DOCTORAT DE L’UNIVERSITÉ DE TOULOUSE

Présentée et soutenue par :
Alexandru RUSU
le vendredi 12 décembre 2014

Titre :
Path Planning and Autonomous Navigation for a Planetary Exploration Rover

Planification de chemin et navigation autonome pour un rover d’exploration planétaire

École doctorale et discipline ou spécialité :
EDSYS : Robotique

Unité de recherche :
Équipe d'accueil ISAE-ONERA CSDV

Directeur(s) de Thèse :
M. Michel DEVY (directeur de thèse)
Mme Yoko WATANABE (co-directrice de thèse)

Jury :
M. Faïz BEN AMAR, Professeur d’Université - Rapporteur
M. François CHARPILLET, Directeur de recherche - Rapporteur
M. Michel DEVY, Directeur de recherche - Directeur de thèse
Mme Sabine MORENO, Ingénieur de recherche - Examinatrice
Mme Yoko WATANABE, Ingénieur de recherche - Co-directrice de thèse
M. René ZAPATA, Professeur d’Université - Examinateur
Abstract

ESA’s ExoMars mission will deploy a 300kg class rover on Mars, which will serve as a mobile platform for the onboard scientific instruments to reach safely desired locations where subsurface drilling and scientific measurements are scheduled. Due to the limited inter-planetary communication constraints, full autonomous onboard navigation capabilities are crucial as the rover has to drive over 70 meters per sol (Martian day) to reach designated scientific sites. The core of the navigation software to be deployed on the ExoMars rover uses as baseline the autonomous navigation architecture developed by CNES during the last 20 years. Such algorithms are designed to meet the mission-specific constraints imposed by the available spatial technology such as energy consumption, memory, computation power and time costs.

The first objective of this thesis is to improve the performance of the successive local path planning architecture proposed by CNES. First, the use of an incremental local path planner, Fringe Retriving A*, is proposed to reduce the path planning computation load. This is complemented by the introduction of binary heaps in the management structures of the path planner. In-place-turn maneuvers during trajectory execution are further reduced by using a state lattice path planner which encodes the steering capabilities of the rover.

The second research direction concerns global path planning capabilities for robotic planetary exploration. First the onboard memory constraints are relaxed and a study evaluating the use of a global D* lite path planner is performed. Second, a novel multi-resolution representation of the navigation map which covers larger areas at no memory cost increase is proposed. It is further used by a global path planner which automatically reduces the computational load by selecting its search direction based on obstacle shapes and distribution in the navigation space.
I dedicate this work to my son Marc and to my future wife Isa. This thesis would have not been possible without her love. I would also like to dedicate this thesis to my parents, sister and to my future in-laws who supported and encouraged me during stressful moments. Their unconditional support throughout the entire period was invaluable.
Contents

List of Figures viii
List of Tables xii
List of Acronyms xiii

Chapter 1 Introduction 1
  1.1 Mars Exploration Missions ........................................... 1
  1.2 Exploration Rovers on duty ......................................... 2
    1.2.1 Sojourner (Mars Pathfinder Mission) .......................... 2
    1.2.2 Spirit and Opportunity (Mars Exploration Rovers) .......... 3
    1.2.3 Curiosity (Mars Science Laboratory) ......................... 5
  1.3 The ExoMars Mission .................................................. 5
  1.4 Problem statement .................................................... 7
  1.5 Thesis Objectives and Outline ...................................... 7

Chapter 2 Development platform 9
  2.1 EDRES software environment ...................................... 9
  2.2 Autonomous navigation architecture in EDRES .................. 11
    2.2.1 Perception chain ............................................... 12
    2.2.2 Navigation subsystem ......................................... 13
    2.2.3 Trajectory execution control subsystem ...................... 18
  2.3 Testing facilities ................................................... 20
    2.3.1 Mars-like test site ............................................ 20
    2.3.2 Digital elevation models (DEM) database ................... 21
  2.4 Robotic vehicles ..................................................... 22
    2.4.1 IARES Rover .................................................. 22
    2.4.2 ExoMars Rover ................................................ 24

Chapter 3 Path planning architecture proposed by CNES for robotic exploration 27
  3.1 Path planning problem and notation ............................... 27
  3.2 Grid representation .................................................. 28
  3.3 Sub-goal selection and validation .................................. 29
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4 A* algorithm</td>
<td>30</td>
</tr>
<tr>
<td>3.5 Cost functions</td>
<td>33</td>
</tr>
<tr>
<td>3.5.1 Traversability cost function</td>
<td>33</td>
</tr>
<tr>
<td>3.5.2 Rover heading cost function</td>
<td>35</td>
</tr>
<tr>
<td>3.5.3 Guided path planning cost function</td>
<td>36</td>
</tr>
<tr>
<td>3.6 Long range navigation</td>
<td>36</td>
</tr>
<tr>
<td>3.7 Summary</td>
<td>38</td>
</tr>
<tr>
<td>Chapter 4 Incremental path planning</td>
<td>41</td>
</tr>
<tr>
<td>4.1 Real time algorithms</td>
<td>41</td>
</tr>
<tr>
<td>4.2 Incremental search algorithms</td>
<td>42</td>
</tr>
<tr>
<td>4.2.1 Heuristic learning incremental search algorithms</td>
<td>43</td>
</tr>
<tr>
<td>4.2.2 Search tree reusing incremental search algorithms</td>
<td>43</td>
</tr>
<tr>
<td>4.3 Fringe-Retrieving A*</td>
<td>44</td>
</tr>
<tr>
<td>4.3.1 Principle</td>
<td>44</td>
</tr>
<tr>
<td>4.3.2 Operations</td>
<td>45</td>
</tr>
<tr>
<td>4.3.3 FRA* Example</td>
<td>49</td>
</tr>
<tr>
<td>4.4 Implementation in the EDRES environment</td>
<td>51</td>
</tr>
<tr>
<td>4.4.1 OPEN list management performance analysis</td>
<td>51</td>
</tr>
<tr>
<td>4.4.2 Previous search sub-tree restore methods</td>
<td>53</td>
</tr>
<tr>
<td>4.5 Experimental evaluation</td>
<td>56</td>
</tr>
<tr>
<td>4.5.1 OPEN list management performance analysis</td>
<td>56</td>
</tr>
<tr>
<td>4.5.2 FRA* path planning evaluation</td>
<td>57</td>
</tr>
<tr>
<td>4.6 Conclusions</td>
<td>61</td>
</tr>
<tr>
<td>Chapter 5 Nonholonomic motion planning</td>
<td>63</td>
</tr>
<tr>
<td>5.1 State of the art</td>
<td>63</td>
</tr>
<tr>
<td>5.2 Path planning on MERs</td>
<td>65</td>
</tr>
<tr>
<td>5.3 Path planning using motion primitives for the ExoMars rover</td>
<td>66</td>
</tr>
<tr>
<td>5.3.1 State lattice design</td>
<td>68</td>
</tr>
<tr>
<td>5.4 Path planning using control sets</td>
<td>75</td>
</tr>
<tr>
<td>5.4.1 State lattice cost function</td>
<td>76</td>
</tr>
<tr>
<td>5.4.2 State lattice heuristic estimates</td>
<td>77</td>
</tr>
<tr>
<td>5.4.3 Nonholonomic motion control with planned in-place turns</td>
<td>77</td>
</tr>
<tr>
<td>5.5 Experimental evaluation</td>
<td>79</td>
</tr>
<tr>
<td>5.5.1 Choose the best heuristic function</td>
<td>80</td>
</tr>
<tr>
<td>5.5.2 Nonholonomic vs grid path planner</td>
<td>82</td>
</tr>
<tr>
<td>5.5.3 Nonholonomic path planning with planned in-place-turns</td>
<td>85</td>
</tr>
<tr>
<td>5.5.4 Nonholonomic vs grid path planner evaluation on field navigation data</td>
<td>86</td>
</tr>
<tr>
<td>5.5.5 Mission scenario evaluation</td>
<td>87</td>
</tr>
<tr>
<td>5.6 Conclusions</td>
<td>91</td>
</tr>
<tr>
<td>Chapter 6 Incremental path planning using constant resolution global navigation maps</td>
<td>93</td>
</tr>
</tbody>
</table>
6.1 Incremental path planning algorithms for dynamic navigation environments 93
   6.1.1 $D^*$ algorithm 94
   6.1.2 Focused $D^*$ algorithm 97
   6.1.3 $D^*$ lite 100
   6.1.4 Other $D^*$ variants 104
6.2 Performance evaluation in Matlab 104
6.3 Performance evaluation of the $D^*$ lite in the CNES EDRES environment 107
6.4 Conclusions 111

Chapter 7 Global path planning using a multi-resolution navigation map representation 113
7.1 Multi-resolution navigation map representation 113
7.2 Multi-resolution map creation in the EDRES environment 115
7.3 Multi-resolution map update 120
   7.3.1 Update through successive shifts 120
   7.3.2 Update through an obstacle map 122
7.4 Global path planning on a multi-resolution navigation map 125
   7.4.1 Experimental evaluation 127
7.5 Terrain aware global path planning using a multi-resolution navigation map 129
   7.5.1 Performance evaluation 133
7.6 Nonholonomic terrain aware global path planning for a planetary exploration rover 135
   7.6.1 First scenario: Canyon traversal 135
   7.6.2 Second scenario: Dead-end escape 137
   7.6.3 Third scenario: Over-the-rocks drive 139
7.7 Conclusions 142

Chapter 8 Conclusion and perspectives 145

Résumé étendu 1

vii
## List of Figures

1.1 Total driven trajectory by the Sojourner rover ................................................. 2  
1.2 Artistic view of a Mars Exploration Rover (Spirit and Opportunity) .................. 3  
1.3 Summary of driven distance by each rover per sol (Spirit above Opportunity) and used drive mode .......................................................... 4  
1.4 Curiosity rover on the Mars surface ..................................................................... 5  
1.5 Elements of the ExoMars mission: TGO, EDM and ExoMars rover *(Credit: ESA)* 6  
2.1 EDRES modular representation ........................................................................ 10  
2.2 The rover simulator (RS) using the 3D model of the ExoMars Rover ................. 10  
2.3 Robotic planetary exploration scenario ............................................................. 11  
2.4 Flight qualified stereo-bench studied at CNES ................................................. 12  
2.5 Successive steps in the perception chain .......................................................... 14  
2.6 Digitized rover vehicle model .......................................................................... 15  
2.7 Terrain navigability tests .................................................................................. 15  
2.8 Example of navigation map representing the different types of areas ............... 17  
2.9 Path planning intermediary results .................................................................... 18  
2.10 Representation of the locomotion control method ............................................ 19  
2.11 SEROM test site facility .................................................................................. 21  
2.12 SEROM test site navigation map ...................................................................... 22  
2.13 HiRISE DEM and navigation map .................................................................... 23  
2.15 IARES rover .................................................................................................. 23  
2.14 ASU navigation map ....................................................................................... 24  
2.16 ExoMars navigation map ................................................................................ 25  
2.17 Locomotion configurations ............................................................................. 26  
3.1 Neighborhood definition for cell decomposition ............................................. 28  
3.2 Cell decomposition for binary classified navigation space ............................... 29  
3.3 Cell decomposition for binary classified navigation space ............................... 31  
3.4 Example of configuration providing multiple minimum-cost paths ................. 35  
3.5 Example of obstacle map with superposed local navigation map .................... 37  
3.6 Partial tangent graph path planning using the obstacle map ............................ 38  
4.1 Operations of Fringe-Retrievng A* ................................................................. 49  
4.2 FRA* example initial configuration ................................................................. 49  
4.3 FRA* results after first iteration ....................................................................... 50
4.4 FRA* results for the second iteration ................................................. 51
4.5 OPEN list management using double-linked ascending priority queue .... 52
4.6 OPEN list management binary heap priority queue .......................... 54
4.7 A graphical representation of FRA* path planning using previous-step search subtree with map shift. .................................................. 55
4.8 Performance comparison for the priority queue management ............ 58
4.9 Computation time comparison for global path planning with respect to: (a) the length of the global path, (b) the number of added states in the OPEN list, (c) the number of expanded states ............................................. 59
4.10 A graphical representation of validated and non-validated FRA* restore operations. 60

5.1 Terrain Assessment and Path Selection Modules of the GESTALT Navigation Planning ............................................................... 66
5.2 Example for path planning using the Dubins motion primitive set ......... 67
5.3 Reachability tree using the BL nonholonomic planning ....................... 68
5.4 Circular arc displacement for a robotic system ................................. 69
5.5 Trajectory decomposition ............................................................... 70
5.6 Illustration of continuous arc of circle paths and the corresponding state lattice connecting discretized states at initial heading $\theta_0^d = 45^\circ$ ................................. 73
5.7 Control set generated using Algorithm 6 ........................................... 74
5.8 Lattice controls symmetry transformations ...................................... 75
5.9 Motion controls generating the same state during path planning .......... 76
5.10 Illustration of a configuration where an in-place rotation is preferred: (a) Path (in white) given by the path planner using the precalculated control set and a zoom in of the Field of Plan (FOP) of the robot, (b) Path (in white) given by the same path planner but with initial adjusted heading ............................. 78
5.11 A planned in-place-turn is commanded only when the target lies behind the rover 78
5.12 Leaving the corridor scenario ....................................................... 79
5.13 Representation of the precomputed obstacle aware distance maps ....... 81
5.14 Computation time comparison regarding the used heuristic function .... 81
5.15 Performance gain when using planned rover heading correction in-place turns ... 86
5.16 Example of distance measure improvement through planned heading corrective in-place turn. Resulting paths for: (a) $A^*$ grid planner, (b) $R_5H1^1$, (c) $R_5H1^2$ . 87
5.17 Percentage of in-place turns executed during each mission scenario using: (a) $A^*$ grid planner, (b) Nonholonomic path planner with branching factor of 5, (c)Nonholonomic path planner with branching factor of 7 .......................... 89
5.18 Example of in-place turns generated during simulated mission using: (a) $A^*$ grid planner (b) nonholonomic path planner with branching factor of 5 .......................... 90
5.19 Memory use comparison per mission scenario with respect to the total traveled distance using: (a) $A^*$ grid planner, (b) Nonholonomic path planner with branching factor of 5, (c)Nonholonomic path planner with branching factor of 7 .......................... 91

6.1 Comparison of executed trajectories when using $A^*$ or $D^*$ lite with initial low resolution global navigation map ................................................. 107
6.2 Distribution of the result status for the comparison D* lite - LRN ........................................ 109
6.3 Comparison of executed trajectories when using LRN or D* lite ........................................ 109
6.4 Computation time for trajectories LRN or D* lite ................................................................. 109
6.5 Evolution of the number of consistent (Closed list) and inconsistent states (Open list) during D* lite run ................................................................. 110
6.6 Comparison of memory use when running D* lite ................................................................. 111

7.1 Example of local navigation map highlighting an exterior band with unclassified cells ................................................................. 116
7.2 Example of multi-resolution navigation map representation: (a) with two distinctive areas (a high resolution area with the rover model positioning test for the cell marked in red, and a low resolution band), (b) Cell connectivity at the border between different resolution regions ................................................................. 117
7.3 Navigation map coverage, depending on the chosen values for val_{HRES} and val_{LRES} and the used rover model ................................................................. 118
7.4 Example of multi-resolution navigation map (right) created from a high resolution global navigation map (left) ................................................................. 119
7.5 Illustration of shift procedure ................................................................................................. 120
7.6 Multi-resolution navigation map update through successive shifts ........................................ 122
7.7 Multi-resolution navigation map update using an updated obstacle map ........................................ 123
7.8 Example in the Rover Simulator ........................................................................................... 124
7.9 Example on how obstacle shapes can influence the performance of a grid path planner ................................................................. 126
7.10 Synthesis of results obtained during the tests ....................................................................... 128
7.11 Performance evaluation of the multi-resolution methods with respect to SLA* .................. 129
7.12 Gaussian distributions for the ratio of memory use by each of the proposed multi-resolution path planning approach with respect to SLA* ................................................................. 129
7.13 Concaveness score calculation using the Center of gravity displacement (CGD) approach ................................................................................................. 131
7.14 Concaveness score calculation using the Concave regions distribution (CRD) approach ................................................................................................. 132
7.15 Concaveness score calculation using the Opposite concave regions comparison (OCR) approach ................................................................................................. 132
7.16 Concaveness score calculation using the Shortest distance to circumnavigate (SDC) approach ................................................................................................. 133
7.17 Concaveness score calculation using the Extended shortest distance to circumnavigate (ESDC) approach ................................................................................................. 134
7.18 Accuracy evaluation of best search direction prediction using the accumulated concaveness scores ................................................................................................. 135
7.19 Canyon traversal: Driven trajectories ................................................................................... 136
7.20 Dead-end escape: Driven trajectories ..................................................................................... 138
7.21 Over-the-rocks traversal: Driven trajectories .......................................................................... 139
7.22 Canyon traversal: Terrain-aware automatic search direction method assessment .................. 140
List of Tables

2.2 IARES rover characteristics ........................................ 23
2.1 Navigation maps areas distribution .................................. 24
2.3 ExoMars rover characteristics ...................................... 25

3.1 List of heuristic estimations ........................................... 30

4.1 Capabilities of heuristic learning incremental search algorithms ........ 43
4.2 Comparison of time efficiency regarding the used data structure for the priority queue ................................................ 53
4.3 Experimental set summary ............................................. 59
4.4 Performance analysis for incremental path planning .................. 60

5.1 Parameters used to calculate the state lattice control set .............. 80
5.2 Nonholonomic path planning single query experimental set summary .... 83
5.3 Summary for real navigation data path planning experiment ........... 88
5.4 Nonholonomic path planning for mission scenario .................... 89

6.1 Replanning performance comparison on random grids of 100 × 100 ........ 105
6.2 Replanning performance comparison on random grids of 500 × 500 ........ 106
6.3 Performance evaluation of trajectories provided by the D\(^*\) lite algorithm .... 108

7.1 Traversability score for the low resolution cells ...................... 119
7.2 Path planning strategies summary .................................... 127
7.3 Performance evaluation of the multi-resolution methods with respect to LRN, for the failure cases of SLA\(^*\) ........................................ 130
7.4 Performance measure synthesis for the Canyon traversal scenario ....... 137
7.5 Performance measure synthesis for the dead-end escape scenario ........ 138
7.6 Performance measure synthesis for the over-the-rocks scenario .......... 140
## List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALD</td>
<td>ExoMars Rover Analytical Laboratory Drawer</td>
</tr>
<tr>
<td>ANW</td>
<td>Autonomous Navigation Workshop</td>
</tr>
<tr>
<td>ARTEMIS</td>
<td>Autonomous Rover and Testbench for Exploration MISsions</td>
</tr>
<tr>
<td>ASU</td>
<td>Astrium UK Mars-like DEM</td>
</tr>
<tr>
<td>BFS</td>
<td>Best First Search path planning algorithm</td>
</tr>
<tr>
<td>CNES</td>
<td>Centre National d’Études Spatiales, the French National Space Agency</td>
</tr>
<tr>
<td>DEM</td>
<td>Digital Elevation Model</td>
</tr>
<tr>
<td>EDM</td>
<td>Entry, descent and landing Demonstrator Module</td>
</tr>
<tr>
<td>EDRES</td>
<td>Environnement de Développement pour la Robotique d’Exploration Spatiale, Software Environment for Space Exploration Robotics</td>
</tr>
<tr>
<td>ESA</td>
<td>European Space Agency</td>
</tr>
<tr>
<td>FGLA*</td>
<td>Forward Guided Local A*</td>
</tr>
<tr>
<td>FRA*</td>
<td>Fringe Retrieving A*</td>
</tr>
<tr>
<td>GESTALT</td>
<td>Grid-based Estimation of Surface Traversability Applied to Local Terrain</td>
</tr>
<tr>
<td>GNC</td>
<td>Guidance, Navigation and Control</td>
</tr>
<tr>
<td>HiRISE</td>
<td>High Resolution Imaging Science Experiment</td>
</tr>
<tr>
<td>IARES</td>
<td>Illustrateur Autonome de Robotique mobile pour l’Exploration Spatiale, Autonomous Demonstrator for Space Exploration Robotics</td>
</tr>
<tr>
<td>ISS</td>
<td>International Space Station</td>
</tr>
<tr>
<td>LRN</td>
<td>Long Range Navigation path planning</td>
</tr>
<tr>
<td>MER</td>
<td>Mars Exploration Rover</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>MRFA*</td>
<td>Multi-Resolution Forward A* path planning</td>
</tr>
<tr>
<td>MRO</td>
<td>Mars Reconnaissance Orbiter</td>
</tr>
<tr>
<td>MRRA*</td>
<td>Multi-Resolution Reverse A*</td>
</tr>
<tr>
<td>MSL</td>
<td>Mars Science Laboratory</td>
</tr>
<tr>
<td>NASA</td>
<td>National Aeronautics and Space Administration</td>
</tr>
<tr>
<td>NMRES</td>
<td>Nonholonomic global path planning using Multi-RESolution navigation map</td>
</tr>
<tr>
<td>RGLA*</td>
<td>Reverse Guided Local A*</td>
</tr>
<tr>
<td>RS</td>
<td>Rover Simulator</td>
</tr>
<tr>
<td>SEROM</td>
<td>Site d’Essais pour la RObotique Mobile, Test Site for Mobile RObotics</td>
</tr>
<tr>
<td>SLA*</td>
<td>Successive Local A* path planning</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>The Transmission Control Protocol (TCP) and the Internet Protocol (IP)</td>
</tr>
<tr>
<td>TGO</td>
<td>Trace Gas Orbiter</td>
</tr>
<tr>
<td>VME</td>
<td>Visual Motion Estimation</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Mars Exploration Missions

During the last decades planet Mars has experienced an increased scientific interest. The main purpose of planetary exploration missions is to decrypt whether the planet’s past environmental conditions were favorable for the development of microbial life. Mars exploration missions date back to the 1960s, with several unsuccessful flyby attempts, until the US NASA missions Mariner 4 (1964), 6 and 7 (1969) were successfully attained. The first operational orbiters were the US Mariner 9 and the Russian Mars 2 (1971). The latter carried a lander and a 4.5 kg rover that became the first man-made object to crash on the Mars surface. The US Viking 1 and 2 missions were the first two landers which performed successful landing. In total, seven missions have successfully reached the surface of Mars: three landers, Viking 1 and 2 (1976), Phoenix (2008), and four rovers, Sojourner (Mars Pathfinder mission, 1997), Spirit, and Opportunity (MER-A and MER-B, 2004), with the latest one being Curiosity (Mars Science Laboratory, 2012).

Due to engineering complexity of interplanetary journey, approximately two thirds of all missions designed to reach and study planet Mars failed either before they reached their destination or before they completed their missions. Nevertheless, astonishing achievements were made on missions like Mars Exploration Rover (2004) or Mars Science Laboratory (2011): the two rovers onboard MER operated over at least 25 times more than their nominal mission duration, and the MSL mission performed an outstanding landing of a car-size rover, named Curiosity. In the past 20 years, mobile robotic systems were deployed to undertake in-situ scientific measurements, while new technologies are under development for human spaceflight and exploration.

As of 10th of October 2014, there are two rovers (Opportunity and Curiosity) operating and actively exploring the Mars surface and five orbiters on the Martian orbit: Mars Odyssey, Mars Express, Mars Reconnaissance Orbiter, Mars Orbiter Mission and MAVEN. Besides taking in-orbit scientific measurements, they also serve as communication relay satellites for landers and rovers on the Martian surface.

Future missions to Mars include NASA Insight Mission (2016) and ESA ExoMars Mission (2016, 2018). Currently under study by NASA, the Mars 2020 rover mission is expected to place on the Martian surface a rover derived from the design of Curiosity rover.
1.2 Exploration Rovers on duty

1.2.1 Sojourner (Mars Pathfinder Mission)

Being the first robotic vehicle reaching the Martian surface on 4th of July 1997, Sojourner laid the foundation for autonomous robotic planetary exploration. Given a nominal mission duration of only 7 sols (Martian day), it operated during 83 sols. It mainly relied on the Pathfinder lander to communicate with the Rover Control Station on Earth. The lander had the capability to build 3D models of the surrounding landing site by using stereo images. Human operators used this environmental information within a graphical interface along with the kinematic model of Sojourner to provide 1-2 meters spaced waypoints to the rover for its navigation.

In order to carry out its daily missions, the rover had onboard autonomous capabilities to detect and avoid rocks, slopes and drop-off hazards during traversals. It was also able to autonomously manage its resources during mission execution: available energy to perform the given tasks and communication link with the lander (in case of communication loss, an emergency reversal maneuver was performed). Using these capabilities, the Sojourner rover completed its mission with approximately 100 meters driven on the Mars surface, while staying always in a radius of 12 m around the lander. Figure 1.1 contains the total rover traverse on a panoramic view from the lander.
1.2.2 Spirit and Opportunity (Mars Exploration Rovers)

During the third part of NASA’s Mars Exploration Program, two identical rovers, Spirit and Opportunity in Figure 1.2, landed in two widely separated locations on the Martian surface on 4th of January 2004 and on 25th of January 2004. The performance achieved by the two rovers has surpassed any expectation for robotic autonomous planetary exploration. Unlike Sojourner, they do not rely on their landers, and hence can drive long traversals independently. They are equipped with three types of stereo camera pairs which enables them to produce onboard imagery: Hazard Cameras (HazCams) mounted on the front and back of the chassis, Navigation Cameras (NavCams) and Panoramic Cameras (PanCams) mounted on a mast. These cameras are used to take high resolution images of the surrounding terrain and to provide information on the terrain texture and shape at different scales [Biesiadecki et al., 2005]. Similar to the case of Sojourner, human operators perform terrain traversability analysis and select waypoints that the rover has to reach while driving towards the long-distance goals.

![Artistic view of Mars Exploration Rover](image)

Figure 1.2: Artistic view of a Mars Exploration Rover (Spirit and Opportunity)

Depending on the difficulty of the terrain around the rover (high density of rocks, dangerous slopes where the rover can tilt over the limit or sandy terrain where it can get stuck) and on the amount of allocated time and energy, human operators can select one of the three main drive modes available in the navigation software of the rovers. In the first mode entitled Directed Driving (or Blind Drive), the rover drives directly to a waypoint without any visual sensing. In this setting, the rover can reach its highest speed of 124m/h, covering longer distances but at a high risk as the rover relies only on its wheel odometers for position estimation. This mode is usually used on flat surfaces with low obstacle density and lack of high-slip hazards, where the security of the rover is not threatened. However, the 3D data becomes sparse at longer range and makes it difficult for human operators to decide safe paths. Therefore, in the presence of possible risky areas, the rover can perform hazard avoidance autonomous navigation (AutoNav) by performing traversability analysis of its close surroundings. In this mode, the rover generates 3D models of the surrounding terrain and builds a goodness map representing the roughness of
Chapter 1. Introduction

the terrain with respect to the rover locomotion capabilities. After that, it uses the GESTALT algorithm [Goldberg et al., 2002] detailed in Section 5.2 to choose a best precomputed path to be driven. The computation time for 3D terrain reconstruction and safe path planning affects the available time for path execution. Thus, the maximum speed which can be attained varies from 36 m/h to 96 m/h. Finally, for very difficult terrain, the Visual Odometry mode (VisOdom) can be employed in order to have accurate position estimates of the rover during execution. Due to its high computational load, this mode limits the maximum speed of the rover to only 10 m/h.

Figure 1.3 provides a synthesis of driven distance and driving mode used by each of the rovers during their first 9 months of operation [Biesiadecki et al., 2005]. Although intensively tested in simulations, the AutoNav and VisOdom drive modes are very rarely used because of their high computational time requirements. In consequence, the blind-drive mode with step-by-step commands has been predominantly used in order to maximize the execution speed and the distance covered.

Figure 1.3: Summary of driven distance by each rover per sol (Spirit above Opportunity) and used drive mode

With a nominal mission duration of 90 sols, both rovers operated at least five times more. Spirit ended its mission after 2623 sols with a total driven distance of 7,730.50 m being stuck in
soft soil for approximately one year. Opportunity is still in operation and as of 2nd of October 2014, it holds the record of the longest traverse and lifetime of a vehicle on another planet with 3798 sols and 40.78 km driven.

1.2.3 Curiosity (Mars Science Laboratory)

The last NASA rover, shown in Figure 1.4 landed on the Mars surface on 6th of August 2012. It is a car-sized vehicle, with a mass of 899 kg including 80 kg of scientific payload. The rover is 2.9 m long by 2.7 m wide, with a height of 2.2 m. It is equipped with six 50 cm diameter wheels on a “rocker-bogie” suspension system inherited from Pathfinder and Mars Exploration Rovers. The rover design allows a tilt of maximum 45° in any direction, but tilts over 30° are avoided during navigation. It has a ground clearance of 60 cm, and it can drive over obstacles of up to 65 cm height. It can travel at a speed of up to 90 m/h, but the average speed is approximately 30 m/h.

![Figure 1.4: Curiosity rover on the Mars surface](image)

The autonomous navigation system is similar to the one used for the Martian Exploration Rovers mission. It features a total of 17 cameras, including two pairs of grayscale navigation cameras (NavCams) mounted on its mast and four pairs of hazard avoidance cameras (HazCams) used for autonomous navigation. The onboard computation resources include 256 MB of DRAM, 2GB of Flash Memory and 256 KB of EEPROM. This capacity is approximately 8 times higher than the onboard resources of the Mars Exploration Rovers.

1.3 The ExoMars Mission

The ExoMars mission is currently a cooperation between the European Space Agency (ESA) and the Russian Federal Space Agency. Its main objective is to search for past or present signs of life on Mars. It consists of two missions planned to be launched in 2016 and 2018.

The first mission (2016) will include onboard a Trace Gas Orbiter (TGO) and an Entry, descent and landing Demonstrator Module (EDM). The orbiter will study the Martian atmospheric
trace gases such as methane and identify their sources. The EDM will perform measurements
during the entry phase to assess the capability of the lander to pose a payload safely on the
surface of Mars. In addition, it will perform in-situ analysis of the environment at the landing
site.

The 2018 mission will land on the Martian surface an European rover and a Russian surface
platform. The ExoMars rover will carry onboard both European and Russian instruments. While
the Russian Surface Platform will focus on surface environmental and geophysical investigations,
the ExoMars rover will drive to scientific sites in order to analyze the physical and chemical
properties of subsurface Martian terrain samples. The rover is also equipped with a drill which
enables the subsurface sampling from various depths down to 2 meters. Besides automatic sample
collection, it will be capable of distributing samples to the Rover’s Analytical Laboratory Drawer
(ALD) for further scientific experiments. Figure 1.5 provides an artistic view of all elements
included in the ExoMars Mission.

The ExoMars rover has a nominal mission duration of 218 sols and is expected to ensure
a regional mobility by driving over several kilometers. It is a key component of the ExoMars
mission as it has to traverse the Martian terrain safely while providing the ability to access
locations of high scientific interest.

The rover is designed to drive at a nominal speed of approximately 40 m/h. This speed can
be increased up to 70 m/h on plain terrain while following straight lines. However, depending
on the type of terrain (rocks, slope, texture), its speed can be reduced down to 10 m/h when
very accurate position estimation is requested. Due to mission constraints, the rover is designed
to perform long traverses at a high level of autonomy. Therefore, it is required for the rover
to compute onboard safe navigation solutions to reach any target location specified by human
operators. This autonomous navigation functionality limits the distance the rover can drive
during one sol to 70 m with an average speed of approximately 15 m/h.

Figure 1.5: Elements of the ExoMars mission: TGO, EDM and ExoMars rover (Credit: ESA)
1.4 Problem statement

Onboard autonomous navigation for planetary exploration rovers is a very important resource which can strongly impact the scientific mission return. The faster the rover can reach the selected scientific targets, the more time can be dedicated to scientific measurements. For example, in the case of the ExoMars rover, it is estimated that one quarter of the nominal mission duration will be spent to traverse between selected locations and the rest of time will be used to perform scientific measurements. Through recent tests of astronaut-controlled telerobotic operation of rovers from the International Space Station, it was shown that rover autonomy plays an important role on the enhancement of the operational efficiency and robot utilization. In this test, the rover could robustly perform its tasks with no constraints due to communication latency and even during extended periods of communication loss from ISS to Earth. In the case of MER and Curiosity, autonomous navigation capabilities are only used to keep the rover away from hazards that are poorly known by human operators.

The importance of the autonomous navigation software is emphasized by the limited communication possibilities (3 to 21 minutes delay and up to two communication windows per sol) between the rover and the control center on Earth. This limits the number of remote control commands sent to the rover from human operators, and so affects the efficiency of the exploration activity. If the rover has robust autonomous navigation capabilities, the whole communication bandwidth would be used to transmit scientific data instead of control data to be used by human operators to decide on mission directions for the following sols.

The autonomous navigation algorithms have to respect the mission-specific constraints imposed by the actual technology available for space missions. The autonomous navigation system will run on the rover’s 96MHz LEON2 co-processor with limited onboard memory capacity of only 256MB.

CNES holds a crucial position between worldwide players for planetary exploration space programs through its Space Robotics group in the System Validation Department. An autonomous navigation architecture for robotic planetary exploration has been developed for more than 20 years, and was offered to ESA as a contribution to the ExoMars mission. The maturity of the software was proved through intensive simulation and field validation tests, including two remote experiments of ESA/ESTEC on the SEROM test site [Joudrier et al., 2011] [Joudrier et al., 2012].

1.5 Thesis Objectives and Outline

This thesis aims to contribute to the autonomous navigation system for planetary robotic exploration rovers previously developed at CNES by introducing new path planning algorithms and environment representation methods to improve its performance. Two main directions are addressed in this thesis. First, new methods are developed and validated for local path planning in order to obtain a better management of the onboard computation resources and memory capacity and to reduce the locomotion system wear and execution time during path execution. The second objective is to include global mapping and path planning capabilities without violating the mission constraints. In order to do so, it is required to develop a novel approach for
surrounding terrain representation which allows the rover to plan safe over-the-horizon traverses.

The thesis is organized as follows: Chapter 2 provides a summary of the autonomous navigation architecture developed by CNES. It details all the intermediate steps from perception and navigation map calculation, to path planning and execution. Also, a short descriptions of testing facilities and rover models used in validation tests are provided.

Chapter 3 focuses on the existing algorithms in the CNES architecture for both local and global path planning, which represent the backbone of this thesis.

Chapter 4 proposes the use of incremental path planning algorithms for static environments. It focuses on the reuse of navigation data between subsequent path planning processes for faster path calculation. An optimized data structure is implemented to provide a better management of data structures in path planning algorithms in order to reduce the computational load.

Chapter 5 introduces a path planning approach which takes into account the locomotion capabilities of the rover. First, a method to generate a precomputed state lattice which encodes the steering capabilities of the rover is proposed. This state lattice is further used by an optimal path planner which generates paths with reduced locomotion system wear and path execution time. The main goal of this chapter is to assess the feasibility and applicability of the proposed nonholonomic path planner for robotic planetary exploration.

The second part of this thesis addresses the problem of global autonomous path planning. It consists in the analysis of a family of incremental path planning algorithms operating in dynamic environments, which results in the choice of the D* lite algorithm for further implementation and testing for robotic mission scenarios in Chapter 6. Here, the onboard memory use constraint is relaxed and a global high resolution navigation map is used to store navigation data during a sol.

Lastly, Chapter 7 consists in the proposal of a novel global representation of the navigation environment based on a multi-resolution navigation map. It provides the advantage to cover larger areas with the same memory requirements as the local navigation map used in the CNES autonomous navigation architecture. Then, a terrain-aware global path planner is developed to reduce the computational load by selecting the search direction based on the distribution and shape of the encountered obstacles. In this way, the main contribution of this thesis consists in the proposal of a global path planner which employs a multi-resolution navigation map representation with limited memory use and a nonholonomic local path planner able to calculate cost and energy efficient paths for execution.

The last chapter includes the conclusions of this thesis and suggestions for future work that can be developed based on the advances presented here.
Chapter 2

Development platform

During the past 20 years, the Space Robotics team of the System Validation Department of CNES has worked in the development of state-of-the-art algorithms and techniques to perform autonomous navigation for planetary exploration rovers. The focus is on the development and validation of the Guidance, Navigation and Control (GNC) system of the ExoMars rover including stereoscopic vision, localization and autonomous navigation. Elaborated concepts and algorithms are validated on exclusive test facilities and equipment, including a 3000 m$^2$ outdoor Mars-like terrain and two planetary exploration rover demonstrators IARES and ARTEMIS (a brand new vehicle with the kinematic chain closer to the ExoMars rover, designed and assembled in-house). This chapter aims to provide a detailed description of the currently implemented autonomous navigation software and the available ground demonstrators and test facilities.

2.1 EDRES software environment

The EDRES (Environnement de Développement pour la Robotique d’Exploration Spatiale) software environment is the result of over two decades of software development for autonomous planetary exploration rovers at CNES. It encapsulates in a coherent system all functions, from low-level drivers and onboard programs to high level algorithms and applications needed to perform autonomous motion planning and execution for exploration robotic vehicles. Since 2008 CNES has gradually transferred the EDRES software environment to ESA, providing support for the designs of stereo-vision benches (NAVcams, LOCcams, HazCams) and of perception and navigation algorithms [Bousquet, 2011].

EDRES has a modular organization, shown in Figure 2.1, working on a client/server basis, so that each subsystem can perform independently. This feature brings a high benefit to the entire system by eliminating the dependency of the applications on the physical equipment. As each subsystem has an independent interface with the server, it can be easily modified, added or excluded, with no impact over the functionality of other subsystems. For example, the Autonomous Navigation Workshop (ANW), a tool which allows to generate and execute trajectories for robotic vehicles, uses a TCP/IP protocol to connect to the robot simulator which simulates the robotic dynamics based on the available 3D models. Thus, the link between the server and a subsystem can be established either locally, or through network (distributed
architecture using the TCP/IP protocol). This structure has the advantage that each subsystem performance can be evaluated using both physical equipment or simulation generated data. The aim of the EDRES environment is to serve as a workshop to create and validate new algorithms, tools or applications for robotic planetary exploration.

One of the main tools of EDRES is a 3D real-time rover simulator, shown in Figure 2.2, which is the key module for closing the loop in the autonomous navigation chain. It uses high-resolution Martian-like Digital Elevation Models (DEMs), 3D kinematics models and locomotion systems of different exploration rovers in order to provide accurate dynamics simulations. Moreover it can simulate perception and localization systems, being a crucial asset for reducing the operating costs of the real robotic platforms available at CNES. EDRES is programmed in C/C++ under Linux and it makes use of GTK for user interfaces and OGRE for the 3D simulator.
2.2 Autonomous navigation architecture in EDRES

![Diagram of robotic planetary exploration scenario](image)

Figure 2.3: Robotic planetary exploration scenario

Due to communication time delay between Mars and Earth, human operators cannot send real-time commands to control the rover. Figure 2.3 illustrates the scenario for robotic planetary exploration as suggested in the ExoMars mission. The rover usually receives the instructions set for the current mission at the beginning of each sol. Data is transmitted either through direct communications with Earth or using communication relay satellites on the Martian orbit. The command sequence includes locomotion objectives (mission goal coordinates, distance to drive, heading) [Rastel and Maurette, 2006] and requested scientific tasks. Further on, operations of the rover totally depend on the local data acquired by the rover. The rover is expected to drive over distances of up to 70 meters per sol, when in the autonomous navigation mode is activated, until the selected target is reached and scientific instruments can be deployed. At the end of each sol, the rover transmits to the Operation Center information acquired during the mission, including surrounding landscape images. The transmitted data is analyzed by human operators in order to decide the instruction set for the following sol. As shown in Figure 2.3, locomotion cycles are repeated in a loop, allowing the rover to progress in the direction of the mission goal, until a stopping condition is met. A locomotion cycle is performed by using three subsystems: the perception chain, the navigation subsystem and the trajectory execution control subsystem. Stopping conditions could be:

- the mission goal is successfully reached, according to the localization data;
- locomotion cycle is suspended due to a priority task (e.g. communication window with the control center)
Chapter 2. Development platform

- no safe path is found to reach the goal or no further drive is allowed due to security checks (e.g. slip checks ensure that the rover is not stuck into a sand trap);

Each of the three subsystems in the locomotion cycle will be detailed in the following subsections.

2.2.1 Perception chain

The objective of the perception subsystem is to provide the field data required by further subsystems in order to build an accurate 3D DEM of the surrounding terrain. Perception data is provided by a stereo bench (called NavCam) mounted approximately 2 meters above the ground level, on the rover mast with a pan/tilt mechanism. Figure 2.4 presents the prototype of a flight model stereo-bench studied by CNES, which consists of a pair of space-qualified 1024 × 1024 pixels cameras separated by a titanium plate (stereo base of 100mm). The 8-bit grayscale images are downsampled for further use, in order to emulate low-resolution flight models and to decrease the computational load. While stationary, at the beginning of each locomotion cycle the NavCam takes three pairs of images in the heading direction of the rover used to estimate the 3D model of the observed terrain.

![Flight qualified stereo-bench studied at CNES](image)

Figure 2.4: Flight qualified stereo-bench studied at CNES

Ideally, a point in the landscape should be projected on the same relative image line of the two camera sensors. This condition should be met for the whole common field of view of the two cameras. The conception of such a stereo device leads to a very complex optical and mechanical design. An alternative is to use a stereo system with wide alignment errors which remains stable during the lifetime of the mission. The alignment error of the optical axes of the lenses or of the sensor matrices, the distortions and the mechanical imperfections are compensated using correction maps generated by a CNES patented calibration procedure [Lamboley, 2003]. It is a pixel-by-pixel calibration method, which iteratively compares the position of the image of a star on the sensor matrix to its expected position provided by high precision pitch and yaw rotating tables. The stereo-vision algorithms using the aforementioned correction maps was
validated using a laser scanner which acquires high density point clouds with ±2mm accuracy [Remetean et al., 2011].

The distortion-corrected images are further used to calculate the disparity map. In order to account for the exposure time difference between the two optical sensors, the matching process is performed on the corresponding gradient images, which are the result of the application of a 9 × 9 Gaussian-smoothed Laplacian operator on the grayscale images. The relative depth information is extracted from each image pair by calculating best estimates of the positions of features in the two images. For a given landmark detected at the pixel coordinates \((x, y)\) on the left image, the corresponding pixel on the right image has to be identified. This is performed by analyzing all the positions in a rectangular window centered at \((x, y)\) position on the right image, and retaining the pixel with the best matching score. In order to minimize the computation time, the matching score is defined by a simplified criterion as given in eq. 2.1. \(I_1\) and \(I_2\) are the left and right gradient images, and \(d\) is the considered horizontal shift. The minimum criterion value will provide the corresponding pixel and disparity for the current landmark. Furthermore sub-pixel interpolation is performed to recover a precise, fractional disparity map. Then, outliers are removed to obtain the filtered disparity map. Figure 2.5 provides an example of each intermediary step of the perception chain.

\[
C(x, y, d) = \sum_i \sum_j [I_1(x + i, y + j) - I_2(x + d + i, y + j)]
\] (2.1)

The filtered disparity values are converted into distance measures using the geometric characteristics of the cameras. They are used along with the position and attitude of the stereo-bench to calculate the corresponding projection on a regular grid, to obtain the 3D Digital Elevation Model (DEM) of the observed terrain.

### 2.2.2 Navigation subsystem

The aim of the navigation subsystem is to first calculate a navigation map which represents the traversal difficulty of the surrounding terrain with respect to the locomotion capabilities of the rover. The navigation map is further used to calculate a minimum-cost safe path that the rover should follow to get closer to its final target.

#### Navigation map construction

As the area covered by a single stereoscopic acquisition is limited, the navigation map encapsulates information acquired during subsequent acquisitions. In the EDRES environment, the local navigation map has a fixed size of 351 × 351 pixels centered at the current rover position. This representation provides a coverage of approximately 14 × 14 meters at a resolution of 40 mm per pixel.

Using the DEM of the terrain around the rover, a cell in the navigation map is analyzed based on two criteria: the slope of the terrain (relative to the altitude gradients) and punctual obstacles (called here discontinuities). Theoretically, the full kinematic model of the rover has to be considered for a precise calculation of the two criteria. However, the use of a full kinematic model would be very costly in computation resources, thus infeasible for space applications. For
Figure 2.5: Successive steps in the perception chain
this reason, the simplified rover model shown in Figure 2.6 is used to calculate the aforementioned criteria.

![Figure 2.6: Digitized rover vehicle model](image)

The wheel positioning test consists in analyzing discontinuities in the altitude values of the DEM cells which cannot be crossed safely. To identify such cases, the center of the rover’s wheel model represented as a rectangle shape with the length of each side equal to $WS$ is placed on each DEM cell. If the difference between the maximum and minimum elevation values of the cells covered by the wheel model, exceeds the threshold of the rover’s obstacle clearing capability, the corresponding cell is classified as non-navigable due to discontinuity.

After the wheel positioning test, the inclination test is performed to examine the rover’s ability to traverse without surpassing its roll and pitch constraints. The threshold for the maximum traversable roll and pitch angles can differ depending on the nature of the terrain: cohesive soil (rock) or poor cohesion soil where wheels can hardly gain traction (sand). The center of the rover model is placed on each DEM cell and the maximum lateral inclination is computed. The abstract model used in this process includes an axle with the width equal two times the rover radius $RR$ and two wheels of size $WS$, as given in Figure 2.6. The positioning of the model is repeated for a set of different orientations as shown in Figure 2.7, and the maximum inclination value is retained. If this value exceeds the maximum acceptable angle for the rover, the cell is classified as non-navigable due to slope.

![Figure 2.7: Terrain navigability tests](image)
Chapter 2. Development platform

For security reasons, cells which successfully passed the first two tests, but with marginal values, will be classified as non-unifiable due to discontinuity or slope. This stage aims to account for hysteresis in the wheel positioning and inclination tests. Such cells will be avoided by path planning algorithms unless they are the only way to rescue the rover from a dangerous area. Unknown cells in the navigation map correspond to cells of the DEM which do not provide enough information to perform the navigability test. All the rest of cells with acceptable discontinuity and slope values are classified as navigable and assigned a transferability score as a function of the two measures.

Once the navigation map corresponding to the current perception is calculated, it is merged with the navigation map built during previous locomotion cycles. For overlapping areas, priority is given to the most recent data. Here, uncertain cells in the merged navigation map are defined in order to account for localization and locomotion errors accumulated between consecutive perceptions. They are cells which were initially classified as navigable during previous perceptions but which change status due to the dilation of all non-navigable and non-planifiable areas in the merged navigation map outside the field of view of the current perception. These cells are not considered to be forbidden for planning, but when a trajectory contains such cells the rover has to take new perceptions to update the navigation data and eliminate the accumulated uncertainties.

In summary, the navigation map is a constant resolution grid based representation which can contain up to 7 cell types as follows:

- **Navigable cell**, the discontinuity and slope measures satisfy the constraints imposed by the locomotion system of the rover.
- **Unknown cell**, where there is not enough information to decide whether it is navigable or not,
- **Non-navigable cell due to discontinuity**, the center of the robot wheel cannot be posed on the current cell safely,
- **Non-navigable cell due to slope**, the center of the robot model cannot be posed on the current cell safely,
- **Non-planifiable cell due to discontinuity**, the discontinuity measure is close to the non-navigable discontinuity threshold,
- **Non-planifiable cell due to slope**, the slope measure is very close to the non-navigable slope threshold,
- **Uncertain cells**, areas which become uncertain due to localization and locomotion errors.

Further, two new variables are defined to be used during the path planning stage: path planning perimeter and execution perimeter. The path planning perimeter decides how areas in the navigation map are treated differently depending on their distance to the rover. For example, unknown areas will be considered as forbidden when they lie inside the path planning perimeter. Otherwise they are temporarily tagged as navigable until new perceptions are available. The
2.2. Autonomous navigation architecture in EDRES

Even if the merged navigation map covers a larger area and the path planner uses a larger horizon, each computed trajectory will be executed only inside the execution perimeter. In order to account for locomotion errors during path execution, a control corridor is defined around the calculated optimal path. It will be used to supervise and secure the rover during path execution. To make sure that the rover will not enter any dangerous area, the control corridor has to be kept as well away from any forbidden region. This is translated in the navigation map by a dilation proportional to the width of the control corridor of all forbidden areas. Figure 2.8 provides an example of a navigation map and the corresponding legend to identify each represented region.

