PATH PLANNING USING STATE LATTICES PRIMITIVES FOR PLANETARY SURFACE ROVERS

TURIN, ITALY
4-6TH SEPTEMBER, 2012

Pol Guixé\(^{(1)}\), Giovanni Binet\(^{(1)}\), Alberto Medina\(^{(1)}\)

\(^{(1)}\) GMV S. A U., Isaac Newton 11, P.T.M. Tres Cantos, 28760 Madrid (SPAIN)
pol.guixe@gmail.com, gbinet@gmv.com, amedina@gmv.com

ABSTRACT
This paper describes a solution to non-holonomic path planning based on the use of motion primitives (state lattices approach). Innovation arises from the optimum generation of the motion primitives taking into account not only rover geometry constraints but also back-driving capability, grid resolution, and computational cost.

First, the generation of the state lattice space is described by defining the generation of smooth trajectories as a key requirement. Hence, the curvature of the trajectory has been included among the state lattice variables. As the generation of the motion primitives is a high-intensive CPU process, the required resolution is analyzed to lower the needed computational effort without compromising the precision of the generated curves.

The second part of the paper describes the integration of the generated motion primitives into a grid-search algorithm (A*). Obstacle avoidance scenarios are studied scaling computed solutions at different grid resolutions. The cost function of the path planning algorithm includes the possibility of generating complicated maneuvers as well, by including back-driving capabilities.

This work has been done in the frame of GMV’s internal R&D program related to “Planetary Surface Mobility”.

1. INTRODUCTION
Path planning is an important and critic top-level task for the operation of planetary surface rovers. Previous GMV experiences like the EGP-Rover [1][2] (a 800 kg centaur-rover with 4 wheels and 2 arms) and the MoonHound rover [3] (a 60 kg class skid-steering rover) demonstrated the need of generating smooth curved trajectories rather than sharp trajectories as they bring benefits in terms of energetic efficiency, slippage reduction and steering–motor efforts. Therefore, we have defined as the main objective of this activity the development of a path planning algorithm capable of considering differential constraints and of generating optimal trajectories.

Initially, this paper presents an analysis of the state-of-the-art technology in search-based algorithms (section 2) in order to analyze the different alternatives and choose the one that fits better for the planetary surface exploration requirements.

Then, the state lattice generator algorithm (section 3) is explained, focusing on the mathematical formulation of the primitive curves generation.

Afterwards, the integration of the state lattice and the search algorithm is analyzed (section 4) and back-driving capabilities are described (section 5). Section 6 covers some experimental results, section 7 presents the conclusions of this activity, and finally, section 8 provides future works.

2. PATH-PLANNING AND SEARCH-BASED ALGORITHMS
Several search-based algorithms have been proposed and developed to solve the rover path-planning problem. They can be classified into different groups:

- **Combinational Motion Planning or Geometric-Based Algorithm:** Visibility and Voronoi graphs [4].
- **Sampling-Based Search:** Potential-based search, rapidly-exploring random trees (RRT) and probabilistic roadmaps (PRM) [4][5].
- **Grid Search Based or Discrete Feasible Planning:** A*[4][5], D*[6], ARA* [7] and AD* [8].
A state lattice is a discretized set of all reachable configurations of the system. It is based on discretizing the space and the possible states and attempting to connect the origin state with every final state using a feasible path. In this context, a feasible path is defined as a path that fulfills the differential constraints or the required non-holonomic characteristics (i.e., maximum wheels steering and non-slippage constraints).

In Fig. 1 a dimensionless state lattice is presented, discretizing the space in a grid and the orientation in different defined headings. The primitive path presented in Fig. 1 meets the differential constraints imposed, while the unfeasible paths are not included in the state lattice.

### Table 1. Search Algorithm Analysis

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Combinational Motion Planning</th>
<th>Potential-Based Search</th>
<th>RRT</th>
<th>PRM</th>
<th>Grid-Based Search</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completeness</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Systematic</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Computational Cost</td>
<td>Medium</td>
<td>Low</td>
<td>Medium</td>
<td>Low-Medium</td>
<td>Medium-High</td>
</tr>
<tr>
<td>Optimality</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Differential Constraints</td>
<td>Not directly</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

These algorithms have been analyzed based on these trade-off parameters:

- **Completeness**: An algorithm is considered to be complete if it is capable of finding a solution in a certain time, as long as the solution exists. If there is no solution, the algorithm reports an error.
- **Systematic**: An algorithm is considered to be systematic if it can visit all the possible states without redundancy.
- **Computational Cost**: The time that would take to get a solution in comparison with other algorithms in an identical situation.
- **Optimality**: The capability of the algorithm to obtain as a solution the path at the lowest cost.
- **Differential Constraints**: The capability of the algorithm of accepting differential constraints from a non-holonomic system.

