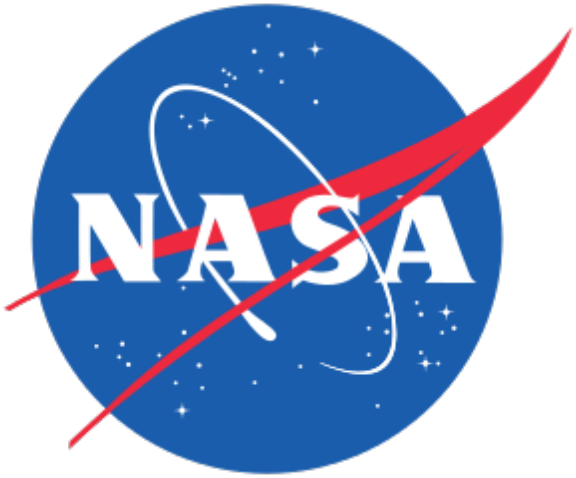
Designing the path planning algorithm for Mars2020

Guillaume Matheron, ENS

Internship supervised by Olivier Toupet

Jet Propulsion Laboratory / California Institute of Technology

Feb - Jun 2016



Abstract

During this internship, we designed and tested a method allowing an autonomous rover to navigate its unknown environment to reach a target. This algorithm is novel because it does not rely on global positioning or staying on a fixed lattice. It uses a model of its own geometry that gives a pessimistic (but safe) guess to whether a state is safe for the rover's mechanical integrity, based on a terrain model.

This path-planning algorithm needs to predict where the wheels will drive, and ensure that there is sufficient margin between the belly pan of the rover and the top of each rock the rover will straddle. This constraint is especially difficult given that the rover has a rocker-bogie suspension system which varies the height of the belly pan based on the geometry of the terrain the wheels drive on.

This algorithm was designed with a specific mission in mind: Mars2020. As such, it is already adapted to use on Mars, with features such as a very low computational cost and a conservative approach to navigation.

In May 2016, the results provided by the early tests of this method convinced NASA to approve an Engineering Change Request, effectively funding the further development and testing of this method until mission launch.

Contents

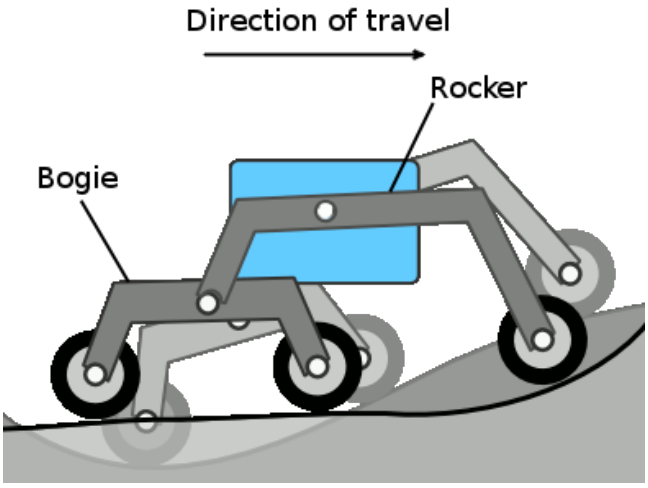


Figure 1: Rocker-bogie suspension (simplified)

1 Anatomy of a Mars rover

1.1 Rocker-bogie suspension

All NASA Mars rovers use the same suspension mechanism, called a rocker-bogie suspension. It has three wheels on each side. The middle and rear wheels are attached to a bogie. The front wheel and the bogie joint are attached to a rocker, which is itself mounted on the rover body using a free pivot joint.

Since the rover's body is only supported by two coaxial pivot joints, its pitch is constrained by a differential. When all six wheels are on flat ground, the rover body is level. When one rocker is rotated raised by an angle β relative to the rocker body, the other rocker is rotated by an angle $-\beta$, meaning the pitch of the body is always the average of the pitch of the two rockers.

This configuration allows the body to be minimally impacted by rocks that the wheels traverse, keeping it as level as possible.

1.2 Actuators

Each of the six wheels of the rover is torque-driven by a motor, and the front and rear wheels are mounted on steering servos. This allows the rover to perform in-place turns. However, hardware engineering constrains the number of actuators in use simultaneously. As a consequence, the rover has to stop before being able to steer its wheels.

This means that Mars path planning algorithms can only output circular arcs.

1.3 Inputs

Mars rovers have many on-board cameras. The captured images are sent to Earth on a daily basis, during the night.

The main cameras useful for navigation are the NavCams (located on the mast, they can see the farthest), and the HazCams (they look at a wide-angle picture of the front and rear wheels).

The NavCams are arranged as a stereo pair, and their images are used to construct an elevation map of the surface.

Rear cameras are used to perform visual odometry: small ground features are tracked to evaluate the motion of the rover. This motion is integrated and correlated with the data from the IMU (Inertial Measurement Unit) to give an approximation of the rover's global position on Mars. Since Mars' magnetic field is not strong enough to have an accurate heading, the position of the Sun can be used.

2 Challenges of driving on Mars

Mars terrain is composed mainly of rocks and sand. Both are inherent dangers to the integrity of the rover. Wheels tend to get stuck in sand (which is how rover Spirit was lost, see Figure 4a), and rocks can damage wheels (see Figure 4b) and block the rover's progression.

The size distribution of rocks on a Martian surface is described by its Cumulative Fractional Area (CFA) [2]. This notion was proposed by Goldberg in 2002. He showed that the rock size distribution on Mars can be

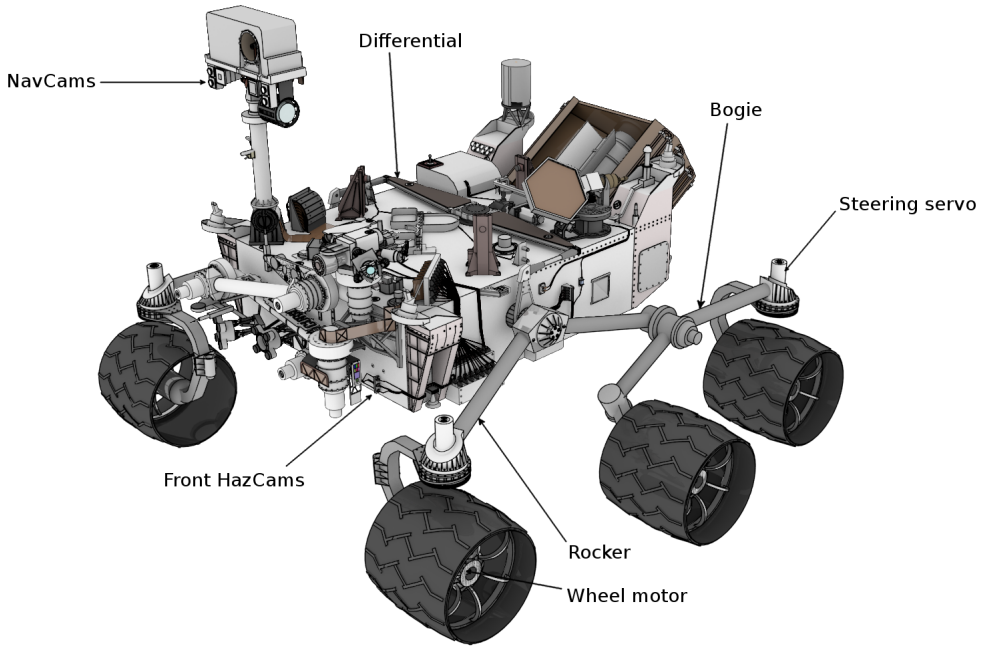


Figure 2: Rocker-bogie suspension on Curiosity

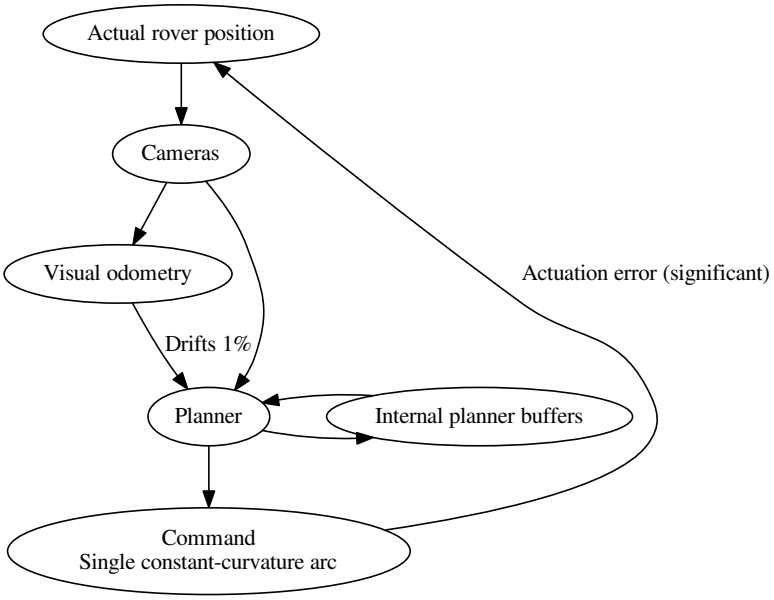
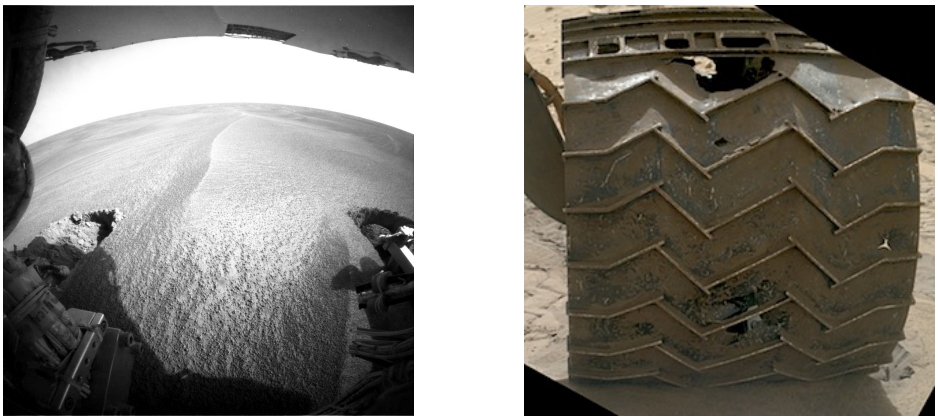


