Space Robotic Systems - Curiosity Rover

Valentina Piccione 2030930

February 10, 2024

Abstract

This project report presents a comprehensive study on the path planning and localization of a Mars Rover, specifically modeled after NASA's Curiosity Rover. This study encompasses three major tasks: Navigation, Path Planning, and Rover Localization. In the Navigation task, the rover is simulated to traverse from an initial to a final pose, avoiding steep slopes, with a focus on its trajectory, velocity, heading angle, and rate of change of the heading angle. The Path Planning task involves implementing the A* algorithm for minimum distance path finding in an 8-way grid environment, while avoiding obstacles represented as steep slopes. Finally, the Rover Localization task examines the accuracy of an onboard autonomous localization system using dead reckoning and an Extended Kalman Filter combining odometer and LIDAR data. This report details the methodologies, simulations, and analysis conducted for these tasks, highlighting the challenges and efficiency of the algorithms used in the complex and variable terrain of Mars.

Contents

1 Introduction

1.1 Project structure

The project has been coded in Python, and follows this structure:

- The project **main** is composed by a Jupiter notebook, in which the functions created to perform the requested tasks are called.
- All the functions are stored in the **Library** folder as simple scripts
- the **Data** folder collect all images of the outputs of the tasks, shown also in this report.

1.2 Environment

The environment in which the rover operates is the Gale Crater on Mars. The project encapsulates it within a 45×30 km^2 area, and is digitally represented through three distinct grayscale images:

- The *operational environment* map: representation of the landscape;
- The *obstacles* map: a 2D matrix containing values whose range varies from 0 to 255, and where obstacles are represented as white pixels;
- The *landmarks* map: augmented version of the operational environment including the locations of the landmarks used for localization purposes.

Each image has a resolution of $10m/px$, meaning that the actual dimensions of the 2D matrices is of 4500×3000 pixels.

1.3 Rover's model

The Curiosity rover is a robotic explorer navigating Gale Crater on Mars. The project's design and implementation take in consideration distinct characteristics of the rover. Key specifications considered include:

- $v_{max} = 4$ cm/s: is the maximum velocity that the rover can achieve;
- $L = 3$ m: is the axles distance of the rover;

The rover is modeled as a *car-like vehicle*. Its configuration is expressed in a 2D environment through the generalized coordinates (x, y, θ), utilizing a bicycle model for motion simulation. The equations of motion of the rover are:

$$\begin{cases} \dot{x} = v \cos\theta \\ \dot{y} = v \sin\theta \\ \dot{\theta} = \frac{v}{L} \tan\gamma \end{cases} \tag{1.1}$$

where v is the velocity, L is the axle distance and γ is the steering angle.
Three positions for the rover have been specified in the environment, representing start and goal positions for the three different tasks required:

- $P_0 = (x_0, y_0, \theta_0) = (42380, 11590, \pi/2)$ (m, m, rad) marked in light green in Figure 1
- $P_1 = (x_1, y_1, \theta_1) = (33070, 19010, \pi)$ (m, m, rad) marked in blue in Figure 1
- $P_2 = (x_2, y_2) = (10870, 25670)$ (m, m) marked in red in Figure 1

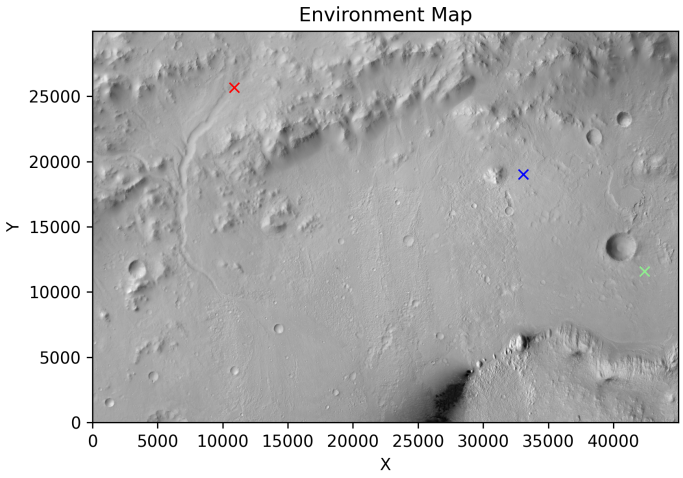


Figure 1: Environment map with the 3 positions

2 Task 1 - Navigation

The first task of the project is to guide the Curiosity rover from its starting point P_0 to the target location P_1, while avoiding steep slopes regions.This task entails determining the rover's path, velocity over time, heading angle over time, and its rate of change. Additionally, an important part of the task is to calculate the time it will take for the rover to arrive at its destination. The goal is then to maneuver the rover through the challenging Martian terrain represented in the environment map and in the obstacle map, carefully navigating around obstacle areas.

2.1 Move to a pose

In order to perform this task the *Moving to a Pose* control has been implemented. This requires the conversion of the pose state variables (x, y, θ) into the polar state variables (ρ, α, β).

