State lattice generation and nonholonomic path planning for a planetary exploration rover

Alexandru Rusu, Sabine Moreno, Yoko Watanabe, Mathieu Rognant, Michel Devy

IAC-14,A3,3C.2x25723

State lattice generation and nonholonomic path planning for a planetary exploration rover

Alexandru Rusu
and Sabine Moreno
System Validation Department
Centre National d'Etudes Spatiales,
Toulouse, France

Yoko Watanabe
and Mathieu Rognant
Department of Systems Control
and Flight Dynamics
ONERA, Toulouse, France

Michel Devy
LAAS-CNRS,
Toulouse, France

Abstract

ESA's Exomars mission will deploy on Mars a 300kg class rover , which is able to perform subsurface drilling, automatic sample collection and distribution to the onboard instruments for in-situ scientific measurements. Due to limited communication to Earth, the use of autonomous navigation in driving the rover towards a scientific target will increase the mission return by minimizing human intervention. The autonomous navigation on-board software should be designed with respect to the mission-specific constraints like energy consumption, memory and computation power, and time costs. This paper proposes an effective method of nonholonomic path planning using a pre-computed state lattice to generate trajectories which meets vehicle locomotion constraints. The objective is to allow the rover to autonomously reach the selected mission target while minimizing in-place-turn manoeuvres which have high impact on the total travelled distance and execution time during a mission. The proposed approach was implemented in the CNES EDRES environment and tested on real data acquired during navigation campaigns on the SEROM Mars-like test site and on simulated data using a digital elevation model constructed from HiRISE stereo images acquired onboard the MRO.

I. Introduction

During the last decades, we have experienced an important increase in scientific interests in exploring planet Mars. The main purposes of the exploration are to decrypt its geological history and to gather detailed information on whether its past environmental conditions were favourable for the development of microbial life. While new technologies are currently under development towards human planetary exploration, mobile robotic systems are deployed to undertake local scientific measurements. Up to now, four robots, Sojourner (1997), Spirit (2004), Opportunity (2004) and Curiosity (2012) have reached the Martian surface, among which only the last two are still operating.

ESA's ExoMars rover (Figure 1) is planned to be launched in 2018 in order to demonstrate the European technological capabilities to perform autonomous planetary robotic exploration. The core of its autonomous navigation software is based on algorithms developed at CNES in the last 20 years. The maturity of these navigation algorithms has been proved through intense field validation tests, including two remote experiments of ESA/ESTEC on the CNES SEROM test site [1] [2].

During Mars exploration missions, human operators cannot send control commands to the rover in real time due to communication time delay between Mars and Earth. Figure 2 illustrates a mission scenario defined in frame of the Exomars mission. The rover receives the command sequence for the current mission at the beginning of each sol either through

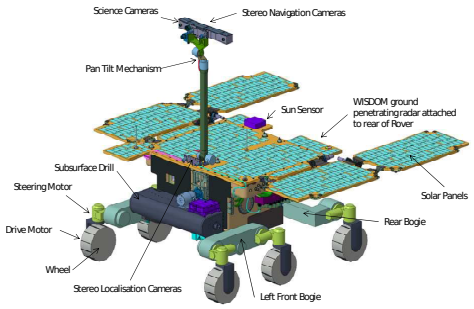


Fig. 1: The ExoMars Rover

direct communication with Earth or using communication relay satellites on the Martian orbit. The command sequence includes the locomotion objective(s) (mission goal coordinates, heading and distance to be travelled) [3] and the scientific experiments to be performed. Further on, the rover operates using only data acquired locally. It is expected that the robotic vehicle would drive up to 70 meters per sol in order to reach pre-selected scientific targets and to deploy its instruments. At the end of each sol, the rover transmits to the Operation Center information acquired during the sol, including surrounding landscape images. The data is analysed by human operators in order to decide the control sequence for the following sol. As shown in Figure 2, locomotion cycles are repeated until one of the stopping conditions is met (the target is reached, no safe trajectory is found by the path planning module or progression is stopped due to a higher priority task).

A locomotion cycle is performed by three subsystems: the perception chain, the navigation algorithms and the trajectory execution and localization controller .

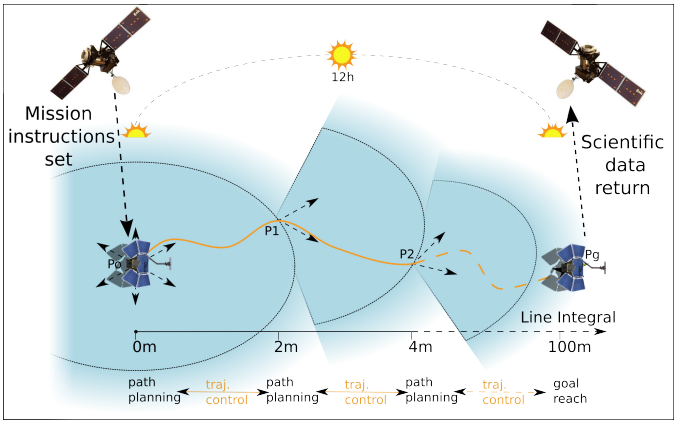


Fig. 2: Robotic planetary exploration scenario

Perception data is provided by on-board stereo cameras mounted on the mast of the rover1. It is used to estimate the features of the surrounding terrain (obstacles, slopes, terrain roughness) and to iteratively build a local navigation map which encodes the navigation difficulty with respect to the kinematic model of the rover. Following, the navigation algorithms use this navigation map to plan a safe trajectory to be driven towards the selected target position. Finally, the locomotion sub-system ensures the accurate execution of the planned path. It makes use of onboard wheel odometers and the Attitude and Heading Reference System (AHRS) to actively identify and correct anomalies during trajectory execution.

The locomotion subsystem of the ExoMars rover is formed of 6 wheels disposed on a 3-boogie system. All wheels can drive and steer simultaneously providing to the rover a very high manoeuvrability comparing to NASA rovers (MER and MSL). Figure 3 displays the main steering movements the locomotion sub-system is able to perform. The rover can follow a straight line or a curved trajectory of constant radius with its centre along the middle wheel axis, also called Ackerman configuration (Figure 3a). The rover can also perform in-place turn manoeuvres, when the wheels are set in the so called ”Envelope” configuration (Figure 3b). Finally, the wheels of the rover can be aligned at the same angle and move along that direction in the ”Crab” configuration((Figure 3c). The latter configuration is not considered in the path planning stage.