Figure 3: Actuation model of autonomous Mars rovers



(a) Front HAZCAM on MER-B, sol 468, stuck in a 30cm sand dune

(b) Damaged wheel on Curiosity, on sol 713. Several holes are visible in the wheel skin

Figure 4: Mars hazards

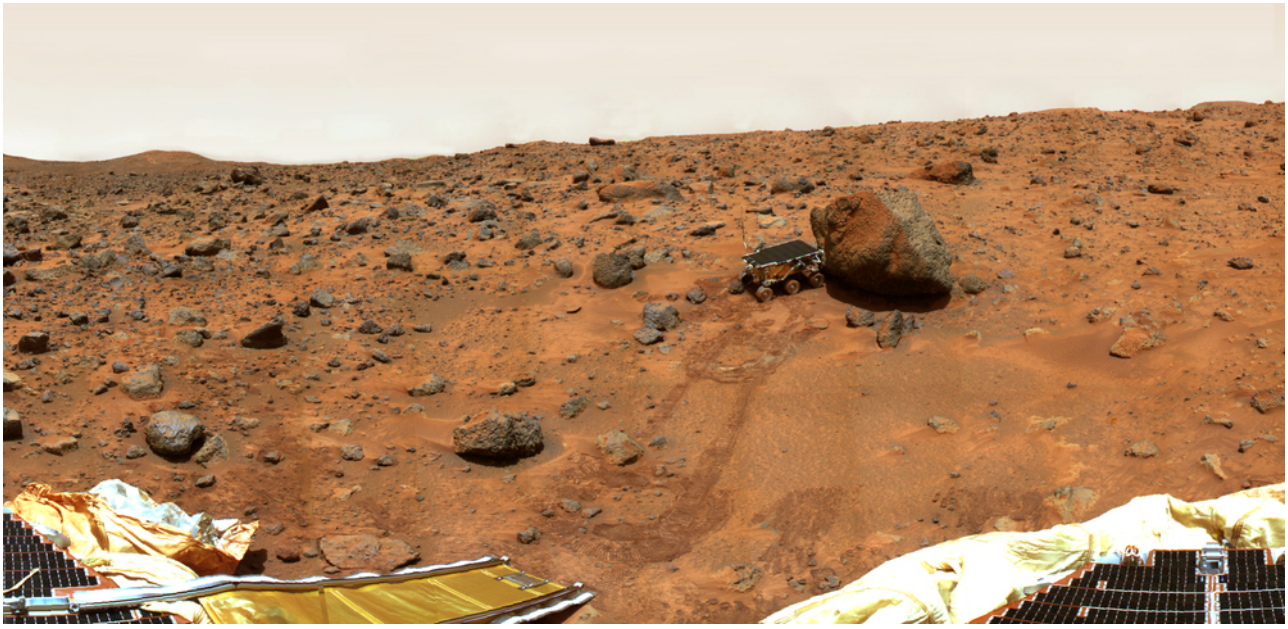


Figure 5: Extract of a panoramic view of the Pathfinder landing site. For scale, the small rover in the picture called Sojourner is about 30cm tall. CFA in this picture is about 20%

described by a single parameter, the fraction of the surface that is covered by rocks (this parameter is called CFA).

The CFA of the Pathfinder landing site (shown in Figure 5) is about 20% [2].

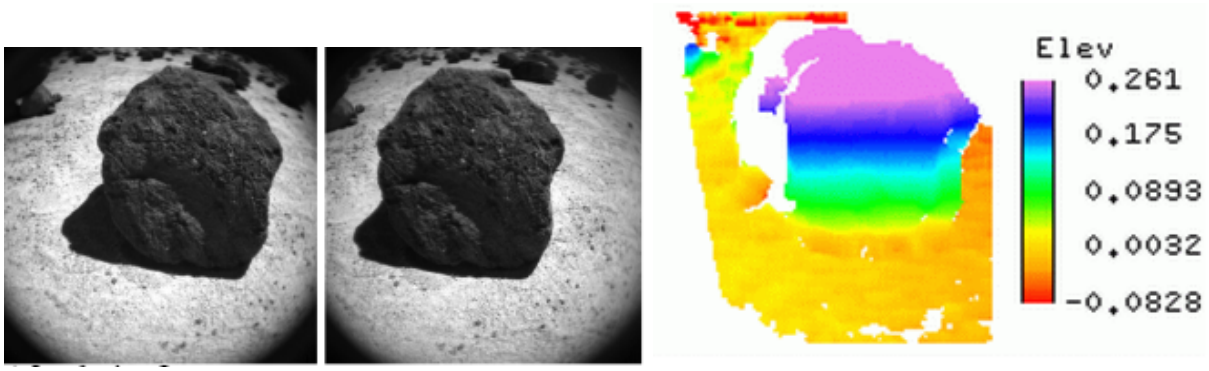
The high-radiation environment means all electronic components that go to Mars undergo extensive testing processes, so the choice of components is very limited. Curiosity runs on a few hundreds megahertz CPU, and a few hundreds megabytes of RAM memory.

3 The need for autonomy

Most Mars rovers have some form of autonomy. The two main problems with driving a rover manually on Mars are the latency, bandwidth and frequent occlusions of the radio connexion.

The light round-trip-time to Mars varies between 8 and 50 minutes depending on the relative position of Mars and the Earth. This makes real-time driving impractical. This does not however prohibit making a short sequence of commands on Earth, then sending them to Mars in a batch.

However, the bandwidth of the radio signals relayed to Earth is very low and depends on the position of the orbiters around Mars. In addition, all Mars missions share the same orbiters and antennas on Earth for data return, so the pictures from Mars are only retrieved every day (which makes sending commands manually very slow). However, this method is still often used on Curiosity.



(a) Raw NavCam images (b) Resulting elevation map

Figure 6: Vision capabilities of MSL

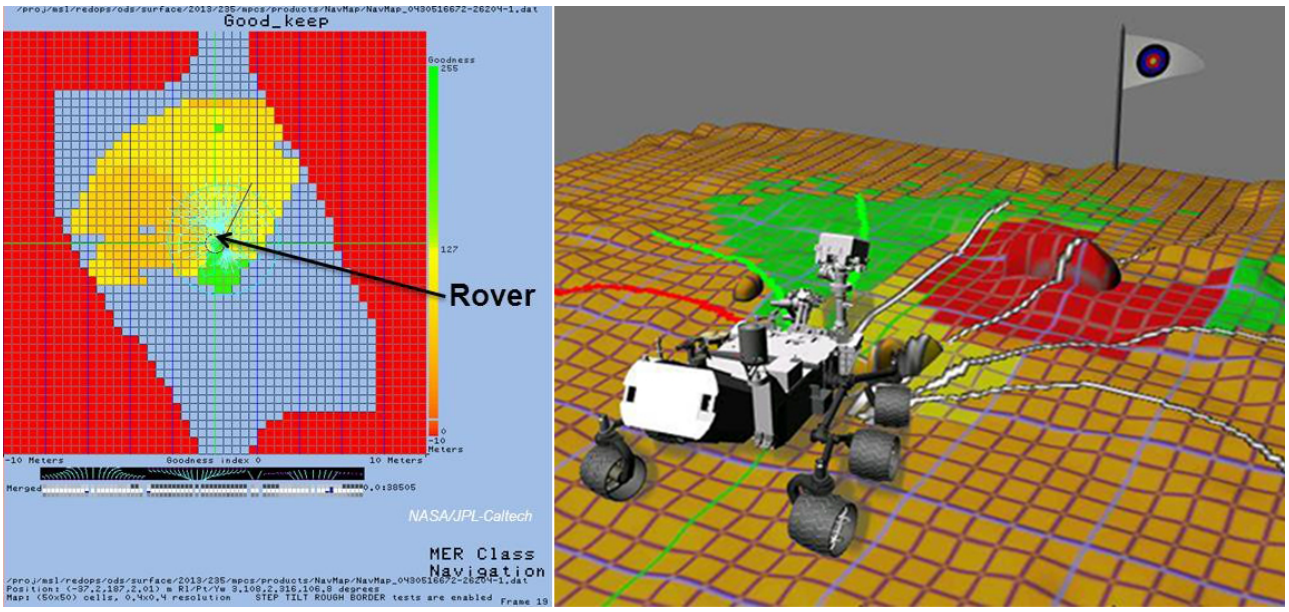


Figure 7: Gestalt choosing the best candidate arc

Autonomous navigation algorithms don't have to deal with the limitations of radio transmission to Earth, because they can use the pictures taken by the rover instantly, and discard them afterwards. This greatly increases the distance the rover can travel each day.

4 Previous work

4.1 Gestalt

The Mars rover Curiosity (also known as Mars Science Laboratory or MSL), launched in 2011, uses two driving modes: manual and AutoNav. Manual driving means a ground operator analyzes the pictures sent overnight, and manually defines the arcs to be executed. Unfortunately, since the NavCams cannot see very far, this limits the distance driven per day to a few dozen meters.

AutoNav implements the Gestalt [1] algorithm to choose its path without having to wait for operator input. It uses the elevation map produced by the stereo vision module (Figure 6) to construct a cost map that indicates which areas are dangerous (because of rocks and slope most notably) and which are safe.

