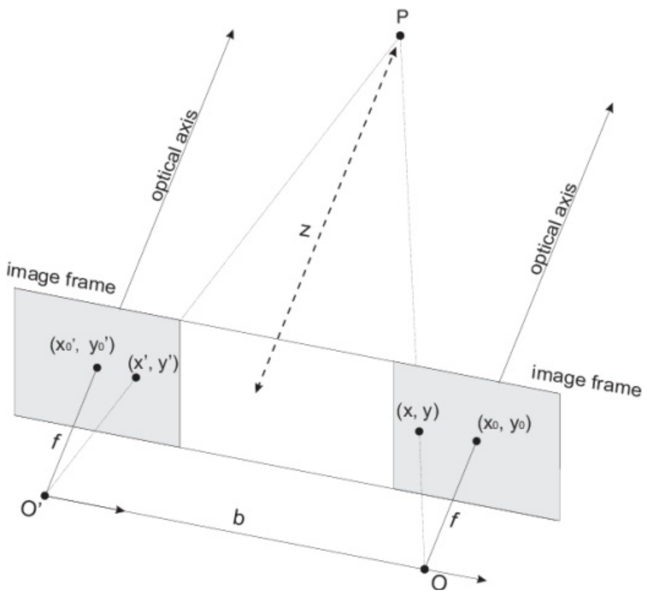


Figure 55. Stereo system of parallel axes. (x_0, y_0) and (x_0', y_0') are the images central points and (x, y) and (x', y') the coordinates of point P in each image of the pair (Bhatti, 2012)

As a result of the matching process, relative depth information is obtained, which are inversely proportional to distance to objects. Since depth is inversely proportional to disparity, there is obviously a nonlinear relationship between these two terms. When disparity is near 0, small disparity differences make for large depth differences. When disparity is large, small disparity differences do not change the depth by much. The consequence is that stereo vision systems have high depth resolution only for objects relatively near the camera, see Figure 56.

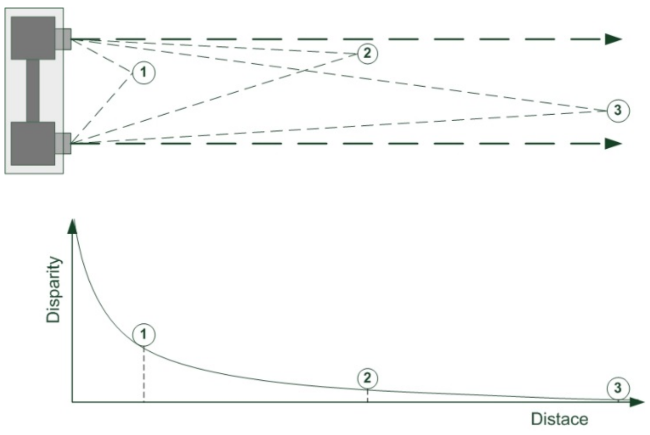


Figure 56. Depth and disparity are inversely related, so fine-grain depth measurement is restricted to nearby objects

Depth can be established by triangulation of the obtained disparities, provided that the position of centers of projection, the focal length and the orientation of the optical axis are known. Focal length, usually represented in millimeters (mm), is the basic description of a photographic lens. It is not a measurement of the actual length of a lens, but a calculation of an optical distance from the point where light rays converge to form a sharp image of an object to the digital sensor or 35mm film at the focal plane in the camera. The focal length of a lens is determined when the lens is focused at infinity. The focal length tells the angle of view—how much of the scene will be captured—and the magnification—how large individual elements will be. The longer the focal length, the narrower the angle of view and the higher the magnification. The shorter the focal length, the wider the angle of view and the lower the magnification. The line from the camera center perpendicular to the image plane is called the principal axis or optical axis of the camera. The plane parallel to the image plane containing the optical center is called the principal plane or focal plane of the camera. The relationship between the 3D coordinates of a scene point and the coordinates of its projection onto the image plane is described by the central or perspective projection.

Therefore, the point's coordinates in the camera system reference can be computed as (X', Y', Z) for the first camera and (X. Y, Z) for the second. It can be calculated how far away the matched point is (depth Z) by derivation of the next expression:

$$\gamma \cdot x' = f\frac{X'}{Z} \qquad \gamma \cdot x = f\frac{X}{Z} \tag{79}$$

$$Z = \frac{f(X'-X)}{\gamma(x'-x)} \tag{80}$$

knowing the camera intrinsic parameters: focal length (f), camera baseline (b), and pixel dimension γ.

5.2 Background

As introduced in the previous chapter, a visual-based autonomous navigation software that allows a robot to move safely through an unknown terrain has been designed and created (Correal and Pajares, 2011b). It uses stereo cameras mounted on the robot and builds a map of the environment as a result of a series of computations performed on the input images, described with more detail below.

In the initial phases of the development of the autonomous navigation software, when there wasn't any vehicle available, it was emulated using the simulation capabilities of the framework, where vehicles and terrains were replicated. Images of simulated terrains were captured from simulated cameras. Several stereo algorithms were evaluated to produce the disparity maps of those images. The stereo vision process was based in the Block-Matching (BM) algorithm (Konolige, 1997b). It produced very satisfactory results, finding almost 100% of the potential matches. Figure 57 shows an example of synthetically generated images from the simulator and the disparity map obtained using this algorithm.

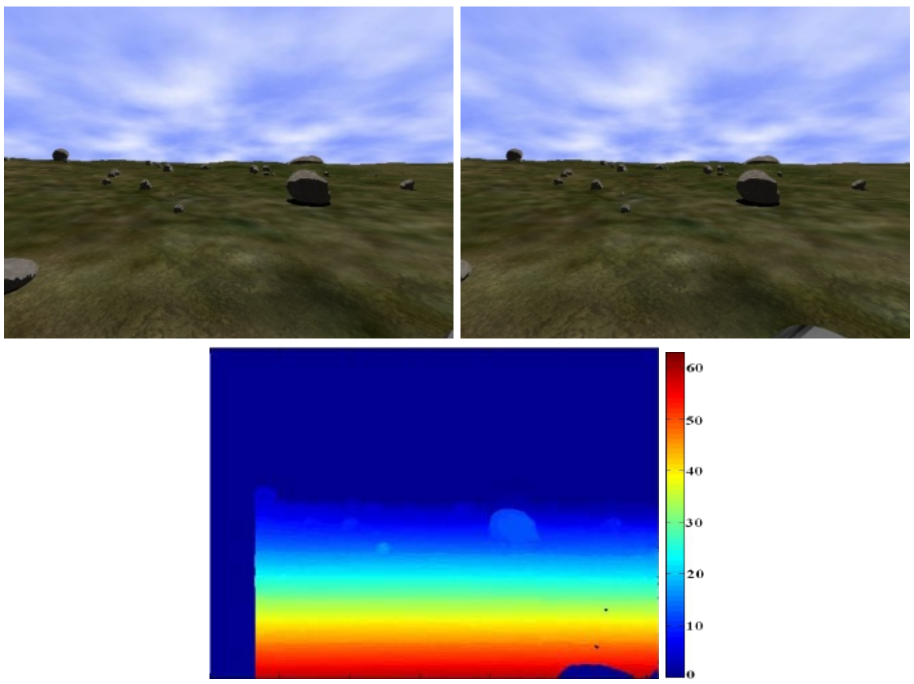


Figure 57. Synthetic stereo images obtained from the simulated environment and disparities map computed with the BM algorithm. Right side bar represents disparity values (maximum: 64 pixels)

The problem arose when the navigation software was ported to a real robot, equipped with a stereoscopic system of parallel optical axes Videre STH-DCSG-9 color 640x480, with two 3mm lenses. The same algorithm that obtained very satisfactory results with the synthetic images did not performed so well with the real ones. This was detected during the field testing phase, as indicated in the corresponding section of the previous chapter. Figure 58 shows a pair of real images obtained with this system and the disparity map the BM algorithm produces.

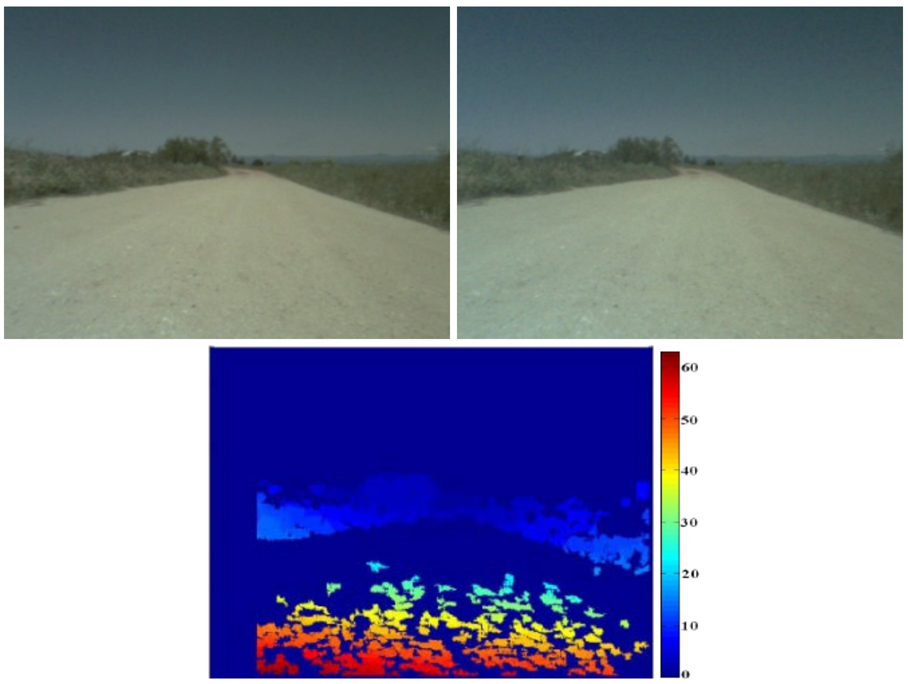


Figure 58. Disparities computed with the BM algorithm applied to real images

The pair of images shown in the previous Figure 58 contains 370,200 pixels, and 157,049 of them have potential matches; the other pixels belong to remote areas with no computable disparity, like the sky. The BM algorithm is able to find 57,598 correspondences, representing 36.68% of the total possible matches, see Figure 59. This results in a poor 3D reconstruction of the environment, producing a map with large empty areas, where accurate trajectories for robot navigation cannot be computed. This problem appears systematically in all stereo pairs analyzed from the real system, due to some factors present in the real world that do not occur in simulation, such as the different response from the camera sensors to the light from the scene; it produces different intensity levels, critically affecting the disparity computation. Therefore, some other algorithm has to be employed.

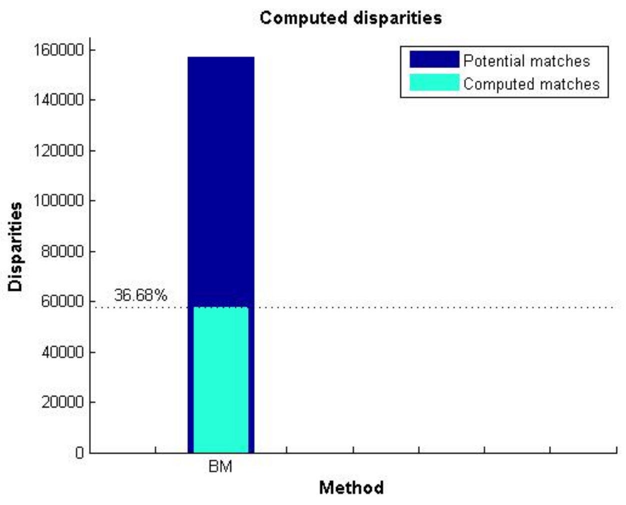


Figure 59. Disparities computed by the BM algorithm

A thoughtful analysis of available literature and existing stereo matching algorithms must be made in order to verify if there is any existing algorithm able to fulfill the requirements. It has been identified this situation happens recurrently; whenever the nature of the input images, the camera manufacturer or the operational conditions change the results are highly affected, and it is required to adapt the algorithms and its parameters or even design a new solution and implement a brand new and different approach and algorithm.

Actually, an analogous analysis process was already performed in previous phases when algorithms to process the synthetically generated images from the simulator were studied to be incorporated into the perception subsystem, described in section 4.3.1 *Perception Subsystem*, for the planetary exploration rover. As a result, the BM algorithm was selected then for implementation and included within this rover's autonomous navigation system, obtaining quite good results, see Figure 57. Now, this analysis must be repeated again to identify a suitable algorithm to process real images.

In accomplishing such a task, analyzing available literature and resources, some problems have to be faced: algorithms are implemented in different languages, input/output formats differ from one to another, tuning parameters requires often recompilation, etc. what slows down the process. This is that has motivated the design of a stereo testbed, described in this chapter, and where it provides a real advantage. As detailed next, it allows testing many different algorithms, easily tuning its parameters and comparing results in a very short time with just a few clicks, saving lots of time. The performance of the algorithms included within the testbed can be checked without the need to write any code, avoiding unnecessary efforts. This will facilitate the task of choosing a particular algorithm for stereovision when required, e.g.: for inclusion in the architecture proposed in the previous chapter.

5.2.1 Revision of Methods

Stereo vision is a heavily researched field; a vast amount of published literature and a wide range of algorithms, techniques, implementations and libraries are available (Scharstein and Szeliski, 2002; Tombari et al., 2011). However, there is no single approach that could be considered the best one, able to solve every possible problem. Usually, each technique is suitable for a set of conditions but not for others.

A review of the state of the art in stereo matching allows us to distinguish two main classes of techniques: feature-based (sparse) and area-based (dense) methods (Ozanian, 1995). Feature-based methods use sets of pixels with similar attributes, picking out feature points of high distinctiveness, either pixels belonging to edges (Grimson, 1985) or the corresponding edges themselves (Ayache and Feverjon, 1987); leading to a sparse depth map used mainly for object segmentation and recognition, sometimes reconstructing the rest of the surface by interpolation (Pajares et al., 2000a; Pajares and Cruz, 2006). Examples of feature-based approaches are SIFT (Scale-Invariant Feature Transform) or FAST (Features from Accelerated Segment Test). Area-based stereo algorithms are used instead to find matches for all points in the images; they use correlation between brightness (intensity) patterns in the local neighborhood of a pixel in one image with respect to the other (Baker, 1982); the number of possible matches is intrinsically high. These methods are useful if the system needs to recover the detailed geometry of the scene, in order to facilitate obstacle avoidance or object recognition. The work presented in this thesis is devoted solely to this last approach.

For any project that uses stereo vision, there is usually a set of requirements to be met. The first step is to analyze the state of the art, verify if there is already any algorithm or technique fulfilling the needs and to what extent. In case there is not any, a new strategy has to be developed, either from scratch or based on previous developments and existing approaches. This evaluation phase is not trivial at all. Given the range of available resources and literature, it is necessary to make a considerable effort to analyze and determine what strategy may fit each concrete set of requirements. This becomes especially crucial when stereo-based techniques are to be applied to real situations. Several problems arise when theoretical methods or strategies are to be investigated for posterior application in real scenarios. This chapter analyses such problems and provides a testbed for solving them.

One of the difficulties is checking the availability of a suitable implementation. Many algorithms and techniques are outlined within the published literatures, described in more or less detail depending on each publication; sometimes the level of detail covered is enough to implement it using any programming language, but sometimes it is not. There exist also some online sites containing stereo vision algorithms; in some cases they are available to be downloaded just in binary format, which may require a concrete platform such as Linux, Windows or any other system. In other cases source code access may be granted, finding algorithms implemented in C/C++, Java, Matlab or any other language, requiring compilation before using it. There are a series of computer vision libraries in a variety of languages, where some code has to be written to make use of them. This heterogeneity in the available resources and the lack of standardization makes the state of the art analysis, algorithms evaluation and comparison a hard, complicated and time consuming task. Therefore, it is crucial to establish a common framework where this broad spectrum of heterogeneous resources could be put together facilitating its analysis and comparison, and where they can be integrated to obtain even more sophisticated approaches.

A previous work in classification and characterization of stereo correspondence algorithms was made at Middlebury College by (Scharstein and Szeliski, 2002), where the authors presented a taxonomy of two-frame dense methods. They developed a stand-alone software containing some representative stereo algorithms implemented in C++ and a collection of multi-frame data sets. Despite the fact that it is a very interesting, useful approach and a reference work in the field, it presents some significant drawbacks. The source code has to be compiled by the user, which is not a trivial step. As the authors point out, while it is still being maintained, it does not represent an implementation of state of the art stereo methods. Also, in case a new algorithm is to be added in the future, as new approaches are constantly coming up, the internal software structure should be analyzed to understand where to properly insert it and the interdependencies with other modules, and recompile all together again. As the authors point out, there is no documentation provided beyond the comments embedded in the source code, what may be challenging for non-expert users. In addition, it does not allow including algorithms developed in different languages; despite many available algorithms are implemented in C++, there are also a large amount of resources developed using other technologies such as Java or Matlab. In this case, just C++ code can be integrated. Finally, given the lack of a graphical user interface, they proposed a sophisticated mechanism for specifying parameter values and input images that supports recursive script files, which syntax is not trivial. New scripts shall be created in case additional algorithms or images are included in a predefined structure of directories. Results are stored in text files, lacking of any graphical representation.

5.3 Stereo Testbed

This section describes the design and implementation experience of an open-source testbed to centralize, standardize and facilitate the evaluation, functional analysis and comparison of heterogeneous stereo vision matching algorithms. It allows the integration of different approaches and algorithms, both existing and forthcoming ones, developed by different authors in diverse languages, together under a common testbed representing a new approach in relation to previous works in the field. Additionally, this testbed includes a set of pre and post-filtering processes in order to correct problems derived from the illumination conditions in outdoor environments and remove spurious matched pixels, both representing important contributions.

Some other contributions of the presented testbed to the computer vision community with respect to previous related works cited above are:

- Portability, as the testbed is architecture and operative system independent.
- Openness of source code allowing analysis of algorithms and adaptations.
- Scalability of the system to incorporate new published algorithms and contributions in a collaborative and incremental approach.
- Flexibility in the inclusion of these algorithms in different programming languages.
- Use of diverse image formats.
- Graphical interface for friendly interaction and configuration of tests and algorithms.
- Graphical presentation of results.

This is intended with the aim of providing an efficient tool to apply stereovision-based strategies in real applications with high degree of efficiency and accuracy. Researches and developers will find a useful tool to guarantee the success and progress of their works when stereo-vision based methods are to be applied. This makes the main contribution of this work with a high content of applicability. The first version of this stereo testbed is already available online (Correal, 2012), and have had one thousand downloads up to now, denoting a strong interest by the research community.

5.3.1 Requirements

The main objective of the testbed is to focus on the problems of distribution of resources and lack of standardization, centralizing heterogeneous resources under a common framework. It purports to serve as a starting point, bringing different algorithms and approaches together in an organized way, facilitating its analysis, evaluation and comparison of results and reducing the necessary effort. The lack of standardization is one of the main issues. Implementations of some of the algorithms proposed in the literature can be found, however they are written in different languages and for different platforms, making its evaluation hard and time consuming. For that reason, this testbed aims to serve as a common platform to integrate such a variety, being language and platform transparent.

To allow algorithms and processes to interface with each other, inputs and outputs must be standardized, so that results from some processes can be used as inputs for others. It eases also the evaluation and comparison of algorithms by presenting results from different approaches in a common format, such as disparities obtained, right and wrong matches or processing time. The testbed shall support working with images in several formats (e.g.: jpg, bmp, png, pnm, etc), to use them as input for the algorithms, including and managing them in a straightforward way. It allows several algorithms to be compared using the same input images or a given algorithm to be tested using different images. Ideally each stereo pair would include ground truth information to verify the obtained results with trustworthy data, although this shall not be a requirement as this data is not always available.

Given the parametric nature of the stereo matching process and the large amount and variety of algorithms that can be included in the testbed, the algorithms selection and its configuration shall be done in an easy and straightforward way. This is important for testing purposes, as algorithms' parameters have to be frequently adjusted, given they critically influence the result and performance. There shall be no need to get into the source code for parameters definition or to modify its values, as it would require to recompile, implying a considerable effort to analyze and understand the code. Ideally it shall provide a graphical interface, both for configuration and presentation of results, avoiding the use of scripts and input/output text mode.

As researchers in the stereo vision community work in heterogeneity of platforms; the testbed shall be easily portable, capable of work in a multitude of systems, ideally avoiding recompilation of sources, which is often a challenge, achieving the highest possible degree of platform independency and portability. Regarding source code, although C++ is the dominant language for computer vision algorithms, many researchers use languages such as Java or Matlab to implement their works. The testbed shall allow for the inclusion of these heterogeneous resources without requiring to rewrite them in any other language, taking advantage of the work already done; it ease the necessary effort to integrate new resources into the testbed and allow to concentrate efforts in its evaluation instead. It leads to a core requirement of the testbed, which is openness. Researchers may have access to the sources of the algorithms included on it to extend or adapt any piece of code to each concrete needs, as well as to include new algorithms and processes. However, it shall be also possible to include resources in a closed form, such as libraries or compiled code, for cases where an author decides not to release its code or they are just not allowed to do it.

The testbed is expected to be a dynamic and evolutionary platform growing over time, in a collaborative and incremental development, releasing new and more sophisticated versions periodically including more algorithms and capabilities. For that reason, besides the capability of including new algorithms on a given local copy of the testbed, there shall be means to contribute with new resources to be included in subsequent releases.

5.3.2 Design

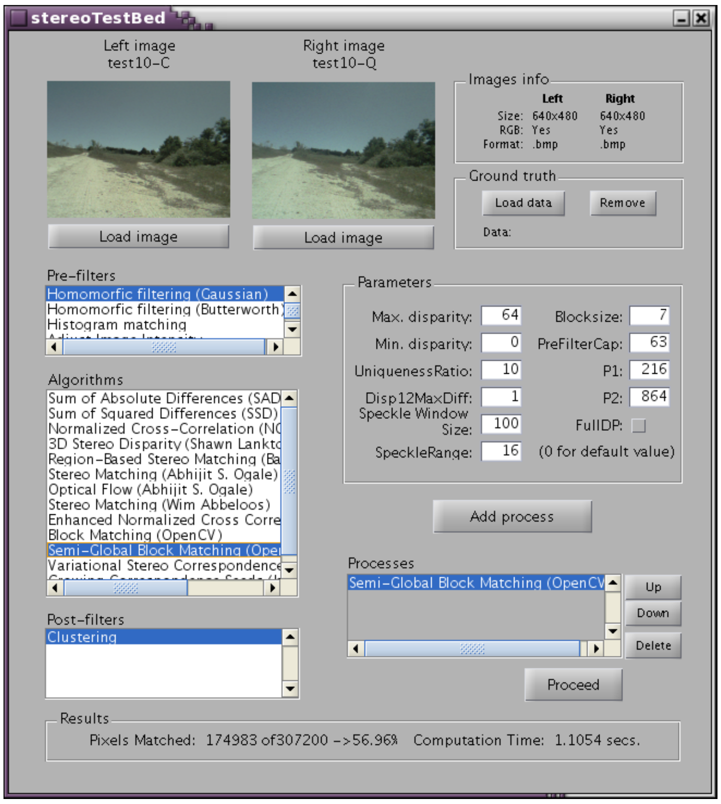
After a thoughtful analysis, Matlab has been chosen as the development platform to build up the testbed, as it allows meeting the set of requirements introduced in the previous section. Matlab is a well-known programming environment for algorithm development, data analysis, visualization and numerical computation. It is widely used in academic, research institutions and industrial enterprises, with a large community of users and a broad range of specific packages and toolboxes that help to reduce considerably the development efforts.

Regarding the requirement of addressing the heterogeneity of available resources, Matlab allows interfacing with programs written in other languages, such as C, C++, Java and Fortran, avoiding the need to rewrite them. Currently, there is a large number of stereo vision algorithms available implemented in C++. However, the number of stereo and computer vision resources available for the Matlab community is growing, as every day more researchers implement their algorithms directly using Matlab code. Computer vision libraries, such as the well-known OpenCV (Bradski and Kaehler, 2008) can be also interfaced with Matlab. It also allows addressing the fact the research community works in heterogeneity of platforms as there exist Matlab versions for different platforms such as Windows, Linux and Mac, so the stereo testbed and the algorithms included on it are easily portable, and capable of executing in a multitude of systems, avoiding recompilation of sources and meeting the transparency and OS independency requirements.

The testbed code itself is open and accessible, as it has been developed using Matlab code, as well as most of the algorithms included in the testbed, meeting the openness requirement, so they can both be extended and adapted to each concrete needs. But it allows also for the inclusion of resources in closed-source, as libraries or compiled files, using MEX-files -Matlab Executable. They are dynamically linked subroutines produced from C, C++ or Fortran that, when compiled, can be run from within Matlab in the same way as .m files -open source files- or built-in functions. Besides protecting the source code when access to it is not allowed, this format is more efficient, as these files execute much faster than interpret code. Although the speed of the stereo process is usually crucial and it is known code executed from Matlab is slower than native languages like C/C++, even for MEX-compiled files, it is important to emphasize the main focus of the presented stereo tested is not to include or evaluate the most efficient or real-time algorithmic implementation. In contrast, it is to facilitate the functional analysis of algorithms, checking performance and validating approaches, saving the effort of implementing algorithms just to test how they work. Once a concrete algorithm or process has been identified as functionally appropriate for a given set of requirements, efforts can be focused in coding just that one and in the most efficient way. There exist multiple approaches to optimize algorithms for embedded and real-time systems such as using low level languages -C, assembler-, using multi-processor boards for parallel computing or coding the algorithm in a FPGA (Darabiha et al., 2006; Lim et al., 2012).

Inputs and outputs have been standardized to ease algorithms integration with the testbed and to interface processes with each other. As introduced before, besides the stereo matching algorithms, the testbed includes a set of pre and post-filtering processes to correct problems derived from the illumination conditions and remove spurious matched pixels. Input to any matching algorithm has been established as a pair of stereo images and a structure with corresponding parameters' values, while the expected output is a disparity matrix. Any process or filter applicable prior to the matching step receive the stereo pair and parameters as input and return the same pair of images modified according to each process' functionality; post-filters take the disparities matrix computed in the matching process and return the improved and enhanced results, also as a disparities matrix.

To include any given algorithm within the testbed it does not require an intimate understanding of the algorithm. Actually it can be used as a black box. The only requirement is to adapt the parameters of the algorithm, generally by creating a wrapper, to transform the standardized inputs from the testbed described above to the format required by the concrete algorithm and the equivalent for the outputs. This standardization mechanism allows not just an easy integration of algorithms with the testbed but the possibility to interface with each other and chain processes. The only steps left are the inclusions of the algorithm in the appropriate list within the user interface, described below, include its call in the testbed code and add a graphical panel to tune its parameters, if any. Places in code to accomplish these steps are localized and details are clearly indicated within the testbed source code. Anyway, as indicated in the requirements section, new algorithms can be sent to the maintainers of the project for its inclusion in future versions. Once an algorithm has been integrated into the testbed, just a few clicks are necessary to select, configure and execute it.



