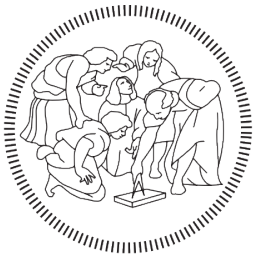
Politecnico di Milano
Scuola di Ingegneria Industriale
e dell'informazione
Space Engineering



POLITECNICO
MILANO 1863

Visual SLAM algorithm for a space exploration rover using Extended Kalman Filtering

Advisor:
Prof. Mauro Massari

Candidate:
Alessandro Turano
ID: 883273

ACADEMIC YEAR 2018-2019

Abstract

This thesis is about the development of a Extended Kalman Filter SLAM algorithm for a rover equipped with a stereo camera, wheels encoders and gravity sensor exploring an unknown 3D surface. SLAM algorithms can reconstruct the position and orientation of a rover using landmarks and at the same time build a map of the observed landmarks.

The Differential Algebra Computational Toolbox (DACE) library for C++ has been used for the estimation of the involved Jacobians.

In the Numerical results part it is shown how this algorithm reconstruct with more accuracy the position and orientation of the rover than an approach based merely on odometry integration.

Abstract

Questa tesi tratta lo sviluppo di un algoritmo SLAM basato su un filtro esteso di Kalman per un rover equipaggiato con una fotocamera stereoscopica, ruote con encoders e sensore di gravità che esplora una superficie 3D sconosciuta. Gli algoritmi SLAM possono ricostruire la posizione e l'orientamento di un rover usando le misurazioni di alcuni punti di riferimento e allo stesso tempo possono costruire una mappa dei punti di riferimento osservati.

È stata usata la libreria di Algebra Differenziale DACE per C++ per valutare le matrici Jacobiane coinvolte.

Nella parte dei risultati numerici viene mostrato come questo algoritmo permetta di ricostruire con molta più precisione la posizione e l'orientamento del rover rispetto ad un approccio basato solo sull'integrazione dell'odometria.

Ringraziamenti

In primo luogo desidero ringraziare il Prof. Mauro Massari per la sua disponibilità, la sua puntualità e i suoi preziosissimi consigli.

In questa tesi vengono riportati solo il mio nome e quello del mio Relatore, ma non sarebbe giusto non menzionare anche chi, in un modo o nell'altro, mi ha aiutato durante questo percorso.

Il primo ringraziamento è rivolto alla mia Famiglia, per avermi supportato durante questi anni lontano da casa e per avermi dato i mezzi per raggiungere questo traguardo altrimenti impossibile.

Un grazie anche ai miei amici più stretti, quelli con cui ho condiviso i momenti di vita quotidiana; vorrei fare una menzione particolare a Francesco (meglio conosciuto come Ciccio) e Raffaella.

Non potrei non ringraziare tutti i miei amici del Collegio San Paolo che ho conosciuto durante questi anni. Grazie a voi ho trascorso il periodo più bello della mia vita, in un posto speciale che ricorderò per sempre.

Un grazie ai miei amici di Lecce, perché nonostante le strade diverse che abbiamo intrapreso l'amicizia è sempre rimasta intatta.

Infine un ringraziamento speciale a Maria Grazia, che mi è stata sempre accanto, motivandomi e supportandomi ogni volta che ne avessi bisogno.

Contents

List of Figures

List of Tables

Nomenclature

DA	Differential Algebra
DCM	Direction Cosine Matrix
EKF	Extended Kalman Filter
FOV	Field Of View
GPS	Global Positioning System
HFOV	Horizontal FOV
SLAM	Simultaneous Localization and Mapping
VFOV	Vertical FOV

Chapter 1

Introduction

Localization and mapping are two fundamental problems for space exploration missions. Indeed on other planets GPS or similar satellite-based navigation systems are not available. The surface of the planet can be slippery and irregular, so relying only on the odometry can be too inaccurate. An alternative way of localization is needed.

SLAM is a good solution since it can solve simultaneously both the problems and exploit the created map to improve the precision of the localization.

1.1 Aim of the thesis

The aim of this thesis is to build and study an Extended Kalman Filter SLAM algorithm for the 3D case of a rover exploring an unknown surface. The algorithm will use the Differential Algebra Computational Toolbox (DACE) library for C++ for the estimation of the Jacobians.

The filter won't include a dynamic model since this is not relevant in the case analyzed (due to the small velocity of the rover and the unknown and irregular surface shape).