Considering the points $P_0 = (x_0, y_0, \theta_0)$ and $P_1 = (x_1, y_1, \theta_1)$, the conversion is performed through the following formulas:
Computation of the relative position:

$$\Delta x = x_1 - x_0 \tag{2.1}$$
$$\Delta y = y_1 - y_0 \tag{2.2}$$

Polar coordinate transformation:

$$\rho = \sqrt{\Delta x^2 + \Delta y^2} \tag{2.3}$$
$$\alpha = \arctan 2(\Delta y, \Delta x) - \theta_0 \tag{2.4}$$
$$\beta = -\alpha - \theta_0 + \theta_1 \tag{2.5}$$

Retrieval of a new set of kinematic equations:

$$\begin{cases} \dot{\rho} = -\cos(\alpha) v \\ \dot{\alpha} = \frac{\sin(\alpha) v}{\rho} - \omega \\ \dot{\beta} = -\frac{\sin(\alpha) v}{\rho} \end{cases} \tag{2.6}$$

The computation of the values of v and ω is performed though a linear control law:

$$v = K_\rho \rho \tag{2.7}$$
$$\omega = K_\alpha \alpha + K_\beta \beta \tag{2.8}$$

with the values of the gains K_ρ, K_α and K_β chosen experimentally as:

$$K = \begin{bmatrix} K_\rho \\ K_\alpha \\ K_\beta \end{bmatrix} = \begin{bmatrix} 1.3 \cdot 10^{-3} \\ 1.5 \cdot 10^{-3} \\ -1.9 \cdot 10^{-3} \end{bmatrix}$$

These values allows a stable control, since $K_\rho > 0$ and $K_\beta < 0$ and $K_\alpha - K_\rho > 0$.
In the end the computation of the trajectory that the rover must follow is performed solving a system of differential equations; then each trajectory point is converted back to generalized coordinates (x, y, θ) and checked for the possibility of collision with obstacles.

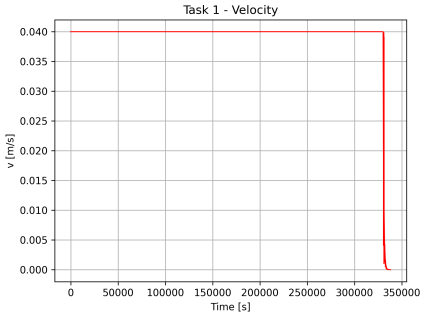
2.2 Results

The obtained results of computing the trajectory for the given start and goal positions, P_0 and P_1, with a sampling frequency of 0.1 Hz, are the following:

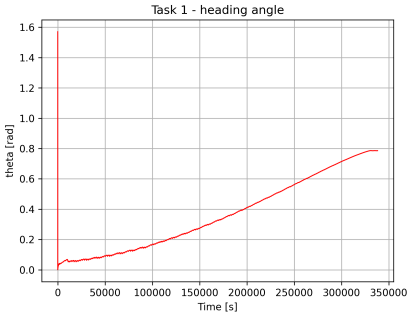
- the retrieved trajectory is a $[33134 \times 3]$ array that avoids steep slope regions
- initial state $P_0 = [42380, 11590, \pi/2]$
- final state $P_{final} = [33070.001, 19010.000, \pi]$
- the error between the desired state P_1 and the final state P_{final}, with the chosen gains, is $error = [0.001, 0.000, 0.000]$

- the rover takes 338190.000 seconds to reach the goal, that are equivalent to 3.914 days

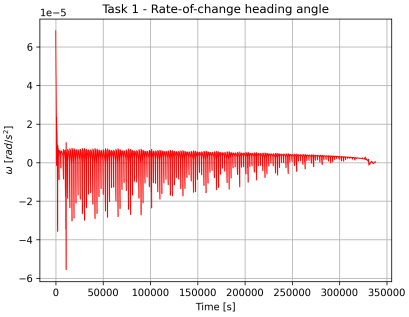
The velocity v of the rover, the heading angle θ, and its rate of change over time ω are plotted in Figure 2. The velocity in figure 2a is constant at 0.04 m/s for the majority of the time period. Towards the end of the time period, there is a sharp vertical line where the velocity drops to zero almost instantaneously, indicating that the rover came to an abrupt stop.



(a) v



(b) θ



(c) ω

Figure 2: v, θ and ω

The graph of the heading angle, image 2b, starts at a theta angle of $\pi/2$ *rad*, which indicates that the rover was initially oriented perpendicular to the reference direction. There is an immediate decrease from this peak, which suggests that the heading angle was adjusted sharply from the initial value to a lower angle shortly after the start. After this initial adjustment, the heading angle increases at a relatively steady rate, indicating a gradual turning or change in orientation over time.
Regarding the rate of change of the heading angle, visible in Figure 2c, the spikes present in the graph are relatively low since the scale of the vertical axis is of the order of 10^{-5}. In addiction to an initial spike, there are a series of oscillations that diminish over time. This pattern is typical of damped oscillations, where the amplitude decreases over time due to some form of resistance or damping force. As time progresses, the amplitude of the oscillations decreases and the angular velocity appears to stabilize around a value that

oscillates around 0 rad/s^2. Towards the end of the graph, the angular velocity still shows minor fluctuations but remains close to zero. This decreasing amplitude of oscillations over time is indicative of the system becoming more stable in its heading angle as time progresses.

The final trajectory successfully avoids the obstacles present between the two positions. It coasts near the borders of inaccessible regions in a single, smooth maneuver, without the need of intermediate via points, as can be seen in Figure 3 and Figure 4.