Left image
test10-C

Right image
test10-Q

Images info

	Left	Right
Size:	640x480	640x480
RGB:	Yes	Yes
Format:	.bmp	.bmp

Ground truth
Data:

Pre-filters
- Homomorfic filtering (Gaussian)
- Homomorfic filtering (Butterworth)
- Histogram matching

Algorithms
- Sum of Absolute Differences (SAD
- Sum of Squared Differences (SSD)
- Normalized Cross-Correlation (NC
- 3D Stereo Disparity (Shawn Lankto
- Region-Based Stereo Matching (Ba
- Stereo Matching (Abhijit S. Ogale)
- Optical Flow (Abhijit S. Ogale)
- Stereo Matching (Wim Abbeloos)
- Enhanced Normalized Cross Corre
- Block Matching (OpenCV)
- Semi-Global Block Matching (Ope
- Variational Stereo Correspondence

Post-filters
- Clustering

Parameters

Max. disparity:	64	Blocksize:	7
Min. disparity:	0	PreFilterCap:	63
UniquenessRatio:	10	P1:	216
Disp12MaxDiff:	1	P2:	864
Speckle Window Size:	100	FullDP:	☐
SpeckleRange:	16	(0 for default value)	

Processes
- Semi-Global Block Matching (OpenCV

Results
Pixels Matched: 174983 of307200 ->56.96% Computation Time: 1.1054 secs.

Figure 60. Stereo Testbed user interface

To meet the requirement of creating a friendly user interface, the Matlab Graphical User Interface Development Environment –GUIDE- has been employed. It includes common components such as textboxes, labels, lists, radio/check buttons and panel, allowing the user to easily select, configure and execute any of the algorithms included. A series of contextual panels are presented to the user to set parameters' values in a straightforward way, avoiding any source code recompilation to adjust these values. Figure 60 shows the testbed graphical user interface.

Processes are organized in three categories within the user interface: pre-filters, matching algorithms and post-filters. *Pre-filters* refer to processes performed on the images before executing the matching algorithm, like adjusting images' intensities. The *algorithms* list includes the set of stereo matching algorithms available within the current version of the testbed. Processes listed in the *post-filters* list check the computed disparities and filter out wrong matches, improving and optimizing results. To set a chain of processes, each one is selected and configured independently. Once an algorithm has been selected from its list, its parameters, if any, are displayed within the graphical interface along with its default values, so they can be tuned to be adapted to the input images and application. It is then introduced in the *Processes* list, which contains the chain of algorithms to be executed so far. The order of execution and parameters of any process can be modified at any time using the appropriate controls within the user interface. Also, processes can be removed and included in the execution chain at any time, easily checking how it affects the whole process.

Input images can be loaded from a standard open dialog box, either from the datasets provided with the testbed or from anywhere in the system or external devices. As required, many different formats are supported: bmp, gif, jpg, pbm, pcx, pgm, png, pnm, ppm and tiff among others. Ground truth information can be optionally specified and loaded if available; it is actually an image representing real disparities. Ground-truth data, although not mandatory, is very useful for algorithms evaluation. After an algorithm is executed, the testbed reports the total number of potential matches, the number of right, wrong, false and missed correspondences obtained according to this ground truth data, both in absolute values and percentages. Final results and intermediate products are displayed as images, graphs or messages in console, including information such as the number of correspondences found, number of right, wrong and missed matches or execution time, easily comparing results from different algorithms; extended information is presented in case ground truth data is available.

5.3.3 Integrating Algorithms within the Testbed

For the first version of the testbed, a number of stereo matching algorithms and some additional image processes have been included. From the two approaches for stereo algorithms introduced previously, this work focuses on the area-based (dense) methods, given its applicability to this research domain, which is robotic autonomous navigation in outdoor terrains. However, the same approach is valid for featured-based and other stereo algorithms. They are listed below, as mentioned before, grouped in three categories: *pre-filters*, processes executed on the input images, matching algorithms, the actual correlation, *matching algorithms*, and *post-filters*, processes that filter out wrong matches and improve results, as well as a series of datasets for testing purposes.

Pre-filters:

- Homomorphic filtering (Gonzalez and Woods, 2008; Pajares and Cruz, 2007): based on the assumption that images can be expressed as the product of illumination and reflectance components. Illumination is associated with low frequencies of the Fourier transform and reflectance with high frequencies. Thanks to the product and applying logarithms followed by high pass filtering, illumination component can be eliminated. This allows retain only the reflectance associated with intrinsic properties of the objects in the scene. In this implementation there are two different filters available: a Butterworth High Pass (Butterworth, 1930) and a Gaussian filters. They can be applied in the RGB o HSV color models.
- Histogram Matching (Jensen, 1982; Gonzalez and Wintz, 1987): normalizes the images of the stereo pair, adjusting the color distribution of one image with respect to the other. It can be performed in the grayscale, RGB o HSV color models.

- Adjust image intensity: adjust the intensity values such that 1% of data is saturated at low and high intensities, increasing the contrast of each image of the stereo pair.

Matching algorithms:

- Sum of Absolute Differences, SAD (Barnea and Silverman, 1972): a classical approach. It measures similarity between image blocks by taking the absolute difference between each pixel in the original block and the corresponding pixel in the block being used for comparison. These differences are summed to create a simple metric of block similarity. In this implementation, three similarity measures can be applied: SAD, zero-mean SAD or locally scaled SAD.
- Sum of Squared Differences, SSD (Shirai, 1987): measures similarity between image blocks by squaring the differences between each pixel in the original block and the corresponding pixel in the block being used for comparison. In this implementation, three similarity measures can be applied: SSD, zero-mean SSD or locally scaled SSD.
- Normalized Cross Correlation, NCC (Faugeras and Keriven, 1998): a common matching technique to tolerate radiometric differences between stereo images. In this implementation, two similarity measures can be applied: NCC or zero-mean NCC.
- 3D Stereo Disparity (Lankton, 2007): estimates pixel disparity by sliding an image over another one used as a reference, subtracting their intensity values and gradient information (spatial derivatives). It also performs a post filtering phase, segmenting the reference image using a "Mean Shift Segmentation" technique, creating clusters with the same disparity value, computed as the median disparity of all the pixels within that segment.
- Region Based Stereo Matching (Alagoz, 2008): provides two different algorithms based on region growing: a) Global Error Energy Minimization by Smoothing Functions method, using a block-matching technique to construct an Error Energy matrix for every disparity found and b) Line Growing method, locating starting points from which regions will grow and then expanding them according to a predefined rule.

- Stereo Matching (Ogale and Aloimonos, 2005; Ogale and Aloimonos, 2007): develops a compositional matching algorithm using binary local evidence which can match images containing slanted surfaces and images having different contrast. It finds connected components and for each pixel, picks the shift which corresponds to the largest connected component containing that pixel, while respecting the uniqueness constraint.
- Optical Flow (Ogale and Aloimonos, 2005; Ogale and Aloimonos, 2007): similar to the previous stereo matching algorithm, but the main difference is this considers the shifts are two dimensional, unlike the previous one that considered just horizontal shifts, and connections across edges parallel to the flow being considered must be severed.
- Stereo Matching (Abbeloos, 2010): implements a fast algorithm based on sum of absolute differences (SAD), matching two rectified and undistorted stereo images with subpixel accuracy.
- Enhanced Normalized Cross Correlation (Psarakis and Evangelidis, 2005): implements a local (window-based) algorithm that estimates the disparity using a linear kernel that is embodied into the normalized cross correlation function leading to a continuous function of subpixel correction parameter for each candidate right window. It presents two modes: a) a typical local mode where the disparity decision regards the central pixel only of each investigated left window and b) a shiftable-window mode where each pixel is matched based on the best window that participates into, no matter what its position inside the window is.
- Block-Matching (Konolige, 1997b): based on sum of absolute differences between pixels of the left and right images. This algorithm is optimized by applying the epipolar constraint, searching for matches only over the same horizontal lines which cross the two images, what necessarily implies the two images of the stereo pair must have previously been rectified. The OpenCV 2.3.0 (Bradski and Kaehler, 2008) implementation is used for this process.
- Semi-Global Block-Matching (Hirschmüller, 2005): a dense hybrid approach that divides the image into windows or blocks of pixels and looks for matches trying to minimize a global energy function, based on the content of the window, instead of

looking for similarities between individual pixels. It includes two parameters that control the penalty for disparity changes between neighboring pixels, regulating the "softness" in the changes of disparity. The OpenCV 2.3.0 (Bradski and Kaehler, 2008) implementation is used for this process.

- Variational Stereo Correspondence: implements a modification of the variational stereo correspondence algorithm described in (Kosov et al., 2009), consisting on a multi-grid approach combined with a multi-level adaptive technique that refines the grid only at peculiarities in the solution, allowing the variational algorithm to achieve real-time performance. It includes also a technique that adapts the regularizer, used in the variational approach, as a function of the current state of the optimization. The OpenCV 2.3.0 (Bradski and Kaehler, 2008) implementation is used for this process.
- Growing Correspondence Seeds (Cech and Sara, 2007): a fast matching algorithm for binocular stereo suitable for large images. It avoids visiting the entire disparity space by growing high similarity components. The final decision is performed by Confidently Stable Matching, which selects among competing correspondence hypotheses. The algorithm is not a direct combination of unconstrained growth followed by a filtration step. Instead, the growing procedure is designed to fit the final matching algorithm in the sense the growing is stopped whenever the correspondence hypothesis cannot win the final matching competition.

Post-filters:

- Clustering: to remove false matches after disparities computation. The algorithm groups pixels in connected components based on the principle of spatial continuity, where pixels belonging to the same region are all reachable from any other pixel from the same region, either by direct contact or at very short distance. Once clusters have been computed, they are filtered out depending on certain configurable criteria such as cluster's size or density.

Datasets:

- Middlebury: a subset of the well-known Middlebury datasets (Scharstein and Szeliski, 2002). They have been taken using structured light under controlled conditions and include ground truth data, very useful for assessing algorithms' correctness.
- Gazebo: a set of synthetic images of terrains simulated in Gazebo (Koenig and Howard, 2004).
- MER: images taken from the Spirit and Opportunity spacecrafts launched in 2004 as part of the MER Mission to Mars (Goldberg et al., 2002).
- Videre mini: images of terrains captured with a Videre STH-DCSG color stereo camera, using standard 3mm miniature lenses, separated 9 cm and 640x480 pixels resolution.
- Videre Var-C: images of terrains captured with a Videre STH-DCSG-VAR color stereo camera, using standard 8mm C-Mount locking lenses, separated 15 cm and 640x480 pixels resolution.
- Others: a set of terrain images obtained online, for further testing.

All these algorithms have been adapted to ensure compatibility to be included within the testbed; their inputs and outputs have been standardized to interact with each other and with the testbed itself. For extended details on the insights of each algorithm, understanding of their performance, implementation details and how they differ from the other to confirm which one could better suit each own needs and requirements, it is recommended to consult the available literature associated to each algorithm, as a thoughtful analysis of these aspects is out of the scope of this work and heavily depends on the concrete application and constraints of each project.

For an initial version of the testbed, the selection of the algorithms to be included into the testbed does not try to represent a sampling of state of the art algorithms. The main objective is to demonstrate the integration of heterogeneous algorithms developed by different authors using different programming languages. Some of them had to be implemented while others had an implementation already available; some are older, the classical approaches, although still very commonly applied, while others are quite recent; others, such as the Block-Matching (Konolige, 1997b), Semi-Global Block-Matching (Hirschmüller, 2005) and Variational Stereo Correspondence, are part of well-known and popular libraries in the computer vision community, in this case part of the OpenCV library (Bradski and Kaehler, 2008). This variety of algorithms has been actually chosen that way on purpose, in an attempt to balance approaches from different ages and diverse technologies, to be integrated together under the common testbed framework.

As shown in (Scharstein and Blasiak, 2014) and (Geiger et al., 2014), the immense number of dense stereo matching algorithms makes the idea of implementing all of them a titanic, and almost unachievable, effort; a definitely not attainable for an initial version. Besides, new approaches are continuously coming up and being published. For that reason the testbed has been conceived as an incremental development, open to collaborations to produce more sophisticated versions with more algorithms and features over time. The algorithms selection presented here constitutes just a first attempt to demonstrate how a set of fairly random and diverse algorithms are integrated together.

5.3.4 Experiments and Results

5.3.4.1 Use Case: Matching Algorithms Comparison

In this section, an experiment carried out to evaluate the performance and results of the set of matching algorithms included in the testbed is described. The same stereo pair has been used as input as a representative example to test every algorithm and obtain results from comparison; analogous results have been obtained in other experiments using different input images, around 20 pairs.

This reference stereo pair, shown in Figure 61, belongs to the Middlebury dataset (Scharstein and Szeliski, 2002). Although they are not images of terrain intended for navigation purposes in autonomous robots, they are very suitable for assessing the results and validate algorithms correctness, as they have been taken in an illumination controlled environment and include ground truth data. Input images, Figure 61, have a 437x370 resolution (161,690 pixels) and 151,707 possible matches, as some regions do not contain disparity information (dark blue areas in Figure 61).

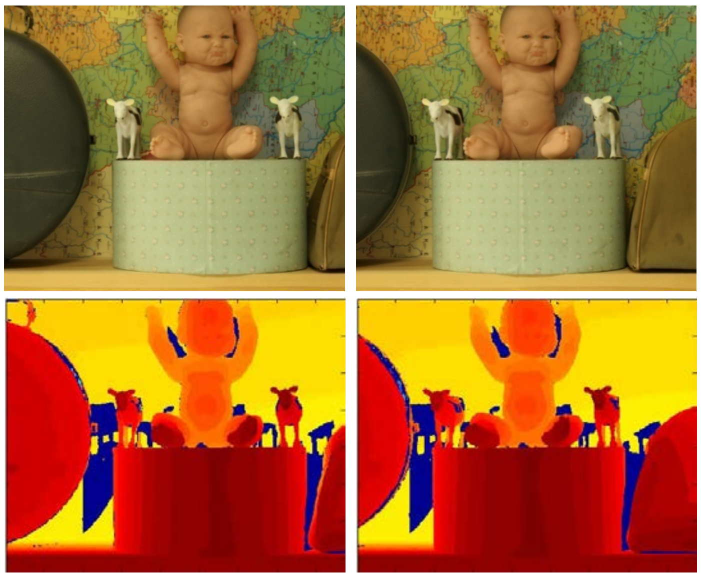
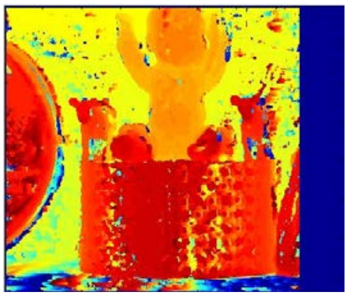
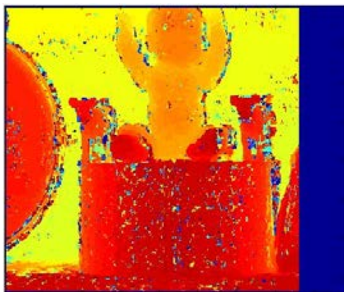
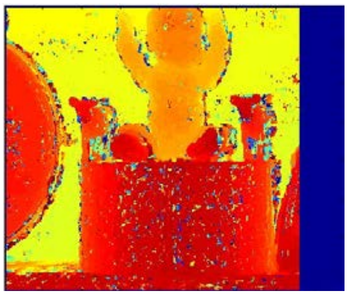


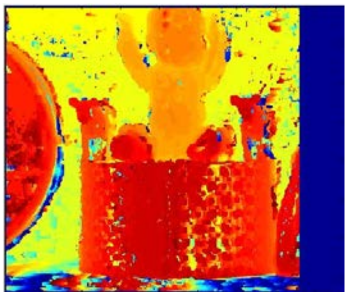
Figure 61. Input stereo images and ground truth data (Hirschmüller and Scharstein, 2007)

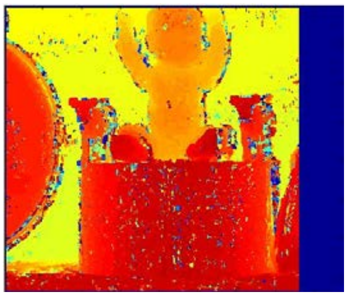
Using the stereo testbed, each algorithm has been executed several times adjusting its parameters, taking the execution that produced the best results. Figure 62 shows disparities maps obtained from the execution of these algorithms. NCC and ZNCC algorithms do not produce any results for these input images.

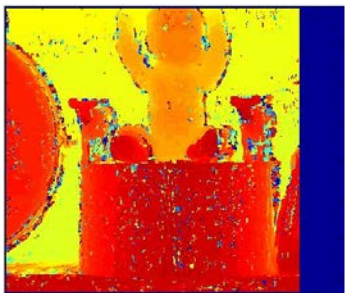

a) SAD


b) ZSAD


c) LSAD

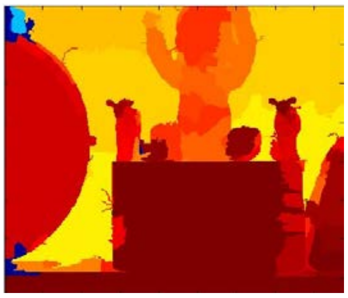

d) SSD

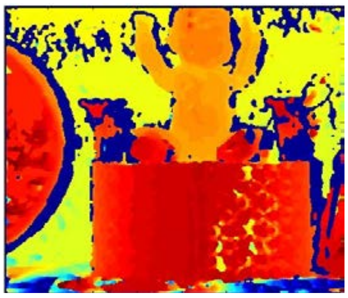

e) ZSSD

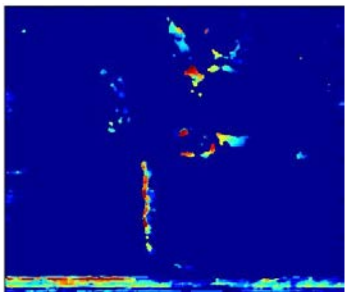

f) LSSD

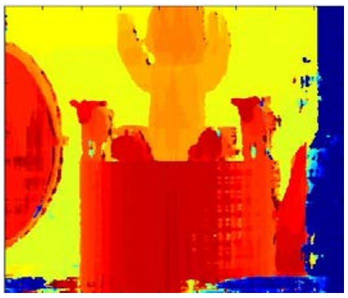
No results produced

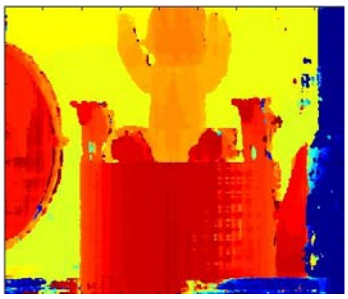
g) NCC / ZNCC

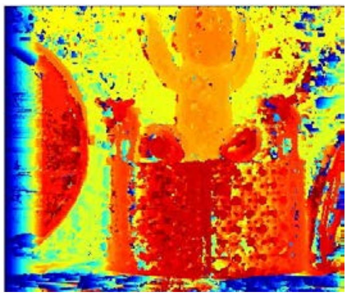

h) (Lankton, 2007)

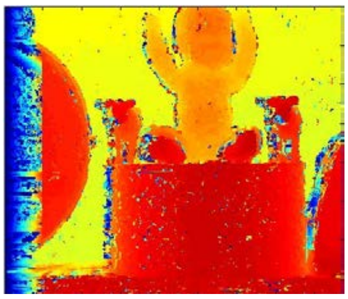

i) Global Error Energy Minimization (Alagoz, 2008)

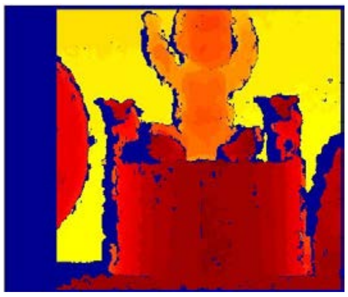

j) Line Growing (Alagoz, 2008)

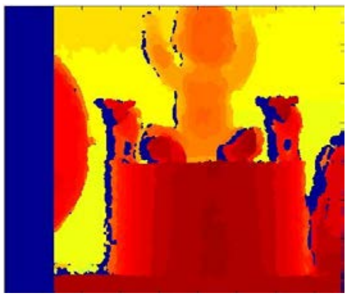

k) Stereo Matching (Ogale and Aloimonos, 2005; Ogale and Aloimonos, 2007)

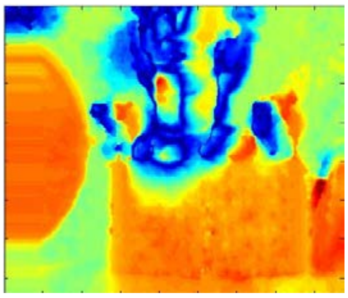

l) Optical Flow (Ogale and Aloimonos, 2005; Ogale and Aloimonos, 2007)


m) Stereo Matching (Abbeloos, 2010)


n) Enhanced Normalized Cross Correlation (Psarakis and Evangelidis, 2005)


o) BM (Konolige, 1997a)


p) SGBM (Hirschmüller, 2005)


q) Variational Stereo Correspondence (Kosov et al., 2009)

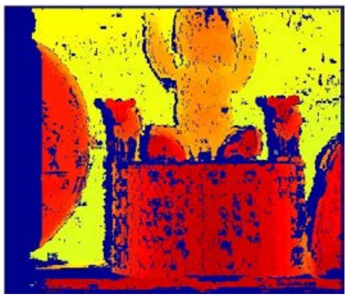

r) Growing CorrespondenceSeeds (Cech and Sara, 2007)

Figure 62. Disparities maps produced by the different stereo algorithms included within the testbed

Table 5 shows the numerical results, including the total number of matches found, broken down in right, wrong, false and missed correspondences, and execution time. *Right* matches (M_r) counts the set of correspondences computed by the algorithm that coincide with ground truth data or its value differs in just 1 disparity value, while *wrong* (M_w) indicates the number of computed correspondences that do not coincide with ground truth data or have a difference of more than 1; meaning the computed disparity value for that pixel differs in more than 1 from the one stored as ground truth; *false* (M_f) account correspondences computed by the algorithm that do not appear in the ground truth data and therefore should not have been detected as a match, while *missed* (M_m) are those pixels which in spite of having a correspondence in ground truth data the algorithm was not able to compute.

Table 5. Stereo algorithms performance on a PC Intel Core2 1.86 GHz.

Algorithm	Matches					Time (s)
	Found	Right	Wrong	False	Missed	
SAD	137,039	44,170 (29.12%)	85,320 (56.24%)	7,549	22,217	640
ZSAD	138,122	59,035 (38.91%)	71,544 (47.16%)	7,543	21,128	653
LSAD	138,096	58,623 (38.64%)	71,930 (47.41%)	7,543	21,154	657
SSD	137,065	46,891 (30.91%)	82,628 (54.47%)	7,546	22,188	662
ZSSD	138,121	59,998 (39.55%)	70,580 (46.52%)	7,543	21,129	692
LSSD	138,097	59,604 (39.29%)	70,946 (46.77%)	7,574	21,157	683
NCC	0	0 (0.00%)	0 (0.00%)	0	0	647
ZNCC	0	0 (0.00%)	0 (0.00%)	0	0	665
3D Stereo Disparity (Lankton, 2007)	161,690	**117,578 (77.5%)**	34,129 (22.5%)	9,983	0	185
Global Error Energy Minimization (Alagoz, 2008)	117,281	46,824 (30.86%)	64,758 (42.69%)	5,699	21,408	6,289
Line Growing (Alagoz, 2008)	11,065	137 (0.09%)	10,833 (7.14%)	95	122,020	2,223
Stereo Matching (Ogale and Aloimonos, 2005; Ogale and Aloimonos, 2007)	147,777	**111,387 (73.42%)**	26,291 (17.33%)	10,099	13,828	7
Optical Flow (Ogale and Aloimonos, 2005; Ogale and Aloimonos, 2007)	148,053	**110,508 (72.84%)**	27,454 (18.1%)	10,091	13,544	19
Stereo Matching (Abbeloos, 2010)	157,475	79,638 (52.49%)	67,928 (44.78%)	9,909	4,141	21
Enhanced Normalized Cross Correlation (Psarakis and Evangelidis, 2005)	161,320	**118,942 (78.40%)**	32,398 (21.36%)	9,980	367	22
Block-Matching (Konolige, 1997a)	116,184	**107,281 (70.72%)**	**3,174 (2.1%)**	5,729	41,252	2
SGBM (Hirschmüller, 2005)	130,677	**118,026 (77.8%)**	**5,486 (3.6%)**	7,165	28,195	2,5
Variational Stereo Correspondence (Kosov et al., 2009)	161,489	828 (0.55%)	150,688 (99.33%)	9,973	121	2,5
Growing Correspondence Seeds (Cech and Sara, 2007)	161,690	101,719 (67.05%)	49,988 (32.95%)	9,983	0	11,5

The total number of matches *found* (M_F) is equal to the sum of *right*, *wrong* and *false* matches, equation (81); the number of matches *found* minus *false* plus *missed* matches adds up the total number of potential matches (M), equation (82). The percentage shown besides *right* and *wrong* matches is computed with respect to the total number of potential correspondences the image has; in the example used in this section, Figure 61, it is 151,707 pixels. Best results have been highlighted on each column, meaning an algorithm obtaining over 70% of potential correspondences while producing less than 5% of error -wrong matches. Figure 63 shows these results graphically.

$$M_F = M_r + M_w + M_f \tag{81}$$

$$M = M_F - M_f + M_m \tag{82}$$

It is possible to observe the variety of results as well as how different algorithms perform with the same input images. Most of them find a large number of matches, although in some cases they make a large number of mistakes when results are compared with ground truth data; that is the case of the classical and simplest approaches –SAD, SSD and NCC. Regarding computation time, those take up to two orders of magnitude longer than newer algorithms. This is due to modern approaches that are usually more sophisticated and specifically designed to improve performance in terms of correctness and computing time. However, that does not mean some algorithms is incorrect or should be dismissed, as the results are very dependent of the input images; it is just a matter of finding the most appropriate approach to each concrete needs and application.