The considered sensors will be:

- A parallel stereo camera oriented like the trajectory.
- Wheels encoders used to get the odometry (distance traveled by the right and left wheels).
- Gravity sensor used to correct the local z axis orientation.

The considered map will be a randomly generated surface and the landmarks will be extracted as the local maxima points of the map.

The trajectory examined will be circular.

1.2 Comparison with other approaches

SLAM algorithm is not the only way to determine the position and the orientation of a rover. A more widespread methods is the only use of proprioceptive based odometry, for example based on the distance traveled by the wheels [6].

Instead a more complex approach is the use of visual odometry. This method has been used on the Mars Exploration Rovers.

The advantages of a SLAM algorithm with respect to these approaches are mainly two:

- A more precise estimation of the position and the orientation of the rover with respect to the proprioceptive based odometry, specially on high-slip environments [11, 12].
- The creation of a map made by the landmarks observed during the path. More complex SLAM algorithm than the one developed in this thesis could have global consistency. This is achieved by realizing that a previously mapped area has been previously observed (loop closure) and this information is used to reduce the drift in the estimates [25].

In this thesis the results obtained with the SLAM algorithm will be compared to an odometry approach based on the distance traveled by the wheels of a rover. The high-slip environment will be simulated with a Gaussian error on the odometry data.

1.3 SLAM algorithms

"Simultaneous Localization and Mapping (SLAM) is the problem of concurrently estimating in real time the structure of the surrounding world (the map), perceived by moving exteroceptor sensors, while simultaneously getting localized in it." [19]

SLAM algorithms solve the problem of spatial exploration. When a person is in an unknown space, he observes what is around him while moving.

Without even knowing he is building a map of the world around him, and at the same time he is being spatially located in this map. This is also what a SLAM algorithm does [25].

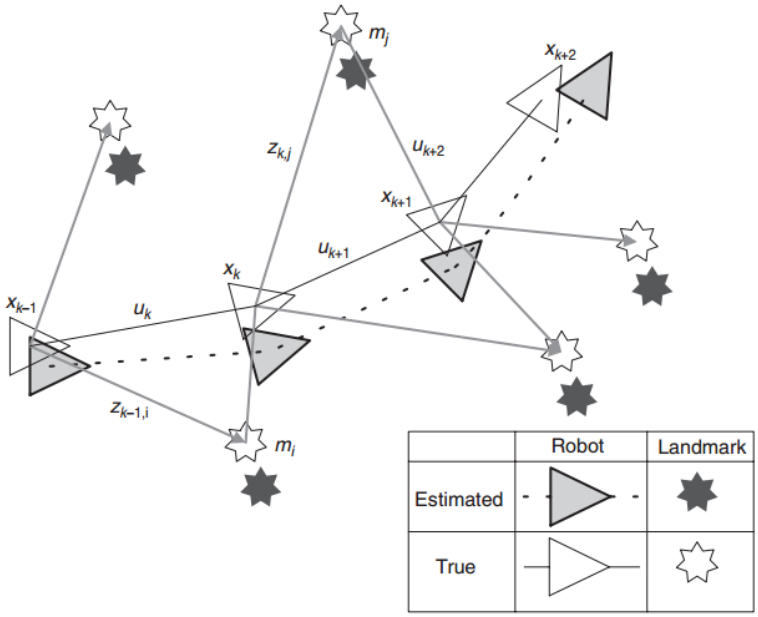


Figure 1.1: SLAM algorithm

The filter used in this thesis is an Extended Kalman Filter. This filter is likely the most widespread technique for state estimation. EKF is based on the Bayes filter for the filtering and prediction of non linear functions, which are linearized using a 1^{st} order Taylor series expansion [25, 20].

EKF is also based on the assumption that the estimated states has a Gaussian distribution and, by applying the transformations, these keep Gaussian distributions [7].

The main features of SLAM will be explained in the following lines:

SLAM algorithm basically consist of three different operations, which are iterated at each time step [21]:

- **The rover observes new landmarks,** which can be used to create a map. The position reconstructed by the rover is uncertain due to errors. The mathematical model that computes the position of the landmarks using the measurements and the position of the rover is called *inverse observation model*.
- **The rover moves,** and reaches a new point of the map. Due to the errors this increases the uncertainties of the position. The mathematical model for simulating this process is called *motion model*.
- **The rover reobserves the landmarks,** and uses them to correct its position, orientation and also the position of all landmarks in space. After this operation all the uncertainties decrease. The mathematical model that predict the measurements using both the position of the landmarks, the position of the rover and its orientation is called *direct observation model* [19].