Figure 2.8: Example of navigation map representing the different types of areas

Path planning

This subsection provides a brief description of the path planning algorithm currently implemented in EDRES, which is regarded as a starting point of this thesis. Due to the limited coverage of the navigation map, the mission goal is not necessarily included in the covered area. Hence, the path planning stage starts with deciding a sub-goal position for the current locomotion cycle. The sub-goal is chosen to lie on the border of the navigation map, so that it maximizes the vehicle progression towards the final target. As the navigation map may consist of several navigable areas separated by forbidden regions, the selected goal position and the current rover position should belong to the same navigable field. Subsequently, the path planner finds an optimal path (with respect to traveled distance, roughness of the terrain and heading changes) to reach the sub-goal according to the following procedure:

- Run a greedy best-first search to examine the reachability of the selected target;
- Perform a post-processing smoothing step on the path provided by the greedy best-first search algorithm which will be used as a path primitive for the final path planner;
- Calculate the optimal path using an optimized version of the Best-First (A*) search algorithm, prioritizing cells closer to the path primitive.
Once the optimal path is calculated, the perception for the next locomotion cycle is planned. The perception plan consists of a set of attitude configurations of the stereo-bench to be used for stereo acquisition at the final position of the executed trajectory, where a new locomotion cycle will start.

Figure 2.9 provides an example for different stages of the path planning procedure. The current position of the rover is at the center of the navigation map and the sub-goal position lies at the bottom of the image on the border of the navigation map. The advantage of the greedy best-first search is that it can provide in very short time a sub-optimal path, as shown in Figure 2.9a. Here, due to the distribution of obstacles in the navigation map the path planning algorithm may be misled to search for possible paths in the dead-end configuration. This is expensive with respect to onboard available resources for an optimal path planner like A*. Therefore, the path provided by the greedy best-first search is smoothed, as shown in Figure 2.9b and used as a path primitive for the following step. Finally, the optimized best first search focuses on the areas around the path primitive to provide the final optimal path to be executed. Figure 2.9c shows the calculated optimal path divided in two parts by the execution perimeter marked with a dark grey circle. The first part, shown in white, represents the trajectory which will be executed during the current locomotion cycle. The second part, marked in red, shows the rest of the path calculated by the best-first search algorithm to reach the sub-goal. Finally the planning perimeter and the fields of view for the planned perceptions are shown in light blue.

![Figure 2.9: Path planning intermediary results](image)

2.2.3 Trajectory execution control subsystem

The last step of a locomotion cycle is performed by the trajectory execution control subsystem. The locomotion system of the rover is controlled to track the commanded trajectory accurately. The correctness of the drive is assessed using localization data.

**Locomotion control**

In EDRES, the trajectory to be executed is given in the form of successive waypoints. The control corridor is divided into three different areas, as shown in Figure 2.10, which decide the
trajectory execution policy, as follows:

- The interior corridor, where the locomotion error is minimal; the rover displacement is commanded at nominal velocity with locked heading (no steering commands);

- The intermediate corridor where the execution speed is decreased as the distance to the commanded trajectory increases; Also, when the difference between the current heading and the heading to the next waypoint becomes higher than a certain threshold, an arc turning is commanded until the desired direction is achieved;

- The outer corridor, where the same control policy as the intermediate corridor is applied, but with a supplementary test on the steering limits. If the limit is attained, the rover stops, performs an in-place rotation to correct its heading towards the next waypoint and resumes the drive.

If at any moment, due to an error or perturbation, the rover is placed outside the outer corridor, it stops and executes an in-place rotation until its heading is perpendicular to the control corridors. Then, it performs straight displacement until it reaches the interior control corridor, followed by another in-place turn to head towards the next waypoint. The in-place turns are considered to be very expensive commands as they increase the path execution time, the locomotion system wear and the localization error. This is the reason for which extensive work has been undertaken to predict and avoid them in the path planning process.

![Diagram showing interior, intermediate, and outer corridors with rover movements](image)

**Figure 2.10:** Representation of the locomotion control method

**Localization**

During displacements, the rover makes use of idiothetic and allothetic sources to localize itself with respect to the environment. This localization phase is very important for different stages in the locomotion cycle as follows:

- Navigation map generation, where consecutive perceptions are merged by using the rover’s estimated displacement;

- Path planning, for accurate coordinates of the start and goal positions;
Chapter 2. Development platform

- Locomotion subsystem for an accurate tracking of the rover position and orientation with respect to the commanded trajectory;

- Perception planning and execution, to precisely determine the end of the commanded trajectory to trigger the planned perceptions for the following locomotion cycle.

In the context of robotic planetary exploration, one very common localization approach like satellite navigation systems is not available. In such cases, dead-reckoning is a classical technique which estimates the displacement based on wheel revolutions count and angular velocity measurements, assuming no skidding. However, in real tests, this approach is subject to cumulative error leading to a position estimation drift of approximately 10%. Recently, a state-of-the-art localization technique based on visual odometry was developed and validated in EDRES, providing localization error inferior to 1% [Souvannavong et al., 2010]. It is entitled Visual Motion Estimation (VME) and it works according to the following procedure:

- Dedicated stereo cameras (VisOdom Cam) are used to acquire ground images.

- Robust features are selected in each image using an optimized version of Harris corner detector algorithm [Harris and Stephens, 1988] making sure that they are well distributed over the image.

- Following, the 3D positions of the selected features are estimated as explained in Section 2.2.1.

- Finally, as the rover moves, the features are tracked in subsequent stereo images and the true vehicle motion is estimated.

Even though the proposed VME technique provides good short term localization performances, it has increasing error over long runs. To address this issue, a local bundle adjustment approach is performed to limit the localization error.

2.3 Testing facilities

2.3.1 Mars-like test site

CNES disposes of two sites to perform ground tests for rovers, named SEROM (Site d’Essai pour les Rovers Mobiles). This facility is used to quantify the performances of the developed techniques and to assess the feasibility of the proposed planetary exploration missions. The main purpose is to reproduce specific conditions of a Martian environment, like traction, type of soil and roughness of the terrain. Thus, the soil was specially selected and is made of pozzolan and stones of various sizes and shapes. Its granularity simulates the behavior of a regolith soil exposed to a lower gravity in order to get closer to the adherence on Martian surface. The SEROM facility is composed of two sites:

- An outdoor terrain, the biggest outdoor Mars Yard in Europe, with a rough size of 80m × 50m. It is a very important asset for testing locomotion and perception systems of rovers on
Mars-like terrain. It consists of areas with variable densities (rocky, sandy) and geometries to meet the different constraints required by locomotion, perception or localization tests. It allows thus to perform tests on flat areas, areas with slopes, with surmountable or non-surmountable obstacles, or difficult configurations like canyons, dunes or valleys. A view of the CNES Mars Yard is given in Figure 2.11a. It can be seen that the Mars-like terrain is bounded by high slope areas, steeper than the authorized navigation slope for navigation, so that the rover will remain in this field while performing autonomous navigation.

- An indoor test site, smaller in size than the outdoor terrain (20m x 20m), shown in Figure 2.11b. It presents the same kind of heterogeneous areas and can be used for preliminary tests when the weather is not favorable. It has been used for validation tests of vision systems to evaluate the absolute precision for 3D scene reconstruction.

![Figure 2.11: SEROM test site facility](image)

2.3.2 Digital elevation models (DEM)s database

Newly developed navigation algorithms are initially tested and validated using the EDRES Robotic Simulator introduced in Section 2.1 and Mars-like terrain DEMs. This subsection introduces the DEMs which will be extensively used during experiments of the proposed methods throughout this thesis.

First of all, a high resolution DEM of the SEROM test site of CNES is available. Figure 2.12 represents the navigation map corresponding to the SEROM test site DEM when using the ExoMars rover model. The SEROM navigation map has a high percentage of navigable areas and few obstacles. The navigation map has a 25mm resolution and covers an area of 100m x 70m. It features a U shaped high slope which is the security limit of the SEROM test site. Only for simulation purposes, two virtual getaways have been added at the bottom of the U shape to avoid limiting the path planning solutions during simulations.

A DEM which comes from real Martian surface observation is also available in EDRES, as shown in Figure 2.13a. It has been built from image data acquired by the HiRISE camera.
Chapter 2. Development platform

onboard the Mars Reconnaissance Orbiter (MRO)\(^1\) scaled to the size of rover models used in EDRES. The corresponding navigation map, given in Figure 2.13b, has a predominance of large navigable areas at its right bottom, with important obstacles and slopes in the left top regions. It also contains several long continuous non navigable areas which are very important to prove the need of over the horizon path planning capabilities of the autonomous navigation software.

Finally the DEM used in the Astrium UK (ASU) study ”Rover Vehicle Navigation Design Report” is also used in this study. It has been generated using a statistical distribution for slopes and rocks of the real Mars surface. The calculated navigation map is mainly composed of navigable areas with no major difficult configurations for path planning tasks. The navigation map, shown in Figure 2.14, covers a squared region of 62.5\(m\) × 62.5\(m\) at 50\(mm\) resolution.

Table 2.1 provides a synthesis of the DEMs and navigation maps used for validation tests throughout this thesis. It presents the distribution of different area types for each DEM, providing a pseudo-measure for the difficulty of path planning and trajectory execution.

2.4 Robotic vehicles

2.4.1 IARES Rover

The IARES rover shown in Figure 2.15 (Autonomous Demonstrator for Space Exploration Robotics) was developed in 1996 for CNES for autonomous navigation and locomotion studies. It has a locomotion system with 19 degrees of freedom from which only 17 can be actively controlled. Each of the six wheels can have individually commanded rotational and angular

\(^1\)http://hirise.lpl.arizona.edu/dtm/dtm.php?ID=ESP_015985_2040
speed. Moreover it has the capacity to perform peristaltic movements by adjusting the inter-
axle distance and lifting one axle from the ground in order to climb high obstacles. Finally, the
center of inertia of the vehicle can be adjusted by sideways and forward/backward tilting. This
movement aims to provide better stability and grip on high slopes and to adjust the orienta-
tion of the stereo-bench mounted on the mast of the rover for improved perception. Table 2.2
provides the main characteristics of the IARES rover.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Total mass</td>
<td>170 kg</td>
</tr>
<tr>
<td>Width</td>
<td>120 cm</td>
</tr>
<tr>
<td>Length</td>
<td>85 – 135 cm</td>
</tr>
<tr>
<td>Height</td>
<td>150 – 210 cm</td>
</tr>
<tr>
<td>Wheel size</td>
<td>146 mm</td>
</tr>
<tr>
<td>Max. speed</td>
<td>35 cm/s</td>
</tr>
<tr>
<td>Max. slope</td>
<td>40°</td>
</tr>
<tr>
<td>Max. terrain step</td>
<td>50 cm</td>
</tr>
</tbody>
</table>

Figure 2.15: IARES rover

Table 2.2: IARES rover characteristics
Chapter 2. Development platform

Figure 2.14: ASU navigation map

<table>
<thead>
<tr>
<th>Area</th>
<th>SEROM</th>
<th>HiRISE</th>
<th>ASU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unknown area (map border)</td>
<td>3%</td>
<td>2%</td>
<td>4%</td>
</tr>
<tr>
<td>Navigable area</td>
<td>78%</td>
<td>62%</td>
<td>81%</td>
</tr>
<tr>
<td>Non-planifiable area due to slope</td>
<td>4%</td>
<td>8%</td>
<td>7%</td>
</tr>
<tr>
<td>Non-navigable area due to slope</td>
<td>6%</td>
<td>13%</td>
<td>4%</td>
</tr>
<tr>
<td>Non-planifiable area due to discontinuity</td>
<td>2%</td>
<td>5%</td>
<td>2%</td>
</tr>
<tr>
<td>Non-navigable area due to discontinuity</td>
<td>7%</td>
<td>10%</td>
<td>2%</td>
</tr>
</tbody>
</table>

Table 2.1: Navigation maps areas distribution

2.4.2 ExoMars Rover

As the ExoMars rover is currently under development, a model whose characteristics are detailed herein is used in simulation. Figure 2.16 illustrates the rover model with its scientific payloads. Similar to the IARES rover, the ExoMars rover has independently drivable and steerable wheels. It has a pair of navigation cameras (NavCams) placed approximately two meters above the ground level on the pan/tilt mechanism of its mast.

The wheels are mounted on three bogies, located on each side of the rover and on the rear (transverse bogie). The locomotion system is designed in this manner, such that it passively keeps all six wheels on the ground independent of the terrain shape. In addition the wheels are flexible in order to provided increased contact surface with the ground. The wheels’ diameter is 285 mm and their width is 120 mm. The rover’s nominal ground clearance is 30 cm and it is designed to drive over 25 cm step obstacles and over crevasses of 15 mm width. A summary of the ExoMars rover characteristics is provided in Table 2.3.
2.4. Robotic vehicles

Figure 2.16: ExoMars rover

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mass</td>
<td>300 kg</td>
</tr>
<tr>
<td>Width</td>
<td>150 cm</td>
</tr>
<tr>
<td>Length</td>
<td>170 cm</td>
</tr>
<tr>
<td>Wheel size</td>
<td>120 mm</td>
</tr>
<tr>
<td>Max. speed</td>
<td>70 m/h</td>
</tr>
<tr>
<td>Max. slope</td>
<td>20°</td>
</tr>
<tr>
<td>Max. terrain step</td>
<td>25 cm</td>
</tr>
</tbody>
</table>

Table 2.3: ExoMars rover characteristics

The locomotion system design of both IARES and ExoMars rover offer a high range of trajectory execution, as represented in Figure 2.17. The first locomotion configuration, the most often used one, is used to drive straight forward or backward or perform a turn on an arc-like path. Such a turn motion around a point situated outside the rover chassis is generated by controlling the steering and speed of the front and rear wheels, as shown in Figure 2.17a. However, the minimum turning radius (maximum turning curvature) is limited by the steering angle of the innermost wheels. Thus, important heading changes are performed using the second locomotion configuration, entitled "envelope configuration", represented in Figure 2.17b. Front and rear wheels are oriented so that their axes meet at the center of the rover. Following, wheel speeds are commanded so that left and right wheels roll in opposite directions. To avoid skidding, the speed of the middle wheels is also controlled. These two locomotion configurations were nominally used on Spirit, Opportunity and Curiosity rovers as they have fixed steering middle wheels.
Chapter 2. Development platform

A locomotion configuration unique for a planetary exploration rover is the crab driving mode, represented in Figure 2.17c. All six wheels are commanded to have the same steering, permitting the rover to perform a sideways diagonal translational displacement without affecting its heading. This drive mode is useful for efficient driving and accurate positioning near scientific targets.

Figure 2.17: Locomotion configurations
Chapter 3

Path planning architecture proposed by CNES for robotic exploration

This chapter provides a detailed description of the path planning techniques developed at CNES. First, the optimized successive local path planning approach is introduced. It aims to solve the path planning problem for robotic planetary exploration only using a local navigation map updated at each new perception, as described in Section 2.2.2. This approach is the backbone of this thesis and will be used as a reference for the performance assessment of methods proposed throughout following chapters. However, due to the limited coverage of the local navigation map, the successive local path planning approach may encounter difficulties when the rover has to traverse dense obstacle configurations or dead-ends. Further, the novel global path planning approach currently under development at CNES is presented. It will be also used to evaluate global path planning methods proposed in the last chapters of this thesis.

3.1 Path planning problem and notation

The main objective of path planning algorithms is to find a path from the current position of the rover to a selected goal, while minimizing a given cost. This path planning problem can be solved by performing a graph search over the state space. Let $S$ denote the finite set of states and $s_{\text{start}} \in S$ and $s_{\text{goal}} \in S$ represent the start and goal states of the current search. During path planning, a graph $G = (N, T)$ is built, where $N \subset S$ represents the set of explored states and $T$ is the set of validated transitions between the nodes of the graph. For each state $s \in S$, $\text{Pred}(s)$ represents the finite set of predecessor states (or parents) and $\text{Succ}(s)$, the finite set of successor states (or children). Also a finite set of actions $A(s)$ is defined which generate the finite set of possible transitions from state $s \in S$, noted $T(s)$. A transition is a 4-uple $t(s_i, s_j, a \in A(s_i), c(s_i, s_j)) \in T(s_i)$, where $s_i, s_j \in S$, $s_i \neq s_j$, $s_j \in \text{Succ}(s_i)$ and $c(s_i, s_j) \in \mathbb{R}_{>0}$ is the associated cost. The state space is represented by all states $s \in S$ and the corresponding transitions $t(s)$. The path from $s_{\text{start}}$ to $s_{\text{goal}}$ is defined as $P(s_{\text{start}}, s_{\text{goal}}) = \{s_1 = s_{\text{start}}, s_2, s_3, ..., s_n = s_{\text{goal}}\}, \{t_1, t_2, t_3, ..., t_{n-1}\}$ with the associated cost equal to $\sum_{i=1}^{n-1} c(s_i, s_{i+1})$. $P^*(s_{\text{start}}, s_{\text{goal}})$ denotes a cost-optimal path which has the minimum accumulated cost among...
all possible paths from $s_{\text{start}}$ to $s_{\text{goal}}$.

### 3.2 Grid representation

Approximate cell decomposition is one of the approaches that are commonly used in robotics path planning tasks [Ferguson, 2006] [Likhachev, 2005] [Koenig and Likhachev, 2002] [Ishida and Korf, 1995] [Koenig et al., 2007]. The most common way is to define a regular grid which decomposes the navigation space. This approach has high advantages from the software implementation point of view as the space is naturally mapped to array data structures and cell neighborhoods are implicitly encoded. On a grid representation, for a given state, one can define three main types of neighborhoods, as shown in Figure 3.1. The current state is highlighted with a dark grey cell and its neighborhood with bright gray cells. The choice of the neighborhood has a high impact on the time and memory use for the path planning task and on the quality of the final path as well. The neighborhood areas defined in Figure 3.1a and 3.1b consider a unitary radius around the central cell. However, one can define extended neighborhoods by increasing the radius, thus increasing the number of possible successors for a given state, as shown in Figure 3.1c.

![Figure 3.1: Neighborhood definition for cell decomposition](image)

Grid cell decomposition can be used to represent navigation space in different ways. One of the common approaches is to use a binary representation of the environment, tagging cells as obstacle or free space [Sun, 2013] [Tompkins, 2005]. Figure 3.2 gives an example of binary classification grid representation, where black areas represent obstacles (blocked for path planning) and white areas represent free (traversable) regions. It can be noticed that due to the approximation of the boundaries of the free space, cell decomposition approaches using binary terrain classification are incomplete. This leads to the fact that performing path planning on a grid representation cannot guarantee to find the minimum-distance path between two points. Moreover, if large areas are covered with the same type of terrain (obstacle or free space), grid representations become memory inefficient.
3.3 Sub-goal selection and validation

Another approach to represent navigation space is to assign a navigation value to each cell in the grid, in order to keep as much information as possible regarding the characteristics of the terrain. The navigation values correspond to the traversal difficulty of the represented region. In the case of a high resolution grid representation, it cannot be stated that the approach is memory inefficient as there is a low probability of homogeneous navigable areas representing natural environments.

The CNES path planning method uses a hybrid cell decomposition approach: homogeneous values for unknown, non-planifiable or non-navigable classified cells, and multi-valued representation which encode the ease-of-traversal of navigable areas. This has the advantage that the roughness of the terrain can be further used by the path planner in order to minimize the impact on the locomotion system of the rover and the difficulty of the traverse.

3.3 Sub-goal selection and validation

As mentioned in Section 2.2.2, the assigned mission target for the rover to reach is usually outside the navigation map due to its limited coverage. This is the reason for which the CNES autonomous navigation architecture uses successive locomotion cycles consisting in perception, navigation map building, path planning calculation and execution. Once the local navigation map is updated using the latest perception information, a sub-goal is selected according to the following criteria:

- the sub-goal cell has to lie on the border of a navigable region continuously connected to the rover location (so that a safe path from the rover position to the sub-goal exists)
- the distance between the sub-goal location and the imaginary line connecting the rover position and the mission target should be minimized.

Finally, the selected sub-goal position is validated if a greedy best first search algorithm is able find a path from the rover position. The algorithm creates a graph by expanding each state in the navigation map with the highest score of reaching the selected goal position. The score of a developed state is calculated using a heuristic evaluation function which estimates how close the search algorithm is to find a solution. At each moment, the state which is considered
to be the closest to the goal state is expanded first. When expanding a state different from the goal state, all successors of the current state are added in a list which keeps track of all remaining possible paths. Given two states $s_1, s_2 \in S$ in a grid representation, the Cartesian coordinates of the centers of the corresponding grid cells are defined, $(x_1, y_1)$ and $(x_2, y_2)$, with $x_1, y_1, x_2, y_2 \in \mathbb{N}$. The heuristic estimate between $s_1$ and $s_2$ marked as $h(s_1, s_2)$ approximates the distance between the centers of the two grid cells and it can be calculated using one of the measures in Table 3.1.

<table>
<thead>
<tr>
<th>Measure</th>
<th>Distance calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Euclidian</td>
<td>$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$</td>
</tr>
<tr>
<td>Manhattan</td>
<td>$</td>
</tr>
<tr>
<td>Octile</td>
<td>$</td>
</tr>
</tbody>
</table>

Table 3.1: List of heuristic estimations

The heuristic estimate of a state $s$ must be non-negative and provide an underestimate of the real cost of moving from $s$ to $s_{goal}$. It is called to be consistent with respect to $s_{goal}$ if it obeys the triangle inequality, such that $h(s_{goal}) = 0$ and $h(s_1, s_3) \leq h(s_1, s_2) + h(s_2, s_3), \forall s_1, s_2, s_3 \in S$.

While the Euclidean distance is the heuristic which is most commonly used in path planning tasks, one can use other measures which better comply with the discretization effect introduced by the cell decomposition. A heuristic measure is called "well-informed" when it provides an accurate estimation of the distance to goal. Thus, the Manhattan distance [Geem, 1994] is a "well-informed" heuristic when using a von Newmann neighbourhood and the octile distance [Zhang et al., ] in the case of Moore neighbourhood.

Algorithm 1 shows the pseudocode of the greedy Best First search. It is a simple algorithm which guarantees to provide a path between the two input states if one exists. In case the algorithm terminates with the message "no path found", there is no traversable path between the start and goal locations. Otherwise, in case the message "path found" is returned, one can extract the found trajectory by back-tracking the parent pointers from $s_{goal}$ towards $s_{start}$. Even if the provided trajectory is non-optimal, this procedure’s objective is to identify if at least one path exists which connects the given states at a complexity of $O(n)$. For a given configuration, Figure 3.3 shows an example configuration and the trajectory provided by the greedy BFS. It has to be mentioned that as suggested in Algorithm 1 [Line 13], the binary classification cell decomposition is used for this approach, as shown in Figure 3.3a. Following, the post-processed smoothed path is shown in red in Figure 3.3b. As mentioned in Section 2.2.2, Figure 3.3c marks in gray all states which were expanded during the current search, showing that the search algorithm analyzes all states in the dead-end configuration before a traversable path is found.

### 3.4 A* algorithm

After selecting and validating the sub-goal location, the A* algorithm [Hart et al., 1968] is applied to find the optimal path from the current rover state to the sub-goal. It performs a best-first search to find the least-cost path between two states in the navigation map. As shown...
Algorithm 1 Greedy Best First Search Algorithm

1: /* Search for a path from \(s_{\text{start}}\) to \(s_{\text{goal}}\) and return the result */
2: function BFS_search_path\(s_{\text{start}},s_{\text{goal}}\)
3: \(\text{OPEN} = \emptyset\)
4: \(\text{CLOSED} = \emptyset\)
5: \(\text{parent}(s_{\text{start}}) = \text{NULL}\)
6: \(\text{OPEN}.\text{Insert}(s_{\text{start}},h(s_{\text{start}},s_{\text{goal}}))\)
7: while OPEN \(\neq \emptyset\) do
8: \(s = \text{OPEN}.\text{Pop}()\)
9: if \(s = s_{\text{goal}}\) then
10: return "path found"
11: \(\text{CLOSED} = \text{CLOSED} \cup s\)
12: for all \(s' \in \text{Succ}(s)\) do
13: if \(s \in \text{Free Space}\) then
14: if \((s' \notin \text{OPEN})\) and \((s' \notin \text{CLOSED})\) then
15: \(\text{parent}(s') = s\)
16: \(\text{OPEN}.\text{Insert}(s',h(s',s_{\text{goal}}))\)
17: return "path not found"
18: end function

Figure 3.3: Cell decomposition for binary classified navigation space

in the pseudocode for the \(A^*\) algorithm (Algorithm 2), it maintains the following three values for every analyzed state \(s\): follows:

- The g-value \(g(s)\) represents the cost of the optimal path found so far to reach state \(s\) from \(s_{\text{start}}\); it is also called the start distance of state \(s\).

- The h-value \(h(s)\) is the estimation of the goal distance for vertex \(s\), as detailed in Section 3.2. \(A^*\) is an informed path planning algorithm which uses the heuristic estimate in order
to guide its search. For each state, it calculates the key, or the f-value \( f(s) = g(s) + h(s) \) which is the estimation of the total cost of traveling from \( s_{\text{start}} \) to \( s_{\text{goal}} \) through state \( s \).

- The parent pointer \( \text{parent}(s) \in \text{Pred}(s) \) is the state through which the current state \( s \) can be reached from \( s_{\text{start}} \) at a minimal cost. The parent pointers are used to extract the optimal path from \( s_{\text{start}} \) to any analyzed state.

The A* algorithm, similar to any BFS path planner (like greedy BFS in Algorithm 1) maintains two data structures, as follows:

- The OPEN list which keeps track of the states that are considered for expansion. It is implemented as a priority queue using the key values to sort the containing states. In the pseudocode of Algorithms 1 and 2, the \( \text{OPEN.Insert}(s, \text{key}) \), \( \text{OPEN.Update()} \) and \( \text{OPEN.Pop()} \) are list management functions for inserting, updating the position in the OPEN list regarding the updated key value and removing from the OPEN list the state with the smallest key respectively. A* ensures the cost-optimality of the provided final path by expanding at each iteration the state with the lowest total estimated cost.

- The CLOSED list contains states that A* has already expanded and thus have been removed from the OPEN list.

Initially, the OPEN list contains only \( s_{\text{start}} \) whose g-value is zero, and the CLOSED list is empty. Each time the search algorithm encounters for the first time a state [Line 37-38], its g-value is set to infinity, its parent is initialized with the NULL pointer [Lines 5-8], and the \text{UpdateState} routine is called. The main routine of A* is performed in a while loop [Lines 23-38]. At each iteration it extracts from the OPEN list the state \( s \) with the lowest f-value [Line 24]. The expansion of state \( s \) means to insert state \( s \) in the CLOSED list [Line 27] and to perform the following operations on all the neighbor states \( s' \in \text{Succ}(s) \). First it is checked if state \( s' \) belongs already to the OPEN list [Line 29]. If so, in case the total estimated cost through state \( s \) is smaller than the actual cost of state \( s' \) found so far, state \( s' \) is updated[Line 30-31]. If state \( s' \) is already in the CLOSED list, but a better estimated path cost through state \( s \) is found, then state \( s' \) is removed from the CLOSED list and updated [Line 33-35]. The \text{UpdateState} routine [Line 9-15] consists in setting the g-value of state \( s' \) to the g-value of state \( s \) plus the cost of traversing from \( s \) to \( s' \) and assigning the parent of \( s' \) to state \( s \). If state \( s' \) is already in the OPEN list, only its position in the priority queue is updated with respect to its new f-value returned by the \text{CalculateKey} function [Line 2-4]. Otherwise, it is inserted in the OPEN list by following the same routine. All this procedure is repeated until either the goal state \( s_{\text{goal}} \) is expanded [Line 24-26], or the OPEN list is empty [Line 23]. Expanding the goal state means that the search algorithm found an optimal path from \( s_{\text{start}} \) to \( s_{\text{goal}} \). The path is extracted by following in reverse order the parent pointers from \( s_{\text{goal}} \) to \( s_{\text{start}} \). In case the algorithm terminates due to an empty OPEN list, all possible paths have been checked but none reaches \( s_{\text{goal}} \), thus no path solution is available for the current configuration.

The difference between greedy BFS and A* is that greedy BFS uses only the heuristic value as key value for a given state, while A* considers both the real cost to reach the current state and the heuristic estimate. That is to say that greedy BFS considers the cost of moving from
3.5. Cost functions

The choice of a cost function has a high impact on the amount of the analyzed states during a path search, thus on the computation time and memory use.

A* has the following properties:

**Property 1** Correctness: Once the A* terminates, one can trace back the shortest path from \(s_{\text{start}}\) to any cell \(s \in S\) by following the parents back-pointers starting from \(s\). The extracted path is a cost-minimal path if \(s \in \text{CLOSED}\).

**Property 2** The \(f\)-values of the states which are chosen for expansion during an A* search [Line 24] are monotonically non-decreasing. Thus \(\text{key}(s) \leq \text{key}(s_{\text{goal}}), \forall s \in \text{CLOSED}\) and \(\text{key}(s_{\text{goal}}) \leq \text{key}(s), \forall s \in \text{OPEN}\) when the search terminates.

**Property 3** During a search, every expanded state \(s \in \text{CLOSED}\) has parent \((s) \in \text{CLOSED}\) and \(g(s) = g(\text{parent}(s)) + c(\text{parent}(s), s)\).

**Property 4** During a search, the A* algorithm builds up a search tree with the root at \(s_{\text{start}}\), each state \(s \in \text{CLOSED} \cup \text{OPEN}\) pointing towards its parent. For each state \(s \in \text{CLOSED}\), the search sub-tree rooted at state \(s \in S\) consists of \(s\), and any state \(s' \in S\) which is linked to \(s\) through a path (can reach \(s\) by following repeatedly the parent pointers of \(s'\)). Thus, when running an A* search on a grid representation, the generated \(\text{CLOSED}\) list represents a contiguous area. [Sun and Koenig, 2007]

**Property 5** The \(\text{OPEN}\) list contains only \(s_{\text{start}}\) if the \(\text{CLOSED}\) list is empty. Otherwise it contains all states \(s \in S\), \(s \notin \text{CLOSED}\) and \(\text{parent}(s) \in \text{CLOSED}\). That is to say that on a grid representation the \(\text{OPEN}\) list represents the contour of the contiguous area defined by the \(\text{CLOSED}\) list.

3.5 Cost functions

The cost function is an important factor for path planning as it guides the search and directly influences its performance. Two simple cost functions which are widely used in shortest path search algorithms on grid representations are the Manhattan distance and the Octile distance (Table 3.1). The Manhattan distance associates an unitary cost \(c(s, s') = 1\) to every edge which is added in the search tree, when the Von Neumann neighborhood is used. The result of a graph search when using this cost function is a path which minimizes the number of cell transitions. When performing graph search using the Octile distance, the cost associated to the transition from \(s\) to \(s'\) is the Euclidean distance between the center coordinates of each state, resulting a path that minimizes the traveled distance over the grid representation.

3.5.1 Traversability cost function

When performing path planning for rovers navigating in natural uneven terrain, a more realistic cost function encompasses the slope and discontinuities of the terrain where the rover navigates.
Algorithm 2  A* Algorithm

1: /* Search for a path from \(s_{\text{start}}\) to \(s_{\text{goal}}\) and return the result */
2: function CalculateKey(s)
3:    return \(g(s) + h(s, s_{\text{goal}})\)
4: end function

5: function InitializeState(s)
6:    \(g(s) = \infty\)
7:    parent(s) = NULL
8: end function

9: function UpdateState(s, \(s'\))
10:    \(g(s') = g(s) + c(s, s')\)
11:    parent(s') = s
12:    if \((s' \in \text{OPEN})\) then
13:        OPEN.Update(s', CalculateKey(s'))
14:    else
15:        OPEN.Insert(s', CalculateKey(s'))
16: end function

17: function Main()
18:    InitializeState(s_{\text{start}})
19:    \(g(s_{\text{start}}) = 0\)
20:    OPEN = \(\emptyset\)
21:    CLOSED = \(\emptyset\)
22:    OPEN.Insert(s_{\text{start}}, CalculateKey(s_{\text{start}}))
23:    while OPEN \(\neq \emptyset\) do
24:        \(s = \text{OPEN.Pop}()\)
25:        if \(s = s_{\text{goal}}\) then
26:            return "path found"
27:        CLOSED = CLOSED \(\cup\) s
28:        for all \(s' \in \text{Succ}(s)\) do
29:            if \((s' \in \text{OPEN})\) then
30:                if \(g(s') > g(s) + c(s, s')\) then
31:                    UpdateState(s, s')
32:            else if \((s' \in \text{CLOSED})\) then
33:                if \(g(s') > g(s) + c(s, s')\) then
34:                    CLOSED = CLOSED \(\setminus\) s'
35:                    UpdateState(s, s')
36:            else
37:                InitializeState(s')
38:                UpdateState(s, s')
39:            end if
40:        end for
41:    end while
42:    return "path not found"
43: end function
Thus the cost function is chosen as the sum of the terrain difficulty measure and the distance to travel, as shown in eq. 3.1.

\[ c_1(s_i, s_j) = (W_{\text{nav}i} \cdot \text{nav}_i + W_{\text{nav}j} \cdot \text{nav}_j) + \text{dist}(s_i, s_j) \tag{3.1} \]

The terrain difficulty measure is a weighted sum of the navigation values of the two states \((\text{nav}_i, \text{nav}_j)\). The weights \(W_{\text{nav}i}, W_{\text{nav}j}\) are penalty multipliers which decide how much the navigation value of a state influences the current cost function, where \(W_{\text{nav}i}, W_{\text{nav}j} \in [0, 1]\) and \(W_{\text{nav}i} + W_{\text{nav}j} = 1\).

### 3.5.2 Rover heading cost function

![Diagram](image.png)

Figure 3.4: Example of configuration providing multiple minimum-cost paths

Figure 3.4 emphasizes the drawback of using the traversability cost function over a homogeneous area. For the given configuration, the path planning would have multiple solutions for the lowest cost path. In black, yellow and red are marked all possible paths to be taken in order to reach state \(G\) from state \(S\) at the same cost. All cells shaded in gray are comprised in the so-called "equal cost area". However, even the paths have the same traversability cost, some of them should be considered as more difficult from the point of view of the locomotion system. For example, the wear of the locomotion system would be higher when executing one of the red trajectories than when executing the other proposed trajectories shown in black.

In order to overcome this drawback, the rover heading cost function tries to reduce in-place-turns as shown with arrows for the red trajectories. It is a one-step-look-behind function and gives advantage to Knight movements like in the chess game. By using this cost function in path planning, the yellow trajectory is selected which is the closest to the ideal solution marked in dashed gray.

First, the orientation of a state is defined in the following manner. A state \(s\) is said to have a diagonal orientation if the path between states \(s\) and \(\text{parent}(s)\) is a diagonal over the grid representation. Otherwise, state \(s\) has a straight orientation. Then, a new cost function
Chapter 3. Path planning architecture proposed by CNES for robotic exploration

augmented with the rover heading information is defined in eq. 3.2.

\[ c_2(s_i, s_j) = \begin{cases} 
(W_{nav_i} + W_{nav_j}) + \sqrt{5} - \text{dist}(s_i, \text{parent}(s_i)), & \text{if } \text{orientation}(s_i) \neq \text{orientation}(s_j) \\
(W_{nav_i} + W_{nav_j}) + \text{dist}(s_i, s_j), & \text{otherwise}
\end{cases} \]  

(3.2)

3.5.3 Guided path planning cost function

In the method proposed by CNES, the path provided by the greedy BFS algorithm is smoothed and given to the A* algorithm as a primitive path. The use of a primitive path in A* limits the search tree size, thus results in lower computation time and memory use. Let \( P \) denote the primitive path, the guided path planning cost function is given in eq. 3.3. It encompasses the cost function calculated using eq. 3.2 and the distance from the current state to the primitive path. This cost function is used throughout all the tests undertaken in this thesis, if not stated otherwise.

\[ c_3(s_i, s_j) = c_2(s_i, s_j) + \text{dist}(s_j, P) \]  

(3.3)

3.6 Long range navigation

Recently, a new technique for global path planning for planetary exploration rovers was developed at CNES. It aims to build a global navigation map covering large areas explored by the rover during a sol, which is further used to provide a better informed sub-goal to the optimal local path planner, as detailed in Section 3.3.

In order to limit the memory use, the global navigation map, entitled throughout this thesis obstacle map, contains only the high resolution contours of all obstacles encountered during the mission. Figure 3.5 provides and example of an obstacle map, where the stored obstacle contours are shown in red. The local navigation map used to update the obstacle map is also shown to highlight the different coverage provided in each case. As well, it should be noted that the information regarding the traversability of the terrain is discarded in the obstacle map representation. This reduces the memory use and simplifies the global path planning process, considering that all the space outside the stored contours is navigable with the same difficulty.

Further, the LRN approach builds a partial tangent graph over the obstacle map in order to decide if the mission goal is reachable, and if so, it provides a more informed sub-goal for the local path planning stage. It is based on the tangent graph proposed by [Liu and Arimoto, 1992], where states are represented by common tangent points on obstacle contours and transitions between states are collision-free common tangents between obstacle contours and convex contour segments linking tangent points. However, this approach becomes costly in dense obstacle fields, as the tangent graph requires \( O(K^2) \) memory, \( K \) representing the total number of convex segments on the obstacle contours. In order to comply with the mission-constraints of the robotic planetary exploration mission, the LRN builds only a part of the tangent graph. The main procedure consists of the following steps:
3.6. Long range navigation

Figure 3.5: Example of obstacle map with superposed local navigation map

1. Initialize the start state of the search algorithm with the current rover position
2. Identify the obstacle contours in the obstacle map which intersect the straight line from the start state to the mission goal
3. Select the intersected obstacle contour which is the closest to the start state and compute its tangents from the start and goal position
4. Among the generated 4 tangent points, select the one which is closest to the mission goal to become the new start state of the search algorithm
5. Repeat the procedure from Step 2, until the mission goal is achieved.

Figure 3.6 gives an example of path planning using the partial tangent graph algorithm. At any stage of the search procedure the start state is marked with a red circle, while the mission goal is shown in yellow. Given the initial setting shown in Figure 3.6a, the leftmost obstacle contour is selected and the corresponding tangents are calculated (Figure 3.6b). At each step, the green line represents the straight line from the current start state and the mission goal, white and gray lines represent tangents from the start and goal states to the selected obstacle, and the validated path from the rover position to the current start state is shown in purple. After the start state is updated in Figure 3.6c, no other obstacle intersects the straight line towards
Chapter 3. Path planning architecture proposed by CNES for robotic exploration

Figure 3.6: Partial tangent graph path planning using the obstacle map

the mission goal, and the final path is given in Figure 3.6d. Despite the fact that this procedure provides global paths with low computational loads, it has been shown in a recent Monte Carlo study that in 44% of the cases the given path is sub-optimal. Indeed, such a case is shown in Figure 3.6, where a shorter global path can be found only using the tangents to the rightmost obstacle.

3.7 Summary

This chapter provides a detailed description of the path planning architectures developed at CNES for autonomous planetary exploration rovers. A mission target is attained through successive locomotion cycles, where optimal local paths are planned and executed. The main difference is represented by the way the sub-goal in the local navigation map is chosen. In the first approach, referred as Successive Local A* (SLA*) throughout this thesis, the local sub-goal is chosen on the border of the updated local navigation map as close as possible to the imaginary straight line connecting the current rover position and the mission goal location. However, the limited coverage of the navigation map, and the rough information used for the sub-goal selection might lead to failures in reaching the mission goal.

The second approach, entitled Long Range Navigation (LRN), uses a global navigation map built during the mission and a partial tangent graph algorithm to provide a global sub-optimal
3.7. Summary

path towards the mission goal. Following, the sub-goal for the local path planning step is chosen on the given global path, inside the area covered by the local navigation map. This way, the global path planning problem is solved with minimum onboard resource requirements.

Both path planning approaches represent the backbone of the work in this thesis, and will be used as reference for the performance analysis of the proposed path planning methods.
Chapter 3. Path planning architecture proposed by CNES for robotic exploration
Chapter 4

Incremental path planning

This chapter suggests improvements for the current local path planning approach used by CNES. As for security reasons, the rover executes only a part of the planned path, incremental path planning techniques are suggested to fasten the path search procedure while sharing navigation data between consecutive path planning stages. Thus, the Fringe Retrieving A* is used under the assumption of static navigation environment. Following, the use of the binary heap data structure is proposed for the management of the priority queue of the search algorithm. Both approaches are implemented in EDRES of CNES and their performance tested with respect to the current approaches.

4.1 Real time algorithms

Real-time search algorithms have been extensively proposed for path planning tasks where a path has to be calculated in a fixed time bound, known as ”the real-time property” [Korf, 1990] [Bulitko and Lee, 2006] [Sturtevant and Bulitko, 2011].

Since the introduction of the Learning Real-Time A* algorithm [Korf, 1990], many variants were proposed to solve agent centric real-time path planning problems where a distant goal has to be reached. The agent centric path planning refers to the limitation of each planning step in a bounded state space around the robot’s current position. The principle of the Learning Real-Time-A* is to perform successively local searches and trajectory execution, at the risk of decreasing the overall performance. This is due to the fact that the goal position is unreachable during most of the local path planning tasks, leading to situations where the robot gets stuck in dead-end configurations.

The Learning Real-Time A* algorithm assumes that the final target is static. It interleaves path planning and execution stages towards the target by taking into account the environment configuration only in a limited lookahead region around the current rover location. The main principle of this approach is to build a look-up table which contains heuristic estimates of each state in the search space to the goal location. This relies on the condition that the heuristic estimate is admissible, meaning that it represents the lower bound of the actual distance from the state to the target. Through repeated exploration of the search space, more accurate heuristic estimates are learned until the exact distance values are eventually found.
Chapter 4. Incremental path planning

An extension of the Learning Real-Time A* algorithm for cases where the goal state evolves in time is given by the Moving Target Search algorithm [Ishida and Korf, 1995]. This approach is a generalization of the Learning Real Time A* search, where heuristic estimations between any pairs of states are learned. This aims to learn accurate heuristic estimates for any possible configuration of the start and goal positions in the search space. In order to tackle the possible memory load to store all the distance estimates, a sparse representation is used to keep only updated values which are different from the initial heuristic estimate.

The Trailblazer Search [Chimura and Tokoro, 1994] minimizes the number of search iterations of Moving Target Search at the cost of the real-time property. It follows the same procedure as Moving Target Search, while storing in a graph the trajectories of the robot and the target. Once the two trajectories intersect, the path to be followed is provided by running the Dijkstra algorithm [Dijkstra, 1959] on the built graph. As Dijkstra algorithm cannot guarantee bounded planning time, the real-time property of the Tailblazer Search is affected.

The quality of the calculated path solutions using real-time search algorithms can be affected by the limited lookahead areas and the repetitive learning process of the heuristic estimates. Moreover, the learning process can become slow as the heuristic estimates are calculated using inaccurate neighboring distance approximations [Sturtevant and Bulitko, 2011].

4.2 Incremental search algorithms

Online path planning techniques outperform offline approaches which consider all possible contingencies in the state space before a path is given for execution to the rover. The principle behind online path planning strategies is to calculate in a loop a local path using the available navigation data and move along it until changes in the state space are perceived or the target is achieved. However, the quantity of cost changes perceived in the environment can be limited [Koenig et al., 2004a], and hence replanning paths from scratch at each perception could become inefficient. The objective of incremental search algorithms is to fasten the path planning process by reusing previously computed search data. Mainly, incremental search algorithms are proven to be efficient when planning over a large state space with little navigation data changes over time. In such cases, the amount of information which can be shared between consecutive searches is maximized [Koenig et al., 2004b].

The concept of incremental path planning dates back in the 90s, where several incremental search approaches are introduced in a classification of shortest path planning algorithms [Deo and Pang, 1984]. Since then, new techniques of incremental path planning been proposed. [Ausiello et al., 1990] and [Feuerstein and Marchetti-Spaccamela, 1993] address the all-pairs shortest path problem in a directed graph, which can handle the cost changes of its edges. Others propose dynamic algorithms for solving single-source minimum distance path planning [Frigioni et al., 1996] [Franciosa et al., ]. Various assumptions regarding the performance measures, the graph topologies and edge costs (and how they change over time) are presented in [Italiano, 1988], [Goto and Sangiovanni-Vincentelli, 1978], [Klein and Subramanian, 1998]. Recent techniques refer to incremental heuristic search algorithms based on A* path planning, validated for a wide range of robotic applications [Likhachev et al., 2005b], [Urmson et al., 2008], [Maxim Likhachev, 2008], [Likhachev et al., 2008].
4.2. Incremental search algorithms

4.2.1 Heuristic learning incremental search algorithms

Heuristic learning incremental search algorithms use information computed during previous searches to update the heuristic estimation function. Thus, as the heuristic values become more informed, the subsequent path planning steps become more focused and reach the goal position faster. The Adaptive A* [Koenig and Likhachev, 2005a] performs repeated A* online searches for stationary target position in order to update the h-values to be more consistent. Thus it guarantees to find minimal cost paths in a state space where the traversal costs are constant or increase over time. However, in case the traversal costs decrease, the consistency property of the updated h-values can be violated.

Even though the Moving Target Adapted A* [Koenig et al., 2007] extends the capabilities of Adaptive A* to moving-target search problem, it does still have the drawback of not handling costs which decrease between searches. It can either perform successive forward A* searches (Forward Moving Target Adaptive A*) where the start state represents the current rover position and the goal state corresponds to the target location, or subsequent backward A* searches (Backward Moving Target Adaptive A*) where the assignation of the start and goal states is inverted.

Generalized Adaptive A* [Sun et al., 2008] (GAA*) overcomes the disadvantages of both Adaptive A* and Moving Target Adaptive A*. It is able to solve search problems where both start and goal states are not static and the traversal costs can change (increase or decrease) between consecutive plans. Table 4.1 provides a summary of the capabilities of the presented heuristic learning incremental search algorithms [Sun, 2013].

<table>
<thead>
<tr>
<th>Heuristic learning incremental algorithm</th>
<th>Traversal cost can increase between searches</th>
<th>Traversal cost can decrease between searches</th>
<th>Start state can change between searches</th>
<th>Goal state can change between searches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptive A*</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>MT-Adaptive A*</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GAA*</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 4.1: Capabilities of heuristic learning incremental search algorithms

4.2.2 Search tree reusing incremental search algorithms

The other type of incremental search algorithms is represented by search tree reusing techniques. Instead of starting each path planning from scratch, this approach aims to try to restore information from previous searches to be used for the current search. Some of the well known algorithms which belong to this class include Dynamic SWSF − FP algorithm [Ramalingam and Reps, 1996], Differential A* [Trovato, 1989], Dynamic A* (D*) [Stentz, 1995], Lifelong Planning A* (LPA*) [Koenig et al., 2004a], its generalized version D* lite [Koenig and Likhachev, 2002] and its interpolation based extension Field D* [Ferguson and Stentz, 2005a].
Chapter 4. Incremental path planning

The D* lite algorithm represents the state-of-the-art incremental path planning algorithm widely used in robotic applications. It has the capability of providing the minimum cost path in a dynamic unknown environment, where either the goal position or the start state are fixed. It is often used for global path planning tasks where running Repeated A* searches each time a traversal cost change is identified would be very expensive. The D* lite dynamically updates the search tree when new data on the environment is available. Global path planning using D* lite will be reviewed and tested in Section 6.

Extensive work has addressed the limitations of D* lite when applied to moving target path planning. Several algorithms have been proposed, such as Generalized Fringe-Retrieving A* (G-FRA*) [Sun et al., ], Fringe-Retriving A* (FRA*) [Sun et al., 2009], Basic Moving Target D* lite (Basic MT-D* Lite) and Moving Target D* lite (MT-D* lite) [Sun et al., ].

G-FRA* is able to perform incremental search in a known static environment, where start and goal states can change between searches. While G-FRA* is designed to plan on state lattices, FRA* is an extension to work on cell decomposition representations. Both algorithms require that the current rover position remains in the search tree provided by the previous search. This allows the current search process to reuse the part of the previous search tree rooted at the current start state. However, if the start state falls outside the region covered by the previous search tree, no information can be reused and an usual A* search is run for the current iteration. It has been shown that FRA* and G-FRA* have better performances than Repeated A* and GAA* for moving target search problems in known static environment, with reduced computation times up to one order of magnitude [Sun, 2013].

A generalized version of D* lite for search problems with moving start and goal states in dynamic unknown environments is proposed by Basic MT-D* lite. Its optimized version which applies also the principles of G-FRA* is entitled MT-D* lite. Both approaches perform forward searches and have the same requirements as FRA*.