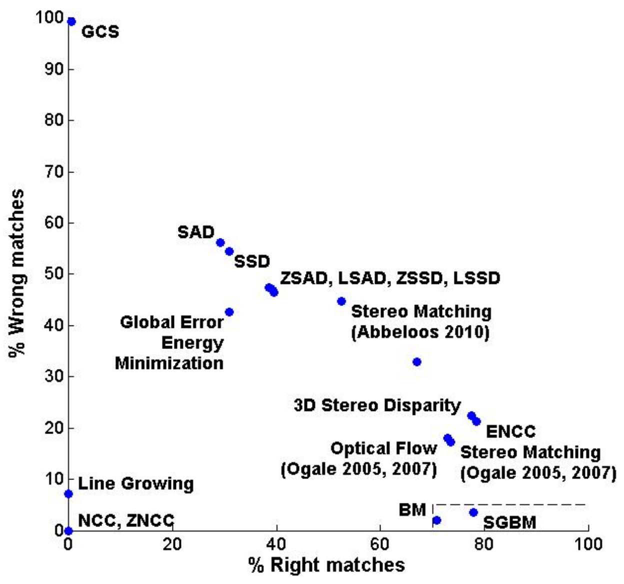


Figure 63. Comparison of matching algorithms' results in terms of right and wrong percentage of correspondences

There are some factors to be considered when analyzing these results. From Table 5, it can be observed in all cases that there are a number of false positives. The ground truth data contains 9,983 pixels, in the case of the left image, with no disparity information –dark blue areas in Figure 61. It is due in part to occlusions and to the method the authors have used to compute them, as many of the algorithms tested are actually able to find matches in those areas. When comparing results with ground truth data, any correspondence found in that area will be counted as false match. There is also another important aspect to be considered. As both images from the pair are taken at the same time from different points of view, there are certain areas of the scene seen from one camera but not from the other. That is the case of the leftmost and rightmost areas, depending on which image is taken as a reference, causing a band in the disparities map of a certain width, coinciding with the configured maximum value of disparity to be searched, observed in many results in Figure 62. Pixels from these areas cannot be matched as they belong to portions of the scene that are not present in both images. However, ground truth data provided in this dataset contains disparity information for the whole image, including those areas, and they are considered as missed correspondences in the results shown in Table 5.

For this experiment and this concrete stereo pair, the Semi-Global Block Matching (SGBM) algorithm (Hirschmüller, 2005) obtains the best results in terms of number of right, wrong, false and missed matches among the set of algorithms tested. It is able to find a large set of matches, most of them correct, see Figure 64. Resulting false matches are due to the lack of information in some areas of the ground truth data as introduced before, as this algorithm is capable of computing correspondences in those areas. Most of the missed matches obtained are due to the portion of the scene present in one image but not in the other –vertical left or right bands, see Figure 62; this is an unavoidable fact, but the ground truth data contains information for this area also (see Figure 61) causing those missed correspondences.

This SGBM algorithm presents a very good performance in terms of computation time in comparison to other approaches. It must be taken into account when executing code from Matlab, computing time is increased as compared to other implementations. Once a given algorithm has been selected for a concrete application and characteristics of the operational settings and input images, it can be implemented in a much more efficient way using either native languages such as C++ or assembler, or even codified and parallelized using FPGAs (Darabiha et al., 2006; Lim et al., 2012).

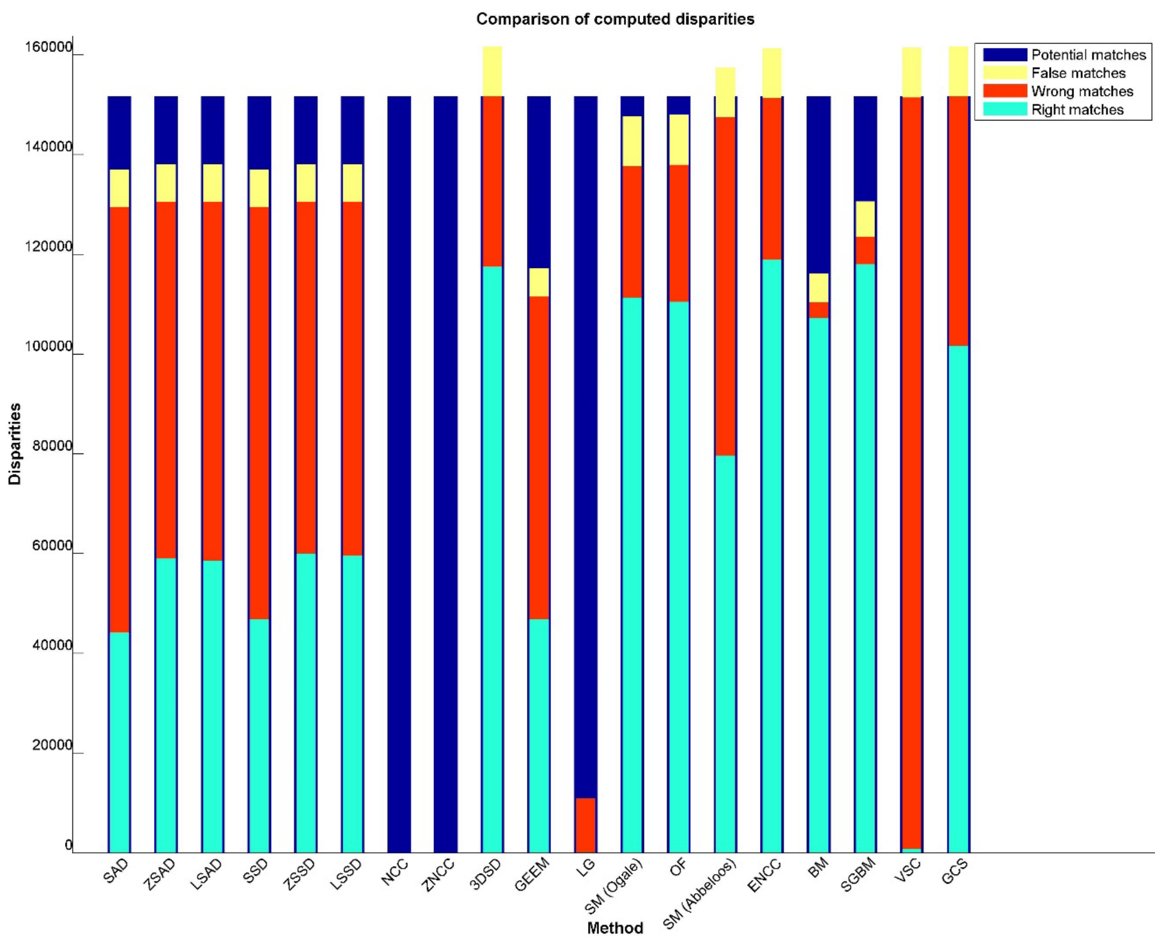


Figure 64. Comparison of matching algorithms' results in terms of right, wrong and false matches

5.3.4.2 Use Case: from the Simulated to the Real World

This section details a use case to illustrate the usage of the testbed and its utility and application to a real problem. As introduced before at the beginning of this chapter, at the initial phases of the development of the autonomous navigation software there wasn't any vehicle available and the operational environment and vehicle were simulated within the framework. Images of simulated terrains were captured from simulated cameras. A problem was detected when the algorithm used for processing the synthetic images was used with real images; it did not produce satisfactory results and new algorithms had to be analyzed.

For 3D reconstruction, and in order to create a reliable map of the terrain, it is desirable to achieve the greatest possible number of pixel correspondences between the input stereo images. It happens that each algorithm performs well with a series of images, usually taken in controlled environments in regards to the lighting and texture. However, they do not usually produce good results when conditions change. Such is the case of some algorithms evaluated with the real images captured from the robot's camera.

The first step that has been considered is the assessment of the behavior of some representative algorithms with real images captured. A complete taxonomy of stereo correspondence algorithms can be found in (Scharstein and Szeliski, 2002), where a thorough analysis and a comparative evaluation of a large number of algorithms is made. More recently, (Tombari et al., 2011) also examines the state of the art algorithms and evaluation for its applications to 3D objects recognition. The objective is to determine which one has a better performance for subsequent adaptation to the real case

In this scenario, the testbed has resulted of invaluable help. It has been used to test a set of real images with different matching algorithms, analyzing its results and performance. Figure 65 shows the disparity maps obtained when applying some of these stereo algorithms to the pair of input images. The ones shown in the figure belongs to the Lankton method (Lankton, 2007), based on the growth of regions; it is divided into two phases; in the first one, a starting point is located so that the region will grow from; in a second stage, the region related to each starting point expands according to a predefined rule. This method also implements a 2D filter based on the statistical mode to replace pixels with low confidence level by information obtained from neighboring pixels with higher reliability. (Alagoz, 2008) applies a region-based method that implements a filter to eliminate unreliable disparities; it in turn has two variants: Global Error Energy Minimization and Line Growing. It is also shown the results obtained when applying the methods developed by Ogale and Aloimonos, (2005); Ogale and Aloimonos (2007); Abbeloos (2010), all based on the sum of absolute differences (SAD).

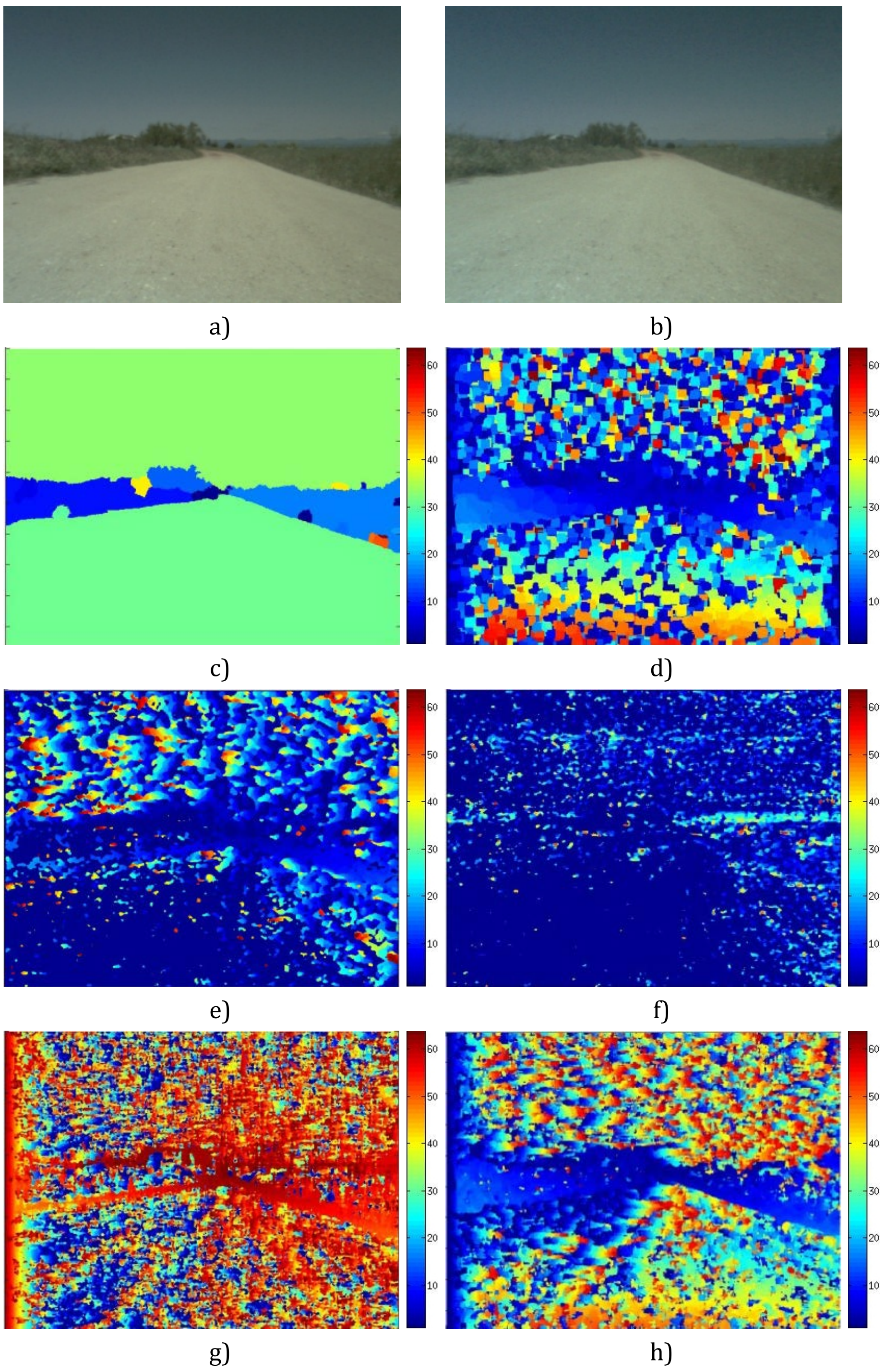


Figure 65. Disparity map obtained for the input images using: c) Lankton, d) Lankton mode filter, e) Alagoz (Global Error Energy Minimization), f) Alagoz (Growing Line), g) Ogale, h) Abbeloos

All these approaches have been tested with real images, without satisfactory results, as shown in Figure 65. None of these disparity maps are correct or usable for a 3D scene reconstruction. The Lankton (2007) method computes a series of regions where every pixel of the region is adjusted to the same disparity value, which is far from what a disparity map of a terrain would be, which should be similar to the ones shown in Figure 66 (c) of (d). The other methods, (Alagoz, 2008, Ogale and Aloimonos, 2005; Ogale and Aloimonos, 2007; Abbeloos, 2010), compute a too noisy map, not usable either for a 3D reconstruction of the terrain. This behavior is extensible to the complete set of analyzed images captured with the real stereoscopic system.

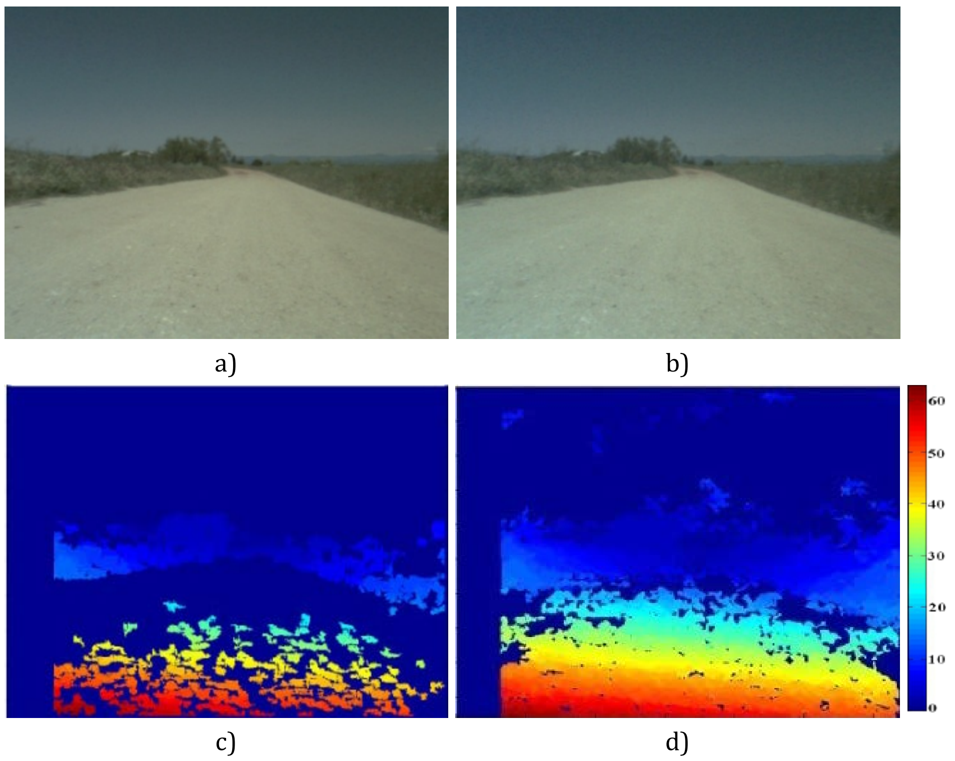


Figure 66. (a), (b) Images taken from the onboard Videre stereocamera, (c) stereo process results using the block-matching algorithm and (d) using the semi-global block matching algorithm

From the set of algorithms tested and integrated within the testbed, it was verified the best results for the set of captured images were obtained using the Semi-Global Block Matching algorithm (Hirschmüller, 2005), implemented in the OpenCV library. Other authors have evaluated matching algorithms (Scharstein and Szeliski, 2002; Tombari et al., 2011) and agree that SGBM has an excellent quality and runtime relationship. SGBM divides the image into windows or blocks of pixels and searches for correspondences trying to minimize an energy function globally, instead of looking for similarities between individual pixels. It includes two parameters that control the penalty for disparity changes between neighboring pixels, while allowing regulate the "softness" in those disparity changes.

This algorithm is appropriate for images where texture is more or less uniform and have low variation in the spectral levels of its pixels. The energy reduction function detects correspondences in regions where the Block-Matching algorithm fails, as it is a local method based on SAD which does not generate significant differences within neighboring pixels.

The disparities map obtained using the SGBM algorithm with respect to BM is shown in Figure 66 (d). It finds 144,827 matches, achieving a great improvement with respect to the 57,598 correspondences obtained from the BM algorithm, see Figure 67. The behavior of the algorithms described thus far is similar for the set of real images analyzed. As a result, the BM algorithm was replaced by the SGBM algorithm in the robot onboard software.

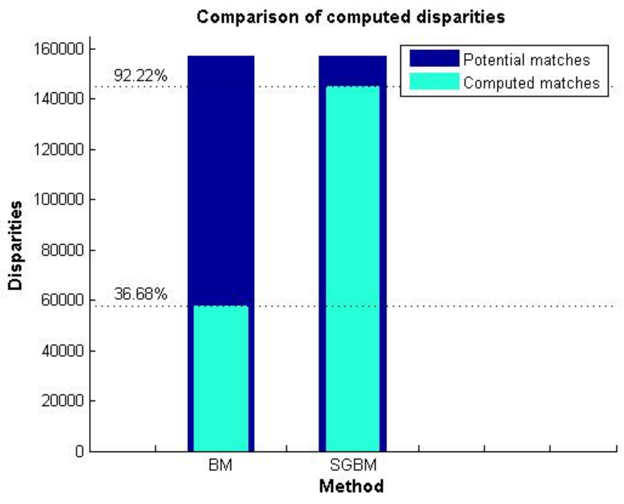


Figure 67. Comparison of disparities computed by the BM and SGBM algorithms

Chapter 6

AUTOMATIC EXPERT SYSTEM FOR 3D SCENE RECONSTRUCTION BASED ON ENHANCED STEREO VISION BY THE APPLICATION OF NOVEL IMAGE FILTERS

This chapter addresses the problem of computation of correspondences in pairs of images obtained with real stereoscopic systems. It has been observed that the problem concerning the low efficiency of many matching algorithms when they are applied to real images are due to differences in intensities in both images of the stereoscopic pair. It requires a global solution to this general problem. This chapter addresses this problem by proposing an automatic expert system for 3D scene reconstruction based on stereovision with automatic intensity correction through the application of some of the filters included in the testbed introduced in the previous chapter. The proposed expert system exploits the human knowledge which is mapped into three modules based on image processing techniques. The first one is intended for correcting intensities of the stereo pair coordinately. The second one is based in computing disparity, obtaining a set of correspondences. The last one computes a reconstruction of the terrain by reprojecting the computed points to 2D and applying a series of geometrical transformations. The performance of this method is verified favorably. The work described in this chapter is based on Correal et al. (2013) and Correal et al. (2014a).

6.1 Motivational research of the proposed strategy

As already mentioned, the perception of the environment and the information extracted from the pair of images is a crucial factor for the autonomy and navigation capabilities of a robot; as the result of the stereo process a set of 3D points that represent the part of the visible environment are obtained. This 3D-points structure represents distances to objects in the scene, by which the robot can calculate safe routes through the environment and navigate autonomously to a particular target. Therefore, the mapping, path planning and navigation processes are heavily dependent on the result of the perception process and more specifically of the matching process. Within this process, false positives, obtained when computing correspondences of pixels between images, will cause errors in the produced map; missed matches, will cause voids in the map; both factors will have a major impact in the environment reconstruction and subsequent processes. To perform a high-quality environment reconstruction is desirable to avoid gaps or areas without information, as well as avoid mistakes in the disparities calculation.

According to results obtained in the previous chapter, the use of the SGBM algorithm implies a significant improvement in the level of efficiency of the process regarding the use of the BM algorithm. It has been shown that the SGBM algorithm was able to find 144,827 matches with the sample images presented in Figure 66. The obtained disparities map using this algorithm with respect to using BM is notable, which computed just 57,598 correspondences. However, it is observed the SGBM algorithm introduces some error. Concretely, following with the previous example, 10,738 pixels were erroneously matched; these are called false positives. The fact of introducing such errors has important consequences for the 3D reconstruction, where such correspondences can be considered as –nonexistent- objects in the environment, which can represent obstacles to the robot and areas that cannot be navigate through. Therefore, aside from those errors, only 134,089 of the computed correspondences are correct, representing 85.38% of the total correspondences contained in that pair of images.

Different methods and strategies for 3D environment reconstruction using stereo vision have been applied in different works (Goldberg et al., 2002; Bakambu et al., 2006; Morisset et al., 2009; Lin and Zhou, 2009; Xing-zhe et al., 2010; Song et al., 2012). Most matching algorithms are based on the minimization of differences, using correlation between brightness (intensity) patterns in the local neighborhood of a pixel in one image with respect the other (Baker, 1982).

A number of matching algorithms, as part of the overall stereo process aimed to perform a 3D reconstruction for robotic autonomous navigation in natural and unstructured environments, have been analyzed and tested, including the ones integrated into the testbed introduced in the previous chapter. Most of them give unsatisfactory results, as seen from the results presented in section 5.3.4. This fact allows us to deduce that the problem concerning the low efficiency of some of the above algorithms comes from their application to real images. It has been observed that the problem concerning the low efficiency of some of these algorithms comes from their application to real images, due to differences in intensities in both images of the stereoscopic pair. Because of the above undesired effects and differences in illumination, the stereo matching process is significantly affected, as many correspondence algorithms are very sensitive to these deviations in the brightness pattern, leading to a poor and inaccurate computation of disparities and resulting in an inaccurate or incorrect terrain reconstruction.

While most existing strategies focus in the problem of computation of disparities and the matching process, there is little work devoted to the correction and validation of the input images, beyond vertical alignment and rectification (Papadimitriou and Dennis, 2006; Kang and Ho, 2012). Therefore, as mentioned before, it is necessary a global solution to this general problem, a method capable of correcting the differences in both images of the pair.

6.2 Introduction

The problem of correspondence in stereoscopic systems stems from the fact that images from cameras, although similar, show different intensity levels for the same physical entity in the 3D scene. The main reason for this feature lies in the different response from the camera sensors to the signal light from the scene and also from the different mapping of the scene over each image due to the different relative points of view of each camera. That makes necessary to devote a major research effort to correct these deviations typical of any stereo system. This problem is not yet satisfactorily solved, particularly in unstructured and uncontrolled environments. That is the main reason why literature about this topic is so broad.

The constant image brightness (CIB) approach assumes that the intensities of corresponding points in two images of a stereo pair are equal. This assumption is central to much of computer vision works. However, surprisingly little work has been performed to support this assumption, despite the fact the many of the algorithms are very sensitive to deviations from CIB. In Cox and Hingorani (1995) a study revealed that after an examination of 49 images pairs contained in the SRI JISCT stereo database (Bolles et al., 1993), a dataset that includes images provided by research groups at JPL, INRIA, SRI, CMU, and Teleos, the constant image brightness assumption is indeed often false. The same issue has been also found in the set of stereo images taken from out robot's camera, the Videre STH- DCSG 9 mm.

This justifies the need for a pretreatment of the images. The aim is to achieve the maximum similarity in the spectral levels of the stereo pair images, at the pixel level, which is exactly what happens in the case of the simulated images. Therefore, it is necessary a method capable of correcting, from the radiometric point of view, the differences in both images, confirming the need for the research work presented in this chapter.

There exist several strategies to correct intensity values (brightness) in images. Next, some of the most commonly used techniques are introduced:

(1) Histogram equalization is a method used in image processing for contrast adjustment using the image's histogram (Pajares and Cruz, 2007; Laia et al., 2012). This method usually increases the global contrast, especially when the usable data of the image is represented by close contrast values. Through this adjustment, the intensities can be better distributed on the histogram. This allows for areas of lower local contrast to gain a higher contrast.

(2) In Kawai and Tomita (1998) authors propose a method to calibrate intensity for images based on segment correspondence. First, the edges are detected in each image. Each section is defined as a segment by dividing the edges using some characteristic points such as turning, wiggle, inflection, transition, etc. This data is then converted into boundary representation. The intensity information consists of the intensity value and a derivative at the point, which is the point where the derivative is the smallest point in the neighborhood in the direction of the normal in the region. The segment divides regions with different

intensities. The correspondence of the segments between I_1, the reference image, and an image to be corrected I_2 is obtained using a segment-based stereo method, finding similar boundaries in I_1 and I_2. Next, the intensity correspondences between images is found, as the correspondence between the points is obtained from the segment correspondences. The intensity calibration equation between images (I_1, I_2) is derived from the distribution. If the correspondence is correct, the points are distributed on a straight line, and a straight line is fitted using the next equation: $I_2(0) = a\, I_1 + b$, where a and b are coefficients and $I_i(n)$ is the image after the *n*th iteration. The image $I_2(0)$ is calibrated based on this equation, and a refined image $I_2(1)$ is calculated. The process is repeated using subsequently refined image, until $I_2(n) \approx I_1$.

(3) Homomorphic filtering (Pajares and Cruz, 2007; Gonzalez and Woods, 2008). It is a filtering process in the frequency domain to compress the brightness based on the lighting conditions, while enhancing the contrast from the reflectance properties of the objects. It is based on the fact that each image is formed by the concurrence of two-component image: reflectance and illumination. The illumination component comes from the light conditions in the scene when the image is captured and may change as the light conditions also change. The reflectance component depends on how the objects in the scene reflect light, which is determined by the intrinsic properties of the objects themselves, which (usually) do not change. In many practical applications it is useful to enhance the reflectance component, while the illumination is reduced.

(4) In Cruz et al. (1995a); Cruz et al. (1995b) Pajares et al. (1998a) authors have applied different learning and optimization-based strategies, including Bayesian (Pajares and Cruz, 2002b), neural networks (Pajares and Cruz, 2001), optimization (Pajares et al. 1998c), perceptron, self-organizing feature maps (Pajares and Cruz, 2002a), hebbian learning (Pajares and Cruz, 1999), support vector machines (Pajares and Cruz, 2003), simulated annealing (Pajares and Cruz, 2004), fuzzy clustering (Pajares and Cruz, 2000b) and fuzzy cognitive maps (Pajares and Cruz, 2006) to learn these claimed differences between the two images in the stereoscopic pair. The main idea was to estimate appropriate parameters that allow intensities correction.

The methods described in points (1) and (2) are intended to correct and adjust brightness in images. However, the main drawback of these approaches is that they were originally designed for single images; and even in stereo vision scenarios, they are typically applied separately, processing each image of the pair independently of one another. However, as introduced before, most matching algorithms are based in correlation between brightness patterns in the local neighborhood of each pixel to find its correspondence in the other image. The method described in (3), although used in some computer vision applications (Ponomarev, 1995; Hailan, 2012), it has never been actually applied to stereo pair of images to be used subsequently as input for stereo matching algorithms. The methods described in (4) are based on learning and optimization strategies, where a set of sample patterns are always required. This represents an important drawback because it is usually hard to verify if all the scenarios have been incorporated during the training phase to integrate all relevant information. This could be especially dramatic when unknown scenarios appear for the first time and the system has not been trained, a common situation in robot navigation.