The results of this analysis are shown in Table 1. Obviously, the search algorithm must be able to integrate differential constraints; furthermore, for planetary surface operations a complete, systematic, and optimal algorithm is preferred. Based on these considerations, the grid-based search algorithms are selected to implement the path planning algorithm, despite of their relative high computational cost compared to the other search algorithms.

### 3. STATE LATTICE GENERATION

#### 3.1. State Lattice Concept

A state lattice is a discretized set of all reachable configurations of the system. It is based on discretizing the space and the possible states and attempting to connect the origin state with every final state using a feasible path. In this context, a feasible path is defined as a path that fulfills the differential constraints or the required non-holonomic characteristics (i.e., maximum wheels steering and non-slippage constraints).

In Fig. 1 a dimensionless state lattice is presented, discretizing the space in a grid and the orientation in different defined headings. The primitive path presented

### 3.2. State Lattice Generation

The state lattice generation is based on the following steps:

1. State discretization
2. Primitive path generation
3. Primitive path evaluation
4. State lattice assembly

#### 3.2.1. State Discretization

The state discretization must be regular and consistent with the scale used in the grid search algorithm. Besides, the state discretization may vary depending on the accuracy required for its application.

From the point of view of 2D rover navigation, the basic variables are the Cartesian coordinates \((x, y)\) and the heading \(\theta\) variables. Additionally, due to the non-holonomic constraints of car-like vehicles or Ackerman primitive-based vehicles, the curvature \((k)\) (reciprocal of the radius of the osculating circle of the trajectory) is an important parameter of the trajectory.

Therefore, a state of the system will be defined by the next 4 variables:

\[
\text{State}: (x, y, \theta, k)
\]
3.2.2. Primitive Path Generation

Within this step we generate trajectories capable of connecting an original state to a target state. Different types of curves (like clothoids, polynomial spirals, etc. [10]) have been studied in order to obtain proper trajectories from an original state to a target state. Trajectories based on curvature polynomials of cubic order are considered to be ideal for our purpose, because they can be used to determine a unique trajectory to a target state using a single primitive. Such curves are also the lowest order curves (high smoothness), which are continuous in the torque applied to the rover steering mechanism, allowing to minimize the power consumption at steering motors level.

The cubic curvature polynomial \( k(s) \) is defined depending on the curve length \( s \) by Eq. 1:

\[
k(s) = a + bs + cs^2 + ds^3
\]  

(1)

From the curvature, the heading and the position of the body can be obtained by Eq. 2-4:

\[
\theta(s) = k(s) ds = as + \frac{bs^2}{2} + \frac{cs^3}{3} + \frac{ds^4}{4} + e
\]  

(2)

\[
x(s) = \cos \theta(s) ds = \cos as + \frac{bs^2}{2} + \frac{cs^3}{3} + \frac{ds^4}{4} + e \ ds
\]  

(3)

\[
y(s) = \sin \theta(s) ds = \sin as + \frac{bs^2}{2} + \frac{cs^3}{3} + \frac{ds^4}{4} + e \ ds
\]  

(4)

Eqs. 3-4 are the generalized Fresnel integrals [10] (these integrals can be computed numerically by the adaptive Simpson quadrature). The computation of these integrals is the major computational burden.

In order to calculate the parameters \( a, b, c, d, e \) and \( s_f \) (the final length of a particular curve) of the curve, only Eqs. 1-2 can be used, since isolating the parameters from Eqs. 3-4 is far more complex. This leaves six unknown factors and two equations. By imposing the initial conditions at the origin \( s = 0 \rightarrow x_0, y_0, \theta_0, k_0 = 0,0, \theta_0, k_0 \), two parameters can be obtained directly:

\[
k(0) = k_0 = a
\]  

(5)

\[
\theta(0) = \theta_0 = e
\]  

(6)

There are four parameters which are still unknown and only two equations that can be used. Assuming the parameters \( d \) and \( s_f \) are known and imposing the boundary constraints at the target state \( \mathbf{x}_0 \in \mathbb{R}^2 \rightarrow (x_f, y_f, \theta_f, k_f) \), an equation system Eq. 7, which solves the parameters \( b \) and \( c \), is obtained:

\[
\begin{align*}
b &= k_f - k_0 - ds_f^4 \frac{s_f^2}{4} + \theta_f - \theta_0 - k_0 s_f - \frac{ds_f^4}{2} + \frac{s_f^2}{3} + \frac{s_f^2}{3} \frac{s_f^2}{2} \end{align*}
\]  

(7)

Using this procedure, the parameters \( k_0, \theta_0, \theta_f \) and \( k_f \) have been fixed while \( x_f \) and \( y_f \) have not been fixed (as depicted in Fig. 2). This means that both initial and final curvature and heading are always accomplished, while the final position will change depending on the parameters \( d \) and \( s_f \), which have been assumed as known.