Then the rover computes several candidate arcs, and any arc that comes closer than 3 meters of an unsafe area is excluded. The remaining arcs are sorted based on the shortest path between their tip and the goal, using a grid-based global planning algorithm (Figure 7).

4.2 Shortcomings of Gestalt

Because Gestalt only evaluates one arc at a time, there is a trade off depending on the length of the arc. Longer arcs will allow the rover to turn early when facing obstacles, but are more likely to be obstructed. Shorter arcs

are more agile, but make the rover myopic.

Furthermore, tests using a Voronoi map have shown that when the terrain's CFA is greater than 10%, there is often no path at all that stays far enough from the obstacles. Mars 2020's mission requirements state that the rover must be able to autonomously traverse terrain up to 15% CFA. The test was coducted as such: generate a random terrain with a fixed CFA and fixed start and end points, then build a Voronoi graph from all the rock centers. Remove all segments that are less than three meters from the nearest rock. If the resulting graph has no path from start to goal, then Gestalt is sure to fail.

4.3 Other approaches

Other approaches were considered as a replacement for Gestalt, and part of my job at JPL during my internship was to test these approaches.

The rover's state is described by three parameters: (x, y, θ) where θ is the heading. However, the vehicle is non-holonomous since it has only two degrees of freedom. This causes a differential constraint that can be written $-\dot{x}\sin\theta + \dot{y}\cos\theta = 0$.

Actuation error can be very substantial because of slip, terrain geometry and other factors. Since reactive driving is not possible (because of actuation constraints), the only way to keep the rover on a pre-defined set of accessible states is to use periodic local path re-acquisition, as described for instance in [3].

If the problem of staying in a pre-defined set of accessible states is solved, the most promising approaches to Mars path planning seem to be lattice-based approaches, such as [6] or [4]. This approach was tested in the Mars Yard at JPL, and the results are exposed in [5].

The main problem of lattice-based approaches are their computational complexity, the difficulty of re-acquiring a given path after actuation errors (that can be as dramatic as 100% on sand slopes for instance), and the difficulty of knowing the exact position of the rover on Mars. Indeed, the position is tracked using visual odometry, that can drift up to 2% in terms of position. Because of the lack of magnetic field on Mars, heading cannot be determined using a compass. Methods of getting global heading data include tracking the sun or overhead orbiters (which is a lengthy process).

All of these factors led us to abandon lattice-based planning for Mars2020, and instead concentrate on tree-based planning (see Section 5.2).

5 Our approach

5.1 Kinematic state estimation

5.1.1 Motivations

One of the inherent problems of Gestalt is that the traversable configuration space does not depend on the heading of the rover: any pose that is deemed safe will be safe in any orientation. This prevents the rover from straddling obstacles, and induces an uncontrolled margin.

A way to address this problem would be to compute the full kinematic settling of the rover on the terrain surface and compute various hazard metrics (such as ground clearance and tilt) each time we want to know whether a pose is safe. However, path planning in complex terrain requires many such evaluations and the flight hardware is not capable of such intensive computations.

We designed a way to compute bounds on several hazard metrics very quickly. The proposed algorithm is described below.

5.1.2 Kinematic state estimation algorithm

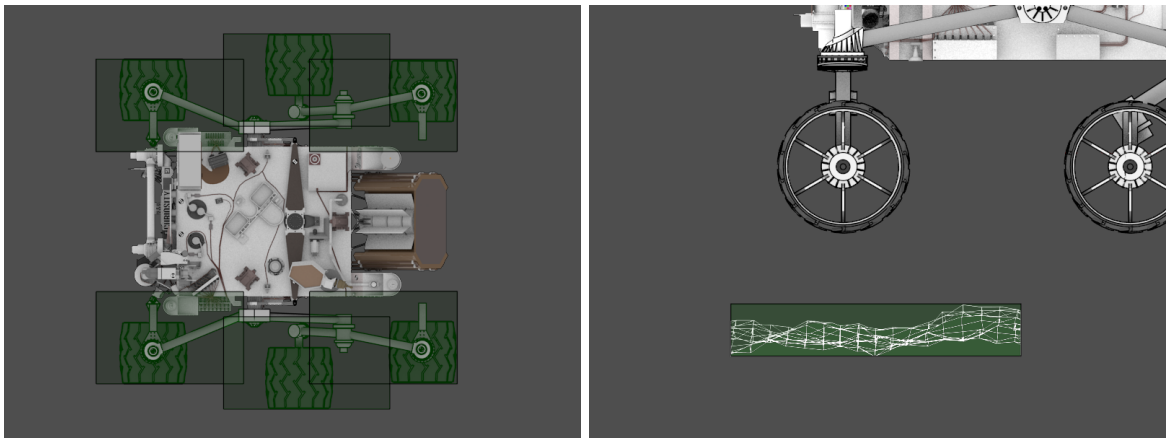
Figure 1 shows a schematic of a rocker-bogie suspension.

The rover origin is on the ground between the two middle wheels when the rover is on flat ground, and is fixed relative to the body.

We assume we know the position of the rocker's origin (x, y), and the rover's heading (θ). We also assume we have a model of the terrain under the rover.

1. All the algorithm is performed in a referential local to the rover. Axis X is forward when the rover is in (x, y) and heading θ, with neutral pitch and roll. Similarly, Y is right in this pose, and Z is down.

2. Because of mechanical constraints (endpoints and vehicle stability), we have bounds on the pitch of each suspension, and the roll of the rover. Given these bounds, we can compute bounds on the X and Y position of each wheel's center. Knowing the size of the wheels, and assuming the contact point between a wheel and the ground is on the lower half of the wheel, we compute X and Y bounds on the terrain points that may be in contact with each wheel (see Figure 8a)



(a) Step 1: XY bounds on wheels

(b) Step 2: XYZ bounds on wheels

(c)

Figure 8: Kinematic state estimation algorithm

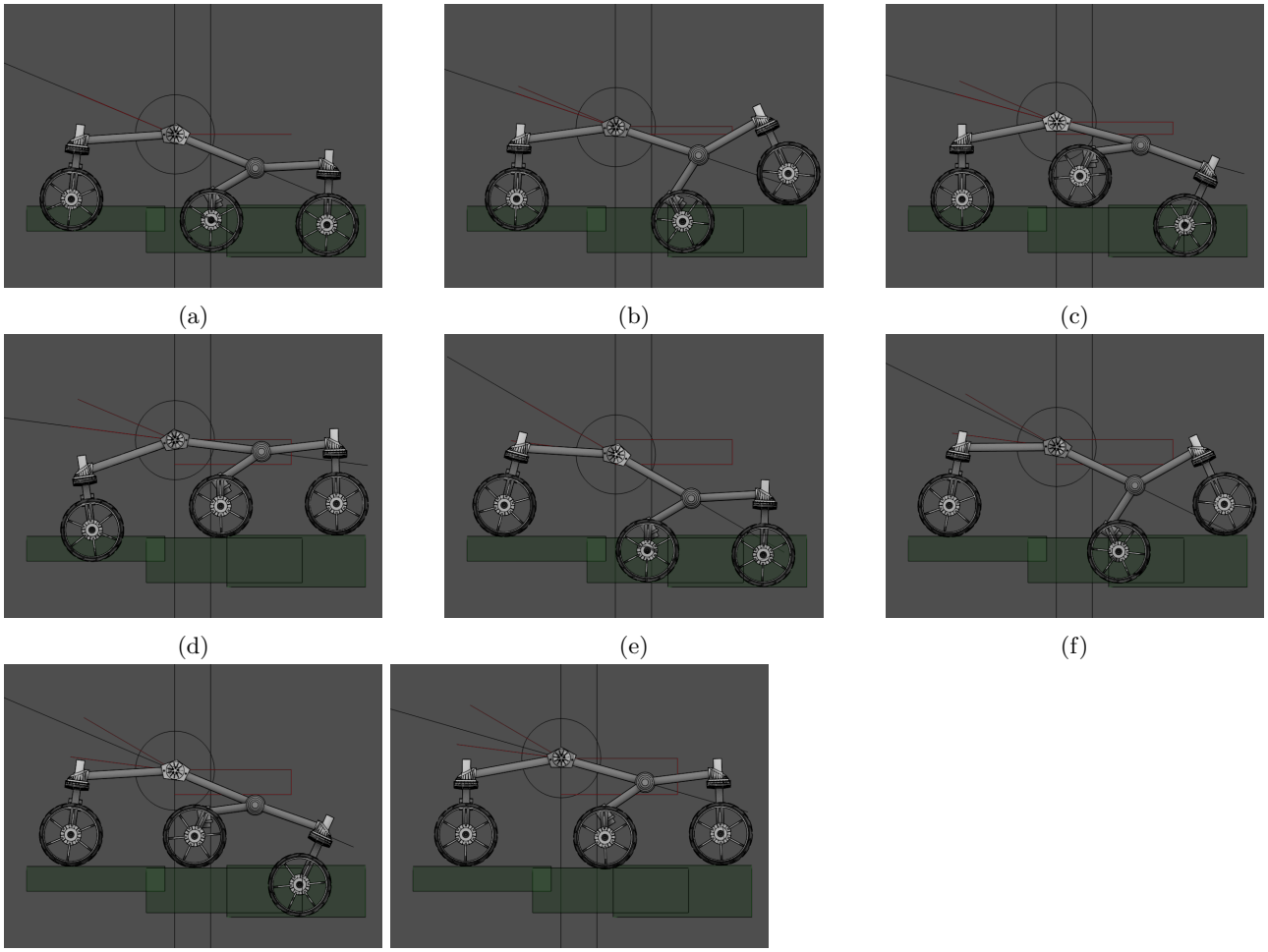


Figure 9: Kinematic state estimation algorithm for a single suspension

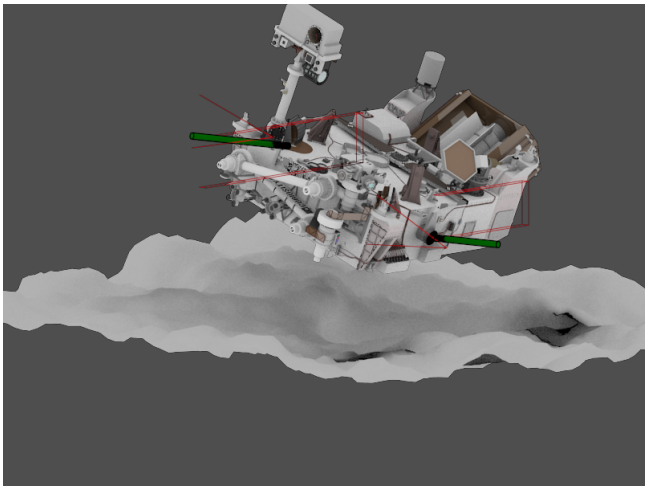


Figure 10: Worst-case scenario for clearance

3. For each wheel, we use the terrain data and the X,Y bounds to compute Z bounds on the height of each wheel (see Figures 8b and 8c).