As introduced before, when working in stereo vision applications, a method to correct these deviations and problems derived from the illumination conditions adjusting the pair of images coordinately is needed, or one as a function of the other, to compensate for those differences typical of all stereoscopic system. Thus, the idea presented in this chapter is to apply an automatic image correction strategy for 3D terrain reconstruction, similar to that a human expert would apply to a similar problematic situation. This knowledge is mapped into the presented design following the logical strategy the expert human applies. In this scheme the problem concerning with different illumination patterns receives special attention from the point of view of human reasoning.

To do that, an expert system has been designed and implemented so that it makes use of some of the previously introduced techniques; concretely two of them stand out: a) homomorphic filtering and b) histogram matching, based on the previously introduced (1) histogram equalization approach. In the case of the homomorphic filtering, the process is applied to eliminate the illumination component of the input images while preserving the reflectance. In the case of the histogram matching approach, the histograms of both images of the stereo pair are obtained and compared; they are then automatically matched to adjust their intensity levels, both channel by channel in RGB images or converting them to grayscale images and matching then their histograms, previous to the stereo matching process.

This reasoning or knowledge, based on several stages, is the kernel of the proposed expert system. The strategy followed is the application of some pretreatment to the images so that the maximum similarity in the spectral levels of the images can be achieved. Although the main stage is the one concerning the image processing phase, the other stages are conveniently linked to form the body of the proposed reasoning. Each stage is designed for a given purpose and specific processes are applied for achieving the goal at each stage.

First, a study of the effects of applying both the homomorphic filtering and the histogram matching processes on the input images as a pre-matching step is performed. By means of this filtering, a significant improvement of the disparity map is achieved, getting a higher number of true correspondences in contrast to the results obtained by the correspondence processes without these filtering. Then, an additional and novel filtering process directed by clusters and based on the principle of spatial continuity is performed on the resulting disparity map to eliminate false positives and erroneous correspondences. These filtering processes, along with the design and development of the automatic expert system, constitute the main contributions of this chapter.

6.3 Histogram Matching

6.3.1 Introduction

Histogram equalization (not histogram matching) is a method used in image processing for contrast adjustment using the image's histogram (Pajares and Cruz, 2007; Laia et al., 2012). This method usually increases the global contrast, especially when the usable data of the image is represented by close contrast values. Through this adjustment, the intensities can be better distributed on the histogram. This allows for areas of lower local contrast to gain a higher contrast. Histogram equalization accomplishes this by effectively spreading out the most frequent intensity values. Consider a discrete grayscale image $\{x\}$ and let n_i be the number of occurrences of gray level *i*. The probability of an occurrence of a pixel of level *i* in the image is:

$$p_x(i) = p(x=i) = n_i/n,\ 0 \le i < L \tag{83}$$

L being the total number of gray levels in the image, *n* being the total number of pixels in the image, and $p_x(i)$ being in fact the image's histogram for pixel value *i*, normalized to [0, 1]. The Cumulative Distribution Function (CDF) corresponding to p_x is:

$$\mathrm{cd}f_x(i) = \Sigma\, p_x(j) \tag{84}$$

which is also the image's accumulated normalized histogram. Then, a transformation of the form $y = T(x)$ produces a new image $\{y\}$, such that its CDF will be linearized across the value range, i.e. $\mathrm{cd}f_y(i) = iK$ for some constant K. The properties of the CDF allows to perform such a transform; it is defined as:

$$y = T(x) = \mathrm{cd}f_x(x) \tag{85}$$

The *T* function maps the levels into the range [0, 1]. In order to map the values back into their original range, the following transformation needs to be applied on the result:

$$y' = y\,(\max\{x\} - \min\{x\}) + \min\{x\} \tag{86}$$

This method can also be used on color images by applying the same method separately to the Red, Green and Blue components of the RGB color values of the image. Different histogram equalization methods have been applied in stereo vision (Cox and Hingorani, 1995; Nalpantidis and Gasteratos, 2010; Zhang et al., 2010; Liling et al., 2012).

6.3.2 Description of the Method

Comparing the intensity histograms of the stereo images it has been found that corresponding pairs of histograms could vary significantly. As introduced before, there are several techniques for histogram equalization, typically applied to single images (Acharya and Ray, 2005; Pajares and Cruz, 2007). Some have been applied to stereo vision (Cox and Hingorani, 1995; Nalpantidis and Gasteratos, 2012; Zhang et al., 2010; Liling et al., 2012). In this work, an automatic histogram matching procedure has been implemented. This technique enhances the contrast of images using cumulative distribution functions to transform the values in an intensity image, so that the histogram of the output image approximately matches a reference; in this case it uses the histogram of one of the images in the stereo pair to adjust the other, approximating both illumination components.

The algorithm chooses the grayscale transformation T of the reference histogram to minimize:

$$\min |c_1(T(k))\text{-}c_0(k)| \tag{87}$$

where c_0 is the cumulative histogram of the image to be adjusted and c_1 is the cumulative sum of the reference histogram for all intensities k. This minimization is subject to the constraints that T must be monotonic and $c_1(T(a))$ cannot overshoot $c_0(a)$ by more than half the distance between the histogram counts at a. The procedure uses the transformation $b = T(a)$ to map the gray levels in X to their new values.

In the case of having a stereo pair of images, A and B, this process is performed from the histograms h_A and h_B, respectively, from which are obtained cumulative probability values for each gray level k_a and k_b to the respective images A and B as follows,

$$P(k_a) = \sum_{i=0}^{k_a} p(k_i); \quad P(k_b) = \sum_{i=0}^{k_b} p(k_i) \tag{88}$$

The matching procedure is to find for each value P(k_b) associated with the intensity level k_b, which is the closest P(k_a) so that it allows to exchange the value k_b by k_a in the image. After this exchange of intensity levels a new transformed image B' is obtained.

6.4 Homomorphic Filtering

The homomorphic filtering technique is based on the fact that each image is formed by the concurrence of two image components (Pajares and Cruz, 2007; Gonzalez and Woods, 2008): reflectance (r) and illumination (i). The first comes from the nature of the objects in the scene and how they reflect the light according to the intrinsic properties of the materials of which they are composed. The second is the result of the surrounding light in the scene. The CCD-based cameras used in our experiments are mainly impressed by the incidence of light. The homomorphic filtering process eliminates the illumination component of the input images while preserving reflectance, which is supposed most similar between the two stereoscopic images. The idea is that the same object must produce similar intensity levels on both the right and left images, achieving maximum similarity in the spectral levels of the stereoscopic pair, as it happens in the case of simulated images.

The homomorphic filtering technique has not been used so far for stereoscopic applications. It can be applied on the two images independently as described below. On each of the images in the RGB color model the corresponding transformation is applied to the HSI model. Two intensity images in this model are obtained related to the respective right and left images of the stereoscopic pair. An image is formed by the algebraic product of reflectance and illumination components:

$$I(\mathrm{x}, \mathrm{y}) = r(\mathrm{x}, \mathrm{y})\, i(\mathrm{x}, \mathrm{y}) \tag{89}$$

The natural logarithm is applied to I(x, y), obtaining:

$$\ln I(\mathrm{x}, \mathrm{y}) = \ln r(\mathrm{x}, \mathrm{y}) + \ln i(\mathrm{x}, \mathrm{y}) \tag{90}$$

At this point it is possible to apply the Fourier transformation of the sum of the results obtained by the logarithm, achieving the transform of the sum:

$$Z(u, v) = F_i(u, v) + F_r(u, v) \tag{91}$$

Since the materials of the objects in the scene are different, the reflectance component has high variations in the boundaries between objects. Therefore, the reflectance component is associated with the high frequencies present in the image while the intensity is associated with the low ones. Once the components are separated by applying the Fourier transformation, if a high-pass filtering is applied, the low frequencies (illumination) are removed, while preserving the high frequencies (reflectance). After filtering, the inverse Fourier transformation is applied and then the exponential function as inverse to the initial logarithmic transformation. The procedure can be seen as a succession of processes such as: $f(\text{x, y})$ -> ln -> DFT -> $H(u, v)$ -> DFT^{-1} -> exp -> $g(\text{x, y})$. Thus, a filtered image is achieved, retaining the reflectance component without the illumination component.

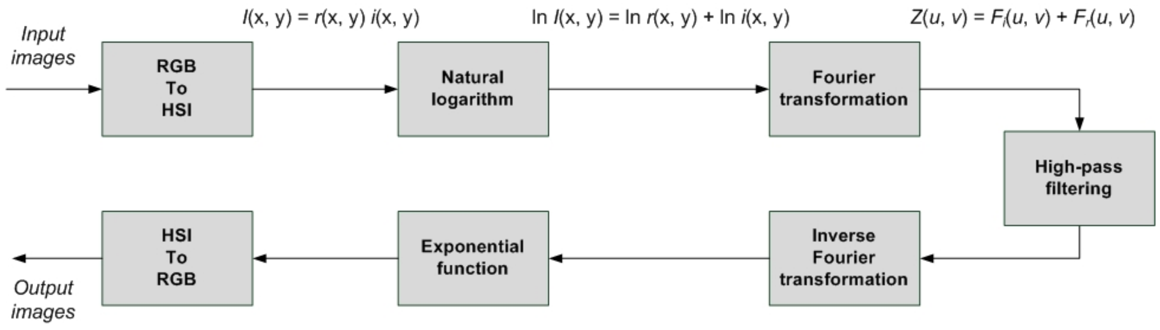


Figure 68. Homomorphic filtering process

6.5 Clustering Filter

The objective of the filtering process after the computation of disparities is to eliminate correspondence errors that may have occurred, taking out false positives. According to the experiments carried out, it happens that in most cases these do not appear scattered throughout the image, but are located in groups of pixels with very high disparity, close to the maximum. This same behavior has been appreciated for all terrain images captured with the stereo system used in these experiments.

Given the problem domain, robot navigation in unstructured natural terrain and, when designing the filter described in this section it has been taken into account the following premise: the input images are taken from natural terrains, composed of just one single continuous and connected component without discontinuities; in the environment there is not any discontinuity, as may be the case of some laboratory or indoors scenes which may contain separate objects and what is sought it is the identification of certain elements. In the natural terrain images case, a dense map is built from the calculated disparities, where the ground is taken as a whole, despite there might exist some gaps, being this a single and unique connected component.

The method described in this section is based on the philosophy of clustering. The result of the process is a list of clusters. It consists in grouping pixels in connected components based on the principle of spatial continuity. The pixels belonging to the same spatially connected region have the property that they are all reachable from any pixel of the region without leaving it; therefore, they must be in contact with each other, either directly or separated by a very short distance. Despite the presence of voids, gaps or pixels for which a value of correspondence has not been obtained, it does not cause the segmentation of the image in more than one component. Applying this filter, a list of clusters is obtained, where each one may represent one of these elements: 1) a large cluster composed by the majority of the computed correspondences, identifying the terrain itself; 2) isolated clusters that represent correspondence errors; or 3) isolated clusters that correspond to unconnected terrain areas, see Figure 78.

This is true for all the experiments performed with a large set of images captured with a real stereoscopic system. In all experiments, virtually every cluster except the principal one are formed by false positives and thus can be eliminated. Only in some very specific cases, some clusters represent small portions of terrain isolated from the rest. In such cases, the loss of pixels because the application of this filter is very small and not significant, less than 0.1 % of the pixels of the image, which is an acceptable magnitude; disregarding these pixels has almost no impact on the 3D reconstruction process. Anyway, these clusters can be often found very close to the principal cluster, and just by increasing slightly the configurable connection distance parameter, the algorithm will consider them part of the principal cluster and will not be filtered out.

Further details and an example of the application and performance of this filter can be found later in the experiments and results section of this chapter.

6.6 Expert System Design

6.6.1 Reasoning for Knowledge Extraction

As mentioned before, based on a logical human reasoning, the proposed expert system is designed according to the modular architecture displayed in Figure 69. It contains three stages, which are sequentially linked to form the expert system as a whole. Each stage contains the required automatic image and data processing modules.

(1) Image Processing: performs the automatic processes necessary to adjust intensities in the stereo pair of images coordinately.

(2) Stereo Matching: is the process of identifying features in both images, searching for correspondences, including any subsequent process to filter out any potential mismatch or error.

(3) 3D Scene Reconstruction: once the list of correspondences has been computed, data is reprojected from 3D to a 2D plane, in order to build the 3D scene reconstruction. That data is fused with information coming from other sensors, like an IMU to obtain the robot pose, and a series of mathematical 3D transformations are executed to transform data from the camera reference system to the robot and world reference systems.

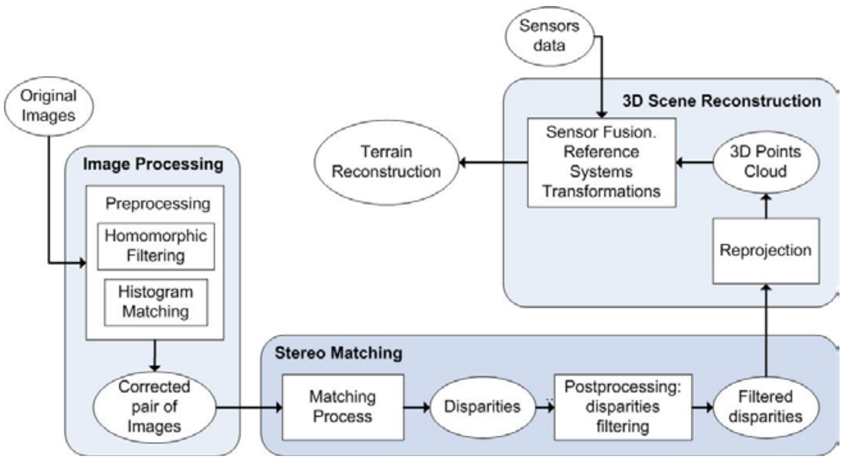


Figure 69. Expert System for 3D Scene Reconstruction Architecture

6.6.2 Automatic Image Processing Modules

Following the three previous stages, at each stage a sequence of image processing techniques are applied for automatic purposes, they are outlined in the graphic displayed in Figure 69, being grouped and linked conveniently. In this work, emphasis has been put in the first step mainly, where the images are automatically corrected, one as a function of the other, as a human expert would perform.

(1) *Image Processing*: Performs the automatic processes necessary to adjust intensities in the stereo pair of images coordinately. This process is guided by the intuitive human criterion that two images of the same scene will be alike according to the similarity of their corresponding spectral values.

Currently, two processes have been included within this module: histogram matching and homomorphic filtering. By now, there is not any automatic method for selecting what is the most suitable strategy to be applied, but the operator of the system configures it to indicate which filter to use. It is envisioned adding the capability of automatically select which filter to be used as a function of the input images, cited as future works later in this thesis.

As can be seen in Figure 69, the expert system has been designed to be very modular, so that it will be very maintainable an easy to extend. In that sense, any of these filters proposed in this work could be replaced by a new version or even a new algorithm to test its effect on the overall system. In the near future, when the automatic strategy selection capabilities will be included, it would not be necessary to replace one algorithm by another, but just adding new filters and update the selection criteria so that the system be able to choose among a broad range of algorithms and image filters processes to be applied as a function of the application and the input images.

Further details on the application of the image filters included on this expert system as well as some results on their performance can be found in the experiments and results section of this chapter.

(2) *Stereo Matching*: Once intensities of the input images have been adjusted, the common way to determine depth, with two stereo cameras, is by calculating disparity, see section 5 *Stereoscopic* Vision for further details.

It may be interesting to recall at this point the two main classes of disparity computation processes: feature-based and area-based methods (Ozanian, 1995). Briefly, feature-based methods use sets of pixels with similar attributes, either pixels belonging to edges (Grimson, 1985) or the corresponding edges themselves (Ayache and Faverjon, 1987); leading to a sparse depth map used mainly for object segmentation and recognition. Area-based stereo techniques use pixel by pixel correlation between brightness (intensity) patterns in the local neighborhood of a pixel in one image with respect the other (Baker, 1982); that is the reason why it is so important to coordinately adjust the intensities in both images of the pair are previously to the matching process, done in step (1).

As the application of the present work is scene reconstruction for path planning and robotic navigation, the objective is to obtain the highest possible number of disparities. Therefore, the matching process employs an area-based method, where the number of possible matches is intrinsically high; concretely, the SGBM algorithm (Hirschmüller, 2005) has been employed, a dense hybrid approach that divides the image into windows or blocks of pixels and looks for matches trying to minimize a global energy function, based on the content of the window, instead of looking for similarities between individual pixels. Again this represents the mapping of the expert knowledge for the proposed expert system. From the point of view of the computational approach, it includes two parameters that control the penalty for disparity changes between neighboring pixels, regulating the "softness" in the changes of disparity. Further details on the application of this algorithm to terrain images as well as post-processing methods to filter out mismatches correspondences can be found in Correal et al. (2013).

(3) *3D Scene Reconstruction*: After disparities have been computed in the previous stage, and following the previously commented human intuitive perception of depth, according to the position of each object in the scene perceived by each eye a human estimates the size and location of an object in the world with respect to him/her point of view. This is carried out by association in brain; unlike the human approach the proposed machine vision system applies a mathematical triangulation method to find distances. The human does not compute distances but only locations. Moreover, the system only deals with parallel optical axes and humans have the ability to apply convergence. Regarding this, some stereovision matching systems have applied vergence (Krotkov, 1989). Nevertheless, the main effort in machine vision has been put on parallel systems that apply the concept that humans can see also in the far distance. This is the approach proposed in this expert system.

To reproject the 3D scene to 2D for digital representation, it is necessary to correctly match a point of the environment, seen in both stereo images, with pixel coordinates (x', y') in the first image and (x, y) in the second, see Figure 55. Knowing the robot current location in the world coordinates system and the position of the camera with respect to it, as the camera is mechanically fixed to the robot, the position of the camera in world coordinates is a 3D point C_w. To transform any point P_w (X_w, Y_w, Z_w) into the camera's coordinate system:

$$P_c = R\,(P_w - C_w) \tag{92}$$

where P_c is the point coordinates in the camera coordinates system, R is a rotation matrix. The transformation from world to camera coordinates in homogeneous coordinates:

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix} = \begin{pmatrix} R & -RC_w \\ 0 & 1 \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} \tag{93}$$

The rotation matrix R and the translation vector C_w define the camera's extrinsic coordinates, namely its orientation and position, respectively, in world coordinates. The matrix R transforms from world to camera coordinates.

$$\begin{pmatrix} R & -RC_w \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} R & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} I & -C_w \\ 0 & 1 \end{pmatrix} \tag{94}$$

where I is a 3 x 3 identity matrix.

As a result of this process, a cloud of 3D points, expressed in the camera coordinates system, is obtained, representing the perceived environment. The position of these points shall be transformed to the world coordinate system (O, X, Y, Z), which is independent of the camera, to represent the terrain, objects and obstacles independently of where the camera is. The "Sensor Fusion. Reference Systems Transformations" process, see Figure 69, performs the transformation between these coordinate systems, from point (X_c, Y_c, Z_c) in camera coordinates to its location (X_w, Y_w, Z_w) in world coordinates, once the Z component has been computed thanks to the stereovision system.

In order for the system to automatically reconstruct the surface of the terrain, it is necessary to know the robot pose at the moment the images were captured, expressed as position (X, Y, Z) and orientation (*roll*, *pitch* and *yaw*). The rover's position is estimated from odometry, as indicated in section 4.3.4.5 Position Estimation, and orientation data is obtained from an onboard Inertial Measurement Unit (IMU). This rover's pose data is used then to transform the points cloud obtained as a result of the stereo process for a correct 3D scene reconstruction.

6.7 Experiments and Results

The images used for this study were acquired with a commercially available Videre stereoscopic system of parallel optical axes STH-DCSG-9 color, with two optics separated 9cm, 640x480 pixels resolution and 3mm miniature lenses. They were captured in different locations, days, hours and illumination conditions, see Figure 70. This was intended to verify the robustness of the proposed approach against high variability in the environment.

These digital images were captured and stored as 24-bit color images with resolutions of 640x480 pixels, and saved in RGB raw format (BMP) so no data is lost by compression. Two sets of 90 pair of images were captured and processed. In all of them, it has been observed differences in the illumination pattern, as it can be expected in any stereo system due to the inherent differences in the cameras and optics characteristics.

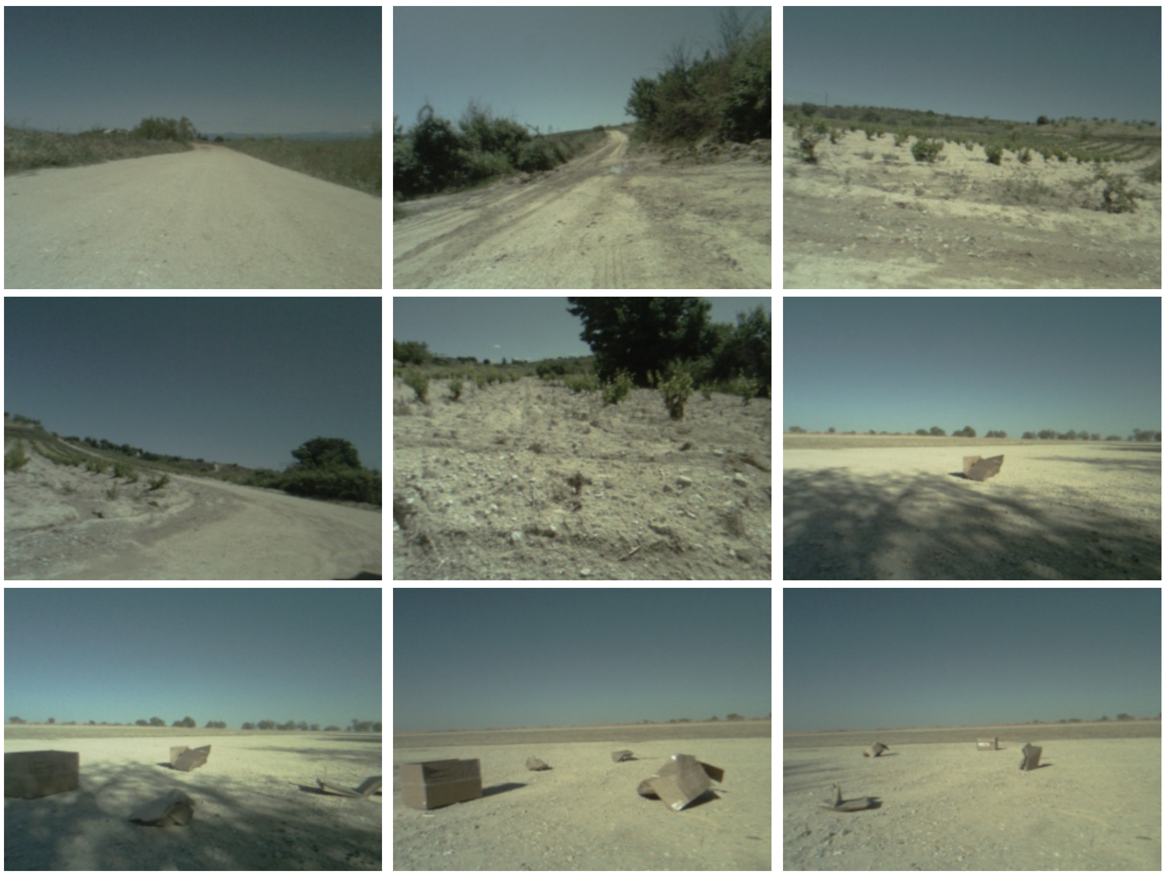


Figure 70. Example of terrain images captured with the stereoscopic system

For the rest of this section, the pair of images shown in Figure 70 (top left) will be used as representative example to illustrate the performance of the proposed system. Analogous results have been obtained using any other stereo pair from these sets of images. It has been arrived to this result after the analysis of 180 stereo pairs of images; thus for simplicity the analysis is carried out with the stereo pair shown in Figure 70 (top left). The images of this pair have 307,200 pixels. From those, this concrete example stereo pair has 156,111 potential matches, the rest belong to remote areas, like sky, with no computable disparity.

The Semi-Global Block Matching (SGBM) algorithm (Hirschmüller, 2005) has been employed to carry out these tests as, according to results obtained in the previous chapter, see section 5.3.4 *Experiments and Results*, the use of this algorithm implies a significant improvement in the efficacy and efficiency of the process with respect to any other matching algorithm. It is, therefore, a promising algorithm to be employed as a baseline and try to improve results, so that the most can be taken out of the stereo process. Executing the Semi-Global Block Matching (SGBM) algorithm on the original images, that is without any previous image processing, it is able to compute 143,296 correspondences. From those, 120,738 are right while 22,558 are incorrectly matched pixels, see Figure 71. The algorithm also misses 35,373 matches that should have been detected. Therefore, 77.34% of total possible correspondences have been detected by the stereo matching algorithm. Figure 74(a) shows graphically the set of disparities obtained.

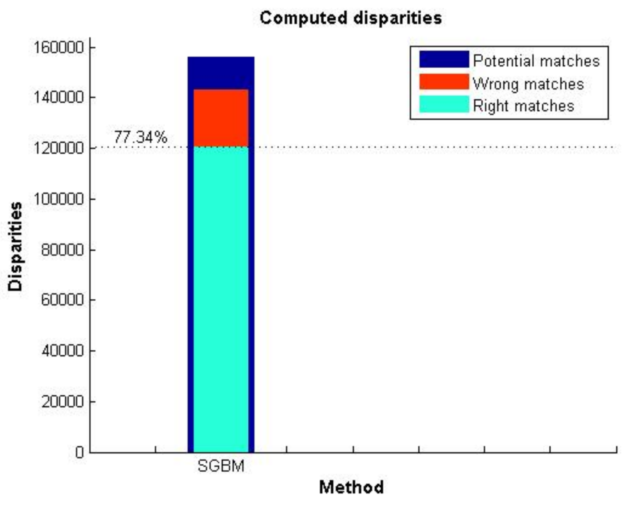


Figure 71. Right and wrong disparities computed by the SGBM algorithm

6.7.1 Histogram matching

In this experiment, instead using the images directly as input to the stereo matching images, the proposed expert system performs an automatic histogram matching process at the initial stage, adjusting the right image as a function of the left one, remaining this unaltered. To analyze these results, examining the images of the stereo pair and comparing their histograms it can be seen they have different intensity distributions, see Figure 72 and Figure 73. These figures show corresponding histograms of left, right and corrected right images, both in RGB and grayscale respectively, ranging [0, 255] in the X axis and number of pixels of each value in the Y axis. The resulting histograms after the matching process can be observed in the last column.