![Figure 2. Errors in primitive trajectories generation: boundary conditions definition error at the left side, and \( d \) and \( s_f \) errors at the right side](image)

In order to find the parameters \( d \) and \( s_f \) to generate a trajectory with an acceptable error, a scan process along a logical range of these parameters has to be carried out. This leads to a computationally expensive process. In order to reduce the computational cost, different scans with dynamic ranges can be carried out. First, the limits of the ranges have to be imposed. The parameter \( d \) affects the curvature and therefore it has to be maintained within certain limits. Moreover, the range of \( s_f \), which is the length of the curve, has to be around the distance between the two points joined by the curve. After the first scans with coarse resolution, the ranges of both parameters are updated towards the solutions while the resolution is progressively refined.

Using this technique, the computational cost can be reduced significantly. In any case, it has to be noted that the primitive generation is not a critical process because it can be done offline.

These cubic curvature polynomial curves are sensitive to the boundary conditions; hence, the origin and target constraints have to be defined carefully, especially the headings and the curvatures. Otherwise, undesired results can be obtained.
3.2.3. Primitive Path Evaluation

Once the primitive trajectories have been generated, they have to be evaluated in order to check if they fulfill the differential constraints or the non-holonomic limitations of a vehicle. One of the parameters that define better the smoothness of a trajectory is the maximum curvature. As shown in Fig. 3, some primitive trajectories can be dismissed just based on their maximum curvature.

![Figure 3. Primitive trajectories for points at a Manhattan distance smaller than 3 and 16 possible headings with a maximum dimensionless curvature of 1.6 (top image) and 0.63 (bottom image)](image)

This evaluation process is used to select which primitive trajectories are going to be finally included in the state lattice and it is strongly influenced by the rover platform.

3.2.4. State Lattice Assembly

For the purpose of reducing the computational cost of the generation of primitive trajectories, these are just calculated for a quadrant as shown in Fig. 3. To obtain a state lattice like in Fig. 1, the proper symmetry operations have to be done. In order to be used by the search algorithm, all the accepted trajectories are finally saved in an internal structure by the state lattice assembly process.

3.3. From Dimensionless to Real

The process explained in section 3.2 is based on a dimensionless space, but it also can be used for real dimension space.

The dimensionless state lattice can be scaled up or down to meet the spatial resolution or accuracy chosen for the targeted application. However, this solution is not preferred because it is relatively expensive from a computational point of view and some states can be missed when scaling.

The preferred solution is to calculate a new state lattice depending on the vehicle characteristics (size, differential constraints, etc.) and the required spatial resolution. This solution has a high computational cost, but allows the analysis of all the possible states. As shown in Fig. 4, the difference in number of states and primitive trajectories is considerable depending on the spatial resolution.

![Figure 4. State lattice for points at a distance smaller than 2.3 meters and 16 heading with maximum curvature of 1 m⁻¹ and a spatial resolution of 0.5 meters (top image) and 0.25 meters (bottom image)](image)

4. STATE LATTICE AND GRID-SEARCH-BASED ALGORITHM INTEGRATION

4.1. State Lattice Integration

In order to integrate our state lattice approach, conventional grid-search algorithms have to be modified. First, the state lattice is integrated with the well-known A* search algorithm. The A* is the base for more complex algorithms such as D* or AD*. Hence, once the A* and the state lattice integration is defined, the integration with these algorithms will be straightforward.

A*[4][5] is an algorithm that analyzes new possible states
from the current system state, and always chooses the cheapest one based on a defined cost function. As shown in Fig. 5, the state lattice integration affects the search of the new states. The state lattice approach only allows the search algorithm to analyze the states that can be reached by the primitive trajectories in the state lattice.

![Diagram of state lattice integration into A* algorithm](image)

**Figure 5. State lattice integration into A* algorithm**

Furthermore, conventional A* algorithm is based on a two-dimension state \((x, y)\) search, but as the state lattice is based on a four dimension state \((x, y, \theta, k)\), the A* has to be modified to perform a four-dimension search. This leads to a higher computational cost, which increases with each extra dimension.