4. For each suspension (a suspension is composed of a rocker and a bogie, there are two suspensions on the rover, the left suspension and the right suspension):

    - The Z position and pitch of the rocker's attachment point on the rover's body are functions of the Z position of each of the three wheels. These two functions are monotonous in each of their three variables, so their extrema are reached when their arguments are extremal. Therefore, we only need to test eight configurations (corresponding to the eight vertices of the cube representing the domain of the function) to get bounds on the rocker's pitch and Z position (see Figures 9a to 9h).

5. Since the rover's body is only attached to the wheels through the two rockers' attachment points, its configuration is fully determined by the height (Z position) and pitch of these attachment points. The bounds on the height of the attachment points give bounds on the rover's roll and height, and the bounds on the pitch of the rockers give bounds on the rover's pitch (see Figure 10).

6. The bounds on the rover body's height, roll and pitch give bounds on the clearance between the belly pan of the rover and the ground.

Three metrics are used to evaluate vehicle safety:

- *Ground clearance* must stay greater than 20 cm.
- *Tilt* (which can be computed from the pitch and roll bounds must stay below 35 deg.)
- *Wheel Z uncertainty* (the span between the Z bounds of each wheel, which we also call *wheel drop*) must stay below 40 cm. Indeed, larger uncertainty means the wheel may suddenly drop off a high ledge, causing mechanical damage to the wheel (which is not designed to endure high dynamic forces) or to the suspension itself.

5.1.3 Performance and safety

Benchmarks on my work computer show a single kinematic approximation takes about 1.5 milliseconds, with very little deviation. With appropriate scaling, this algorithm is expected to run on the flight hardware in about 15 milliseconds.

We conducted Monte-Carlo testing to ensure that the kinematic approximation always returned bounds that matched the ground truth.

5.2 Tree-based sampling of the configuration space

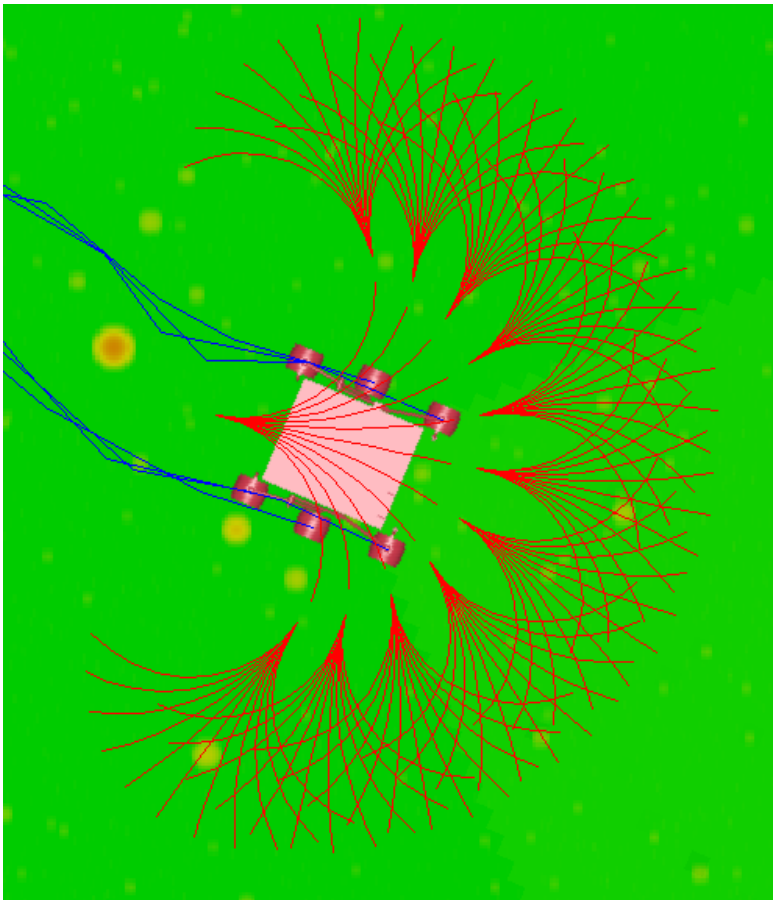


Figure 11: Example of a tree-based search. Using the conventions defined in Section 5.2.2, this tree is written as the set: $A(3, [-0.5 : 0.5 : 11]) \times A(3, [-0.5 : 0.5 : 11])$. In this figure, the rover has already performed motion $A(1, 0.125)$ since the tree was generated.

5.2.1 Motivation

Since we don't know all the terrain, we can only compute a greedy estimate of our path, and re-evaluate once we have more information. In addition, the drift of visual odometry means that data that was accumulated earlier is less reliable. Since the on-board hardware is not capable of stitching the vision data, the elevation maps produced by the stereo pair is stored in a rolling heightmap buffer according to the current position estimate given by visual odometry. This buffer has a limited size because of memory constraints, so the rover forgets the exact geometry of any terrain point farther than 10 meters.

Since the Mars mobility system is not holonomous, and we can only run visual odometry every meter (because of performance constraints), we can't do any predictive control. Once we choose to execute an arc, we may not arrive exactly where we predicted, and there is no way to get back to the expected path in reasonable time. This means that we have to stop and re-plan often, and we can't use any of the data computed previously.

5.2.2 Definitions

Here are the motions that are allowed by the rover:

Definition 1. $A(d, \Delta\theta)$ Turn the wheels so that the turning radius is $\frac{d}{\Delta\theta}$, then go forward d meters (so that the path of the rover's center is d meters long).

Definition 2. $TIP(\Delta\theta)$ Turn in place so that the heading changes by $\Delta\theta$.

Definition 3. The set of allowable motions is defined as $M := \{A(d, \Delta\theta) | d \in \mathbb{R}, \Delta\theta \in \mathbb{R}\} \bigcup \{TIP(\Delta\theta) | \theta \in \mathbb{R}\}$

Definition 4. Evenly-spaced set. $\forall a \in \mathbb{R}, n \in \mathbb{N}^*, b \in \mathbb{R}, [a : b : n] := \left\{a + k\frac{b-a}{n} | k = 0..n\right\}$

Definition 5. The tuple (x, y, θ) describes a pose of the Rover in C-space (cartesian coordinates and heading). The set of all such poses is $P := \mathbb{R} \times \mathbb{R} \times \mathbb{S}$.

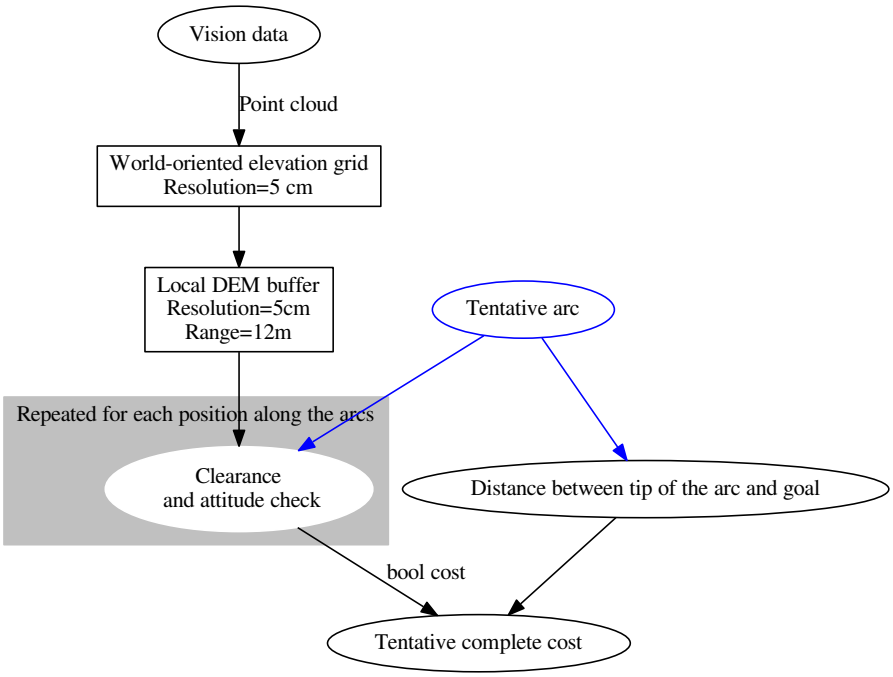


Figure 12: Flowchart for the computation of the cost of a tentative action

Definition 6. Let $p = (x, y, \theta) \in P$ be a pose and $m \in M$ a motion.
$p \cdot m$ is defined as the pose obtained by moving the rover along the arc m from the pose p.