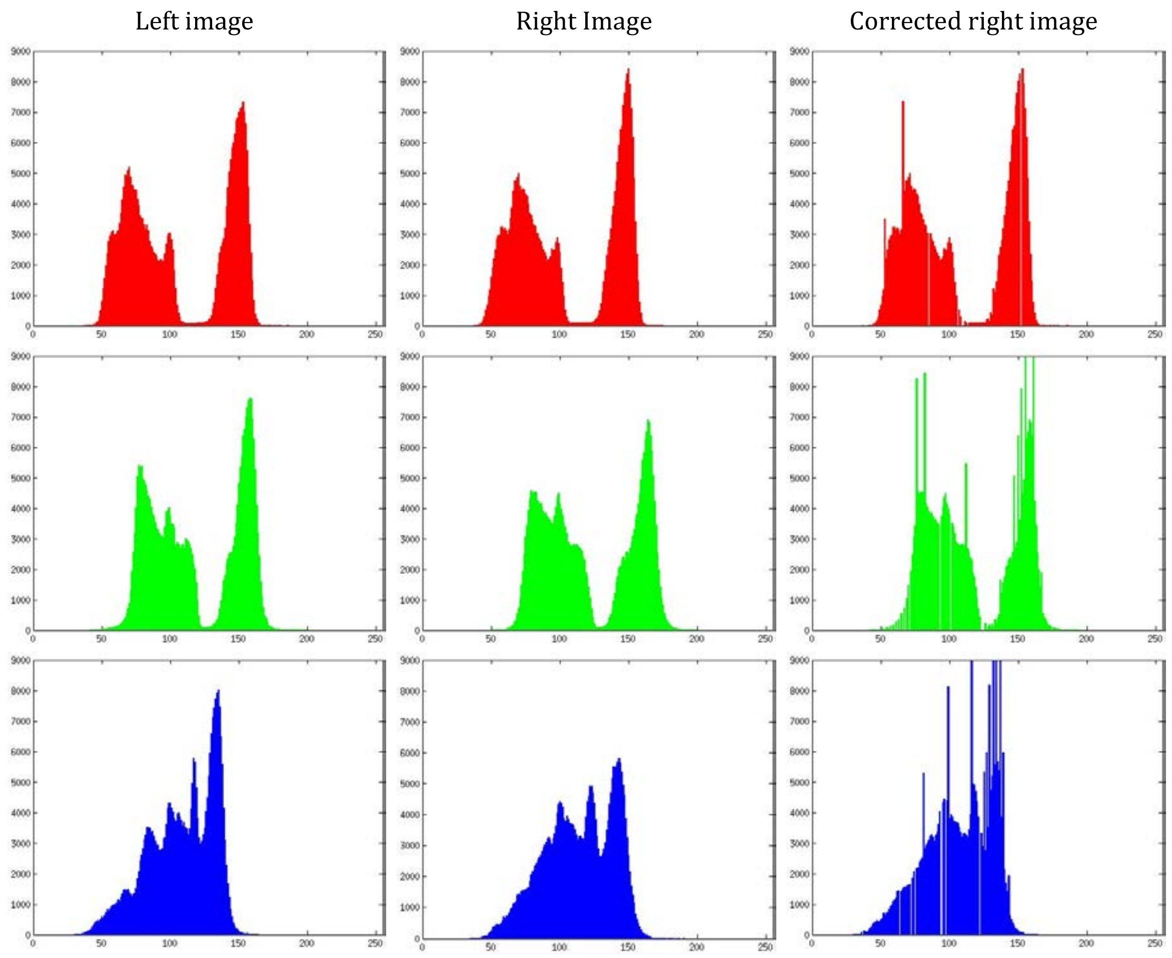


Figure 72. Corresponding left, right and corrected right image RGB channels' histograms

When working in the RGB space, each channel is adjusted separately. From Figure 72, it can be noticed some channels are more alike than others; in this pair of images the blue channels are particularly dissimilar, which could be also appreciated straight away by a human expert observing the images, see Figure 70. However, other channels dissimilarities can go unnoticed to the naked eye.

Despite both images capture the same scene –with a little variation in the point of view- and have been taken simultaneously with two identical cameras, their histograms show different spectral levels. They should be identical theoretically, but they are not in the practice, reinforcing the hypothesis that the constant image brightness (CIB) assumption, introduced previously by what it is assumed that the intensities of corresponding points in two images of a stereo pair are equal, is indeed often false.

It can be noticed how the resulting histograms of the right image after the matching process are more similar to the reference left image. As a result, for this pair of images, the stereo matching process, using the same algorithm, is able to compute 4.86% more correct matches, see Table 6 and Figure 75. That implies obtaining 5,870 right correspondences more, what translates in 5,870 3D points more the reprojection process will be able to use to build a reconstruction of the terrain. Moreover, the number of errors is reduced; fewer wrong matches are computed and fewer correspondences missed. Concretely, after matching the histograms the stereo algorithm returns 31.8% fewer errors and 16.7% fewer missed matches than using the original images. Summarizing, when the system performs the automatic histogram matching of each channel separately previous to the stereo process, for this example the matching algorithm finds 5,870 new correspondences, which also means 5,870 fewer misses than before, and prevent 7,173 mistakes, or incorrectly matched pixels. The resulting computed disparities can be seen in Figure 74 (b).

There is another approach for automatic histogram matching. This is converting the images to grayscale and then matching their histograms, see Figure 73, instead adjusting each RGB channel separately.

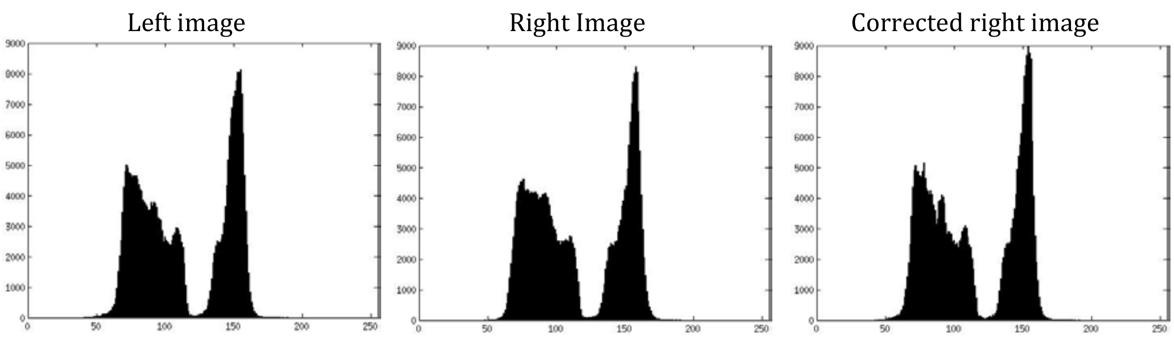


Figure 73. Corresponding grayscale left, right and corrected right image histograms

Following this strategy of matching the histograms of the converted grayscale images, the stereo algorithm is able to compute more disparities than adjusting each channel independently; concretely, it computes 15.92% right matches more, 19.217 points, with respect to performing the stereo process on the original images, see Figure 75. That increment in the number of points computed has a great impact on the reprojection and terrain reconstruction processes. In addition, the number of wrong and missed matches decreases, obtaining 9,208 fewer errors, 40.8% less than before, see Table 6. The resulting computed disparities can be seen graphically in Figure 74 (c).

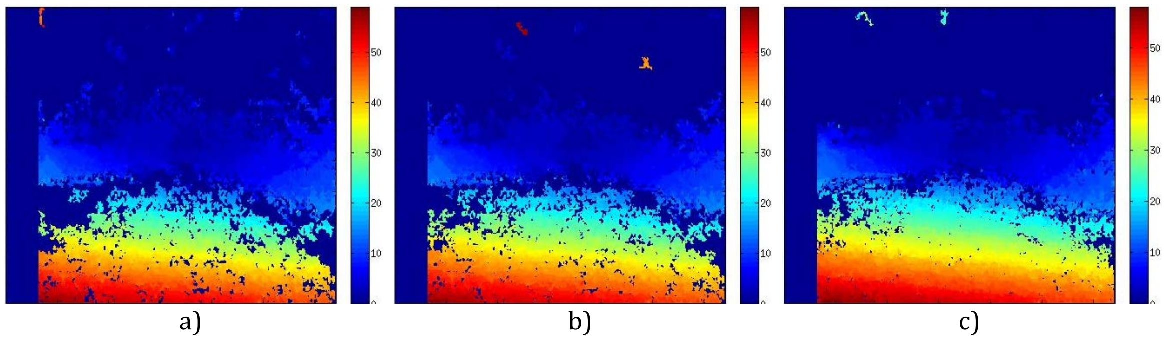


Figure 74. Disparity computed by the SGBM stereo matching algorithm performed on the a) original images, b) images with histograms matched channel by channel, c) images converted to grayscale and matching their histograms

Table 6 shows the results obtained when performing the stereo matching process alone or in combination with the histogram matching filter. *Potential matches* accounts for the total number of possible matches in this pair of images. *Matches found* accounts for the number of matches computed by the stereo algorithm. *Right matches* count the set of correctly computed correspondences; these are the ones that coincide with ground truth data. *Wrong* *matches* indicate the number of correspondences computed by the algorithm that do not coincide with ground truth data and therefore should not have been detected as a valid match. *Missed matches* are those ones that, in spite of having a valid correspondence, the algorithm is not able to compute. The percentages shown below *right* and *missed* matches are computed with respect to the total number of potential correspondences. The number of *matches found* is equal to the sum of *right* and *wrong* matches; the number of *matches found* minus *wrong* plus *missed matches* adds up the total number of *potential matches*.

Ground truth is established by creating templates where all pixels with no possible disparity, like the sky, very far areas and portions of the scene than cannot be seen from both images (left-most vertical area) are removed. Results obtained from the stereo process are compared against this template and the set of matches labeled as correct are supervised by a human expert to double check for possible mismatches.

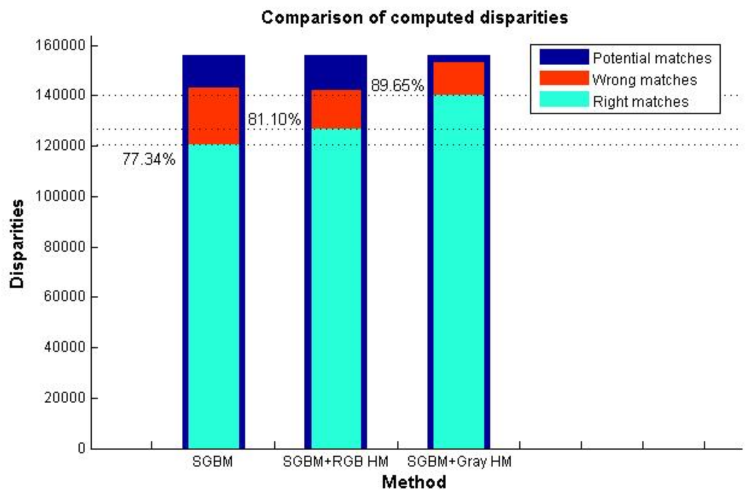


Figure 75. Right and wrong disparities computed by the SGBM algorithm when applying the histogram matching technique

Results in Table 6 show that a higher number of right correspondences are computed and fewer errors are made when images are preprocessed before the stereo process. This means that the automatic process of matching the histograms results in a significant improvement of the stereo process, and therefore of the terrain reconstruction expert system, verifying and supporting the initial hypothesis. The results obtained in this example are extensible to the whole set of images, over 180 pairs.

It can be also observed better results are obtained when images are converted to gray scale before matching their histograms previous to the stereo matching process instead first matching the histograms of each channel separately and then converting the images to gray scale previous to the stereo process. These results are influenced by the matching algorithm used in the stereo process. In this case, the SGBM algorithm computes correspondences based on intensity values of each candidate pixel and its surroundings on gray scale images. In case it receives RGB images, the algorithm initially converts them to gray scale. The process of converting an image from RGB to gray scale is done by eliminating the hue and saturation information while retaining the luminance. Therefore, when the histogram matching process is initially performed on RGB images on each channel separately and then converted to gray scale, hue and saturation information is removed after the histogram matching process is done, altering the input images and influencing the results obtained by the stereo matching algorithm. However, if RGB images and first converted to gray scale the hue and saturation information are removed from both images of the pair prior to the histogram matching process, where pixels intensities are then matched, obtaining better results by the SGBM stereo matching algorithm. Results may differ in case of implementing a different design of a stereo matching algorithm able to work directly on RGB images using the information of the three different channels to compute correspondences.

Table 6. Results obtained by the stereo matching algorithm implemented in the expert system using as input the original images and the ones with the histograms matched

Stereo matching algorithm	Original Images	Histogram Matching (RGB)	Histogram Matching (Gray)
Total image's pixels	307,200	307,200	307,200
Potential matches	156,111	156,111	156,111
Matches found	143,296	141,993	153,305
Right matches (%)	120,738 77.34%	126,608 81.1%	139,955 89.65%
Wrong matches	22,558	15,385	13,350
Missed matches (%)	35,373 22.66%	29,503 18.9%	16,156 10.35%

6.7.2 Homomorphic Filtering

The homomorphic filtering process has been implemented and integrated both, within the testbed introduced in previous chapter in the so called pre-filters category, and within the automatic expert system. Using any of them, a chain of processes is configured so that the pre-filter "homomorphic filtering" is first applied to the input images and the result is passed to the SGBM matching algorithm for disparities computation. The high-pass filter applied in this work, as part of the homomorphic filtering process, is a second order Butterworth filter (Butterworth, 1930). The result of the process and its effect on the subsequent disparity map can be shown in Figure 76.

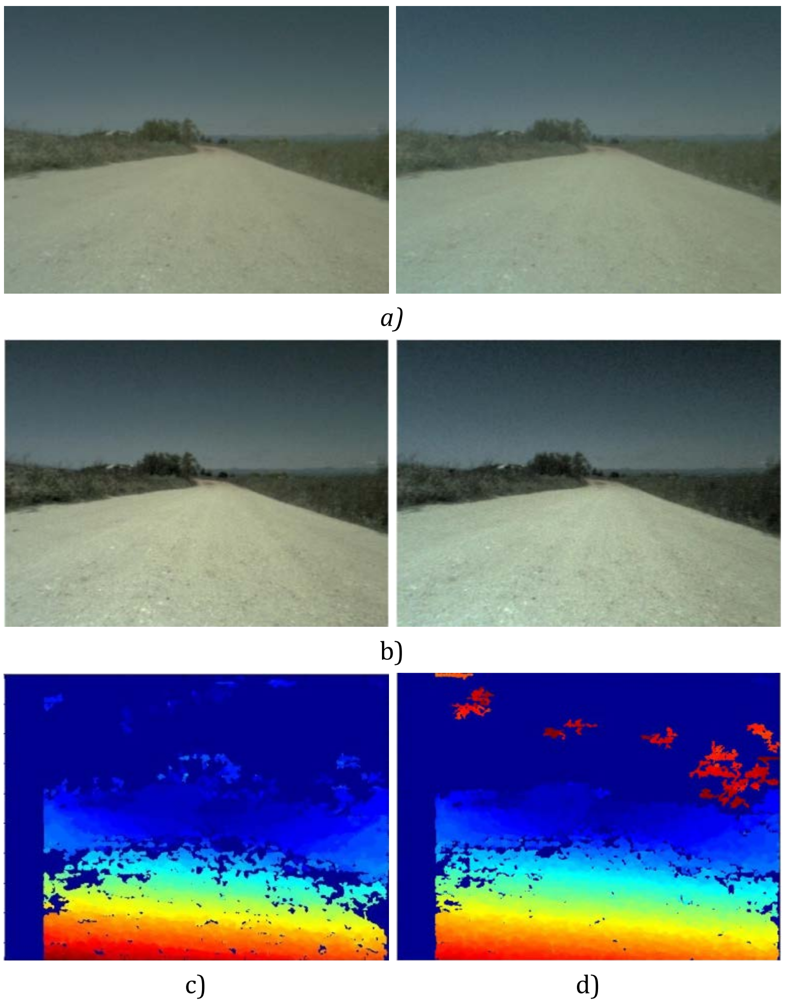


Figure 76. a) Original images, b) images after applying homomorphic filtering, c) disparity map (SGBM algorithm) from the original images d) disparity map (SGBM algorithm) from the filtered images

When the homomorphic filtering process is applied to the input images prior to the matching algorithm, it can be observed the disparity computation process is improved. The same matching algorithm used previously –SGBM- is able to find 11,497 more right correspondences than using the original images directly as input, for the pair of images used as example so far. In this case it computes 132,235 right matches out of the possible 156,111, meaning 84.71%, representing an improvement of 7.37%, see Table 7. It can be seen using this method, besides increasing the number of right correspondences, the number of wrong matches has dramatically decreased, having a great impact when performing the subsequent 3D terrain reconstruction. These results can be graphically seen in Figure 77.

Table 7. Results obtained by the stereo matching algorithm implemented in the expert system using as input the original images and after the homomorphic filtering process ones

Stereo matching algorithm	Original Images	Homomorphic Filtering
Total image's pixels	307,200	307,200
Potential matches	156,111	156,111
Matches found	143,296	141,200
Right matches (%)	120,738 77.34%	132,235 84.71%
Wrong matches	22,558	8,965
Missed matches (%)	35,373 22.66%	23,876 15.29%

However, it has been observed that the process introduces some new errors. In the image shown in Figure 76 (d), they are represented by the reddish areas in the upper part. These errors are wrong matches, or false positives. Specifically, for this example 10,136 false positives were detected. It is a collateral effect of using the homomorphic filtering. This commonly happens in areas where textures are more uniform, and whose values of disparity are close to the maximum value, previously set as the search boundary of the algorithm. The same effect has been appreciated in all experiments using images captured with the stereo system. These false positives have to be removed from the disparities map; otherwise these will be considered as nearby objects and taken as obstacles. Therefore, it is necessary to filter out these errors before proceed to the reconstruction phase. To do that, a novel process has been developed, the *clustering filter*.

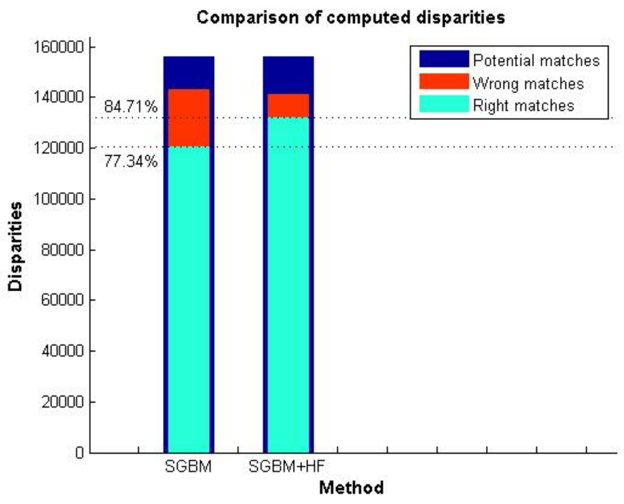


Figure 77. Comparison of disparities computed by the SGBM algorithm when applying the homomorphic filtering technique

6.7.3 Clustering Filter

The clustering filter process has been implemented and integrated both, within the testbed introduced in previous chapter in the so called post-filters category, and within the automatic expert system. As before, chain of processes is configured so that this post-filter applied to the disparity map after the stereo matching process.

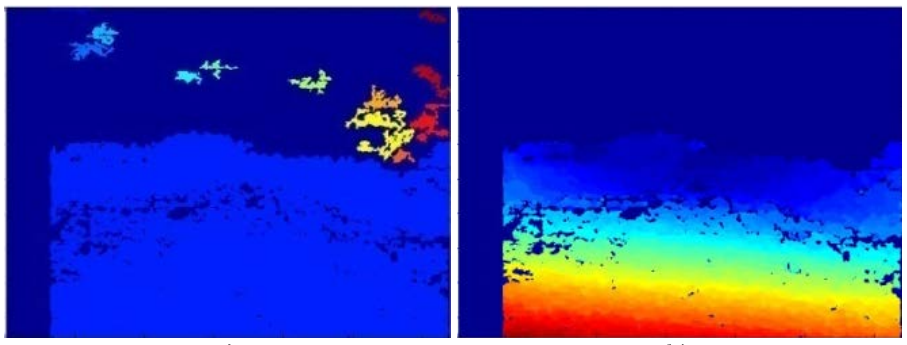


Figure 78. (a) Clusters detected by the filtering algorithm and (b) final disparities computation after filtering out these errors

Applying this technique to the disparity computation example shown in Figure 76, it can be verified the process detects 13 clusters. A large, principal cluster representing the terrain itself and a series of smaller clusters formed by groups of false positives, see Figure 78 (a). The final result of filtering all these clusters out but the principal is shown in Figure 78 (b). The filtering of these errors is achieved without causing any considerable collateral loss of right matches, obtaining an improved and error-free disparity map and a subsequent more dense and correct 3D terrain reconstruction and environment representation, see Table 8.

Table 8. Results obtained by the stereo matching algorithm after the application of the homomorphic filtering and clustering filter processes

Stereo matching algorithm	Original Images	Homomorphic Filtering	Clustering Filter
Total image's pixels	307,200	307,200	307,200
Potential matches	156,111	156,111	156,111
Matches found	143,296	141,200	135,086
Right matches (%)	120,738 77.34%	132,235 84.71%	131,862 84.47%
Wrong matches	22,558	8,965	3,224
Missed matches (%)	35,373 22.66%	23,876 15.29%	24,249 15.53%

The described procedure is configurable, being possible to vary both the spatial neighborhood relation (4 or 8), used to expand each region to compute the different connected components, and the connection distance, which is the maximum distance, or gap, allowed between two pixels to be considered as part of the same region. It can also be specified what clusters are to be kept or removed. It can be indicated as a function of its size, filtering out the smallest clusters, below a given threshold, or the opposite, keeping the largest cluster, about a given threshold. In the experiments carried out with the set of terrain images captured with the real stereoscopic system it has been considered eight neighbors relation and as maximum connection distance of just one pixel. The approach followed with regards the computed clusters has been keeping just the principal one.

It is important to clarify that the presence of rocks or other objects in the images does not cause the detection of them as separate clusters. This method performs grouping based on the spatial continuity of the correspondences found, regardless of its value. As for the objects in the scene, as they are in direct contact with the ground, they do not generate any disconnected component that may form an independent cluster (Correal et al., 2013). This is illustrated in Figure 79, where diverse objects and bushes appear in the environment but they do not form a separate component of grouped pixel separate from the rest. Such items, even though they present a different disparity value than their neighbors, as they are in direct contact with them, do not form any disjointed component and therefore do not cause to be detected as an independent cluster.

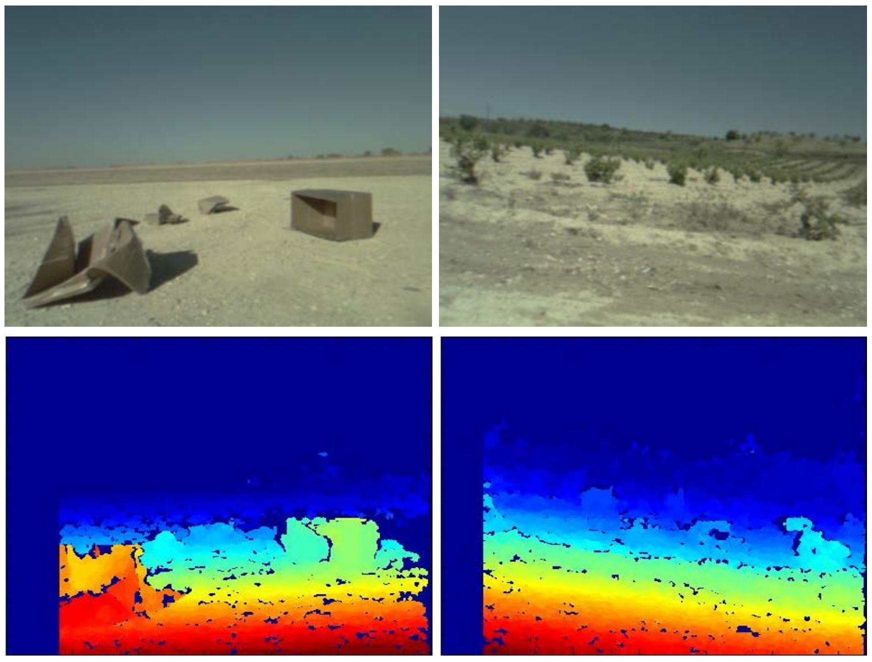


Figure 79. Objects in the environment, even though they have different disparity values than their neighbors, do not form separate clusters

It is possible to conclude the application of a homomorphic filtering process to remove the illumination component of the stereo pair prior to the matching algorithm, and a subsequent clustering process, to filter out wrong matches, introduce significant improvements in the disparity map computation. It has an important impact and represents an advance in the matching process for real images.

Images used in this experiment, are just a representative example to illustrate the differences found in the algorithms when using synthetic or real images and the applicability of some image filtering; analogous results have been obtained from a large set experiments using 180 different stereo pairs, taken with a general-purpose low-cost stereo system under different illumination conditions, days and locations. Thus the results expressed above are valid for the whole set of experiments carried out.

Chapter 7

CONCLUSIONS AND FUTURE WORK

At this point it is necessary to lay out an overall balance on the followed research lines, leading to the following conclusions. Moreover, a number of prospects regarding the research topics addressed and future expansion possibilities are open. In this chapter the contributions made and the most important conclusions derived from them are synthesized. The core of the research involved the development of a vision-based autonomous navigation system for planetary exploration rovers; a framework to support these developments, including simulation and modeling capabilities; analysis and identification of the critical points and areas for improvement, concentrated in the perception subsystem; use and development of techniques such as homomorphic filtering, histogram matching and clustering filter to improve the results of the stereo matching process when working with images of terrains taken from real environments with common and real devices; and development of a testbed where the techniques mentioned above, as well as a number of heterogeneous methods from different authors and sources, are integrated to facilitate the study and analysis of the state of the art in stereo matching methods, which constitutes one of the main current open research areas, as evidenced in the literature.

7.1 Conclusions

In this thesis it has been presented a study and analysis of the autonomy capabilities for planetary exploration vehicles in current and future space exploration missions, according to the stated objective 1, with special emphasis in autonomous navigation, with the aim to reduce missions' operational costs by increasing the level of onboard autonomy.

A framework has been purposely created to support autonomy developments, testing and functional validation of approaches and strategies in the robotic space exploration domain, according to the stated objective 2. The justification of the creation of such a framework has emerged from the lack of availability and accessibility to environments able to support this type of research, which has typically been under the purview of national space agencies. It is partially based on the integration of existing components, open-source tools and packages, with adaptations and extensions.