The path execution module is crucial for the safety of the rover during exploration missions. The autonomous navigation software developed by CNES includes a closed-loop-on-the-move function which corrects the trajectory of the rover when it gets away from the desired path. It defines three types of corridors around the given trajectory (Figure 4) and the control policy is decided depending on the position of the rover. First, the interior corridor represents an area where the locomotion error is considered to be minimal. While in the

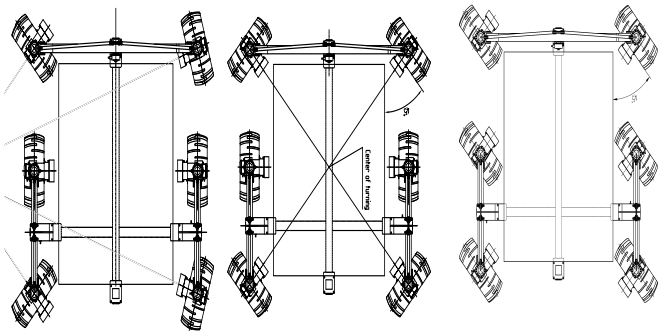


Fig. 3: Steering modes of the ExoMars locomotion system: (a) Ackerman configuration, (b) Envelope configuration, (c) Crab configuration

interior corridor the rover performs path execution at nominal velocity with locked heading (no steering commands). As the locomotion error increases, the rover can reach the intermediate corridor. Here the driving speed decreases in function of the locomotion error. Also, the rover can perform arc turns using the Ackerman configuration in order to correct the current heading with respect to the commanded path. Finally, when the rover reaches the outer corridor the same control policy as in the intermediated corridor is used. However, when the computed corrective command might violate the steering limits, the rover stops, performs an in-place-turn manoeuvre to adjust its heading so that it comes back on the reference trajectory.

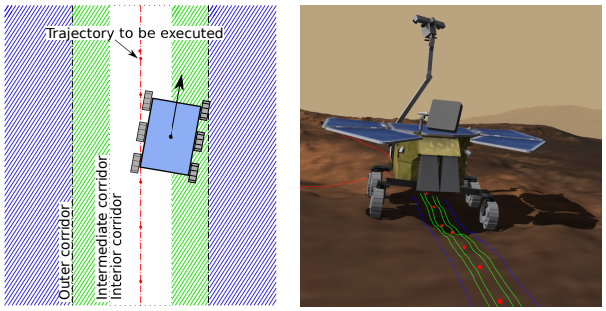


Fig. 4: Representation of the locomotion control method

II. Motivation

Path planning is an important stage in the autonomous navigation software of a planetary exploration rover. The current approach proposed by CNES uses a grid-based path planner which takes into account the roughness of the surrounding terrain encoded in the navigation map. Generated paths can comprise sharp heading changes which result in in-place-turn manoeuvres the rover has to perform during trajectory execution. First, such manoeuvres are expensive in terms of execution time and accuracy as the rover has to stop, position its wheels in ”envelope” configuration and perform the heading correction before the trajectory execution

is resumed. Second, the locomotion system wear is increased throughout the mission. Finally, the mission scientific return can be affected due to the fact that the distance covered by the rover in a given time is consequently reduced.

Our approach in this paper addresses the use of nonholonomic path planning to generate smooth paths which are easy to follow by the rover while minimizing in-place-turn manoeuvres. We propose a method to create a search space for constrained path planning, which encodes only feasible paths with respect to the locomotion capabilities of the rover. Such an approach was already used on the MERs, but the proposed search space limits the reachability of the rover and the complexity of the generated paths. Our method uses a precomputed motion control set which encodes a local connectivity of constrained motions. The motion control set can be used and replicated by an optimal A^*-like path planner to generate complex nonholonomic paths. We evaluate the efficiency of the proposed method compared with the existent grid-based path planner developed at CNES. The algorithm performance is evaluated with respect to the robotic planetary exploration mission-specific requirements, such as computation time, memory use or path length.

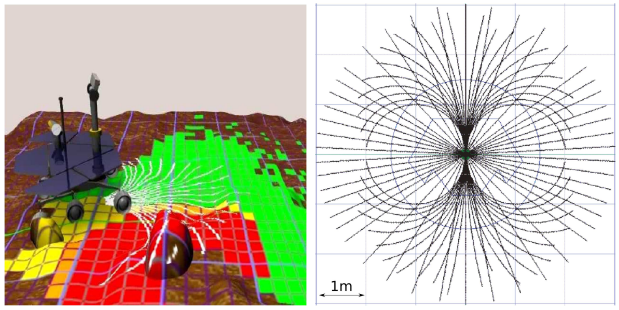
III. State of the art

Recently, a significant interest has been dedicated to the problem of trajectory generation including model motion constraints. The basis for path planning for differentially constrained vehicles was set by [4] and [5]. Their ideas were developed by generating smooth trajectories using segments of clothoids (curves having the curvature as a linear function of their length), arcs or straight lines [6] [7] [8].The focus is to generate paths which can be followed by a car-like vehicle. Thus besides the classical kinematic constraints, three constraints are added: the curvature of the path must remain continuous and it must 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) [9] [10]. On the other hand, different approaches suggest that the problem of smooth obstacle-free path generation can be solved analytically [11] [12]. Fast numerical optimisation techniques are used for trajectory generation for nonholonomic vehicles navigating in rough terrain [13] [14]. This approach can take into account the roughness of the navigation environment, vehicle dynamics and wheel-terrain interaction models while minimising an objective function with respect to risk, time or energy consumption.

The autonomous ability to plan obstacle-free drivable trajectories 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 [15] proposes two approaches: one refers to the generation of feasible motion plans by inverse dynamic (kinematic velocity) model optimisation and it is further used to build a finely discretized state lattice (which captures the full manoeuvrability of the rover) to be used by standard search algorithms. Another way to generate trajectories based on cubic curvature polynomials which takes into account the rover geometry as well as its driving back capabilities, grid resolution and computational cost is provided in [16].

Despite all the works listed above, up to date only one method has been validated and extensively used for generating constrained motion plans for planetary exploration rovers [17]. It uses a pre-defined set of 23 forward and 23 backward circular arcs of varying radius and two point turns to evaluate at each planning step the best candidate trajectory to be followed.