4.3 Fringe-Retriving A*

Fringe-Retriving A* is an incremental search algorithm which makes use of geometric properties specific to grid representations. It performs repetitive forward A* searches, and uses previously computed information in the current search as long as the current robot state belongs to the previous search tree.

4.3.1 Principle

Given a mission goal to be reached, the first path planning iteration is identical to an usual A* search. After a part of the computed path is executed and new perception data is taken, a new sub-goal is selected on the locally updated navigation map. Instead of performing a new A* search from scratch, the FRA* algorithm transforms the previous search tree into an initial search tree for the current A* search. This implies to determine the initial OPEN and CLOSED lists, including the cost values and the parent pointers of the states in them. Once Properties 4, 5 for the initial OPEN and CLOSED lists hold again, the current A* search is launched. This approach fastens the path planning process by avoiding to expand twice the states in the search.
tree.

4.3.2 Operations

The pseudocode of the FRA* algorithm is provided in Algorithms 3 and 4. Similar to the A* algorithm, the FRA* builds an OPEN and CLOSED list containing all added and expanded states during a path search. Both lists will be further used to be transformed into the initial search tree for the current search. The OPEN list is called to be complete if it satisfies Property 5, otherwise it is incomplete. The information regarding the completeness of the OPEN list is indicated by the variable open_incomplete [Lines 58, 68, 69]. Since the OPEN list satisfies Property 5 after each search, the open_incomplete variable is set to false after each ComputePath() call. Otherwise, it is set to true after restoring the OPEN and CLOSED list from the previous search [Line 68], as this can result in an incomplete OPEN list. Similar to the A* algorithm, FRA* terminates after the goal state is expanded and added to the CLOSED list [Line 16].

The first FRA* search is identical to an A* search. It builds a search tree from scratch in order to find a cost optimal path from the current start state to the current goal state. At the beginning of each search, the g- and h-values of all unvisited states \( s \in S \) are initialized by the InitializeState() function. Once a path is found, the robot starts executing a part of the provided trajectory. When a new perception is available, the robot updates the navigation map and the actual position of the goal. If the goal state has changed, a new FRA* search is performed by running an A* search on initial OPEN and CLOSED lists calculated from the previous search tree rather than from scratch.

A detailed description of the steps performed by the FRA* algorithm is provided below:

- **Step 1: Starting A* immediately:** This step is executed if for some reason the robot did not move between two consecutive iterations, that is \( s = s' \). In this case, the OPEN and CLOSED lists generated during the latest call of ComputePath() function [Line 56] are used as the initial lists. Depending on the current goal state, the following cases are identified:

  - **Case 1 [Lines 59-61]:** If the current goal state lies in the range of the initial CLOSED list, then a minimum cost path from the current start state to the current goal can be identified (in conformity with Property 1) and executed.
  
  - **Case 2 [Lines 72-73]:** If the current goal state does not belong to the initial CLOSED list, execute an A* search with the initial OPEN and CLOSED lists. If the goal state has changed since the last call of ComputePath(), before starting the search, the \( h^- \) and \( f^- \) values of all states contained by the OPEN list are updated correspondingly [Lines 72-73].

- **Step 2: Deleting States:** First of all, the anchor state is defined by the parent pointer of the current state in the previous search tree. It will be used further on at Step 4 when filling in the restored OPEN list in order to satisfy Properties 4 and 5.

All states in the initial CLOSED list for the current search have to satisfy Property 4 with respect to the current start state. While all states which belong to the search sub-tree
Chapter 4. Incremental path planning

Algorithm 3 Fringe Retrieving A* Algorithm (Part 1)

1: function InitializeState(s)
2:    if generated_iteration(s) \neq current_iteration then
3:        g(s) = \infty
4:        generated_iteration(s) = current_iteration
5:        parent(s) = NULL
6: end function

7: function ComputePath()
8:    while OPEN \neq \emptyset do
9:        s = OPEN.Pop()
10:           CLOSED = CLOSED \cup s
11:           for all s' \in Succ(s) do
12:               if s' \notin CLOSED then
13:                   InitializeState(s')
14:                       if g(s') > g(s) + c(s, s') then
15:                           UpdateState(s, s')
16:                   if s = s_{goal} then
17:                       return "path found"
18:               return "path not found"
19: end function

20: procedure Step2()
21:    parent(s_{start} = NULL)
22:    for all s \in S in the subtree rooted at s_{oldstart} do
23:        parent(s) = NULL
24:        if s \in OPEN then
25:            OPEN.Delete(s)
26:        if s \in CLOSED then
27:            CLOSED.Delete(s)
28: end procedure

29: procedure Step4()
30:    for all s \in S on the outer perimeter of CLOSED, starting with anchor do
31:       if s is navigable AND s \notin OPEN AND \exists s' \in Pred(s) : s' \in CLOSED then
32:           OPEN = OPEN \cup s
33:    for all s \in OPEN do
34:       InitializeState(s)
35:       for all s' \in Pred(s) do
36:           if s' \in CLOSED AND g(s) > g(s') + c(s', s) then
37:              UpdateState(s', s)
38: end procedure
Algorithm 4 Fringe Retrieving A* Algorithm (Part 2)

39: \textbf{procedure} Step5()
40: \hspace{1em} \textbf{for all} \ s \in OPEN \ \textbf{do}
41: \hspace{2em} h(s) = h(s, s_{goal})
42: \hspace{2em} OPEN.Update(s, CalculateKey(s))
43: \textbf{end procedure}

44: \textbf{function} Main()
45: \hspace{1em} current\_iteration = 1
46: \hspace{2em} s_{start} = \text{the current state of the rover}
47: \hspace{2em} s_{goal} = \text{the current state of the goal/sub-goal}
48: \hspace{2em} \textbf{for all} \ s \in S \ \textbf{do}
49: \hspace{3em} generated\_iteration(s) = 0
50: \hspace{2em} \text{InitializeState}(s_{start})
51: \hspace{2em} g(start) = 0
52: \hspace{2em} OPEN = \emptyset
53: \hspace{2em} CLOSED = \emptyset
54: \hspace{2em} OPEN.Insert(s_{start}, CalculateKey(s_{start}))
55: \hspace{2em} \textbf{while} Mission goal not reached \ \textbf{do}
56: \hspace{3em} \textbf{if} ComputePath() = path not found \ \textbf{then}
57: \hspace{4em} \text{return} "false" \hspace{3em} \triangleright \text{The mission goal cannot be reached}
58: \hspace{3em} \text{open\_incomplete} = false
59: \hspace{3em} \textbf{while} s_{goal} \in CLOSED \ \textbf{do}
60: \hspace{4em} \textbf{while} mission goal not reached AND target on path from s_{start} to s_{goal} \ \textbf{do}
61: \hspace{5em} Follow path from s_{start} to s_{goal}
62: \hspace{4em} \textbf{if} Reached mission goal \ \textbf{then}
63: \hspace{5em} \text{return} "true"
64: \hspace{5em} s_{oldstart} = s_{start} \hspace{2em} \triangleright s_{start} = \text{current state of the robot}, s_{goal} = \text{current state of the goal/sub-goal}
65: \hspace{5em} \textbf{if} s_{start} \neq s_{oldstart} \ \textbf{then}
66: \hspace{6em} anchor = \text{parent}(s_{start})
67: \hspace{6em} Step2()
68: \hspace{6em} open\_incomplete = true
69: \hspace{5em} \textbf{if} open\_incomplete = true \ \textbf{then}
70: \hspace{6em} current\_iteration = current\_iteration + 1
71: \hspace{6em} Step4()
72: \hspace{5em} \textbf{else}
73: \hspace{6em} Step5()
74: \hspace{5em} \text{return} "true"
75: \textbf{end function}
Chapter 4. Incremental path planning

rooted at the current start state hold this property, states which are not in the sub-tree do not satisfy this condition. Thus, the aim of this step is to determine all states in the previous CLOSED and OPEN lists which belong to the sub-tree rooted at the current start state.

First, FRA* sets the parent of the current start state to NULL [Line 21], in order to disconnect the sub-tree rooted to the current start state from the rest of the previous search tree. Following, FRA* iterates over all states remained in the sub-tree rooted at the previous start state [Lines 22-27] to set their parent pointers to NULL [Line 23]. Finally, those states are removed from the OPEN and CLOSED lists [Lines 25,27]. Figure 4.1b visualizes the OPEN and CLOSED lists computed in Step 2. The dotted line represents the deleted states from the previously computed OPEN list, while the solid line represents the states remaining in the initial OPEN list, which are part of the sub-tree rooted at the current start state. The area in yellow represents the deleted states from the previously computed CLOSED list, while states in the green area are the ones remaining in the initial CLOSED list.

• Step 3: Terminating early: This step is similar to Case 1 of Step 1. The difference is that between two consecutive searches the start state has changed and the goal state belong to the initial CLOSED list. Comparing to Step 1 where the OPEN list is complete, here only a part of the previous search tree is restored and the OPEN list should be completed. However, as the goal state belongs to the initial CLOSED list, a minimal cost path can be already extracted for the current configuration.

• Step 4: Inserting States: The initial OPEN list restored from the previous search tree has to satisfy Property 5, so that it contains all states on the outer perimeter of the restored initial CLOSED list and states in it have their parents in the initial CLOSED list. This stage of FRA* makes use of geometric properties specific to grid representations. According to Property 4, the initial CLOSED list represents a contiguous area. Thus, FRA* fills up the initial OPEN list by circumnavigating the outer perimeter of the initial CLOSED list starting with the anchor state. Each encountered navigable state, which does not belong to the initial OPEN list, is inserted in the initial OPEN list if it has at least one predecessor in the initial CLOSED list. After performing Step 3, the initial OPEN list becomes complete. Further on, the $h-$ and $f-$ values of all states in the initial OPEN list are updated in case the goal state has changed [Line 73].

• Step 5: Update $h-$ values: FRA* performs this step only when the start state remains unchanged between two consecutive searches. Otherwise, if the goal state changed after perception (Step 1, Case 2), the new $h-$ and $f-$ values of all states in the initial OPEN list have to be updated [Lines 40-42].

• Step 6: Perform A* search: Once the initial OPEN and CLOSED lists are determined, the A* search for the current configuration is performed to find the new optimal cost path to be executed[Line 56].
4.3. Fringe-Retrieving $A^*$

![Diagram of Fringe-Retrieving $A^*$ operations]

Figure 4.1: Operations of Fringe-Retrieving $A^*$

4.3.3 FRA* Example

For a better understanding of the working principle of FRA*, a short example is provided on a binary classified navigation environment as shown in Figures 4.2, 4.3 and 4.4. Cells in white represent navigable states and the ones in black obstacles. Figure 4.2a shows the initial configuration with the initial starting position at cell D1 in yellow (marked S) and in red the goal position to be reached at cell D5 (marked G). The robot navigates in initially unknown static terrain and it can perceive the occupancy of cells in a window of width equal to 3 cells. Figure 4.2b marks in grey the cells which fall in the field of view of the robot when located at cell D1, and the obstacle cells discovered during the first perception. As in the current configuration the mission goal is not in the field of view of the robot, a sub-goal for the current iteration is selected at cell B4 marked in orange.

![Initial configuration and first perception]

Figure 4.2: FRA* example initial configuration

Following, FRA* performs its first A* search from D1 to B4. The cost function for the path planning algorithm is equal to 1 for each translation, and 4-connectivity is used in the grid representation.

As detailed in Algorithm 3 and 4, the $g$, $h$ and $f$ values are calculated for all analyzed states. All this information and the search iteration at which a state was analyzed are displayed in Figure 4.3a. The parent pointer for each node is represented with an outgoing arrow towards the parent cell. During the search process, FRA* breaks ties between cells with equal $f$-values.
in by favoring the states with higher $g$-values. Figure 4.3b displays the results for the first FRA* search, where cells in grey represent the expanded cells which belong to the CLOSED list. Cells in white are the cells in the OPEN list after the search has finished. The rover extracts the optimal cost path from the search tree represented in Figure 4.3c and moves along it from cell D1 to D2. Then, the rover performs a new perception. As the mission goal position D5 becomes visible, there is no need to define a new sub-goal. Thus, the current search is performed with start state cell D2 and goal state cell D5.

As the start state changed between the two searches, the conditions for running Step 1 are not fulfilled. During Step 2, a part of the search tree built during the previous iteration is restored. First the anchor cell is defined to be D1 (the parent cell of the current start state). Next, the parent pointer of the current state D2 is set to NULL, and all states which do not belong to the sub-tree rooted at state D2 are deleted. Thus, states E1 and B1 are removed from the initial OPEN list and states D1 and C1 are deleted from the initial CLOSED list. The results of this stage are shown in Figure 4.4a, where parent pointers are removed for the deleted states.

For the current configuration, FRA* cannot terminate early (Step 3) as the goal state has changed between the two searches. Hence, Step 4 is performed and the initial OPEN list is completed so that Property 5 is fulfilled. It starts circumnavigating the outer perimeter of the initial CLOSED list starting from the anchor cell D1. Then D1, C1 and B1 are inserted in the initial OPEN list with their correct $g-$, $h-$, $f-$ values and the corresponding parent pointers. Figure 4.4b shows the initial OPEN and CLOSED lists which will be used in the current search, and Figure 4.4c displays the final results obtained after the second A* search. During Step 6, only seven cells E2, D1, E3, C1, B5, C5, D5 are expanded in order to find the optimal path from the current start state to the mission goal, while an A* search would expand 13 cells.
4.4 Implementation in the EDRES environment

4.4.1 OPEN list management optimization

As described in Algorithm 2, the OPEN list contains all states which have not yet been developed during the current search. The A* algorithm guarantees to find the minimum-cost path between two nodes by developing at each step the most promising route, represented by the state with the lowest priority key [Line 24]. For this purpose, a priority queue is used for the management of the OPEN list.

Double-linked ascending priority queue

The current priority queue implementation used by CNES uses a double-linked ascending priority queue. It is a data structure whose elements can be inserted in a random order and the position of newly inserted elements is determined based on the priority-key value, so that at any moment all elements are positioned in increasing order of their priority-key values. Thus, this data structure is able to provide at any moment the lowest-cost state for further development by the A* algorithm. Each priority queue provides three basic operations: inserting a new node with its corresponding key value [Lines 15, 22], minimum-cost node extraction [Line 24] and key and position update [Line 13]. Figure 4.5 provides a visual example for inserting a new node in the priority queue. Figure 4.5a displays the environment configuration with B2 the start state and A5 the goal state, marked in blue. All states framed with green belong to the OPEN list as given in Figure 4.5b. At the current step, node B3 is added to the priority queue, being the last neighbor of state B2 to be analyzed. Initially, it is added at the end of the current OPEN list (Figure 4.5c). Then, its priority key value is compared to the one of the previous state in the priority queue. Figure 4.5a displays the environment configuration with B2 the start state and A5 the goal state, marked in blue. All states framed with green belong to the OPEN list as given in Figure 4.5b. At the current step, node B3 is added to the priority queue, being the last neighbor of state B2 to be analyzed. Initially, it is added at the end of the current OPEN list (Figure 4.5c). Then, its priority key value is compared to the one of the previous state in the OPEN list. If it has a lower key value, the positions of the two states are swapped. This procedure continues until the right position of the current node in the priority queue (Figure 4.5d) is reached.
Chapter 4. Incremental path planning

Binary heap

Onboard path planning for planetary exploration rovers is constrained by the limited computation power and time costs. A more efficient data structure to be used to manage the priority queue of the A* algorithm is that of binary heaps. It has been extensively used for robotic path planning algorithms [Koenig and Likhachev, 2002], [Ferguson and Stentz, 2005b], [Koenig et al., 2007], [Sun et al., 2009],[Sun et al., 2010], [Uras et al., 2013].

The main properties of the binary heap data structure are the following:

- A binary heap is organized as a binary tree and uses an array as underlying representation.

- The **heap property** - Each node in the tree structure has the priority key lower than its children (also called *min-heap*).

- The **shape property** - All levels of the tree (except possibly the last one) are fully filled.

Both insert and delete operations for binary heap trees modify initially the shape property. During an insert operation, the new node initial position is the end of the tree, while during a delete operation, the emptied position is filled with the last node of the tree. Following, the tree is traversed upwards or downwards, starting with the new state, in order to restore the
4.4. Implementation in the EDRES environment

Heap property. Figure 4.6a provides the equivalent binary heap representation for the initial OPEN list presented in Figure 4.5. Node B3, marked in blue, is first added at the end of the array representing the heap tree (Figure 4.6b). Then, its key value is compared with that of its parent, state B1. As the heap property is violated, the two states swap positions. This operation is called heap percolate and it will be further used for performance analysis of priority queue management. When the heap property is fulfilled, the inserting process stops (Figure 4.6c). It has to be noticed that the insertion process for state B3 took a single heap percolate for the binary heap representation, while it needed five heap percolates in the case of the double-linked ascending list representation. Table 4.2 gives a comparison of time complexities associated to each elementary operation for the two data structures, where \( N \) denotes the number of states in the list.

<table>
<thead>
<tr>
<th>Priority queue</th>
<th>Insert/Update Key</th>
<th>Delete min</th>
<th>Delete state</th>
<th>Find min</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double-linked ascending</td>
<td>( O(N) )</td>
<td>( O(1) )</td>
<td>( O(1) )</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>Binary heaps</td>
<td>( O(\log N) )</td>
<td>( O(\log N) )</td>
<td>( O(\log N) )</td>
<td>( O(1) )</td>
</tr>
</tbody>
</table>

Table 4.2: Comparison of time efficiency regarding the used data structure for the priority queue

The underlying array representation of the binary heap data structure is given at each stage of the example in Figure 4.6. The root of the binary heap is state A3 which has the minimum key and it lies on the first position of the array. At any moment, for a given state \( s \), one can recover its parent or children nodes by using the following formulas:

- \( Children(s) = State(2 \times \text{Index}(s)), State(2 \times \text{Index}(s) + 1) \)
- \( Parent(s) = State[\text{floor}(\text{Index}(s)/2)] \)

4.4.2 Previous search sub-tree restore methods

The successive path planning approach with a limited range navigation map proposed by CNES demands a supplementary stage when applying FRA*. The local navigation map is always centered at the current rover location. A supplementary shifting operation is required to restore the previous search sub-tree rooted at the current start position.

Figure 4.7 illustrates the steps to be taken between two consecutive path planning stages. Figure 4.7a displays the OPEN list (in red), the CLOSED list (in green) and the computed path (in blue) after an A* iteration using the start and goal positions highlighted by the yellow and red squares. As mentioned in Section 3.4, the CLOSED list forms a contiguous area and the OPEN list is its contour. After a part of the given path is executed, as represented in dark orange in Figure 4.7b, the new start position is set for the following A* search. The reusable search subtree rooted at the new start position is also highlighted in this figure: the NEWCLOSED list in magenta and the NEWOPEN list in blue. Due to the displacement of the local navigation map with the rover, the reusable search subtree is restored and shifted with the new start position. The NEWOPEN list is filled in by applying Step 4, as shown in Figure 4.7c. Finally, having a new goal position shown by the red square in Figure 4.7d, a new
Chapter 4. Incremental path planning

Figure 4.6: OPEN list management binary heap priority queue

(a) Initial binary heap tree

(b) New node added at the end of the binary heap tree

(c) Final binary heap tree (after insertion of node B3)
A* search is performed using the restored information in \textit{NEWCLOSED} and \textit{NEWOPEN} lists and a new path is found, as shown in blue.

Concerning the implementation of Step 2 to restore the subtree rooted at the current rover location, [Sun, 2013] suggests the use of a BFS which maintains a first-in-first-out (FIFO) queue. Initially the FIFO queue contains only the previous start position, shown with a yellow square in Figure 4.7a. Then, BFS repeatedly removes state $s$ from the FIFO queue, and adds all states $s' \in \textit{Succ}(s)$ with $\text{parent}(s') = s$ to the queue. When state $s$ is removed from the FIFO queue, it is also removed from the previous \textit{OPEN} or \textit{CLOSED} list [Line 20-28 in Algorithm 3]. However, removing all states which belong to the previous \textit{OPEN} list but not rooted at the current start location (states in red in Figure 4.7b) can become expensive since the number of states in the previous \textit{OPEN} list is usually much smaller than the number of states to be deleted.

Thus, two approaches are proposed for the implementation of Step 2 in this thesis. The first approach uses the technique proposed in [Sun, 2013] which creates the \textit{NEWOPEN} list by deleting the outdated states from the previous \textit{OPEN} list. Note that after this restoring process, the \textit{NEWOPEN} list cells should be shifted so that the restored subtree is rooted at the center of the new navigation map. In this thesis this approach is entitled Classic Subtree Restore (CSR).

The second approach, entitled Optimized Subtree Restore (OSR), proposes an optimized way of reusing the previous search tree to calculate the initial \textit{OPEN} and \textit{CLOSED} lists. The pseudocode for this proposed approach is provided in Algorithm 5. The procedure starts...
Chapter 4. Incremental path planning

with two empty lists named NEWOPEN and NEWCLOSED [Lines 2,3]. Then, a BFS is performed on the subtree rooted at the current rover location and all information regarding the encountered states \((g-, h-, f-\) values and parent pointers) are copied to a corresponding state after the shift from the previous start to the current start locations [Line 5]. It is then inserted in the NEWOPEN or NEWCLOSED lists depending on the membership of state \(s\) [Lines 6-9]. Finally, the information contained in the OPEN and CLOSED list is replaced by the NEWOPEN and NEWCLOSED lists. The OSR approach runs faster than the CSR because less heap percolate operations are performed for the generation of the NEWOPEN list.

Algorithm 5 Optimized subtree restore (OSR)

1: procedure Step2()
2: NEWOPEN = \(\emptyset\)
3: NEWCLOSED = \(\emptyset\)
4: for all \(s \in S\) in the subtree rooted at \(s_{\text{start}}\) do
5: \(\text{copy-navigation\_information(OPEN, NEWOPEN, s, s_{\text{new}})}\)
6: if \(s \in OPEN\) then
7: \(\text{NEWOPEN.Insert(s_{\text{new}})}\)
8: if \(s \in CLOSED\) then
9: \(\text{NEWCLOSED.Insert(s_{\text{new}})}\)
10: OPEN = NEWOPEN
11: CLOSED = NEWCLOSED
12: end procedure

4.5 Experimental evaluation

4.5.1 OPEN list management performance analysis

The performance of the successive path planning approach using the aforementioned two priority queue representations is compared in the context of path planning for planetary exploration rovers navigating in unknown natural terrain. A simulation study is conducted by running the rover simulator provided by the CNES EDRES environment on the Mars HiRISE DEM. The experimental results are averaged over 100 mission scenarios with randomly assigned mission goal positions with total traveled distance per mission ranging from 60m to 160m. During this experimental campaign, 3250 locomotion cycles were performed with local path planning on the updated local navigation map, as described in Section 2.2.2.

Figure 4.8 reports four measures for the difficulty of path planning query, namely the number of added nodes and of developed nodes in the A* algorithm per search iteration (Figures 4.8a and 4.8b). These numbers are the same for both cases using different priority queue representations, as the functionality of the A* algorithm is not affected. It is shown that an average of 6000 insertion operations in the OPEN list are performed, while approximately 10000 states are inserted in case of difficult path planning configurations. However, only an average of 3000 states are developed and added to the CLOSED list, while a maximum of 5000 developed states
4.5. Experimental evaluation

can be reached. Following, the number of heap percolates in the OPEN list per search iteration is reported (Figures 4.8c and 4.8d).

The computation time performance for planetary exploration applications is very important, as it has a direct influence on the mission evolution and can limit the distance the rover can drive during a sol. Figures 4.8e and 4.8f report the performance regarding the computation time per search on a Linux PC with an Intel Xeon 3.60 GHz × 8 CPU and 16 GB RAM. First the raw run time per A* search when using either a double-linked ascending list (timeDLAL) or a binary heap tree representation (timeBH) for the priority queue is compared. It is clearly shown that the use of binary heap significantly reduces the computation time. For a better visualization, the histogram of "the time gain" measure defined in eq. 4.1 is displayed in Figure 4.8f. This proves that the use of a binary heap data structure brings an important computation gain, with an average of 60% over the current data structure, which is an important asset for space robotic exploration missions.

\[
\text{time}\text{gain} = \frac{\text{timeDLAL} - \text{timeBH}}{\text{timeDLAL}} \quad (4.1)
\]

The influence of the priority queue management on the time performance given by the A* algorithm is also studied in the case of global path planning, for a navigation map which covers the distance an exploration rover should drive over a Martian sol. Using the HiRISE navigation map, 2000 trajectories are generated with random start and goal coordinates and global A* path planning requests are performed. The resulting paths have lengths between 5\(m\) and 140\(m\).

Figure 4.9 provides a time comparison for global path planning queries. First, in Figure 4.9a is shown that the computation time has a proportional increase with respect to the length on the global path. However, the configuration of the navigation space has a higher influence on the performance of a grid path planning algorithm. Therefore, more states are added in the OPEN list and expanded when the planned path has to avoid dense obstacle fields of difficult configurations like dead-ends. Figures 4.9b and 4.9c show that when using a double-linked ascending priority queue, the computation time has an exponential growth tendency with respect to the number of added and developed states. On the other hand, the binary heap representation provides a linear computation time increase in the same conditions. Thus, the use of a binary heap priority queue management is crucial for an optimal use of the computation resources.

4.5.2 FRA* path planning evaluation

This section details the experimental study performed in order to assess the performance improvement when using incremental search algorithms. The FRA* algorithm was implemented, tested and validated in the CNES EDRES environment. Table 4.3 provides a summary of the performed tests. All three navigation maps introduced in Section 2.3.2 were used for planing and executing missions with randomly generated coordinates for the start and target positions. A total of approximately 198 trajectories have been tested, covering distances between 5\(m\) and 104\(m\) and running more than 3300 path planning iterations. The resolution used for the SEROM navigation map is 25\(mm\), while for the other navigation maps it is 50\(mm\) per pixel. This difference influences the results for the path planning iterations as it will be discussed further on.
Figure 4.8: Performance comparison for the priority queue management
4.5. Experimental evaluation

![Graphs showing computation time comparison](image)

Figure 4.9: Computation time comparison for global path planning with respect to: (a) the length of the global path, (b) the number of added states in the OPEN list, (c) the number of expanded states.

<table>
<thead>
<tr>
<th></th>
<th>SEROM</th>
<th>ASU</th>
<th>HiRISE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Navigation map resolution</td>
<td>25mm</td>
<td>50mm</td>
<td>50mm</td>
</tr>
<tr>
<td>Mission tests</td>
<td>55</td>
<td>51</td>
<td>92</td>
</tr>
<tr>
<td>Search iterations</td>
<td>1019</td>
<td>1071</td>
<td>1214</td>
</tr>
<tr>
<td>Minimum traveled distance</td>
<td>8.71m</td>
<td>6.45m</td>
<td>5.02m</td>
</tr>
<tr>
<td>Maximum traveled distance</td>
<td>82.54m</td>
<td>101.97m</td>
<td>103.82m</td>
</tr>
</tbody>
</table>

Table 4.3: Experimental set summary

Figure 4.10 illustrates an example for the FRA* search sub-tree restore operation. Figure 4.10a shows the obtained search tree over a grid representation at Nth iteration. For each state in the search tree the outgoing arrows point towards the corresponding parent cells. The same color code as in Figure 4.7 was used. The computed path shown in blue is given to the rover for execution. Due to locomotion errors, the rover current location after execution, marked in orange, can fall in a different state than planned. Figure 4.10b shows the resulting position when an accurate path execution is performed. After locomotion, the rover current state falls on the previous calculated minimum-cost path (cell B5) and the cells marked in gray can be restored for the next FRA* search. However, if the current rover state after execution falls off the previous optimal path (cell B4), as shown in Figure 4.10c, the available sub-tree for the further FRA* search is only a branch of the search tree calculated during the previous FRA* search. Such a case is called a *non-validated restore* because the restore operation is considered to be sub-optimal. Implementation-wise a validated restore operation can be easily identified, as the rover current location should fall on the previous minimum-path which enables to maximize the size of the restored search sub-tree.

The comparison for the results provided by the successive A* path planning approach of CNES and by the incremental path planning FRA* is shown in Table 4.4. The measure regarding the efficiency includes the average number of added and developed states per search. Regarding the time performance, the count of heap percolates for the binary heap management of the
Figure 4.10: A graphical representation of validated and non-validated FRA* restore operations.

The main objective was to evaluate the applicability of the proposed incremental search path planning algorithm for robotic navigation over natural, rough terrain. In the case of the SEROM navigation map, only 21.15% of the FRA* iterations were initialized using previously calculated
4.6 Conclusions

This chapter evaluated two main directions for improvement of the current path planning algorithms developed by CNES. First, the use of the binary heap data structure for the management of the OPEN list of the A* algorithm is proposed and assessed, compared to a double-linked ascending priority queue currently used in EDRES. It is shown that the use of the binary heap data structure provides an average of 60% computational load reduction. This is due to the fact that the double-linked ascending priority queue is a fully sorted list, while the binary heap data structure keeps track of only the state with the minimum key. It is further shown that the double-linked ascending priority queue has an exponential growth tendency of the computation time as the dimension of the OPEN list increases. In the same conditions, the binary heap data structure has a linear computation time increase, proving that its use is crucial for an optimal use of the computation resources.

Following, it is shown that when using the Fringe Retrieving A*, a high amount of navigation information can be shared between two consecutive path planning stages providing a significant improvement of the computational load. The quantity of reused navigation information depends on the resolution of the navigation map and on the length of the executed part of the computed path during the previous path planning procedure. It is shown than an important time gain can be reached due to restored search sub-tree and thus decreased number of heap percolates. Finally, two methods are proposed for the search sub-tree retrieval between two consecutive path planning stages, where an optimized version provides an important computation time improvement through a reduction of heap percolates per iteration of approximately 90%.

However, the sole disadvantage of an incremental path planning approach is that its performance is highly influenced by the relation between the resolution of the navigation map and the locomotion error of the rover. In order to maximize the amount of navigation data which can be reused between two consecutive path planning procedures, the rover has to accurately follow search trees. This is due to the high rate of non-validated restore operations as an effect of the higher resolution of the navigation map. The amount of validated restore operations increases to 50.69% for the ASU navigation map and 70.29% for the HiRISE navigation map. In the case of validated restore operations, the highest rate of restored OPEN and CLOSED states is given for trajectories on the HiRISE map. The average restored states per iteration is given for each map, showing in parenthesis the maximum rate of reused nodes.

Finally, two methods (CSR and OSR) are assessed to restore the search sub-tree rooted at the current rover location when using the FRA* search. As expected, the proposed optimized approach (OSR) is more efficient for restoring the same search sub-tree comparing to CSR. This is due to the fact that CSR performs operations directly on the previously calculated OPEN and CLOSED list. Thus, it removes all unnecessary states from the OPEN list resulting in higher counts of heap percolates during the restoring process. This is hinted by the highest gain obtained in the case of the SEROM map of 89.3% when using OSR. As in this case the search sub-tree to be restored is smaller due to the higher resolution of the navigation map, more states are removed from the OPEN list during the restore process generating a higher amount of heap percolates.

4.6 Conclusions
the provided paths.
Chapter 5

Nonholonomic motion planning

This chapter is devoted to the application of constrained path planning techniques for a planetary exploration rover. The main objective is to generate paths which incorporate the steering capabilities of the rover in order to fasten the trajectory execution and to reduce the rover locomotion system wear. First, a state lattice is defined, that is discretized in all state parameters of interest such as position, heading and curvature. It provides a precomputed control set which encodes the motion capabilities of the rover. Then, an adapted version of the A∗ path planner is employed to find efficient, feasible motions through the navigation space. The focus is to evaluate the constrained path planning performance with respect to mission-constraints of a planetary exploration rover, given the exponential growth in complexity due to the increased dimension of the search space.

5.1 State of the art

Recently, a significant interest has been dedicated to the problem of path generation under model motion constraints. The basis of path planning for differentially constrained vehicles was set by [Dubins, 1957] and [Reeds and Shepp, 1990]. Their idea was further developed to generate smooth paths by using segments of clothoids (curves with the curvature as a linear function of their length), arcs or straight lines [Scheuer and Laugier, 1998] [Fraichard and Alhuactzin, 2001] [Fraichard and Scheuer, 2004]. The focus is on generating paths which can be followed by a car-like vehicle. In order to do so, three constraints are added besides the classical kinematic constraint: the curvature of the path must remain continuous and be upper-bounded (respecting the minimum turning radius constraint), and the curvature derivative must be bounded (as the car-like vehicle can reorient its directing wheels at a limited speed) [Scheuer and Fraichard, 1997] [Lamiraux and Lammond, 2001]. While different approaches suggest that the problem of smooth obstacle-free path generation can be solved analytically [Frazzoli et al., 2001] [Bicchi et al., 2002], fast numerical optimization techniques are used for path generation for nonholonomic vehicles navigating in rough terrain [Kelly and Nagy, 2003] [Howard and Kelly, 2007]. This approach takes into account the roughness of the navigation environment, vehicle dynamics and wheel-terrain interaction models while optimizing an objective function with respect to risk, length, time or energy consumption.
Chapter 5. Nonholonomic motion planning

Research has been conducted for motion path planning methods which construct boundary representations of configuration space obstacles [Lozano-Pérez and Wesley, 1979] [Lozano-Perez, 1983] [Reif, 1979], and their algorithm complexity has been evaluated [Alt et al., 1990] [Canny, 1988].

An initial interest was shown for algorithms based on deterministic sampling [Barraquand and Latombe, 1991]. Some proposed methods of path generation by sets of steering velocity commands and evaluated them in forward model dynamic simulation [Lacaze et al., 1998]. Moreover, a two-level motion planning based on physical modeling of the vehicle, terrain and their interaction is proposed in [Cherif, 1999]. The high planning level performs a waypoint planning in a low dimensional subset of the vehicle C-space with relaxed locomotion constraints, while the second planning level provides locomotion-constrained feasible paths to navigate between the selected waypoints.

Since early 1990s randomized sampling was introduced in solving motion constrained path planning [Barraquand and Latombe, 1990]. First, Probabilistic Roadmap methods (PRM) were proven to perform well in C-spaces with many degrees of freedom [Hsu and Dept, 2000] [Kavraki, 1995] [Kavraki et al., 1996] or with complex constraints [Casal and Yim, 1999] [Kindel, 2002] [Kuffner and Dept, 1999]. The idea behind PRM is to perform random sampling of the robot configuration space and to build a graph which connects all the sampled configurations in the free space. However, as it cannot handle path planning with differential constraints, the Rapidly-exploring Random Trees (RRT) algorithm was introduced [Lavalle et al., 2000]. Even though randomized approaches are considered to be incomplete, they proved to solve complex problems efficiently when using quasi-PRM and regular lattice roadmap (LRM) methods with low discrepancy Halto/Hammersley sequences [Branicky et al., 2001].

Recent research studied "lazy" versions of roadmap planning methods [Bohlin and Kavraki, 2000] [Bohlin, 2001] [Sánchez and claude Latombe, 2001] [Sanchez and Latombe, 2002]. First, the roadmap construction is performed considering that the entire navigation space is navigable. Further on, the path which connects the given start and goal configurations is calculated and validated after collision checks. In case of invalidation due to encountered obstacles, the invalid configurations are removed from the roadmap and a new path is checked. In this manner the initially built roadmap can be used for multiple settings of the start and goal configurations. Moreover, [Branicky et al., 2001] proposes a version where the initial graph is not explicitly represented. This approach uses an implicit lattice for performing motion planning on a precomputed control set which describes only local connectivity. In this regard, a high interest was shown to develop motion primitives which represent feasible motion controls under kinematic or dynamic constraints. The role of motion primitives in the context of robot navigation with obstacle avoidance capabilities was presented in [Branicky et al., 2008] [Green and Kelly, 2007]. When applied to deterministic approaches, the precomputed motion primitives set provides a set of edges that emanate from a given state during its expansion [Barraquand and Latombe, 1991] [Donald et al., 1993] [Go et al., 2006] [Pivtoraiko and Kelly, 2005] [Likhachev and Ferguson, 2009].

The autonomous ability to plan obstacle-free drivable paths is of high interest for planetary exploration rovers as it has a direct influence on the mission resource management and the scientific return. The work presented in [Pivtoraiko et al., 2008] proposes two approaches: one
refers to the generation of feasible motion plans by inverse dynamic (kinematic velocity) model optimization and it is further used to build a finely discretized state lattice (which captures the full maneuverability of the rover) to be used by standard search algorithms. Another way to generate paths is based on cubic curvature polynomials which take into account the rover geometry, driving back capabilities, grid resolution and computational cost [Guixé et al., 2012]. However, up to date only one method has been validated and extensively used to generate constrained motion plans for planetary exploration rovers [Goldberg et al., 2002]. It uses a predefined set of 23 forward and 23 backward circular arcs of varying radius and two point turns for evaluating at each planning step the best path to be followed.

5.2 Path planning on MERs

The autonomous navigation software onboard the Mars Exploration Rovers (MERs) of NASA uses an architecture entitled GESTALT (Grid-based Estimation of Surface Traversability Applied to Local Terrain) [Maimone et al., 2004]. Similar to the CNES autonomous navigation architecture, it uses onboard stereo vision to evaluate terrain safety and to perform autonomous path selection [Biesiadecki et al., 2005].

The Terrain Assessment (or Predictive Hazard Detection) module performs one or more stereo image acquisitions to obtain range data. Then, range data is analyzed for potential geometric hazards by applying thresholds for the maximum terrain tilt and maximum traversable obstacle size. The most conservative scores are distributed into a local traversability or goodness map centered at the rover (50 × 50 cells for Spirit and 60 × 60 cells for Opportunity), with 20 cm cell resolution. It uses a configuration space representation, meaning that each cell in the map represents whether a rover-sized object centered over it can navigate safely. The goodness map keeps track of the accumulated navigation data over time, over the covered bounded area. It is also updated based on the FIFO principle, that is, new data overwrites old data in the map.

If the Path selection sub-module is activated, the rover can autonomously select a path among a fixed precomputed set of candidate steering paths as given in Figure 5.1a. The rover is heading up in this view and each blue grid line indicates 1 m spacing. Each path is projected on the goodness map and a corresponding traversability score is calculated as the weighted sum of the goodness values of all affected cells. The projection of a sub-set of the precomputed paths is represented in Figure 5.1b, where red cells represent unsafe areas (around the rock), yellow and orange cells indicate traversable areas but which are more difficult to traverse and green cells illustrate flat areas. The higher the accumulated score for a given path candidate, the easier to drive the path.

Independently from the nature of the terrain, the same set of precomputed paths receives a reachability score with respect to the actual configuration of the rover and the desired goal location. Thus, the motion primitive which directs the rover the best towards the goal receives the highest reachability score which is the peak of a Gaussian function applied to the entire set of trajectories. The variance of the Gaussian curve is a parameter of the GESTALT system, entitled vote-index-variance [Goldberg et al., 2002].

Following, the traversability and the reachability scores for each candidate path are merged in a conservative manner. If any of the values is below a threshold, the minimum score is
chosen. Otherwise a weighted sum of the two score is computed. The path with the highest merged score is considered as the best path candidate and is selected for execution. However, if even the highest merged score is below a threshold, new perceptions are requested in the direction opposite to the goal and the candidate paths in this direction are evaluated. If after the direction change, the highest merged score is still below a threshold, no path is chosen for execution and the drive stops.

![Figure 5.1: Terrain Assessment and Path Selection Modules of the GESTALT Navigation Planning](image)

(a) Set of the 96 precomputed paths consisting in arc and point turns along with straight line drives [Biesiadecki et al., 2005]

(b) Projection of precomputed paths over the goodness map [Biesiadecki and Maimone, 2006]

### 5.3 Path planning using motion primitives for the ExoMars rover

This section addresses the problem of performing path planning using motion primitives in the context of a planetary exploration rover. It has to be noted that, in the case of the GESTALT navigation architecture, the set of precomputed paths restricts the navigation possibilities of the rover. Thus the number of headings the rover can reach after path execution is limited. Furthermore, the rover cannot perform trajectories with inflection points when avoiding close obstacles.

The aim of this section is to develop a precomputed path set which can be reused by the rover in successive planning stages so that paths with a higher complexity than simple arcs or straight lines can be performed. Although the literature proposes many probabilistic approaches, this work explores the regular state lattice case produced by deterministic sampling mechanisms.
One property of this approach is the translational invariance which is a crucial advantage of using the same control set to connect different pairs of states. The connectivity of the search space can be encoded by considering no obstacles and the precomputed control set of the state lattice can be stored in a compact way.

An initial approach to define a motion primitive set is based on the Dubins and Reeds-Shepp car [Dubins, 1957] [Reeds and Shepp, 1990]. Here, the control set is made of three primitives as follows: one straight line with the current heading of the rover and two arcs at the minimum turning radius on each side of the heading direction. This configuration considers also the backward corresponding arcs, which are not feasible for planetary exploration rovers. Figure 5.2 displays the control set and the corresponding path planning result as proposed by [Dubins, 1957].

![Motion primitive set](image)

Figure 5.2: Example for path planning using the Dubins motion primitive set [Pivtoraiko et al., 2008]

A more representative nonholonomic path planner widely used in robotic applications [Baker et al., 2004] [Morris et al., 2005] was proposed by Barraquand and Latombe [Barraquand and Latombe, 1991]. It mainly integrates the forward kinematic model of the rover to generate motion primitives. The proposed BL planner iteratively builds a search tree which keeps track of all reachable configurations up to a depth of $H$ arcs. The set of outbound arcs for each configuration contains a straight line and two arcs with steering angles $\theta \in (-\pi/2, +\pi/2)$. Figure 5.3 shows the reachability tree built using the BL planner over five exploration levels. The control set is displayed over a grid with a resolution of 50mm, with the arcs’ length equal to 400mm and the minimum turning radius set to 1m. The grey shade of an arc represents its depth in the reachability tree, from bright the first level to dark the last one.

The disadvantage of the BL planner is that it does not work on a regular search space. That is, the reachability tree is not position invariant and cannot be used to further extend the coverage of the search tree. In order to reduce memory requirements and computational time, this thesis proposes a reduced regular state lattice which implements a discretized version of the BL planner. Once computed, position-invariant motion primitives can be further used in the path planning process to extend the search tree at minimum cost while including the motion constraints of the rover.
5.3.1 State lattice design

The state lattice encodes the motion capabilities of the robotic system. The regularity property and position invariance allows to represent the entire navigation space of the rover as a compact set of motion controls localized at any reachable position. The list of parameters needed to define the compact state lattice is given as follows:

- **Minimum turning radius**: It refers to a kinematic constraint on the smallest radius of a circular path that the rover can follow, avoiding in-place-turn maneuvers.

- **Branching factor**: It represents the number of motion controls leaving from a given state. It usually contains a straight line and arcs of circle with radius values covering homogeneously the steering capabilities of the robotic system.

- **Translational discretization**: The resolution of the grid representing the navigation space environment.

- **Heading discretization**: The resolution of the heading direction of the rover.

- **Primitive length**: The distance the rover would drive on a given path primitive (straight line or arc).

In this thesis, the regular state lattice is calculated by following a two stage procedure:

1. Conventional forward motion control set generation which takes into account the locomotion capabilities of the rover and the desired control set density given by the aforementioned parameters.
2. Inverse state lattice generator providing a smooth and feasible path between two discrete states of the navigation environment. This operation grants for the desired translational and regularity properties of the state lattice.

**Forward motion control set generation**

The objective of this first procedure is to generate a continuous set of controls so that the rover executes a circular path. Given an initial heading $\theta_0$ and position $(x_0, y_0)$, the rover is expected to travel a certain distance $L$ (primitive length) on a curve of radius $R$. This displacement is the result of a mixture of linear (forward) and rotational motion controls. For simplicity, constant linear and rotational velocities are used execute arc of circle trajectories. In result, the state space can be expressed by a 3-dimensional variable $(x, y, \theta)$.

![Circular arc displacement for a robotic system](image)

Figure 5.4: Circular arc displacement for a robotic system

Figure 5.4 provides an example for generating a circular path. Having the initial state of the system $(x_0, y_0, \theta_0)$ providing the Cartesian coordinates and the angle between the robot’s main axis and the positive x-axis expressed in radians, the final system state $(x_f, y_f, \theta_f)$ is determined after traveling a distance $L$ on a circle of a given radius $R$. By definition, the value of angle $\theta$ in radians is proportional to the arc length and inversely proportional to the radius of the circle, as given in eq. 5.1.

$$\theta = \frac{L}{R} \quad (5.1)$$

The heading of the rover at the end of the trajectory is given by $\theta_f = \theta_0 + \theta$. Once the final
heading is given, one can compute the final Cartesian coordinates as given in eq. 5.2.

\[ x_f = x_0 - R \sin(\theta_0) + R \sin(\theta_f) \]
\[ y_f = y_0 + R \cos(\theta_0) - R \cos(\theta_f) \]  
(5.2)

An example of generated arc paths using this procedure is presented in Figure 5.3. In the bottom left of the image, a zoomed area of the reachability tree is given to show that arcs generated in such manner are not suited to be applied in a discretized environment using precomputed motion primitives because it does not necessarily reach the center coordinates of grid cells.

**Inverse state lattice generator for discrete states**

As shown in Figure 5.3, the coordinates and orientation \((x_f, y_f, \theta_f)\) at the end of the arc do not fall exactly on the cell center in the grid decomposition of the navigation environment. The objective of this second stage is to calculate a motion control sequence which links two discrete states in the navigation space, denoted by \((x_{d0}, y_{d0}, \theta_{d0})\) for the initial and \((x_{df}, y_{df}, \theta_{df})\) for the final state. The final state is composed of the cell which contains \((x_f, y_f)\) calculated in the first procedure and the discretized value of the final heading \(\theta_f\). The coordinates \((x_{d0}, y_{d0})\) and \((x_{df}, y_{df})\) represent the center Cartesian coordinates of the corresponding grid cells and \(\theta_{d0}\) and \(\theta_{df}\) are the discretized heading values of each state.

![Figure 5.5: Trajectory decomposition](image)

This procedure connects the two discrete states with a line segment starting at the initial state followed by an arc segment reaching the final state as shown in Figure 5.5. Let \((x_l, y_l)\) be the last point of the line segment, as represented in eq. 5.3. This point has to satisfy both
5.3. Path planning using motion primitives for the ExoMars rover

Equations of the straight line and of the arc, as shown in eq. 5.4, where $R$ is the circle radius of the arc.

$$
x_l = x^d_0 + l \cos(\theta^d_0)
$$
$$
y_l = y^d_0 + l \sin(\theta^d_0)
$$
$$
\theta_l = \theta^d_0
$$

(5.3)

From eq. 5.3 and eq. 5.4 one can determine the length of the straight line $l$ and the radius of the arc of circle $R$ by solving the equation expressed in matrix form as given in eq. 5.5.

$$
B = AX,
$$

(5.5)

The system presented in eq. 5.5 has a unique solution, $X = A^{-1}B$ when $\theta^d_0 \neq \theta^d_f$.

Figure 5.6 displays in red five outgoing continuous arcs of a fixed length of 800 mm and radius between 1 m and 5 m for a given grid cell at three initial heading values: 0 degrees, 30 degrees and 45 degrees. The corresponding motion primitives composed of a straight line and an arc of circle are shown in black. It has to be noticed that the motion primitives start and end at the centers of the grid cells, assuring the regularity property for the state lattice. The given example uses a grid decomposition resolution of 50 mm, and a branching factor of 5 arcs per heading. All cells marked with grey background represent the projection of the arcs over the grid.

Control set generation

As an overview, this section provides a practical formulation of the proposed algorithm of control set generation which is further used during the path planning process. Considering a fixed resolution grid decomposition of the navigation environment, a primitive control set for an origin position is generated. For a given grid position $(x^d, y^d)$, the control set is generated taking into account the parameters regarding the length and radius of the curved paths, all the set of discretized headings and branching factor. Further on, due to the regularity property all the control set generated at the origin can be reproduced at any other position in the grid.