The integration of different pieces of software together to build up a system has been achieved intra and inter system homogenizing interfaces by creating wrappers and communication interfaces. Software reuse is never a trivial task, as many authors have documented before (Garlan et al., 2009), usually taking far longer than anticipated and resulting in sluggish, huge, brittle and difficult to maintain code. The main drawbacks when working with open-source resources is the fact of relying on sometimes non-proven or immature tools, not supported by a company, where the lack of documentation in many cases makes the learning process hard and time consuming. However, in this case, using existing software and libraries have greatly eased and sped up the development process; accessibility to source code have enabled obtaining a complete control over processes, subsystems and modules' internals, making possible carry out the necessary adaptations and extensions to its original capabilities.

A simulation model of a robotic vehicle based on the NASA's Spirit and Opportunity rovers has been developed, including its sensors and actuators, according to the stated objective 3. The simulation capabilities, based in Gazebo, has some limitations though; aspects such as meteorological conditions, advanced mechanical interactions, sophisticated wheel-soil contact forces and phenomena like slipping or sinking cannot be emulated. However, it is important to emphasize this was not the aim of this research. Instead, it is focused on supporting the development of high-level autonomous navigation strategies, analyzing its performance and validating them at the functional level. Despite these limitations, the simulator fulfills the necessary requirements to carry out these developments and validations.

Supported by the previous framework, an autonomous navigation system for a robotic explorer has been designed and developed from the ground up, according to the stated objective 4. A navigation strategy comparable to the works from NASA have been adopted, but some new tactics have been designed and developed. One of such strategies that differentiate this work to the NASA approach has to do with the computation of the traversability map, evaluating just the strictly necessary areas and cells of the map, speeding up its computation. Also, besides computing straight candidate paths or arcs, a more sophisticated path planning strategy based on splines have been designed. It maximizes and optimizes the navigation process by computing the largest possible safe path within the available map, minimizing the number of required perceptions stages and saving a large amount of time the rover spent stopped performing computation; it speeds up the overall navigation process with respect to the original MER mission approach, ultimately increasing the scientific return of the mission.

According to objective 5, a software architecture based in a layered model has been designed with the purpose to be used as a dynamic research platform. Each level is composed by a set of independent and highly configurable modules, where algorithms can be easily updated or replaced, and more sophisticated and advanced functionality can be integrated over time. This has been actually verified when porting the system from the simulated to the real world, where the perception module had to be modified, replacing some algorithms while the rest of the system remained unaltered and unaware of these changes, confirming the success and utility of the modular design. Devices and hardware independence have been achieved by implementing an abstraction layer. It makes possible porting the designed autonomous navigation system to different vehicles, just by adapting this hardware abstraction layer to the concrete robotic platform, while the rest of the layers and modules remain unaltered.

The navigation system has been tested both, in simulation and with a physical mobile robot, according to the stated objective 6. Once the navigation system were mature enough and validated in simulation, it was ported to a physical robotic platform for field testing and final validation. This portage resulted in a smooth process thanks to the extensive preceding test campaigns in simulation and the modularity of the architecture.

Following the objective 7, analysis of results revealed computation resources requirements are in accordance to the ones available in previous missions and to the capabilities of present and future flight processors used for space mission. The designed navigation system performance have been validated according to the results obtained from the field testing campaign, as stated in objective 8. It is important to note that these developments are not only valid for space exploration but also for terrestrial applications in outdoor, natural environments such as patrolling and surveillance, borders' security, carriage of goods or country and forests' firefighting to name a few.

Results revealed also the perception process, based in stereovision, is one of the most critical and consuming processes, in terms of resources and computing requirements, identifying this as the focus of potential optimization efforts, in accordance to objective 9. In the process of analyzing the state of the art in stereovision algorithms, it has been detected the broad range of publications and the heterogeneity of the available resources makes this analysis a hard and complex task. To assist that task, a flexible testbed with a friendly interface has been designed and created, part of the objective 9. Major contributions of this stereo testbed with respect to previous related works, like the Middlebury stereo correspondence software (Scharstein and Szeliski, 2002), are: a graphical interface that allow a friendly interaction for selection and configuration of algorithms and its parameters and the avoidance of the use of scripts; it allows to perform multitude of tests in very short time; graphical representation of results for easy analysis and comparison; use of different formats of input images, not constrained to the set of images distributed with the testbed; portability, architecture and operative system independence, as the testbed can been executed in different platforms without the recompiling the source code or configuring a runtime environment, representing one of the most important contributions; flexibility in the inclusion of algorithms written in different source languages, as well as easy connection with computer vision libraries such as OpenCV; openness of source code, so users can adapt both the algorithms and the testbed or its graphical interface to their concrete needs, and include new features, what represents a crucial aspect, as published algorithms may serve as inspiration for others, who can extend or adapt them to achieve more robust and optimized approaches over time the whole community can benefit from; scalability, as it has been designed to evolve and include more algorithms, image processes and functionality over time, receiving contributions from the research community as new works are published; once integrated, new algorithms can be easily tested and compared with others, analyzing its strengths and weaknesses to assess its contribution to the state of the art.

A series of heterogeneous stereo matching algorithms and image processes have been adapted, standardized and integrated into the testbed, taking advantage of the large collection of existing resources and work already done by the research community. This integration and standardization process does not require a major effort as the testbed is designed to be very flexible and allow the inclusion of algorithms written in different programming languages. The set of stereo algorithms has been analyzed. The testbed allows testing many algorithms varying its parameters in a very short time, until the most suitable one for each concrete application is identified. It prevents the implementation of algorithms just for testing purposes, what can be done within the testbed, representing a considerable saving of time and effort, demonstrating the utility of the testbed.

The undertaken research is motivated by the evidence of the unfavorable performance of various matching algorithms against real images of uncontrolled and unstructured outdoor environments. As a result, it has been verified several processes when combined to be executed consecutively, results in a major improvement when computing stereo correspondences. Therefore, following the stated objective 10, an automatic expert systems for image correction and terrain reconstruction in stereo vision applications has been proposed. It adjusts these intensities by the automatic successive application of a combination of processes and filters in three consecutive stages, mapping the expert knowledge. These processes include previous treatments on the input images and filtering techniques following the matching phase.

One of such identified techniques is the histogram matching process, used to automatically adjust the intensity of one image of the pair as a function of the other. Another applied technique is the homomorphic filtering; it removes the illumination component of the input stereoscopic images, causing an improvement in the disparity map. The homomorphic filtering process however introduces some errors; they can be easily filtered with clusters-based process following the principle of spatial continuity, which eliminates false positives.

Both processes applied together represent a breakthrough and a significant improvement in the efficiency of the matching process for real terrain images, assuming a minor additional computational cost with respect to applying solely the matching process, although it does not suppose a significant overhead. Once the intensities have been corrected by applying these methods, the stereo matching process is able to obtain a larger number of correspondences, reducing also errors and missed matches, what have a great impact in the subsequent 3D reprojection and terrain reconstruction processes. It also probes the constant image brightness (CIB) assumption is often erroneous.

The proposed expert system could be extended to be applied to other applications such 3D object segmentation and recognition. This expert system has been designed with an open architecture, so that in the future it will be possible to replace or add new modules, being of particular interest to study different stereo matching algorithms, or to add a knowledge-base for improving image correction based on the accumulated knowledge.

7.2 Future Work

There is still room for improvements in the subsystems, methods and tools proposed. In addition, there are many open research lines within the context of the work associated with this thesis. Next, some of the most promising lines of future related research are indicated.

Within the support framework, the control center can be extended to allow operators to create sophisticated plans of activities to be uploaded to the rover, including usage of scientific instruments, and modeling communication constraints to emulate the real interplanetary communications cycle. Including 3D reconstruction capabilities from rover imagery may support the operator/researcher plans design, as well as incorporating tests automation capabilities for debugging purposes. Regarding simulation, terrains and rovers models can be further sophisticated by including phenomena such as soft sand, wheels' slipping and sinking, as well as models of scientific instruments that the robot should use to perform experiments and energy usage models, which have an important impact on the overall autonomy strategy. Current devices, sensors and actuators' models can be extended with noise and error models to enhance algorithms' management of uncertainty.

New and more sophisticated navigation strategies can be designed. Stereo vision algorithms to manage shadows, environmental dust or sun glaring can be analyzed, as well as visual odometry techniques to correct localization errors accumulated from mechanical odometry. The rover can also be provided with onboard task planning capabilities by incorporating AI (Artificial Intelligence) planners, taking into account aspects such as rover resources, power consumption, instruments usage, solar panels loading rate or sun position when designing activities and contingency plans.

Regarding the initiated stereo testbed open-source project, it can be extended to include new matching algorithms and more image processes as well as extended functionality such as test automation capabilities, scheduling a series of tests to be executed without supervision during low activity periods or by night; this way, lots of data from different tests can be gathered together for later analysis, automatically storing it to be presented in a structured mode to ease its subsequent analysis. This analysis can be somehow automated, reporting evaluations, comparisons and elaborated conclusions from the raw gathered data. Some contributions have been already received from a number of authors to be integrated into the next version of the testbed.

Further research can be done to enhance the current image filters and processes, or design new techniques for automatic image correction. Potential enhancements in the clusters-based filter can be done to take into account the disparity values from pixels within a given cluster in comparison with nearby clusters and as a function of the separation distance, to automatically determine if the cluster should be removed or kept. The use of techniques such as simulated annealing to remove exceptional errors that could stand for isolated pixels after disparities computation and filtering can be analyzed.

A current disadvantage of the current expert system's proposed approach is precisely the lack of awareness of the system regarding how to perform the histogram matching process in the most effective way. It may occur that matching each RGB channel separately lead to obtain results not as good as they could be performing the process over previously gray scale-converted images. Therefore, the capability of automatically select the most appropriate image process and optimally configure its parameters as a function of the concrete application and operational conditions is a promising line of work, ensuring the maximum effectiveness and adaptation of the system.

As a final conclusion, it can be said incoming missions to Mars, other planets and celestial bodies in the Solar system are currently being scheduled. Once a first approach to an autonomous navigation system, a series of algorithms and techniques and the basic and necessary infrastructure has been developed, which has been the focus of this work, more sophisticated algorithms and navigation strategies can be analyzed, designed and developed with the support of the created tools, that make possible to design future rovers more autonomous, robust and reliable, advancing the state of the art in this domain and taking mankind to the next level in robotic planetary exploration.

RESUMEN EN ESPAÑOL

8.1 Introducción

La exploración de superficies planetarias mediante el uso de robots o rovers, términos que se usarán indistintamente para referirnos a la misma entidad, es una tarea complicada, con requisitos de seguridad críticos y grandes limitaciones de comunicación. Por lo tanto es esencial un cierto nivel de autonomía local a bordo de los robots, de manera que éstos puedan tomar sus propias decisiones de forma independiente del control de Tierra, lo que reduce los costes operativos y maximiza el retorno científico de la misión. Además, existe una creciente demanda de las capacidades de autonomía para futuras misiones planetarias. Se espera que las siguientes misiones incluyan un alto nivel de autonomía, donde los científicos en Tierra simplemente designen localizaciones a visitar por el rover sobre las imágenes adquiridas y descargadas previamente.

La navegación autónoma es uno de los aspectos más importantes y arriesgados de estas operaciones. El vehículo tiene que ser capaz de trasladarse de un lugar a otro para tomar medidas, muestras y realizar experimentos científicos. Si el vehículo llega a una zona muy empinada, pasa sobre una roca, un hueco o por un área arenosa, por citar algunas situaciones potencialmente peligrosas, éste puede volcar, encallar, atascarse o perder tracción, y poner toda la misión en riesgo, ya que nadie puede ayudar al vehículo a salir y continuar. Para lograr esto, el robot debe realizar una serie de tareas específicas basadas en una estrategia de navegación exploratoria consistente en un ciclo iterativo de percepción-mapeado-planificación de trayectorias-navegación. Es decir, el rover debe percibir su entorno haciendo uso de los sensores apropiados; crear una representación interna del entorno; y calcular trayectorias seguras y adecuadas para llegar a la ubicación que haya sido comandado.

Sin embargo, una de las primeras y principales dificultades que los investigadores en este ámbito tienen que afrontar es que las herramientas existentes para apoyar la investigación suelen ser propiedad de las agencias espaciales, fuera del alcance de la mayoría de investigadores. Además, es indispensable hacer un uso extensivo de la simulación, que permita la creación de modelos que repliquen el vehículo, los sensores, el terreno y las condiciones operacionales, así como de herramientas que apoyen el desarrollo de los algoritmos necesarios para conseguir una estrategia de navegación autónoma basada en visión artificial para exploración robótica de planetas, que permita analizar y validar de manera funcional los algoritmos y métodos de autonomía, así como la monitorización de datos. Por lo tanto, esta dificultad debe ser superada de algún modo. En el caso de la propuesta que se presenta, la estrategia seguida ha sido diseñar a propósito un entorno de trabajo, basado en la integración y adaptación de herramientas existentes, enfocado en el ámbito de exploración planetaria para apoyar los desarrollos y trabajos de investigación, así como el diseño y desarrollo de una estrategia de navegación autónoma.

Dentro de la navegación autónoma, la percepción es una de las tareas más cruciales. Las cámaras estéreo resultan ser el dispositivo preferido para la percepción de entornos 3D en muchas aplicaciones de exteriores. La visión estereoscópica es un mecanismo para obtener datos de profundidad o de rango basados en imágenes. Para la obtención de la visión estereoscópica se dispone de un dispositivo consistente en dos cámaras separadas por una distancia dada de manera que se obtienen dos puntos de vista diferentes de una misma escena que se proyectan sobre cada una de las cámaras para obtener un par de imágenes estereoscópicas, de forma similar a la visión binocular humana. Este par de imágenes constituye la entrada al proceso de correspondencia estéreo. Al comparar las dos imágenes, se obtiene información relativa de profundidad, en forma de desigualdades, que son inversamente proporcional a la distancia a los objetos. Este proceso calcula la diferencia en la posición de un conjunto de características o píxeles de una imagen con respecto a la otra, generalmente a lo largo del eje horizontal. La profundidad a la que se encuentran los objetos en la escena se establece mediante la triangulación de las disparidades obtenidas del proceso de correspondencia, siempre que la posición de los centros de proyección, la distancia focal y la orientación de los ejes ópticos sean conocidas. A continuación se realiza una reproyección al espacio 3D mediante una transformación de perspectiva, obteniendo un conjunto de coordenadas con respecto al sistema de referencia del mundo.

El cálculo de correspondencias en imágenes estéreo es un área muy investigada, que cuenta con una extensa base de literatura publicada y un amplio espectro de algoritmos heterogéneos disponibles en diversos lenguajes de programación. Sin embargo, no existe un enfoque único que pueda ser considerado como el mejor, capaz de resolver todos los problemas posibles. Por lo general, cada técnica es adecuada para un conjunto de condiciones pero no para otras. Por lo tanto, como en todos los ámbitos de la investigación, el primer paso consiste en realizar un análisis del estado del arte para verificar si ya existe alguna solución o técnica capaz de satisfacer las necesidades concretas y en qué medida. Esta es, de hecho, una de las principales dificultades que investigadores en el área de la visión estéreo –como en otras muchas áreas- han de afrontar, ya que esta tarea no es trivial en absoluto, dada la amplia gama de recursos y literatura disponible, de manera que es necesario realizar un considerable esfuerzo para analizarlo a fin de determinar qué estrategia puede adaptarse a cada conjunto concreto de requisitos. Esta dificultad se debe principalmente a la naturaleza heterogénea de los algoritmos y técnicas disponibles. Esta heterogeneidad y falta de estandarización hacen del análisis del estado del arte, así como de la evaluación y comparación de algoritmos, una tarea difícil, complicada y lenta.

Una vez que una técnica o algoritmo ha sido evaluado como prometedora para ser probado para navegación de robots en terrenos naturales, surgen nuevos problemas cuando se aplican en entornos reales, en contraste con su aplicación a entornos controlados y de laboratorio, lo que ocurre en una infinidad de trabajos publicados. El problema de la correspondencia en sistemas estereoscópicos deriva del hecho de que las imágenes de las cámaras, aunque similar, muestran diferentes niveles de intensidad para la misma entidad física en la escena 3D. La razón principal de esta característica radica en la diferente respuesta de los sensores de la cámara a la luz de la escena, así como a la diferencia en la proyección de la escena sobre cada imagen debido a los diferentes puntos de vista relativos de cada cámara. Eso hace necesario dedicar un importante esfuerzo de investigación para corregir estas desviaciones típicas de cualquier sistema estéreo. Este problema no ha sido todavía satisfactoriamente resuelto, especialmente en entornos no controlados ni estructurados. Esta es la razón principal por la que la literatura sobre este tema es tan amplia.

Este hecho permite deducir que el problema relativo a la baja eficiencia de algunos algoritmos proviene de su aplicación a imágenes reales, lo que requiere una solución global a este problema general. Esto justifica la necesidad de un tratamiento previo de las imágenes. El objetivo es conseguir la máxima similitud en los niveles espectrales de las imágenes del par estéreo, a nivel de píxel, que es exactamente lo que sucede en el caso de las imágenes simuladas. Por lo tanto, son necesarios métodos capaces de corregir las diferencias en ambas imágenes desde el punto de vista radiométrico.

Por último, el propósito principal de la fase de percepción y el uso de la visión estereoscópica, es realizar una reconstrucción 3D del terreno que sirva al robot para crear una representación interna de su entorno y poder planificar rutas y trayectorias adecuadas para navegar de forma efectiva de un lugar a otro. Eso requiere una reconstrucción de alta calidad de la superficie y el desarrollo de un sistema capaz de realizar tal tarea, similar a la que un experto humano haría en una situación similar. Por lo tanto, el conocimiento necesario tiene que ser proyectado en el sistema siguiendo la estrategia lógica que un experto aplicaría.

El trabajo que se presenta en esta tesis, aunque está centrado en la navegación autónoma de robots basada en visión para aplicaciones de exploración espacial, puede ser fácilmente portado y aplicado en entornos terrestres. Cualquier aplicación en la que un robot tenga que atravesar de forma autónoma un terreno natural y abrupto sin información a priori, en la que el entorno sea desconocido y tenga por tanto que ser explorado y descubierto a medida que el robot avance haciendo uso de cámaras estéreo para una percepción 3D, se enfrentará a análogas dificultades y condiciones. Algunos ejemplos de aplicaciones terrestres similares son patrullaje autónomo con robots en entornos rurales, tales como detección de incendios en bosques, vigilancia en instalaciones agrarias o control fronterizo. Por lo tanto, futuros proyectos y trabajos de investigación en estas áreas de aplicación también pueden beneficiarse de los trabajos presentados en esta tesis.

8.2 Motivación

El trabajo de investigación de esta tesis es parte de una colaboración entre la Universidad Complutense de Madrid y la compañía TCP Sistemas e Ingeniería en el área de navegación autónoma de rovers de exploración planetaria. Ambas entidades han participado en dos proyectos, titulados: *AutoRover: estudio de Autonomía Basada en Imágenes para rovers de exploración planetaria* y *Visión estereoscópica párrafo Auto-rover: estudio de Autonomía Basada en Imágenes*. El primero proviene de la participación de la mencionada empresa en la convocatoria pública 2259/2007 (BOCM 272 de 15/11/2007) para promover la innovación de la Comunidad de Madrid en el sector aeroespacial con financiación de la Fundación Regional Europea (Referencia 04-AEC0800-000035 / 2008). El segundo corresponde a una extensión del programa anterior dentro del Programa Nacional de Investigación en el sector aeroespacial, según el orden PRE / 998/2008 (BOE 11/04/2008) Ministerio de la Presidencia (de referencia SAE-20081093). Ambos proyectos se enmarcan en el área de navegación autónoma de robots sobre la superficie de Marte, cuyo objetivo principal es el diseño y desarrollo de un sistema basado en visión para navegación autónoma de rovers de exploración planetaria.

Los trabajos de investigación relacionados con los proyectos anteriormente mencionados se han llevado a cabo en la Facultad de Informática de la Universidad Complutense de Madrid bajo la dirección de los supervisores de esta tesis.

Desde el punto de vista de su utilidad industrial, el trabajo desarrollado se debe a la necesidad planteada por la Agencia Espacial Europea para su próxima misión robótica a Marte llamada ExoMars, donde un rover navegará a través de la superficie marciana en búsqueda de señales de vida. El objetivo principal es aterrizar el vehículo en un lugar con alto potencial para la búsqueda de material orgánico bien conservado, sobre todo desde los inicios de la historia del planeta. Se recogerán muestras con un taladro y serán analizados con instrumentos de última generación para el establecimiento de las propiedades físicas y químicas de estas muestras. Este futuro rover estará diseñado para ser altamente autónomo. El rover construirá un modelo 3D de su entorno utilizando imágenes estéreo, que analizará con el fin de establecer los objetivos científicos más adecuados. A continuación, navegará de forma autónoma al destino seleccionado y hará uso de su brazo robótico e instrumentos a bordo para recoger y analizar datos. La intención es disminuir por tres o más el tiempo que los últimos rovers de la NASA necesitaban para planificar y programar sus acciones para obtener las muestras, lo que aumentaría drásticamente el retorno científico de la misión.

La Agencia está en la necesidad de licitar múltiples contratos a la industria europea para el diseño y desarrollo de los numerosos sistemas que integran una misión de una complejidad y características como éstas. Los trabajos presentados en esta tesis se centran en el desarrollo de las capacidades autónomas de vehículos robóticos, y más concretamente en el sistema de navegación, con especial énfasis en el subsistema de percepción, basado en visión estereoscópica.

Por otra parte, desde el punto de vista tecnológico, los presentes trabajos están motivados por el hecho de tratar de mejorar los resultados obtenidos a partir de los algoritmos de correspondencia estereoscópica, teniendo en cuenta los problemas de su aplicación a situaciones e imágenes reales, lo que implica ciertas complejidades y problemas no tenidos en cuenta en muchos trabajos y publicaciones relacionadas, que suelen estar realizados en entornos controlados o limitados a imágenes generadas sintéticamente.

8.3 Objetivos

En vista de las consideraciones indicadas en los párrafos anteriores, se proponen los siguientes objetivos de investigación:

1. Llevar a cabo un estudio y análisis de las capacidades actuales de autonomía de los vehículos de exploración planetaria, con especial énfasis en la navegación autónoma. Comprender el estado del arte en este campo y en las tecnologías y herramientas disponibles relacionadas con el desarrollo de esta área de investigación.

2. Diseño y desarrollo de un entorno de trabajo que apoye el desarrollo de las capacidades de autonomía, estrategias de navegación, algoritmos y su validación

funcional, dada la falta de disponibilidad y accesibilidad a entornos equivalentes, típicamente de acceso limitado para las agencias espaciales nacionales.

3. Creación de los modelos de simulación necesarios para replicar el vehículo, sus sensores, el terreno y las condiciones operacionales, incluyendo aspectos tales como la cinemática y la dinámica del vehículo, fuerzas de contacto, gravedad o fricción para analizar la interacción del rover con su medio con el nivel apropiado de fidelidad. El objetivo es emular todo el proceso de operación del rover en una superficie planetaria.

4. Diseño y desarrollo de un sistema de navegación autónomo basado en visión artificial para exploración espacial robótica de forma que un vehículo sea capaz de desplazarse de un lugar a otro para tomar medidas, muestras y realizar experimentos científicos. La autonomía de un rover implica el conjunto de algoritmos que han de ser embarcados en el mismo para proporcionar a éste la inteligencia y capacidades necesarias para tomar sus propias decisiones, operar en entornos desconocidos y manejar situaciones inesperadas, minimizando la dependencia de los operadores de control en Tierra.

5. Diseño de una arquitectura software modular y escalable para estructurar y organizar el conjunto de algoritmos previamente desarrollados de manera que los subsistemas sean independientes y a la vez capaces de interactuar entre sí para lograr las capacidades deseadas de autonomía de alto nivel.

6. Realizar una serie de experimentos, tanto haciendo uso de las capacidades de simulación del entorno de trabajo como llevando a cabo una campaña de experimentos de campo con un robot físico y hardware real, a fin de analizar y validar las capacidades de autonomía del sistema de navegación autónomo desarrollado y la utilidad del entorno de trabajo para apoyar este tipo de desarrollos y pruebas.

7. Analizar los resultados obtenidos tras los experimentos realizados y el comportamiento del sistema de navegación.

8. Validar la estrategia de navegación diseñada, la arquitectura software y los algoritmos desarrollados.

9. Identificar las potenciales áreas de mejora y desarrollar las herramientas necesarias para facilitar la identificación de dichas áreas. Poner estas herramientas a disposición de la comunidad científica siempre que sea posible.

10. Desarrollar los métodos y técnicas previamente identificadas como áreas de mejora dentro de la estrategia de navegación autónoma; desarrollar las herramientas necesarias para facilitar la creación de estos métodos y tecnologías.

11. Identificar líneas futuras de investigación.

12. Difusión de los resultados obtenidos, conocimientos, descubrimientos y desarrollos a través de publicaciones científicas o de otros mecanismos como la publicación de software siempre que sea posible.

8.4 Resultados

A partir de los objetivos propuestos y teniendo en cuenta los aspectos anteriormente destacados, el propósito es resolver los problemas considerados y proporcionar a la comunidad científica un conjunto de estrategias de solución y herramientas que también pueden propagarse a otros ámbitos de naturaleza similar. Los resultados y aportaciones realizadas por esta investigación se resumen en los siguientes puntos:

1. Se ha realizado un estudio y análisis de las capacidades de autonomía para vehículos de exploración planetaria, con especial énfasis en navegación autónoma. El estado actual del arte en este campo y en las tecnologías y herramientas disponibles

relacionadas con los propuestos temas de investigación ha sido asimismo analizado. Este trabajo se describe en el capítulo 2 de esta tesis.

2. Se ha diseñado y desarrollado un entorno de trabajo para apoyar el desarrollo de los algoritmos de autonomía, incluyendo las estrategias de navegación y su validación funcional, creado en parte integrando herramientas y algunos paquetes de terceros. Este trabajo ha sido motivado por la falta de disponibilidad y accesibilidad a herramientas equivalentes, típicamente de acceso limitado para las agencias espaciales nacionales. Este trabajo se describe en el capítulo 3 de esta tesis y en Correal y Pajares (2011a) y Correal et al. (2014B).

3. Se han creado modelos de simulación para replicar un vehículo robótico de exploración espacial, sus sensores, terrenos y las condiciones operacionales, incluyendo aspectos tales como la cinemática y la dinámica del vehículo, la fuerza, la gravedad, o fricción, emulando el proceso completo de operación del rover sobre una superficie planetaria. Este trabajo se describe en el capítulo 3 de esta tesis y en Correal y Pajares (2011a) y Correal et al. (2014B).

4. Se ha diseñado y desarrollado desde el inicio un sistema de navegación autónoma basado en visión para exploración espacial robótica, apoyándose en librerías específicas de terceros. Éste permite que un vehículo robótico se desplace de un lugar a otro de forma autónoma, tomando medidas, muestras y realizando experimentos científicos. Esta autonomía conlleva un conjunto de algoritmos que deberán ser embarcados en el robot para dotar a éste con la inteligencia y capacidades necesarias para tomar sus propias decisiones, operar en entornos desconocidos y manejar situaciones inesperadas, minimizando la dependencia de operadores de control en Tierra. Ello incluye el diseño y desarrollo de varias estrategias de navegación exploratoria para calcular trayectorias de forma autónoma en entornos planetarios remotos. Este trabajo se describe en el capítulo 4 de esta tesis, y está basado en Correal y Pajares (2011b) y Correal et al. (2014B).