The autonomous navigation on-board the Mars Exploration Rovers (MERs) uses a navigation technique entitled Grid-Based Estimation of Surface Traversability Applied to Local Terrain (GESTALT) [18]. Similar to the autonomous navigation architecture proposed by CNES, the rover builds a local traversability or *goodness* map of the surrounding terrain from 3D range data acquired using on-board stereo vision. Following, if the *Path selection* sub-module is activated, the rover can autonomously perform candidate path evaluation to decide on the trajectory to be followed. It uses a fixed pre-computed set of candidate steering trajectories as given in Figure 5b. The rover is heading up in this view and each grid line indicates 1m spacing. Each path is projected on the goodness map and a corresponding traversability score is calculated as the weighted sum of the goodness values in all affected cells. The projection of a sub-set of the precomputed paths is represented in Figure 5a, where red cells represent unsafe areas (around the rock), yellow and orange cells indicate traversable but more difficult to navigate areas and green cells illustrate flat areas. The higher the accumulated score for a given path candidate, the easier to drive that trajectory.



(a) Projection of pre-computed paths over the goodness map [19]

(b) Set of the 96 pre-computed paths consisting in arc and point turns along with straight line drives [20]

Fig. 5: Illustration of Terrain Assessment and Path Selection modules of the GESTALT Navigation Planning

Independently of 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 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* [17].

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 scores is computed. The trajectory with the highest merged score is considered the best path candidate and is selected for execution. However, if its merged score is below a threshold, new perceptions are requested in the opposite (away from the goal) direction and the candidate paths in this area are evaluated. If in this case the merged score of the best candidate is still bellow a threshold, no path is chosen for execution and the drive stops.

The disadvantage of this approach is that 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 and no path with inflection points can be planned to avoid close obstacles.

IV. State lattice design

The aim of this section is to develop a method to calculate a precomputed set of paths, also called state lattice, which encodes the motion capabilities of the robotic vehicle in an obstacle-free environment. The method proposed in this paper takes into account the differential constraints of the rover at the moment where the vehicle-specific search space is created. In order to overcome the disadvantage of the state lattice used in GESTALT, this paper designs the state lattice so that the path planner can generate trajectories with higher complexity than simple arcs or straight lines. For a given state, the state lattice encodes all outgoing motion controls and the neighbour states which can be reached in the navigation map. The use of the translational invariance and regularity property allows to represent the entire navigation space of the rover as the union of the precomputed state lattice localized at any reachable position.

To calculate the compact set of paths, one needs the following parameters:

- **Regular lattice** : Represents locations the rover can reach in its environment. The rectangular grid representation is consistent with the navigation map representation which encodes the features of the surrounding terrain. The CNES autonomous navigation architecture uses a 351×351 navigation map.
- **Translational discretization** : The resolution of the grid representing the navigation environment, and the minimum displacement between two grid cells. For a translational discretization of $50mm$, the navigation map covers a region of $17.5m \times 17.5m$ around the rover.
- **Branching factor** : It is given by an odd number, which represents the number of possible motion controls at a given state. In our design, the rover can choose among a straight line of a certain length and a set of arcs of circle with different radius values covering homogeneously the steering capabilities of the robotic system.
- **Heading discretization** : The resolution of the heading component of the state space. It provides the number of possible heading values the rover can have for a given position in the grid representation. For example, when using a 1° discretization, the rover can have 360 possible heading directions for a given position.
- **Primitive length** : The distance the rover would drive on the given motion primitive.
- **Minimum turning radius** : It represents a locomotion constraint of the rover, referring to the smallest radius of circular path that the rover is able follow.

Figure 6 displays the generated motion primitives given an initial state (position and heading). For visibility reasons, the length of the motion primitives is set here to $800mm$ for a translational discretization value of $50mm$. The control set has a branching factor of 7 (1 straight line and 6 arcs), and the minimum turning radius is $1m$. However, these motion primitives cannot be included in the state lattice as their points of arrival do not fall exactly on the centre of the cells composing the regular lattice. The main property of the state lattice is the regularity of the control set, that is to say that the edges in the state lattice connect two discrete states in the search space. For our approach the motion primitives are used to identify the arriving state for each motion control. Then we perform inverse calculus to determine a motion control with the desired initial and final discrete states. Figure6 shows in red the motion primitives used to identify the arriving states and in black the generated motion controls for the given discrete states.

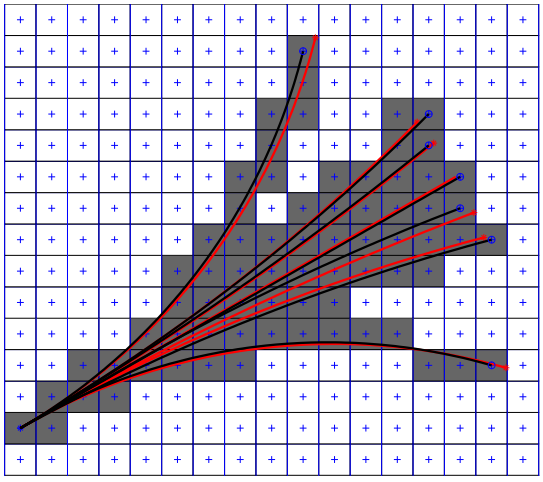


Fig. 6: Illustration of arc of circle paths (motion primitives) and the corresponding state lattice connecting discretized states (motion controls)

The procedure to generate the control set is summarized in Algorithm 1. The process starts at the lattice origin, usually the centre of the grid representation. Then, all possible states which are placed at the lattice origin with discretized heading values are analysed [Line 3-11]. For each state (x, y, θ), a set with outgoing paths is determined. The cardinality of

this set is given by the $BranchingFactor$ parameter. First, the continuous final position of the outgoing straight path is calculated and 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 control between the origin and the discrete state[Lines 14-16]. Given the start and goal discrete states, a motion control is composed of a straight line of length l followed by an arc of circle of radius R. Finally we evaluate the feasibility of the generated motion control with respect to the locomotion constraints of the robotic vehicle. If no condition is violated, the motion control 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.