The procedure for generating the control set is summarized in Algorithm 6. The process starts at the lattice origin, usually the center of the grid representation. Then, all states placed at the lattice origin with sampled heading values are analyzed [Line 3-11]. For each state $(x, y, \theta)$, a set of outgoing paths is determined. The cardinality of this set is given by the
Chapter 5. Nonholonomic motion planning

Algorithm 6 Algorithm to generate the state lattice

Require: ArcLength : Nominal length of the generated arcs
Require: NumberArcs : Number of outgoing arcs for a given state (branching factor)
Require: MinRadius : The curvature limitation for the calculated arcs as imposed by the robotic vehicle
Require: MaxRadius : Value to avoid the generation of arcs close to the straight line
Require: NumberHeadings : Number of heading samples for a given position

1: ControlSet = ∅
2: \((x^d_0, y^d_0)\) = Lattice origin
3: for all headings \(\theta_i\) do
4: \((x_f, y_f) = \text{CalculateStraightPath}(x^d_0, y^d_0, \theta_i, \text{ArcLength})\)
5: \(\text{CalculateMotionPrimitive}(x^d_0, y^d_0, \theta_i, x_f, y_f, \theta_f)\)
6: for \(IdxArc = 1 : (NumberArcs - 1)/2\) do
7: \(R_{IdxArc} = \text{GetRadius}(\text{MinRadius}, \text{MaxRadius})\)
8: \((x_f, y_f, \theta_f) = \text{CalculateArc}(x^d_0, y^d_0, \theta_i, \text{ArcLength}, R_{IdxArc})\)
9: \(\text{CalculateMotionPrimitive}(x^d_0, y^d_0, \theta_i, x_f, y_f, \theta_f)\)
10: \((x_f, y_f, \theta_f) = \text{CalculateArc}(x^d_0, y^d_0, \theta_i, \text{ArcLength}, -R_{IdxArc})\)
11: \(\text{CalculateMotionPrimitive}(x^d_0, y^d_0, \theta_i, x_f, y_f, \theta_f)\)
12: return ControlSet

13: function \(\text{CalculateMotionPrimitive}(x^d_0, y^d_0, \theta_0, x_f, y_f, \theta_f)\)
14: \((x_f^l, y_f^l) = \text{GetNearestLatticeNode}(x_f, y_f)\)
15: \(\theta_f^l = \text{GetNearestHeadingSample}(\theta_f)\)
16: \((l, R) = \text{CalculateMotionPrimitiveDecomposition}(x^d_0, y^d_0, \theta_0, x_f^l, y_f^l, \theta_f^l)\)
17: if IsValid \(\text{MotionPrimitive}(x^d_0, y^d_0, \theta_0, l, R)\) then
18: \ Add MotionPrimitive\( (x^d_0, y^d_0, \theta_0, l, R)\) to ControlSet
19: end function
5.3. Path planning using motion primitives for the ExoMars rover

Figure 5.6: Illustration of continuous arc of circle paths and the corresponding state lattice connecting discretized states at initial heading $\theta^d_0 = 45^\circ$

*NumberArcs* parameter which is an odd number. This is due to the fact that for a given initial configuration, a straight outgoing path and an equal number of arc paths on either side of it are calculated. First, the final position of the outgoing straight path is calculated using eq. 5.3, with $l = ArcLength$[Line 4]. Following, the CalculateMotionPrimitive routine is run in order to determine a motion control for the current initial and final configurations[Line 5]. It mainly consists in finding the closest discrete state to the provided final configuration and calculate a motion primitive between the origin and the discrete state[Lines 14-16]. Finally the feasibility of the generated motion control is evaluated with respect to the locomotion constraints of the robotic vehicle. If no condition is violated, the motion primitive is added to the control set [Line 18], otherwise it is discarded. Subsequently, groups of two arcs are generated for a given radius on either side of the straight line. The procedure is similar: once the final position and heading of each arc is determined, the CalculateMotionPrimitive routine is used to calculate the closest arc with discrete initial and final states and eventually update the control set.

An example of a control set generated using Algorithm 6 is provided in Figure 5.7 having the following parameter setting: *NumberArcs* = 7, *ArcLength* = 800mm, *MinRadius* = 1m, *MaxRadius* = 5m, *NumberHeadings* = 72, *GridResolution* = 50mm.

The memory use is a specific constraint for robotic planetary exploration missions. This issue can be addressed by storing only a subset of the generated control set. First, due to the translation regularity property of the state lattice, the control set calculation can be limited to
queries originating only at a certain grid cell (as shown in Figure 5.7). Moreover, the control set exhibits symmetry with respect to the x-axis, y-axis and the axis at 45°, as shown in Figure 5.8. By exploiting these properties, a huge memory capacity can be saved. Only $1/8 + 1$ of the initial discrete headings are precalculated and stored. The rest of the controls are restored at request by applying the corresponding transformation to the already computed limited control set. Furthermore, the storage memory requirements are limited by encoding the swaths of the motion controls by using Freeman chain encoding. In result, the memory requirement for the control set in Figure 5.7 is reduced from 31.5KB to 0.4KB.
5.4 Path planning using control sets

This section addresses the problem of performing constrained path planning using control sets generated in the previous section. The objective is to continue performing optimal path planning while addressing one important disadvantage, so called “the curse of dimensionality”. The state of the rover includes the heading as well as the position in the navigation map. Therefore, when performing path planning using control sets, state \( s \in S \) is a 3-dimensional state \((x, y, \theta)\). This results in exponential growth in complexity as the dimension of the search space increases. This is why this thesis suggests using an adapted version of the \( A^* \) path planner.

The capabilities of the \( A^* \) algorithm presented in Algorithm 2 were adapted to handle the supplementary dimension of the search space regarding the heading value. It has to be noted that during a state expansion the set of successors [Line 28] comprises the final states of all control primitives leaving the expanded state. The cardinality of the set of successors is given by the branching factor parameter of the control set generator. For example, when using the control set presented in Figure 5.7, each expanded node would have maximum 7 successors. The cost decrease [Lines 30,33] for a given state is handled as well. When more than one motion primitives reach the same grid cell with the same heading, only the lower cost path is retained. Such an example is provided in Figure 5.9. Here, three different arcs generate the same final state \((x_f, y_f, \theta_f)\). Thus, the path planner can reach the cell in the grid \((x_f, y_f)\) with a heading of 15° either through a straight line with an initial heading of 15° or through 2 arcs with opposite radius values. The resolution of the grid decomposition is 50mm.

Finally, the stopping condition of the path planner has changed so that a solution path is provided when one of the following conditions is accomplished:

- The latest developed state from the OPEN list lies in a given neighborhood of the goal state. Usually the neighborhood is of the shape of a circle, centered at the goal state and of radius given by the primitive length parameter of the motion control set. In this situation, the planner returns a path which reaches the goal state.

- The maximum number of nodes in the search tree is attained. The size of the search tree is bounded by a parameter of the nonholonomic path planner entitled Exploration Depth. The exploration depth of a given state is equal to the number of used motion controls to reach it. This way, the size of the search tree generated by the nonholonomic path planner is bounded and so a solution is guaranteed to be given in a finite amount of time. In such
a case, a solution to the path planning request is found but the goal state is not reached. In this respect, the path returned by the path planner leads the rover towards the goal by taking into account the locomotion system constraints.

![Figure 5.9: Motion controls generating the same state during path planning](image)

### 5.4.1 State lattice cost function

One of the most expensive operations during the path planning process is represented by the collision checks and the calculation of the cost function between two states. As discussed in Section 3.5, a cost function includes components regarding the length of the path, the difficulty of the traversed terrain and the distance to a precomputed path primitive. Path planning using a precalculated control set has the advantage of having stored the projection of the motion control over the grid decomposition. Also called trajectory swath, the projection of an arc over the grid contains the set of all navigation map cells that are affected during the path execution. An example of trajectory swaths is given in Figure 5.6, where all affected cells are marked in grey. Thus the calculation of terrain difficulty component of edge cost is reduced to the sum of the navigation values in the navigation map encountered by the precomputed arc swath. The cost function used throughout this section is equivalent to the one introduced in eq. 3.3 and is given in eq. 5.6. The importance of each component of the cost function is weighted through \( W_{\text{nav}}, W_{\text{dist}}, \) and \( W_{\text{prim}} \). The last component of the cost function refers to the distance of the arriving state to a previously calculated path primitive. As explained in Section 2.2.2, the use
of a primitive rough path to guide the search improves the path planner computational load by limiting the width of the search tree.

\[
c(s_i, s_j) = W_{\text{nav}} \sum_{k \in \text{swath}} nav_k + W_{\text{dist}} \text{length}_{arc}(s_i, s_j) + W_{\text{prim}} \text{dist}(s_j, P) \tag{5.6}
\]

### 5.4.2 State lattice heuristic estimates

The heuristic estimate has direct effects on the performance of the path planner. When planning using precomputed control sets, common heuristics include the Euclidean, Manhattan or Octile distance to the goal as presented in Table 3.1. Even though it is computationally efficient, the Euclidean metric is often considered to underestimate the real remained distance due to the unknown distribution of the navigation space. However, when being used with state lattices, it prioritizes straight lines over low radius arc paths. For this reason, Manhattan and Octile metrics are to be used. Recent work proposes the off-line calculation of a Heuristic Look-Up Table (HLUT) which stores distance to the goal estimations calculated by the planner through previous requests [Pivtoraiko et al., 2007]. However the capability of the HLUT table limits the cost estimation to the length of the path for a certain query. The focus in this thesis is the applicability of the planner in natural terrain. Since the navigability of the environment represents an important factor to choose the optimal path, a precalculated HLUT table is not applicable to the problem considered in this thesis.

### 5.4.3 Nonholonomic motion control with planned in-place turns

The main objective of the use of state lattice path planners is to avoid in-place-turn maneuvers which are considered to be very expensive in time and energy. This is due to the fact that the rover has to stop the locomotion, reorient the wheels in envelope configuration, perform in-place turn until the desired heading is reached and continue the execution of the planned path. However, there are situations where performing an in-place rotation can significantly reduce the path execution time, as shown in the example provided in Figure 5.10. The rover is at the center of the navigation map with initial heading towards the bottom (Figure 5.10a). The goal position is placed on the top left corner of the navigation map, marked with a rippling flag on a red base. Figure 5.10a displays the path calculated using the precalculated control set which models the nonholonomic constraints of the rover. A zoom in on the center of the navigation map is provided and the so-called Field of Plan of the rover is shaded in grey. The Field of Plan represents the area which can be explored by the rover given its heading and the minimum turning radius constraint, without taking into account the use of in-place turn maneuvers. The solid grey arcs of circle mark the minimum turning radius of the robotic vehicle. The arcs with the minimum turning radius are invalidated by obstacles. The path planner provides a nonholonomic trajectory which avoids the group of obstacles in the lower left of the navigation map and then heads up straight towards the target. The total traveled distance in this case is 23.60m. In such situation, the routine of adapting the heading of the rover towards the target before path planning is allowed. Figure 5.10b provides (in white) the computed 8m long path.
once the azimuth of the rover is corrected. It is clear that in such a situation, both rover security and trajectory execution time are improved after one in-place turn.

![Figure 5.10: Illustration of a configuration where an in-place rotation is preferred: (a) Path (in white) given by the path planner using the precalculated control set and a zoom in of the Field of Plan (FOP) of the robot, (b) Path (in white) given by the same path planner but with initial adjusted heading](image)

The policy to decide a rover heading correction is illustrated in Figure 5.11. The decision to command a heading correction depends on the navigation space configuration and on the position of the target with respect to the rover. An in-place-turn maneuver is commanded only when the target is behind the rover, lying in the 2D plane entitled "Rover back". In this case, the heading change of the rover $\theta_R$ is commanded to reach the value of the angle $\theta_P$ given by the path primitive used to guide the path search. It is important to use the path primitive orientation when guiding the rover heading change, because it takes into account the obstacle distribution in the navigation environment. In such a situation it is avoided that the rover travels for at least $\pi \times R_m$ ($R_m$ is the minimum turning radius) to reach an appropriate heading towards the target. If the target lies in the plane named "Rover front" the nonholonomic path planning is expected to provide a feasible path and no in-place turn is commanded.

![Figure 5.11: A planned in-place-turn is commanded only when the target lies behind the rover](image)
In Section 2.2.2 control corridor areas in the navigation map are introduced, which account for the locomotion errors during path execution, as shown in Figure 2.8. To make sure that the rover is kept away from any dangerous area, any forbidden region of the navigation map is dilated with respect to the width of the trajectory execution control corridor. Therefore, it might happen that the departing position for a path planning request falls on a control corridor navigation cell, marked with a black star in Figure 5.12. This results in the triggering of an emergency strategy of reaching a navigable area the fastest possible before planning a path towards the target. Thus, in such situation a path search which provides the navigable cell outside the control corridor closest to the robot is performed. This latter position will be the departing position for the nonholonomic path planning, marked with a red star in Figure 5.12. Note that this situation might generate in-place turns during the rovers’ maneuvers to leave the dangerous area. Therefore, once the rover reaches the starting position for the nonholonomic path planning, it might have an altered heading with respect to the chosen target and thus might limit the success rate of the state lattice path planner to find a path. Hence, an adjustment of the heading of the rover once it reaches again a navigable area has to be performed. The heading adjustment is performed taking into account the rover heading when reaching the navigable area (noted $\theta_C$, represented in Figure 5.12 with a green arrow) and the general heading towards the target to be reached (noted $\theta_G$, represented with an yellow arrow in Figure 5.12). The adjusted heading value of the rover after leaving the control corridor area, noted $\theta_R$ is computed using equation 5.7. This will be further used as the initial heading for the state lattice path planner.

\[
\theta_R = \begin{cases} 
\theta_G & \text{, if } |\theta_C - \theta_G| < \pi/2 \\
(\theta_C + \theta_G)/2 & \text{, otherwise}
\end{cases}, \text{with } \theta_C, \theta_G \in (-\pi, \pi]
\]  

(5.7)

5.5 Experimental evaluation

A large set of experiments was performed to assess the performance of this state lattice path planner compared with the optimized grid path planner developed by CNES. During these experiments, a random set of configurations regarding the navigation map, start and goal positions
Chapter 5. Nonholonomic motion planning

are generated.

5.5.1 Choose the best heuristic function

An initial focus was put on the use of the heuristic function to guide the development of the search tree towards the selected goal position. Better heuristic estimates are more informed and provide better computation performances to the path planning algorithms. Thus, first, the influence of the heuristic function on the time performance of the nonholonomic path planner has been analyzed. The parameters used for the generation of the motion control set are specified in Table 5.1.

<table>
<thead>
<tr>
<th>Translational discretization</th>
<th>50mm square cell</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heading discretization</td>
<td>360 uniform values (1°)</td>
</tr>
<tr>
<td>Primitive length</td>
<td>400mm</td>
</tr>
<tr>
<td>Branching factor</td>
<td>5 outgoing arcs per state</td>
</tr>
<tr>
<td>Minimum turning radius</td>
<td>1m</td>
</tr>
<tr>
<td>Maximum turning radius</td>
<td>5m</td>
</tr>
<tr>
<td>Radius discretization</td>
<td>4 values, uniform</td>
</tr>
</tbody>
</table>

Table 5.1: Parameters used to calculate the state lattice control set

A set of 1900 local navigation maps with random start and goal states is generated for this test. It is recalled that a navigation map has a limited coverage of 17.5m × 17.5m at a resolution of 50mm per pixel. As well, the initial rover position is always at the center of the navigation map and the goal position anywhere in the navigable area encouraging to generate the maximum possible variety of queries. Depending on the configuration, the nonholonomic motion planner provided paths of length ranging from 0.5m to 9.5m. Through this experiment, the performance of four heuristic functions is assessed: Euclidean distance, Manhattan distance, Octile distance and the so called Manhattan-Octile distance. In addition to the well known Manhattan and Octile distance measures, a hybrid version of these two, entitled here Manhattan-Octile distance was also used. It is a measure which is calculated based on alternating the Manhattan and Octile distances from the goal state to any state in the navigation environment. It aims to reduce the effects of each of the two measures and not to favor straight or diagonal paths. Moreover, while the Euclidean distance is calculated online during the path planning process, the other three measures are precomputed and stored in a distance map. At the construction, the distance map takes into account the obstacle distribution trying to provide a highly accurate heuristic estimate.

Figure 5.13 provides the color-coded precomputed distance maps with different measures for a given navigation map shown in Figure 5.13a with the starting position at its center and the goal position at the top left of the image. The three corresponding distance maps are shown.

The total computation time per query is the used metric to assess the performance of the nonholonomic path planner given the distance map. This metric has been chosen based on the fact that the computational load is proportional to the developed search tree, whose dimensions is directly influenced by the heuristic estimate.
5.5. Experimental evaluation

![Navigate map with path](image1)
![Manhattan distance map](image2)
![Octile distance map](image3)
![Manhattan-Octile distance map](image4)

Figure 5.13: Representation of the precomputed obstacle aware distance maps

![Computation time comparison](image5)

Figure 5.14: Computation time comparison regarding the used heuristic function
Chapter 5. Nonholonomic motion planning

Figure 5.14 compares the time performance of the path planning queries. It is shown that, even if it is computed online, the Euclidean heuristic outperforms the rest of the distance measures. The time performance is plotted versus the path length which remains the same for all the heuristic estimates. Given that the Euclidean distance measure provides the fastest path planning solutions without affecting the path quality, it will be used as heuristic estimate throughout the rest of the experiments in this section.

5.5.2 Nonholonomic vs grid path planner

An extensive simulation experiment was undertaken in order to assess the performance of single query nonholonomic path planning with various parameters for the generation of the motion control set. It consists of 2800 randomly generated configurations over the HiRISE DEM. Table 5.2 provides a summary of the results obtained throughout this analysis.

For each configuration, the path planning query solution provided by the nonholonomic path planner is compared with the one provided by the A* grid planner. The nonholonomic path planner uses motion control sets with various branching factors (R) and heading discretization values (H). For example, in Table 5.2, R5H1 denotes the nonholonomic path planner with branching factor of 5 and heading discretization of 1. One important evaluation measure is the success rate of the path planning query. As mentioned in Section 5.4, the Exploration Depth parameter is used to limit the size of the search tree. Throughout these tests it is set to 20 motion controls, meaning that the goal state is expected to be reached through paths of maximum 8m length. While the A* grid planner has a success rate of 100%, nonholonomic path planners are not able to provide a solution for configurations in which the locomotion constraints of the rover do not allow the path generation. Moreover, it is observed that the success rate is directly influenced by the branching factor of the motion control set. The smaller the branching factor is, a lower success rate is obtained. This is due to the fact that a higher branching factor increases the reachability of the search tree and improves the chance of finding a feasible path solution for a given configuration. Consequently, the highest success rate (97%) of the nonholonomic path planner obtained during this experimental evaluation is achieved with a branching factor of 9.

The actual effectiveness of the nonholonomic path planner is evaluated using the goal reach rate measure. It counts the rate at which the provided path solutions actually reach the neighborhood region around the goal state. Thus, an average of 96% goal reach rate over the solutions provided by the 2800 path planning queries is achieved. Similar to the success rate, an increase in the goal reach rate with respect to the branching factor of the motion control set is observed.
<table>
<thead>
<tr>
<th></th>
<th>A*</th>
<th>R3H1</th>
<th>R5H1</th>
<th>R5H3</th>
<th>R7H1</th>
<th>R7H3</th>
<th>R9H1</th>
<th>R9H3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success rate</td>
<td>100.00%</td>
<td>94.50%</td>
<td>95.00%</td>
<td>96.61%</td>
<td>97.07%</td>
<td>97.11%</td>
<td>97.22%</td>
<td>97.22%</td>
</tr>
<tr>
<td>Goal reach rate</td>
<td>100.00%</td>
<td>93.08%</td>
<td>93.90%</td>
<td>96.01%</td>
<td>96.62%</td>
<td>96.62%</td>
<td>96.81%</td>
<td>96.88%</td>
</tr>
<tr>
<td>Runtime (ms)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>min</td>
<td>0.11</td>
<td>0.07</td>
<td>0.06</td>
<td>0.08</td>
<td>0.06</td>
<td>0.06</td>
<td>0.07</td>
<td>0.08</td>
</tr>
<tr>
<td>mean</td>
<td>2.01</td>
<td>0.16</td>
<td>0.99</td>
<td>0.97</td>
<td>6.56</td>
<td>6.54</td>
<td>33.18</td>
<td>31.89</td>
</tr>
<tr>
<td>max</td>
<td>12.82</td>
<td>0.83</td>
<td>26.84</td>
<td>10.87</td>
<td>88.33</td>
<td>83.33</td>
<td>445.00</td>
<td>445.00</td>
</tr>
<tr>
<td>median</td>
<td>2.07</td>
<td>0.16</td>
<td>0.67</td>
<td>0.68</td>
<td>4.13</td>
<td>4.07</td>
<td>20.83</td>
<td>19.62</td>
</tr>
<tr>
<td>Runtime wrt A*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>min</td>
<td>100.00%</td>
<td>1.25%</td>
<td>1.48%</td>
<td>1.64%</td>
<td>2.96%</td>
<td>3.12%</td>
<td>3.42%</td>
<td>3.42%</td>
</tr>
<tr>
<td>mean</td>
<td>100.00%</td>
<td>10.38%</td>
<td>46.15%</td>
<td>45.32%</td>
<td>285.41%</td>
<td>284.43%</td>
<td>1421.31%</td>
<td>1365.35%</td>
</tr>
<tr>
<td>max</td>
<td>100.00%</td>
<td>118.18%</td>
<td>785.11%</td>
<td>785.11%</td>
<td>6150.00%</td>
<td>6150.00%</td>
<td>32840.99%</td>
<td>32840.99%</td>
</tr>
<tr>
<td>median</td>
<td>100.00%</td>
<td>8.02%</td>
<td>35.71%</td>
<td>36.18%</td>
<td>205.43%</td>
<td>203.73%</td>
<td>1011.76%</td>
<td>970.59%</td>
</tr>
<tr>
<td>Time improvement rate</td>
<td>N/A</td>
<td>99.92%</td>
<td>93.32%</td>
<td>93.71%</td>
<td>32.19%</td>
<td>32.61%</td>
<td>13.89%</td>
<td>14.20%</td>
</tr>
<tr>
<td>Average cell expansions</td>
<td>226.00</td>
<td>1.00</td>
<td>9.76</td>
<td>9.50</td>
<td>46.25</td>
<td>45.93</td>
<td>151.74</td>
<td>146.86</td>
</tr>
<tr>
<td>Average path difficulty</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>min</td>
<td>100.00%</td>
<td>63.72%</td>
<td>56.86%</td>
<td>63.76%</td>
<td>65.82%</td>
<td>67.07%</td>
<td>63.58%</td>
<td>63.98%</td>
</tr>
<tr>
<td>mean</td>
<td>100.00%</td>
<td>98.74%</td>
<td>98.82%</td>
<td>98.89%</td>
<td>98.80%</td>
<td>98.75%</td>
<td>98.91%</td>
<td>98.91%</td>
</tr>
<tr>
<td>max</td>
<td>100.00%</td>
<td>155.70%</td>
<td>125.29%</td>
<td>125.29%</td>
<td>122.90%</td>
<td>122.62%</td>
<td>122.90%</td>
<td>123.64%</td>
</tr>
<tr>
<td>median</td>
<td>100.00%</td>
<td>99.35%</td>
<td>99.43%</td>
<td>99.39%</td>
<td>99.30%</td>
<td>99.27%</td>
<td>99.42%</td>
<td>99.44%</td>
</tr>
<tr>
<td>Difficulty improvement rate</td>
<td>N/A</td>
<td>63.37%</td>
<td>62.76%</td>
<td>63.30%</td>
<td>65.38%</td>
<td>66.62%</td>
<td>62.02%</td>
<td>62.18%</td>
</tr>
<tr>
<td>Distance measure</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>min</td>
<td>N/A</td>
<td>91.57%</td>
<td>88.86%</td>
<td>88.96%</td>
<td>88.57%</td>
<td>92.71%</td>
<td>91.35%</td>
<td>89.81%</td>
</tr>
<tr>
<td>mean</td>
<td>N/A</td>
<td>100.2%</td>
<td>106.23%</td>
<td>103.62%</td>
<td>102.72%</td>
<td>102.72%</td>
<td>102%</td>
<td>102.04%</td>
</tr>
<tr>
<td>max</td>
<td>N/A</td>
<td>368.66%</td>
<td>316.15%</td>
<td>316.15%</td>
<td>282.87%</td>
<td>282.87%</td>
<td>269.05%</td>
<td>267.64%</td>
</tr>
<tr>
<td>median</td>
<td>N/A</td>
<td>103.5%</td>
<td>100.78%</td>
<td>100.54%</td>
<td>100.48%</td>
<td>100.35%</td>
<td>100.08%</td>
<td>100.00%</td>
</tr>
</tbody>
</table>

Table 5.2: Nonholonomic path planning single query experimental set summary
The runtime of the nonholonomic path planner is compared with the grid based optimized A* path planner of CNES. First, the statistical measures (minimum, maximum, mean and median values) of the raw computation time obtained during the experimental test are shown for each path planner. Then, the statistical measures normalized by those of the A* path planner results are also shown in order to focus on the improvement. As a synthesis, the time improvement rate provides the amount of cases where the nonholonomic path planner outperforms the A* grid path planner. Thus, path planning using precomputed motion control sets with a branching factor of up to 5 outperforms in average by up to an order of magnitude the grid-based path planner. Especially, nonholonomic path planning with a branching factor of 3 dominates in over 99% of the queries. As the branching factor is increased to 5, the computation time in the search tree expansion step is affected but still the nonholonomic path planner is faster in 93% of the test cases. Finally, even if they improve the goal reach rate, higher branching factors of the motion control set result in much slower computation time than the A* grid-based planner. Hence, a branching factor of 7 brings a time improvement rate of 32%, and the use of 9 motion primitives performs better in only 13% of the cases.

Another measure which reflects the computational load of the path planner is given by the average expansion steps. This measure is normalized by the result of the $R^3H_1$ path planner. The $R^3H_1$ path planner is taken as reference because it has the lowest cardinality for the neighborhood of a given state (3 outgoing motion primitives) and consequently it has the lowest count of expanded states per search. As expected, increasing the branching factor results in an exponential increase of the search tree and thus of the count of expanded states. Up to 150 times more states are expanded for a branching factor of 9. However, the highest number of cell expansions is acquired by the A* grid path planner. This is a natural evolution, because the grid path planner evaluates each cell in the grid decomposition while the search tree is built, while the motion primitives of the nonholonomic path planner cover more than one cell in a single expansion. Thus, longer the motion primitives, lower cell expansions to reach the desired target.

The particularity of the proposed algorithms is that the objective function of the path planner aims to minimize the effect of the roughness of the terrain where the robot navigates. Therefore, the quality of the resulting path is represented by the average path difficulty. This measure is also normalized by the result of the A* grid path planner. It can be noticed that in average the nonholonomic path planner provides paths easier to navigate. The amount of cases where the nonholonomic path planner provided paths with lower difficulty than the A* grid planner is also given. This result shows that at least in 62% of the cases the nonholonomic path planner outperforms the grid planner. This rate increases up to 66% in the case of $R^7H_3$ mainly due to the reachability diversity provided by the increased branching factor.

Finally, the length of the calculated paths is analyzed. It is also a crucial factor to assess the overall performance of the path planner as it has direct effects on the deployment of an autonomous robotic exploration mission. In general, longer path would demand longer execution time having as direct consequence a limitation of the total traveled distance during an exploration mission. A distance measure is also defined by the ratio between the length of the path provided by the nonholonomic path planner and the one given by the A* grid planner. However, it should be noted that the path provided by the nonholonomic path planner aims to reach a close
neighborhood of the goal state. Therefore, the distance ratio is calculated using eq. 5.8, where
\( A \) represents the length of the path provided by the A* grid planner, \( B \) represents the length
of the path provided by the nonholonomic path planner reaching state \( P \), and \( G \) represents the
goal state with \( P \in \text{Neighborhood}(G) \). If the path provided by the nonholonomic path planner
is shorter than the one given by the A* grid planner the resulted distance measure is a number
lower then one.

\[
Distance_{\text{measure}} = \frac{B + \text{Eucl}_\text{dist}(P,G)}{A}
\]

(5.8)

The analysis of the statistical values for the distance measure indicates that paths calculated
using precomputed motion control sets are longer than the ones provided by the grid-based A*
planner. It is also shown that increasing the branching factor results in shorter paths. It is the
expected outcome because the A* grid path planner provides paths with sharp turns over the
grid decomposition. Contrary, the use of motion controls by the nonholonomic path planner
excludes sharp turns by generating longer paths.

Finally, Table 5.2 presents the performance measures for the variations of the nonholonomic
path planner which use a heading discretization of 3 degrees for the calculation of the motion
control set. Despite the fact that they have slightly better performance measures, it is considered
that the magnitude of the improvement is not significant.

5.5.3 Nonholonomic path planning with planned in-place-turns

This subsection evaluates the performance of the nonholonomic path planner when one in-place-
turn is authorized at the beginning of the path planning query, as explained in Section 5.4.3.
Therefore two nonholonomic path planners with branching factor of 5 and heading discretization
of 1 with and without planned in-place turns are compared in this test. The evolution of the two
nonholonomic path planning versions have in average similar performance measures. However,
when using one planned in-place turn, the performance of the \( R5H1 \) nonholonomic path planner
is increased both from the point of view of the computation time (due to lower average state
expansions) and distance measure. Throughout the experimental test, the routine of corrective
planned in-place turn was performed in 161 cases out of 2801, thus 5.75% of the total queries.
Figure 5.15 provides a summary of the performance gain obtained in these 161 cases, where
\( R5H1^1 \) denotes the nonholonomic path planner without in-place turns and \( R5H1^2 \) denotes the
nonholonomic path planner with corrective in-place turn. The use of planned in-place turns
increased the success rate by approximately 25% and the goal reach rate by approximately 34%.
Moreover, the average distance measure rate is enhanced. The nonholonomic path planner with
planned in-place turn provides paths in average 4% longer than the ones provided by the A*
grid path planner, while the nonholonomic path planner without in-place turns provides paths
in average 55% longer. A such increase in the path length can by crucial for the scientific return
of the robotic exploration mission.

Figure 5.16 illustrates a configuration for which the use of planned corrective heading in-place
turn results in an improvement of the distance measure of the provided path of approximately
50%. The initial rover heading is towards the bottom of the navigation map. In Figure 5.16a
the path provided by the A* grid planner is illustrated. It is obvious that the rover would not be
Figure 5.15: Performance gain when using planned rover heading correction in-place turns

able to execute such a trajectory without performing an in-place turn. Then, in Figure 5.16b the path provided by the nonholonomic path planner without authorized in-place rotations is shown. This trajectory makes a U-turn following the curve of minimum turning radius and continues with several arcs until the desired goal neighborhood is reached. Using this path planner, the rover has to execute a path of approximately 9.3m which is more than 50% longer than the previously generated path. Lastly, in Figure 5.16c the solution provided by the nonholonomic path planner which allows a planned corrective in-place turn is displayed. This path is much closer to the A* grid-based path from the point of view of the performance measures. Thus, in such situation an in-place turn is preferred to a longer trajectory to execute.

The behavior of the nonholonomic path planner was studied as well for the emergency cases where the rover position lies over a control corridor area in the navigation map. This situation occurred for 64 queries out of the 2800 configurations of the experimental test. However, using the heading adjustment approach presented in Section 5.4.3, more than 90% of queries provided a reliable path. Also, an increased success rate was obtained for higher branching factors of the precomputed motion control set. This suggests that the reachability of the motion control set with respect to the obstacle distribution in the navigation environment is also an important factor to be taken into account.

5.5.4 Nonholonomic vs grid path planner evaluation on field navigation data

This section presents the performance comparison of the nonholonomic path planner against the grid path planner using real navigation data acquired onboard the IARES rover (Figure 2.15) during autonomous navigation field experiments performed on the SEROM Mars-like test site. The dataset consists of 141 path planning configurations comprising the local navigation map of the close environment with the rover at its center and a random position of the goal in the navigable area around the rover. Throughout this field test the navigation map has a resolution
5.5. Experimental evaluation

(a) A* grid-based path planner
Path length 6.18m

(b) R5H1
Path length 9.29m
Distance measure 150.32%

(c) R5H1
Path length 6.30m
Distance measure 101.94%

Figure 5.16: Example of distance measure improvement through planned heading corrective in-place turn. Resulting paths for: (a) A* grid planner, (b) R5H1, (c) R5H1

Table 5.3 provides a synthesis of the obtained results, where three settings were considered for the nonholonomic path planner regarding the branching factor. Similar to the previous experiment, the nonholonomic path planner has 5% failure rate with respect to the grid path planner.

First, regarding the statistical measures of the computation time, the nonholonomic path planner outperforms the A* grid path planner in all cases when a branching factor of 5 is used, or in more than 90% of cases for a branching factor of 7. Second, in more than 60% of cases in this experiment, the nonholonomic path planner provides paths easier to navigate. Finally, the A* grid path planner provided slightly shorter paths.

The particularity of this experiment is that the nonholonomic path planner with a branching factor of 7 outperforms the A* grid path planner with respect to the computational load and the average path difficulty. This is due to the increased resolution of the navigation map. Thus, a fixed length arc-like path covers more cells in such a navigation map. As discussed in Section 5.5.2, this situation results in a lower number of expanded states, the target being reached faster.

5.5.5 Mission scenario evaluation

Another experiment was conducted to assess the capabilities of the nonholonomic path planner to perform autonomous path planning in realistic settings. A set of 300 mission scenarios over the ASU DEM was generated, with the distance from start to target between 25m and 103m. As in such scenario the target cannot be reached after a single perception, the capability of the nonholonomic path planner to provide solutions for successive path planning queries is evaluated.

Table 5.4 provides a summary of the results obtained throughout this experimental test. Here, only two versions of the nonolonomic path planner were tested: one using a branching factor of 5 and another with a branching factor of 7. First the success rate of each approach
is evaluated. It refers to the percentage of cases where the robot reaches the mission target through successive cycles of perception, path planning and execution. It is observed that there is a minor rate of failure, and the highest is attained by the $A^*$ grid path planner. This is due to the limited coverage of the navigation map, which prevents the rover to reach the target when having to drive past large obstacles which can not be covered by a local navigation map. This issue is solved by using global path planning which will be discussed in Chapters 6 and 7.

Similar to the previous experiment, it can be concluded that the nonholonomic path planner generates slightly longer trajectories than the $A^*$ grid planner. The calculated paths have a total distance improvement over the grid planner only in 32.44% of the cases for $R5H1$ and 35.79% of the total scenarios in for $R7H1$. Subsequently, the results are also consistent for the average roughness of the computed paths. Thus, $R5H1$ outperforms the grid planner in approximately 86% of the queries, with an increase to 90% for $R7H1$.

Finally, the total simulation time for each mission scenario is evaluated. This comprises the simulation time needed to reach a given mission scenario, including taking perceptions of the navigation environment, DEM and navigation map building, path planning and execution during the total number of cycles. The number of iterations depends on the distance to be traveled to reach the mission target. Throughout this test, the length of mission trajectories vary from 25.5m to 78m. An average of 6470 iterations per path planner (single path planning query) were performed. The nonholonomic path planner has lower computation time than the
5.5. Experimental evaluation

<table>
<thead>
<tr>
<th></th>
<th>A*</th>
<th>R5H1</th>
<th>R7H1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success rate</td>
<td>98.68%</td>
<td>99.34%</td>
<td>99.34%</td>
</tr>
<tr>
<td>Traveled distance</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wrt to A* min</td>
<td>100.00%</td>
<td>92.50%</td>
<td>92.44%</td>
</tr>
<tr>
<td>mean</td>
<td>100.00%</td>
<td>100.41%</td>
<td>100.08%</td>
</tr>
<tr>
<td>max</td>
<td>100.00%</td>
<td>138.71%</td>
<td>105.84%</td>
</tr>
<tr>
<td>median</td>
<td>100.00%</td>
<td>100.28%</td>
<td>100.18%</td>
</tr>
<tr>
<td>Traveled distance</td>
<td>N/A</td>
<td>32.44%</td>
<td>35.79%</td>
</tr>
<tr>
<td>improvement rate</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average path</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>difficulty wrt to A*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>min</td>
<td>100.00%</td>
<td>93.20%</td>
<td>92.23%</td>
</tr>
<tr>
<td>mean</td>
<td>100.00%</td>
<td>98.76%</td>
<td>98.55%</td>
</tr>
<tr>
<td>max</td>
<td>100.00%</td>
<td>103.55%</td>
<td>103.79%</td>
</tr>
<tr>
<td>median</td>
<td>100.00%</td>
<td>98.82%</td>
<td>98.68%</td>
</tr>
<tr>
<td>Average path</td>
<td>N/A</td>
<td>85.95%</td>
<td>90.30%</td>
</tr>
<tr>
<td>difficulty improvement rate</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time improvement</td>
<td>N/A</td>
<td>62.41%</td>
<td>62.04%</td>
</tr>
<tr>
<td>rate wrt to A*</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.4: Nonholonomic path planning for mission scenario

A* grid path planner in approximately 62% of the mission scenarios used in this test.

![Percentage of in-place turns during a mission using the A* grid planner](image)

![Percentage of in-place turns during a mission using R5H1](image)

![Percentage of in-place turns during a mission using R7H1](image)

Figure 5.17: Percentage of in-place turns executed during each mission scenario using: (a) A* grid planner, (b) Nonholonomic path planner with branching factor of 5, (c) Nonholonomic path planner with branching factor of 7

The main objective of the nonholonomic path planner is to include the locomotion constraints...
in the path planning stage, in order to avoid in-place-turn maneuvers. Thus, the number of in-place turns carried out during the mission execution is compared for the three path planners. Figure 5.17 illustrates the distribution of the counts of in-place turns throughout the experimental set. First, it has to be noted that the A$^*$ grid path planner caused the highest total number of in-place-turn maneuvers. This is an expected behavior, as it does not take into account the locomotion constraints of the rover. The nonholonomic motion planner reduces the number of in-place turns of approximately 38% for $R5H1$ and by 42% for $R7H1$. Moreover, the A$^*$ grid path planner performs at least 1 in-place rotation for at least 75% of the mission test cases, and executes in-place-turn free paths in only 10% of the scenarios. It has to be remarked that the grid planner has at least 15% of mission queries where more than 1 in-place turns are performed during path execution. This amount is drastically reduced when using the nonholonomic path planner. Thus only in 2% of the mission scenarios more than one in-place turn is performed. In addition, 38% of the mission targets are reached with no in-place-turn maneuvers when using the nonholonomic path planner. The distribution of the in-place-turn maneuvers stays identical for the two nonholonomic path planners, even though $R7H1$ has a lower count.

Figure 5.18 provides an example of executed trajectories in a simulated mission scenario. The rover is supposed to reach the right side of a rock by avoiding other obstacles in its navigation environment. The executed trajectories have a length of 16.56m for the grid path planner and 16.07m for the nonholonomic path planner. While the nonholonomic path planner did not generate any in-place turns during path execution, the grid path planner planned to perform 3 in-place-turn maneuvers. The positions of the in-place-turn maneuvers are marked with a green turning arrows sign. Concerning the memory load during the mission simulation, path planning stages used 11.85MB in the case of the grid path planner against 1.66MB for the nonholonomic path planner.

Figure 5.18: Example of in-place turns generated during simulated mission using: (a) A$^*$ grid planner (b) nonholonomic path planner with branching factor of 5

Finally, another crucial performance measure for planetary exploration path planning algorithms concerns the memory load. Figure 5.19 provides a comparison of the memory use by each path planning approach throughout each mission scenario. The amount of used memory is plotted versus the total length of the executed trajectory which is proportional to the total amount of expanded states. The graph proves that the nonholonomic path planning using a
branching factor of 5 has a much less memory use than when using a branching factor of 7. This was also suggested throughout the single query experimental test, where \( R7H1 \) expanded in average 5 times more states than \( R5H1 \). Finally it is shown that the A* grid planner has a much high memory use than the proposed nonholonomic path planning approaches.

Figure 5.19: Memory use comparison per mission scenario with respect to the total traveled distance using: (a) A* grid planner, (b) Nonholonomic path planner with branching factor of 5, (c) Nonholonomic path planner with branching factor of 7

5.6 Conclusions

This chapter introduced constrained path planning and assessed its performance measures with respect to mission-constraints of a planetary exploration rover. With respect to the path planning architecture onboard the MERs and Curiosity, this approach provides optimal paths which take into account the steering capabilities of the rover. Moreover the computed paths have a higher complexity than simple arcs, allowing the rover to drive safely around obstacles encountered in its navigation environment.

First, a state lattice is designed, given desired parameters regarding the discretization of the search space, the branching factor or steering capabilities of the rover. Following the pre-computed state lattice is used during the online search path procedure by a nonholonomic path planner.

An initial experimental test showed that the Euclidean distance measure is the best choice for the heuristic function of the proposed nonholonomic path planner. This is because the used state lattice includes only forward motion controls (no in-place-turn controls or backward drives). Following, the nonholonomic path planner is assessed with respect to the grid path
planner currently in EDRES. During single query path planning experiments, both on simulated and on field test navigation data, it is shown that the nonholonomic path planner outperforms the grid path planner when using precomputed control sets with a branching factor of up to 7. It provides better computation time, memory use and average path difficulty. Specific cases are also considered, when one in-place turn is planned at the beginning of the path search procedure with the aim of reducing the path length and thus the trajectory execution time. Lastly, it is shown that the aim to reduce in-place turns during mission scenario execution is achieved. This is an important accomplishment with respect to trajectory execution time and locomotion system wear as it has a direct influence on the scientific return of a planetary exploration rover.
Chapter 6

Incremental path planning using constant resolution global navigation maps

The first part of this thesis performed only successive local path planning stages over a continuously updated local navigation map because of the onboard memory constraint. However, it was shown in Section 5 that due to the limited coverage of the local navigation map, the performance of the path planning algorithm is limited when the rover has to traverse dense obstacle fields or dead-end configurations. This chapter relaxes the memory-use limitation by assuming that there is enough memory to store a navigation map which covers the entire area the rover can explore during a sol and performs an evaluation of incremental global path planning algorithms. Previous chapters considered a static navigation environment, as the local path planning was performed over the updated local navigation map. Unlike these, this section addresses the problem of path planning in a dynamic navigation environment using a global navigation map which can incorporate information computed from low-resolution orbiter data and which is updated using high-resolution locally acquired perceptions. The first part of this section provides a state of the art of incremental path planning algorithms to be used in dynamic environments and an analysis of their performance. Then, the D* lite path planner is chosen to be implemented and evaluated in the CNES EDRES environment. The main objective of this study is to overcome the drawback of successive local path planning without violating the mission-specific constraints, other than the memory-use.

6.1 Incremental path planning algorithms for dynamic navigation environments

Recent developments of incremental search algorithms have focused on the replanning capabilities when changes in the robot environment are perceived. Therefore, several incremental search algorithms have been proposed using as baseline the A* algorithm [Stentz, 1994] [Stentz, 1995] [Koenig and Likhachev, 2002] [Ferguson and Stentz, 2005a]. These algorithms guarantee the op-
timality of the provided path solution, which explains their wide usage in robotic path planning applications [Stentz and Hebert, 1995] [Singh et al., 2000] [Kelly et al., 2006]. The advantage of such incremental algorithms is that they can handle changes in the navigation environment by only locally updating the search tree instead of rebuilding an entire search tree which is very time consuming.

6.1.1 D* algorithm

D* algorithm [Stentz, 1994] is one of the first incremental search heuristic path planners that allows optimal replanning in real-time. The advantage of this algorithm is that it can be applied to path planning in a partially-known environment. Each time the navigation information is updated, it performs replanning by repairing only the parts of the global path which are affected. It can be said that D* algorithm is a dynamic version of the Dijkstra’s algorithm or a version of the A* algorithm without heuristic function [LaValle, 2006]. Similar to the A* algorithm, during each expansion of the D* algorithm, three values are computed for every analyzed state s:

- The g-value $g(s)$ represents the actual cost of the path from state $s$ to the goal state $s_{goal}$. Since $D^*$ starts the search from the goal state towards the start state, the g-value is also called the goal distance for vertex $s$.
- The key value $k(s)$ which stores old g-values, before changes in the graph occurred.
- The parent pointer $parent(s)$ is the state through which the current state $s$ can reach the goal state at optimal cost.

Again, similar to the A* algorithm, D* keeps track of the expanded states in an OPEN list. It is mainly used to propagate cost changes in the search tree and to calculate the updated cost values of the affected states. Each state in the OPEN list can be classified into two groups: a LOWER state if $k(s) = g(s)$ and a RAISE state if $k(s) < g(s)$. When updates in the navigation environment are available, RAISE states are used to propagate path cost increases and LOWER states to propagate path cost decreases. This is performed by successively removing states from the OPEN list and propagating the cost changes to all affected neighbor states. These neighbor states are in turn placed in the OPEN list and the procedure continues until the optimal cost values are updated for all affected states. All nodes in the OPEN list are sorted in increasing order of their key values. The minimum key $k_{min}$ of all states in the OPEN list represents an important threshold, as all states having a key value lower or equal to $k_{min}$ are considered to have an optimal cost. Conversely, paths generated from other states are considered to be non-optimal.

Algorithm 7 provides the D* algorithm which follows two different procedures: one initial planning stage and multiple replanning stages. The initial planning stage builds the initial search tree starting from the goal state towards the current state of the rover and provides a path which is optimal with respect to the a-priori known navigation data. Algorithm 7 provides the routine followed by the rover. First, all states in the search space are initialized with infinite cost and key values and empty parent pointers [Line 4]. Then, the algorithm starts by initializing the OPEN list with the goal state $s_{goal}$, whose g-value is zero [Line 6] and the initial search tree
is built [Line 13]. If the path planning terminates because the OPEN list is empty it means that no path solution is available for the given configuration [Line 15]. Otherwise an optimal path can be extracted for the current configuration and the rover starts executing the given trajectory [Line 17] until it either reaches the target position [Line 7], or new perceptions of the environment are available. If changes in the navigation map are observed, the affected states are updated in the search tree and added to the OPEN list before the replanning procedure is executed [Lines 8-13].

Algorithm 7: D* Execute Trajectory

1: function ($D^*$_move)
2:   $s_{Robot} = s_{start}$
3:   for all $s \in S$ do
4:     $k(s) = \infty; g(s) = \infty; parent(s) = NULL$
5:   OPEN = $\emptyset$
6:   $D^*$.Insert($s_{goal}, 0$
7:   while $s_{Robot} \neq s_{goal}$ do
8:     $S_{changed} = Changed\_costs(S, s_{Robot})$
9:     if $S_{changed} \neq \emptyset$ then
10:        for all $s \in S_{changed}$ do
11:           if $parent(s) \neq NULL$ then
12:              $D^*$.Insert($s, g(s)$
13:          $D^*$.search()
14:          if $k_{min} \leq g(s_{Robot})$ then
15:             return "No path found"
16:          else
17:             $s_{Robot} = parent(s_{Robot})$
18:   end function

Algorithm 8 provides the D* search procedure. The main routine is performed in a while loop [Lines 11-33] which iteratively extracts from the OPEN list the state $s$ with the lowest key value. During the initial planning stage all expanded states are LOWER states and therefore only [Lines 19-23] are executed. Following, state $s$ is expanded and all neighbor LOWER states are inserted in the OPEN list with updated $g$-values, $k$-values and parent pointers. When state $s_{start}$ is extracted from the OPEN list and expanded, the search algorithm stops. At this point, an optimal path from $s_{start}$ to $s_{goal}$ can be extracted by tracking the parent pointers from $s_{start}$.