Figure 11 uses these definitions and provides an example of a configuration tree.

5.2.3 Algorithm

At each planning step, we consider the following set of $11 \times 11 \times 25$ possible actions:

$$T = TIP([-3:3:25]) \times A(4, [-0.5:0.5:11]) \times A(4, [-0.5:0.5:11])$$

For each action, we compute a cost (see Section 5.2.4), then choose the action with the lowest cost.
If the chosen action is $a = TIP(r) \cdot A(4, l_1) \cdot A(4, l_2)$, we only send the command $TIP(r) \cdot A(2, \frac{l_1}{2})$ to the actuators. Indeed, the expected execution error does not allow us to drive eight meters blind. By stopping early and re-evaluating our options, we are also able to adapt our strategy to obstacles as soon as they are six meters away from the rover.

5.2.4 Cost function

Given the current pose p, the cost of a candidate action a is:

$$C(p, a) = C_e(a) + C_k(p, a) + C_t(p, a)$$

Where:

- $C_e(a)$ is the *execution* cost, which is the expected time (in seconds) to turn the wheels, drive, turn the wheels again, etc.
- $C_k(p, a)$ is the *kinematic* cost. It is computed using the metrics provided by the kinematic state estimation algorithm that is executed every 25 centimeters along the planned arcs. This is either ∞ if vehicle safety is not guaranteed, or 0 if vehicle safety is guaranteed.
- $C_t(p, a)$ is the *terminal* cost. It is the estimated time required to drive from the end of the candidate action $p \cdot a$ to the global goal. The nature of this cost is further discussed in section 5.3.

This cost function is summarized in Figure 12.

5.3 Global planning

5.3.1 Motivation

As discussed previously, it is neither desirable or feasible to keep a heightmap of all the terrain we traversed. This means that inevitably, we are going to forget the exact geometry of the terrain when we go further than ten meters from it.

In simple terrain, this is not a problem because the path of the rover from the start to the goal is mostly straight. The rover never has to come back to the same area after going away further than ten meters. In these cases, at position p, the terminal cost of a candidate action a can simply be defined as the euclidean distance between the tip of the action $p \cdot a$ and the global goal.

However, some Martian geological features are concave and simply forgetting everything about the past terrain may bring us in an infinite loop: the rover keeps going back to the same area without remembering its path was blocked. The likelihood of finding such large-scale concave obstacles on Mars is not known yet, but EnhancedNav has to be able to deal with these. We tried various approached for global planning.

5.3.2 Algorithm

The global planner keeps track of a grid that is 100 meters wide and 100 meters long, and is aligned with a global reference frame. This grid is kept centered around the rover using a 2D rolling buffer. Each cell is 1 meter by 1 meter, and contains a cost based on the terrain under and around the cell.

The cost for each cell is obtained using this procedure:

1. Select all the terrain data in a disk of radius 2 m centered around the center of the cell.
2. Perform a least-squares plane fitting of this data using a RANSAC algorithm.
3. Compute the slope σ of the plane, and the maximum distance ρ between the terrain and the plane (this is called the roughness of the terrain).
4. The cost of the cell is defined as:

$$C = \begin{cases} \alpha\sigma + \beta\rho & \text{if } \sigma \leq \sigma_{max} \text{ and } \rho \leq \rho_{max} \\ \infty & \text{otherwise} \end{cases}$$

   Where α, β, σ_{max} and ρ_{max} are constants. These constants are chosen so that the cost is infinite only if the terrain is extremely rough and is very likely to be non-traversable.

The updated cost function is summarized in Figure 13.

Assuming cost 0 for unknown terrains proved unreliable since the rover kept trying to go into unknown terrain instead of driving forwards.

The chosen solution was to give unknown cells the initial cost $\alpha\sigma_{max} + \beta\rho_{max}$.

As soon as vision data is collected for a cell, a cost is computed as described above, and overwrites any previously stored cost.

The terminal cost $C_t(p, a)$ is defined as the lowest-cost grid-based path from $p \cdot a$ to the global goal. This can be computed using any grid-based path planning algorithm such as A* or Dijkstra.

The chosen implementation uses Dijkstra to compute the terminal costs for each candidate arc simultaneously.

5.3.3 Local cell cost

The use of this global planning method means that we are now adding a terrain-dependent cost. However, if we simply summed this cost along the best global, medium slope areas for instance, would be assigned a non-zero cost as long as they are far enough to be in the terminal path, but this cost would stop being considered once it reaches the tree planning region. To avoid that, we change the cost of a candidate action as defined in 5.2.4 from

$$C(p, a) = C_e(a) + C_k(p, a) + C_t(p, a)$$

to

$$C(p, a) = C_e(a) + C_k(p, a) + C_t(p, a) + C_{lcc}(p, a)$$

Where:

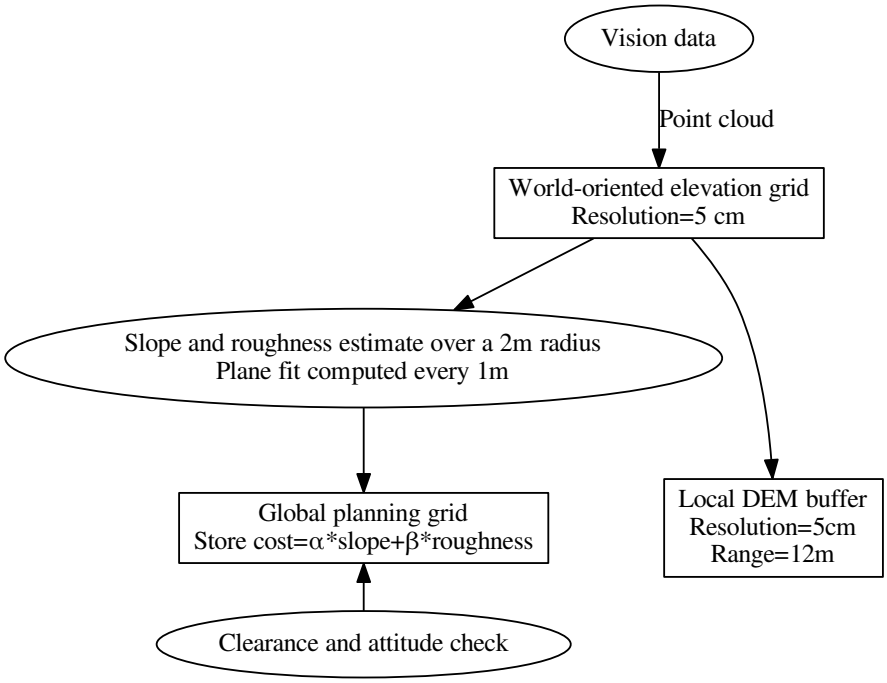


Figure 13: Flowchart for the computation of the cost of a global grid cell

- $C_t(p, a)$ = sum of the costs of each cell on the optimal path from $p \cdot a$ to the global goal.
- $C_{lcc}(p, a)$ is the *local cell cost*, which is the sum of the costs of the global planner cells that are under the path defining the action a.

This keeps the local and global planner's costs consistent. The updated cost function is summarized in Figure 14.

5.3.4 Cross-registration

Another possible inconsistency between the local and global planner arises when the kinematic evaluation deems an action a unsafe ($C_k(p, a) = \infty$), but the cell cost of the corresponding terrain is finite ($C_{lcc}(p, a) < \infty$). The effect of this inconsistency is that the global planner will keep the rover away from high slope or high roughness areas, but when the global planner decides this is still the best path (because unknown terrain is even higher cost, and the kinematic approximation decides this path is blocked by obstacles, the global planner will keep bringing the rover back to this location.

The way we fixed this inconsistency is by flagging any cell containing a pose that threatens vehicle safety as "red". Its cost becomes infinite when computing C_t, but stays the same when computing C_{lcc}.

Red cells don't need to lead to an infinite local cell cost (LCC) because if they are close enough to the rover not to appear in the terminal cost C_t, the corresponding vehicle safety will be re-evaluated by the tree planning algorithm anyways. Assuming that any path that goes through the same 1 meter cell as an unsafe pose is unsafe would negate the benefits of using a kinematic approach to vehicle safety evaluation.

5.4 Lazy evaluation

We remind the cost of a candidate arc a when the rover is at position p:

$$C(p, a) = C_e(a) + C_k(p, a) + C_t(p, a) + C_{lcc}(p, a)$$

The most computationally expensive cost to compute is $C_k(p, a)$ since it involves performing the approximate kinematic settling algorithm every 25 centimeters along the action path, which is about 8 meters long. $C_t(p, a)$ has no marginal cost since we obtain $C_t(p, a)$ for every a simultaneously by using Dijkstra.