### 4.2. Cost Function

The cost function of the A* search algorithm has to be defined properly, because it affects the type of path obtained and the time needed to find a solution. The heuristic cost is discussed separately in section 6.2. Then, the cost function related to the travelled distance is enhanced by including the heading and the curvature introduced by the state lattice approach as shown in Eq. (8):

\[
\text{Cost} = A \cdot \text{Dist} + B \cdot \text{Head.} + C \cdot \text{Curv.} \quad (8)
\]

As shown in Eq. 8, the parameters may have different relative weights that will lead to different types of trajectories for similar scenarios, as in Fig. 6. If the changes in heading and the large curvatures are penalized, the obtained path will be as straight as possible, whereas if they are not penalized, the trajectories obtained will be more curved.

The computational time to obtain a path can be optimized by choosing the proper weighting variables \(A, B\) and \(C\). Fig. 6 shows two cases: the first case requires fewer states to be analyzed before the optimal path is found, while the second case requires more states to find an optimal path.

### 4.3. Obstacle Avoidance

The path-planning algorithm obtained in section 4.1. provides good results with large obstacles, larger than half of the state lattice size. In the case of small obstacles, the path-planning algorithm tries to cross-navigate them, being not suitable for real rover navigation.

This inconvenience can be solved by adding an obstacle-avoidance function within the function in charge of searching the new states. This obstacle-avoidance function analyzes the grid points close to the primitive trajectory inside the state lattice and avoids trajectories too close to obstacles (see Fig. 7). The number of grid points to be analyzed depends on the size of the vehicle, the size of the state lattice and the grid resolution.

![Diagram of path planning and obstacle avoidance](image)

**Figure 6. Path planning results in a similar scenario. Red dots: obstacles. Green lines: possible paths analyzed. Blue line: optimal path. Top image: the distance has a heavier weight than the heading and the curvature. Bottom image: the heading and the curvature have a heavier weight than the distance**

**Figure 7. Obstacle avoidance: primitive trajectory (green line) and grid points (red dots) to be analyzed**
Fig. 8 shows how the path-planning algorithm is capable of dealing successfully with complex scenarios with small obstacles.

5. REAR MOTION MANEUVRRES

The capability of performing rear motion maneuvers is considered to be particularly interesting, because it allows solving some path planning problems that would be impossible to solve without this capability, as shown in Fig. 9.

This capability was integrated into the path-planning by adding an additional dimension into the space state: the sense (or direction) of motion. The search algorithm has to know the sense of the motion of the current state, and whether the new state will be reached moving forwards or backwards. If it is accepted that the vehicle has the same differential constraints when moving forwards or backwards, previously generated state lattice does not need to be modified. Hence, this modification only affects the search algorithm, which has to be modified to allow a five dimensional search \((x, y, \theta, k, \text{motion sense})\).

Changing the sense of the motion is not easy in real applications, as the vehicle has to stop completely and start moving again, and in terms of energy consumed it is not an effective maneuver. Therefore, as this capability has to be used only when it is strictly needed, it will be strongly penalized in the cost function.

Note that the computational cost of the path-planning process is increased by adding an additional dimension in the algorithm and using a more complex cost function.

6. APPLICATION TO REAL SCENARIOS

Up to this point, a synthetic scenario with obstacles modeled as points in the grid has been used to design the path-planning algorithm. In this section the simulation of real scenarios using this path-planning algorithm is presented. A proper management of the spatial resolution and the computational time has to be done when the path-planning algorithm wants to be simulated in real conditions.

6.1. Space Discretization

The obstacles in a real scenario are not single dots in a grid; they are obstructions and barriers without a regular shape. Generally, real obstacles are not faithfully represented; their size is increased by considering the vehicles size and an error budget in order to reduce collision risk. The resulting map is ready to be analyzed (see Fig. 10). However, first the space has to be discretized according to the desired spatial resolution, a parameter that also affects the grid search and the state lattice.

Changing the sense of the motion is not easy in real applications, as the vehicle has to stop completely and start moving again, and in terms of energy consumed it is not an effective maneuver. Therefore, as this capability has to be used only when it is strictly needed, it will be strongly penalized in the cost function.

Note that the computational cost of the path-planning process is increased by adding an additional dimension in the algorithm and using a more complex cost function.

6. APPLICATION TO REAL SCENARIOS

Up to this point, a synthetic scenario with obstacles modeled as points in the grid has been used to design the path-planning algorithm. In this section the simulation of real scenarios using this path-planning algorithm is presented. A proper management of the spatial resolution and the computational time has to be done when the path-planning algorithm wants to be simulated in real conditions.