Algorithm 1 Algorithm to generate the state lattice

Require: ArcLength : Nominal length of the generated arcs
Require: BranchingFactor: 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 = $\emptyset$
2: (x_0^d, y_0^d) = Lattice origin
3: **for all** headings θ_i **do**
4: (x_f, y_f) = CalculateStraightPath$(x_0^d, y_0^d, \theta_i, ArcLength)$
5: CalculateMotionPrimitive$(x_0^d, y_0^d, \theta_i, x_f, y_f, \theta_i)$
6: **for** $IdxArc = 1 : (BranchingFactor - 1)/2$ **do**
7: R_{IdxArc} = GetRadius(MinRadius,MaxRadius)
8: (x_f, y_f, θ_f) = CalculateArc$(x_0^d, y_0^d, \theta_i, ArcLength, R_{IdxArc})$
9: CalculateMotionPrimitive$(x_0^d, y_0^d, \theta_i, x_f, y_f, \theta_f)$
10: (x_f, y_f, θ_f) = CalculateArc$(x_0^d, y_0^d, \theta_i, ArcLength, -R_{IdxArc})$
11: CalculateMotionPrimitive$(x_0^d, y_0^d, \theta_i, x_f, y_f, \theta_f)$
12: **return** ControlSet

13: **function** CalculateMotionPrimitive$(x_0^d, y_0^d, \theta_0, x_f, y_f, \theta_f)$
14: (x_f^d, y_f^d) = GetNearestLatticeNode(x_f, y_f)
15: θ_f^d = GetNearestHeadingSample(θ_f)
16: (l, R) = CalculateMotionPrimitiveDecomposition $(x_0^d, y_0^d, \theta_0, x_f^d, y_f^d, \theta_f^d)$
17: **if** IsValid (MotionPrimitive$(x_0^d, y_0^d, \theta_0, l, R)$) **then**
18: Add MotionPrimitive$(x_0^d, y_0^d, \theta_0, l, R)$ to ControlSet
19: **end function**

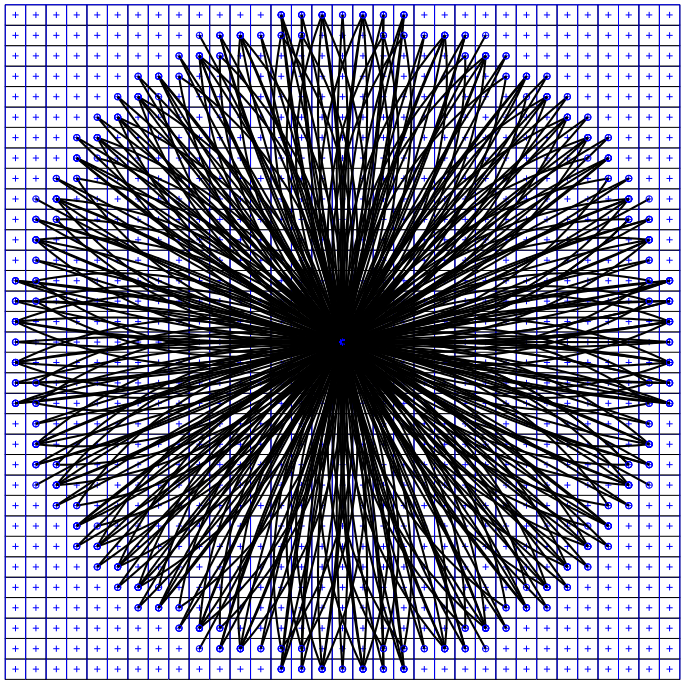


Fig. 7: Control set generated using Algorithm 1

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

V. Motion planning using state lattices

The path planner should be designed to efficiently make use of the precomputed state lattice over the navigation map of the surrounding terrain in order to find a safe and optimal drivable path to reach a selected target. The main path planning algorithm used in this work is an optimised version of the A* algorithm, widely used in robotic path planning applications due to its completeness and optimality with respect to a selected cost function.

A. State lattice cost function

One of the most expensive operations during the path planning process are the collision checks and the calculation of the cost function between two states. We define a cost function by a weighted sum of the length of the edge, the difficulty of the traversed terrain and the distance to a rough path primitive, if available. Path planning using a precomputed motion control set has the advantage of having stored the projection of the motion control over the grid decomposition, called trajectory swath, consisting of all the navigation map cells that are affected during path execution. An example of trajectory swaths is given in Figure 6, where all affected cells are marked in grey. Thus, the calculation of terrain difficulty cost is reduced to the sum of the navigation values in the navigation map encountered by the precomputed arc swath. Thus, the cost function used throughout this paper

is given in eq. 1. The importance of each term of the cost function is weighted through W_{nav},W_{dist} and W_{prim}. The last component of the cost function refers to the distance of the arriving state to a previously calculated path primitive. Through empirical tests, it has been shown that 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.

$$c(s_i, s_j) = W_{nav} \sum_{k \in swath} nav_k + W_{dist} length_{arc}(s_i, s_j) + \\ + W_{prim} dist(s_j, P) \quad (1)$$

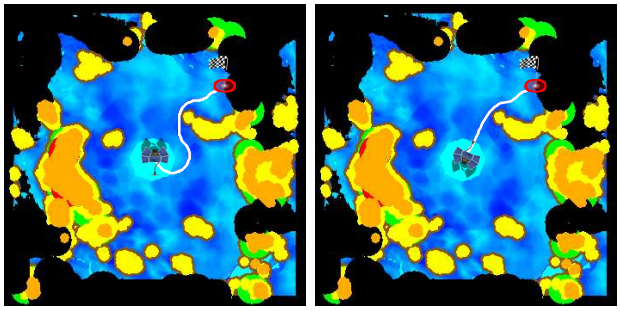
The rough path primitive is provided by a greedy best first search followed by a post-processing smoothing step. It has two main roles: first to validate the current path planning request, by the existence of at least one path between the rover and the goal position, and, second, to provide a general search direction where the nonholonomic path planner should focus for finding the lowest cost path.

B. State lattice heuristic estimates

The heuristic estimate has direct effects on the performance of the path planner. When planning using pre-computed control sets, common heuristics include the Euclidean, Manhattan or Octile metrics to estimate the distance to the goal. 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. Also, when being used with state lattices it is considered to favour straight lines over of low radius arc paths. For this reason, Manhattan and Octile metrics are preferred. Recent work proposes the off-line calculation of a Heuristic Look-Up Table (HLUT) which stores actual costs between discrete states that the path planner will need, using the planner itself [21]. However the applicability of the HLUT is limited when navigating in unknown rough terrain with random obstacle configurations.