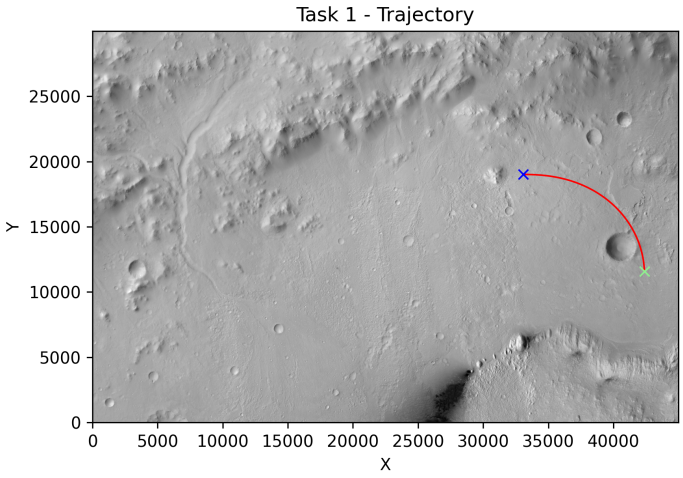


Figure 3: The trajectory of task 1 in the environment map

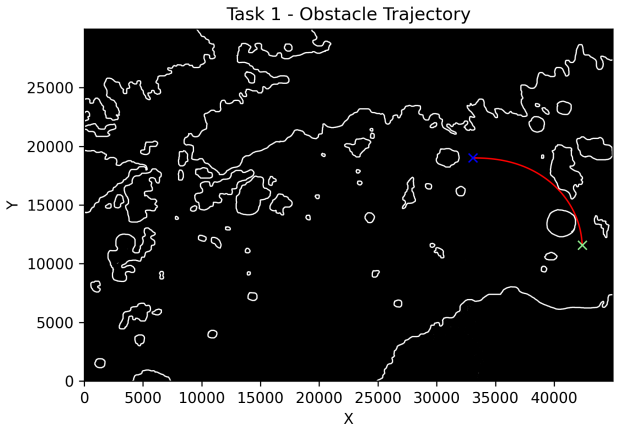


Figure 4: The trajectory of task 1 in the obstacle map

3 Task 2 - Path Planning

After completing the first task, the rover is required to travel from the retrieved goal position of the previous task P_{final}, that will be addressed as P_{1r} and considering only the x and y coordinates of the location, to a new target position P_2, still avoiding obstacles region in the area, with the use of the $A^\star$ algorithm. The map used for the task is the obstacle map in Figure 5.

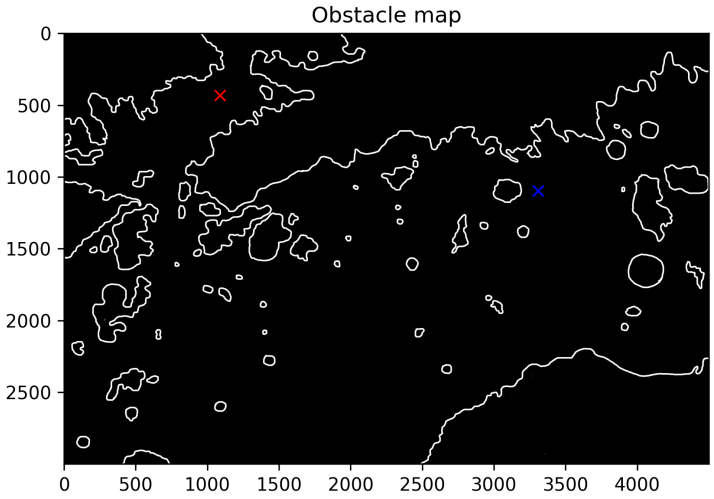


Figure 5: Obstacle map with the 2 positions

In order to complete this task, the start and goal position must be converted into indices, or pixel coordinates, since $A^\star$ uses pixels to find a path. This conversion into pixel coordinates is performed as:

$$j = \left\lfloor \frac{\text{point}[0] - X[0,0] + \text{mapRes}}{\text{mapRes}} \right\rfloor - 1 \tag{3.1}$$

$$i = \left\lfloor \frac{Y[0,0] - \text{point}[1] + \text{mapRes}}{\text{mapRes}} \right\rfloor - 1 \tag{3.2}$$

where j is the column index and i is the row index. The results are rounded down to the nearest integer.
The result of this operation in the two positions is:

$$P_{1r} = [1097, 3307]$$

$$P_2 = [432, 1087]$$

3.1 A* algorithm

$A^\star$ is one of the most common path-finding algorithms. In this paragraph it will be explained the main characteristics of the actual implementation of the algorithm in its *without reopening* variant, with the usage of Octile distance heuristic.
The goal of the algorithm is to find a path from a start point to an end point on a map with obstacles. It creates a `Node` class to represent each point on the map and storing its position, the cost from the start g, the heuristic estimate to the end h, and the total estimated cost f.
The code efficiently explores paths towards the goal by maintaining two lists: the *open list*, containing the nodes to be explored, and the *closed set*, containing the nodes already explored. It starts with the start node in the open list, and in each iteration, picks the node with the lowest f value from the open list, indicating potential to be the shortest path.
Inside the $A^\star$ function there is a `process_node` function that checks the neighbors of the current node, calculating g, h, and f values. If a neighbor is a valid candidate for exploration, not an obstacle, within map bounds, and not already processed, it's considered for addition to the open list. If the end node is reached, it constructs and returns the path by tracing back from the end node to the start node. The path is then returned in reverse order, starting from the start node to the end node. If no path is found, it returns `None`.

The chosen heuristic function combines the maximum and minimum differences in coordinates between two nodes, applying a constant to these differences.

$$dx = |a_x - b_x| \qquad dy = |a_y - b_y|$$

$$heuristic(n) = D \times \max(dx + dy) + (D_2 - D) \times \min(dx, dy) \tag{3.3}$$

Considering the constants $D = 1$ and $D2 = \sqrt{2}$.
This constant applied to the minimum difference, $\sqrt{2} - 1$, is used to approximate the cost of diagonal movement in a grid where moving horizontally or vertically costs 1 unit, and moving diagonally is slightly more expensive, assuming the diagonal cost is $\sqrt{2}$.