1.4 Differential Algebra

The theory of Differential Algebra (DA) has been developed by Martin Benz in the late 80's. This section is about the main idea behind it and its basics.

DA techniques find were created to solve analytical problems by an algebraic approach. The basic idea is to treat functions and operations in a similar way to the real numbers [3, 2, 4].

A real numbers $\mathbb{R}$ is represented on a computer by an approximation, a floating point number $\mathbb{F}$. Each operation on $\mathbb{F}$ is defined so that the resulting number is an approximation of the real computation. Using this environment it is possible to evaluate complicated mathematical equations and obtain a floating point number as result.

The figure below is an example useful to make the concept clearer. The evaluation of the expression $\frac{1}{x+1}$ is computed with real numbers, so without any approximation, on top and with floating point numbers at the bottom.

The final result in the floating point environment is only an approximation of the real computations, but it's precision can be adjusted to satisfy the requirements and in general is good enough for a practical case.

$$\begin{array}{lccccc}
\mathbb{R}: & \mathbf{2} & \xrightarrow{+1} & \mathbf{3} & \xrightarrow{1/} & \mathbf{\frac{1}{3}} \\
 & \Updownarrow = & & \Updownarrow = & & \Updownarrow \approx \\
\mathbb{F}: & \mathbf{2} & \xrightarrow[+1]{} & \mathbf{3} & \xrightarrow[1/]{} & \mathbf{0.3333}
\end{array}$$

Figure 1.2: Evaluation of $\frac{1}{x+1}$ for $x = 2$ in $\mathbb{R}$ and $\mathbb{F}$

Such as real numbers, also functions can be approximated to be operated on in an algebraic way. The key feature of DA is to efficiently represent sufficiently smooth functions in a computer environment so that they can be handled with arithmetic expressions. From this point of view the idea is very similar to floating point numbers.

$$\begin{array}{lccccc}
C^r(0): & \boldsymbol{x} & \xrightarrow{+1} & \boldsymbol{x+1} & \xrightarrow{1/} & \mathbf{\frac{1}{x+1}} \\
 & \Updownarrow = & & \Updownarrow = & & \Updownarrow \approx \\
DA: & \boldsymbol{x} & \xrightarrow[+1]{} & \boldsymbol{x+1} & \xrightarrow[1/]{} & \mathbf{1 - x + x^2 - x^3}
\end{array}$$

Figure 1.3: Evaluation of $\frac{1}{x+1}$ in the function space C^r and DA arithmetic

As with floating point numbers also with DA the representation is an approximation. In fact DA make use of the truncated Taylor expansion of a function f up to a fixed order. The function is evaluated around the origin [3].

In this work the DA is used to compute the Jacobians matrices. In fact derivation in the DA framework is an easy task like selecting the correct coefficient of a polynomial (the Taylor expansion mentioned above).

More traditionally these matrices are computed analytically if the models involved are simple enough or with a numerical differentiation approach if the models are more complex.

1.5 Overview of the thesis

The thesis is divided into five chapters:

- The first explains which model has been considered, the two different reference frames with their mathematical representation and the architecture of the simulation.
- The second chapter is about the generation first of the simulation environment and then of the odometry data and measurements.
- The third chapter is about the SLAM algorithm and how it works step-by-step.
- The fourth chapter is about the obtained results and their analysis, with the SLAM algorithm and, for comparison, with an odometry based approach.
- In the last one there are the conclusions and the future developments.

Chapter 2

Model

2.1 Rover

The size of the considered rover is similar to the Curiosity rover, which is about the size of a small SUV [13]:

	$[m]$
Length	3
Width	2.7
Height	2.2

Table 2.1: Curiosity rover size

The required data for the considered rover, like the camera height or the distance between the cameras has been derived to be compatible with the mentioned size:

	Variable	Value in $[m]$
Camera height	h	1.5
Distance between the left and right wheels	$d_{baseline}$	2
Distance between the cameras	$2 * \Delta stereo$	2

Table 2.2: Considered size

The technical specification used for the simulation are reported here:

	Variable	Value	Unit of measure
Speed	v	0.1	$[m/s]$
Odometry frequency	$f_{odometry}$	3	$[Hz]$
SLAM frequency	f_{SLAM}	1	$[Hz]$
Horizontal Field Of View	$HFOV$	60	$[^\circ]$
Vertical Field Of View	$VFOV$	30	$[^\circ]$
Number of horizontal pixels	$2 * x_{max}$	5120	–
Number of vertical pixels	$2 * y_{max}$	5120	–

Table 2.3: Considered specifications

The rover has always the camera oriented like the velocity vector. This assumption is useful to simplify the model and only account for 6 degrees of freedom, 3 for the position and 3 for the orientation.