C. Authorized planned in-place-turn manoeuvres

The distance the rover can travel towards the selected target is very important for the scientific return of the exploration missions. The objective of nonholonomic path planning is to eliminate planned in-place-turn manoeuvres. This is the reason for which the generated state lattice does not include planned in-place-turns like the GESTALT system. However, as shown in Figure8, when the goal position in the navigation map is located behind the rover, a planned initial in-place-turn is preferred. In the given example, the rover is heading down and the position of the target is marked on the top right side of the navigation map. Figure8a shows the path calculated by the nonholonomic path planner with no authorised in-place turns. The given path consists of the curve of minimum turning radius until the heading of the rover is corrected, followed by several arcs of circle until the goal position is reached. On the other hand, an initial in-place turn is planned in Figure8b followed by nonholonomic path planning using the updated rover heading. The analysis of the resulted paths proves the advantage of using an initial in-place-turn manoeuvre as it helps reduce the distance to be travelled by 33%.



(a) Path length $9.29m$ (b) Path length $6.30m$

Fig. 8: Example of situation when an in-place-turn manoeuvre is preferred

VI. Results

A. Memory use

The memory use is a specific constraint for robotic exploration autonomous navigation missions. This issue can be addressed by storing only a subset of the precomputed state lattice. First due to the translation regularity property of the state lattice we limit the control set calculation to queries originating only from a certain grid cell considered to be the origin of the control set (as shown in Figure 7). Moreover the control set exhibits symmetry with respect to the x-axis, y-axis and the axis with $45°$ orientation, as shown in Figure 9. By exploiting these properties we can gain a huge memory save and pre-calculate and store only $1/8th+1$ of the initial discrete headings. The rest of the controls are restored at request by applying the corresponding transformation to the already computed limited control set. Furthermore we can limit the storage memory requirements by encoding the swaths of the motion controls by using Freeman chain encoding. Thus, by making use of the symmetry properties of the control set and the encoding method for storage, we can reduce the memory requirement for the control set in Figure 7 from $31.5KB$ to $0.4KB$.

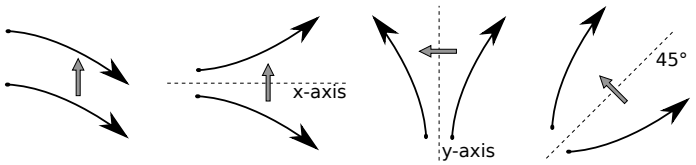


Fig. 9: Lattice controls symmetry transformations

B. Heuristic estimation function performance analysis

An initial focus was on the use of the heuristic function to guide the expansion 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. We perform a simulation study to analyse the influence of the heuristic function on the time performance of the nonholonomic path planner. The parameters used for the generation of the motion control set are specified in Table. I.

Translational discretization	$50mm$ square cell
Heading discretization	360 uniform values (1°)
Primitive length	$400mm$
Branching factor	5 outgoing arcs per state
Minimum turning radius	$1m$
Maximum turning radius	$5m$
Radius discretisation	4 values, uniform

TABLE I: Parameters used to calculate the state lattice control set

A set of 1900 local navigation maps with random start and goal states was generated. We recall that a navigation map has a limited coverage of $17,5m \times 17.5m$ at a resolution of $50mm$ per pixel. As well, the initial rover position is always at the centre of the navigation map and the goal position anywhere in the navigable area encouraging to generate the maximum possible variety of queries. Hence, 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. The objective of path planning for planetary robotic exploration missions is to generate a minimum length, difficulty and execution time path while using circular motion controls to avoid expensive in-place rotations. This is the reason for which we take into consideration the Euclidean distance measure as heuristic estimate. Moreover, In addition to the well known Manhattan and Octile distance measures, we use a hybrid version of these two, entitled here Manhattan-Octile distance. 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 favour straight or diagonal paths. While the Euclidean distance is calculated on line during the path planning process, the other three measures are pre-computed and stored in a distance map. The distance map is calculated starting from the goal position and iteratively filling in each reachable location with the accumulated distance value. This provides an informed heuristic estimate as the distance value reflects the distribution of the obstacles in the navigation space.

The total computation time per query is used as a metric to assess the performance of the nonholonomic path planner given the proposed distance measures. This metric has been chosen knowing that the computational load is proportional to the developed search tree, whose dimension, and consequently used memory, are directly influenced by the heuristic estimate .

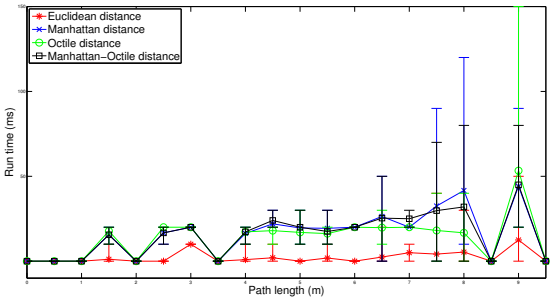


Fig. 10: Computation time comparison regarding the used heuristic function

Figure 10 compares the time performance of the path planning queries when using the different heuristic functions. It is shown that, even if it is computed on-line, the Euclidean heuristic outperforms the rest of the distance measures. The time performance is plotted versus the length of the provided path. Given that the Euclidean distance measure provides the fastest path planning solutions without affecting the trajectory quality, it will be used as heuristic estimate throughout the rest of the experiments in this paper.

C. Real data single query path planning

In this section we discuss the performance comparison of path planning using the precomputed state lattice against the optimized grid path planner proposed by CNES. The experiment is run on real navigation data acquired using the IARES rover, shown in Figure11, during autonomous navigation field experiments performed on the SEROM Mars-like test site.

Fig. 11: IARES rover during field experiments

Overall, the dataset consists of 141 path planning configurations comprising a navigation map of the close environment having the rover at its centre and a random position of the goal in the navigable area around the rover. One configuration is displayed in Figure 12, where the different areas in the navigation map are colour-coded as shown in the legend on the right side of the image. The rover is represented at the centre of the map and the goal position is marked with a red circle and a flag. The generated path is shown in white.