During the replanning stage, the algorithm iteratively removes states from the OPEN list and expands them [Lines 11-33] until the minimum key value in the OPEN list becomes larger than the key value of the current rover position $s_{Robot}$. During state expansion, a RAISE state will propagate the cost increase to the parent states [Lines 24-33]. Regarding LOWER states, it is checked first if the cost estimate can be further decreased [Lines 14-18]. Then, cost decrease is propagated to parent and neighbor states whose cost can be decreased [Lines 19-30]. If the current expanded state lowers the cost of a neighbor state which is not immediate descendant.
Algorithm 8 D* Algorithm

1: function $D^\ast$.Insert$(s,g_{\text{min}})$
2: if $s \in \text{OPEN}$ then
3:   $k(s) = \min(g(s), g_{\text{min}})$
4: else
5:   $k(s) = \min(k(s), g_{\text{min}})$
6: $g(s) = g_{\text{min}}$
7: OPEN = OPEN $\cup$ s
8: end function

9: function $D^\ast$.search()
10: $k_{\text{min}} = 0$
11: while OPEN $\neq \emptyset$ and $k_{\text{min}} \leq g(s_{\text{Robot}})$ do
12:   $k_{\text{min}} = k(\text{OPEN}.\text{Top}())$
13:   s = OPEN.Pop()
14:   if $k_{\text{min}} < g(s)$ then
15:      for all $s' \in \text{Succ}(s)$ do
16:         if $g(s') \leq k_{\text{min}}$ and $g(s) > g(s') + c(s', s)$ then
17:            $g(s) = g(s') + c(s', s)$
18:            parent($s$) = $s'$
19:      if $k_{\text{min}} = g(s)$ then
20:         for all $s' \in \text{Succ}(s)$ do
21:            if ($\text{parent}(s') = s$ and $g(s') \neq g(s) + c(s, s')$)
22:               or ($\text{parent}(s') \neq s$ and $g(s') > g(s) + c(s, s')$) then
23:               $D^\ast$.Insert($s', g(s') + c(s, s')$)
24:               parent($s'$) = $s$
25:      else
26:         for all $s' \in \text{Succ}(s)$ do
27:            if $\text{parent}(s') = \text{NULL}$
28:               or ($\text{parent}(s') = s$ or $g(s') \neq g(s) + c(s, s')$) then
29:               $D^\ast$.Insert($s', g(s) + c(s, s')$)
30:            else
31:               if $\text{parent}(s') \neq s$ and $g(s') > g(s) + c(s, s')$ and $s \notin \text{OPEN}$ then
32:                  $D^\ast$.Insert($s', g(s)$)
33:               else
34:                  if $\text{parent}(s') \neq s$ and $g(s') > g(s') + c(s', s)$
35:                     and $s' \notin \text{OPEN}$ and $g(s') > k_{\text{min}}$ then
36:                     $D^\ast$.Insert($s', g(s')$)
37:      end function

96
6.1. Incremental path planning algorithms for dynamic navigation environments

[Lines 29,30], it is inserted again in the OPEN list for further analysis. This avoids closed loops in the parent pointers. Finally, if a neighbor state can provide a lower accumulated score to the currently expanded state, the neighbor state is placed back in the OPEN list for later examination [Lines 32,33].

6.1.2 Focused D* algorithm

As the name suggests, the Focused D* [Stentz, 1995] is an extension of the D* algorithm which uses a heuristic function to focus the propagation of the cost changes of the LOWER and RAISE states towards the current position of the rover. It performs cost updates only for the states which are important for the current search. Focused D* keeps the same notation as the D* algorithm, while adding three new values for each analyzed state during the search process:

- The f-value \( f(s) \) represents the total estimated rover path going through state \( s \) given by \( g(s) + h(s, s_{Robot}) \), where \( h(s, s_{Robot}) \) represents the heuristic estimate for the traversal cost from the current rover position \( s_{Robot} \) to state \( s \).

- A biased f-value \( f_B(s) \) given by \( f(s) + d(s) \), where \( d(s) \) represents the accrued bias of the rover at the moments state \( s \) was inserted or updated in the OPEN list. Initially set to 0, the bias of the rover is the accumulated heuristic estimate between subsequent rover positions where replanning stages were performed.

- The rover state value \( r(s) \) which stores the state of the rover at the moment state \( s \) was inserted or updated in the OPEN list.

- The tag value \( t(s) \) represents the associated label of state \( s \) such that: \( t(s) = NEW \) if state \( s \) has never been visited, \( t(s) = OPEN \) if state \( s \) is currently in the OPEN list, \( t(s) = CLOSED \) if state \( s \) was expanded and removed from the OPEN list.

Focused D* orders states in the OPEN list by increasing \( f_B(s) \) values, with breaking ties by increasing \( f(s) \) and \( k(s) \). Whenever a given state \( s \) is removed from the OPEN list, it is expanded only if \( f(s) \) and \( f_B(s) \) were calculated based on the most recent rover position. Otherwise, the values are updated and the state is placed back in the OPEN list.

Algorithm 9 provides the pseudocode of the initialization of the Focused D* path planning algorithm, the path execution and replanning stages. First, it initializes the tag values for all states in the search space to \( NEW \) and inserts the goal state in the OPEN list with an initial cost of 0 [Lines 9-15]. Following, the parameter \( val \) is initialized [Line 16]. It is a vector which consists at any moment the f-value and k-value of the state in the OPEN list with minimum cost. Similar to the execution of the D* algorithm, it builds the initial search tree which takes into account the current knowledge of the navigation environment [Lines 17-18]. At each new perception, states with changed cost are updated [Lines 27-30] and the new rover state value and accrued bias are stored. Further, states from OPEN list are expanded to propagate cost changes until a new optimal path is found. Similar to the A* algorithm, the search procedure fails if the OPEN list becomes empty [Lines 33-34].
Algorithm 9 Focused D* Execute Trajectory

1: function MIN VAL()
2:   s = MIN STATE()
3:   if s = NULL then
4:     return NO - VAL
5:   else
6:     return val = ⟨f(s), k(s)⟩
7: end function

8: function FocusedD* move(s_start, s_goal)
9:   s_Robot = s_start
10:  s_Robot_curr = s_start
11:  d_curr = 0
12:  for all s ∈ S do
13:     t(s) = NEW
14:  OPEN = ∅
15:  FocusedD*.Insert(s_goal, 0)
16:  val = ⟨0, 0⟩
17:  while t(s_start) ≠ CLOSED and val ≠ NO - VAL do
18:     val = FocusedD*.search()
19:     if t(s_start) = NEW then
20:       return "No path found"
21:     while s_Robot ≠ s_goal do
22:       S_changed = Changed_costs(S, s_Robot)
23:       if S_changed ≠ ∅ then
24:         if then s_Robot_curr ≠ s_Robot
25:           d_curr = d_curr + h(s_Robot, s_Robot_curr) + ε
26:           s_Robot_curr = s_Robot
27:         for all s ∈ S_changed do
28:           if (s) = CLOSED then
29:             FocusedD*.Insert(s, g(s))
30:             val = MIN_VAL()
31:           while LESS(val, ⟨f(s_Robot), g(s_Robot)⟩) and val ≠ NO - VAL do
32:             val = FocusedD*.search()
33:           if val = NO - VAL then
34:             return "No path found"
35:           else
36:             s_Robot = parent(s_Robot)
37:           return "Goal reached"
38: end function
Algorithm 10 provides the supplementary functions used by Focused D\(^*\) during path planning. First the \textbf{Insert} function is detailed, which updates the \(k\)-value of state \(s\) [Lines 2-8]. Further, state \(s\) is repositioned in the OPEN list according to its new estimated values \((f_B(s), f(s), k(s))\) [Lines 9-11]. Functions \textbf{Delete} and \textbf{Put} remove, respectively insert the given state in the OPEN list and adjust the label of the state \(t(s)\) correspondingly. At each moment, function \textbf{GET\_STATE} provides the state in the OPEN list with the minimum vector value. The state with minimum vector value with respect to the current position of the rover is identified using the \textbf{MIN\_STATE} function. Therefore, it checks if the extracted state is updated with respect to the current position of the rover, otherwise its accrued bias is modified and the state is placed again in the OPEN list.

Algorithm 10 Focused D\(^*\) Algorithm - OPEN list management functions

1: function FocusedD\(^*\).Insert\((s, g_{\text{new}})\)  
2: \hspace{1em} \textbf{if} \(t(s) = \text{NEW}\) \textbf{then}  
3: \hspace{2em} \(k(s) = g_{\text{new}}\)  
4: \textbf{else}  
5: \hspace{2em} \textbf{if} \(t(s) = \text{OPEN}\) \textbf{then}  
6: \hspace{3em} \(k(s) = \min(k(s), g_{\text{new}}); \text{FocusedD}^*.\text{Delete}(s)\)  
7: \hspace{2em} \textbf{else}  
8: \hspace{3em} \(k(s) = \min(g(s), g_{\text{new}})\)  
9: \hspace{2em} \(g(s) = g_{\text{new}}; r(s) = s_{\text{Robot\_curr}}\)  
10: \hspace{2em} \(f(s) = k(s) + h(s, s_{\text{Robot\_curr}}); f_B(s) = f(s) + d_{\text{curr}}\)  
11: \hspace{2em} \text{FocusedD}^*.\text{Put}(s)\)  
12: \textbf{end function}\n
13: function MIN\_STATE\((\)\)  
14: \hspace{1em} \textbf{while} \(s = \text{GET\_STATE}() \neq \text{NULL}\) \textbf{do}  
15: \hspace{2em} \textbf{if} \(r(s) \neq s_{\text{Robot\_curr}}\) \textbf{then}  
16: \hspace{3em} \(g_{\text{new}} = g(s); g(s) = k(s)\)  
17: \hspace{3em} \text{FocusedD}^*.\text{Delete}(s); \text{FocusedD}^*.\text{Insert}(s, g_{\text{new}})\)  
18: \hspace{2em} \textbf{else}  
19: \hspace{3em} \textbf{return} \(s\)  
20: \hspace{2em} \textbf{return} \(\text{NULL}\)  
21: \textbf{end function}\n
The main function of the Focused D\(^*\) algorithm is given in Algorithm 11. Similar to D\(^*\), the state with the lowest \(f\)-value is extracted from the OPEN list [Lines 1-7]. If state \(s\) is a LOWER state, the cost decrease is propagated to all the possibly affected neighbor states [Lines 13-17]. If state \(s\) is a RAISE state, it is first checked if \(g(s)\) can be reduced using other neighbor state [Lines 8-12]. Following, the cost increase is propagated to parent states and neighbor states which might be affected [Lines 15-17]. In order to avoid loops in parent pointers, state \(s\) is re-inserted in the OPEN list when it provides lower costs to non-immediate descendant states [Lines 24-25]. Moreover, if the \(g\)-value of state \(s\) can be lowered by a neighbor state, the update
is postponed by placing the latter state in the OPEN list for later examination [Lines 27-28].

**Algorithm 11 Focused D* Algorithm**

1. function *FocusedD*_\(^\ast\)_\text{search}( )
2. \(s = \text{MIN\_STATE}\)
3. if \(s = \text{NULL}\) then
4. \(\text{return} \ NO \ - \ VAL\)
5. \(\text{val} = (f(s), k(s))\)
6. \(k_{\text{val}} = k(s)\)
7. *FocusedD*\(^\ast\).Delete(\(s\))
8. if \(k_{\text{val}} < g(s)\) then
9. for all \(s' \in \text{Succ}(s)\) do
10. if \(t(s') \neq \text{NEW}\) and \(\text{LESSEQ}((f(s'), g(s')), \text{val})\) and \(g(s) > g(s') + c(s', s)\) then
11. \(g(s) = g(s') + c(s', s)\)
12. \(\text{parent}(s) = s'\)
13. if \(k_{\text{val}} = g(s)\) then
14. for all \(s' \in \text{Succ}(s)\) do
15. if \(t(Y) = \text{NEW}\) or \((\text{parent}(s') = s\) and \(g(s') \neq g(s) + c(s, s')\)) or \((\text{parent}(s') \neq s\) and \(g(s') > g(s) + c(s, s')\)) then
16. *FocusedD*\(^\ast\).Insert(\(s', g(s) + c(s, s')\))
17. \(\text{parent}(s') = s\)
18. else
19. for all \(s' \in \text{Succ}(s)\) do
20. if \(t(s') = \text{NEW}\)
21. or \((\text{parent}(s') = s\) or \(g(s') \neq g(s) + c(s, s')\)) then
22. *FocusedD*\(^\ast\).Insert(\(s', g(s) + c(s, s')\))
23. \(\text{parent}(s') = s\)
24. else
25. if \(\text{parent}(s') \neq s\) and \(g(s') > g(s) + c(s, s')\) and \(t(s) = \text{CLOSED}\) then
26. *FocusedD*\(^\ast\).Insert(\(s, g(s)\))
27. else
28. if \(\text{parent}(s') \neq s\) and \(g(s') > g(s') + c(s', s)\)
29. and \(t(s') = \text{CLOSED}\) and \(\text{LESS}(\text{val}, (f(s'), g(s'))\)) then
30. *FocusedD*\(^\ast\).Insert(\(s', g(s')\))
31. \(\text{return} \ MIN\_VAL()\)
32. end function

**6.1.3 D* lite**

Built on the Lifelong Planning A\(^\ast\) (LPA\(^\ast\)) method [Koenig et al., 2004a], D* lite is a widely used incremental search algorithm as it calculates an optimal-cost path at least as efficient as D\(^\ast\) but using a simplified algorithmic procedure [Koenig and Likhachev, 2005b].
6.1. Incremental path planning algorithms for dynamic navigation environments

During the search procedure, the D* lite algorithm keeps a heuristic estimate \( h(s) \) for state \( s \), similar to the one used by the A* algorithm. It also uses the following two estimates of the objective function:

- The g-value \( g(s) \), representing the estimated cost of state \( s \) to the goal
- A one step lookahead estimate \( rhs(s) \) which is based on the g-values of the successors of state \( s \). It is defined by eq. 6.1 and aims to have a better informed cost evaluation for state \( s \).

\[
rhs(s) = \begin{cases} 
0 & \text{if } s = s_{goal} \\
\min_{s' \in \text{succ}(s)} (g(s') + \text{cost}(s', s)) & \text{otherwise}
\end{cases}
\]  \hspace{1cm} (6.1)

When running D* lite, a state can be classified as consistent if its g-value is equal to its rhs-value, or inconsistent otherwise. Specifically, a state is defined as overconsistent if \( g(s) > rhs(s) \) and underconsistent if \( g(s) < rhs(s) \).

Similar to other algorithms from the D* family, it uses an OPEN list to expand efficiently the states in the search tree. The priority queue is populated only with inconsistent states, which are ordered based on a priority key function, given in eq. 6.2. This is the optimized version of D* lite proposed in [Koenig and Likhachev, 2005b], which avoids recalculating the key-values and reordering the OPEN list each time the start node is updated. This is achieved by keeping a lower bound on the priority-values using a key modifier variable \( k_m \). A lexicographic ordering is used on the priority keys, so that \( \text{key}(s) < \text{key}(s') \) if \( k_1(s) < k_1(s') \) or both \( k_1(s) = k_1(s') \) and \( k_2(s) < k_2(s') \).

\[
\text{key}(s) = [k_1(s); k_2(s)] = [\min(g(s), rhs(s)) + h(s) + k_m; \min(g(s), rhs(s))]
\]  \hspace{1cm} (6.2)

Algorithm 12 provides the supplementary functions used in D* lite. The key modifier variable \( k_m \) is initialized to 0 and the g and rhs values of all states are initialized to \( \infty \), except the goal state whose rhs-value is set to 0. Therefore, at the beginning of the search, the goal state is the only inconsistent state and is inserted in the OPEN list. Further, the UpdateState procedure is given [Lines 11-15], which is used to insert or update each inconsistent state or remove from the OPEN list when a consistent state is encountered. The main loop of the algorithm repeatedly processes states in the OPEN list [Lines 16-37] until either the key function of the state to be processed is higher than the key function of the start state or the start state becomes underconsistent. Each time a state is removed from the priority queue, its key value is checked first. This is due to the fact that when the rover moves between perceptions, states in the priority queue are not reordered and only the \( k_m \) variable is updated. Therefore, if the key value of the processed state \( k_{old} \) is lower than it should be, then its key value and position in the priority queue are updated. When an overconsistent state is removed from the OPEN list, it becomes consistent [Line 25] and all the successors of the current state are updated correspondingly [Lines 27-29]. When the processed state is underconsistent, its g-value is set to \( \infty \), thus making it overconsistent. Further on, all successors of the current state whose rhs-values depend on the old g-value of the current state are updated [Line 33-36].
Algorithm 12 D* lite Algorithm - Management functions

1: function CalculateKey(s)
   2:   return \[ \min(g(s), rhs(s)) + h(s_{Robot}, s) + k_m \cdot \min(g(s), rhs(s)) \]
   3: end function

4: function Initialize( )
   5:   OPEN = ∅
   6:   \(k_m = 0\)
   7:   for all \(s \in S\) do rhs(s) = ∞; g(s) = ∞
   8:   rhs\(_{goal}\) = 0
   9:   D* lite.Insert\(s_{goal}, [h(s_{Robot}, s_{goal}); 0]\)
10: end function

11: function UpdateState(s)
   12:   if g(s) \neq rhs(s) and s \in OPEN then D* lite.Update(s, CalculateKey(s))
   13:   if g(s) \neq rhs(s) and s \notin OPEN then D* lite.Insert(s, CalculateKey(s))
   14:   if g(s) = rhs(s) and s \in OPEN then D* lite.Remove(s)
15: end function

16: function ComputeShortestPath( )
   17:   while OPEN.TopKey() < CalculateKey(s_{Robot}) or rhs(s_{Robot}) > g(s_{Robot}) do
   18:     s = OPEN.Top()
   19:     \(k_{old} = OPEN.TopKey()\)
   20:     \(k_{new} = CalculateKey(s)\)
   21:     if \(k_{old} < k_{new}\) then
   22:       OPEN.Update\(s, k_{new}\)
   23:     else
   24:       if g(s) > rhs(s) then
   25:         g(s) = rhs(s)
   26:         D* lite.Remove(s)
   27:       for all \(s' \in Succ(s)\) do
   28:         if \(s' \neq s_{goal}\) then rhs\(s'(s) = \min(rhs(s'), c(s', s) + g(s))\)
   29:         UpdateState\(s'\)
   30:     else
   31:       \(g_{old} = g(s)\)
   32:       g(s) = ∞
   33:       for all \(s' \in Succ(s) \cup s\) do
   34:         if rhs\(s'(s) = c(s', s) + g_{old}\) then
   35:           if \(s' \neq s_{goal}\) then rhs\(s'(s) = \min_{s'' \in Succ(s)}(c(s', s'') + g(s''))\)
   36:           UpdateState\(s'\)
   37:     end function
The main function of D* lite is presented in Algorithm 13. An initial run of the `ComputeShortestPath()` [Line 5] routine determines the minimal cost path from the current position of the rover to the given goal. The rover starts following the computed path [Line 10] until new perceptions of the environment are available. When changes in the navigation environment are perceived, first the key modifier value $k_m$ is recalculated and all cost changes are integrated in the search tree to find the new minimum cost path. If at any moment the $rhs$-value of the current rover state is equal to $\infty$, it means that the rover cannot reach the selected target and the algorithm stops.

**Algorithm 13: D* lite Algorithm**

1: function $D^*$ lite search()
2:   $s_{last} = s_{start}$
3:   $s_{Robot} = s_{start}$
4:   Initialize()
5:   `ComputeShortestPath()`
6:   while $s_{Robot} \neq s_{goal}$ do
7:     if $rhs(s_{Robot}) = \infty$ then
8:       return "No path found"
9:     else
10:        $s_{Robot} = \arg \min_{s \in Succ(s_{Robot})} (c(s_{Robot}, s) + g(s))$
11:        $S_{changed} = \text{Changed cost}(S, s_{Robot})$
12:        if $S_{changed} \neq \emptyset$ then
13:           $k_m = k_m + h(s_{last}, s_{Robot})$
14:           $s_{last} = s_{Robot}$
15:           for all directed edges $(s, s')$ with changed cost do
16:              $c_{old} = c(s, s')$
17:              Update the edge cost $c(s, s')$
18:              if $c_{old} > c(s, s')$ then
19:                 if $s \neq s_{goal}$ then
20:                     $rhs(s) = \min(rhs(s), c(s, s') + g(s'))$
21:                 else
22:                     if $rhs(s) = c_{old} + g(s') and s \neq s_{goal}$ then
23:                        $rhs(s) = \min_{s'' \in Succ(s)} (c(s, s'') + g(s''))$
24:            end if
25:        end if
26:   end while
27: end function

**Property 6** `ComputeShortestPath()` of the D* lite expands at most twice each state and thus terminates. It expands at most once when the state is underconsistent, and at most once when the state is overconsistent.
6.1.4 Other D* variants

There are other variants of the D* algorithms which can be applied to path planning for planetary robotic exploration. Some of them address the problem of optimal path planning and replanning under global constraints [Stentz, 2002], while others use a so-called anytime approach [Likhachev et al., 2005a]. Anytime algorithms typically construct very quickly an initial feasible but not optimal solution. The quality of the solution is then improved during the rest of available path planning time. However, this is not the approach to be used for path planning for robotic planetary exploration due to the fact that a slightly suboptimal solution can have a high impact on the mission return.

A global path planner which was already used for space robotics applications is the Field D* algorithm [Ferguson and Stentz, 2005a][Carsten et al., 2009]. It is an interpolation-based path planning and replanning algorithm which is able to generate any-angle paths through grid navigation maps. However, it has been proven that path solutions provided by Field D* are susceptible to unnecessary heading changes [Daniel et al., 2010]. Moreover, the path planning and replanning process of Field D* algorithm takes almost two times longer than D* lite. Therefore, this study does not include the Field D* algorithm as the computation time is one important performance measure. The advantage of the use of an interpolation-based path planner is relative to the resolution of the grid navigation map. The global navigation map of the Field D* path planner integrated into MER flight software has a resolution of 40cm [Carsten et al., 2009], while the navigation map used in the CNES EDRES software has a resolution of 5cm. The use of higher resolution navigation maps limits the amount of sharp heading changes for the locomotion system in the planned path, reducing the performance increase when using an interpolation-based path planner.

6.2 Performance evaluation in Matlab

A preliminary study is performed to compare the performance of the aforementioned incremental path planning algorithms. The state of the art claims that the use of a successive A* path planner can be faster for easy navigation problems (in which the rover reaches the target with only a small number of searches) [Hernández et al., 2012]. The aim of this experiment is to decide if the use of an incremental search algorithm is suitable for global path planning for planetary exploration rovers. The tests were performed in MATLAB on a Intel Core i5-4200U CPU with 4GB of RAM. The path planning configurations are characterized by two different navigation map sizes (100 × 100 and 500 × 500) with different given percentages of randomly placed point-sized obstacles (30%, 50%, 70%). Each path planning query consists of a given start position, always at the center of the navigation map, and a randomly generated goal position on the border of the navigation map.

The overall navigation strategy is described as follows. The rover starts with an initial empty navigation map (all states are considered to be navigable) and calculates the shortest path from its current state to the goal. It executes the path until either it reaches successfully the target, or at least an obstacle is perceived. The rover performs perceptions after each one-cell displacement. During a perception it can observe the neighborhood states within a given
radius. Whenever the rover observes an obstacle state, the navigation map is updated and the shortest path to the goal is recalculated. The procedure stops when either the rover reaches the goal state, or no path is found to reach the target.

Table 6.1 summarizes the results of the first experiment, where the navigation map consists of $100 \times 100$ cells. This setting provides easy path planning queries, as the Euclidean distance between the start and goal position ranges from 50 to 70.71 cells. The obstacle occupancy rate of the navigation map are either 30%, 50% or 70% and the perception range of the rover is within a radius of 5 or 10 cells. Table 6.1 includes information regarding the path length, the number of replanning steps, the number of state expansions and the runtime until the target is reached. The results are averaged over 500 path planning queries.

<table>
<thead>
<tr>
<th>Occupancy rate</th>
<th>Field of view</th>
<th>Algorithm</th>
<th>Runtime (s)</th>
<th>Length (cells)</th>
<th>Replanning steps</th>
<th>State expansions</th>
</tr>
</thead>
<tbody>
<tr>
<td>30%</td>
<td>5</td>
<td>Repetitive A*</td>
<td>0.43</td>
<td>75.40</td>
<td>50</td>
<td>995</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D*</td>
<td>2.85</td>
<td></td>
<td></td>
<td>4206</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Focused D*</td>
<td>1.86</td>
<td></td>
<td></td>
<td>877</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D* lite</td>
<td>1.42</td>
<td></td>
<td></td>
<td>740</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>Repetitive A*</td>
<td>0.54</td>
<td>73.81</td>
<td>46</td>
<td>1043</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D*</td>
<td>3.60</td>
<td></td>
<td></td>
<td>4925</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Focused D*</td>
<td>2.57</td>
<td></td>
<td></td>
<td>974</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D* lite</td>
<td>2.36</td>
<td></td>
<td></td>
<td>783</td>
</tr>
<tr>
<td>50%</td>
<td>5</td>
<td>Repetitive A*</td>
<td>0.57</td>
<td>83.74</td>
<td>62</td>
<td>1332</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D*</td>
<td>3.29</td>
<td></td>
<td></td>
<td>4905</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Focused D*</td>
<td>2.68</td>
<td></td>
<td></td>
<td>1711</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D* lite</td>
<td>2.22</td>
<td></td>
<td></td>
<td>1414</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>Repetitive A*</td>
<td>0.69</td>
<td>83.15</td>
<td>51</td>
<td>1715</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D*</td>
<td>5.31</td>
<td></td>
<td></td>
<td>6692</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Focused D*</td>
<td>3.85</td>
<td></td>
<td></td>
<td>2165</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D* lite</td>
<td>3.79</td>
<td></td>
<td></td>
<td>2141</td>
</tr>
<tr>
<td>70%</td>
<td>5</td>
<td>Repetitive A*</td>
<td>1.03</td>
<td>92.3</td>
<td>70</td>
<td>2100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D*</td>
<td>3.99</td>
<td></td>
<td></td>
<td>5324</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Focused D*</td>
<td>3.36</td>
<td></td>
<td></td>
<td>2507</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D* lite</td>
<td>3.06</td>
<td></td>
<td></td>
<td>1845</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>Repetitive A*</td>
<td>1.56</td>
<td>91.4</td>
<td>62</td>
<td>3139</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D*</td>
<td>5.99</td>
<td></td>
<td></td>
<td>7562</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Focused D*</td>
<td>4.67</td>
<td></td>
<td></td>
<td>3368</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D* lite</td>
<td>3.38</td>
<td></td>
<td></td>
<td>2862</td>
</tr>
</tbody>
</table>

Table 6.1: Replanning performance comparison on random grids of $100 \times 100$

Results shown in Table 6.1 support the claims stated in [Hernández et al., 2012]. It is noticed that in all cases the successive A* search runs faster. However, the implementation of the algorithms in this test is not optimized, and this can affect the total computation time. Therefore, the performance comparison is performed with respect to the average number of ex-
Chapter 6. Incremental path planning using constant resolution global navigation maps

Expanded states as they have a direct influence on the computation time and memory use. $D^*$ lite has the lowest number of expanded states among all the incremental search algorithms, similar to the performance of the successive $A^*$ approach. The percentage of obstacle occupancy of the navigation map has a direct influence on the performance of all compared algorithms. An increased obstacle occupancy rate results in a higher probability to encounter obstacles on the minimum cost path towards the target. This leads to more replanning steps and longer final paths. Similarly, when using a larger perception field of view (i.e. 10 cells), the average number of expanded states increases as more obstacle cells are perceived and included in the navigation map. On the other hand, this reduces the final path length, as a wider field of view provides better informed paths. Finally, a linear relationship is observed between the perception range and the average number of replanning steps.

The main interest of this section is to evaluate the feasibility of global path planning by using such incremental search algorithms. Therefore, the performance of the aforementioned algorithms is analyzed when using a larger navigation map, which results in more replanning steps per path planning query. Table 6.2 provides the results obtained throughout the second test, where a navigation map with a size of $500 \times 500$ cells was used.

<table>
<thead>
<tr>
<th>Occupancy rate</th>
<th>Field of view</th>
<th>Algorithm</th>
<th>Runtime (s)</th>
<th>Length (cells)</th>
<th>Replanning steps</th>
<th>State expansions</th>
</tr>
</thead>
<tbody>
<tr>
<td>50%</td>
<td>5</td>
<td>Repetitive $A^*$</td>
<td>22.12</td>
<td>409.91</td>
<td>323</td>
<td>47630</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$D^*$</td>
<td>272.33</td>
<td></td>
<td></td>
<td>94390</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Focused $D^*$</td>
<td>120.01</td>
<td></td>
<td></td>
<td>19584</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$D^*$ lite</td>
<td>16.59</td>
<td></td>
<td></td>
<td>11123</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>Repetitive $A^*$</td>
<td>26.48</td>
<td>407.29</td>
<td>316</td>
<td>68712</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$D^*$</td>
<td>329.94</td>
<td></td>
<td></td>
<td>105380</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Focused $D^*$</td>
<td>163.06</td>
<td></td>
<td></td>
<td>21395</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$D^*$ lite</td>
<td>18.92</td>
<td></td>
<td></td>
<td>15566</td>
</tr>
<tr>
<td>70%</td>
<td>5</td>
<td>Repetitive $A^*$</td>
<td>40.65</td>
<td>443.57</td>
<td>396</td>
<td>70721</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$D^*$</td>
<td>217.80</td>
<td></td>
<td></td>
<td>97889</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Focused $D^*$</td>
<td>175.29</td>
<td></td>
<td></td>
<td>29605</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$D^*$ lite</td>
<td>27.10</td>
<td></td>
<td></td>
<td>19639</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>Repetitive $A^*$</td>
<td>51.17</td>
<td>436.93</td>
<td>385</td>
<td>103116</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$D^*$</td>
<td>297.79</td>
<td></td>
<td></td>
<td>110326</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Focused $D^*$</td>
<td>182.07</td>
<td></td>
<td></td>
<td>33850</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$D^*$ lite</td>
<td>28.88</td>
<td></td>
<td></td>
<td>21577</td>
</tr>
</tbody>
</table>

Table 6.2: Replanning performance comparison on random grids of $500 \times 500$

Due to the increase of distance between the start and goal positions (from 250 to 353 cell lengths) and thus the exponential increase of search trees during path planning, the performance of the successive $A^*$ approach decreases drastically. Therefore, in this scenario, the local replanning performed by the $D^*$ lite algorithm provides the best performance among all considered algorithms, both in computation time and in the average expanded states per search. Similar to the previous test, it is shown that the obstacle occupancy of the navigation environment and
the coverage of the perception system have direct influence on the overall performance of the path planning algorithms.

### 6.3 Performance evaluation of the D* lite in the CNES EDRES environment

In order to emphasize the advantages of performing global path planning using an incremental path search approach, the D* lite algorithm is implemented in the CNES EDRES software environment and tested for realistic scenarios of robotic planetary exploration. Figure 6.1 shows a comparison of final executed paths when using successive local A* planning and the D* lite algorithm on an updated global navigation map. In this test the D* path planner is given an initial global navigation map at low resolution to have a better informed guidance of the rover. The initial global map contains navigation data at 1m resolution and it is updated locally when new perceptions are taken and used to calculate local navigation data at a resolution of 2.5cm. The final executed paths are shown over the high resolution navigation map of the SEROM Mars Yard, where bright gray shades represent hazardous areas and dark gray shades represent navigable regions.

![Figure 6.1: Comparison of executed trajectories when using A* or D* lite with initial low resolution global navigation map](image)

Two main differences can be observed for the executed paths using the two approaches. First, the D* lite uses the initial low resolution map to plan in advance the avoidance of known obstacles, while the successive local A* planner needs the rover to approach the hazardous areas before circumventing them. In many cases this results in time and energy consuming in-place-turn maneuvers, as highlighted by the light blue square. Second, a dead-end configuration
may pose difficulties to the successive local A* planner. In the example shown in Figure 6.1, the selected goal location is outside the large dead-end configuration. Therefore it cannot be reached through successive local A* path planning steps using the local navigation map due to its limited coverage. Conversely, when using the D* lite with the given global navigation map, the target is successfully reached after a traveled distance of 140 m. The rover would be able to reach the target using consecutive global A* path plans on the global navigation map, but this option was not taken into consideration due to computation time limitations.

The performance of the global path planning approach using the D* lite algorithm is also compared with the global path planning approach (LRN) currently under development at CNES, as detailed in Section 3.6. A set of 171 pairs of coordinates for mission scenarios has been randomly generated, with Euclidean distance between start and goal positions ranging from 20 m to 100 m on the SEROM DEM. The occurrences of each possible result status during mission scenario execution are shown in Figure 6.2, while the main statistical results of the performance measures (path length and average navigability of the terrain underlying the computed paths) are given in Table 6.3. The types of errors which are returned by the EDRES autonomous navigation architecture are not necessarily due to the failures in the path planning algorithms. Most of them represent situations where obstacles dilated in the navigation map cover the current rover position (ROVER_IN_UEFH) or the rover is surrounded by dangerous areas (RV_SURROUND_NNAV). Due to low risk allowance during robotic planetary exploration missions, when such situations occur, the autonomous navigation procedure is stopped.

On one hand, the global path planning approach using D* lite has an increased success rate over LRN. However, paths executed using LRN are shorter. This is due to the fact that the D* lite algorithm uses the traversability cost function defined in eq. 3.1 which is affected by the drawback of the "equal cost area" detailed in Section 3.5. Moreover, paths provided by the D* lite algorithm are limited by the grid-based representation which provides heading changes with increments of $\pi/4$. The LRN approach uses the partial tangent graph algorithm in order to provide the sub-goal position for the local A* path planner in an any-angle manner towards the global goal. Contrary, the D* lite, avoids the use of intermediate sub-goals eliminating thus the need of intermediary procedures like greedy best search or primitive path calculation through post-processing. Finally, the quality of the global path provided by D* lite is also influenced by inaccurate navigation values outside the field of view of the rover (due to the low resolution initial global navigation map). Nevertheless, the usage of an initial low resolution navigation map in D* lite leads to the generation of paths over easier to navigate terrain.

<table>
<thead>
<tr>
<th>Performance measure</th>
<th>Min</th>
<th>Max</th>
<th>Mean</th>
<th>Median</th>
<th>Variance</th>
<th>Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path length wrt to LRN</td>
<td>49.62%</td>
<td>409.28%</td>
<td>105.87%</td>
<td>102.89%</td>
<td>980.12%</td>
<td>31.31%</td>
</tr>
<tr>
<td>Average difficulty wrt to LRN</td>
<td>97.82%</td>
<td>101.35%</td>
<td>98.01%</td>
<td>98.43%</td>
<td>1.54%</td>
<td>1.24%</td>
</tr>
</tbody>
</table>

Table 6.3: Performance evaluation of trajectories provided by the D* lite algorithm

Figure 6.3 provides a configuration for which the effect of different initial global navigation maps for the D* lite algorithm is analyzed. The path executed when using the LRN method is also shown. For D* lite, initial global navigation maps are given differently, as follows:

- The navigation environment is totally unknown and is considered to be obstacle-free and
6.3. Performance evaluation of the D* lite in the CNES EDRES environment

Figure 6.2: Distribution of the result status for the comparison D* lite - LRN

Figure 6.3: Comparison of executed trajectories when using LRN or D* lite

totally plain. The optimistic case is considered by initializing all the global navigation map with the minimum navigable value possible in EDRES.

- The navigation environment is totally unknown and is considered to be obstacle-free but the roughest possible that the rover can traverse. Therefore, the pessimistic case is considered by initializing the global navigation map with the maximum navigable value possible in EDRES.

- The navigation environment is partially known. This simulates the case where it is possible to use orbiter navigation information to initialize the global navigation map. In this test, the navigation map of the SEROM Mars Yard degraded at a resolution of 1m was used.

Figure 6.4 provides a comparison of the computation time of each case. It is clearly noticed in the upper graph of Figure 6.4 that for each version of D* lite, the first iteration is very computationally expensive. This is due to the fact that it has to build the initial search tree using the initial global navigation map to calculate the corresponding optimal path. This is not a problem in the case of a robotic exploration application, as the initial search tree can
be calculated offline and uploaded at the same time as the initial global navigation map. The bottom graph of Figure 6.4 displays the computation times for the rest of the iterations, when D* lite only updates the optimal path while it includes the information provided by the latest perception. In this case, it is concluded that the computation times are comparable to the one of the LRN method. It should be highlighted that the workload of the $D^*$ lite is much higher (as it provides at any moment the optimal path with respect to the navigation values in the global navigation map), while the LRN approach assures an optimal path only from the current position of the rover to the selected sub-goal, and not a global optimality. Among the three cases of $D^*$ lite, the pessimistic case has the shortest computation time per iteration. This is due to the intense use of Property 6 in order to reduce the number of state expansions. All states are initialized at their maximum possible cost, making them all overconsistent. This means that when a new perception is available, if a given state remains navigable, it can only reduce its cost resulting in only one expansion. In the optimistic case, all states are underconsistent resulting in two expansions per state.

An in-depth comparison between the optimistic and pessimistic cases is provided in Figure 6.5. The values in the initial global navigation map are translated in path cost estimates when the initial search tree is built. When the minimum navigation value is used (optimistic case), the influence of the cost estimate is lower than the heuristic estimate and the search tree becomes more focused on the expanding states which minimize the target estimated distance. On the other hand, when the maximum navigation value is used (pessimistic case), the cost estimate has a higher influence over the heuristic estimate. Therefore the search is less focused on the target, resulting in a wider initial search tree. This is represented in Figure 6.5 which displays the number of consistent (CLOSED list) and inconsistent (OPEN list) states at each search iteration. It is shown that in the optimistic case the search tree consists of a minimum number of states (both consistent and inconsistent). In the pessimistic case, the first search generates approximately five times more consistent states (Closed list). During replanning steps, consistent states might become inconsistent (decrease in size of the Closed list) but no new states in the search space are initialized.

Figure 6.5: Evolution of the number of consistent (Closed list) and inconsistent states (Open list) during $D^*$ lite run
The evolution of the amount of initialized states, therefore used memory, is shown in Figure 6.6. It shows a memory use of five orders of magnitude higher in the pessimistic case compared to the optimistic one. During replanning steps, a decrease in the memory use by the pessimistic approach is observed. When new perceptions are available, states which were initially considered navigable are observed as hazardous areas and so removed from the search tree. In the optimistic case, there are situations where the memory use increases during replanning steps (i.e. iteration 11). This occurs when the rover observes for the first time the obstacles on the top of the path, as shown in Figure 6.3. This results in a peak of the computation time in Figure 6.4 and of the memory use (Figure 6.6) needed to allocate new states in the search tree in order to provide a path which avoids the hazardous areas.

Figure 6.6: Comparison of memory use when running D* lite

6.4 Conclusions

This chapter provides a detailed analysis regarding the use of incremental path planning algorithms to perform global path planning for planetary exploration rovers. First, two experimental tests are undertaken to assess the performance of path planning algorithms from the D* family with respect to a successive global A* path planning approach. It is shown that as the size of the navigation environment increases and more replanning steps are performed, incremental path planning algorithms provide better overall performance with respect to the number of expanded states. This led to the selection of the D* lite algorithm for implementation and testing for robotic planetary exploration mission scenarios.

Following, the importance of performing path planning using a global navigation map is laid out. It is shown that over-the-horizon targets might not be reached by only performing successive local path planning using a local navigation map with limited coverage. Conversely, the use of a global navigation map which covers the region that the rover can explore during a sol or more increases the target reaching rate and the quality of the executed path. Throughout a Monte Carlo experiment on the SEROM DEM, it is shown that the global path planning approach using D* lite can provide better performance (success rate, terrain difficulty) compared to the LRN method. However in what concerns the computational load, the D* lite performs
replanning slower than the entire LRN global path planning approach. Moreover, the memory use of the D* lite does not comply with robotic planetary exploration mission-constraints. Thus, only the global path planning process using D* lite can require up to 50 MB of onboard memory use. As a reference, the entire autonomous navigation procedure developed at CNES which performs successive local path planning, described in Section 3, uses maximum 5 MB of onboard memory. Therefore, it is concluded that the D* lite algorithm is not suitable for global path planning for robotic planetary exploration missions. For this reason, a new navigation map representation is introduced in Section 7, which is used to perform global path planning complying with computation time and memory-use constraints.
Chapter 7

Global path planning using a multi-resolution navigation map representation

Both navigation systems developed at NASA and CNES for planetary exploration rovers use a grid representation for the navigation maps as it is a simple and computationally efficient way to describe the terrain around the rover. A drawback of the uniform high resolution grid representation is that it bounds the coverage area with a limited memory capacity. One way to overcome this issue is to use multi-resolution mapping. By representing the close proximity of the rover at high resolution and far areas at low resolution the coverage of the navigation map can be increased without augmenting the memory use. In addition, the use of a high resolution environment representation close to the rover will not affect the path planning performance.

This chapter suggests the use of a novel multi-resolution navigation map representation which can cover the area the rover can explore during a sol at the same memory load as the local navigation map representation used by CNES. First, details on the map construction and update are given. Following, path planning algorithms currently developed in EDRES can make use of this navigation map representation in order to reach global path planning capabilities. Finally, a terrain aware global path planner is proposed, which is able to autonomously decide the path search direction with respect to the obstacle distribution and appearance. This aims to reduce the size of the search tree and thus to reduce the computational load and memory use of the global path planner.

7.1 Multi-resolution navigation map representation

The use of a multi-resolution representation of the navigation environment has been extensively studied during recent developments in the field of over-the-horizon path planning for robotic applications. A widely used method in this context employs quad-trees [Noborio et al., 1990] [Kambhampati and Davis, 1986], or framed quad-trees [Yahja et al., 1998], by performing a dyadic recursive decomposition of the navigation space. One shortcoming of the quad-trees
is that a path generated on such a representation can consist of many sharp heading changes which are very costly to execute. The framed quad-tree representation addresses this issue by adding high resolution cells around all low resolution cells, which requires a high computational load and probably augmented onboard memory use. Other multi-resolution decomposition techniques refer to the use of triangular cells [Hwang et al., 2003] or a hierarchic representation of spheres encapsulating the rover for collision avoidance [Verwer, 1990].

The discrete wavelet transform is a powerful and computationally effective tool for multi-resolution cell decomposition of the obstacle space. One of the first works concerning the use of wavelets to represent 3D static maps for path planning is proposed in [Pai and Reissell, 1995]. Following, wavelet representation has been widely employed for hierarchical path planning as reported in [Tsotras and Bakolas, 2007] [Cowlagi and Tsiotras, 2010].

Other multi-resolution approaches concern a grid representation of variable resolution as proposed in [Behnke, 2003]. The resolution of the map is high in the proximity of the rover and decreases farther away from it. Montemerlo and Thurn [Montemerlo and Thrun, 2004] address the problem of outdoor terrain modeling using a pyramid comprising of maps at different resolution. Finally, a multi-resolution map created in polar coordinates is used for path planning for miniature air vehicles [Yu et al., 2009].

These methods have been extensively used in terrestrial and aerial robotics but have not yet proven their pertinence for planetary exploration applications. One approach has addressed the challenge of multi-resolution path planning for Martian exploration rovers with limited onboard computation resources [Carsten et al., 2007] by using two grid-based representations. This approach aims to improve the performance of the GESTALT architecture [Maimone et al., 2004] and to guide the MERs around obstacles which expand on regions beyond the coverage of the local navigation map. In addition to the goodness map created by GESTALT, the proposed approach builds a cost map representing the navigation environment at a global scale. Recall that the goodness map is always centered on the rover location and covers only a local area around it (12 m x 12 m with a grid cell resolution of 20 cm). Each grid cell translates the difficulty of the navigation environment in a goodness value, with high values for easy-to-traverse terrain and low values for hazardous areas. Unlike the navigation map, the cost map is fixed to the navigation environment and stores navigation information over larger areas, but at lower resolution to tackle the limited available memory (50 m x 50 m at a cell resolution of 40 cm). Opposite to the goodness map, each grid cell of the cost map represents the cost of traversing the width of the cell, with lower values for easily traversable terrain and higher values for dangerous regions. Further, the cost map is used by a global path planner in order to provide a better informed sub-goal to GESTALT arc selection algorithm. A Field D* algorithm is used to provide the optimal path between the selected goal and the rover position over the updated cost map.

This approach was integrated and tested in extended mission on the Opportunity rover [Carsten et al., 2009]. It consisted in five experimental fully autonomous runs. The aim of the first experiment was to generate the Field D* telemetry during a straight drive of approximately 10 meters over plain terrain and to assess the proposed path solutions. In most of the cases the suggested solutions were consistent with the actual drive commands. The second test was a partial failure as the rover could not reach a target position behind a hazardous area. While the first half of the drive went well until the dangerous area was reached, the hazard avoidance
7.2 Multi-resolution map creation in the EDRES environment

The navigation map has to represent the largest possible area around the rover in order to ensure the generation of optimal paths to reach an over-the-horizon target. However, the size of the navigation map used in the EDRES environment is limited due to the memory constraints of a robotic planetary exploration mission. This section introduces a new approach for representing large areas within a limited size navigation map. It allows the rover to drive over a long distance while avoiding large obstacles and hazardous terrain configurations like dead-ends, that cannot be bypassed using a local path planner.

The creation of the local navigation map is detailed in Section 2.2.2. Overall, a high resolution navigation map is built in the configuration space encoding the navigation difficulty to traverse the represented terrain. The three steps to calculate the navigation value are recalled here: the wheel positioning and inclination tests as shown in Figure 2.7 and security margin inclusion. A cell in the navigation map is considered to be navigable only if one can pose the center of the rover abstract model on it at different orientations without violating the maximum lateral inclination constraint. This results in a navigation map constructed in the configuration space, having as effect the expansion of any hazardous area by the rover radius in all directions. It means that the resulting navigation map has always an exterior band of unclassified cells as the inclination test cannot be performed, resulting in a waste of memory use for its representation. An example of such navigation map is shown in Figure 7.1, where the aforementioned exterior band is visible on the top, right and bottom of the image. The width of this band depends on the radius of the abstract rover model and the resolution of the navigation map as shown in Figure 2.6.

This section proposes to transform the local navigation map into a multi-resolution representation so that the coverage of the fixed size navigation map is increased. The navigation map representation is split in two different areas as shown in Figure 7.2: an internal region, in the close proximity of the rover, where the navigability tests in Section 2.2.2 can be applied (marked in light blue), and an external band (marked in dark gray). The internal region, also called the high resolution area, corresponds to the local navigation map which contains navigation information with respect to the terrain roughness and traversal difficulty. The novelty here is the use of the external band, also called low resolution region, to store low resolution information regarding obstacles far away from the rover.

Let $\text{val}_{HRES}$ be the resolution of the internal region given by the resolution of the local
navigation map in EDRES and \( \text{val}_{LRES} \) be the chosen resolution for the external band, with \( \text{val}_{HRES} < \text{val}_{LRES} \). From the abstract rover model radius \( RR \), indicated in Figure 7.2a the width of the low resolution band can be calculated as shown in eq. 7.1.

\[
\text{width}_{LRB} = \text{round}\left( \frac{RR}{\text{val}_{HRES}} \right) \text{ cells} \tag{7.1}
\]

The proposed method uses three types of low resolution cells in function of their position in the low resolution band. A square multi-resolution navigation map consisting of \( \text{size}_{NAV} \times \text{size}_{NAV} \) cells is considered. Then, the resolution of a cell at \((x,y)\) coordinates in the low resolution band, with \( 1 \leq x \leq \text{size}_{NAV} \) and \( 1 \leq y \leq \text{size}_{NAV} \), is defined as in eq. 7.2. The corresponding coverage of each type of low resolution cell is represented in Figure 7.2b.

\[
\text{res}_\text{cell}(x,y) = \begin{cases} 
\text{val}_{HRES} \times \text{val}_{LRES}, & \text{if } (x > \text{width}_{LRB}) \land (x < (\text{size}_{NAV} - \text{width}_{LRB})) \\
\text{val}_{LRES} \times \text{val}_{HRES}, & \text{if } (y > \text{width}_{LRB}) \land (y < (\text{size}_{NAV} - \text{width}_{LRB})) \\
\text{val}_{LRES} \times \text{val}_{LRES}, & \text{otherwise}
\end{cases} \tag{7.2}
\]

The advantage of such multi-resolution representation is that the 8-neighborhood connectivity between cells holds even on the border between the two distinctive regions, as shown in Figure 7.2b. This allows the navigation map to remain consistent and to be easily handled by the path planning algorithm originally applied to the uniform grid representation. On the contrary, when using other multi-resolution representations such as quad tree, the path planning algorithm has to locally calculate the list of neighbors for a given state.

The proposed multi-resolution representation for the navigation map has the objective to increase the covered region with a limited memory capacity. Figure 7.3 provides a comparison of the represented area by the navigation map, with a size of 351×351 cells in EDRES, depending
Figure 7.2: Example of multi-resolution navigation map representation: (a) with two distinctive areas (a high resolution area with the rover model positioning test for the cell marked in red, and a low resolution band), (b) Cell connectivity at the border between different resolution regions.

on the chosen resolution values of the two distinct regions, and the used rover models. Given $size_{NAV}$, $RR$, $val_{HRES}$ and $val_{LRES}$ the side length of the square region covered by the multi-resolution navigation map is calculated using eq. 7.3.

\[
side\_length = 2 \times width_{LRB} \times val_{LRES} + (size_{NAV} - 2 \times width_{LRB}) \times val_{HRES}
\] (7.3)

The coverage of the uniform resolution navigation map increases linearly, reaching a maximum of $17.5m \times 17.5m$ when each cell in the navigation map represents a square of $50mm \times 50mm$. When using the multi-resolution representation, the covered area can be increased up to approximately $200m \times 200m$ when $val_{HRES} = 25mm$ and $val_{LRES} = 2000mm$. Depending on the used rover model, the highest coverage of the multi-resolution navigation map is achieved when using the EXOMARS rover model due to its larger size, reaching an area of approximately $250m \times 250m$. However, when increasing the value of $val_{HRES}$ up to $50mm$ the represented region is decreased to $134.55m \times 134.55m$ which covers the area the planetary exploration rover can explore during a sol. The advantage of this representation is that the coverage of the multi-resolution navigation map can be decided based on the length of the planned mission traverse, by adjusting the resolution values correspondingly.