5. Se ha diseñado una arquitectura software modular, escalable, basada en capas. Ésta es fundamental para estructurar y organizar el conjunto de algoritmos de los que se compone la inteligencia embarcada en el rover, de modo que cada subsistema sea independiente de los demás y capaz de interactuar con otros para lograr alcanzar el alto nivel de autonomía deseado. Este trabajo se describe en el capítulo 4 de esta tesis y en Correal y Pajares (2011b) y Correal et al. (2014B).

6. Se han realizado un conjunto de experimentos, tanto utilizando las capacidades de simulación del entorno de trabajo desarrollado como un robot real con sus dispositivos, validando tanto las capacidades de autonomía del sistema de navegación como el papel del entorno de trabajo para apoyar tales desarrollos y pruebas. Este trabajo se describe en el capítulo 4 de esta tesis y en Correal y Pajares (2011b) y Correal et al. (2014B).

7. Se han identificado potenciales áreas de mejora, principalmente relacionadas con el subsistema de percepción. Se han desarrollado varios métodos y técnicas para mejorar estos resultados. Una de estas técnicas es la *correspondencia de histogramas*. Ésta normaliza la imagen del par estéreo ajustando la distribución de color de una imagen con respecto a la otra. Esta técnica mejora el contraste de las imágenes utilizando funciones de distribución acumulada para transformar los valores de intensidad en una imagen, o los valores del mapa de colores de una imagen indexada, de manera que el histograma de la imagen de salida coincide aproximadamente con el histograma de la otra imagen, aproximando ambas componentes de iluminación. Este proceso, cuando se aplica a las imágenes de entrada antes del proceso de correspondencia estéreo, conlleva una mejora significativa de los resultados de la fase de percepción, pasando de una detección del 77% de las correspondencias existentes a casi un 90%, ver Table 6 y Figure 75. Este trabajo se describe en el capítulo 6 de esta tesis y en Correal et al. (2014a).

8. Otra contribución significativa a la mejora de la etapa percepción de un explorador robótico ha sido la aplicación a las imágenes de entrada de una técnica llamada *filtrado homomórfico*, que elimina la componente de iluminación de una imagen,

preservando la reflectancia. Cuando se aplica a las imágenes de entrada antes del proceso de correspondencia estéreo, se consigue una mejora significativa de los resultados obtenidos por el sistema de percepción, pasando de una detección del 77% de las correspondencias existentes a casi un 85%, y consiguiendo sobre todo una reducción significativa en el número de errores cometidos durante el proceso que no se conseguía anteriormente, de alrededor de un 250%, ver Table 7 y Figure 77. Este trabajo se describe en el capítulo 6 de esta tesis y en Correal et al. (2013).

9. Una tercera técnica ha sido diseñada, desarrollada e integrada en el sistema de navegación autónoma para mejorar el proceso de percepción del robot; ésta se ha denominado *filtrado por agrupación*. Este proceso, basado en la filosofía de clusters o grupos, se aplica después del cálculo de disparidades para eliminar errores de correspondencia, descartando falsos positivos. Consiste en la agrupación de píxeles en componentes conectados, basados en el principio de continuidad espacial. El resultado del proceso es una lista de grupos que son analizados para mantenerlos o filtrarlos. Este proceso, cuando se aplica sobre el mapa de disparidades obtenido después del proceso de correspondencia estéreo, conduce a alcanzar una mejoría significativa de los resultados de la fase de percepción, consiguiendo reducir significativamente el número de correspondencias erróneas producidas por el efecto colateral de la aplicación del filtrado homomórfico, de alrededor de un 280%, ver Table 8 y Figure 78. Este trabajo se describe en el capítulo 6 de esta tesis y en Correal et al. (2013).

10. Se ha iniciado un proyecto de código abierto como parte de esta tesis, que consiste en un banco de pruebas para algoritmos de visión estéreo para facilitar el análisis del estado del arte y dar soporte a los investigadores para analizar, evaluar y comparar métodos de correspondencia estéreo. Éste integra una serie de algoritmos heterogéneos, y algunas técnicas de pre y post filtrado, adaptando y estandarizando sus interfaces de manera que éstos puedan trabajar conjuntamente y combinar procesos para obtener resultados mejorados en comparación con la aplicación de cada proceso por separado. Este banco de pruebas ha sido puesto libremente al servicio de la comunidad científica, y ha sido descargado más de mil veces hasta la

fecha actual, lo que denota un gran interés. Varias nuevas contribuciones han sido recibidas hasta el momento desde la comunidad y la próxima versión del banco de pruebas está actualmente en desarrollo. Este trabajo se describe en el capítulo 5 de esta tesis.

11. Se ha propuesto y desarrollado un sistema experto de carácter automático para la corrección de imágenes y reconstrucción del terreno en aplicaciones de visión estéreo. Consta de tres etapas, donde la idea principal subyacente es la aplicación sucesiva de procesos automáticos a imágenes, reproduciendo el conocimiento del experto humano. Este sistema experto es capaz para ajustar automáticamente las intensidades del par estéreo de entrada. En este planteamiento también se demuestra que la suposición de brillo constante (CIB) es, con frecuencia, errónea. Este trabajo se describe en el capítulo 6 de esta tesis y en Correal et al. (2014a).

12. Un conjunto de conclusiones y líneas de investigación futuras han sido identificadas y detalladas en el capítulo 7 de esta tesis.

13. Como se puede inferir de los párrafos anteriores, los resultados, conocimiento y descubrimientos han sido difundidos no solo mediante la liberación de código fuente, como se ha indicado anteriormente, sino también a través de publicaciones científicas; la referencia Correal et al. (2014a), que es la única de la que se tienen estadísticas de uso, ha sido visto o descargado en más de 900 ocasiones hasta la fecha actual.

8.5 Conclusiones

En esta tesis se ha presentado un estudio y análisis de las capacidades de autonomía de vehículos de exploración planetaria para las misiones actuales y futuras, de acuerdo con el objetivo 1, con especial énfasis en navegación autónoma, y con el objetivo de reducir los costes operaciones de estas misiones incrementando el nivel de autonomía a bordo.

Se ha creado un entorno de trabajo específicamente diseñado para facilitar desarrollos de autonomía, pruebas y validación funcional en el dominio de la exploración espacial robótica, de acuerdo con el objetivo 2. La justificación de la creación de dicho entorno de trabajo surge de la falta de disponibilidad y accesibilidad a entornos capaces de apoyar este tipo de investigaciones, típicamente propietarios de agencias espaciales nacionales. El que aquí se presenta está parcialmente basado en la integración de componentes, herramientas y paquetes de código abierto existentes, con las necesarias adaptaciones y extensiones.

Se ha realizado una integración intra e inter sistema de los diferentes componentes software utilizados para crear esta infraestructura, a través de la creación de envolturas e interfaces de comunicaciones. La reutilización de software nunca es una tarea trivial, como muchos autores han documentado (Garlan et al., 2009). Habitualmente suele llevar más tiempo de lo previsto y resultar en un código vago y difícil de mantener. Los principales inconvenientes cuando se trabaja con recursos de código abierto estriban en el hecho de depender de herramientas a veces no probadas o inmaduras, no soportadas por una empresa, donde la falta de documentación en muchos casos hace que el tiempo de aprendizaje sea un proceso lento y difícil. Sin embargo, en este caso, el uso de software y librerías existentes ha facilitado y acelerado en gran medida el proceso de desarrollo; la accesibilidad al código fuente ha permitido mantener un control total sobre los procesos, subsistemas e interioridades de cada módulo, así como realizar las adaptaciones y ampliaciones necesarias a las capacidades originales.

Se ha desarrollado un modelo simulado de un vehículo robótico basado en los diseños de los rovers Spirit y Opportunity de la NASA, incluyendo sus sensores y actuadores, de acuerdo con el objetivo 3. El simulador empleado, Gazebo, posee algunas limitaciones; sin embargo, aspectos tales como condiciones meteorológicas, interacciones mecánicas avanzadas, fuerzas de contacto rueda-suelo y fenómenos como deslizamiento o hundimiento no pueden ser emulados. Por otra parte, es importante enfatizar que esto no era realmente el objetivo de esta investigación, sino que ésta está centrada en el desarrollo de estrategias de navegación autónoma de alto nivel, análisis y validación de su rendimiento y funcionamiento a nivel funcional. A pesar de estas limitaciones, el simulador cumple con los requisitos necesarios para llevar a cabo estos desarrollos y validaciones.

Con el apoyo del entorno de trabajo introducido anteriormente, se ha desarrollado por completo y desde el principio un sistema de navegación autónomo, alcanzando el objetivo propuesto 4. Se ha partido de una estrategia de navegación comparable con trabajos previos de NASA, sin embargo, se han diseñado y desarrollado algunas tácticas nuevas. Una de esas estrategias que diferencian este trabajo del enfoque propuesto por NASA está relacionada con el cálculo del mapa de navegabilidad, evaluando solamente las áreas y celdas del mapa estrictamente necesarias, lo que permite acelerar el cálculo computacional. Además, aparte del cálculo de trayectorias basado en tramos rectos o arcos candidatos, se ha diseñado una estrategia de planificación de trayectorias más sofisticada basada en *splines*. Ésta maximiza y optimiza el proceso de navegación, ya que calcula la ruta segura más larga posible dentro del mapa disponible, minimizando el número de percepciones necesarias y ahorrando gran cantidad del tiempo que el rover permanece parado realizando cómputos. Esto acelera el proceso de navegación general con respecto al enfoque original de la misión MER, y en última instancia aumenta el retorno científico de la misma.

En relación con el objetivo 5, se ha diseñado una arquitectura software basada en un modelo multi-capa con el propósito de ser usada como una plataforma dinámica de investigación. Cada nivel está compuesto por un conjunto de módulos independientes y altamente configurables, donde los algoritmos pueden ser fácilmente actualizados y/o reemplazados, así como integrar con el tiempo funciones más sofisticadas y avanzadas. Esto se ha verificado cuando se realizó el portado del sistema de un entorno simulado al mundo real; el módulo de percepción visual tuvo que ser modificado, reemplazando algunos algoritmos, mientras que el resto del sistema se mantuvo inalterado, confirmando por lo tanto la utilidad y el éxito del diseño modular. La independencia de los dispositivos y hardware se ha alcanzado mediante la implementación de una capa de abstracción. Esto hace posible portar el sistema de navegación entre diferentes vehículos, simplemente adaptando esta capa de abstracción de hardware a la plataforma robótica concreta, mientras que el resto de capas y módulos permanecen inalterados.

El sistema de navegación ha sido probado tanto en simulación como en un robot móvil, de acuerdo con el objetivo 6. Una vez que los algoritmos se consideraron suficientemente maduros y validados en simulación, fueron portados a una plataforma robótica física para una campaña de pruebas de campo y validación final. Esta migración resultó ser un proceso suave gracias a las extensas campañas de prueba realizadas previamente en simulación y a la naturaleza modular de la arquitectura.

Siguiendo el objetivo 7, el análisis de los resultados ha revelado que las necesidades de recursos de computación son acordes a las disponibles en misiones previas y a las capacidades de los procesadores actuales y futuros utilizados para misiones espaciales. El comportamiento y rendimiento del sistema de navegación diseñado se ha validado de acuerdo con los resultados obtenidos de las pruebas de campo, como se esperaba del objetivo 8. Es importante resaltar que estos desarrollos no son solamente válidos para la exploración espacial sino que también lo son para aplicaciones terrestres en entornos naturales exteriores, tales como patrullaje y vigilancia, seguridad de fronteras, transporte de mercancías o apoyo a la extinción de incendios en bosques, por nombrar algunos.

El análisis de los resultados ha revelado también que el proceso de percepción, basado en visión estéreo, es uno de los más críticos y exigentes, en términos de recursos y requisitos de computación, identificando a éste como el foco de los potenciales esfuerzos de optimización, de acuerdo al objetivo 9. En el proceso de análisis del estado del arte en algoritmos de visión estéreo, se ha detectado que la amplia gama de publicaciones y la heterogeneidad de los recursos disponibles hacen de este análisis una tarea difícil y compleja. Para facilitar dicha tarea, se ha diseñado un banco de pruebas flexible con una interfaz amigable, como parte del objetivo 9. Las principales aportaciones del banco de pruebas estéreo presentado con respecto a trabajos anteriores relacionados, entre los que destaca el software de correspondencia estéreo de Middlebury (Scharstein y Szeliski, 2002), son: a) el diseño de una interfaz gráfica que permite una interacción amigable para la selección y configuración de los algoritmos y sus parámetros evitando así el uso de scripts; b) la posibilidad de realizar multitud de pruebas en muy poco tiempo; c) la inclusión de una representación gráfica de los resultados para facilitar el análisis y la comparación; d) la capacidad para usar diferentes formatos de imágenes de entrada, no limitados al conjunto de imágenes distribuidos con el banco de pruebas; e) la portabilidad, e independencia de la arquitectura y del sistema operativo, ya que el banco de pruebas se puede ejecutar en diferentes plataformas sin tener que recompilar el código fuente ni configurar un entorno de ejecución, lo que representa una de las contribuciones más importantes; f) la flexibilidad en la incorporación de algoritmos escritos en diferentes lenguajes, así como la fácil conexión con librerías de visión por computador tales como OpenCV; g) la apertura del código fuente, por lo que los usuarios pueden adaptar tanto los algoritmos como el propio banco de pruebas o su interfaz gráfica a sus necesidades concretas, e incluir nuevas características, lo que representa un aspecto crucial, ya que los algoritmos publicados pueden servir de inspiración para otros, que pueden extender o adaptar ellos mismos para lograr enfoques más robustos y optimizados con el tiempo de los que toda la comunidad pueda beneficiarse; h) la escalabilidad, ya que ha sido diseñado para evolucionar con el tiempo e incluir funcionalidades adicionales, algoritmos y procesos de imagen, recibiendo contribuciones de la comunidad investigadora a medida que se publican nuevos trabajos; una vez integrados nuevos algoritmos, éstos pueden ser fácilmente probados y comparados con otros, analizando sus fortalezas y debilidades para evaluar su contribución al estado del arte.

Se han adaptado, estandarizado e integrado en el banco de pruebas un conjunto de algoritmos heterogéneos de correspondencia estéreo y procesado de imágenes, haciendo uso de la gran colección de recursos existentes y de la labor ya realizada por la comunidad investigadora. Este proceso de integración y normalización no ha requerido un gran esfuerzo, ya que el banco de pruebas es muy flexible y permite la inclusión de algoritmos escritos en diferentes lenguajes de programación. Se ha analizado este conjunto de algoritmos estéreo, ya que el banco de pruebas permite probar muchos algoritmos variando sus parámetros en muy poco tiempo, justamente hasta que se identifica el algoritmo más apropiado para cada aplicación concreta. Esto evita el tener que implementar algoritmos sólo para las pruebas, lo que supone un considerable ahorro de tiempo y esfuerzo, demostrando la utilidad de dicho banco.

La investigación realizada está motivada por la evidencia del funcionamiento desfavorable de diversos algoritmos de correspondencia con imágenes reales pertenecientes a entornos de exteriores no controlados ni estructurados. Como resultado, se ha comprobado que cuando se combinan varios procesos y se ejecutan de forma consecutiva, se obtiene una mejora significativa a la hora de calcular correspondencias estereoscópicas. Por lo tanto, siguiendo el objetivo 10, se ha propuesto un sistema experto para la corrección automática de imágenes y reconstrucción de terrenos en aplicaciones de visión estéreo. Dicho sistema ajusta las intensidades de las imágenes mediante la aplicación sucesiva de una combinación de técnicas y filtros en tres etapas consecutivas, emulando el conocimiento del experto. Entre estas técnicas se incluyen tratamientos previos a las imágenes de entrada y técnicas de filtrado posterior a la fase de búsqueda de correspondencias.

Una de esas técnicas identificadas es la correspondencia de histogramas, utilizada para ajustar la intensidad de una de las imágenes del par en función de la otra, haciendo coincidir automáticamente sus histogramas. Otra de las técnicas que se ha aplicado ha sido el filtrado homomórfico, que elimina la componente de iluminación de las imágenes estereoscópicas de entrada, causando una mejora en el mapa de disparidad. Sin embargo, el filtrado homomórfico introduce algunos errores; éstos pueden ser fácilmente filtrados a través de un proceso posterior basado en agrupación siguiendo el principio de continuidad espacial, que elimina falsos positivos. Ambos procesos aplicados en conjunto representan un gran avance y una mejora significativa en la eficiencia del proceso de correspondencia de imágenes reales de terrenos, asumiendo un pequeño coste computacional adicional con respecto a la aplicación únicamente del proceso de correspondencia, aunque no supone una carga significativa. Una vez que las intensidades han sido corregidas, el proceso estéreo es capaz de obtener un mayor número de correspondencias, además de reducir errores, lo que supone un gran impacto en los posteriores procesos de re-proyección 3D y reconstrucción del terreno. Esto también prueba que la suposición de brillo constante en las imágenes (CIB) es a menudo incorrecta.

El sistema experto propuesto puede extenderse para aplicarse en otras áreas tales como segmentación y reconocimiento de objetos 3D. Este sistema experto se ha diseñado con una arquitectura abierta, de forma que en el futuro sea posible sustituir o añadir nuevos módulos; siendo esto de especial interés para el estudio de diferentes algoritmos de correspondencia estéreo, o para agregar una base de conocimiento para mejorar la corrección de imágenes basada en el conocimiento acumulado.

8.6 Trabajos Futuros

Todavía hay espacio para mejoras en los subsistemas, métodos y herramientas propuestas. Además, existen muchas líneas de investigación abiertas dentro del contexto del trabajo asociado con esta tesis. A continuación se indican algunas de tales líneas de investigación futuras relacionadas más prometedoras.

En relación al entorno de soporte, el Centro de Control puede extenderse para permitir a los operadores crear planes de actividades que puedan ser enviados al vehículo, especificando el uso de la instrumentación científica, y modelando las limitaciones y restricciones en las comunicaciones para emular el ciclo real interplanetario. Se pueden incluir capacidades de reconstrucción 3D a partir de imágenes del rover para facilitar a los operadores/investigadores el diseño de estos planes, así como incorporar capacidades de automatización de pruebas con el propósito de depuración. En cuanto a la simulación, los modelos de terrenos y rovers se pueden sofisticar mediante la inclusión de fenómenos tales como arena, deslizamiento y hundimiento de ruedas, modelos de instrumentos científicos que el robot debe utilizar para realizar experimentos en el planeta y modelos de uso de energía, lo que tiene un impacto importante en la estrategia global de autonomía. Los actuales modelos de dispositivos, sensores y actuadores se pueden extender con modelos de ruido y error para mejorar la gestión de la incertidumbre por parte de los algoritmos.

Se pueden diseñar nuevos y más sofisticados algoritmos de visión estéreo para el manejo de sombras, polvo ambiental y sol deslumbrante, así como técnicas de odometría visual para corregir errores de localización acumulados por la odometría mecánica. Se puede dotar al rover con capacidades de planificación de tareas a bordo mediante la incorporación de planificadores de inteligencia artificial, que tengan en cuenta aspectos tales como los recursos del rover, el consumo de energía, uso de instrumentos, tasa de carga de los paneles solares o la posición del sol en el diseño de los planes de actividades y de contingencia.

En cuanto al proyecto de código abierto del banco de pruebas de visión estéreo, éste se puede extender para incluir nuevos algoritmos de correspondencia y procesos de imagen, así como funcionalidad tal como la capacidad de automatización de pruebas, programando una serie de pruebas para ser ejecutadas sin supervisión durante períodos de baja actividad o por la noche; de este modo se puede reunir una gran cantidad de datos procedentes de diferentes experimentos para su posterior análisis, almacenándolos automáticamente y presentándolos de un modo estructurado de forma que facilite su análisis. Este análisis puede ser también hasta cierto punto automatizado, elaborando informes con evaluaciones, comparaciones y conclusiones obtenidas a partir de los datos recogidos. Ya se han recibido algunas contribuciones de varios autores que se integrarán para la próxima versión del banco de pruebas.

Se pueden realizar trabajos de investigación para mejorar los actuales filtros y procesos de imagen, o diseñar nuevas técnicas para la corrección automática de imágenes. Se pueden realizar mejoras en el filtro basado en agrupación para tener en cuenta los valores de disparidad de los píxeles que forman un determinado grupo en comparación con otros grupos cercanos en función de la distancia que los separa, de tal forma que se pueda decidir automáticamente si un grupo ha de ser mantenido o eliminado. En esta línea es posible analizar el uso de técnicas tales como *simulated annealing* para eliminar errores excepcionales que pudieran darse en píxeles aislados con posterioridad al cómputo de disparidades y filtrado.

Una desventaja actual del enfoque propuesto para el sistema experto es precisamente la falta de conocimiento del sistema en cuanto a la forma de realizar el proceso de correspondencia de histogramas de la manera más eficaz. Puede ocurrir que al emparejar cada canal RGB de las imágenes de entrada por separado se obtengan resultados no tan buenos como podrían derivarse si el proceso de correspondencia de histogramas se hubiera realizado sobre las imágenes convertidas previamente a escala de grises. En este sentido, se perfila como trabajo futuro la capacidad de seleccionar automáticamente el proceso más adecuado configurando óptimamente sus parámetros en función de la aplicación concreta y las condiciones operativas, asegurando la máxima eficacia y adaptación del sistema.

Como conclusión final, se puede decir que actualmente están siendo programadas futuras misiones a Marte, otros planetas y cuerpos celestes en el sistema solar. Una vez que se ha desarrollado una primera aproximación a un sistema de navegación autónomo, una serie de algoritmos y técnicas y la infraestructura básica necesaria, que es el foco del trabajo de esta tesis, se pueden analizar, diseñar y desarrollar algoritmos y estrategias de navegación más sofisticadas con el apoyo de las herramientas creadas a tal efecto, que permitan diseñar futuros rovers con mayor autonomía, robustos y fiables, que permitan avanzar el estado del arte en este dominio y contribuir en la medida de lo posible a llevar a la humanidad al siguiente nivel en la exploración planetaria robótica.

REFERENCES

Abbeloos, W. (2010). Stereo Matching (available on-line at http://www.mathworks.com/matlabcentral/fileexchange/28522-stereo-matching).

Acharya, T.; Ray, A., (2005). Image Processing: Principles and Applications. Wiley Interscience.

Aghazarian, H.; Pirjanian, P.; Schenker, P.; Huntsberger T. (2004). An Architecture for Controlling Multiple Robots. NASA Tech Briefs. NPO-30345.

Alagoz, B. (2008). Obtaining Depth Maps from Color Images By Region Based Stereo Matching Algorithms. OncuBilim Algorithm and Systems Labs, vol. 8, no. 4, pp. 1-13.

Angelova, A.; Matthies, L.; Helmick, D.; Perona, P. (2007). Learning and prediction of slip from visual information. Journal of Field Robotics, vol. 24, issue 3, pp 205–231.

Arkin, R.C. (1989). Motor schema based mobile robot navigation. International Journal of Robotics Research, vol. 4(8), pp 92-112.

Ayache, N.; Faverjon, B. (1987). Efficient Registration of Stereo Images by Matching Graph Descriptions of Edge Segments. International Journal of Computer Vision, vol. 1, pp 107-131.

Bakambu, J. N.; Allard, P.; Dupuis, E. (2006). 3D Terrain Modeling for Rover Localization and Navigation. In proceeding of the 3rd Canadian Conference on Computer and Robot Vision, p. 61.

Baker, H.H. (1982). Building and Using Scene Representations in Image Understanding. AGARD-LS-185. Machine Perception, 3.1-3.11.

Barnard, S.; Fishler, M. (1982). Computational stereo. ACM Computing Surveys, vol. 14, pp. 553–572.

Barnea, D.I.; Silverman, H.F. (1972). A Class of algorithms for fast digital Image registration. IEEE Trans. Computers, vol. 21, pp. 179-186.

Beegle, L.; Wilson, M.; Abilleira, F.; Jordan, J.; Wilson, G. (2007). A Concept for NASA's Mars 2016 Astrobiology Field Laboratory. ASTROBIOLOGY, vol. 7, number 4, 2007. © Mary Ann Liebert, Inc. DOI: 10.1089/ast.2007.0153.

Bhatti, A. (2012). Current Advancements in Stereo Vision. InTech.

Biesiadecki, J.; Leger, C.; Maimone, M. (2005). Tradeoffs between directed and autonomous driving on the Mars Exploration Rovers. International Symposium of Robotics Research, vol. 26 no. 1, pp. 91-104.

Biesiadecki, J.; Maimone, M. (2006). The Mars Exploration Rover surface mobility flight software: Driving ambition. Proceeding of the IEEE Aerospace Conference, Big Sky, MT, vol. 5, p. 15.

Blanco, J. L. (2010). Development of Scientific Applications with the Mobile Robot Programming Toolkit. The MRPT reference book, Spain.

Bolles, R. C.; Baker, H. H.; Hannah, M. J. (1993). The JISCT Stereo Evaluation. ARPA Image Understanding Workshop, pp. 263-274.

Borenstein, J.; Feng, L. (1995). Measurement and Correction of Systematic Odometry Errors in Mobile Robots. IEEE Journal of Robotics and Automation, May 1995, vol. 12, issue 6, pp. 869-880.

Borrelly et al. (1998). The ORCCAD Architecture. International Journal of Robotics Research, vol. 17 no. 4, pp. 338-359.

Bradski, G.; Kaehler, A. (2008). Learning OpenCV. Computer Vision with the OpenCV Library. O'Reilly Media

Brooks, R.A. (1986). A robust layered control system for a mobile robot. IEEE Transactions on Robotics and Automation, vol. 2, number 1, pp. 14-23.

Butterworth, S. (1930). On the Theory of Filter Amplifiers. In Wireless Engineer (also called Experimental Wireless and the Wireless Engineer). Vol. 7, pp. 536–541.

Cech, J.; Sara, R. (2007). Efficient Sampling of Disparity Space for Fast and Accurate Matching. In Proc. BenCOS Workshop of IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), pp. 1-8.

Charmeau, M. C.; Bensana, E. (2005). AGATA, A Lab Bench Project for Spacecraft Autonomy. i-SAIRAS 2005 - The 8th International Symposium on Artificial Intelligence, Robotics and Automation in Space, Vol.- No. 603.

CMLabs Simulations. (2003). Vortex Developer Guide: A Manual for the Vortex Simulation Toolkit. Montreal, Canada.

CNES (2014). Centre national d'études spatiales (available online at http://www.cnes.fr).