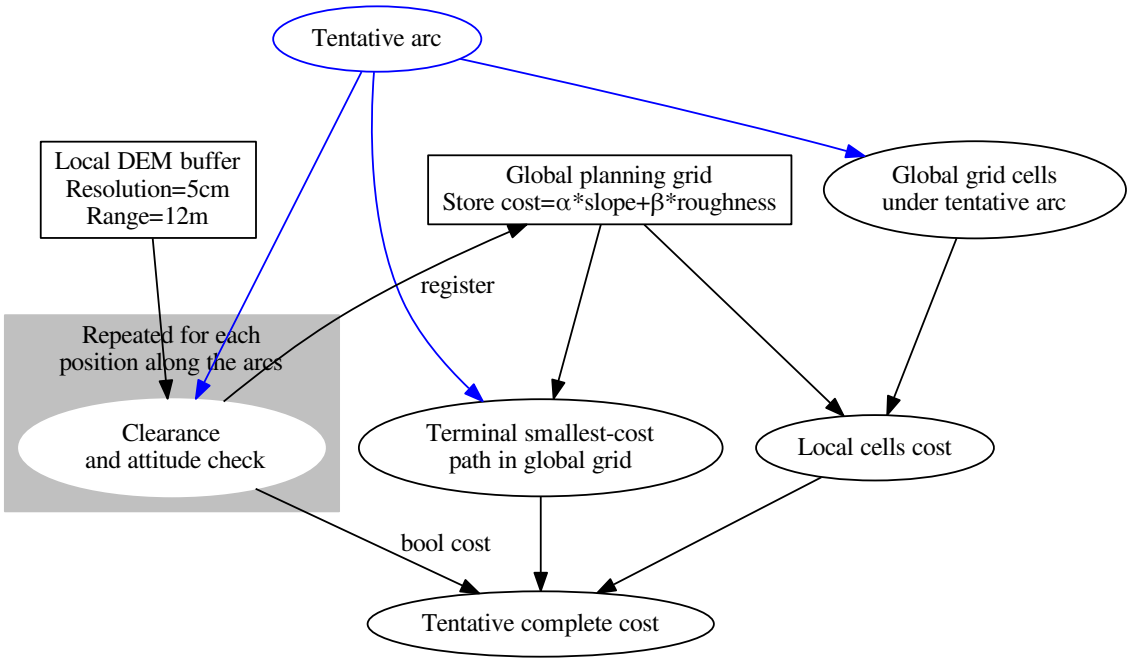


Figure 14: Flowchart for the computation of the cost of a tentative action (updated to include global grid-based planning)

This means that we only need to compute C_e, C_t and C_{lcc} for every available action, then sort the actions based on the associated $C_e + C_t + C_{lcc}$, and compute C_k for each one in increasing order. As soon as we find an arc that is deemed safe by the kinematic approximation metrics, we know it has the lowest $C(p, a)$.

6 Results

6.1 Monte-carlo testing of the kinematic state approximation algorithm

Figure 16 shows the results of Monte-carlo testing the approximation algorithm against a ground-truth oracle. The oracle is a Matlab program that performs the full terrain settling of the rover by using a precise kinematic model of the rover, and the built-in optimization toolbox of Matlab to find the configuration (z position, body rotation and rocker-bogie angles) that minimizes the distance between the skin of the wheels and the terrain. This objective function's domain is $\mathbb{R}^6$ and its image is in $\mathbb{R}^6$.

6.2 Traversal of simple terrains

We call *simple terrains* terrains that don't have any large-scale concave obstacles. In our simulations, we generated many such heightmaps using the CFA model given by [2], and instructed the rover to drive a 60 meters straight line through these terrains (an example drive is shown in Figure 17).

We also developed an analog of the Gestalt algorithm presented in Section 4.1 to compare the new algorithm with. Figure 18 presents the results. The most challenging landing sites proposed for Mars2020 are about 15% CFA, and the initial requirements for the new algorithm were to succeed 90% of the time on 15% CFA terrains. This requirement is fulfilled by our algorithm, at least in simulations.

6.3 Results on complex terrains

The global planning part of our algorithm is most useful when large-scale concave obstacles are present, as in the terrain presented in Figure 19 for instance.

Configuring the global planner required a lot of iterations, and we needed a method of keeping track of the different versions to limit regression and keep a global view of performance on various terrains.

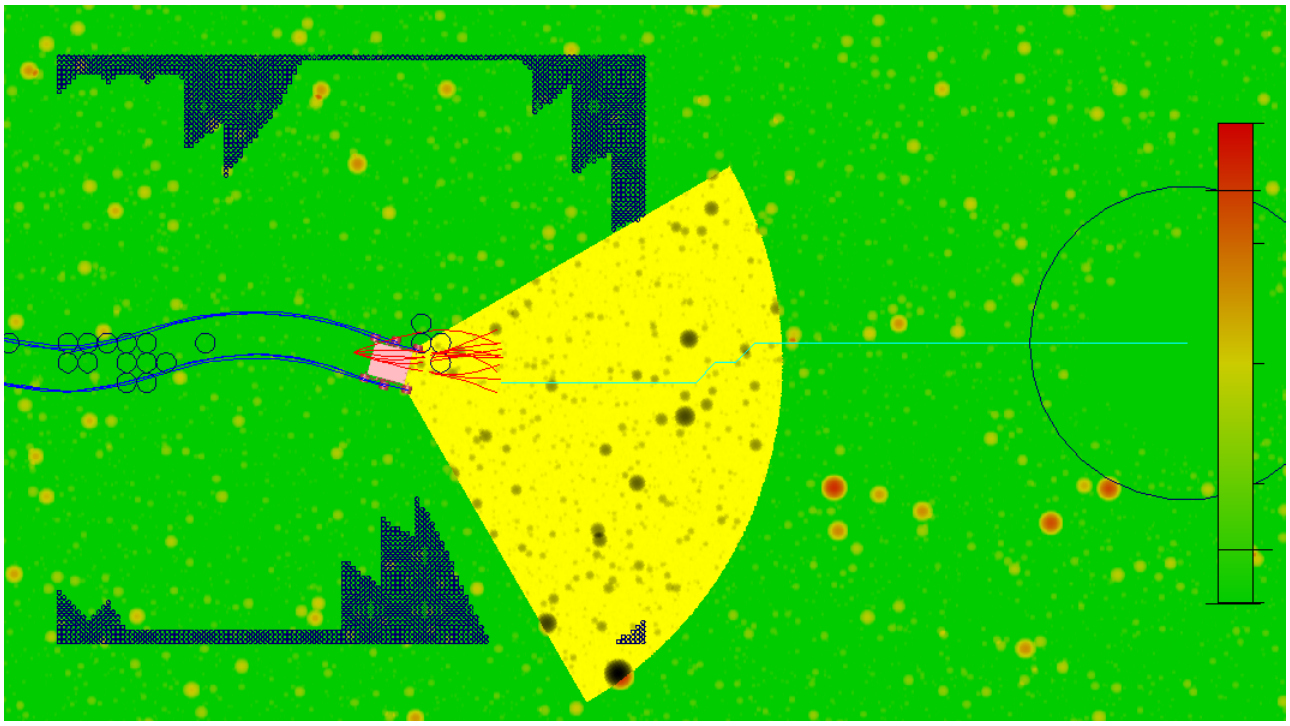


Figure 15: View of all the components of the proposed path planner. The blue areas is the unknown terrain stored in the local heightmap buffer. The small blue circles are the red cells in the global grid. The turquoise path is the terminal path corresponding to the selected action. The red arcs are the actions for which the kinematic safety was evaluated.

A set of 64 heightmaps were generated by Z-scaling and adding various rock densities to 6 base terrains (presented in Figure 20).

Each version of the algorithm was tested against each of the 64 terrains, as presented in Figure 21.

The latest version was able to navigate very challenging terrains successfully (see Figure 22).

7 Remaining work

These results convinced the Mars2020 project leadership to unlock about $2 million to further develop and test this algorithm. A lot of work remains to be done, such as handling terrain occlusions, and modeling slip and actuation errors.

8 Acknowledgements

I would like to thank Olivier Toupet for inviting me to JPL and mentoring me during my internship. His advice was very valuable, and he was always very available and open to listen to my crazy ideas.

I would also like to thank Michael McHenry, Ono Masahiro, Richard Rieber and all the Mars2020 team for engaging in technical discussions and valuing my ideas.

I also express my special thanks to Isabelle Delais, who went to great lengths to get me through the paperwork and funding issues for this internship.

Finally, I thank Jean-Pierre Lhomond, Richard Volpe and Marc Pouzet without whom I would never have been in contact with JPL.