Each cell in the low resolution band of the multi-resolution navigation map is assigned a traversability score. This value is given as a function of the obstacle occupancy of the represented area, without considering the terrain difficulty. Let $min\_nav\_val$ and $max\_nav\_val$ represent the minimum and maximum of the navigation values used in the high resolution navigation map. The range of the values representing navigable areas is defined as $nav\_val\_range = max\_nav\_val - min\_nav\_val$. Then, the traversability score for cells in the low resolution band is determined
Figure 7.3: Navigation map coverage, depending on the chosen values for $val_{HRES}$ and $val_{LRES}$ and the used rover model according to Table 7.1. To simplify the computation, the obstacle occupancy of each cell in the low resolution band is calculated using axis aligned bounding boxes enclosing each hazardous area. If a cell is fully covered by the bounding box of a hazardous area the obstacle $val$ used to represent obstacles in the navigation map is assigned. Otherwise, it is considered to be navigable with a navigation value relative to the obstacle occupancy.

Due to the shape of the low-resolution band and the different coverage of each cell depending on its position, 8 regions are distinguished on the low resolution band, as follows:

- Four corner regions: on the top left and right (marked NW, NE in Figure 7.4) and on the bottom left and right (marked SW, SE), each containing $width_{LRB} \times width_{LRB}$ cells, where each cell represents an area of size $val_{LRES} \times val_{LRES}$.

- Two regions, one on top (marked N) and one on the bottom (marked S) which contain $width_{HRES} \times width_{LRB}$ cells, where $width_{HRES}$ represents the size of the high resolution
7.2. Multi-resolution map creation in the EDRES environment

<table>
<thead>
<tr>
<th>Occupancy</th>
<th>Traversability score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-20%</td>
<td>min_nav_val</td>
</tr>
<tr>
<td>20-40%</td>
<td>min_nav_val + 20%nav_val_range</td>
</tr>
<tr>
<td>40-60%</td>
<td>min_nav_val + 40%nav_val_range</td>
</tr>
<tr>
<td>60-80%</td>
<td>min_nav_val + 60%nav_val_range</td>
</tr>
<tr>
<td>80-100%</td>
<td>min_nav_val + 80%nav_val_range</td>
</tr>
<tr>
<td>100%</td>
<td>obstacle_val</td>
</tr>
</tbody>
</table>

Table 7.1: Traversability score for the low resolution cells.

given by \( width_{HRES} = size_{NAV} - 2 \times width_{LRB} \). Each cell in these two regions have a coverage of \( val_{HRES} \times val_{LRES} \).

- Two regions, one to the left (marked W) and one to the right (marked E), which contain \( width_{LRB} \times width_{HRES} \) cells, each cell covering a region of size \( val_{LRES} \times val_{HRES} \).

Figure 7.4: Example of multi-resolution navigation map (right) created from a high resolution global navigation map (left)

Figure 7.4 provides an example of the proposed multi-resolution representation for a global navigation map acquired during a simulated mission scenario. On the left figure the high resolution global navigation map (which covers an area of 39m × 45m) is shown. The right figure displays the corresponding multi-resolution navigation map which represents the same area with limited memory use, with \( val_{HRES} = 50mm \) and \( val_{LRES} = 2000mm \). The brown thick square in the global high resolution navigation map represents the corresponding area to the high resolution region in the multi-resolution representation. Thin red lines over the global navigation map indicate the limits of the regions covered by low resolution cells. Therefore red square regions indicate the surface covered by each cell in the four corner regions of the low resolution band in the multi-resolution navigation map representation. For visualization purposes some dimensions of the the affected regions of the low resolution band are magnified by ten as marked.
on the image. Each cell in the low resolution band has assigned a traversability score as defined in Table 7.1. Red regions represent blocked cells due to their full obstacle occupancy, while blue tones are used to mark cells depending on their obstacle occupancy, from light blue for low obstacle occupancy to dark blue.

### 7.3 Multi-resolution map update

During an exploration mission, the rover keeps updating the multi-resolution map which is further used during the path planning stage. The fact that the local navigation map is always centered at the rover position imposes to update the low resolution band with the encountered obstacles far away from the rover, which otherwise would be lost. Two approaches to update the low resolution band of the multi-resolution navigation map are suggested in this work. The first approach uses information only from the high resolution region of the navigation map. In this approach, the low resolution band is shifted correspondingly and updated with the bounding boxes of the hazardous areas which would go out of the range of the high resolution navigation map after a rover displacement. The second method uses an intermediate obstacle map, which is updated at each new perception by extracting obstacle contours from the high resolution region of the navigation map using a segmentation algorithm.

#### 7.3.1 Update through successive shifts

The first proposed method to update the low resolution band of the multi-resolution navigation map after each displacement is given in Algorithm 15 and illustrated in Figure 7.6. The first stage is to identify the region to be shifted from the high resolution region to the low resolution band due to the rover displacement in between two consecutive perceptions. Then, a segmentation algorithm is applied on this area to compute axis-aligned bounding boxes of all obstacle regions. Finally, the obstacle occupancy of each low resolution cell on the shift-affected area is calculated by using the axis-aligned bounding boxes. The rest of the low resolution band is updated through a corresponding shift of the already existing values.

Figure 7.5 provides an illustration for the shift procedure during low resolution band update. In this example, a low resolution cell (in red) covers a region five times larger than a high resolution cell (in blue) does. Algorithm 14 provides the procedure to update the affected low resolution cells after a shift. Two variables are defined: $count_{LRES}$ denotes the number of low resolution cells which are entirely affected by the commanded shift and $count_{HRES}$ represents the rest of high resolution cells which will partially affect the corresponding low resolution cell. Then, each cell is in the low resolution band is updated [Line 4], where $c[i]$ represents the obstacle occupancy of the cell with index $i$.

![Figure 7.5: Illustration of shift procedure](image.png)
Algorithm 14 Shift procedure

1: \( \text{count}_{LRES} = \text{floor} \left( \frac{\text{shift}}{\text{val}_{LRES}} \right) \)
2: \( \text{count}_{HRES} = \text{shift} - \text{count}_{LRES} \)
3: \( \text{for } i = 1 : (w_{LRB} - \text{count}_{LRES} - 1) \text{ do} \)
4: \( c[i] = \frac{\text{val}_{HRES} - \text{count}_{HRES}}{\text{val}_{HRES}} \times c[i + \text{count}_{LRES}] + \frac{\text{count}_{HRES}}{\text{val}_{HRES}} \times c[i + \text{count}_{LRES} + 1] \)
5: \( c[w_{LRB} - \text{count}_{LRES}] = \frac{\text{val}_{HRES} - \text{count}_{HRES}}{\text{val}_{HRES}} \times c[w_{LRB}] + \text{shift}[1 : \text{count}_{HRES}] \)

Algorithm 15 Procedure to update the multi-resolution map through successive shifts

1: IN : current 2D navigation map \( \text{NavMap} \), previous multi-resolution map \( \text{PrevMresMap} \)
2: OUT : current multi-resolution map \( \text{MresMap} \)
3: VAR : shift \( \text{shift} \), obstacle bounding boxes \( \text{obs} \), number of obstacles \( \text{nbobs} \)
4: \( \text{obs} = \text{Segmentation}(\text{NavMap}, \text{shift}) \)
5: \( \text{Update}(\text{PrevMresMap}, \text{shift}) \)
6: \( \text{for all } \text{crtobs} : 1..\text{nbobs} \text{ do} \)
7: \( \text{Update}(\text{PrevMresMap}, \text{obst}) \)

Figure 7.6 provides an example for the update procedure of the multi-resolution navigation map using this Successive Shift (SS) method. Figure 7.6a provides the initial multi-resolution map, with the rover at its center marked with a light blue cell. The high resolution region is marked in blue, and contains three obstacles shown in black. The low resolution band is highlighted in red, representing the coverage of each cell. This example uses a low resolution value three times bigger than the high resolution one, with \( width_{LRB} = 2 \). Figure 7.6b shows the area covered by the high resolution region of the multi-resolution navigation map after a displacement of 4 high resolution cells to the right. The area highlighted in light green gets out of the range of the high resolution region and is further used to update the low resolution band. First, the obstacles are segmented and their axis-aligned bounding boxes are computed (marked in dark green in Figure 7.6b). These bounding boxes are used to calculate the obstacle occupancy of the affected low resolution cells, as shown in Figure 7.6c in order to determine the traversability score of each low resolution cell. Darker shades of gray represent areas with higher obstacle occupancy. Then, another displacement of 2 high resolution cells to the right is performed (Figure 7.6d). First, the cell occupancy of the affected low resolution cells is updated (Figure 7.6e) and the corresponding traversability score is set (Figure 7.6f). A supplementary buffer is required to store the obstacle occupancy of all cells in the low resolution band at a given moment. This is used for calculating the traversability scores during map update steps.

One disadvantage of this method is that once the traversability scores for the low resolution cells are calculated, the obstacle localization information is lost. This is due to the fact that the traversability score is calculated using the obstacle occupancy of a cell, considering that the obstacles are equally spread over the represented area. This can be noticed first in Figure 7.6c where the square obstacle on bottom left generates an equal traversability score for two neighbor low resolution cells. After the second displacement, the obstacle occupancy of the low resolution cells is updated and the traversability score for the rightmost low resolution cells becomes low.
but still indicating the presence of obstacles (Figure 7.6f). However, in reality, the area covered by these two cells doesn’t contain any obstacle.

### 7.3.2 Update through an obstacle map

In order to overcome the drawback of the previously presented method, a global obstacle map, as detailed in Section 3.6, is built during the mission. Recall that this map uses subsequent local navigation maps to accumulate all encountered obstacles during a mission. When analyzing the consecutive local navigation maps, it classifies areas in two types:

- **non-navigable area** - all cells classified as obstacle in the local navigation map
- **other area** - all navigable cells in the local navigation map, or regions where no perception has been taken yet (unknown areas)

For memory saving, the global obstacle map stores only high resolution contours of the non-navigable areas without retaining the navigation difficulty scores.

The procedure for updating the multi-resolution navigation map using the global obstacle map (OM) is given in Algorithm 16. First, the region corresponding to the low resolution map is identified in the obstacle contour map. Then, the same procedure as in Algorithm 15 is applied to find bounded boxes and to calculate obstacle occupancy.

Figure 7.7 provides an example of multi-resolution navigation map update using OM for the same conditions as in Figure 7.6. Given the initial setting presented in Figure 7.7a, the
### Algorithm 16
Procedure to update the multi-resolution map using an obstacle map

1. IN : current 2D navigation map $NavMap$, obstacle map $ObsMap$
2. OUT : multi-resolution map $MresMap$
3. VAR : current obstacle given by each of the methods $crtobs$, low resolution band coordinates $lowres\_coords$
4. $Update(ObsMap, NavMap)$
5. $Segmentation(ObsMap)$
6. FOR ALL $crtobs$: 1. $ObsMap.nlobs$
7. $lres\_crtobs = Intersection(crtobs, lowres\_coords)$
8. $Update(MresMap, lres\_crtobs)$

Figure 7.7: Multi-resolution navigation map update using an updated obstacle map

corresponding global obstacle map contains the contours of all obstacles in the local navigation map as highlighted in purple. After the 4-cell displacement to the right, the obstacle map position with respect to the rover is updated and the obstacle contours are used to calculate the axis-aligned bounding boxes, as shown in Figure 7.7b. Following, the obstacle occupancy and traversability scores are calculated. After the second displacement, the same steps are performed and the traversability scores are updated, as shown in Figure 7.7f. It should be noted that the localization information stored in the global obstacle map leads to more accurate traversability scores for the cells in the low resolution band of the multi-resolution navigation map.

Figure 7.8 compares results for the multi-resolution navigation map update using the two procedures, SS and OM. The high and low resolution values were set to $val_{HRES} = 25mm$. 
Chapter 7. Global path planning using a multi-resolution navigation map representation

Figure 7.8: Example in the Rover Simulator
7.4. Global path planning on a multi-resolution navigation map

and $val_{LRES} = 2m$. The low resolution band width when using the IARES rover model is $width_{LRB} = 33$ cells. This results in a coverage of the multi-resolution navigation map of approximately $140m \times 140m$. Figures 7.8a-7.8d provide the results given by the SS update method, while Figures 7.8e-7.8h show results when using the OM procedure. Figure 7.8i displays the global obstacle map built during the run, while marking in green the poses of the rover where the updated multi-resolution navigation maps are shown.

In the case of the SS approach, the low resolution band contains obstacles, but not as prominent as they should be. As discussed for Figure 7.6, this is due to the mathematical formulation for the shift operation in Algorithm 14, which tends to wash out obstacles when representing them on areas larger than their size. On the other hand, when using the OM approach, the location information of hazardous regions is well conserved, even far away from the rover.

7.4 Global path planning on a multi-resolution navigation map

The main objective of the multi-resolution navigation map representation is to be able to store navigation information over large areas at a limited memory cost. This navigation information is further used by the path planner to calculate better informed trajectories towards the over-the-horizon target. Thanks to the advantages of the multi-resolution grid representation, one can extend the applicability of the grid-based optimized path planner developed by CNES in order to perform global path planning. In this section, four grid path planning approaches which make use of the multi-resolution navigation map representation are proposed and their performance is compared with the two existing methods for local and global path planning of CNES, introduced in Section 3.

The performance of a grid path planner is directly influenced by the obstacle distribution in the navigation environment. One disadvantage is that grid path planners have poor performances when concave obstacles are encountered. Therefore, during a path search, all states which are located in the concave regions of the encountered obstacles have to be expanded, resulting in an undesirable overload of the search query. The impact of the concave obstacle configurations can be reduced by changing the direction of the path search.

Figure 7.9 represents the results obtained using two path planners ($A^*$ and Dijkstra) with two search directions for a given grid configuration. In each image, the start location is marked with a red cell and the goal position with a green cell. The navigation environment contains a concave obstacle marked with black cells. The rest of the cells in this representation are considered navigable at cost zero. Therefore the cost estimation for a given cell is equal to the path length of reach it. Cells marked in light blue represent the developed cells during a given path search (cells in the CLOSED list) while cells left in the OPEN list are highlighted with dark blue. Finally, the calculated path for each query is shown in yellow.

First the performance of the forward and inverse $A^*$ algorithm is evaluated, and the results are shown in Figures 7.9a and 7.9b. Due to the presence of the concave obstacle, the overload of the search algorithm is higher in the case of the forward $A^*$ algorithm (336 expanded cells) compared to the inverse $A^*$ algorithm (239 expanded cells). This results in lower computation performance of the forward $A^*$. This effect is even more important in the case of the Dijkstra
algorithm, where 126 cells are expanded during the forward search (Figure 7.9c) with respect to only 32 in the case of the inverse search (Figure 7.9d).

Two first approaches analyzed in this section concern the use of the A* path planning algorithm directly on the multi-resolution navigation map. The multi-resolution A* path planner will simultaneously check the reachability of the mission goal and calculate an optimal global path to reach it. Depending on the search direction, the two path planners are entitled multi-resolution forward A* planner (MRFA*) and multi-resolution reversed A* planner (MRRA*).

In an obstacle-free environment, the two approaches find an identical path. However, when the rover has to drive around large and complex obstacles, the resulting path and the computational load of the two approaches might be different.

The disadvantage of the multi-resolution A* path planner is that the number of cells to be expanded during a search can grow exponentially with respect to the length of the optimal path. This does not comply with the computation time requirements of a planetary exploration rover. Hence, it is more computationally efficient to perform a guided local A* search. Similar to the path planning architecture proposed by CNES, a Dijkstra search is performed first on the multi-resolution navigation map. This allows to check the reachability of the mission goal, to calculate a primitive path for further guidance path and to define a better informed sub-goal in the high resolution region of the multi-resolution navigation map. Then, the local optimal A*
path planner is used to calculate an optimal path to reach the selected sub-goal. In this way, the task of global path planning is performed on the proposed multi-resolution representation at a much lighter computational load. Depending on the direction of the multi-resolution Dijkstra search, two approaches are called forward guided local A*(FGLA*) and reverse guided local A*(RGLA*).

The performances obtained with the proposed four multi-resolution path planning strategies are compared to the path planners proposed by CNES: the successive local path planning technique (SLA*) and the long range navigation planner (LRN), both introduced in Section 3. Table 7.2 provides a summary of all considered path planning strategies.

Table 7.2: Path planning strategies summary

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Sub-goal selection</th>
<th>Optimal path</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRFA*</td>
<td>No</td>
<td>Multi-resolution Forward A*</td>
</tr>
<tr>
<td>MRRA*</td>
<td>No</td>
<td>Multi-resolution Reverse A*</td>
</tr>
<tr>
<td>FGLA*</td>
<td>Multi-resolution Forward Dijkstra</td>
<td>Forward A*</td>
</tr>
<tr>
<td>RGLA*</td>
<td>Multi-resolution Reverse Dijkstra</td>
<td>Forward A*</td>
</tr>
<tr>
<td>SLA*</td>
<td>Forward Dijkstra</td>
<td>Forward A*</td>
</tr>
<tr>
<td>LRN</td>
<td>Partial tangent graph</td>
<td>Forward A*</td>
</tr>
</tbody>
</table>

7.4.1 Experimental evaluation

A simulation study has been conducted with the CNES EDRES simulator on the HiRISE DEM consisting of 177 path planning configurations with randomly generated starting and mission goal positions within an Euclidean distance ranging from 20m to 50m. The navigation map covers a region of $17.55m \times 17.55m$ when used with a constant resolution of 50mm, and $87.75m \times 87.75m$ when it uses the multi-resolution representation with a low resolution of 2m and the width of the low resolution band equal to 18 cells.

Figure 7.10 compares the overall success rate between the four proposed and two existing path planning strategies. It can be easily seen that approximately in 20% of the test cases, the SLA* strategy fails to reach the mission target mainly due to the limited coverage of the local navigation map (represented by the TOO_SMALL_NM error). This failure rate can be reduced by using the multi-resolution approaches and the LRN, justifying the importance of the higher coverage of the navigation map.

Easy configurations for which SLA* finds a solution

First, the 80% of cases with "easy" path planning configurations, in which the SLA* strategy could find a solution, are analyzed. Here, it is proven that the performance attained by the proposed multi-resolution approaches are not degraded compared to those of the SLA*. Two measurements are defined in order to assess the path planning performance: the path length and the average difficulty of the traverse.

Figure 7.11 shows histograms of the performance ratio of each of the multi-resolution strategies with respect to the SLA*. Values lower than 1 imply better performance measures for the multi-resolution methods, while values greater than 1 signify the contrary. From these results,
Chapter 7. Global path planning using a multi-resolution navigation map representation

Figure 7.10: Synthesis of results obtained during the tests

it can be concluded that the multi-resolution approaches execute paths of similar length to the ones provided by SLA*. However, it is noticed that better difficulty cost is obtained for the strategies which perform the global A* search on the entire multi-resolution navigation map. On the contrary, the Gaussian distributions for the ratio of used memory by each multi-resolution path planning approach with respect to SLA*, shown in Figure 7.12, indicate a higher memory load when performing global A* search. This can result in an excessive onboard memory use in average two to three times higher than SLA*. This is not the case of the multi-resolution guided local A* approaches, which exhibit an improvement of onboard memory use for FGLA*.

Complex configurations for which SLA* fails

Now, the results for the cases with complex environment configurations in which the SLA* strategy failed to find a solution are analyzed. As already shown in Fig. 7.10, the multi-resolution strategies are able to provide a solution even in such complex configurations. Since SLA* did not provide a solution in such cases, the performance analysis is done with respect to the paths generated by the LRN method. Table 7.3 shows the statistical distribution of the ratios between the performance measurements provided by the multi-resolution approaches and that of the LRN method. It is observed that the paths generated by the multi-resolution approaches are longer than the LRN ones, but with less difficulty of the traversed terrain. The LRN method provides shorter paths because the sub-goal is computed using the partial tangent graph. The current rover location and the mission goal are connected on a shortest distance criterion without considering the roughness of the navigation environment. On the other hand, the multi-resolution approach degrades the high resolution details of the contours in the obstacle map when updating the low resolution band. Taking into account this information degradation and the performance improvement in the difficulty of the traversed terrain shown in Table 7.3 it can be concluded that the proposed multi-resolution approaches provide promising results for global path planning tasks. However, due to onboard memory use constraints, only the
7.5 Terrain aware global path planning using a multi-resolution navigation map

All grid path planners take into account obstacle presence when performing path search, but disregard the shape properties of the obstacles. As shown in Figure 7.9, concave obstacle configurations have direct effect on the total number of the expanded states during path planning.

Figure 7.11: Performance evaluation of the multi-resolution methods with respect to SLA*

multi-resolution local guided A* strategies are maintained and further evaluated for global path planning for planetary exploration rovers.

7.5 Terrain aware global path planning using a multi-resolution navigation map

Figure 7.12: Gaussian distributions for the ratio of memory use by each of the proposed multi-resolution path planning approach with respect to SLA*

---

129
Table 7.3: Performance evaluation of the multi-resolution methods with respect to LRN, for the failure cases of SLA∗

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Nb_traj</th>
<th>length_{Algorithm} / length_{LRN}</th>
<th>norm_{Algorithm} / norm_{LRN}</th>
</tr>
</thead>
<tbody>
<tr>
<td>FGLA∗</td>
<td>9</td>
<td>1.32 ± 0.45</td>
<td>0.99 ± 0.03</td>
</tr>
<tr>
<td>RGLA∗</td>
<td>20</td>
<td>0.98 ± 0.29</td>
<td>1.00 ± 0.04</td>
</tr>
<tr>
<td>MRA∗</td>
<td>12</td>
<td>1.10 ± 0.20</td>
<td>0.99 ± 0.04</td>
</tr>
<tr>
<td>MRRA∗</td>
<td>13</td>
<td>1.04 ± 0.24</td>
<td>0.99 ± 0.04</td>
</tr>
</tbody>
</table>

Besides the use of environment clearance, terrain slope and roughness in the path planners, this thesis suggests a method which characterizes the shape of encountered obstacles to decide the search direction of the path planner to optimize the computational load.

Algorithm 17 Procedure to decide the path planning direction

1: IN : current multi-resolution navigation map NavMap
2: IN : Current rover position in the multi-resolution map RoverPos
3: IN : Mission goal position in the multi-resolution map GoalPos
4: OUT : Search_direction={forward, reverse}
5: VAR : current obstacle given by each of the methods crtobs, low resolution band coordinates lowres_coords
6: Obstacles = Segmentation(NavMap)
7: Selected_obstacles = ObstaclesOnPath(Obstacles, RoverPos, GoalPos)
8: Score_direction = 0
9: for all CrtObst in Selected_obstacles do
10: \( C = \text{Circularity\_measure}(\text{CrtObst}) \)
11: if \( C < \text{Circularity\_threshold} \) then
12: \( \text{ConvexHull} = \text{Calculate\_convex\_hull}(\text{CrtObst}) \)
13: \( \text{Score\_direction} = \text{Score\_direction} + \text{Score\_concaveness}(\text{CrtObst}, \text{ConvexHull}) \)
14: if \( \text{Score\_direction} \leq 0 \) then
15: \( \text{Search\_direction} = \text{forward} \)
16: else
17: \( \text{Search\_direction} = \text{reverse} \)
18: return Search_direction

For each path planning query, the path planning direction selection procedure given in Algorithm 17 identifies first all obstacle regions in the multi-resolution navigation map. Second, it finds the set of obstacles which lie on the straight line from the actual rover position to the mission goal location. For each of these obstacles the compactness measure, as given in eq. 7.4, where \( A \) and \( P \) represent the obstacle area and perimeter, is used.

\[
C = \frac{4 \times \pi \times A}{P^2}, \text{ with } C \in (0, 1] \quad (7.4)
\]

Compact obstacles with no important concave regions on their contour have a compactness
measure close to 1. Since such obstacles do not have a major impact on the computation load of grid path planners, they are not taken into account in this procedure. If the shape of a given obstacle contains concave regions (compactness measure lower than a given threshold), it will be taken into account for the search direction selection. First the convex hull is calculated for each concave obstacle and used to compute a concaveness score. It encapsulates information regarding the size and position of the concave regions on the contour of the obstacle with respect to the current search configuration (start and goal positions). The concaveness score takes a positive value if the forward search is favorable for the current path planning configuration. Otherwise, it takes a negative value if the reverse search direction is favored. In this manner, after all selected obstacles have been treated, the suggested path planning direction can be determined based on the sign of the accumulated concaveness value called the search direction score.

The performance of five approaches to calculate the concaveness score for a given obstacle is analyzed.

1. **Center of gravity displacement (CGD):** This approach analyzes the vector given by the center of gravity of the obstacle and that of its convex hull. This vector is projected on the straight line between the start and goal positions, and the concaveness score for the current obstacle is given by its magnitude. Figure 7.13 provides an example where the contour of the obstacle is drawn in black and the corresponding convex hull is shown in green. The black and green circles represent the center of gravity of the obstacle and that of the convex hull, respectively, and define the center of gravity displacement vector shown in black. The concaveness score for the current obstacle is given by the magnitude of its projection (marked in blue) on the straight line between the start and goal positions. For the given example, a forward search direction is suggested.

![Figure 7.13: Concaveness score calculation using the Center of gravity displacement (CGD) approach](image)

2. **Concave regions distribution (CRD):** The second approach analyzes the distribution of the concave regions over the contour of the given obstacle, as shown in Figure 7.14. For each edge of the convex hull of the contour, a vector is defined from the center of gravity of the obstacle \(CG_C\) to the center of gravity of the subtended concave region \(P_i\). The magnitude of this vector is equal to the area of the concave region. In Figure 7.14, arrows in black represent the vectors defined for three concave regions on the obstacle contour.
and $P_R$ represents the resultant vector. The magnitude of the projection of the resultant vector on the straight line from the start to the goal location (shown in blue) provides the concaveness score for the current obstacle. In this case an inverse search direction is suggested.

![Figure 7.14: Concaveness score calculation using the Concave regions distribution (CRD) approach](image)

3. **Opposite concave regions comparison (OCR):** The third method identifies first the edges of the convex hull which are intersected by the straight line from the start to the goal position. Then, the areas of the concave regions subtended by these edges ($A_1$, $A_2$) are compared. In Figure 7.15, the concave regions to be compared are shaded in gray. The area of the concave region is considered in this method because it is a good estimate of the number of cells which would be expanded by a grid path planner. In the example in Figure 7.15, due to the difference between the areas of the two concave regions ($A_1 > A_2$), an inverse search direction is suggested.

![Figure 7.15: Concaveness score calculation using the Opposite concave regions comparison (OCR) approach](image)
4. **Shortest distance to circumnavigate (SDC):** This approach identifies first the two vertices of the convex hull which lie at the extremes of the obstacle contour with respect to the straight line from the start to the goal position, and calculates the distance from each extreme vertex to the straight line. Figure 7.16 provides a representation of this approach where the extreme vertices are marked with $E_1$ and $E_2$ and the corresponding distance to the straight line is given by $d_1$ and $d_2$. The side of the obstacle with the closer extreme vertex is chosen and all concave regions between the selected extreme vertex and the straight line from the start to the goal positions are identified. The cumulative areas of the selected concave regions, shaded in gray in Figure 7.16, are compared on either side of the chosen extreme vertex and the search direction is decided. In the case of the configuration shown in Figure 7.16, an inverse search direction is suggested.

![Figure 7.16: Concaveness score calculation using the Shortest distance to circumnavigate (SDC) approach](image)

5. **Extended shortest distance to circumnavigate (ESDC):** Similar to the previous method, the shorter distance to circumnavigate the obstacle is identified. Then, the cumulative area of the concave regions in the range of $min(d_1,d_2)$ on either side of the straight line between from the start to the goal locations are used to calculate the concaveness score. Such an example is give in Figure 7.17, where a forward search direction is suggested.

![Figure 7.17: Extended shortest distance to circumnavigate (ESDC)](image)

### 7.5.1 Performance evaluation

A Monte Carlo test was conducted to evaluate the performance of global path planning using the proposed multi-resolution navigation map and the search direction prediction accuracy. The experiment consists of 100 global path planning queries with randomly generated start and mission goal positions on the HiRISE DEM, which generated driven paths with lengths ranging from 5m to 83.5m. The test includes a total of 1234 single query path planning configurations. For each single query path planning configuration, both the forward and backward global multi-
resolution path planners are run and their performance with respect to the number of expanded states is logged. At the same time, search direction scores are calculated using the five approaches proposed above in order to predict the best path planning search direction. The summary of the resulting prediction accuracy is presented in Figure 7.18.

In order to evaluate the proposed methods, the set of single query path planning configurations where an important decrease in expanded states was achieved is identified. This is accomplished by using a threshold of interest as defined in eq. 7.5. It relies on the number of expanded states for a given single query configuration during path planning using both forward and inverse directions.

\[
\text{threshold}_\text{of}_\text{interest} = \frac{\text{abs}(\text{expanded states}_\text{fwd} - \text{expanded states}_\text{inv})}{\min(\text{expanded states}_\text{fwd}, \text{expanded states}_\text{inv})}
\] (7.5)

Figure 7.18 provides the estimation accuracy for the proposed method for different values of the \text{threshold}_\text{of}_\text{interest}. For example, when using a \text{threshold}_\text{of}_\text{interest} = 5\% the path planning estimation accuracy is evaluated for all cases where the number of expanded states for one search direction can be improved by at least 5\% if the opposite search direction is used. However, 5\% is a very small improvement and the focus is to identify cases where high improvement rates can be realized using accumulated concaveness scores of encountered obstacles. The prediction accuracy is increased up to 70\% for the first two approaches (CGD, CRD) when a \text{threshold}_\text{of}_\text{interest} of 50\% is used. For the last three proposed methods, a lower prediction accuracy of approximately 60\% or higher, is achieved. This is considered to be a natural evolution, because the first two approaches take into account the global shape of the encountered obstacles, and the last three ones are local approaches (consider only a subset of concave regions on the obstacles contours).
7.6 Nonholonomic terrain aware global path planning for a planetary exploration rover

The main objective of this thesis is to propose a global path planning architecture which addresses the mission-specific constraints of a robotic planetary exploration mission. The proposed system (NMRES) makes use of a multi-resolution global navigation map which can provide at least one-sol coverage using a bounded $351 \times 351$ cells grid representation. Besides the limited memory use for storage, it has a huge advantage of limiting the search space for the subsequent path planners which guarantees bounded computation time and memory use. Guided local nonholonomic path planning is performed to generate better informed paths which minimize the path length and difficulty, reducing in-place-turn maneuvers during trajectory execution. The sub-goal location in the high resolution region is provided by a greedy best first search algorithm run on the multi-resolution navigation map. The search direction of the greedy best first search algorithm is decided using the proposed methods for terrain-aware automatic search direction selection.

In this section, the performance of the proposed autonomous navigation architecture is compared with the two existent path planning methods under development at CNES. This test uses three scenarios considered to be difficult for robotic exploration: driving through a canyon, reach a mission target outside a dead-end configuration and reach a target while avoiding a large hazardous region.

7.6.1 First scenario: Canyon traversal

The first scenario consists in the traverse of a canyon-like configuration on the SEROM Mars Yard. The aim of this test is to assess the capability of the proposed global path planner to lead the rover towards an over-the-horizon target in a consistent way. The rover is expected to follow the shortest and easiest trajectory through the canyon to reach the selected goal position at an Euclidean distance of 41.450 m. Figure 7.19a shows the final executed trajectory when
the proposed nonholonomic global path planner is used. The corresponding high resolution navigation map for the traversed area is shown in Figure 7.19b, along with an overlay of all executed trajectories for the three different path planning architectures. Dark areas represent safe-to-navigate areas, with shades of gray encoding the roughness of the terrain (dark for plain to bright for rough) and white regions are keep-out zones. Table 7.4 provides a summary of the performance measures for each of the executed trajectory.

![Image](image.png)

(a) Trajectory executed using the proposed nonholonomic global path planner (NMRES)

(b) Trajectories over the navigation map

Figure 7.19: Canyon traversal: Driven trajectories

During the run, the global multi-resolution navigation map was updated 22 times. Each time, the selected multi-resolution path planner provided a sub-goal for the nonholonomic path planner. Similarly, the LRN achieved the goal after 22 iterations while the SLA* needed 23 iterations. One important performance measure is the average number of expanded states per path planning query which has direct influence on the computational load. This scenario provides easy path planning queries as no major obstacles block the way of the rover during the canyon traversal. However, only NMRES and SLA* choose the solution which results in a total driven trajectory length of approximately 45 m. A difference of 100 orders of magnitude between the average expansions per search for SLA* and NMRES is noticeable for the result. This is due to the difference between the dense search graph built by the grid path planner used by SLA* and the sparse search graph built using a precomputed state lattice in the case of NMRES. In this case, LRN generated a sub-optimal path which doesn’t follow the canyon. This leads to computationally expensive path planning queries (the highest average of expanded states per search) due to encountered obstacles and the longest driven trajectory.

Regarding the memory use, which is relative to the number of state expansions per search, and total simulation time, NMRES performs the best. The total simulation time is also influ-
7.6. Nonholonomic terrain aware global path planning for a planetary exploration rover

Nomenclature:
(a) = Amount of path planning queries
(b) = Average state expansions per path planning query
(c) = Average memory use per path planning query (KB)
(d) = Average trajectory difficulty per path planning query (40 for plain and 99 for very rough terrain)
(e) = Amount of generated in-place-turn maneuvers during trajectory execution
(f) = Total length of executed trajectory (m)
(g) = Simulation time (s)

<table>
<thead>
<tr>
<th>Path planning algorithm</th>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
<th>(d)</th>
<th>(e)</th>
<th>(f)</th>
<th>(g)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMRES</td>
<td>22</td>
<td>886</td>
<td>78.52</td>
<td>55.45</td>
<td>0</td>
<td>45.57</td>
<td>821.34</td>
</tr>
<tr>
<td>LRN</td>
<td>22</td>
<td>12774</td>
<td>948.03</td>
<td>57.09</td>
<td>4</td>
<td>48.77</td>
<td>857.11</td>
</tr>
<tr>
<td>SLA(^*)</td>
<td>23</td>
<td>10942</td>
<td>812.12</td>
<td>55.04</td>
<td>4</td>
<td>45.32</td>
<td>835.09</td>
</tr>
</tbody>
</table>

Table 7.4: Performance measure synthesis for the Canyon traversal scenario

enced by the execution of in-place-turn maneuvers, as explained in Section 5. Both LRN and SLA\(^*\) generated 4 in-place-turn maneuvers during trajectory execution, while no in-place turn was executed in the case of the nonholonomic global path planner. The average terrain roughness underlying the executed paths is also compared. The path planned when using NMRES is more difficult to traverse than SLA\(^*\), but less than LRN.

It should be remarked that this is the only case where SLA\(^*\) reaches the selected target among the three proposed mission scenarios. This is due to the fact that the rover had just to follow the canyon walls towards the target. During the run no obstacle configuration having the size bigger than the coverage capacity of the local navigation map had to be avoided. In the following two mission scenarios, SLA\(^*\) will fail to reach the mission target because of encountered large-size complex obstacles.

7.6.2 Second scenario: Dead-end escape

The second mission scenario aims to asses the capability of the proposed global path planner to guide the rover to reach a target outside a dead-end configuration. This test uses also the SEROM Mars Yard DEM. In this scenario the rover is supposed to reach a target location outside the U shape obstacle which limits the navigation space of the IARES rover on the SEROM Mars Yard. The Euclidean distance between the starting rover position and the selected mission goal is only 10.40 m. Figure 7.20a provides the executed trajectory in the simulator using the proposed nonholonomic global path planner and Figure 7.20b shows the high resolution navigation map of the area with a comparison of the executed trajectories when using the three path planners. Table 7.5 provides a synthesis of the performance measures for the driven trajectories.

First of all, the SLA\(^*\) path planner could not find a path to reach the target. As it uses a limited coverage local navigation map, its successive local path planning results in translational oscillation. After it performs 10 in-place-turn maneuvers trying to reach the target on the other side of the encountered obstacle, the autonomous navigation systems stops the drive. Furthermore, the performance measures (average state expansions, average memory use, average path difficulty) of SLA\(^*\) are the highest with respect to the other two path planning architectures. This is because it expanded many states during each path planning query in order to find a path which avoids the big hazardous area separating the rover from the selected mission target.
Chapter 7. Global path planning using a multi-resolution navigation map representation

(a) Trajectory executed using the proposed nonholonomic global path planner (NMRES)

(b) Trajectories over the navigation map

Figure 7.20: Dead-end escape: Driven trajectories

Nomenclature:
(a) = Amount of path planning queries
(b) = Average state expansions per path planning query
(c) = Average memory use per path planning query (KB)
(d) = Average trajectory difficulty per path planning query (40 for plain and 99 for very rough terrain)
(e) = Amount of generated in-place-turn maneuvers during trajectory execution
(f) = Total length of executed trajectory (m)
(g) = Simulation time (s)
(h) = Target reached

<table>
<thead>
<tr>
<th>Path planning algorithm</th>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
<th>(d)</th>
<th>(e)</th>
<th>(f)</th>
<th>(g)</th>
<th>(h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMRES</td>
<td>22</td>
<td>8699</td>
<td>495.18</td>
<td>57.56</td>
<td>1</td>
<td>50.73</td>
<td>825.86</td>
<td>YES</td>
</tr>
<tr>
<td>LRN</td>
<td>21</td>
<td>18092</td>
<td>1342.79</td>
<td>57.71</td>
<td>4</td>
<td>48.48</td>
<td>793.17</td>
<td>YES</td>
</tr>
<tr>
<td>SLA*</td>
<td>13</td>
<td>24286</td>
<td>1802.54</td>
<td>60.08</td>
<td>10</td>
<td>25.87</td>
<td>N/A</td>
<td>NO</td>
</tr>
</tbody>
</table>

Table 7.5: Performance measure synthesis for the dead-end escape scenario

Both global path planning approaches NMRES and LRN succeeded to guide the rover to reach the mission target. Their resulting trajectories gradually follow the dead-end configuration until a navigable area is found in order to drive the rover to the other side of the obstacle. This is an important achievement for NMRES showing that even the low resolution navigation information can be very useful in guiding the rover towards the selected mission target in a complex obstacle configuration.

Comparing the performance of LRN with that of NMRES, LRN planned path reached the target using one iteration less than NMRES resulting in a shorter total length trajectory and lower total simulation time. However, NMRES has better performance measures with respect to the path planning procedure. Thus, it expands less states per search which results in a lower onboard memory use. Also, the trajectory driven using NMRES has a slightly lower average path difficulty than LRN. Due to the advantage of the nonholonomic path planning, it generated only one in-place-turn maneuver (using the procedure detailed in Section 5) during trajectory execution, while LRN executed four in-place turns.
The final mission scenario evaluates the capability of the proposed global path planner to allow the rover to navigate in a dense obstacle field to reach an over-the-horizon mission target. The simulation is performed on the HiRISE DEM and the mission target is chosen at an Euclidean distance of 25.80 m. Figure 7.21 provides the driven trajectories with the three path planners over the high resolution navigation map (Figure 7.21b) and the NMRES executed trajectory in the simulator (Figure 7.21a). It should be noted here that the initial heading of the rover is towards the top of the navigation map in Figure 7.21b. While NMRES generates a nonholonomic trajectory with limited turning radius to correct the heading towards the target during the first path planning query, the other two path planners start the trajectory execution with an in-place-turn. Table 7.6 provides a summary of the performance measures of trajectories driven using the three path planning approaches.

As in the previous mission scenario, the SLA* does not reach the mission target. After driving half the distance towards the target, it ends up being stuck at the point where the size of the encountered obstacles exceeds the coverage of the local navigation map, after 9 executed in-place-turn maneuvers. Moreover, the performance measures of SLA* show a higher computational load than those of the global path planning approaches, due to the difficult path planning configurations with respect to its capabilities.

Unlike SLA*, the two global path planning approaches lead successfully the rover towards the selected mission goal. NMRES exhibits lower state expansions, lower memory use, lower average trajectory difficulty and less in-place-rotation maneuvers generated during drive compared to LRN. On the other hand, the path solution computed by LRN is shorter, which results in a lower simulation time. This is not due to an artifact of the NMRES path planner. The executed path is longer because during driving, due to the perception of the navigation environment, the global path planner explored a possible shorter path. However, as it turned out that other obstacles blocked the way towards the target, NMRES corrected the heading of the rover using an in-place turn and continued to circumnavigate the dense distribution of obstacles. This is
Chapter 7. Global path planning using a multi-resolution navigation map representation

Nomenclature:
(a) = Amount of path planning queries
(b) = Average state expansions per path planning query
(c) = Average memory use per path planning query (KB)
(d) = Average trajectory difficulty per path planning query (40 for plain and 99 for very rough terrain)
(e) = Amount of generated in-place-turn maneuvers during trajectory execution
(f) = Total length of executed trajectory (m)
(g) = Simulation time (s)
(h) = Target reached

<table>
<thead>
<tr>
<th>Path planning algorithm</th>
<th>(a)</th>
<th>(b)</th>
<th>(c)</th>
<th>(d)</th>
<th>(e)</th>
<th>(f)</th>
<th>(g)</th>
<th>(h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NMRES</td>
<td>24</td>
<td>2194</td>
<td>138.69</td>
<td>63.75</td>
<td>1</td>
<td>51.89</td>
<td>798.21</td>
<td>YES</td>
</tr>
<tr>
<td>LRN</td>
<td>22</td>
<td>19055</td>
<td>1414.27</td>
<td>65.77</td>
<td>3</td>
<td>47.36</td>
<td>760.73</td>
<td>YES</td>
</tr>
<tr>
<td>SLA(^*)</td>
<td>23</td>
<td>24470</td>
<td>1816.16</td>
<td>77</td>
<td>9</td>
<td>43.89</td>
<td>N/A</td>
<td>NO</td>
</tr>
</tbody>
</table>

Table 7.6: Performance measure synthesis for the over-the-rocks scenario

A normal behavior, as every possible shorter path should be inspected in order to reach the mission target the fastest. Of course, low resolution orbiter navigation data can be included in the multi-resolution navigation map representation of NMRES in order to avoid such behavior and to have better informed generated trajectories.

Finally, the performance improvement due to the use of the terrain-aware automatic search direction for NMRES is evaluated. Figures 7.22, 7.23, and 7.24 provide a synthesis of the computational load of the greedy best first search algorithm of NMRES over the tree mission scenarios. The plot of each figure represents the number of expanded states, both for forward and inverse search, during each path planning iteration. On the bottom, the chosen search directions using the proposed concaveness score measures for each iteration are indicated.

![Figure 7.22: Canyon traversal: Terrain-aware automatic search direction method assessment](image)

As previously discussed, during the canyon traversal, no large obstacles obstruct the drive of the rover towards the mission target. This results in easy path planning queries, represented by
a low number of expanded states per search iteration (a maximum of 1287 of expanded states is attained during inverse search at the second iteration), as shown in Figure 7.22. For this scenario, only using forward search at each iteration the minimum total number of expanded states per mission (14617) can be attained. By using the proposed approaches for automatic search direction, the total computational load is increased from at least 4% (OCR) to maximum 7% (CGD). However, it is important to point out that cases where major computational load improvements can be made (iterations 3, 19) are fully identified by the OCR automatic search direction method.

The interest of using the automatic search direction selection is to reduce computational load for difficult path planning scenarios where large hazardous areas are to be avoided by the rover while driving towards the mission target. This is the case of the two last mission scenarios, where a maximum of expanded states of 29344 (Figure 7.23) or 19728 (Figure 7.24) can be attained. The plot in Figure 7.23 displays a high rate of difficult path planning queries during the first part of the drive. This is due to the fact that during the first seven iterations the rover plans paths which avoid the large encountered hazardous area. Once the escape from the dead-end configuration is found (iteration 8), it can be noticed that easy path planning queries are performed with similar computational load for both search directions. In this case, using only forward path planning, a total of 117315 expanded states per mission scenario is attained. This is reduced when using terrain-aware search direction selection methods like OCR, SDC and ESDC. The highest improvement rate is reached when using OCR, when the total number of expanded states is decreased by approximately 11%.

![Figure 7.23: Dead-end escape: Terrain-aware automatic search direction method assessment](image)

The third mission scenario exhibits as well an important performance improvement due to use of the terrain-aware automatic search direction selection approach. Here, the use of each of the proposed methods results in a reduction of the overall computational load of the greedy best first search algorithm compared to the use of only forward or inverse path planning. Therefore,
Chapter 7. Global path planning using a multi-resolution navigation map representation

the mission computational load of path planning when using forward search can be reduced by a maximum of 54% when using OCR. It should be also mentioned that in this case all the five proposed approaches provide similar performance improvement. This is due to the fact that the highest cost path planning queries are fully identified (iterations 7−9 and 13−16) and the search direction with the lowest computational load is correctly selected.

![Search direction selection](image)

Figure 7.24: Over-the-rocks traversal escape: Terrain-aware automatic search direction method assessment

7.7 Conclusions

This chapter provides a solution for the global path planning problem under mission-specific constraints for planetary exploration rovers. First, a novel multi-resolution navigation map representation is proposed. It mainly consists of a high resolution region in the close proximity of the rover and a low resolution band which stores obstacle occupancy information for areas far away form the rover. It has the advantage of covering large areas with the same memory load as the local navigation map defined in the EDRES environment. Two methods are proposed for the multi-resolution map update. The first one, relies on storing in the low resolution band navigation data which would get out of the coverage of the high resolution region and shifting the low resolution band correspondingly after each rover displacement. The second method creates a separate obstacle map containing the contours of all encountered hazardous areas during the drive, which is further used to update the low resolution band of the multi-resolution navigation map. The second method is chosen to be further used as it has the advantage of keeping high resolution information regarding the localization of obstacles far-away from the rover which is very important when the navigation environment consists of complex obstacle configurations.

The multi-resolution navigation map is further used to extend the use of the successive local path planning architecture proposed by CNES, SLA*, in order to perform global path planning
and to provide better informed paths for execution. Throughout a statistical test with randomly generated coordinates for mission scenarios on the HiRISE DEM, it is shown that higher success rate is achieved compared to SLA*. Also, in case of difficult mission scenarios, multi-resolution global path planning provides easier to traverse but slightly longer paths compared to LRN.

A high importance is given to the performance improvement of the global path planner when taking into account the shape and distribution of hazardous areas in the navigation environment. Five approaches are proposed to characterize the shape of the encountered obstacles in order to decide the search direction of a path planning query with the objective of reducing the computational load. Throughout a Monte Carlo experiment, it is shown that a prediction accuracy of 70% is attained when aiming to identify the cases where a computational load reduction of at least 50% can be achieved.

Finally, a full global path planning architecture (NMRES) is proposed for robotic planetary exploration mission. It uses the multi-resolution navigation map representation which can provide at least a one-sol coverage with bounded memory use. A greedy best first search is deployed on the multi-resolution navigation map to provide a better informed sub-goal location in the high resolution region of the navigation map. Finally, low cost paths are calculated using the nonholonomic path planner proposed in Chapter 5. Its performance is compared with the two existent path planning architectures under development at CNES, SLA* and LRN. Throughout three mission scenarios, it is proven that NMRES has the capability to lead the rover towards over-the-horizon mission targets. Moreover it exhibits the lowest memory use and number of state expansions and it provides the easiest to traverse paths compared to SLA* and LRN. A lower locomotion system wear and trajectory execution time is also achieved since the local nonholonomic path planner is used. Regarding the terrain-aware search direction selection, it is shown that a reduction of the computational load between 10% and 54% can be achieved for path planning queries with complex obstacle configurations.
Chapter 8

Conclusion and perspectives

The work in this thesis focuses on the autonomous path planning and navigation map representation problems under the specific constraints of a planetary exploration rover mission such as the future ExoMars by ESA. This final chapter provides a summary of the main contributions of this thesis and suggestions for future work are identified and discussed.

Two main research directions are analyzed and developed in this thesis to address the path planning problem for robotic planetary exploration. The first direction is based on the execution of successive local path planning stages by using an updated local navigation map with a limited coverage. This approach has proved to be suitable for robotic planetary exploration mission scenarios due to its low memory use and computational load. However, it can be successfully used only when the rover navigates in sparse environments. To overcome this issue, the second research direction in this thesis concerns the global path planning problem when the rover has to avoid large hazardous areas or dense obstacle fields in order to reach the selected mission target.