In $A^\star$, the heuristic function should never overestimate the cost to reach the goal. Using $\sqrt{2}-1$ as the multiplier for the minimum axis difference balances the cost between straight and diagonal moves, keeping the heuristic optimistic and ensuring the admissibility. This approach helps the algorithm to efficiently navigate the grid, finding a path that closely approximates the shortest possible route while considering diagonal movements.

$$heuristic(n) = \max(dx, dy) + 0.41421356237 \min(dx, dy) \tag{3.4}$$

3.2 Results

The results of applying the $A^\star$ algorithm to the path finding problem demonstrate its effectiveness in navigating complex terrains. Using the heuristic in Equation 3.4 the algorithm took approximately 45 minutes, performing about 2754205 steps to find a solution. The retrieved path is composed by 3074 nodes, and appear to efficiently connect the start

and end points, taking into account the need to circumvent obstacles, but also appears to be the shortest path.

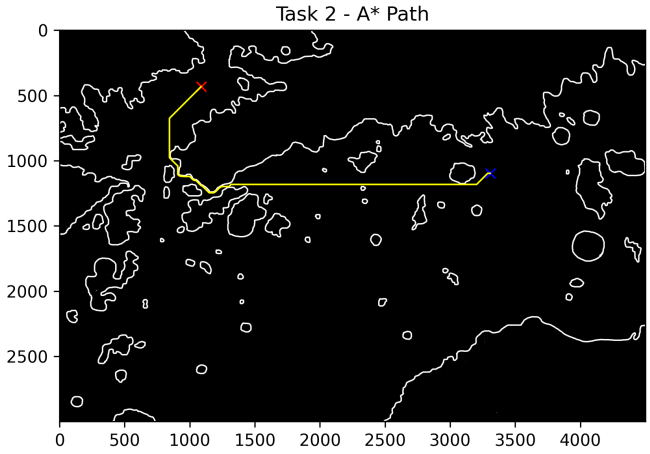


Figure 6: $A^\star$ path

4 Task 3 - Rover Localization

The last task consists in enhancing the reconstruction of the rover's path using data from both its onboard odometer and LIDAR. The task is in fact comprised of two subtasks: first reconstruct the path using only odometry data from the autonomous localization system on board with the *dead reckoning* method, then enhance this reconstruction combining the odometry data with LIDAR terrain mapper data, implementing the *Extended Kalman Filter*.

4.1 Dead Reckoning

For the first part the dead reckoning method has been implemented. With this method, the rover state vector with its associated uncertainty is retrieved and continuously updated as the rover moves through its environment. The associated uncertainty represents the confidence level in the state estimation, expressed through a covariance matrix in probabilistic terms. The considered trajectory is the one from the first task, so the starting and goal position are again $P_0 = (42380, 11590, \pi/2)$ and $P_1 = (33070, 19010, \pi)$.

As the rover travels, sensors provide data on its displacement and change in orientation. The dead reckoning algorithm uses this sensor data to predict the rover's new state. Considering then the discrete-time model of the new configuration of the vehicle as a function of the configuration at the previous step and the odometry measurements of the

distance and heading angle change, the equations are:

$$\begin{cases} x_{k+1} = x_k + \delta_d \cos\theta_k \\ y_{k+1} = y_k + \delta_d \sin\theta_k \\ \theta_{k+1} = \theta_k + \delta_\theta \end{cases} \tag{4.1}$$

However, due to inherent inaccuracies in sensor measurements, uncertainty in the state estimate increases with time and distance traveled.
The odometer data is, in fact, subject to specific noise constraints:

- noise of $\sigma_d = 4$ mm in the traveled distance
- noise of $\sigma_\theta = 0.05$ deg on the heading angle

that allow to define the covariance matrix of the measurements:

$$V = \begin{bmatrix} \sigma_d^2 & 0 \\ 0 & \sigma_\theta^2 \end{bmatrix} \tag{4.2}$$

The noise can be modeled as a zero-mean multivariate Gaussian distribution. This type of noise, also known as white noise, has a constant power spectral density and its amplitude is normally distributed, meaning that most noise values will be near the mean value, with fewer values spread out towards the extremes. This in the code can be generated by a particular function, `np.random.normal`, included in the `numpy` library, that generates random numbers that follow this distribution. It is a good approximation for many types of random environmental noise.

In this way the output of the odometer is the sum of the measurements δ_d and δ_θ and the error associated to the odometer noise, giving the following equations:

$$\begin{cases} x_{k+1} = x_k + (\delta_d + \eta_d) \cos\theta_k \\ y_{k+1} = y_k + (\delta_d + \eta_d) \sin\theta_k \\ \theta_{k+1} = \theta_k + \delta_\theta + \eta_\theta \end{cases} \tag{4.3}$$

The covariance matrix describes the distribution of uncertainty around the mean value. For the initial conditions is of the form:

$$L_0 = \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{bmatrix} \tag{4.4}$$

with the values $\sigma_x = \sigma_y = 6$ m and $\sigma_\theta = 1$ deg
The recursive update of it, is performed by the formula:

$$L_{k+1} = F_q L_k F_q^T + F_v V F_v^T \tag{4.5}$$

where F_q and F_v are the partial derivatives of the state $q_{k+1} = f(q_k, \eta_k)$ with respect to q and η respectively, computed at $\eta = 0$:

$$F_q = \left.\frac{\partial f}{\partial q}\right|_{\eta=0} = \begin{bmatrix} 1 & 0 & -\delta_d \sin(\theta) \\ 0 & 1 & \delta_d \cos(\theta) \\ 0 & 0 & 1 \end{bmatrix} \qquad F_v = \left.\frac{\partial f}{\partial \eta}\right|_{\eta=0} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \tag{4.6}$$

Initially the covariance matrix is symmetric, but proceeding with the updates it will be no more symmetric. The diagonal elements are the estimated variance associated with the states, while the off-diagonal elements are related to correlation coefficients.