Cochran, S. D.; Medioni, G. (1992). 3-D surface description from binocular stereo. IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 14, no. 0, pp. 981–994.

Cogmation Robotics Inc. (2010). RobotSim documentation (available online at http://www.cogmation.com/pdf/robotsim/doc.pdf). Manitoba, Canada.

Correal, R.; Pajares, G. (2010). Framework for Simulation and Rover' Visual-Based Autonomous Navigation in Natural Terrains. 7th Workshop RoboCity2030-II, Madrid, Spain.

Correal, R.; Pajares, G. (2011a). Modeling, simulation and onboard autonomy software for robotic exploration on planetary environments. International Conference on DAta Systems In Aerospace (DASIA), Malta, pp. 1-21.

Correal, R.; Pajares, G. (2011b). Onboard Autonomous Navigation Architecture for a Planetary Surface Exploration Rover and Functional Validation Using Open-Source Tools. ESA Intl. Conf. on Advanced Space Technologies in Robotics and Automation (ASTRA 2011), ESA/ESTEC, Noordwijk, The Netherlands, pp. 1-8.

Correal, R. (2012) Matlab Central (available on-line at http://www.mathworks.com/matlabcentral/fileexchange/36433)

Correal, R.; Pajares, G.; Ruz, J. J. (2013). Mejora del Proceso de Correspondencia en Imágenes Estereoscópicas Mediante Filtrado Homomórfico y Agrupaciones de Disparidad. Revista Iberoamericana de Automática e Informática Industrial, vol. 10, issue 2, pp 178-184.

Correal, R.; Pajares, G.; Ruz, J.J. (2014a). Automatic Expert System for 3D Terrain Reconstruction Based on Stereo Vision and Histogram Matching. Expert Systems with Applications, no. 41, pp. 2043-2051.

Correal, R.; Pajares, G.; Ruz, J.J. (2014b). Autonomy for Ground-level Robotic Space Exploration: Framework, Simulation, Architecture, Algorithms and Experiments. ROBOTICA Journal. June 2014, pp. 1–32.

Cox, I. J.; Hingorani, S. L. (1995). Dynamic histogram warping of image pairs for constant image brightness. In IEEE Proc. Int. Conf. on Image Processing Proceedings, vols I-III, pp. B366-B369, IEEE Computer Soc Press.

Crisp, J. A.; Adler, M.; Matijevic, J. R.; Squyres, S. W.; Arvidson, R. E.; Kass, D. M. (2003). Mars Exploration Rover Mission. Journal of Geophysical Research, vol. 108, issue E12.

Cruz, J.M., Pajares, G., Aranda, J. (1995a). A neural network model in stereovision matching. Neural Networks, vol. 8, no. 5, pp 805-813.

Cruz, J.M.; Pajares, G.; Aranda, J.; Vindel, J.L. (1995b). Stereo matching technique based on the perceptron criterion function. Pattern Recognition Letters, vol. 16, pp. 933-944.

Darabiha, A.; MacLean, W. J.; Rose, J. (2006). Reconfigurable hardware implementation of a phase-correlation stereoalgorithm. Machine Vision and Applications. May 2006, Vol. 17, Issue 2, pp 116-132.

Edwards, L.; Fluckiger, L.; Nguyen, L.; Washington, R. (2001). VIPER: Virtual Intelligent Planetary Exploration Rover. Int'l Symposium on Artificial Intelligence and Robotics & Automation in Space (i-SAIRAS), Quebec, Canada.

ESA (2014). European Space Agency (available online at http://www.esa.int/ESA).

Estlin, T. et al. (1999a). Using continuous planning techniques to coordinate multiple rovers. Proceedings of the IJCAI Workshop, Sweden, pp 4-45.

Estlin, T., et al. (1999b). An Integrated Architecture for Cooperation Rovers. Symposium on AI Robotics Automation Space Noordwijk, The Netherlands, pp. 255-262.

Estlin, T. et al. (2007). Increased Mars Rover Autonomy using AI Planning, Scheduling and Execution. IEEE International Conference on Robotics and Automation. Roma, Italy, pp 4911–4918.

Faugeras, O.; Keriven, R. (1998). Variational Principles, Surface Evolution, PDE's, Level Set Methods and the Stereo Problem. IEEE Transactions on Image Processing, vol. 7 (3), pp. 336-344.

Firby, R. (1989). Adaptive Execution in Complex Dynamic Worlds. PhD thesis, Yale University, Department of Computer Science.

Fisher, F. et al. (1997). An automated deep space communications station. Proceedings IEEE Aerospace Conference, Colorado, vol. 3, pp. 153-162.

Garlan, D.; Allen, R.; Ockerbloom, J. (2009). Architectural Mismatch: Why Reuse is Still So Hard. IEEE Software, vol. 26, no. 4, 66-69.

Geiger, A.; Lenz, P.; Stiller, C.; Urtasun, R. (2014). The KITTI Vision Benchmark Suite. (Available online at http://www.cvlibs.net/datasets/kitti/eval_stereo_flow.php?benchmark=stereo).

Gerkey, B.; Vaughan, R.; Howard, A. (2003). The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In Proceedings of the International Conference on Advanced Robotics, Coimbra, Portugal, pp. 317-323.

Ghallab, M.; Ingrand, F.; Lemai, S.; Py, F. (2001). Architecture and Tools for Autonomy in Space. 6th International Symposium on Artificial Intelligence and Robotics & Automation in Space, Quebec.

Glette, K. (2004). Motion Control for a Planetary Exploration Rover with Six Steerable Wheels. MSc Thesis. Norwegian University of Science and Technology.

Goldberg, S.; Maimone, M.; Matthies, L. (2002). Stereo Vision and Rover Navigation Software for Planetary Exploration. IEEE Aerospace Conference, Big Sky, Montana (USA), vol. 5, pp. 2025–2036.

Gonzalez, R. C.; Wintz, P. (1987). Digital Image Processing, Addison-Wesly, Reading, MA.

Gonzalez, R. C.; Woods, R.E. (2008). Digital Image Processing, Prentice-Hall, Englewood Cliffs, NJ.

Grimson, W.E.L. (1985). Computational Experiments with a Feature-based Stereo Algorithm. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 7, pp. 17-34.

Grotzinger, J. P. et al. (2012). Mars Science Laboratory Mission and Science Investigation. Space Science Reviews, vol. 170, issue 1-4, 5-56.

Hailan, G.; Wenzhe, L. (2012). A Modified Homomorphic Filter for Image Enhancement. In proceedings of the International Conference on Computer Application and System Modeling, pp 176-180.

Helmick D.; Angelova A.; Matthies, L. (2009). Terrain adaptive navigation for planetary rovers. Journal of Field Robotics, Vol. 26, issue 4, pp 391-410.

Henning, M.; Spruiell, M. (2010). Distributed Programming with Ice. ZeroC, Inc.

Hirschmüller, H. (2005). Accurate and Efficient Stereo Processing by Semi-Global Matching and Mutual Information. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), vol. 2, pp. 807-818.

Hirschmüller, H.; Scharstein, D. (2007). Evaluation of cost functions for stereo matching. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007), pp. 1-8.

Hugues, L.; Bredeche, N. (2006). Simbad : an Autonomous Robot Simulation Package for Education and Research. International Conference on Simulation of Adaptive Behavior, Rome, Italy, pp. 831-842.

Huntsberger, T.; Kubota, T.; Rose, J. (1998). Integrated Vision/control System for Autonomous Planetary Rovers. IAPR Workshop on Machine Vision Applications. Japan, pp. 34-37.

Iagnemma, K.; Ward, C. (2009). Classification-Based Wheel Slip Detection and Detector Fusion for Mobile Robots on Outdoor Terrain, Autonomous Robots, Vol. 26, pp. 33-46.

Iagnemma, K.; Senatore, C.; Trease, B.; Arvidson, R.; Shaw, A.; Zhou, F.; Van Dyke, L, and Lindemann, R. (2011). Terramechanics Modeling of Mars Surface Exploration Rovers for Simulation and Parameter Estimation. Proceedings of the ASME International Design Engineering Technical Conference, pp. 805-812.

Ingrand, F.; Lacroix, S.; Lemai, S.; Py, F. (2007). Decisional autonomy of planetary rovers. Journal of Field Robotics, vol. 24, issue 7, pp. 559–580.

ISRO (2014). Indian Space Research Organisation (available online at http://www.isro.org).

Jain, A.; Guineau, J.; Lim, C.; Lincoln, W.; Pomerantz, M.; Sohl, G.; Steele, R. (2003). ROAMS: Planetary Surface Rover Simulation Environment. Int'l Symposium on Artificial Intelligence and Robotics & Automation in Space (i-SAIRAS), Nara, Japan, vol.2, pp. 861-876.

JAXA (2014). Japanese Space Agency (available online at http://global.jaxa.jp).

Jensen, J.R. (1982). Introductory Digital Image Processing, Prentice-Hall, Englewood Cliffs, NJ

Joudrier, L.; Elfving, A. (2009). Challenges of the ExoMars Rover Control. American Institute of Aeronautics and Astronautics, AIAA 2009-1807, Seattle, Washington (USA), pp. 109-120.

Kang, Y.; Ho, Y. (2012). Efficient Stereo Image Rectification Method Using Horizontal Baseline. Advances in Image and Video Technology. Lecture Notes in Computer Science, vol. 7087, pp. 301-310.

Kawai, Y.; Tomita, F. (1998). Intensity calibration for stereo images based on segment correspondence. IAPR Workshop on Machine Vision Applications, Makuhari, Chiba, Japan, pp. 331-334.

Koenig, N.; Howard, A. (2004). Design and Use Paradigms for Gazebo, an Open-Source Multi-Robot Simulator. In Proc. of the International Conference on Intelligent Robots and Systems, Sendai, Japan, vol. 3, pp. 2149-2154.

Konolige, K. et. al. (1997a). The saphira architecture: A design for autonomy. Journal of Experimental and Theoretical AI, vol. 9(1), pp. 215-235.

Konolige, K. (1997b). Small vision system: Hardware and implementation. In Proceedings of the International Symposium on Robotics Research, Hayama, Japan, pp. 111–116.

Kosov, S.; Thormaehlen, T.; Seidel, H.P. (2009). Accurate Real-Time Disparity Estimation with Variational Methods. International Symposium on Visual Computing, pp. 796-807.

Krotkov, E.P. (1989). Active Computer Vision by Cooperative Focus and Stereo. Springer, New York.

Kurien, J.A.; Nayak, P.; Williams, B. (1998). Model-Based Autonomy for Robust Mars Operations. Founding Convention of the Mars Society, pp. 421-428.

Laia, Y.; Chunga, K.; Chena, C.; Lina, G.; Wangb, C. (2012). Novel mean-shift based histogram equalization using textured regions. Expert Systems with Applications, vol. 39, issue 3, pages 2750–2758.

Lankton, S. (2007). 3D Vision with Stereo Disparity (available on-line at http://www.shawnlankton.com/2007/12/3d-vision-with-stereo-disparity).

Leger, C. (1999). Automated Synthesis and Optimization of Robot Configurations: An Evolutionary Approach. PhD thesis. The Robotics Institute, Carnegie Mellon University.

Leger, C.; Trebi-Ollennu, A.; Wright, J. et al (2005). Mars Exploration Rover surface operations: Driving Spirit at Gusev Crater. In proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, Waikoloa, HI, pp. 1815-1822.

Liling, Z.; Yuhui, Z.; Quansen, S.; Deshen, X. (2012). Suppression for Luminance Difference of Stereo Image-Pair Based on Improved Histogram Equalization. In proceedings of the Computer Science and Technology, vol. 6, pp. 107-118.

Lim, Y. K.; Kleeman, L.; Drummond, T. (2012). Algorithmic methodologies for FPGA-based vision. Machine Vision and Applications, vol. 24, issue 6, pp. 1197-1211.

Lin, L.; Zhou, W. (2009). Interested Sample Point Pre-Selection Based Dense Terrain Reconstruction for Autonomous Navigation. Third International Symposium on Intelligent Information Technology Application, vol. 3, pp. 339-343.

Lucas, G.W. (2000). A Tutorial and Elementary Trajectory Model for the Differential Steering System of Robot Wheel Actuators. The Rossum Project (available online at http://rossum.sourceforge.net/papers/DiffSteer/DiffSteer.html)

Magid, E.; Keren, D.; Rivlin, E.; Yavneh, I. (2006). Spline-Based Robot Navigation. Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, Beijing, China, pp. 2296-2301

Maimone, M.; Biesiadecki, J.; Tunstel, E.; Cheng, Y.; Leger. C. (2006a). Surface navigation and mobility intelligence on the Mars Exploration Rovers. Intelligence for Space Robotics, Chapter 3, TSI Press.

Maimone, M.; Johnson, A.; Cheng, Y.; Willson, R.; Matthies, L. (2006b). Autonomous Navigation Results from the Mars Exploration Rover (MER) Mission. Springer Tracts in Advanced Robotics, vol. 21, pp. 3–13.

Maimone, M.; Leger, C.; Biesiadecki, J. (2007). Overview of the Mars Exploration Rovers' Autonomous Mobility and Vision Capabilities. IEEE International Conference on Robotics and Automation (ICRA) Space Robotics Workshop, Roma, Italy.

Martin, A.; Peuter, W.; Putz, P. (1994). A Unified Control Architecture for Planetary Rovers. I-SAIRAS 94, 3rd International Symposium on Artificial Intelligence, Robotics and Automation for Space. California.

Martin-Alvarez, A. (1999). Advanced Design and Implementation of a Control Architecture for Long Range Autonomous Planetary Rover. International Symposium on Artificial Intelligence, Robotics and Automation in Space. Noordwijk, The Netherlands.

Mataric, M.J. (2007). The Robotics Primer. The MIT Press.

Matthies, L. et al. (2007). Computer Vision on Mars. International Journal of Computer Vision, vol. 75, issue 1, pp. 67-92.

Maurette, M.; Rastel, L. (2002). Planetary rover simulation and operation. ESA Workshop on Advanced Space Technologies for Robotics and Automation (ASTRA), ESA/ESTEC, Noordwijk, The Netherlands.

McMillan, S. (2003). DynaMechs: A multibody dynamic simulation library. Available online at http://dynamechs.sourceforge.net/.

Michel, O. (2004). Webots: Professional Mobile Robot Simulation. International Journal of Advanced Robotic Systems, Vol. 1, Num. 1, pages 39-42.

Mishkin, A. H.; Morrison, J. C.; Nguyen, T. T.; Stone, H. W.; Cooper, B. K.; Wilcox, B. H. (1998). Experiences with operations and autonomy of the Mars Pathfinder microrover. In Proceedings of the IEEE Aerospace Conference, Snowmass, Colorado (USA), vol. 2, pp. 337-351.

Montemerlo, M.; Roy, N.; Thrun, S. (2003). CARMEN: Carnegie Mellon Robot Navigation Toolkit.

Morisset, B. et al. (2009). Leaving flatland: toward real-time 3D navigation. In proceeding of the IEEE International Conference on Robotics and Automation, pp 3384-3391.

Muscettola, N.; Dorais, G.; Fry, C.; Levinson, R.; Plaunt, C. (2002). IDEA: Planning at the Core of Autonomous Reactive Agents. 3rd International NASA Workshop on Planning and Scheduling for Space.

Nalpantidis, L.; Gasteratos, A. (2010). Stereo vision for robotic applications in the presence of non-ideal lighting conditions. Image and Vision Computing, vol. 28, pp. 940–951.

NASA (2014). National Aeronautics and Space Administration (available online at http://www.nasa.gov).

National Instruments. (2013). NI Vision 2013 Concepts Help (available on-line at http://zone.ni.com/reference/en-XX/help/372916P-01/nivisionconceptsdita/guid-c9c3535b-faf7-4ade-9166-513a49d1b90a/).

Nesnas, I.; Wright, A.; Bajracharya, M.; Simmons, R.; Estlin, T.; Kim, W.S. (2003). CLARAty: An Architecture for Reusable Robotic Software. SPIE Aerosense Conference, Florida, Vol. 5083, pp. 253-264.

Neveu, C.; Shirley, M. (2003). LiveInventor: An interactive development environment for robot autonomy. International Symposium on Artificial Intelligence and Robotics & Automation in Space (i-SAIRAS), Japan.

Odwyer, A.; Correal, R. (2008). Experiences in Producing a Preliminary Navigation OBSW Prototype for the Exomars Rover Based on EDRES. ESA Workshop on Advanced Space Technologies for Robotics and Automation (ASTRA 2008), ESA/ESTEC, Noordwijk, The Netherlands.

Ogale, A. S.; Aloimonos, Y. (2005). Shape and the Stereo Correspondence Problem. Intl. Journal of Computer Vision, Vol. 65, no. 3, pp. 147-162.

Ogale, A.S.; Aloimonos, Y. (2007). A Roadmap to the Integration of Early Visual Modules, Intl. Journal. of Computer Vision. Vol. 72, no. 1, pp. 9-25.

Olson, C.; Matthies, L.; Schoppers, M.; Maimone, M. (2000). Robust stereo ego-motion for long distance navigation. International Conference on Computer Vision and Pattern Recognition, Hilton Head, SC, vol. 2, 453 – 458.

Ozanian, T. (1995). Approaches for Stereo Matching - A Review. Modeling Identification Control, vol. 16 (2), pp. 65-94.

Pajares, G.; Cruz, J.M.; Aranda, J. (1998a). Stereo matching based on the self-organizing feature-mapping algorithm. Pattern Recognition Letters, vol. 19, pp. 319-330.

Pajares, G.; Cruz, J. M.; López-Orozco, J.A. (1998b). Improving Stereovision Matching through supervised learning. Pattern Analysis and Applications, vol. 1, pp. 105-120.

Pajares, G.; Cruz, J.M.; Aranda, J. (1998c). Relaxation by Hopfield Network in Stereo Image Matching. Pattern Recognition, vol. 31, nº 5, pp. 561-574.

Pajares G.; Cruz, J. M. (1999). Stereo Matching using Hebbian learning. IEEE Transactions on Systems Man and Cybernetics, Part B: Cybernetics, vol. 29, nº 4, pp. 553-559.

Pajares, G.; Cruz, J. M.; López-Orozco, J.A. (2000a). Relaxation Labeling in Stereo Image Matching. Pattern Recognition, vol. 33, pp. 53-68.

Pajares G.; Cruz, J. M. (2000b). A new learning strategy for stereo matching derived from a fuzzy clustering method. Fuzzy Sets and Systems, vol. 110, nº 3, pp. 413-427.

Pajares, G.; Cruz, J. M. (2001). Local stereovision matching through the ADALINE neural network. Pattern Recognition Letters, vol. 22, nº 14, pp. 1457-1473.

Pajares, G.; Cruz, J. M. (2002a). A Probabilistic Neural Network for Feature Selection in Stereovision Matching. Neural Computing and Applications, vol. 11, nº 2, pp. 83-89.

Pajares, G.; Cruz, J. M. (2002b). The non-Parametric Parzen's window in stereovision matching. IEEE Transactions on Systems Man and Cybernetics, Part B: Cybernetics, vol. 32, nº 2, pp. 225-230.

Pajares, G.; Cruz, J. M. (2003). Stereovision matching through Support Vector Machines. Pattern Recognition Letters, 24(15), pp. 2575-2583.

Pajares, G.; Cruz, J. M, (2004). On combining support vector machines and simulated annealing in stereovision matching. IEEE Trans. Systems Man and Cybernetics, Part B, vol. 34(4), pp. 1646-1657.

Pajares, G.; Cruz, J.M. (2006). Fuzzy Cognitive Maps for Stereovision Matching. Pattern Recognition, vol. 39, pp. 2101-2114.

Pajares, G.; de la Cruz, J. M. (2007). Visión Por Computador: Imágenes Digitales Y Aplicaciones. Editorial Ra-ma. Ch. 4, pp. 102–105.

Papadimitriou, D. V.; Dennis, T. J. (1996). Epipolar line estimation and rectification for stereo image pairs. IEEE Transactions on Image Processing, vol. 5, issue 4, pp. 672–676.

Pardo-Castellote, G. et.al, (1998). Controlshell: A software architecture for complex electromechanical systems. International Journal of Robotic Research, vol. 17, no. 4, pp. 360-380.

Parker, L. (1995). Alliance: An architecture for fault tolerant multi-robot cooperation. ORNL TM12920, Oak Ridge National Laboratory, Oak Ridge, TN, vol. 14, pp. 220-240.

Pirjanian, P. et al. (2000). CAMPOUT: A control architecture for multi-robot planetary outposts. Proceedings of the SPIE Symposium on Sensor Fusion and Decentralized Control in Robotic Systems III, pp. 221-230.

Pirjanian, P.; Huntsberger, T.; Schenker, P. (2001). Development of CAMPOUT and its further applications to planetary rover operations: A multirobot control architecture. Proc. SPIE Sensor Fusion and Decentralized Control in Robotic Systems IV, pp. 108-119.

Ponomarev, V.; Pogrebnyak, O. (1995). Image enhancement by homomorphic filters. In proceedings of the Conference on Applications of Digital Image Processing, San Diego, CA, Vol. 2564.

Poulakis, P.; Joudrier, L.; Wailliez, S.; Kapellos, K. (2008). 3DROV: A Planetary Rover System Design, Simulation and Verification Tool. Int'l Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS), Los Angeles, USA.

Psarakis, E. Z.; Evangelidis, G. D. (2005). An enhanced correlation-based method for stereo correspondence with subpixel accuracy. IEEE Int. Conf. on Computer Vision, vol. 1, pp. 907–912.

Quigley, M.; Conley, K.; Gerkey, B. (2009). ROS: An open-source robot operating system. Open-Source Software Workshop, IEEE ICRA.

Roscosmos (2014). Russian Federal Space Agency (available online at http://www.roscosmos.ru).

Scharstein, G.; Szeliski, R. (2002). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. Intl. Journal of Computer Vision. Vol. 47, pp. 7–42.

Scharstein, D.; Blasiak, A. (2014) Middlebury stereo evaluation site. (Available online at http://vision.middlebury.edu/stereo/eval).

Shirai, Y. (1987). Three-dimensional Computer Vision. Springer-Verlag, Berlin.

Simmons, R.; Apfelbaum, D. (1998). A Task Description Language for Robot Control. IEEE/RSJ Intelligent Robotics and Systems. Conf. Canada, vol. 3, 1931–1937.

Simon, D.; Arias, S.; Pissard-Gibollet, R. (2006). Orccad, a framework for safe control design and implementation. CAR'06, 1st National Workshop on Control Architectures of Robots: software approaches and issues. Montpellier, France

Smith, P.H. et al. (2008). Introduction to special section on the Phoenix Mission: Landing Site Characterization Experiments, Mission Overviews, and Expected Science. Journal of Geophysical Research, vol. 113, E00A18.

Smith, W.; Melanz, D.; Senatore, C.; Iagnemma, K.; Peng, H. (2013). Comparison of DEM and Traditional Modeling Methods for Simulating Steady-State Wheel-Terrain Interaction for Small Vehicles. 7th Americas Regional Conference of the ISTVS, Tampa, FL, USA.

Song, W. et al. (2012). Intuitive Terrain Reconstruction Using Height Observation-Based Ground Segmentation and 3D Object Boundary Estimation. Sensors Journal, Vol. 12, pp. 17186-17207.

Staranowicz, A.; Mariottini, G. L. (2011). A survey and comparison of commercial and open-source robotic simulator software. International Conference on PErvasive Technologies Related to Assistive Environments (PETRA), Crete, Greece.

Stone, H. W. (1996). Mars Pathfinder Microrover, A Small, Low-Cost, Low-Power Spacecraft. AIAA Forum on Advanced Developments in Space Robotics.

Thueer, T.; Krebs, A.; Siegwart, R.; Lamon, P. (2007). Performance comparison of rough-terrain robots—simulation and hardware. Journal of Field Robotics, vol. 24(3), pp. 251-271.

Timothy, D. (2010). MATLAB Primer, Eighth Edition. CRC Press. Print ISBN: 978-1-4398-2862-5.

Tombari, F.; Gori, F.; Di Stefano, L. (2011). Evaluation of Stereo Algorithms for 3D Object Recognition. IEEE Intl. Conf. of Computer Vision, pp. 990-997.

Tsai, R. (1986). An Efficient and Accurate Camera Calibration Technique for 3D Machine Vision. Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Miami Beach, FL, pp. 364-374.

Visentin, G. (2007). Autonomy in ESA Planetary Robotics Missions. Technical report, ESA, Noordwijk, The Netherlands.

Volpe, R.; Nesnas, I.; Estlin, T.; Mutz, D.; Petras, R.; Das, H. (2001). The CLARAty Architecture for Robotic Autonomy. In Proceedings of IEEE Aerospace Conference, Big Sky, Montana (USA), vol. 1, pp. 121-132.

Ward, C.; and Iagnemma, K. (2008). A Dynamic Model-Based Wheel Slip Detector for Mobile Robots on Outdoor Terrain, IEEE Transactions on Robotics, Vol. 24, No. 4, pp. 821-831.

Washington, R.; Golden, K.; Bresina, J. (1999). Plan Execution Monitoring and Adaptation for Planetary Rovers. Electronic Transactions on Artificial Intelligence, vol. 4, pp. 3-21.

Werger, B.; Mataric, M. (2001). From Insect to Internet: Situated Control for Networked Robot Teams. Annals of Mathematics and AI, vol. 31, issue 1-4, pp. 173-197.

Wolberg, G. (1988). Cubic Spline Interpolation: A Review. Technical Report CUCS-389-88. Department of Computer Science Columbia University New York.

Xing-zhe, X. et al. (2010). 3D Terrain Reconstruction for Patrol Robot Using Point Grey Research Stereo Vision Cameras. International Conference on Artificial Intelligence and Computational Intelligence (AICI), Vol. 1, pp. 47-51.

Yanco H.A.; Drury J. (2004). Classifying human-robot interaction: an updated taxonomy. IEEE International Conference Systems, Man and Cybernetics, vol. 3, pp. 2841–2846.

Yen, J.; Jain, A.; Balaram, J. (1999). ROAMS: Rover Analysis, Modeling and Simulation Software. Int'l. Symposium on Artificial Intelligence and Robotics & Automation in Space (i-SAIRAS), Noordwijk, the Netherlands, pp. 1-3.

Yen. J. (2008). Slip validation and prediction for Mars Exploration Rovers. Sensor & Transducers Magazine. Vol. 90, pp. 233-242.

Zhang, K.; Lafruit, G.; Lauwereins, R.; Van Gool, L. (2010). Joint integral histograms and its application in stereo matching. In proceedings of the International Conference on Image Processing, pp. 817-820. Hong Kong, China, pp. 817-820.

Zhou, F.; Arvidson, R.; Bennett, K.; Trease, B.; Lindemann, R.; Iagnemma, K.; Senatore, C.; Belluta, P.; and Maxwell, S. (2014). Simulations of Mars Rover Traverses. Journal of Field Robotics, Volume 31, Issue 1, pp. 141-160.