The first part of this thesis is devoted to the former of these research directions, that is to say, the local path planning problem. Thus, the first contribution is an improvement in performance of the successive local path planning architecture developed at CNES, tested and validated to be used for the ExoMars rover. The management of the priority queue of the optimal path planning algorithm (A*) is improved by using the binary heap data structure. It reduces the number of heap percolates when a state is inserted in the OPEN list, resulting in lower computation time for the entire path planning process. Then, the use of incremental path planning algorithms is proposed in order to reuse previously computed navigation data in successive path planning stages. In the local path planning stage, the rover calculates an optimal path towards a selected sub-goal in the local navigation map at a distance between 4 and 6 meters. However, for security reasons, only a part of this path is executed (maximum 2.4 meters) before the rover takes new perceptions and a new locomotion cycle begins. As a consequence, an important part of the previously computed path is not executed and the corresponding search sub-tree can be restored and reused for the current path planning stage. The proposed approach applies the Fringe Retrieving A* algorithm to perform path planning using restored search sub-trees. It is proven that FRA* provides an important computation time gain due to the reduction of state expansions and heap percolates. Two methods to restore previously computed search sub-trees...
are suggested and their performance are compared. The results of Monte Carlo experiments show that using the proposed incremental path planning strategy, a maximum of 80% of previously computed navigation data can be restored between consecutive path planning stages, which yields an important improvement achievement in onboard resource management.

The second contribution addresses the problem of constrained local path planning to take into account the locomotion capabilities of the rover. Nonholonomic path planning has been already used in robotic planetary exploration onboard the MERs and Curiosity rover, where only 23 possible arc-like paths are evaluated for execution. This approach highly limits the reachability of the rover in the navigation environment, and a state lattice path planner is proposed in this thesis to be used for the ExoMars rover. Evaluation has been undertaken of whether constrained optimal local path planning can be performed in compliance with the reduced computational resources and memory capacity onboard an exploration rover. First, a state lattice generator is proposed which computes a set of forward motion controls which connect discrete states in the search space. The proposed state lattice does not contain in-place turns as this would result in an counterproductive increase of the search tree. The proposed approach considers only one in-place-turn maneuver at the beginning of a path search query which aims to correct the heading of the rover towards the selected target. The state lattice is used online by a nonholonomic A* path planner to compute feasible paths which exploit the rover’s maneuverability over rough terrain. Through Monte Carlo experiments and evaluation on field navigation data, it is shown that the nonholonomic path planner outperforms the grid-based path planner developed at CNES. Due to the fact that it builds a sparse search graph, the nonholonomic path planner reduces drastically the memory use and the computation time when low branching factors are used. It is significant that, throughout simulated mission scenarios, the nonholonomic path planner reduced by approximately 40% the number of in-place-turn maneuvers during path execution. Such maneuvers increase the rover locomotion system wear and are time consuming, having a negative impact on the mission scientific return, as the distance the rover can drive during a sol is reduced.

The second part of this thesis is devoted to the global path planning problem in order to allow the rover to avoid large obstacle configurations during its traverse. The third contribution of this thesis consists of an analysis of state-of-the-art incremental path planning algorithms operating in unknown or poorly known navigation environment. The D* lite algorithm is selected to be further tested in real robotic planetary exploration mission scenarios and its performance compared to the global path planner based on the partial tangent graph algorithm (LRN) currently under development at CNES. It is concluded that the D* lite algorithm does not comply with the mission-specific constraints, being both computationally expensive and memory consuming. This is due to the fact that during replanning steps all observed states which belong to the search tree are updated. This leads to a huge amount of updated states compared to the number of states expanded by a local path planner towards a locally selected goal.

The fourth contribution consists of the development of novel multi-resolution navigation map representation. It has several advantages, as it allows the rover to store navigation data for all the area it can explore at the same memory cost as the previously used local navigation map. By using this representation, the capabilities of the local successive path planning method are extended to global path planning. Moreover, this representation excludes the use of a sub-goal.
It performs the greedy best first search to validate the reachability of the mission target in the multi-resolution navigation map representation, which is further used as path primitive for the local optimal path planner.

Finally, the fifth and final achievement of this thesis is to propose a global path planner which employs a multi-resolution navigation map representation in order to represent large environments with limited the memory use combined with the nonholonomic local path planner to calculate cost and energy efficient paths to execute. Performance increase in time and memory use is achieved by using a terrain-aware method to predict the search direction for the greedy best first search in order to minimize the effect of the obstacle field shape and distribution on the path planning performance. Three final mission scenarios are defined in order to prove the feasibility of the proposed global path planner compared to the path planning architectures currently available at CNES. First, the capability of NMRES to reach over-the-horizon mission targets is proven. It is also shown that NMRES performs path planning with the lowest memory use and state expansions and that the path solutions are the easier to navigate both with respect to the roughness of the terrain and the generated in-place-turn maneuvers during execution. Also, the terrain aware method to predict the optimal search direction proved to reduce the computational load by up to 54%. Of all five proposed approaches, the OCR method had the best improvement rate in all three mission scenarios.

Future work can be developed in the two main research directions that have been explored in this thesis. A first improvement concerns the nonholonomic path planning. The state lattice used in this thesis consists of motion controls which are equally distributed with respect to the turning radius of the motion primitive. Throughout experimental tests it was observed that the state lattice has a higher density of motion controls close the straight line and a lower density around the maximum steering capabilities of the rover. This might influence the success rate of the nonholonomic path planner. It is suggested that future work focuses on the development of new state lattices which have a homogeneous distribution of motion controls over the locomotion capabilities of the rover. This would result in a higher reachability of the state lattice and thus in increased performance of the nonhomonomic path planner. The state lattice generator introduced in Chapter 5 takes into account only information regarding position, heading and steering capabilities of the rover. It is suggested that future state lattice designs include differential constraints at the connection between motion controls to assure the feasibility of the computed path. Another interesting state lattice design would consist of different motion control sets which are chosen online by the path planner in order to vary the rover locomotion capability depending on the nature of the navigation terrain.

Another perspective consists in improving the global path planning capabilities of the autonomous navigation architecture. As of February 2013, the High Resolution Stereo Camera on ESA’s Mars Express Orbiter has mapped nearly 90% of the planet’s surface at a resolution of 10m per pixel, with selected areas of 2m per pixel. All this information could be used in the path planning process of the autonomous navigation software. Further research can study the possibility of fusioning navigation data at different resolutions (coming from different sensors) in order to update the global multi-resolution navigation map. Furthermore, a probabilistic global path planner taking into account the accuracy of the fused navigation information could be designed to calculate global optimal trajectories.
Finally, the proposed local nonholonomic path planner and global path planning architecture with terrain-aware search direction selection should be evaluated onboard the CNES rover platforms through autonomous navigation field tests. Moreover, a quantitative study should be performed using the ExoMars rover model in order to assess the real locomotion system wear during navigation on natural terrain and the influence of path length and difficulty, or in-place-turn maneuvers during execution on the overall mission scientific return.
Bibliography

[Alt et al., 1990] Alt, H., Fleischer, R., Kaufmann, M., Mehlhorn, K., Näher, S., Schirra, S.,
of the union of simple geometric figures. In Proceedings of the Sixth Annual Symposium on
Computational Geometry, SCG ’90, pages 281–289, New York, NY, USA. ACM.

Incremental algorithms for minimal length paths. In Proceedings of the First Annual ACM-
SIAM Symposium on Discrete Algorithms, SODA ’90, pages 12–21, Philadelphia, PA, USA.
Society for Industrial and Applied Mathematics.

[Baker et al., 2004] Baker, C. R., Morris, A., Ferguson, D. I., Thayer, S., Whittaker, W., Omo-

algorithm for path planning with many degrees of freedom. In Robotics and Automation,

multibody mobile robots: controllability and motion planning in the presence of obstacles.
2328–2335 vol.3.

RoboCup International Symposium, pages 332–343.


between directed and autonomous driving on the mars exploration rovers. In Thrun, S.,
Brooks, R. A., and Durrant-Whyte, H. F., editors, ISRR, volume 28 of Springer Tracts in

exploration rover surface mobility flight software: Driving ambition. In In IEEE Aerospace
Conference, page 15.


150
Bibliography


En vue de l’obtention du

DOCTORAT DE L’UNIVERSITÉ DE TOULOUSE

Délivré par : l’Institut Supérieur de l’Aéronautique et de l’Espace (ISAE)

Présentée et soutenue le 12/12/2014 par :
Alexandru RUSU

Path Planning and Autonomous Navigation for a Planetary Exploration Rover
Planification de Chemin et Navigation Autonome pour un Rover d’Exploration Planétaire

JURY
M. RENÉ ZAPATA
M. FRANÇOIS CHARPILLET
M. FAÏZ BEN AMAR
M. MICHEL DEVY
MME. YOKO WATANABE
MME. SABINE MORENO
Professeur d’Université
Directeur de Recherche
Professeur d’Université
Directeur de Recherche
Ingénieur de recherche
Ingénieur de recherche

Examinateur
Rapporteur
Rapporteur
Directeur de Thèse
Co-Directeur de Thèse
Examinateur

École doctorale et spécialité :
EDSYS : Robotique 4200046

Unité de Recherche :
CSDV - Commande des Systèmes et Dynamique du Vol - ONERA

Directeur(s) de Thèse :
M. Michel DEVY et Mme. Yoko WATANABE

Rapporteurs :
M. François CHARPILLET et M. Faïz BEN AMAR
Résumé

Dans le cadre du programme ExoMars, l’ESA va déployer un rover sur Mars dont la mission sera de réaliser des prélèvements d’échantillons par forage souterrain et de les analyser à l’aide des instruments scientifiques embarqués. Pour atteindre en toute sécurité les différents points d’intérêt où seront effectués ces prélèvements, le rover devra être capable de parcourir plus de 70 mètres par sol (jour martien) tout en respectant les limitations des communications interplanétaires. Les performances des algorithmes de navigation autonome embarqués impactent directement la réussite scientifique de cette mission.

Les logiciels de navigation embarqués sur le rover ExoMars utilisent comme référence l’architecture de navigation autonome développée par le CNES au cours de ces 20 dernières années. Ces algorithmes sont conçus pour répondre aux limitations imposées par la technologie spatiale disponible, telles que la consommation d’énergie, la capacité mémoire et la puissance de calcul.

Le premier objectif de cette thèse est d’améliorer les performances de l’architecture de planification de chemin local itératif proposée par le CNES. Tout d’abord, l’utilisation d’un planificateur incrémental de chemin local "Fringe Retrieving A∗" permettant de réduire la charge de calcul est proposée. Il est complété par l’introduction de tas binaires dans les structures de gestion de la liste de priorité du planificateur de chemin. Ensuite, les manœuvres de rotation sur place pendant l’exécution des trajectoires sont réduites à l’aide d’un planificateur de chemins non-holonomes. Ce planificateur utilise un ensemble de chemins pré-calculés en tenant compte des capacités de bruitage du rover.

Le second axe de recherche concerne la planification de chemin global d’un rover d’exploration planétaire. Dans un premier temps, la contrainte de mémoire embarquée est détendue et une étude statistique évalue la pertinence d’un planificateur de chemin de type D∗ lite. Dans un deuxième temps, une nouvelle représentation multi-résolution de la carte de navigation est proposée pour stocker de plus grandes zones explorées par le rover sans augmenter l’utilisation de la mémoire embarquée. Cette représentation est utilisée par la suite par un planificateur de chemin global qui réduit automatiquement la charge de calcul en adaptant le sens de recherche en fonction de la forme et de la distribution des obstacles dans l’espace de navigation.
Résumé étendu

1 Introduction

1.1 Missions d’exploration martienne

Au cours de ces dernières décennies, l’exploration martienne a tenu une place particulièrement importante dans les programmes scientifiques des principaux acteurs spatiaux. Pendant que les nouvelles technologies pour les vols habités sont en cours de développement, des systèmes robotiques mobiles sont déployés sur la surface martienne pour prélever des mesures scientifiques in situ. L’objectif principal de ces missions est de décrypter si les conditions environnementales passées de la planète Mars étaient favorables au développement de la vie microbienne.


1.2 Rovers d’exploration planétaire sur Mars

Sojourner est le premier rover d’exploration planétaire qui a été déployé sur Mars par l’atterrisseur Mars Pathfinder le 5 juillet 1997. Avec une durée nominale de mission de seulement 7 sols (jours martiens), Sojourner a exploré les environs pendant 83 sols. Pendant sa mission, le rover est toujours resté dépendant de son module d’atterrissage pour ses communications avec le Centre d’Opérations sur Terre qui lui fournissait le programme de ses missions scientifiques ainsi que des points de passage espacés d’environ 1 à 2 mètres pour ses déplacements. Sojourner disposait des capacités de détecter et d’éviter de manière autonome des roches, des pentes et des risques de décrochage lors de ses traversées. Pendant sa mission d’exploration, Sojourner a parcouru environ 100 mètres en restant toujours dans un rayon de 12 mètres autour de l’atterrisseur.

Les deux rovers jumeaux, Spirit et Opportunity (Mars Exploration Rovers), arrivés sur
la planète rouge en Janvier 2004, ont atteint des performances qui ont dépassé toute attente dans le domaine de l’exploration planétaire robotisée. Ils sont capables de parcourir des longues distances en utilisant les trois paires de caméras stéréo embarquées. Ces caméras leur permettent de produire des images à bord pour caractériser la texture et la forme du terrain environnant: caméras de détection d’obstacles (HazCams) montées sur l’avant et l’arrière du châssis, caméras de navigation (NavCams) et caméras panoramiques (PanCams) montées sur un mât. Comme dans le cas de Sojourner, les opérateurs humains effectuent une analyse de traversabilité du terrain et sélectionnent des points de passage que le rover doit franchir successivement pour atteindre les objectifs scientifiques désignés et situés à longue distance. Les opérateurs humains peuvent sélectionner un des trois principaux modes de conduite disponibles dans le logiciel de navigation de rovers: Directed Driving (ou Blind Drive), AutoNav et VisOdom. Le premier mode de conduite, le plus souvent utilisé, exécute de manière séquentielle une série d’instructions (distance et direction) décrivant le parcours à suivre, tel que défini par le Centre d’Opérations. Dans ce cadre, le rover peut atteindre la vitesse maximale de 124 m/h, couvrant de longues distances mais à un risque élevé. En présence de zones à risque, le rover peut utiliser ses capacités de navigation autonome (AutoNav) en effectuant une analyse de traversabilité de ses environs proches. Dans ce mode, le rover génère des modèles 3D du terrain environnant et construit une carte de navigation représentant la difficulté du terrain en termes de capacités de locomotion. Ensuite, l’algorithme GESTALT [Goldberg et al., 2002] est employé pour choisir le meilleur chemin pré-calculé pour l’exécution de trajectoire. Le temps de calcul pour la reconstruction 3D du terrain et la planification de chemin affecte la distance que le rover peut parcourir, en réduisant sa vitesse maximale qui descend entre 36 m/h et 96 m/h. Le dernier mode de navigation, VisOdom est utilisé afin d’obtenir des estimations précises de position du rover lors d’exécution de trajectoire sur un terrain très difficile. Lors de la mission de Spirit et Opportunity, c’est le mode de navigation Directed Driving qui a été principalement utilisé afin de maximiser la vitesse d’exécution et la distance parcourue.

Le dernier rover de la NASA, Curiosity, a atterri sur la surface de Mars le 6 août 2012. Le système de navigation autonome embarqué est similaire à celui utilisé pour la mission Mars Exploration Rovers. Il dispose d’un total de 17 caméras, dont deux paires de caméras de navigation (NavCams) montées sur son mât et quatre paires de caméras de détection et évitement d’obstacles (HazCams) utilisées pour la navigation autonome. Les ressources de calcul à bord comprennent 256 Mo de DRAM, 2 Go de mémoire Flash et 256 Ko de mémoire EEPROM. Cette capacité est environ 8 fois plus élevée que les ressources présentes à bord des Mars Exploration Rovers.

1.3 Le programme ExoMars

1. Introduction

La deuxième mission, prévue pour 2018, va déposer sur Mars un rover européen et une sonde russe. La particularité de ce rover est sa foreuse qui lui permettra d’analyser des échantillons extraits du sous-sol martien jusqu’à 2 mètres de profondeur. Le rover ExoMars a une durée nominale de mission de 218 sols et, en raison des contraintes de la mission, est conçu pour parcourir de longues distances avec un niveau élevé d’autonomie qui doit lui permettre d’atteindre en toute sécurité des endroits identifiés comme de grand intérêt scientifique. C’est pourquoi il est nécessaire que le rover puisse calculer de manière autonome des solutions de navigation pour pouvoir atteindre les cibles scientifiques sélectionnées par les opérateurs au sol. Cette fonctionnalité de navigation autonome limite la distance que le rover peut parcourir pendant un sol à 70 m avec une vitesse moyenne d’environ 15 m/h.

1.4 Problématique et objectifs

La navigation autonome à bord d’un rover d’exploration planétaire est une ressource très importante parce qu’il peut avoir une forte influence sur le retour scientifique de la mission. Plus le rover peut atteindre rapidement les objectifs scientifiques sélectionnés, plus il sera possible de consacrer du temps à des mesures scientifiques. Par exemple, dans le cas du rover ExoMars, il est estimé qu’un quart de la durée nominale de la mission sera consacrée aux déplacements entre les sites scientifiques sélectionnés et le reste du temps sera utilisé pour effectuer des mesures scientifiques. Grâce à des tests récents de téléopération d’un rover réalisé depuis la Station Spatiale Internationale, il a été montré que l’autonomie joue un rôle important dans l’amélioration de l’efficacité opérationnelle et d’utilisation du rover. Au cours de ces tests, le rover a pu effectuer ses tâches d’une manière robuste, sans contraintes liées aux délais de communication ou même pendant de longues périodes de perte de communication.

L’importance des capacités de navigation autonome est accentuée par la limitation des moyens de communication avec la Terre (entre 3 et 21 minutes de délai et jusqu’à deux fenêtres de communication par sol). Cela limite le nombre de commandes à distance envoyées par les opérateurs, et peut affecter l’efficacité de l’activité d’exploration. Par exemple, si le rover a des capacités de navigation autonome de haut niveau, toute la bande passante de communication peut être utilisée pour transmettre des données scientifiques, au lieu d’être utilisée par les opérateurs pour décider des trajectoires et directions pour le sol suivant. De plus, les algorithmes de navigation autonome doivent respecter les contraintes spécifiques de la mission, imposées par la technologie spatialisable disponible au moment de la mission. Par exemple, le rover Exomars va devoir mettre en œuvre le système de navigation autonome en utilisant le co-processeur LEON2 à 96 MHz, avec une capacité de mémoire embarquée de seulement 256 Mo.

Le Centre National d’Études Spatiales (CNES) est un des acteurs du spatial à l’échelle mondiale qui participe à plusieurs missions d’exploration de Mars telles que MSL, ExoMars ou encore Mars2020. Depuis plus de 20 ans, le Groupe de Robotique Spatiale du CNES a développé une architecture de navigation autonome pour un rover d’exploration planétaire, qui a été livré à l’ESA en tant que contribution à la mission ExoMars. Ce logiciel a été validé par de nombreux tests de simulation et essais sur le site martien du CNES (nommé SEROM), et notamment par deux essais de commande à distance depuis le site de l’ESA/ESTEC [Joudrier et al., 2011] [Joudrier et al., 2012].
L’objectif de cette thèse est de contribuer au système de navigation autonome développé au CNES pour un rover d’exploration planétaire par l’introduction de nouveaux algorithmes de planification de chemin et de méthodes de représentation de l’environnement pour améliorer sa performance. Deux directions principales sont abordées dans cette thèse. En premier lieu, de nouveaux procédés sont développés et validés pour la planification de chemin local afin d’obtenir une meilleure gestion des ressources embarquées (puissance de calcul et capacité de mémoire) tout en réduisant l’usure du système de locomotion du rover (en limitant les rotations sur place) et le temps requis pour l’exécution du trajet. Le deuxième objectif de cette thèse est d’inclure des capacités de cartographie et de planification de chemin globales sans enfreindre les contraintes spécifiques à la mission. Pour ce faire, une nouvelle approche à faible consommation de mémoire est proposée pour pouvoir représenter le terrain environnant qui sera ensuite utilisé par le rover pour planifier son chemin et atteindre en toute sécurité des cibles lointaines.

Tout d’abord, une présentation de l’architecture de navigation autonome développée par le CNES est fournie. Toutes les étapes intermédiaires concernant la perception et le calcul de la carte de navigation, la planification et l’exécution de chemin sont détaillées. En outre, les installations d’essai et les modèles des rovers utilisés pendant les tests de validation sont présentés. Ensuite, les algorithmes existants dans l’architecture de navigation autonome du CNES pour la planification locale et globale de chemin sont décrits. Cette partie a une grande importance parce qu’elle représente le point de départ des travaux présentés dans cette thèse.

Une première contribution propose l’utilisation des algorithmes de planification incrémentale de chemin pour des environnements de navigation statiques. Le calcul de trajectoire est accéléré par réutilisation des données de navigation entre les processus de planification de chemins consécutifs. Une structure de données optimisée est mise en œuvre pour permettre une meilleure gestion des structures de données pour les algorithmes de planification de chemin, afin de réduire le temps de calcul. Ensuite, une approche de planification de trajectoire qui prend en compte les capacités de locomotion du rover est présentée. Tout d’abord, une méthode pour générer un ensemble de trajectoires pré-calculées qui respectent les contraintes non holonomes du rover est proposée. Ces trajectoires pré-calculées sont ensuite utilisées par un planificateur de chemin optimal qui génère des chemins qui minimisent la consommation énergétique et le temps d’exécution de la trajectoire. L’objectif principal de ce chapitre est d’évaluer la faisabilité et l’applicabilité d’un tel planificateur de chemin non holonome pour un rover d’exploration planétaire.

La deuxième partie de cette thèse traite du problème de la planification globale de chemin. Une étude initiale consiste en l’analyse des algorithmes de planification incrémentale de chemin opérant dans des environnements dynamiques, qui entraîne le choix de l’algorithme D* lite qui est ensuite mis en œuvre et testé pour des scénarios de mission d’exploration planétaire. Ici, la contrainte d’utilisation de la mémoire est relâchée et une carte de navigation globale à haute résolution est utilisée pour stocker les données de navigation obtenues au cours d’un sol. Enfin, une nouvelle représentation globale de l’environnement de navigation basée sur une carte de navigation multi-résolution est proposée. Cette représentation offre l’avantage de couvrir de plus grandes surfaces avec les mêmes contraintes en termes de mémoire que la carte de navigation locale utilisée dans l’architecture de navigation autonome du CNES. Puis, un planificateur de chemin global est conçu pour réduire la charge de calcul en sélectionnant le sens de recherche à partir de la répartition et de la forme des obstacles rencontrés. De cette manière,
la contribution principale de cette thèse consiste en la proposition d’un planificateur de chemin global qui emploie une carte de navigation multi-résolution en respectant de fortes contraintes de mémoire et un planificateur non holonome de chemin local en mesure de guider le rover vers des cibles lointaines avec une efficacité énergétique améliorée.

2 Plate-forme de développement

2.1 EDRES: Environnement de Développement pour la Robotique d’Exploration Spatiale


2.2 Chaîne de navigation autonome pour un rover d’exploration planétaire

Les délais de transmission prohibitifs rendent la commande directe du rover très peu réactive, ce qui s’avère problématique sur un terrain inconnu et difficilement praticable. La chaîne de navigation autonome du rover permet à la fois de réduire les coûts énergétiques dus à la communication avec la Terre et d’augmenter significativement l’efficacité de l’exploration du terrain, et donc la distance parcourue par le rover chaque jour (augmentation du retour scientifique de la mission). Dans un scénario de mission d’exploration planétaire, le rover reçoit un jeu d’instructions définies pour la mission en cours au début de chaque sol, qui peut se composer des objectifs de locomotion (coordonnées de but de la mission, la distance à parcourir ou cap du rover) [Rastel and Maurette, 2006] et des tâches scientifiques à effectuer. Ensuite, le rover utilise ses capacités de navigation autonome pour rejoindre la cible sélectionnée en choisissant lui-même le meilleur chemin à suivre. A la fin de chaque sol, le rover transmet au centre de contrôle les données acquises au cours de la mission, y compris des images panoramiques du terrain environnant, qui sont ensuite analysées par des opérateurs humains pour décider du jeu d’instructions pour le sol suivant.

La chaîne de navigation autonome fonctionne par succession de cycles de locomotion jusqu’à ce que le but de la mission soit atteint. La première étape d’un cycle de locomotion est réalisée par le sous-système de perception. Le modèle numérique du terrain environnant est construit à
partir des images prises avec un banc stéréoscopique monté sur le mât du rover. Il est ensuite utilisé par le sous-système de navigation qui construit une carte de navigation en regroupant les informations locales liées aux capacités de locomotion du rover. Pour des raisons de capacité de mémoire embarquée limitée, la taille de la carte de navigation est limitée à $351 \times 351$ pixels, couvrant une région restreinte autour du rover. Les différents étapes à suivre pour construire la carte de navigation sont détaillées dans la Section 2.2.2, et peuvent générer jusqu'à six types de zones, représentées par la Figure 1. Ensuite, la carte de navigation est utilisée par un planificateur de chemin local qui décide la trajectoire que le rover doit suivre afin de se rapprocher de la cible sélectionnée. La dernière étape d'un cycle de locomotion est réalisée par le sous-système de contrôle d'exécution de la trajectoire. Son objectif principal est de suivre avec exactitude la trajectoire calculée en choisissant la meilleure stratégie de déplacement. Par défaut, les données issues de l'odométrie sont utilisées pour la localisation en temps réel du véhicule. Pour avoir une estimation de position du rover plus précise, l'odométrie visuelle peut également être utilisée. Comme cette approche est plus coûteuse (du point de vue de la puissance de calcul et de la capacité de mémoire utilisée), elle est utilisée uniquement pour la navigation autonome sur des terrains très difficiles (densité élevée de roches) ou à faible adhérence (sable).

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure1.png}
\caption{Exemple de carte de navigation}
\end{figure}

### 2.3 Moyens d’essais et validation

Le CNES possède deux sites d’essais (un terrain abrité et un terrain extérieur) en grandeur réelle qui sont utilisés pour tester individuellement les différents sous-systèmes d’un rover, ainsi que pour réaliser des essais complets de démonstrateurs de robotique mobile pour des missions d’exploration planétaire. L’ensemble du terrain a été conçu pour imiter de façon aussi réaliste que possible les caractéristiques du terrain martien. Il comprend des zones avec des densités variables (rocheux, sableux) pour répondre aux différentes contraintes exigées par les tests de locomotion, de perception ou de localisation. Il permet ainsi d’effectuer des tests sur des zones plates, des zones avec des pentes multiples, des grands obstacles, ou des configurations difficiles comme des canyons, des dunes ou des vallées.

Les algorithmes proposés dans cette thèse ont été validés en utilisant des données de navi-
gation obtenues sur le SEROM ou dans le simulateur du rover d’EDRES à l’aide des modèles numériques de terrain (MNT) martien et des modèles 3D des rovers d’exploration planétaire. En plus de celui du SEROM, deux autres MNTs sont utilisés: HiRISE, construit à partir des données réelles acquises par la caméra HiRISE à bord de la sonde Mars Reconnaissance Orbiter (MRO) et ASU, généré par Astrium UK en utilisant une distribution statistique des pentes et des roches sur la surface martienne. Deux modèles de rovers sont utilisés en simulation: celui de IARES, le châssis de démonstration pour l’exploration planétaire actuellement utilisé au CNES et celui du rover ExoMars. Les deux rovers sont équipés de six roues et leurs systèmes de locomotion leur permettent d’avoir une manœuvrabilité élevée. D’abord les rovers peuvent suivre une ligne droite et un chemin en forme d’arc en contrôlant la direction et la vitesse des roues avant et arrière. Ensuite, d’importants changements de cap peuvent être exécutés par rotation sur place, en orientant les roues avant et arrière de telle sorte que les roues gauches se trouvent dans l’orientation opposée aux roues droites. Les rotations sur place sont très coûteuses, que ce soit en temps d’exécution ou en termes d’usure mécanique du robot. C’est pour cela que les travaux de cette thèse proposent des techniques pour éviter autant que possible ces rotations sur place pendant l’exécution de la trajectoire d’un rover d’exploration planétaire. Finalement, les six roues peuvent être commandées dans une même orientation, ce qui permet au rover de réaliser un mouvement de translation en diagonale sur le côté sans affecter son cap (mouvement dit de crabe), pour des manœuvres qui exigent une précision élevée de positionnement près des objectifs scientifiques.

3 Approches de planification de chemin pour un rover d’exploration planétaire proposées par le CNES

Cette partie expose les deux principes adoptés par le CNES pour planifier le chemin d’un rover d’exploration planétaire. Ces deux approches représentent le point de départ des travaux présentés dans cette thèse et sont utilisées en tant que référence pour l’évaluation des performances des algorithmes proposés.

Pour des raisons de simplicité, l’environnement de navigation est décomposé sur une grille à résolution constante, appelée carte de navigation. La carte de navigation utilisée dans ces travaux utilise une représentation hybride d’espaces de navigation: les cellules qui représentent les zones inconnues, non-navigables ou non-planifiables ont des valeurs fixes, tandis que les cellules des zones navigables ou incertaines sont représentées par des plages de valeurs qui codent la difficulté de la traversée en fonction des capacités locomotrices du rover d’une part et des caractéristiques du terrain d’autre part.

Un rover d’exploration planétaire peut atteindre une cible lointaine en effectuant une succession de cycles de planification et d’exécution de chemins. En raison de la couverture limitée de la carte locale de navigation, le but de la mission se trouve rarement dans la région représentée par la carte de navigation. C’est pour cela qu’un sous-but doit être choisi dans la carte locale pour calculer un chemin à suivre qui permette au rover de se rapprocher de la cible mission. Une fois le sous-but sélectionné dans la carte de navigation, le planificateur de chemin local trouve la trajectoire optimale entre la position courante du rover et l’emplacement du sous-but (par rapport à la distance parcourue, la difficulté du terrain et au nombre de changements de cap )
Résulté étendu selon la procédure suivante:

- Lancer un algorithme de recherche de type best-first glouton pour examiner si le sous-but sélectionné est atteignable;

- Calculer un chemin de référence, suite à une étape de post-traitement de lissage du chemin fourni par algorithme de recherche de type best-first glouton. Ce chemin, constitué de segments de ligne droite entre la position courante du rover et le sous-but sélectionné, nommé primitive, est minorant en terme de distance à parcourir.

- Calculer le meilleur chemin à suivre par rapport à la difficulté des zones traversées et en limitant les rotations sur place. Pour limiter l’utilisation des ressources embarquées, une version optimisée de l’algorithme de recherche de type best-first (A∗) est utilisé, tout en guidant la recherche en utilisant la primitive du chemin.

La Figure 2 illustre les résultats intermédiaires obtenus pendant la procédure de planification d’un cycle de locomotion. La position courante du rover est au centre de la carte de navigation et le sous-but sélectionné se trouve en bas de la carte de navigation. La Figure 2a présente en blanc le chemin fourni par l’algorithme de recherche de type best-first glouton. L’avantage de cette étape est qu’un chemin non-optimal peut être trouvé en consommant un minimum des ressources embarquées. Ensuite, la Figure 2b représente le chemin primitif issu de la phase de post-traitement de lissage. Celui-ci sera utilisé pour guider la recherche de l’algorithme A∗ et pour éviter le développement des trajectoires possibles dans le cul-de-sac en bas à gauche de la carte de navigation. Cela permet un calcul de meilleur chemin qui réduise la taille d’arbre de recherche d’A∗, l’utilisation de la mémoire ainsi que le temps de calcul. La Figure 2c présente le chemin à suivre fourni par l’algorithme A∗. Les périmètres de planification et d’exécution sont également illustrés. Ces deux périmètres ont une interprétation différente des zones inconnues présentes dans la carte de navigation en fonction de leur distance au rover. Le périmètre exécution limite le déplacement du rover pendant un cycle de locomotion. Dans la Figure 2c seule la partie de la trajectoire illustrée en blanc va être exécutée. Finalement, le centre et le champ de vue des
3. Approches de planification de chemin pour un rover d’exploration planétaire proposées par le CNES

perceptions planifiées pour le cycle de locomotion suivant sont représentées. Les détails en ce qui concerne le fonctionnement des algorithmes de planification best-first glouton et $A^*$ sont présentés dans le Chapitre 3 de ce manuscrit.

La façon dont le sous-but est choisi génère deux approches principales proposées par le CNES pour la planification de chemin d’un robot d’exploration planétaire, qui sont détaillées par la suite.

3.1 Planification locale successive (SLA$^*$)

La première approche permet au rover d’atteindre un but mission en effectuant uniquement des procédures de planification locale successives. Dans ce cas, la position du sous-but d’un cycle de locomotion est choisie selon la procédure suivante:

- Il se situe à la limite de la carte de navigation, dans une région navigable connectée en permanence à la position courante du rover (de sorte qu’un chemin existe entre la position courante du rover et le sous-but).

- La distance entre le sous-but choisi et la ligne imaginaire reliant la position courante du rover et le but mission doit être minimisée.

En utilisant cette approche, la capacité du rover à naviguer autour de structures complexes d’obstacles est limitée par la taille de la région couverte par la carte de navigation locale.

3.2 Planification long terme (LRN)

Récemment, une nouvelle technique pour la planification globale de chemin de rovers d’exploration planétaire a été développée au CNES. L’objectif est d’établir un sous-but mieux informé pour la procédure de planification de chemin local. Cette approche consiste à construire une carte de navigation globale qui peut couvrir la totalité de la surface que le rover peut explorer pendant un sol. Pour des raisons de contraintes de mémoire, la carte globale, intitulée carte d’obstacles, ne contient que les contours d’obstacles rencontrés pendant la mission courante. La carte d’obstacle est utilisée ensuite pour calculer un chemin entre la position courante du rover et le but mission en construisant un graphe partiel des tangentes. Une fois le chemin trouvé, la position du sous-but dans la carte de navigation locale peut être déterminée. Le fonctionnement détaillé de cette approche est présenté en Section 3.6. Bien que cette méthode fournisse des chemins globaux en respectant la contrainte de faible utilisation des ressources de calcul embarquées, il a été démontré que dans 44% des cas, le chemin trouvé et donc l’emplacement du sous-but identifié pour la planification locale ne sont pas optimaux.

En décrivant les deux méthodes proposées par le CNES pour la planification de chemin d’un rover d’exploration planétaire, il est mis en évidence un besoin continu de développer de nouvelles techniques pour améliorer les performances des algorithmes de recherche de chemin local et global.
4 Planification incrémentale de chemin local


Un rover d’exploration planétaire atteint sa cible mission à travers des requêtes successives de planification de chemin local. Une requête de planification de chemin se compose de la position courante du rover, la carte de navigation locale du terrain environnant et la position du sous-but choisi pour une meilleure progression vers l’objectif de la mission. Pour chaque requête, le planificateur de chemin construit un arbre de recherche constitué des états du rover (positions dans l’espace de navigation) et des transitions entre les états voisins, et fournit le chemin à suivre par le rover pour le cycle de locomotion courant. Pour des raisons de sécurité, seule une partie du chemin fourni est exécuté. Pour cette raison, le sous-arbre de recherche correspondant à la partie de chemin non-utilisée peut être restauré et utilisé au cours de la prochaine requête de planification de chemin. Cela permet de réduire le temps de calcul pour la planification de chemin tout en évitant de recalculer les données de recherche restaurées.

FRA∗ est un algorithme de recherche incrémentale qui rend l’utilisation de propriétés géométriques spécifiques aux représentations de l’espace de navigation au travers d’une grille. Au lieu d’effectuer une nouvelle recherche de chemin de type A∗ à partir de zéro pour chaque cycle de locomotion, l’algorithme FRA∗ utilise l’arbre de recherche précédent pour initialiser l’arbre de recherche pour le cycle de locomotion courant. L’algorithme de planification de chemin A∗ construit deux structures: la liste OPEN et la liste CLOSED. La liste CLOSED contient tous les états de l’espace de navigation qui ont été analysés pendant la recherche courante et qui ont abouti à un chemin au coût minimum depuis la position de départ du rover. L’utilisation d’une grille fait que la liste CLOSED représente une zone contiguë sur la carte de navigation. La liste
OPEN contient tous les états du rover qui doivent être analysés lors des étapes suivantes. Dans ce cas, la liste OPEN contient tous les états possibles du rover qui se trouvent sur la frontière de la zone contiguë représentée par la liste CLOSED. Le principe de fonctionnement de FRA* et les étapes à suivre pour la restauration des listes OPEN et CLOSED sont détaillés dans la Section 4.3.

En ce qui concerne l’implémentation de cet algorithme dans l’environnement EDRES, deux éléments principaux sont abordés. Tout d’abord, une nouvelle structure de données est utilisée pour la gestion de la liste OPEN pour l’algorithme A*. Ainsi la file de priorité de type liste chaînée initialement utilisée est remplacée par une structure de type tas binaire. Ensuite, deux approches pour la restauration des sous-arbres de recherche sont étudiées, en tenant compte du fait que la carte de navigation est centrée à chaque cycle de locomotion sur la position courante du rover. Ainsi, un décalage correspondant au déplacement du rover entre deux cycles de locomotion consécutifs doit être appliqué aux sous-arbres de recherche restaurés. Une première méthode, intitulée Classic Subtree Restore (CSR), supprime d’abord tous les états dans les listes OPEN et CLOSED initiales qui n’appartiennent plus au sous-arbre enraciné à la position actuelle du rover et applique ensuite le déplacement correspondant aux états restaurés. La deuxième méthode, Optimised Subtree Restore (OSR), génère de nouvelles listes OPEN et CLOSED, en reproduisant les états du sous-arbre enraciné à la position actuelle du rover. Les données de localisation sur la carte de navigation des états restaurés sont mis à jour en prenant en compte le déplacement du rover entre les deux cycles de locomotion successifs.

Une première étude évalue la performance d’une étape de planification de chemin, du point de vue de la structure utilisée pour la gestion de la liste OPEN de l’algorithme A*. Il est d’abord montré en Section 4.5.1 que les performances en temps de calcul d’une recherche de type A* dépendent du nombre d’états ajoutés et développés dans la liste OPEN. Chaque fois qu’un élément est ajouté ou retiré dans la liste OPEN, des manipulations, nommées heap percolates, sont faites pour assurer la condition d’ordre de la liste. La Figure 3 illustre une comparaison entre les performances obtenues par l’algorithme A* en utilisant une file de priorité de type liste chaînée ou un tas binaire lors d’une campagne expérimentale contenant 3250 cycles de locomotion.

Les Figures 3a et 3b montrent les histogrammes concernant le nombre de heap percolates à chaque utilisation du A*. Il apparaît de façon évidente que l’utilisation d’une file de priorité de type tas binaire réduit de manière significative les manipulations des éléments dans la liste OPEN. La Figure 3c fournit une comparaison du temps de calcul nécessaire pour le A* en utilisant une liste de priorité de type liste chaînée ($temps_{LC}$) ou de type tas binaire ($temps_{TB}$) qui est directement influencé par le nombre de heap percolates. Finalement, le gain en termes de temps de calcul obtenu par l’utilisation d’une structure de type tas binaire est montré en Figure 3d.

Ensuite, l’amélioration des performances lors de l’utilisation de la planification de chemin de manière incrémentale est évaluée. Une étude statistique a été réalisée parallèlement sur trois MNTs et s’est intéressée à 198 scénarios de mission et à plus de 3300 cycles de locomotion. De plus amples détails sur cette campagne d’évaluation peuvent être trouvés en Section 4.5.2. Une synthèse des résultats obtenus par l’algorithme de planification incrémentale de chemin FRA* par comparaison aux planifications de chemin à partir de zéro obtenues en utilisant la méthode
Figure 3: Comparaison de performance de l’algorithme A* par rapport à la gestion de la liste de priorité
A* est fournie en Tableau 1. On peut conclure que la méthode FRA* représente un gain pour ce qui est du nombre d’états ajoutés ou supprimés de la liste OPEN et de heap percolates. Ce gain est minimum sur le MNT du SEROM à cause de la haute résolution de la carte de navigation (25 mm) qui entraîne une exécution d’une partie plus longue de la trajectoire planifiée et donc des sous-arbres de recherche plus petits à restaurer entre deux cycles de locomotion consécutifs. Concernant les deux autres MNTs (dont la résolution est de 50 mm pour la carte de navigation) des gains de respectivement 16.24% et 14.78% quant au nombre de heap percolates sont obtenus.

<table>
<thead>
<tr>
<th>Mesure (par cycle de locomotion)</th>
<th>SEROM</th>
<th>ASU</th>
<th>HiRISE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FRA</strong> gain</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># états ajoutés</td>
<td>4.65%</td>
<td>18.21%</td>
<td>16.06%</td>
</tr>
<tr>
<td># états développés</td>
<td>4.71%</td>
<td>12.40%</td>
<td>17.06%</td>
</tr>
<tr>
<td># heap percolates</td>
<td>3.8%</td>
<td>16.24%</td>
<td>14.78%</td>
</tr>
<tr>
<td><strong>FRA</strong> restauration de sous-arbre</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restaurations validées</td>
<td>21.15%</td>
<td>50.69%</td>
<td>70.29%</td>
</tr>
<tr>
<td># états restaurés in OPEN</td>
<td>19.22%(47.49%)</td>
<td>28.56%(64.72%)</td>
<td>31.23%(79.23%)</td>
</tr>
<tr>
<td># états restaurés in CLOSED</td>
<td>8.85%(36.41%)</td>
<td>25.86%(53.11%)</td>
<td>26.14%(66.27%)</td>
</tr>
<tr>
<td>CSR heap percolates</td>
<td>587</td>
<td>1003</td>
<td>689</td>
</tr>
<tr>
<td>OSR heap percolates</td>
<td>66</td>
<td>368</td>
<td>243</td>
</tr>
<tr>
<td>Gain heap percolates</td>
<td>89.37%</td>
<td>64.66%</td>
<td>67.62%</td>
</tr>
</tbody>
</table>

Table 1: Performance du l’algorithme de planification incrémentale de chemin FRA* par rapport à A*

Dans le cadre de l’utilisation du FRA*, des opérations de restauration des sous-arbres de recherche sont évaluées. Une restauration est validée quand, depuis l’exécution de la partie sélectionnée du chemin planifié, le rover se trouve dans un état qui fait partie du chemin optimal précédemment calculé. Cela assure une maximisation de la taille du sous-arbre de recherche qui peut être utilisé comme initialisation pour la recherche de chemin courante. Pour des raisons liées à la haute résolution de la carte de navigation comme mentionné précédemment, le plus faible taux de restauration validée est obtenu pour le MNT du SEROM. Avec une résolution inférieure, des taux de restaurations validées de 50% et 70% sont obtenus pour les MNTs ASU et HiRISE. L’utilisation de la planification incrémentale de chemin résulte en un taux moyen de restauration des états des listes OPEN et CLOSED entre 8.85% et 31%, avec un taux maximum atteint de 79.23%. Finalement la performance des deux méthodes de restauration de sous-arbre de recherche, CSR et OSR, est évaluée. Cette étude expérimentale démontre que l’utilisation de la méthode OSR fournit un gain maximum de 89.37% sur le nombre de heap percolates par rapport à la méthode CSR.

Ce chapitre a évalué deux directions principales d’amélioration de l’algorithme de planification de chemin local tel que développé par le CNES. Tout d’abord, l’utilisation d’une file de priorité de type tas binaire pour la gestion de la liste OPEN de l’algorithme A* se traduit par une réduction moyenne de la charge de calcul de 60% par rapport au cas de l’utilisation d’une liste chaînée. Cela génère une réduction importante du temps de calcul, qui peut avoir un impact élevé sur l’évolution de la mission d’exploration planétaire du rover. Ensuite, on a démontré qu’il est possible de partager une quantité importante de données de navigation entre deux cycles de locomotion consécutifs en utilisant l’algorithme FRA* qui conduit aussi à une amélioration du temps de calcul de l’algorithme de planification de chemin. Cependant, la per-
 distancing de l’algorithme de planification incrémentale de chemin reste directement influencée par la résolution de la carte de navigation et par les erreurs de locomotion du rover.

5 Planification locale de chemins non holonomes

Les bases de la planification de chemins non holonomes ont été fixées par [Dubins, 1957] et [Reeds and Shepp, 1990]. Afin de générer des chemins qui peuvent être suivis par un véhicule non holonome, trois contraintes doivent être remplies en plus de la contrainte cinématique classique: la courbure de la trajectoire doit être continue et bornée (en respectant la contrainte du rayon de braquage minimum), et la dérivée de la courbure doit être elle-même bornée (le véhicule non holonome peut réorienter ses roues à une vitesse limitée) [Scheuer and Laugier, 1998] [Fraichard and Ahuactzin, 2001] [Fraichard and Scheuer, 2004].

La capacité à planifier de manière autonome des chemins non holonomes en évitant les obstacles rencontrés est d’un grand intérêt pour les rovers d’exploration planétaire, car elle a une influence directe sur la gestion des ressources et le retour scientifique de la mission. Le travail présenté dans [Pivtoraiko et al., 2008] propose deux approches: l’une fait référence à la génération de plans de mouvements réalisables par optimisation du modèle dynamique inverse utilisé ensuite pour construire un réseau de chemins finement discrétisés pour être utilisés par des algorithmes standards de recherche ; l’autre permet de générer des chemins en se basant sur des polynômes de courbure cubique qui tiennent compte de la géométrie du rover, des possibilités de marche arrière, de la résolution de la grille et des coûts en terme de puissance de calcul [Guixé et al., 2012]. Toutefois, jusqu’à maintenant, une seule méthode a été validée et est largement utilisée pour générer des chemins non holonomes pour rovers d’exploration planétaire [Goldberg et al., 2002]. Elle utilise un ensemble prédéfini de 23 arcs de cercle vers l’avant et 23 arcs de cercle vers l’arrière avec un rayon de courbure variable et deux manoeuvres de rotation sur place qui est évalué à chaque étape de la planification de chemin pour décider de la meilleure trajectoire à suivre.

5.1 Synthèse d’architecture de planification de chemin embarquée sur MERs

Le logiciel de navigation autonome embarqué initialement à bord des Mars Exploration Rovers de Curiosity de la NASA utilise une architecture intitulé GESTALT (Grid-based Estimation of Surface Traversability Applied to Local Terrain) [Maimone et al., 2004]. Les caractéristiques du terrain environnant sont évaluées à partir des images stéréo et une carte de navigation nommée goodness map est construite. Ensuite, l’ensemble des chemins pré-calculés est projeté sur la carte de navigation et un score de traversabilité est calculé pour chaque trajectoire candidate. La Figure 4 illustre un ensemble de chemins pré-calculé constitué de 23 arcs de cercle vers l’avant et vers l’arrière, de plusieurs lignes droites et la projection des arcs de cercle vers l’avant sur la carte de navigation. Indépendamment de la nature du terrain, le même ensemble de chemins pré-calculés reçoit un score d’atteignabilité par rapport à la position actuelle du rover et l’emplacement du but désiré. Ainsi, le chemin avec le meilleur score, issu de la fusion des scores de traversabilité et d’atteignabilité, est sélectionné pour être suivi par le rover.
5. Planification locale de chemins non holonomes

(a) Ensemble de 96 chemins pré-calculés constitué des arcs de cercle et lignes droites [Biesiadecki et al., 2005]  
(b) Projection des arcs de cercle vers l’avant sur la carte de navigation [Biesiadecki and Maimone, 2006]

Figure 4: Évaluation des chemins pré-calculés dans l’architecture GESTALT

5.2 Planification de chemins non holonomes pour le rover ExoMars

Un des axes principaux de recherche proposé dans cette thèse aborde le problème de la planification de chemins non holonomes dans le cadre d’un rover d’exploration planétaire. Il est évident que dans le cas de l’architecture de navigation GESTALT l’ensemble des chemins pré-calculés restreint les possibilités de navigation du rover. De plus, les valeurs de cap atteintes après l’exécution du chemin sont elles-mêmes limitées. En outre, le rover ne peut pas effectuer de trajectoires qui contiendraient des points d’inflexion tout en évitant les obstacles proches. Le but de ces travaux est donc de développer un ensemble de chemins pré-calculés qui puisse être réutilisé par le rover pendant ses étapes successives de planification de chemin, afin que les trajectoires avec une complexité plus élevée (autres que des arcs de cercle ou des lignes droites) puissent être effectuées. Bien que l’état de l’art propose de nombreuses approches probabilistes, ce travail explore l’utilisation d’un treillis régulier sur l’espace d’états produit par des mécanismes d’échantillonnage déterministes. Un des avantages de cette approche est l’invariance par translation qui permet d’utiliser le même ensemble de chemins pré-calculés pour connecter différentes paires d’états dans l’espace de navigation. La connectivité de l’espace de recherche est codée en considérant un environnement totalement navigable et sans obstacles et l’ensemble de chemins pré-calculés est stocké de manière compacte pour pouvoir être inclus dans le logiciel de vol d’un rover d’exploration planétaire.