4.2 Results

The results of the method can be seen in Figure 7. The trajectory represented in blue is the one retrieved in the first task, while the one in yellow is the dead reckoning reconstruction.

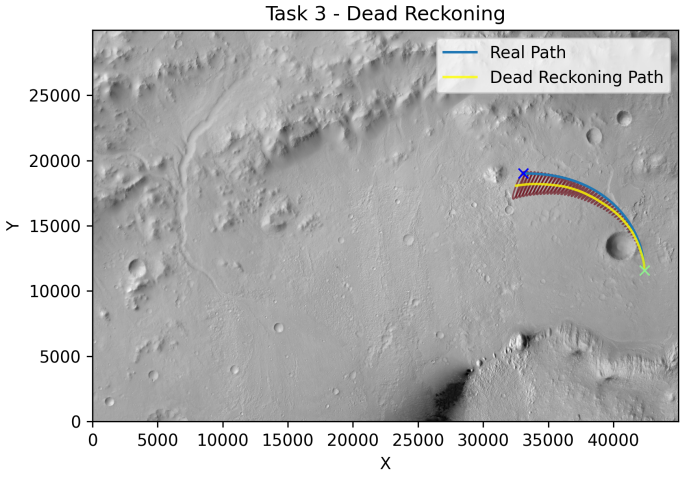


Figure 7: Dead reckoning path with ellipses of uncertainties

Initially, both paths appear to coincide, but as the path progresses, the dead reckoning path deviates from the real path. This divergence indicates that the dead reckoning method is accumulating error over time. In the end of the course in fact the dead reckoning path's end is quite far from the real path's end.

The results suggest that periodic corrections to the dead reckoning path would be necessary to maintain an accurate navigation system. These corrections would likely come from external reference points, which could be physical landmarks, like in the following subsection.

The ellipses of uncertainty are displayed in red, and they represent the uncertainty in the estimated position over time, which increases with each step due to the instrument noise introduced.

The trend of the square-root of the determinant of covariance matrix is shown in Figure 8. This determinant represents the area of the error ellipse in the state-space at a given confidence level. At the beginning, the uncertainty is low and increases very slowly, suggesting that the initial state estimate is quite accurate or the system is in a relatively certain state. As time progresses however, the uncertainty increases at an accelerating rate. The curve's steepening slope indicates that the uncertainty in the state estimate is growing exponentially. This exponential growth reflects the compounding nature of the error in a dead reckoning system, where each new estimate depends on the previous estimate, thereby incorporating all the previous errors. The trend suggests that the longer the system operates without external correction, the less reliable the state estimate becomes, and is consistent with the previous analysis of divergence of the path and the ellipses of uncertainties result.

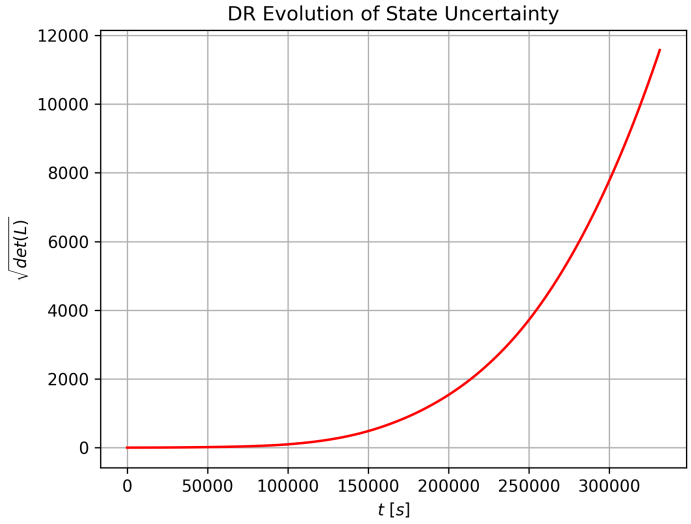


Figure 8: Evolution of the square root of the covariance matrix

4.3 EKF

Dead reckoning method can be improved combining to the odometer measurements, LIDAR terrain mapper data. Doing so significantly enhances the dead reckoning method's accuracy. The odometer provides information about the distance traveled, while LIDAR offers relative distance measurements to known features, that in this case are landmarks located in the space, enabling a more detailed understanding of the rover's environment. Integrating these data sources mitigates the limitations of each sensor type, leading to improved path estimation and reduced uncertainty. In order to perform this second part

of the task, the EKF method was implemented, to allow the system to effectively fuse the information from both the odometer and LIDAR.

The map of the area is then combined to landmark points, displayed in yellow, which the rover is capable of identifying at a maximum range of 500 m thanks to the LIDAR system, with a field of view of 360 deg.