6.1. Space Discretization

The obstacles in a real scenario are not single dots in a grid; they are obstructions and barriers without a regular shape. Generally, real obstacles are not faithfully represented; their size is increased by considering the vehicles size and an error budget in order to reduce collision risk. The resulting map is ready to be analyzed (see Fig. 10). However, first the space has to be discretized according to the desired spatial resolution, a parameter that also affects the grid search and the state lattice.

Changing the sense of the motion is not easy in real applications, as the vehicle has to stop completely and start moving again, and in terms of energy consumed it is not an effective maneuver. Therefore, as this capability has to be used only when it is strictly needed, it will be strongly penalized in the cost function.

Note that the computational cost of the path-planning process is increased by adding an additional dimension in the algorithm and using a more complex cost function.
another approach has been used: each grid node has a certain probability to have an object, where 100% means that there is an obstacle (and therefore must be avoided) and 0% means that it is a completely obstacle-free space (see Fig. 11). This means that the grid nodes have an associated risk that can be considered inside the cost function, and consequently a path can go through a high-risk obstacle node or area only when it is worth it or there is not another solution. The benefit of this method is that it allows more solutions that can be directly controlled by the search algorithm cost function.

Additionally, the importance of the spatial resolution has to be noticed. Not only it affects computational time (finer spatial resolution will require a higher number of calculations), but also the solutions obtained. As shown in Fig. 11, different solutions have been obtained for the same case and set up of the path-planning algorithm, just depending on the spatial resolution of the space.

Obviously, to obtain an accurate and reliable path a finer spatial resolution gives better performances, however the computational cost increases; hence a trade-off has to be done depending on the operational requirements. Moreover, in order to check if the solution uncertainty is a critical issue, proper sensitivity studies have to be done for each application.

6.2. Computational Effort Management

In some applications or operations the time to obtain a feasible path can be limited and critical. The number of operations that are needed by the search algorithm to find an optimal solution is not fixed, and therefore the time needed to calculate a path may exceed an acceptable limit. In those cases, the optimality of the path may be less important than the maximum computational time, since increasing the heuristic cost of the search algorithm helps reducing the computational time by finding sub-optimal paths. While there is available time, the algorithm can reduce the incremental factor of the heuristic cost and search optimal paths. This algorithm applied to simple A* is known as ARA* (A* replanning algorithm) (more details in [8]).

As shown in Fig. 13, in some cases the results obtained with different heuristic costs are similar, but the number of states to be analyzed can be reduced drastically, and so may be the computational time.

Figure 11. Simulation in a real scenario of 20x20 meters. Obstacles: black. Free space: white. Grid nodes with obstacle risk: grayscale. Path: blue line. Different solutions depending on the spatial resolution used: 0.25 meters (top image) and 0.5 meters (bottom image)
7. CONCLUSIONS

This paper describes an implementation of the state lattice approach for a rover platform that has been performed at GMV in an internal project. The objective of the project was to implement a path planner that inherently takes into account the geometrical characteristics of the vehicle that has to traverse the generated paths.

The paper has recalled the concept of motion primitives and state lattices as well as their generation. A generated state lattice is unique for a defined spatial resolution (i.e. DEM resolution), heading and curvature discretization: However, it is independent from the motion constraints of the vehicle or rover. Once the motion constraints are applied to the full state lattice, a smaller state lattice can be generated, thereby creating a number of curves that can be combined by a search algorithm into forming feasible and smooth paths for non-holonomic vehicles.

The backwards maneuvering that has been introduced is a great enhancement to the path-planner algorithm, as it can reduce the complexity of the paths or even achieve targets that would not be reachable. It does, however, add complexity to the cost function and therefore increases the computational effort to reach a solution. In this context, an implementation of the ARA search algorithm manages the allowable time for the path-planner function online while always providing the best solution in the available time.

8. FUTURE WORK

Within GMV’s internal R&D program, it is foreseen to further expand this project by studying the benefits of primitive trajectories with initial and final curvatures different than 0. In addition, the concept of an optimal state lattice should also be addressed, as a single optimal state lattice could be used in various grid resolutions.

Finally, an implementation on a rover would also require studying the possibility of implementing other search algorithms that would enable long traverse, such as the D* search algorithm.

9. REFERENCES


Figure 12. Simulation of the same path-planning case with different incremental heuristic cost factors. Obstacles: black. Free space: white. In grayscale, areas likely to be avoid (steep slopes, irregular soil, etc.). Path: blue line.