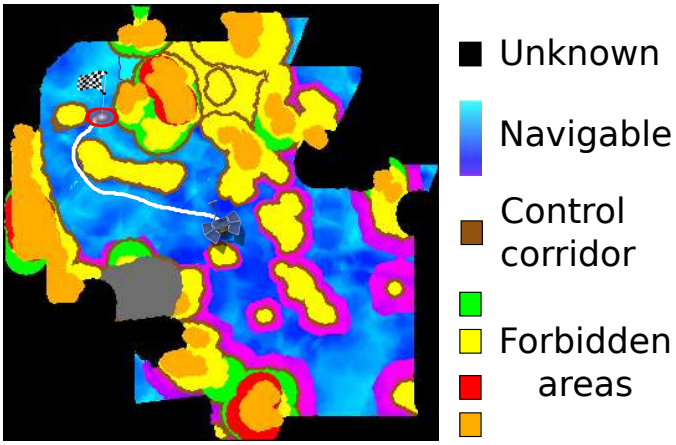


Fig. 12: Example of single query path planning configuration

Nomenclature:
A^* : A^* search on a grid using 8-connected neighbourhood
$R5H1$: Nonholonomic path planning (branching factor=5, heading discretization=1)
$R7H1$: Nonholonomic path planning (branching factor=7, heading discretization=1)
$R9H1$: Nonholonomic path planning (branching factor=9, heading discretization=1)

		A^*	$R5H1$	$R7H1$	$R9H1$
Goal reach rate		100.00%	95.74%	95.74%	95.74%
Runtime (ms)	min	0.20	0.11	0.08	0.12
	mean	2.67	0.48	1.57	11.45
	max	6.85	2.11	7.69	63.75
	median	2.58	0.36	1.18	7.52
Runtime wrt A^*	min	N/A	6.02%	12.24%	13.26%
	mean	N/A	18.70%	51.22%	337.63%
	max	N/A	65.22%	186.36%	1448.67%
	median	N/A	16.41%	42.44%	243.99%
Time improvement rate		N/A	100.00%	90.30%	32.09%
Average path difficulty wrt A^*	min	N/A	74.67%	75.81%	75.95%
	mean	N/A	99.34%	99.11%	99.18%
	max	N/A	118.01%	118.01%	118.01%
	median	N/A	99.81%	99.62%	99.62%
Difficulty improvement rate		N/A	61.19%	68.66%	62.69%
Path length wrt A^*	min	N/A	96.48%	96.48%	96.39%
	mean	N/A	101.74%	101.53%	101.27%
	max	N/A	217.75%	216.02%	215.76%
	median	N/A	99.82%	99.77%	99.71%
Path length improvement rate		N/A	60.45%	63.43%	67.91%

TABLE II: Summary for real navigation data path planning experiment

Table II provides a synthesis of the results obtained when performing the proposed state lattice path planner on real navigation data. We run the nonholonomic path planner using three settings for the branching factor of the state lattince: 5, 7 and 9 outgoing motion controls. The heading discretization is set to $1°$, thus 360 possible heading values per discrete position. The aim is to analyse how the branching factor affects the performances obtained by the nonholonomic path planner compared to the optimized grid path planner proposed by CNES. The performance measures refer to the mission-constraints regarding robotic planetary exploration applications: computation time, length of trajectory, average path roughness. First, the goal reach rate of each method is evaluated. While the grid planner reaches the goal position in all cases, the nonholonomic path planner has 5% of failure rate. This is natural because the nonholonomic path planners are not able to provide a path solution where the environment configuration does not allow a trajectory which obeys the imposed locomotion constraints of the rover.

Second, the statistical measures of the raw computation time values obtained throughout the test are analysed. For a better idea on the performance gain, the normalized computation times with respect to the grid path planner are given. Thus, the state lattice path planner outperforms the grid path planner in all cases when using a branching factor of 5 or in more than 90% of the cases when having 7 outgoing motion controls per state. On the other hand, when using a branching factor of 9 the nonholonomic path planner is slower in approximately 70% of the cases.

Another performance measure concerns the roughness of the terrain lying under the trajectory provided by the path planner. It is concluded that an increase in the terrain smoothness over the generated paths is reached in more than 60% of the cases. The length of the provided solution paths is also very important. Even if in average the paths provided by the nonholonomic path planners are slightly longer than the grid path planner ones, we reach an improvement rate ranging from 60% to 67% depending on the branching factor of the state lattice.

Finally, the memory used by each path planner is also very important in the specific context of planetary exploration. Figure 13 provides a comparison of the memory used by each path planner with respect to the length of the path solution. This comparison excludes the nonholonomic path planner with branching factor of 9 due to its low computation time performance, as represented in Table II. The figure shows that the nonholonomic path planners with 5 and 7 branching factor outperform the grid path planner. This is due to the fact that the grid path planner expands much more nodes as it evaluates each cell in the grid representation during the search procedure. On the other hand, nonholonomic path planners use motion control primitives which cover more cells in a single state expansion. Thus, the use of longer motion controls implies lower memory consumption but has higher risk of path planning failure.

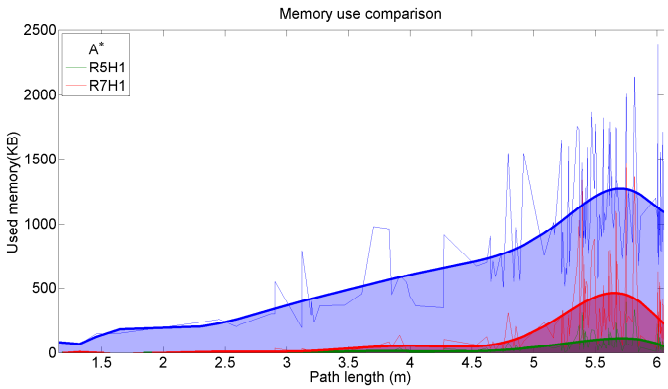


Fig. 13: Memory use comparison per single query path planning with respect to the length of the generated path using: the optimized grid planner proposed by CNES, state lattice path planner with branching factor of 5 and 7

D. Simulated mission scenario path planning

A mission scenario implies successive local path planning stages in the direction of an designated over-the-horizon target, until it is reached. Thus, we conducted an experiment to assess the capabilities of the nonholonomic path planner to perform autonomous path planning in realistic settings by using the CNES simulator. We generated a set of 300 mission scenarios over a Mars-like digital elevation model, with the Euclidean distance between the start and target positions between $25m$ and $70m$.

Table III provides a summary of the results obtained throughout this experimental test. Here, only two versions of the nonolonomic path planner were tested: using a branching factor of 5 or 7. First, the mission goal reach rate of each approach is evaluated. It is observed that there is a minor rate of failure, higher in the case of the A^* grid 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 obstacle configurations. This issue is solved through global path planning which is not the focus of the current paper.