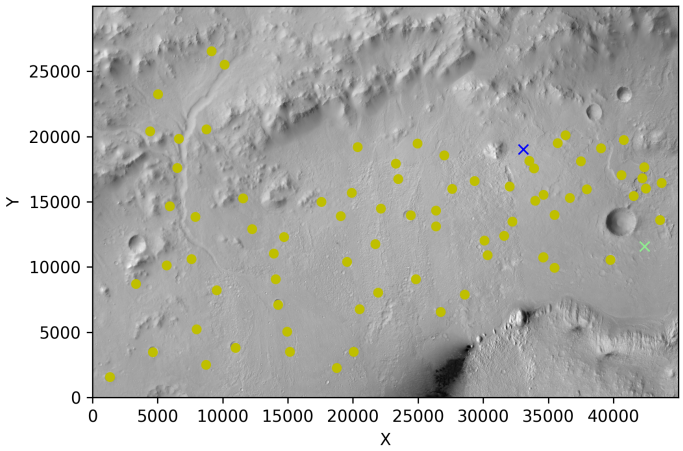


Figure 9: Environment map with the landmarks

The EKF starts with an initial estimate of the state vector and its covariance matrix, representing the system's initial condition and the associated uncertainty, respectively. Being the trajectory the same of the previous part of the task, the system's initial state is again the position P_0, and its associated covariance matrix L_0 as in equation 4.4.

The filter, then, predicts the next state of the system using the current state estimate and the system model, which describes how the state evolves over time. This correspond to have the output of the odometer measurements with also the odometer noise, as previously done in the dead reckoning:

$$\begin{cases} x_{k+1} = x_k + (\delta_d + \eta_d)\cos\theta_k \\ y_{k+1} = y_k + (\delta_d + \eta_d)\sin\theta_k \\ \theta_{k+1} = \theta_k + \delta_\theta + \eta_\theta \end{cases} \tag{4.7}$$

The prediction also updates the estimate's covariance, as before:

$$L_{k+1} = F_q L_k F_q^T + F_v V F_v^T \tag{4.8}$$

When new measurements are received, and so landmarks are visible, the EKF updates the predicted state using a measurement model, that is a mathematical function that describes

how the expected sensor measurements relate to the current state estimate of the system. The distance between the landmark in sight and the rover state vector is given by the h function, where the range, relative distance between landmark and rover, and the bearing angle, orientation difference between the landmark position and the robot's heading, are computed:

$$h(q_{k+1}, p_i) = \begin{bmatrix} \sqrt{(x - x_{lm_i})^2 + (y - y_{lm_i})^2} \\ \text{atan2}\left[(y_{lm_i} - y), (x_{lm_i} - x)\right] - \theta \end{bmatrix} \tag{4.9}$$

Measurements from the LIDAR sensor can be influenced by variations and inaccuracies. The noise to be considered is in fact:

- an error of $\sigma_r = 10$ cm on the range
- an error of $\sigma_\beta = 0.25$ deg on the bearing error

These inaccuracies can be modeled, as before, as a zero-mean Gaussian random variable that accounts for error sources:

$$W = \begin{bmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\beta^2 \end{bmatrix} \tag{4.10}$$

The measurements are then given by the h function summed to the measurements error:

$$z_{k+1} = h(q_{k+1}, p_i) + w_{k+1} \tag{4.11}$$

After computing the measurements, the filter linearizes the measurement model around the predicted state. Since the relationship between the state and the measurements is often nonlinear, the EKF approximates this relationship near the current state estimate using the measurement model's Jacobian, computed with noise equal to 0, and the the measurement's noise Jacobian:

$$H_q = \frac{\partial z}{\partial q} = \begin{bmatrix} \frac{x_{k+1} - x_{lm_i}}{\sqrt{(x_{k+1} - x_{lm_i})^2 + (y_{k+1} - y_{lm_i})^2}} & \frac{y_{k+1} - y_{lm_i}}{\sqrt{(x_{k+1} - x_{lm_i})^2 + (y_{k+1} - y_{lm_i})^2}} & 0 \\ \frac{-(y_{k+1} - y_{lm_i})}{(x_{k+1} - x_{lm_i})^2 + (y_{k+1} - y_{lm_i})^2} & \frac{x_{k+1} - x_{lm_i}}{(x_{k+1} - x_{lm_i})^2 + (y_{k+1} - y_{lm_i})^2} & -1 \end{bmatrix} \tag{4.12}$$

$$H_w = \frac{\partial z}{\partial w} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{4.13}$$

The next step is to compute the Kalman gain, which balances the uncertainty in the prediction and the uncertainty in the measurement. This factor determines how much weight to give to the new measurements versus the predicted state. If the measurements are considered very reliable, so having low uncertainty, the Kalman Gain will be larger, and the update will rely more heavily on the new measurements. If the prediction is deemed more reliable, the gain will be smaller, and the update will favor the prediction.

$$K_{k+1} = L_{k+1} H_{q,k+1}^T \left[(H_{q,k+1} L_{k+1} H_{q,k+1}^T) + (H_w W H_w^T) \right]^{-1} \tag{4.14}$$

Using this gain, the EKF updates the state estimate to minimize the error between the actual measurements and the measurements predicted by the model. The covariance matrix is also updated to reflect the reduced uncertainty after incorporating the measurement.

$$z_{k+1} = h(q_{k+1}) + w_{k+1} \tag{4.15}$$

$$\hat{q}_{k+1} = q_{k+1} + K_{k+1} \left[z_{k+1} - h(q) \right] \tag{4.16}$$

$$\hat{L}_{k+1} = \left[I - K_{k+1} H_{q,k+1} \right] L_k \tag{4.17}$$

The EKF corrects the predicted state by adding a weighted difference between the actual measurements and the measurements predicted by the model. This step is crucial because it combines the information from the prediction and the new data to improve the accuracy of the state estimate.