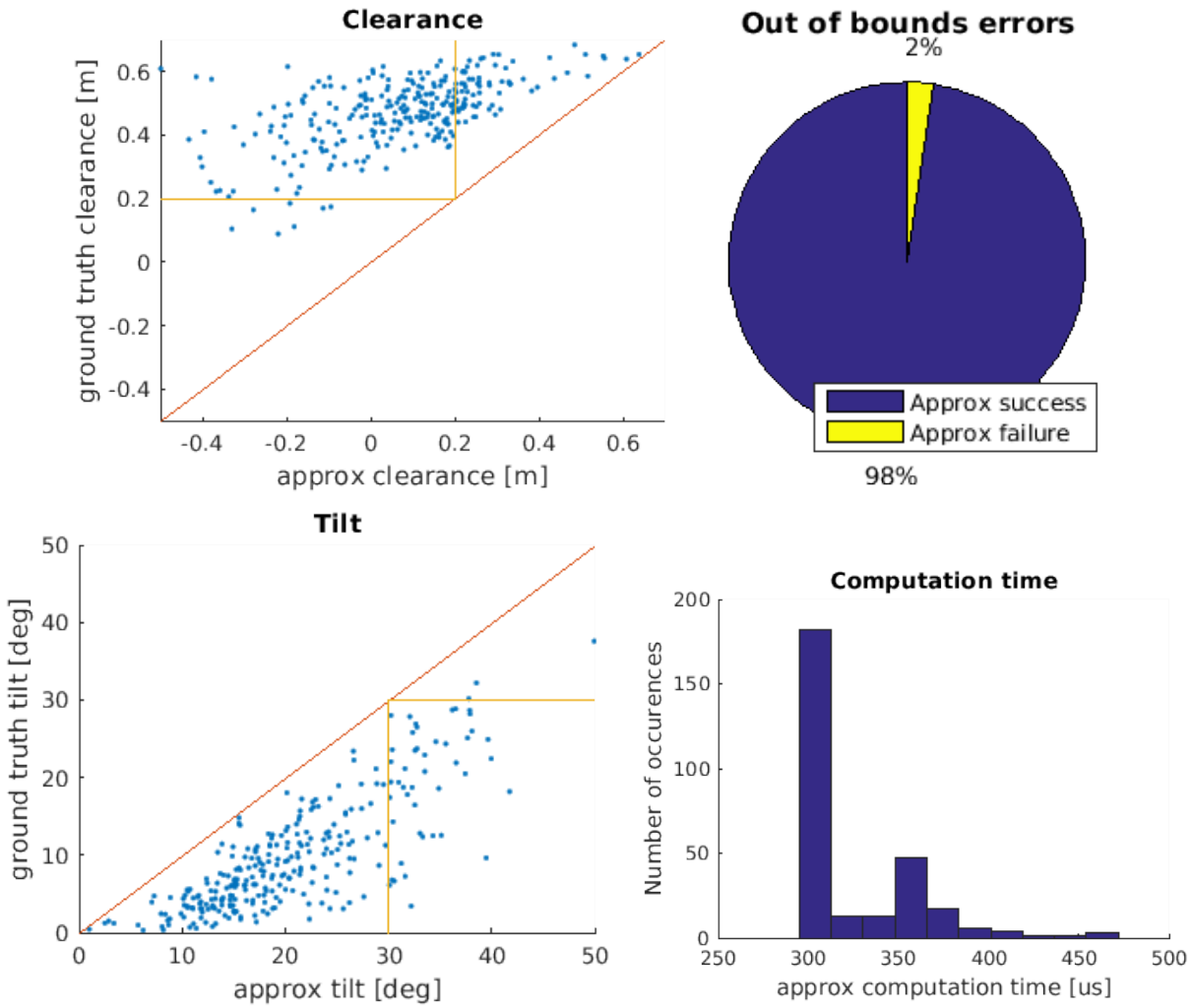


Figure 16: Summary screen of the Monte-carlo testing of the kinematic state approximation algorithm. On a fixed terrain, many (x, y, θ) configurations are generated, and the clearance and tilt of the approximation are compared to a ground-truth oracle

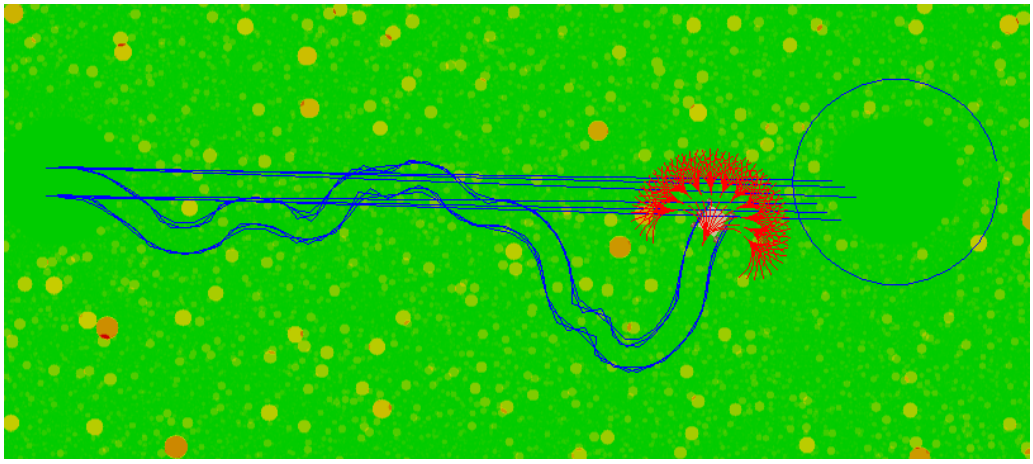


Figure 17: Rover navigating a *simple terrain* in the simulation environment (the straight blue lines are a graphical glitch).

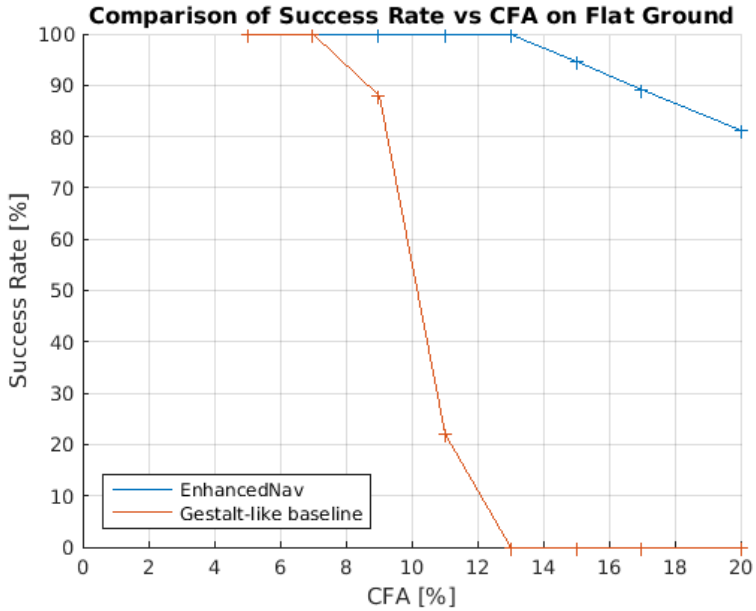


Figure 18: Performance comparison of Gestalt (former algorithm) and EnhancedNav (proposed algorithm) on simple terrains. For each CFA test value, each algorithm was run on several procedurally-generated terrains. The fraction of terrains on which the rover was able to complete the drive is shown on the vertical axis.

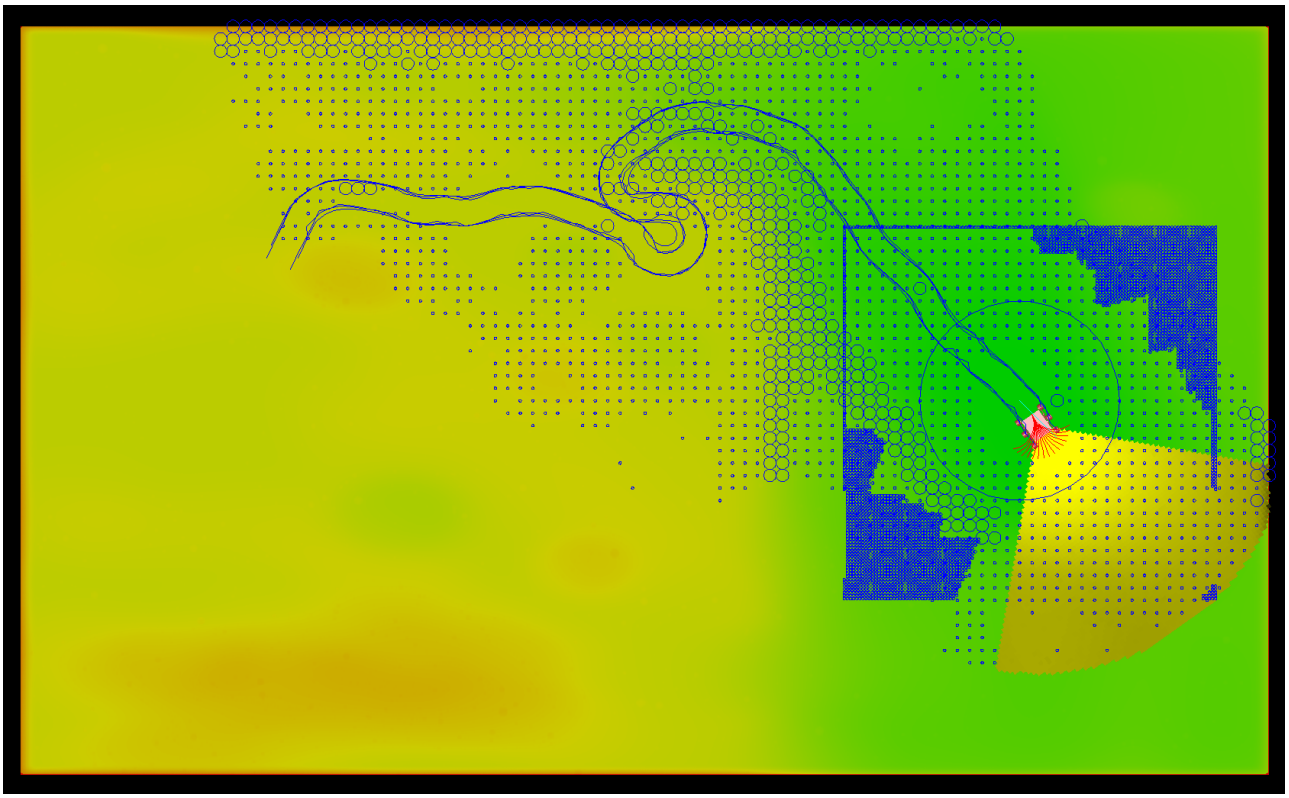
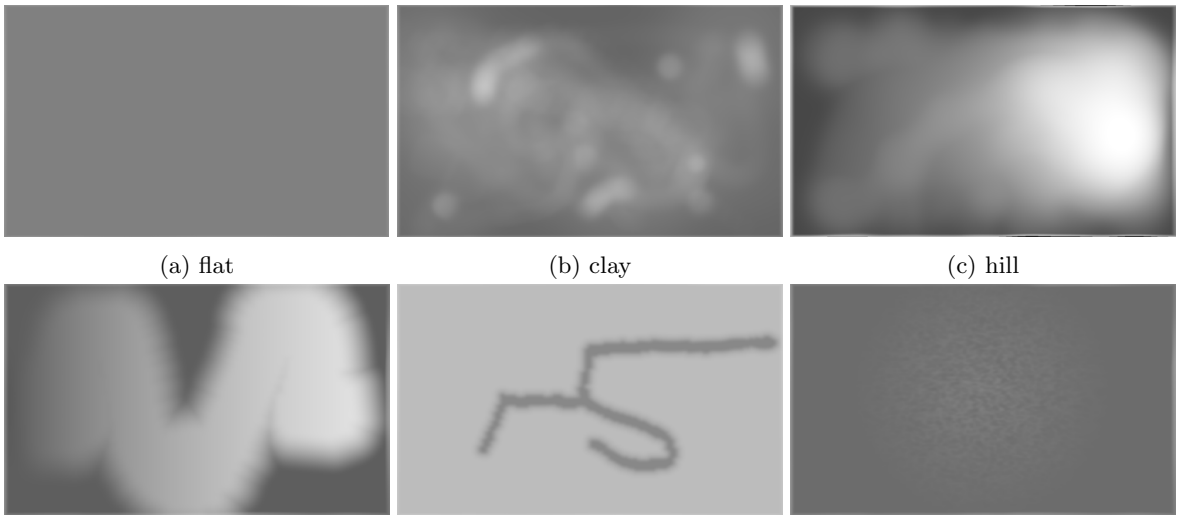


Figure 19: Complex terrain loaded in the simulation environment. Notice how the rover has to turn around because the slope was too steep at the bottom of the terrain, and take the ramp at the top instead.