Nomenclature:
A^* : A^* search on a grid using 8-connected neighbourhood
$R5H1$: Nonholonomic path planning (branching factor=5, heading discretization=1)
$R7H1$: Nonholonomic path planning (branching factor=7, heading discretization=1)

		A^*	$R5H1$	$R7H1$
Success rate		98.68%	99.34%	99.34%
Travelled distance wrt to A^*	min	N/A	92.50%	92.44%
	mean	N/A	100.41%	100.08%
	max	N/A	138.71%	105.84%
	median	N/A	100.28%	100.18%
Travelled distance improvement rate		N/A	32.44%	35.79%
Average path difficulty wrt to A^*	min	N/A	93.20%	92.23%
	mean	N/A	98.76%	98.55%
	max	N/A	103.55%	103.79%
	median	N/A	98.82%	98.68%
Average path difficulty improvement rate		N/A	85.95%	90.30%
Time improvement rate wrt to A^*		N/A	62.41%	62.04%

TABLE III: Nonholonomic path planning for mission scenario

Next, the same performance parameters used in the previous test are compared. Similar to the previous experiment, we can conclude that the nonholonomic path planner generates slightly longer trajectories than the grid planner. Moreover, 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 $R7H1$. This is due to the fact the the grid path planner provides trajectories exactly on the border of the obstacles, while the nonholonomic path planner suggests paths which avoid the obstacles at a minimum safety distance. Providing paths very close to obstacles can lead the rover in dangerous areas making the autonomous navigation software to stop due to security reasons. Subsequently, regarding the roughness of the terrain for the computed paths, $R5H1$ outperforms the grid planner in approximately 86% of the queries, with an increase to 90% for $R7H1$.

Finally, the total execution 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 trajectory execution during the total number of cycles. The number of iterations depends on the distance to be travelled to reach the mission target. Throughout this experimental the driven mission trajectories had with lengths varying from $25.5m$ to $78m$ and approximately 6470 path planning iterations (single path planning query) were performed. As expected, the nonholonomic path planner has lower computation time in approximately 62% of the mission scenarios.

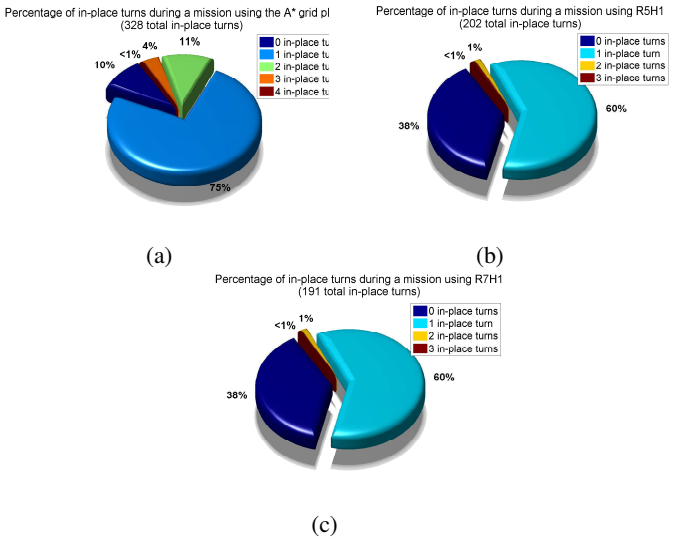
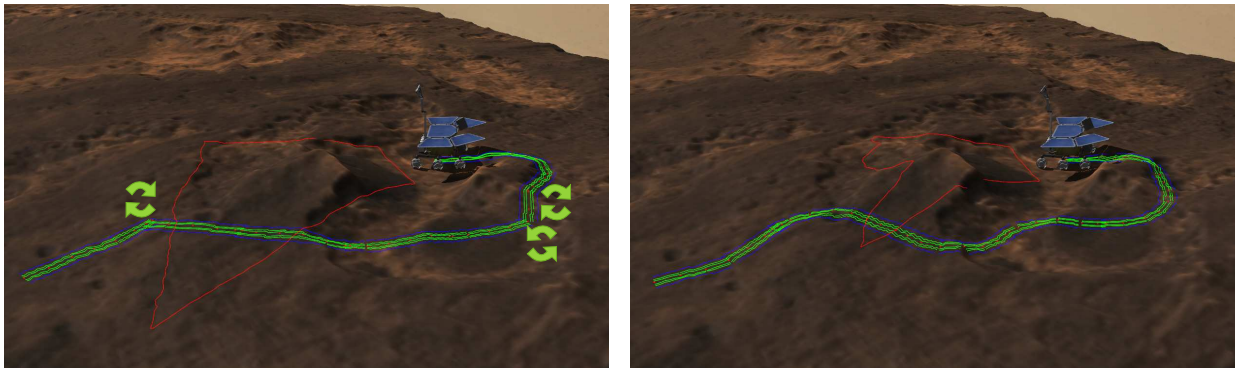


Fig. 14: Percentage of in-place turns executed during each mission scenario using: (a) A^* grid planner, (b) Nonholonomic path planner with branching factor of 5, (c) Nonholonomic path planner with branching factor of 7

The main objective of the nonholonomic path planner is to include the locomotion constraints in the path planning stage, so that during trajectory execution no undesirable in-place-turn manoeuvres occur. Thus, we perform a comparison of the count of in-place turns carried out during trajectory execution in each mission scenario. Figure 14 illustrates the distribution of the counts of in-place turns throughout the experimental set. First, it has to be noticed that the A^* grid planner caused the highest total count of in-place-turn manoeuvres. This is an expected result, 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 $R7H1$. We analyse the number of in-place-turn manoeuvres during a mission trajectory execution. The 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 trajectories 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 nonholonomic path planner. Thus only in maximum 2% of the mission scenarios more than 1 in-place-turns are performed. However, 38% of the mission targets are reached with no in-place-turn manoeuvre, while in 60% of cases one in-place turn is performed during trajectory execution. The distribution



(a) (b)

Fig. 15: Example of in-place turns generated during simulated mission using: (a) A^* grid planner (b) nonholonomic path planner with branching factor of 5

of the in-place-turn manoeuvres stays identical for the two nonholonomic path planners, even though $R7H1$ has a lower count.

Figure 15 provides a comparison of executed trajectories in a simulated mission scenario. The rover is supposed to reach the right side of a rock by avoiding obstacles in its navigation environment. The executed trajectories have a length of $16.56m$ for the grid path planner (Figure 15a) and $16.07m$ for the nonholonomic version(Figure 15b). While the nonholonomic path planner did not generate any in-place turns during path execution, the grid path planner to planned perform 3 in-place-turn manoeuvres. The positions of the in-place manoeuvres are highlighted in Figure 15a using the 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.