4.4 Results

The results of the EKF filter can be seen in the Figure 10. The reconstructed path seems to start closely following the actual path. There is a small divergence between the EKF path and the real path as the rover moves further along the trajectory. The EKF is experiencing increasing uncertainty as can be seen from the ellipses of uncertainty that become bigger in some parts of the trajectory.

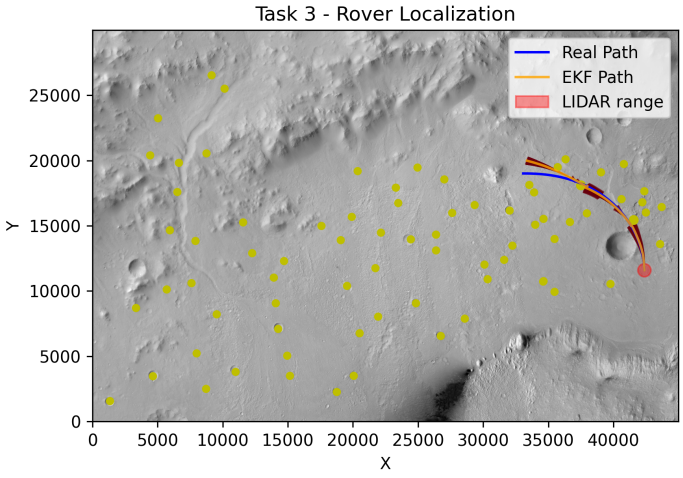


Figure 10: EKF reconstruction of the path

The performance of the LIDAR sensor, as well as how its data is integrated into the EKF, seems to be adequate for the initial part of the path but may need recalibration or refinement to handle the latter parts, as suggested by the path divergence. The EKF's role

in continually correcting the path estimation based on sensor measurements is evident. However, the corrections might not be fully compensating for the errors, leading to the observed deviation from the real path in the end of the trajectory.

In addiction, the evolution of the square root of the covariance matrix, as can be seen in Figure 11, is coherent with the ellipses of uncertainties plotted in the path. This matrix, in fact, represents the estimated uncertainty of the state estimate. Looking at the shape of the graph, there are three distinct peaks which suggest moments when the uncertainty in the system's state sharply increased.
After incorporating the new measurements, the EKF updates this matrix to reflect the new level of uncertainty. Since the update phase adds information, the uncertainty decreases, meaning the system is more confident in the state estimate after the update than it was before.
Towards the end of the graph, the uncertainty is increasing again, which could indicate again loss of confidence in the state estimates over time.

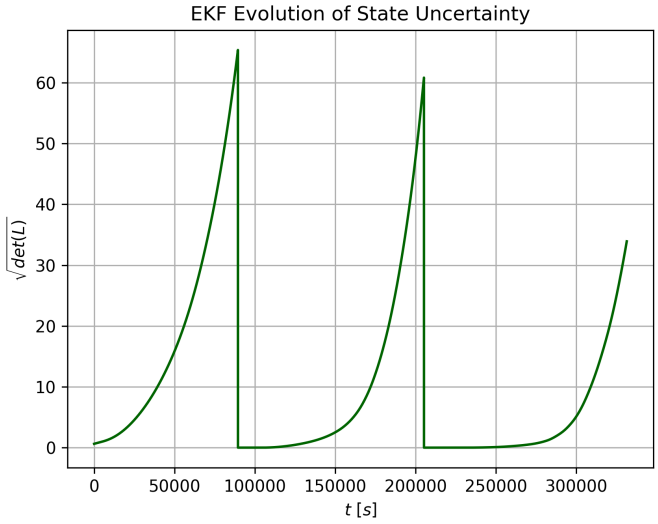


Figure 11: Evolution of the square root of the covariance matrix

5 Optional Task

The last and optional task requested to perform the EKF reconstruction of the path retrieved in task 2 from the $A^\star$ algorithm. The starting point of the rover is then the position P_{1r} and the target position is P_2. In order to perform this task, the path of the $A^\star$ algorithm is retrieved and converted from pixel into meters coordinates.
This conversion is performed as:

$$x = (X[0,0] + (\text{indices}[1] + 1) \cdot \text{mapRes}) - \text{mapRes} \tag{5.1}$$

$$y = (Y[0,0] - (\text{indices}[0] + 1) \cdot \text{mapRes}) + \text{mapRes} \tag{5.2}$$

Once the path in geographic coordinates is retrieved, a 5^{th} degree polynomial interpolation is performed in order to create a smoother version of the trajectory with more points. The spline consists of multiple polynomial segments, and each segment i of the spline can be represented as:

$$S_i(x) = a_{i,0} + a_{i,1}(x - x_i) + a_{i,2}(x - x_i)^2 + \cdots + a_{i,k}(x - x_i)^k \tag{5.3}$$

with $k = 5$.
The resulting output is an interpolated path of 10000 points. As the points have been subject to interpolation, the overall path is smoother than the one generated by $A^\star$. While the difference is imperceptible at such large scales, the two paths can be seen in Figure 12. In the Figure, the green line represents the interpolated path, while the yellow line the path generated by $A^\star$.