(a) flat (b) clay (c) hill

(d) ladder (e) maze (f) random

Figure 20: Root terrains used to test the global planning capabilities of our algorithm.

Test case	Baseline	Depth_3	Small_arcs	increase_ts_on_fail	no_wheel_drop	no_global_planning	smaller_arcs_depth_3	smaller_arcs_depth_3_no_global	smart_tip	lazy	smart_tip2	tip_then_8m	tip_last_resort	backwards	tip_then_arc	tip_straight_arc	tip_arc_arc
bumps_03_05.csv	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
bumps_03_10.csv	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
bumps_03_15.csv	0	0	0	0	0	0	0	1	0	1	1	0	1	1	1	0	1
bumps_03_20.csv	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
bumps_05_05.csv	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
bumps_05_10.csv	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
bumps_05_15.csv	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	1
bumps_05_20.csv	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bumps_10_05.csv	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
bumps_10_10.csv	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
bumps_10_15.csv	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
bumps_10_20.csv	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
clay_03_05.csv	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
clay_03_10.csv	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
clay_03_15.csv	0	1	1	1	1	1	1	1	1	0	1	0	1	1	1	0	1
clay_03_20.csv	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
clay_05_05.csv	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
clay_05_10.csv	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1
clay_05_15.csv	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1
clay_05_20.csv	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
clay_10_05.csv	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
clay_10_10.csv	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	1	1
clay_10_15.csv	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
clay_10_20.csv	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
flat_05.csv	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
flat_10.csv	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
flat_15.csv	1	0	0	1	1	1	1	0	1	0	1	1	1	0	1	1	1
flat_20.csv	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
hill_03_05.csv	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
hill_03_10.csv	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
hill_03_15.csv	1	1	0	1	1	1	1	1	1	1	1	0	0	1	1	1	1
hill_03_20.csv	0	0	0	0	1	0	0	0	0	0	1	0	0	0	1	1	0
hill_05_05.csv	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
hill_05_10.csv	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
hill_05_15.csv	1	0	1	1	0	1	1	1	1	1	1	1	0	1	1	1	1
hill_05_20.csv	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0
hill_10_05.csv	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
hill_10_10.csv	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
hill_10_15.csv	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1
hill_10_20.csv	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
ladder_03_05.csv	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ladder_03_10.csv	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1
ladder_03_15.csv	0	1	0	0	1	0	0	1	0	0	1	0	1	1	1	1	1
ladder_03_20.csv	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
ladder_05_05.csv	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ladder_05_10.csv	0	1	0	1	1	1	0	1	1	0	1	0	1	1	1	1	1
ladder_05_15.csv	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
ladder_05_20.csv	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ladder_10_05.csv	1	1	1	1	1	0	1	1	0	1	1	1	1	1	1	1	1
ladder_10_10.csv	1	1	0	1	0	0	1	1	0	0	0	0	1	0	1	1	1
ladder_10_15.csv	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ladder_10_20.csv	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
maze_03_05.csv	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	1	1
maze_03_10.csv	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0
maze_03_15.csv	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
maze_03_20.csv	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
maze_05_05.csv	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	1	1
maze_05_10.csv	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0
maze_05_15.csv	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
maze_05_20.csv	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
maze_10_05.csv	0	0	0	0	0	0	0	0	0	2	1	1	0	1	1	0	1
maze_10_10.csv	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1
maze_10_15.csv	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
maze_10_20.csv	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Baseline	Depth_3	Small_arcs	increase_ts_on_fail	no_wheel_drop	no_global_planning	smaller_arcs_depth_3	smaller_arcs_depth_3_no_global	smart_tip	lazy	smart_tip2	tip_then_8m	tip_last_resort	backwards	tip_then_arc	tip_straight_arc	tip_arc_arc
	24	27	21	26	27	23	26	27	28	32	30	27	30	34	40	34	48

Figure 21: Test matrix for global planning. Each column is a version of our algorithm, and each line is a test terrain.

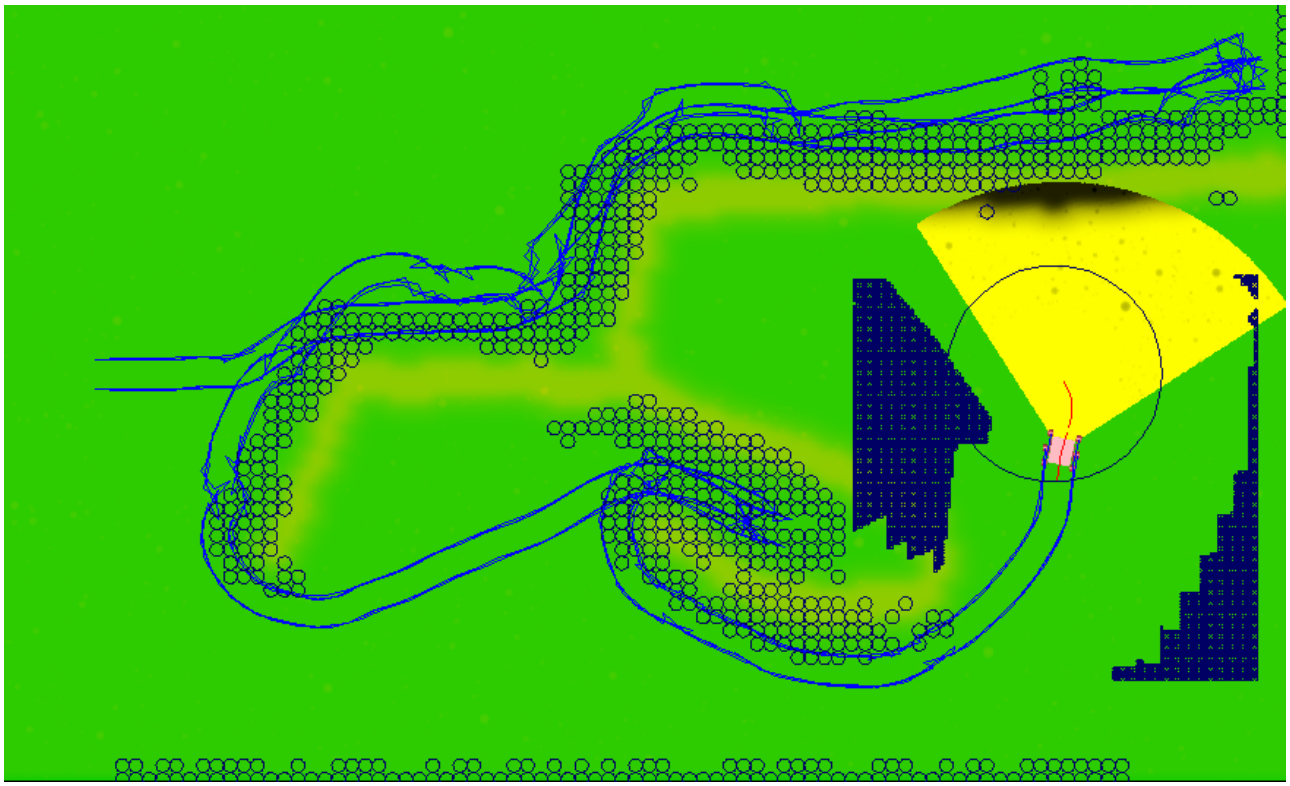


Figure 22: Successful path on the *maze* terrain.

9 Bibliography

References

[1] S. B. Goldberg, M. W. Maimone, and L. Matthies. Stereo vision and rover navigation software for planetary exploration. In *Aerospace Conference Proceedings, 2002. IEEE*, volume 5, pages 5–2025–5–2036 vol.5, 2002.

[2] M. P. Golombek, A. F. C. Haldemann, N. K. Forsberg-Taylor, E. N. DiMaggio, R. D. Schroeder, B. M. Jakosky, M. T. Mellon, and J. R. Matijevic. Rock size-frequency distributions on Mars and implications for Mars Exploration Rover landing safety and operations. *Journal of Geophysical Research: Planets*, 108(E12):n/a–n/a, 2003. 8086.

[3] Thomas Howard, Ross Alan Knepper, and Alonzo Kelly . Constrained optimization path following of wheeled robots in natural terrain. In *Proceedings of the 10th International Symposium on Experimental Robotics 2006 (ISER '06)*, July 2006.

[4] Mikhail Pivtoraiko. *Differentially Constrained Motion Planning with State Lattice Motion Primitives*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, February 2012.

[5] Mikhail Pivtoraiko, Thomas Howard, Issa Nesnas, and Alonzo 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.

[6] Mikhail Pivtoraiko, Ross Alan Knepper, and Alonzo Kelly . Optimal, smooth, nonholonomic mobile robot motion planning in state lattices. Technical Report CMU-RI-TR-07-15, Robotics Institute, Pittsburgh, PA, May 2007.