Afin de réduire les besoins de l’étape de planification de chemin en termes d’utilisation mémoire et temps de calcul, cette thèse propose la conception d’un treillis d’états réguliers réduit qui implémente une version discrétisée du planificateur proposé par Barraquand et Latombe.
Résumé étendu

[Barraquand and Latombe, 1991]. Une fois calculé, l’ensemble de chemins pré-calculés est utilisé dans la procédure de planification de chemin pour construire l’arbre de recherche à moindre coût tout en incluant les contraintes cinématiques du rover.

La génération d’un treillis d’états est paramétrée comme suit:

- **Rayon de courbure minimum** Il représente une contrainte cinématique du rover (angle de braquage maximal des roues pour suivre une trajectoire curviligne)

- **Nombre de chemins** Il s’agit du nombre de chemins que le rover peut être amené à suivre en partant d’un état donné. C’est un nombre impair, permettant d’avoir un chemin en ligne droite et un nombre égal de chemins de type arc de cercle de part et d’autre de cette ligne.

- **Discrétisation en translation** La résolution de la grille représentant l’espace de navigation.

- **Discrétisation en orientation** La résolution de l’orientation du rover.

- **Longueur du chemin** Distance en mètres parcourue par le rover en suivant le chemin donné.

Une procédure en deux étapes est utilisée pour générer le treillis d’états réguliers:

1. Génération d’un ensemble de contrôles de mouvement continus.
2. Génération de chemins non holonomes entre des paires d’états discrets dans l’environnement de navigation.

Connaissant l’orientation et la position initiale du rover, la première étape consiste à calculer la trajectoire à suivre avec une longueur et un rayon de courbure donnés. Le but principal de cette phase est de déterminer l’état discret correspondant au point d’arrivée de cette trajectoire. Ensuite, la deuxième étape calcule le chemin non holonome reliant l’état discret de départ du rover à l’état discret d’arrivée de la trajectoire continue. Ce chemin est composé d’une ligne droite partant de l’état discret du rover suivie d’un arc de cercle qui atteint l’état discret d’arrivée choisi. La procédure entière pour la génération d’un tel ensemble de chemins pré-calculés est détaillée en Section 5.3.

La Figure 5 montre un exemple d’un treillis d’états calculé en utilisant la procédure proposée. Ici, pour des raisons de visibilité la longueur de chemin a été fixée à 800 mm, sachant que la résolution de la grille est de 50 mm. Une discrétisation en orientation à 5° près a été utilisée, résultant en 72 orientations possibles du rover pour une position donnée. Ici, la position de départ du rover est située au centre de la grille. Pour chaque orientation du rover, au maximum 7 chemins sont calculés en prenant en compte le rayon minimum de braquage du rover (fixé à 1 m).

L’utilisation de la mémoire embarquée est une contrainte spécifique pour les missions d’exploration planétaire. Ce problème peut être résolu en utilisant les propriétés de symétrie de l’ensemble de chemins pré-calculés. Ainsi, seul $1/8$ ($+1$) des orientations possibles du rover est calculés et mémorisés dans le logiciel du vol du rover. Le reste des chemins du treillis d’états sont
5. Planification locale de chemins non holonomes

restaurés à la demande en appliquant la transformation correspondante pendant l’étape de planification de chemin. En outre, les besoins en mémoire de stockage sont limités en représentant les couloirs discrets de chaque chemin à l’aide d’un codage de type Freeman. En conséquence, l’exigence en mémoire pour le treillis d’états illustré en Figure 1.5 est considérablement réduite de 31.5KB à 0.4KB.

L’ensemble des chemins pré-calculés est ensuite répliqué plusieurs fois par un planificateur de chemin optimal de type A* pour générer des chemins non holonomes. Chaque état discret analysé par le planificateur de chemin représente la position (correspondant au centre d’une cellule dans la représentation grille) et la valeur discrétisée de l’orientation du rover. Le fait d’augmenter la dimension de l’espace de recherche peut rendre cette approche non-optimale.
Résumé étendu

en termes de temps de calcul. L’objectif de ces travaux est de prouver l’applicabilité de cette approche au contexte particulier de l’exploration planétaire et d’en identifier ses limites vis à vis des contraintes d’une mission d’exploration planétaire. Les détails concernant les modifications apportées à l’algorithme A* (initiallement utilisé sur un espace de recherche bidimensionnel) sont présentés en Section 5.4.

Contrairement à l’approche utilisée par GESTALT qui contient des contrôles de type rotation sur place dans l’ensemble des chemins pré-calculés, la méthode présentée dans cette thèse prend en compte la possibilité d’effectuer au maximum une rotation sur place planifiée pendant un cycle de locomotion. Ainsi, si le rover est mal orienté par rapport à la cible qu’il doit atteindre, une correction du cap du rover est planifiée au tout début de l’exécution de la trajectoire. Cela évite d’inclure des contrôles de type rotation sur place dans l’ensemble des chemins pré-calculés, ce qui entraînerait une augmentation inutile de la taille de l’arbre de recherche construit par l’algorithme A*.

5.3 Évaluation

Plusieurs études ont été réalisées pour évaluer les performances du planificateur de chemins non holonomes en utilisant un ensemble de chemins pré-calculés. Les paramètres utilisés pour générer le treillis d’états sont synthétisés en Table 2.

| Discretisation en translation | résolution de la grille 50mm |
| Discretisation en orientation | 360 valeurs réparties de façon homogène (1° près) |
| Longueur du chemin | 400mm |
| Nombre de chemins | 5 chemins pour chaque orientation du rover |
| Rayon minimum de braquage | 1m |

Table 2: Paramètres utilisés pour générer l’ensemble de chemins pré-calculés

Un premier objectif a été de déterminer la meilleure fonction heuristique pour l’algorithme de recherche de type A*. Suite à une étude statistique contenant 1900 requêtes de planification de chemin, il a été conclu que la distance euclidienne est la meilleure estimation heuristique. Les autres choix abordés incluaient des cartes de distance construites en utilisant des estimations de distance comme Manhattan, octile et une version hybride entre Manhattan et octile.

Des études statistiques ont ensuite été effectuées pour comparer les performances de planificateur de chemins non holonomes par rapport au planificateur de chemin sur grille initialement utilisé par le CNES. La synthèse des performances obtenues est fournie en Section 5.5.2. Plusieurs versions du planificateur de chemins non holonomes ont été évaluées en faisant varier le nombre de chemins pré-calculés pour chaque orientation du rover (entre 3 et 9) et la valeur de discrétisation d’orientation. Les résultats montrent que le planificateur de chemins non holonomes avec un maximum de 5 chemins sortant de chaque état du rover, s’avère plus performant que celui initialement utilisé en ce qui concerne le temps de calcul et la difficulté de la trajectoire proposée. Par contre, les chemins calculés sont en moyenne 6% plus longs et le taux de réussite du planificateur de chemins non holonomes baisse en moyenne de 5% par rapport à celui du planificateur sur la grille. De plus, l’approche qui planifie une rotation sur place au début de
l’exécution de la trajectoire est validée. Les résultats montrés en Section 5.5.3 prouvent que cette approche augmente le taux de réussite du planificateur de chemins non holonomes et réduit la distance parcourue par le rover pour atteindre la cible sélectionnée. La distance parcourue est un facteur très important, surtout pour un rover d’exploration planétaire. En réduisant la longueur des trajectoires du rover, on maximise le temps que le rover peut dépenser pour effectuer des mesures scientifiques. L’utilisation de la mémoire embarquée est aussi une contrainte spécifique à une mission d’exploration planétaire. La Figure 6 fournit une représentation graphique de la quantité de mémoire utilisée par rapport à la longueur totale de la trajectoire exécutée (qui est proportionnelle à la taille de l’arbre de recherche par chaque planificateur de chemin). Ainsi, il est montré que le planificateur de chemins non holonomes apporte une réduction considérable de l’utilisation de la mémoire grâce à son arbre de recherche épars.

Figure 6: Comparaison de l’utilisation de mémoire en utilisant: (a) le planificateur de chemin sur grille du CNES, ou planificateur de chemins non holonomes avec 5 (b) ou 7 (c) chemins sortant de chaque état.

La performance du planificateur de chemins non holonomes a aussi été évaluée en utilisant des données de navigations acquises à bord du rover IARES sur le site d’essais du CNES (SEROM). Les résultats obtenus lors de cette évaluation sont présentés dans la Section 5.5.4. La même amélioration de performance par rapport au planificateur de chemin sur grille est démontrée. De plus, grâce à la résolution plus élevée de la carte de navigation par rapport à l’étude précédente, des performances identiques ont pu être obtenues pour le planificateur de chemins non holonomes avec 7 chemins pré-calculés pour chaque orientation du rover.

Finalement, une étude statistique a été réalisée pour évaluer la capacité du rover à atteindre des cibles lointaines grâce aux étapes successives de planification locale des chemins non holonomes. La synthèse des résultats obtenus fournie en Section 5.5 démontre l’amélioration
Résultats des performances par rapport au planificateur de chemin sur grille. Le résultat le plus important obtenu lors de cette étude est la réduction significative (en moyenne 40%) des rotations sur place pendant l’exécution de la trajectoire. Cette réussite permet une meilleure gestion de temps d’exécution de la trajectoire en réduisant également l’usure du système de locomotion du rover et la consommation énergétique. Ainsi, l’utilisation de chemins non holonomes augmente le taux de missions accomplies sans rotations sur place de 28% par rapport au planificateur de chemin sur grille.

Un exemple de scénario de mission est illustré en Figure 7. Le rover est censé atteindre le côté droit d’un rocher en évitant tous les obstacles rencontrés dans son environnement de navigation. La trajectoire d’une longueur de 16,56m exécutée en utilisant de manière successive le planificateur initialement utilisé par le CNES est montrée sur la Figure 7a. La trajectoire générée par le planificateur de chemins non holonomes est illustrée sur la Figure 7b. Il est à noter que pendant l’exécution de la trajectoire calculée en utilisant le planificateur sur grille, le rover a effectué 3 rotations sur place. En utilisant la méthode proposée par ces travaux, la cible est atteinte en évitant toute manœuvre de rotation sur place. Concernant l’utilisation de la mémoire embarquée lors de la simulation de mission, le planificateur de chemin sur grille a utilisé 11,85MB contre 1,66MB requis par le planificateur de chemins non holonomes.

Figure 7: Exemple de trajectoire exécutée pendant un scénario mission en utilisant: (a) le planificateur de chemin sur grille (b) planificateur de chemins non holonomes avec un taux de ramification de 5

Ce chapitre a introduit un nouveau planificateur de chemins non holonomes qui utilise un ensemble de chemins pré-calculés en tenant compte des capacités de locomotion du rover. Les performances d’un tel planificateur de chemin ont été évaluées en prenant en compte les contraintes spécifiques à une mission d’exploration planétaire. Des études expérimentales ont montré que l’approche proposée par ces travaux apporte de nombreuses améliorations par rapport à la méthode initialement proposée par le CNES. En synthèse, l’emploi des ressources embarquées est réduit, en ce qui concerne la charge de calcul et l’utilisation de mémoire. De plus, les trajectoires générées sont exécutées plus facilement en réduisant leur difficulté et le nombre de rotations sur place. Cela permet notamment au rover d’atteindre ses cibles plus rapidement et de disposer ainsi de plus de temps pour effectuer des mesures scientifiques.
6 Planification incrémentale de chemin sur des cartes de navigation globales à résolution constante

La deuxième partie de la thèse aborde le problème de planification de chemin global pour permettre au rover d’atteindre des cibles lointaines (au-delà des dimensions de sa carte locale) tout en évitant des champs d’obstacles denses ou des configurations de type cul-de-sac. Dans un premier temps, la contrainte concernant l’utilisation de la mémoire embarquée est détendue, et il est considéré qu’il y a suffisamment de mémoire disponible pour stocker une carte globale à haute résolution qui couvre toute la région que le rover peut explorer pendant un sol. Ces premiers résultats sont ensuite utilisés pour évaluer les performances des algorithmes de planification incrémentale de chemin global. Le chapitre 6 fournit une description détaillée des algorithmes capables d’effectuer une planification de chemin optimal et largement utilisés pour la navigation des robots dans des environnements dynamiques inconnus ou partiellement connus [Stentz and Hebert, 1995] [Singh et al., 2000] [Kelly et al., 2006]. Ainsi, plusieurs algorithmes qui reposent sur le fonctionnement de l’algorithme A* sont évalués: D* [Stentz, 1994], Focused D* [Stentz, 1995] et D* lite [Koenig and Likhachev, 2002] [Ferguson and Stentz, 2005a].

Il existe d’autres variantes des algorithmes de type D* qui peuvent être utilisées pour la planification de chemin d’un robot mobile. Certains abordent le problème de la planification de trajectoire optimale sous contraintes globales [Stentz, 2002], tandis que d’autres utilisent une approche dite de tout moment [Likhachev et al., 2005a]. Les algorithmes de planification de chemin de type tout-moment construisent généralement très rapidement une trajectoire réalisable, mais pas optimale. La qualité de la trajectoire est ensuite améliorée pendant le reste du temps disponible pour la planification de trajectoire. Cependant, cette approche n’est pas recommandée pour la planification de chemin pour un rover d’exploration planétaire dans la mesure où une solution légèrement sous-optimale peut avoir un impact négatif sur la sécurité du rover et donc de la mission d’exploration elle-même.

Un planificateur de chemin global a déjà été embarqué sur les rovers de la NASA : l’algorithme Field D* [Ferguson and Stentz, 2005a] [Carsten et al., 2009]. Il s’agit d’un planificateur de chemin qui utilise des techniques d’interpolation pour générer des chemins indépendamment de la résolution de la grille représentant l’espace de navigation. L’inconvénient est que les solutions de chemin fournies par l’algorithme Field D* sont susceptibles de générer des changements de cap du rover inutiles [Daniel et al., 2010]. De plus, l’utilisation de l’algorithme Field D* prend presque deux fois plus de temps que l’algorithme D* lite. Le temps de calcul étant une mesure de performance importante, cette étude n’inclut pas l’algorithme Field D*.

6.1 Évaluation de performance en Matlab

Une étude préliminaire a été menée pour évaluer les avantages apportés par l’utilisation de chacune des trois versions d’algorithmes de planification incrémentale sélectionnés par rapport à une planification successive de type A* [Hernández et al., 2012].

La stratégie de navigation est la suivante :

- le rover commence toujours au centre de la carte de navigation vide (l’espace environnant est inconnu et est initialement considéré navigable) et calcule le plus court chemin à suivre
pour atteindre une cible sélectionnée au hasard sur la frontière de la carte de navigation.

- Il commence à exécuter ce chemin en prenant de nouvelles perceptions dans un rayon donné autour de sa position courante, chaque fois qu’un nouvel état dans l’espace de navigation est atteint.

- Chaque fois que le rover observe un obstacle, la carte de navigation est mise à jour et le chemin le plus court vers sa cible est recalculé. La procédure s’arrête lorsque le rover atteint la position cible ou s’il se rend compte que la cible n’est pas atteignable.

L’étude a consisté à réaliser des simulations sur des cartes de navigation de deux tailles différentes ($100 \times 100$ et $500 \times 500$) avec des obstacles situés au hasard sur celles-ci, avec une densité d’obstacle de respectivement 30%, 50% et 70%. Deux valeurs sont prises en compte pour le rayon de perception du rover: 5 et 10 cellules. Lors des simulations sur la carte de navigation ayant $100 \times 100$ cellules, comme [Hernández et al., 2012] l’indique, la planification successive de chemin de type $A^*$ est généralement moins coûteuse, en termes d’états analysés, que les algorithmes de planification incrémentale. D’autre part, lors des simulations sur une carte de navigation plus grande ($500 \times 500$), l’utilisation de la planification successive de type $A^*$ devient sous-optimale en raison de l’augmentation de manière exponentielle de l’arbre de recherche. C’est le cas typiquement d’une carte de navigation globale qui couvre une zone plus vaste. Par conséquent, dans un tel scénario, parmi tous les algorithmes considérés, c’est l’algorithme $D^*$ lite qui obtient les meilleures performances, à la fois en termes de temps de calcul et de nombre d’états analysés.

6.2 Évaluation des performances de la planification globale de type $D^*$ lite pour un rover d’exploration planétaire

Afin de mettre l’accent sur la nécessité de la planification de chemin global, la Figure 8 illustre une comparaison entre les trajectoires exécutées lors de l’utilisation successive de la planification locale de type $A^*$ et l’utilisation du $D^*$ Lite sur une carte de navigation globale. Dans cet exemple, le planificateur de chemin $D^*$ lite utilise une carte de navigation globale initiale à basse résolution pour pouvoir guider d’une manière mieux informée le rover vers la cible lointaine. La carte de navigation globale initiale contient des données de navigation d’une résolution de 1m et chaque perception haute résolution (2.5 cm) est utilisée pour la mettre à jour localement. Les trajectoires exécutées sont indiquées sur la carte de navigation haute résolution du site d’essais du SEROM, où les niveaux de gris plus clairs représentent des zones dangereuses et les niveaux de gris plus foncés représentent les régions navigables. Ce qu’il est important de retenir, c’est qu’en utilisant un planificateur global de chemin il est possible de planifier à l’avance d’éviter des obstacles (même en l’absence de données de navigation très précises) et de contourner des obstacles avec des configurations complexes. À l’inverse, le SLA* permet de contourner les zones dangereuses uniquement lorsque le rover s’est suffisamment rapproché, ce qui peut générer des manoeuvres de rotation sur place, très coûteuses en énergie et en temps d’exécution. En outre, en raison de la couverture limitée de la carte de navigation du SLA*, la cible située en dehors de la configuration de type cul-de-sac ne peut pas être atteinte.
6. Planification incrémentale de chemin sur des cartes de navigation globales à résolution constante

La performance du D* lite est évaluée par rapport à l’approche de planification de chemin global (LRN) développé par le CNES. Sur une série de 171 scénarios mission impliquant des cibles choisies au hasard sur le MNT du SEROM, le D* lite obtient un meilleur taux de réussite, en exécutant des trajectoires sur un terrain de moindre difficulté. Par contre, les trajectoires générées par l’algorithme D* sont plus longues, à cause de l’effet de la discrétisation de la représentation de type grille.

La performance du D* lite est aussi évaluée par rapport à la méthode LRN en prenant en compte trois types d’initialisation de la carte de navigation globale. Les trois cartes de navigation globales initiales sont calculées comme suit:

- L’environnement de navigation est totalement inconnu et un scénario optimiste initialise la carte de navigation globale entière avec la valeur de navigabilité minimale disponible dans EDRES.
- L’environnement de navigation est totalement inconnu et un scénario pessimiste initialise la carte de navigation globale entière avec la valeur de navigabilité maximale disponible dans EDRES.
- L’environnement de navigation est partiellement connu et la carte de navigation globale initiale est construite à partir des données de navigation à basse résolution provenant d’un orbiteur.

En ce qui concerne le temps de calcul de l’algorithme D* lite, il est montré que c’est toujours la première étape de planification de chemin qui est la plus coûteuse. Ce n’est pas un problème pour
une mission d’exploration planétaire puisque l’arbre de recherche initial peut être téléchargé par le rover en même temps que la carte de navigation globale initiale. Des trois versions proposées pour le D* lite, celle qui utilise le scénario pessimiste en termes de navigabilité obtient le temps de calcul le plus court par itération. Cela est dû au fait que, pendant la première étape de planification, un arbre de recherche plus ample est construit, ce qui réduit par conséquence le volume de travail lors des itérations successives suivantes. Quoiqu’il en soit, les temps de calcul du D* sont généralement plus élevés que ceux obtenus avec l’approche LRN du CNES. Une analyse de la consommation de mémoire est également effectuée concernant les cas pessimiste et optimiste du D*. Il est conclu que même si le cas optimiste est en moyenne 5 fois moins coûteux que le cas pessimiste, l’utilisation de mémoire n’est pas compatible avec les contraintes spécifiques d’une mission d’exploration planétaire. Ceci va finalement éliminer le choix d’utiliser l’algorithme de planification incrémentale de chemin de type D* lite avec une carte de navigation globale à haute résolution pour guider le rover ExoMars.

7 Planification globale de chemin à l’aide d’une carte de navigation multi-échelles

Les logiciels de navigation autonome pour les rovers d’exploration planétaire développés par la NASA et le CNES utilisent une représentation de type grille. L’inconvénient d’une telle représentation à haute résolution est qu’elle limite la zone couverte par la carte de navigation en raison des capacités de mémoire limitées. Une solution à ce problème est mise en œuvre dans cette étude ; la cartographie multi-échelles. La zone couverte par la carte de navigation est agrandie sans pour autant augmenter la consommation de mémoire en introduisant la possibilité de représenter la proximité du rover à haute résolution et les régions éloignées à faible résolution. Cela permet d’utiliser une carte de navigation capable de couvrir toute la zone que le rover peut explorer pendant un sol sans augmenter les ressources de mémoire déjà utilisées par la carte de navigation locale utilisée par le CNES dans l’approche SLA*. L’utilisation d’une représentation multi-échelles de l’environnement de navigation a été largement étudiée lors de développements récents dans le domaine de la planification de chemins pour des robots qui doivent atteindre des cibles lointaines. Une méthode largement utilisée dans ce contexte emploie les quad-trees [Noborio et al., 1990] [Kambhampati and Davis, 1986], ou framed quad-trees [Yahja et al., 1998], en effectuant une décomposition dyadique récursive de l’espace de navigation. Un inconvénient des quad-trees est qu’un chemin généré sur une telle représentation peut comporter des nombreux changements de cap, très coûteux en terme d’énergie et de temps de calcul. La représentation de type framed quad-tree remédie à ce désavantage en ajoutant des cellules à haute résolution autour de toutes les cellules de faible résolution, ce qui exige une utilisation accrue de la mémoire et de la charge de calcul. D’autres approches concernant la décomposition multi-échelle utilisent des cellules triangulaires [Hwang et al., 2003] ou une représentation hiérarchique de sphères encapsulant le rover pour éviter les collisions [Verwer, 1990]. D’autres approches encore incluent l’utilisation des ondelettes pour la représentation d’environnements de navigation statiques [Pai and Reissell, 1995] [Tsiontros and Bakolas, 2007] [Cowlagi and Tsiontros, 2010] ou des cartes de navigation multi-échelles en coordonnées polaires utilisés pour des véhicules aériens [Yu et al., 2009].
A ce jour, seule une approche a traité le problème de la planification globale de chemin en abordant le contexte spécifique d’un rover d’exploration planétaire avec des ressources de calcul embarquées limitées [Carsten et al., 2007]. Cette approche utilise deux cartes de navigation: la carte de navigation à haute résolution (20 cm par cellule) de GESTALT et une carte de navigation globale à basse résolution (40 cm par cellule). La carte de navigation globale est utilisée par le planificateur de chemin global par interpolation Field D** qui fournit un sous-but mieux informé pour le planificateur de chemin local de GESTALT. Cette approche a été intégrée et testée en mission prolongée sur le rover Opportunity [Carsten et al., 2009] lors de cinq séries expérimentales de navigation autonome.

7.1 Construction de la carte de navigation multi-échelle

La carte de navigation utilisée par le planificateur de chemin doit représenter la plus grande surface possible autour du rover, afin d’assurer la génération de chemins optimaux pour atteindre une cible lointaine en traversant des champs avec une densité élevée d’obstacles. Cependant, la taille de la carte de navigation utilisée dans l’environnement EDRES est limitée en raison des contraintes de mémoire disponible à bord d’un rover d’exploration planétaire. Cette section présente une nouvelle approche pour la représentation de grandes zones avec une carte de navigation de taille limitée. Cela permet au rover de se déplacer sur une longue distance tout en évitant les obstacles et les configurations de terrain dangereuses comme des culs-de-sac, qui ne peuvent pas être contournés en utilisant seulement un planificateur de chemin local.

La construction de la carte de navigation locale détaillée en Section 2.2.2 utilise une représentation de l’environnement dans l’espace de configuration, c’est-à-dire qu’une cellule est réputée navigable si et seulement si on peut poser le centre du modèle du rover dans toutes les orientations sans violer les contraintes d’inclinaison (roulis et tanchage). En conséquence, la détection d’une discontinuité entraîne l’apparition d’un périmètre de sécurité autour de cette zone (de rayon égal au rayon du rover), ce qui se traduit en bordure de carte par l’apparition d’une bande extérieure de cellules non classifiées en raison de l’échec du positionnement du modèle du rover en périphérie. La largeur de cette bande dépend du rayon du modèle du rover et de la résolution de la carte de navigation.

L’approche proposée dans cette thèse transforme la carte de navigation locale utilisée par l’algorithme SLA* en une carte de navigation multi-échelles de sorte que la zone couverte s’en trouve augmentée. La carte de navigation est divisée en deux zones distinctes: une région interne (à haute résolution), à proximité du rover, où les tests de navigabilité décrits dans la section 2.2.2 peuvent être appliqués, et une bande externe (à basse résolution) utilisée pour stocker des informations sur les obstacles éloignés du rover. Un score de traversabilité est calculé pour chaque cellule de la bande extérieure à basse résolution, en fonction de la densité d’obstacles présents dans la région représentée. La carte de navigation multi-échelles contient des cellules dont la couverture dépend de la position dans la représentation proposée, en utilisant deux paramètres \textit{valHRES} et \textit{valLRES} qui représentent respectivement les valeurs nominales de haute et basse résolution. Les détails concernant la couverture de chaque cellule de la carte de navigation multi-échelles et la procédure utilisée pour le calcul des scores de traversabilité sont fournis en section 7.2. Un avantage de cette représentation est que la 8-connexion entre les deux types de régions est conservée, ce qui la rend facilement utilisable par les planificateurs.
de chemin déjà développés. La figure 9 illustre un exemple de carte multi-échelles (à droite) calculée à partir d’une carte de navigation globale (à gauche) acquise lors d’un scénario de simulation de mission. Les paramètres utilisés pour le calcul de la carte de navigation multi-échelles sont les suivants: \( val_{HRES} = 50\text{mm} \) et \( val_{LRES} = 2000\text{mm} \). Le carré brun épais dans la carte de navigation globale à haute résolution représente la zone correspondant à la région à haute résolution dans la représentation multi-échelles. Les lignes fines rouges sur la carte de navigation globale indiquent les limites des régions couvertes par les cellules à basse résolution de la représentation multi-échelles. Pour des raisons de visibilité, la bande extérieure de la représentation multi-échelles est multipliée par dix.

Figure 9: Exemple d’une carte de navigation multi-échelles (à droite) calculée à partir d’une carte de navigation globale à haute résolution (à gauche)

En utilisant cette approche, la couverture de la carte de navigation multi-échelles est augmentée sans aucun impact sur l’utilisation de la mémoire embarquée. Selon le modèle du rover utilisé, la couverture de la carte de navigation multi-échelles peut atteindre 250m × 250m (EX-OMARS rover, \( val_{HRES} = 25\text{mm} \), \( val_{LRES} = 2000\text{mm} \)), soit un gain considérable par rapport à la région initialement représentée par la carte de navigation locale (17.5m × 17.5m).

En ce qui concerne la mise à jour de la carte de navigation multi-échelles, deux approches sont proposées. La première, nommée Successive Shifts (SS), identifie d’abord la région à haute résolution qui va être déplacée dans la bande extérieure à basse résolution suite au déplacement du rover (entre deux perceptions consécutives), utilise un algorithme de segmentation pour calculer les boîtes englobantes pour tous les obstacles, et les utilise ensuite pour calculer le taux d’occupation d’obstacles de chaque cellule à basse résolution affectée. Le reste de la bande extérieure est mise à jour par un déplacement correspondant des valeurs déjà existantes. La seconde méthode, intitulée Obstacle Map (OM), utilise une carte d’obstacles globale actualisée à chaque perception du rover. De la même manière, le score de traversabilité de chaque cellule dans la bande extérieure est calculé en utilisant les boîtes englobantes des obstacles stockés dans la carte d’obstacles globale.

La section 7.3 fournit une comparaison des performances obtenues pour chaque méthode.
7. Planification globale de chemin à l’aide d’une carte de navigation multi-échelles

d’actualisation de la carte de navigation multi-échelles lors d’un déplacement latéral du rover. Il est alors observé que lors de l’utilisation de la méthode SS, la bande externe à basse résolution contient les obstacles rencontrés pendant la traversée, mais qu’ils ne sont pas aussi importants qu’ils devraient être. A cause des opérations de déplacement, cette méthode tend en effet à représenter des obstacles sur des surfaces plus larges que leur taille initiale, ce qui conduit à une perte d’information concernant leur localisation exacte. Cette effet de dilution n’apparaît en revanche pas pour la méthode OM qui conserve l’information de localisation. Par conséquent, c’est la deuxième méthode, OM, qui sera utilisée lors des études expérimentales. L’utilisation de la carte d’obstacles pour la mise à jour de la carte de navigation multi-échelles rend en effet possible la conservation des données de localisation des obstacles, même loin du rover.

7.2 Planification globale de chemin

La carte de navigation multi-échelles est ensuite utilisée par les algorithmes de planification de chemin déjà développés pour guider le rover vers les cibles lointaines à travers des champs d’obstacles denses. La table 3 fournit une synthèse des techniques de planification de chemin développées dans EDRES et leurs versions lorsqu’ils sont utilisés sur la carte de navigation multi-échelles. Deux premières approches utilisent l’algorithme de planification de chemin de type A* directement sur la carte de navigation multi-échelles pour trouver la trajectoire à suivre entre la position courante du rover et la position de la cible mission dans la représentation multi-échelles. Dans ce cas, le planificateur de chemin A* vérifie simultanément si la cible mission est atteignable et calcule la trajectoire globale optimale correspondante. Selon la direction de la recherche, les deux planificateurs de chemin sont étiquetés MRFA* (multi-resolution forward A*) ou MRRA* (multi-resolution reversed A*). A l’image de l’architecture de planification de chemin proposée par le CNES (qui propose le pré-calcult d’un chemin non-optimal pour guider l’algorithme de recherche A* dans un second temps), deux approches de planification de chemin sur la carte de navigation multi-échelles sont proposées. Tout d’abord, une recherche Dijkstra est effectuée sur la carte de navigation multi-échelles pour vérifier l’accessibilité de la cible mission et calculer un chemin initial sous-optimal vers un sous-but sélectionné dans la région à haute résolution. Puis, le planificateur de chemin de type A* est utilisé pour calculer le chemin optimal vers le sous-but sélectionné. Deux stratégies sont aussi proposées, selon la direction de recherche de type Dijkstra sur la carte de navigation multi-échelles: FGLA* (forward guided local A*) et RGLA* (reverse guided local A*). Dans la section 7.4, il est démontré que la performance de l’algorithme de planification de chemin peut être influencée par sa direction de recherche. Cela est dû au fait que lorsque des obstacles concaves sont rencontrés, un nombre plus important d’états doit être analysé, ce qui conduit à une surcharge de calcul indésirable pour la requête de recherche de chemins.

Lors d’une étude statistique, près de 20% des cibles mission choisies de manière aléatoire et situées à une distance (euclidienne) entre 20m et 50m de rover n’ont pas été atteintes en utilisant le planificateur de chemin local proposé par le CNES, SLA*. En conséquence, les scénarios de missions exécutés ont été divisés en deux classes: des requêtes de recherche dites faciles, et des requêtes de recherche dites complexes. Les performances des stratégies de planification de chemin sur la carte de navigation multi-échelles sont analysées par rapport à celles obtenues avec l’algorithme SLA* dans les cas faciles et par rapport à celles obtenues avec l’algorithme LRN
Table 3: Résumé des stratégies de planification de chemin

<table>
<thead>
<tr>
<th>Stratégie</th>
<th>Sélection de sous-but</th>
<th>Chemin optimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRFA*</td>
<td>Non</td>
<td>A* Direct Multi-échelle</td>
</tr>
<tr>
<td>MRRA*</td>
<td>Non</td>
<td>A* Inverse Multi-échelle</td>
</tr>
<tr>
<td>FGLA*</td>
<td>Dijkstra Direct Multi-échelle</td>
<td>A* Direct</td>
</tr>
<tr>
<td>RGLA*</td>
<td>Dijkstra Inverse Multi-échelle</td>
<td>A* Direct</td>
</tr>
<tr>
<td>SLA*</td>
<td>Dijkstra Direct</td>
<td>A* Direct</td>
</tr>
<tr>
<td>LRN</td>
<td>Graphe Partiel de Tangentes</td>
<td>A* Direct</td>
</tr>
</tbody>
</table>

pour les cas complexes. En ce qui concerne l’étude statistique dans son intégralité, les approches multi-échelles ont obtenu un meilleur taux de réussite que SLA* et génèrent des trajectoires avec une difficulté du terrain traversé améliorée, mais sont légèrement plus longues. Lors de cette étude, il est surtout démontré que l’utilisation de la carte de navigation multi-échelles augmente significativement la capacité des stratégies de planification de chemin déjà développées au CNES en permettant au rover d’atteindre des cibles mission lointaines tout en étant éventuellement amené à traverser des terrains difficiles avec des configurations d’obstacles complexes. En ce qui concerne les contraintes d’utilisation des ressources embarquées, il est démontré en Section 7.4 que seules les stratégies FGLA* et RGLA* peuvent être retenues et utilisées pour résoudre le problème de planification de chemin global pour un rover d’exploration planétaire. L’utilisation du planificateur de chemin de type A* pour planifier un chemin optimal jusqu’à la cible mission se traduit en effet par une augmentation de l’utilisation de la mémoire embarquée en moyenne deux à trois fois plus importante, ce qui rend les deux stratégies MRFA* et MRRA* inapplicables dans ce contexte contraint qu’est celui de l’exploration planétaire.

7.3 Sélection automatique de direction de recherche en prenant en compte la distribution de terrain environnant

L’approche proposée par ces travaux suggère une méthode de caractérisation de la forme d’obstacles rencontrés par le rover afin de décider de la direction de recherche du planificateur de chemin. Le but est d’optimiser la charge de calcul, tout en essayant de réduire autant que possible l’influence de la forme des obstacles sur la performance du planificateur de chemin. La procédure utilisée pour décider de la direction de recherche est présentée en Section 7.5. D’abord, tous les obstacles placés entre la position courante du rover et l’emplacement de la cible mission sont sélectionnés et leur mesure de compacité est calculée. Les obstacles avec une mesure de compacité élevée (proche de 1) ne comportent pas de régions concaves importantes et n’ont pas un impact majeur sur la charge de calcul de l’algorithme de recherche de chemin et ne seront pas pris en compte dans cette procédure. Pour tous les autres obstacles, un score de concavité est calculé, représentant la taille et la répartition des zones concaves sur le contour de chaque obstacle. Le score de concavité cumulé pour tous les obstacles pris en compte va enfin décider de la meilleure direction de recherche du planificateur de chemin. Le score de concavité d’un obstacle donné peut être calculé en utilisant une des cinq démarches proposées, dont une synthèse est fournie en Figure 10:

1. Déplacement du centre de gravité (CGD): calcul d’un vecteur qui caractérise le
7. Planification globale de chemin à l’aide d’une carte de navigation multi-échelles

Figure 10: Synthèse des approches pour le calcul du score de concavité d’un obstacle

déplacement entre le centre de gravité du contour d’obstacle et le centre de gravité de son enveloppe convexe.

2. Distribution des régions concaves (CRD): calcul d’un vecteur résultant des vecteurs définis par chaque région concave sur le contour d’obstacle. Le vecteur correspondant à une région concave à l’origine dans le centre de gravité du contour, la direction vers le centre de gravité de la région concave et la magnitude égale à la surface de la région concave.

3. Comparaison des régions concaves opposées (OCR): Le score de concavité est issu de la différence de surface entre les deux régions concaves présentes sur le contour d’obstacle et qui se trouvent à l’intersection avec la ligne imaginaire entre le point de départ et le point d’arrivée de la recherche courante.

4. Distance la plus courte à contourner l’obstacle (SDC): Cette méthode trouve d’abord l’extrémité de l’enveloppe convexe du contour la plus proche de la ligne imaginaire entre le point de départ et le point d’arrivée intersectant l’obstacle. Ensuite la surface cumulée de toutes les régions concaves qui se trouvent entre cette extrémité et la ligne imaginaire est utilisée pour calculer le score de concavité.

5. Distance la plus courte à contourner l’obstacle étendue (ESDC): Cette approche est similaire à la méthode précédente, mais on calcule la surface cumulée des régions concaves de chaque côté de la ligne imaginaire qui se trouvent dans l’intervalle de distance défini par l’extrémité du contour sélectionné.

Une étude statistique a été menée pour évaluer la précision de la prédiction de la direction de recherche optimale des méthodes proposées. Il est montré que les deux premières approches (CGD et CRD) obtiennent les meilleures performances, avec une précision de prédiction...
d’environ 70% dans le cas où la différence de charge de calcul entre les deux directions de recherche est supérieure à 50%. Pour le même cas, les trois dernières approches (OCR, SDC et ESDC) obtiennent une précision de prédiction supérieure à 60%.

Le but final de cette thèse est atteint en proposant une stratégie de planification de chemin global totalement innovante. La carte de navigation multi-échelles est utilisée pour obtenir des capacités de planification globale. Ensuite, l’algorithme Dijkstra est utilisé pour vérifier l’atteignabilité de la cible mission et pour fournir une position de sous-but dans la région à haute résolution et un chemin initial non-optimal. La trajectoire finale à exécuter est alors calculée en utilisant le planificateur de chemin non holonome proposé. Cette stratégie est testée en simulation et ses performances sont comparées à celles des méthodes déjà développées par le CNES (SLA* et LRN). Trois scénarios de missions d’exploration planétaire sont détaillés comme suit: atteindre une cible au bout d’une formation de type canyon, sortir d’une configuration de type cul-de-sac et traverser un champ dense d’obstacle vers une cible lointaine. La stratégie de planification de chemin global a réussi à guider le rover vers ses cibles mission, avec des meilleurs coûts en termes de charge de calcul, de difficulté du terrain, et de temps d’exécution et avec moins de manœuvres de rotation sur place que les méthodes actuelles du CNES. De plus, l’utilisation de la technique de détection de façon automatique de la direction de recherche de type OCR s’est révélée avoir le meilleur taux d’amélioration de charge de calcul avec un maximum de 54%.

8 Bilan et perspectives

Les travaux décrits dans cette thèse se concentrent sur des problèmes de planification autonome de chemin et de représentation de l’espace de navigation avec les contraintes spécifiques d’une mission d’exploration planétaire telle que la future mission de l’ESA, ExoMars.

Deux axes principaux de recherche sont analysés et développés dans cette thèse pour résoudre le problème de la planification de chemin par un rover d’exploration planétaire. La première direction est basée sur l’exécution des étapes successives locales de planification de chemin en utilisant uniquement une carte de navigation locale. Cette approche s’est avérée appropriée pour des scénarios de missions d’exploration planétaire en raison de sa faible utilisation des ressources embarquées (capacité de mémoire et charge de calcul). Cependant, cela ne peut être utilisé avec succès que lorsque le rover navigue dans des environnements à faible densité d’obstacles. La seconde direction de recherche dans cette thèse concerne le problème de planification de chemin global lorsque le rover doit éviter de grandes zones dangereuses ou des champs avec une densité élevée d’obstacles pour pouvoir atteindre la cible choisie.

La première contribution de cette thèse consiste en l’amélioration des performances de l’architecture de planification de chemin local par étapes successives développée au CNES, testée et validée pour être utilisée pour le rover d’ExoMars. La gestion de la file de priorité de l’algorithme de planification de type A* est améliorée en utilisant la structure de données de type tas binaire. Ce faisant, le nombre de heap percolates nécessaire pour l’insertion d’un état dans la liste OPEN est réduit, entraînant une réduction du temps de calcul pour le processus entier de planification de chemin. Ensuite, l’utilisation d’algorithmes de planification incrémentale de chemin est proposé afin de réutiliser les données de navigation entre les étapes de planifica-
tion de chemin successives. Dans la phase de planification de chemin local, le rover calcule un chemin optimal vers un sous-objectif sélectionné dans la carte de navigation locale à une distance comprise entre 4 et 6 mètres. Pour des raisons de sécurité, seulement une partie de ce chemin est exécutée (maximum 2.4 mètres) au cours d’un cycle de locomotion. En conséquence, une partie importante de la trajectoire préalablement calculée n’est pas exécutée et le sous-arbre de recherche correspondant peut être restauré et réutilisé pendant l’étape de planification de chemin suivante. L’approche proposée utilise l’algorithme Fringe Retrieving A* pour effectuer la planification de chemin courante en utilisant des sous-arbres de recherche restaurés. Il est prouvé que l’utilisation de FRA* fournit un gain important en termes de temps de calcul. Deux méthodes sont suggérées pour restaurer des sous-arbres de recherche précédemment calculés et leurs performances sont comparées. Les résultats des études statistiques montrent qu’en utilisant la stratégie de planification incrémentale de chemin proposée, un maximum de 80% des données de navigation peut être restauré entre les étapes de planification de chemin consécutives, ce qui se traduit par une amélioration importante de la gestion des ressources embarquées du rover.

La deuxième contribution traite le problème de la planification de chemins locaux non holonomes en prenant en compte les capacités de locomotion du rover. La planification de chemin non holonne a déjà été utilisée pendant des missions d’exploration planétaire à bord du rovers MER et Curiosity, mais l’approche utilisée limite fortement les capacités de déplacement du rover dans l’environnement de navigation. Ces travaux ont proposé et évalué l’applicabilité d’un planificateur de chemins non holonomes à l’aide d’un ensemble de chemins pré-calculés tout en respectant les contraintes concernant l’utilisation des ressources informatiques réduites embarquées. Un générateur de treillis d’états régulier est détaillé, qui est ensuite reproduit en ligne par un planificateur de chemin de type A*. En tenant compte du fait que la longueur de la trajectoire exécutée pendant un cycle de locomotion est limitée à 2.4m et que la longueur des chemins pré-calculés est de 0.4 m, on peut en déduire que les trajectoires générées avec le planificateur proposé ont une complexité élevée (maximum 5 points d’inflexion). Des études statistiques ont montré que le planificateur de chemins non holonomes est plus performant que le planificateur de chemin sur une grille initialement utilisé par le CNES. Grâce à la taille réduite de l’arbre de recherche construit par le planificateur de chemins non holonomes, l’approche proposée réduit considérablement l’utilisation de la mémoire embarquée et le temps de calcul lorsque de faibles taux de ramification sont utilisés. L’achèvement le plus important est que tout au long des scénarios de missions simulés, le planificateur de chemins non holonomes a réduit d’environ 40% le nombre de manoeuvres de rotation sur place pendant l’exécution de la trajectoire. Ces manoeuvres sont à éviter autant que possible dans la mesure où elles accencent l’usure du système de locomotion du rover et augmentent le temps de parcours et l’énergie consommée. De plus, elles ont un impact négatif sur le retour scientifique de la mission, puisqu’elles limitent la distance que le rover peut conduire pendant un sol.

La deuxième partie de cette thèse est consacrée au problème de planification de chemin global afin de permettre au rover d’éviter des configurations étendues d’obstacles pendant sa traversée vers la cible de la mission. La troisième contribution de cette thèse consiste en une analyse des algorithmes de planification incrémentale de chemin dans un environnement de navigation inconnu ou partiellement connu. L’algorithme D* lite est sélectionné pour être testé sur des simulations de missions robotiques d’exploration planétaire et sa performance est évaluée par
Résumé étendu

rapport à la planification de chemin global basée sur l'algorithme de type tangent graph partiel (LRN) actuellement en cours de développement au CNES. Il est conclu que l'algorithme D∗ Lite ne respecte pas les contraintes spécifiques à la mission d'exploration, puisqu'il se trouve être coûteux à la fois en termes de puissance de calcul et de mémoire utilisée.

La quatrième contribution consiste en le développement d’une nouvelle représentation multi-échelles de la carte de navigation. Cela présente plusieurs avantages car cela permet au rover de stocker des données de navigation concernant toute la région qu’il peut explorer, sans pour autant générer de surcoût en termes d’utilisation de mémoire embarquée. En utilisant cette représentation, on peut de plus réaliser une planification de chemin global sans modifier l’architecture de navigation autonome d’EDRES. Enfin, cette représentation ne nécessite pas l’utilisation d’un sous-but.

Enfin, le cinquième et dernier apport de cette thèse est de proposer un planificateur de chemin global qui emploie une représentation multi-échelles de la carte de navigation et le planificateur local de chemins non holonomes pour calculer les trajectoires à exécuter avec un faible coût en termes de consommation de ressources mais avec une efficacité énergétique accrue. L’augmentation de performances en termes de temps de calcul et d’utilisation de la mémoire est obtenue en utilisant une méthode capable de prédire la meilleure direction de recherche du planificateur de chemin afin de minimiser l’effet de la forme et de la distribution du champ d’obstacles. Trois scénarios de missions sont définis pour prouver la faisabilité de l’utilisation du planificateur de chemin global proposé par rapport aux architectures de planification de chemin actuellement disponibles au CNES. Ainsi, il est démontré que l’approche proposée réduit l’utilisation des ressources embarquées, en générant des chemins plus faciles à naviguer et en réduisant les manœuvres de rotation sur place pendant l’exécution de la trajectoire. En outre, l’utilisation du procédé de prédiction de la direction de recherche optimale de chemin par rapport à la distribution du terrain environnant peut conduire à une réduction de la charge de calcul d’un maximum 54%.

Les travaux futurs pourront être concentrés sur les deux axes principaux de recherche qui ont été explorés dans cette thèse. Une première amélioration envisageable concerne la planification de chemins non holonomes. L’ensemble de chemins pré-calculés utilisé dans cette thèse se compose de chemins qui couvrent de manière homogène les capacités de braquage de rover. Tout au long des essais expérimentaux, il a été observé que le treillis d’états avait une densité plus élevée de chemins autour de la ligne droite et une densité plus faible autour des capacités de braquage maximales du rover. Cela pourrait influencer le taux de réussite du planificateur de chemins non holonomes. Il est suggéré que les travaux futurs se concentrent sur le développement des nouveaux treillis d’états qui ont une distribution homogène des chemins en termes de positions atteignables cette fois-ci, en offrant une accessibilité plus élevée dans l’espace de navigation. De plus, le générateur de treillis d’états introduit dans le Chapitre 5 ne tient pas compte des informations concernant la position, l’orientation et les capacités de braquage du rover. Il est suggéré d’introduire dans la conception du treillis d’états des contraintes différentielles à la connexion entre les chemins pré-calculés pour assurer la faisabilité de la trajectoire calculée.

Un autre point de vue consiste à améliorer les capacités de planification de chemin global de l’architecture de navigation autonome d’EDRES. Jusqu’en Février 2013, la camera stéréo haute résolution embarquée à bord de l’orbiteur Mars Express de l’ESA a cartographié plus de 90% de
la surface de la planète à une résolution de 10m par pixel, avec une précision qui peut atteindre 2m par pixel pour des zones sélectionnées. Toutes ces informations pourraient être utilisées dans le processus de planification de trajectoire du système de navigation autonome. D’autres recherches peuvent également étudier la possibilité de fusionner les données de navigation de résolutions différentes (venant de différents capteurs) afin de mettre à jour la carte de navigation globale multi-échelles. En outre, un planificateur de chemin global probabiliste qui prendrait en compte la précision des données de navigation fusionnées pourrait être conçu pour calculer des trajectoires globales optimales.

Enfin, le planificateur de chemin global proposé par ces travaux devra être évalué à bord des rovers utilisés par CNES pour valider les techniques de navigation autonome développées. Une étude quantitative devra notamment être effectuée en utilisant le modèle du rover ExoMars, afin d’évaluer l’usure réelle du système de locomotion lors de la navigation sur un terrain martien, sa consommation énergétique et l’influence de la longueur et de la difficulté de la trajectoire, ou des manœuvres de rotation sur place sur le retour scientifique de la mission d’exploration planétaire.