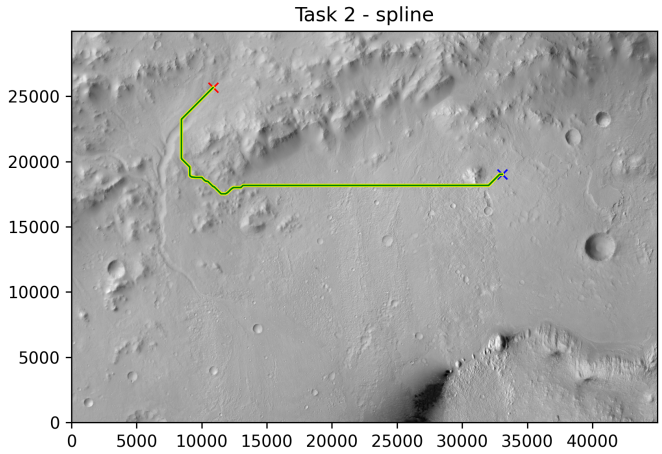


Figure 12: The two paths.

However this path is composed by just the x and y coordinates. In order to perform a reconstruction of the trajectory with the EKF method, the heading angle of the robot is required.
This can be calculated at each point of the path and added to the array as third component, by first calculating the difference in position at each point as:

$$dx = x_{i+1} - x_i \qquad dy = y_{i+1} - y_i \tag{5.4}$$

This difference represent the changes in x and y between each pair of consecutive points. Then the angle is calculated at each point along the path by using the $atan2$ function to compute the arc tangent of dy/dx, giving the angle in radians between each line segment and the x-axis. These values are then unwrapped to compute a smoothly varying angle, and added to the previous path array.

The result of this computation is a 10000-element long array comprised of triples.

The EKF computation is then performed as in the chapter 4.3 on this path. The result of the reconstruction can be seen in Figure 13. The figure clearly shows that the filter has some uncertainties in the part of the path that is located in a zone with less landmarks, confirmed in fact by a small deviation and by the presence of ellipses of uncertainties. As the rover passes near a new landmark, new measurements are embedded, the covariance matrix is updated and the trajectory corrected.

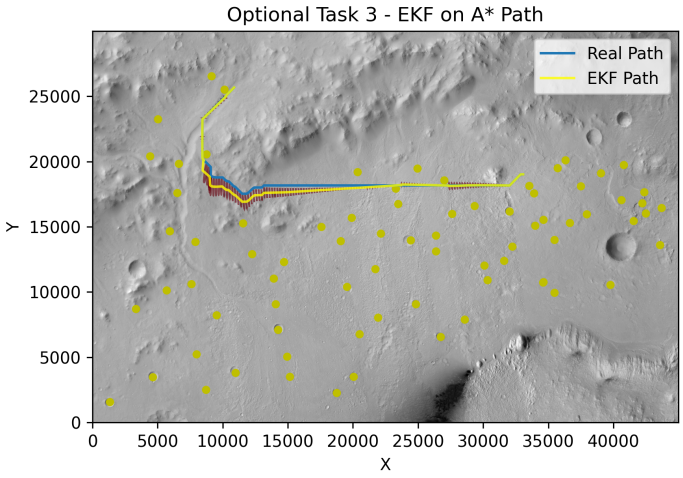


Figure 13: EKF reconstruction on $A^\star$ path

The evolution of the square root of the determinant of the covariance matrix is displayed in Figure 14. It is represented over the estimated time that the rover takes to perform this path. This is computed assuming that the rover moves at the maximum speed allowed, $v_{max} = 4$ cm/s, first computing the Euclidean distance between consecutive points and summing up the distances between consecutive points to get the total distance of the path:

$$tot\_distance = \sum_i d_i = \sum \sqrt{(x_{i,2} - x_{i,1})^2 + (y_{i,2} - y_{i,1})^2} \tag{5.5}$$

Then the time it takes to traverse the path is computed by dividing the total distance by the rover's speed:

$$time\_estimate = \frac{tot\_distance}{v_{max}} \tag{5.6}$$

The calculations approximate the travel time in 830543.406 seconds, 9.613 days, that are necessary for the rover to traverse the path.
Lastly the time it takes to travel between each pair of consecutive points is obtained by

dividing the distances by the rover's speed, and the time at each point along the path is the sum of the times between points.

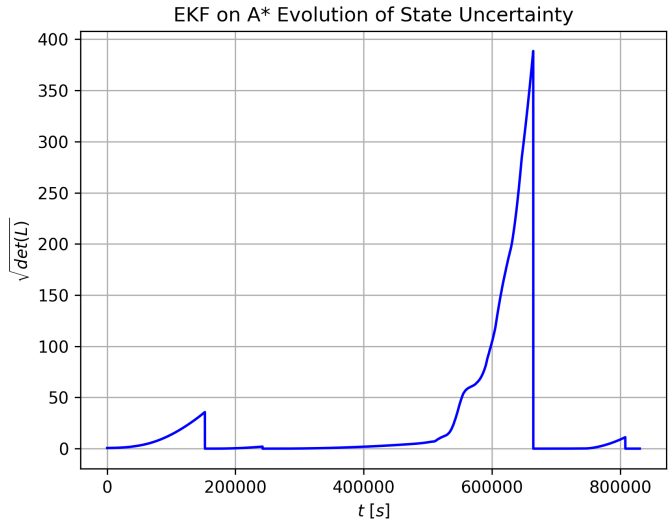


Figure 14: Evolution of the square root of the covariance matrix

In the figure we notice an important peak in the value of the square root of the determinant of the matrix, that correspond to the representation of the ellipses of uncertainties in the display of the path, visible in Figure 13 . Other minor peaks are visible in Figure 14, due to small intervals of uncertainties at the beginning of the path and at the end, but of minor importance since did not cause important deviations in the trajectory.

List of Figures