VII. Conclusions

This paper introduced a method to generate and use a state lattice in an A^*-like path planning system for a planetary exploration rover. The precomputed state lattice is further used by an optimal path planner of type A^*. The performance of the resulting nonholonomic path planner is compared to the optimized A^*-based grid path planner developed at CNES. Through tests on real navigation data acquired with the IARES rover we prove that the nonholonomic path planner has lower computation time, terrain difficulty and memory use than the grid path planner. Moreover, we validate the capabilities of the nonholonomic path planner to perform autonomous path planning in the context of planetary exploration missions. We proved as well that the inclusion of the locomotion constraints of the rover in the generated state lattice reduced the in-place-turn manoeuvres during trajectory execution. This implies a gain in execution time, locomotion system wear and overall mission scientific return.

Current work is focused on integrating this nonholonomic path planner with a global path planner. As the distance to be planned during a single query increases, the performance of the nonholonomic path planner decreases drastically. This is due to the exponential increase of the dimension of the search tree with the distance and the "curse of dimensionality" as a $3D$ search space is used (position and heading). The idea is to use a multi-resolution global navigation map [22] and a global path planner which can provide an informed sub-goal for the nonholonomic path planner in order to assure the progression of the rover towards the mission target.

References

[1] L. Joudrier, A. M. Garcia, and X. Rave, "ESTEC-CNES Rover Remote Experiment," in *Advanced Space Technologies for Robotics and Automation (ASTRA)*, 2011.

[2] L. Joudrier, M. Malinowski, X. Rave, D. Barnes, G. Paar, V. Ciarletti, and J.-L. Josset, "ESTEC-CNES Rover Remote Experiment #2 with Pan-Cam, WISDOM and CLUPI," in *International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*, 2012.

[3] L. Rastel and M. Maurette, "Autonomous navigation: a development roadmap for exomars," in *9th ESA Workshop on Advanced Space Technologies in Robotics and Automation (ASTRA 2006)*, 2006.

[4] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, vol. 79, pp. 497–516, 1957.

[5] J. A. Reeds and L. A. Shepp, "Optimal paths for a car that goes both forwards and backwards." *Pacific Journal of Mathematics*, vol. 145, no. 2, pp. 367–393, 1990.

[6] A. Scheuer and C. Laugier, "Planning sub-optimal and continuous-curvature paths for car-like robots," in *Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on*, vol. 1, Oct 1998, pp. 25–31 vol.1.

[7] T. Fraichard and J. M. Ahuactzin, "Smooth path planning for cars." in *ICRA*. IEEE, 2001, pp. 3722–3727. [Online]. Available: http://dblp.uni-trier.de/db/conf/icra/icra2001.htmlFraichardA01

[8] T. Fraichard and A. Scheuer, "From reeds and shepp's to continuous-curvature paths," *Robotics, IEEE Transactions on*, vol. 20, no. 6, pp. 1025–1035, Dec 2004.

[9] A. Scheuer and T. Fraichard, "Collision-free and continuous-curvature path planning for car-like robots," in *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, vol. 1, Apr 1997, pp. 867–873 vol.1.

[10] F. Lamiraux and J.-P. Lammond, "Smooth motion planning for car-like vehicles," *Robotics and Automation, IEEE Transactions on*, vol. 17, no. 4, pp. 498–501, Aug 2001.

[11] E. Frazzoli, M. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," in *American Control Conference, 2001. Proceedings of the 2001*, vol. 1, 2001, pp. 43–49 vol.1.

[12] A. Bicchi, A. Marigo, and B. Piccoli, "On the reachability of quantized control systems," *Automatic Control, IEEE Transactions on*, vol. 47, no. 4, pp. 546–563, Apr 2002.

[13] A. Kelly and B. Nagy, "Reactive nonholonomic trajectory generation via parametric optimal control," *International Journal of Robotics Research*, vol. 22, no. 7, pp. 583–601, 2003.

[14] T. M. Howard and A. Kelly, "Optimal rough terrain trajectory generation for wheeled mobile robots," *Int. J. Rob. Res.*, vol. 26, no. 2, pp. 141–166, Feb. 2007. [Online]. Available: http://dx.doi.org/10.1177/0278364906075328

[15] M. Pivtoraiko, T. Howard, I. Nesnas, and A. Kelly, "Field experiments in rover navigation via model-based trajectory generation and nonholonomic motion planning in state lattices," in *Proceedings of the 9th International Symposium on Artificial Intelligence, Robotics, and Automation in Space (i-SAIRAS '08)*, February 2008.

[16] P. Guixé, G. Binet, and A. Medina, "Path planning using state lattices primitives for planetary surface rovers," in *Proceedings of the 11th International Symposium on Artificial Intelligence, Robotics, and Automation in Space (i-SAIRAS '12)*, September 2012.

[17] S. Goldberg, M. Maimone, and L. Matthies, "Stereo vision and rover navigation software for planetary exploration," in *Aerospace Conference Proceedings, 2002. IEEE*, vol. 5, 2002, pp. 5–2025–5–2036 vol.5.

[18] M. W. Maimone, A. E. Johnson, Y. Cheng, R. G. Willson, and L. Matthies, "Autonomous navigation results from the mars exploration rover (mer) mission." in *ISER*, ser. Springer Tracts in Advanced Robotics, M. H. A. Jr. and O. Khatib, Eds., vol. 21. Springer, 2004, pp. 3–13.

[19] J. J. Biesiadecki and M. W. Maimone, "The mars exploration rover surface mobility flight software: Driving ambition," in *In IEEE Aerospace Conference*, 2006, p. 15.

[20] J. J. Biesiadecki, C. Leger, and M. W. Maimone, "Tradeoffs between directed and autonomous driving on the mars exploration rovers." in *ISRR*, ser. Springer Tracts in Advanced Robotics, S. Thrun, R. A. Brooks, and H. F. Durrant-Whyte, Eds., vol. 28. Springer, 2005, pp. 254–267.

[21] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Optimal, smooth, nonholonomic mobile robot motion planning in state lattices," Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-07-15, May 2007.

[22] A. Rusu, S. Moreno, Y. Watanabe, M. Rognant, and M. Devy, "Towards multi-resolution path planning on-board a planetary exploration rover," in *64th International Astronautical Congress*, 2013.