Actually the state vector s contains 7 variables only for the rover. That is because the orientation is expressed with a non-singular representation, the rotation quaternion, so an extra variable is needed.

2.2 Models

The are different models involved in this work. This subsection is a summary of the most important ones to better understand the development of the thesis.

2.3 Odometry

The odometry uses the distance traveled by the left and right set of wheels (only two distances are considered, one for the left side and one for the right side, so the model is independent on the number of wheels) to determine the displacement and the angle of curvature for every integration step. The measurements of the distances account a Gaussian error model.

Since the surface shape is unknown a priori, in the SLAM algorithm the displacement is considered in the $x - y$ plane of the local reference frame (so with a zero local z component). This is maybe the assumption that has the biggest limitation on the prediction step (4.2) but it is a forced choice because of the absence of other information.

To better understand the orientation of the rover the SLAM algorithm has access to the normal to the surface (altered by a Gaussian error) only

at the first integration step of the odometry data. This simulates the presence of a gravity sensor.

The trajectory considered is circular, around the center of the randomly generated map.

2.4 Camera setup

The camera considered is a stereo camera. Using the two different acquisitions made by this camera it is possible to reconstruct the relative position of the landmark without other information. The technical specification of this camera are written in 2.1. The data association has not been simulated, so the landmarks are recognized every times without failures.

2.5 Architecture of the simulation

The simulation is divided in three phases. First there is the generation of the simulation environment and of the landmarks, selected as the local maxima of the surface.

Then there is the generation of the odometry data and the measurements. This part is done before the start of the SLAM algorithm with the exact knowledge of the surface, the position and the orientation. The data generated is subject to an error. The last step is the SLAM algorithm. The algorithm uses the odometry data and the measurements generated in the previous phase.

2.6 Reference frames

Two different reference frames are considered, the absolute frame, fixed with the map, and the local frame, which follows the motion of the rover and orientation:
The **absolute frame** has the center in $(0, 0)$ and is oriented like the axes of the map.
The **local frame** has the center in the position of the rover and is oriented with:

- x axis defined as $x = y \otimes z$.
- y axis oriented like the velocity vector of the rover.
- z axis oriented like the normal vector to the surface.

2.6.1 Rotation quaternion

A rotation quaternion is used to express the orientation of the rover and to save it in the state vector s.
This non-singular representation is the one with the minimum number of variables as well as being more computationally efficient than the Direction Cosine Matrix (DCM) counterpart.

The following subsections correspond to operations widely used in this work.

2.6.1.1 From DCM to rotation quaternion

Since the local reference frame is defined starting from both the velocity vector of the rover and the normal vector to the surface, it's easy to build the DCM.
The matrix can be composed entering in every row the three absolute components (x, y, z) of the local reference frame axes.
So the DCM that switches from the absolute frame to the local one is made by:

$$\begin{bmatrix} x_x & x_y & x_z \\ y_x & y_y & y_z \\ z_x & z_y & z_z \end{bmatrix}$$

With the three axes defined in 2.6.

One interesting property is that, since the DCM is orthogonal, the inverse of the DCM can be computed as its transposed:

$$DCM^{-1} = DCM^T$$

Given that the chosen representation is the quaternion one, the DCM must be converted in the quaternion form.
This can be done in different ways. The selected approach is the following one:

- If the trace of the DCM is positive:
  - $sq = \frac{1}{2*(trace+1)}$
    $q_0 = \frac{1}{4*sq}$
  - $q_1 = [DCM(2,1) - DCM(1,2)] * sq$
    $q_2 = [DCM(0,2) - DCM(1,2)] * sq$
    $q_3 = [DCM(1,0) - DCM(0,1)] * sq$
- Otherwise if $DCM(0,0) > DCM(1,1)$ and $DCM(0,0) > DCM(2,2)$:

- $sq = 2 * \sqrt{1 + DCM(0,0) - DCM(1,1) - DCM(2,2)}$

  $q_0 = [DCM(2,1) - DCM(1,2)]/sq$
  $q_1 = \frac{1}{4*sq}$
  $q_2 = [DCM(0,1) + DCM(1,0)]/sq$
  $q_3 = [DCM(0,2) + DCM(2,0)]/sq$

- If the first two expressions are not satisfied and $DCM(1,1) > DCM(2,1)$ then:

  - $sq = 2 * \sqrt{1 + DCM(1,1) - DCM(0,0) - DCM(2,2)}$

    $q_0 = [DCM(0,2) - DCM(2,0)]/sq$
    $q_1 = [DCM(0,1) + DCM(1,0)]/sq$
    $q_2 = \frac{1}{4*sq}$
    $q_3 = [DCM(1,2) + DCM(2,1)]/sq$

- If none of the expressions above is true:

  - $sq = 2 * \sqrt{! + DCM(2,2) - DCM(0,0) - DCM(1,1)}$

    $q_0 = [DCM(1,0) - DCM(0,1)]/sq$
    $q_1 = [DCM(0,2) + DCM(2,0)]/sq$
    $q_2 = [DCM(1,2) + DCM(2,1)]/sq$
    $q_3 = \frac{1}{4*sq}$

2.6.1.2 Normalize a rotation quaternion

All the relations defined in this section assume normalized quaternions. So, it's a good practice to always normalize the quaternions before applying these methods.
The normalization of the quaternion is analogous to the vector normalization. It is defined as:

$$q_{normalized} = \frac{q}{\sqrt{q_0 + q_1 + q_2 + q_3}}$$

2.6.1.3 Invert a rotation quaternion

This relation can be used to invert a rotation quaternion:

- $d = \sqrt{q_0} + \sqrt{q_1} + \sqrt{q_2} + \sqrt{q_3}$
- $q_0^{-1} = \ \ q_0/d$
  $q_1^{-1} = -q_1/d$
  $q_2^{-1} = -q_2/d$
  $q_3^{-1} = -q_3/d$

2.6.1.4 Hamiltonian product

The Hamiltonian product is used to multiply two quaternions. Given three quaternions (x, y, z), $z = x * y$ is defined as:

$$z_0 = x_0 * y_0 - x_1 * y_1 - x_2 * y_2 - x_3 * y_3$$
$$z_1 = x_0 * y_1 + x_1 * y_0 + x_2 * y_3 - x_3 * y_2$$
$$z_2 = x_0 * y_2 - x_1 * y_3 + x_2 * y_0 + x_3 * y_1$$
$$z_3 = x_0 * y_3 + x_1 * y_2 - x_2 * y_1 + x_3 * y_0$$

2.6.1.5 Rotate a vector

To switch from one reference frame to another a rotation quaternion is used.
To rotate a vector from the absolute reference frame to the local one these steps are followed:

- Transformation from vector v to quaternion p:

  $$\begin{aligned} p_0 &= 0 \\ p_1 &= v_x \\ p_2 &= v_y \\ p_3 &= v_z \end{aligned}$$

- Rotation of the quaternion p using the rotation quaternion q (the product used in the following equation is the Hamiltonian product 2.6.1.4):

  $$p' = q * p * q^{-1}$$

- Extraction of the vector v' from the quaternion p':

  $$\begin{aligned} v'_x &= p_1 \\ v'_y &= p_2 \\ v'_z &= p_3 \end{aligned}$$

2.6.1.6 Combine two rotations

Two consecutive rotations (first by q_1 and then by q_2) can be combined into an equivalent one with this relation, using the Hamiltonian Product (2.6.1.4):

$$q' = q_2 * q_1$$

Chapter 3

Simulation environment

The generation of the environment is done using a MATLAB script.
It generates the surface, the normals to the surface, the landmarks and exports these data as plain text files readable from the odometry data and measurements generator.

3.1 Map and landmarks

3.1.1 Map

The map is generated as a random surface using a MATLAB function that takes as inputs these parameters:

- σ - Standard deviation.
  Changes the steepness of the surface (higher means steeper).
- H - Hurst exponent ($0 \leqq H \leq 1$).
  Changes the roughness of the surface (higher means rougher).
- Length of topography in direction x.
- Number of pixels in x direction.
- Number of pixels in y direction.

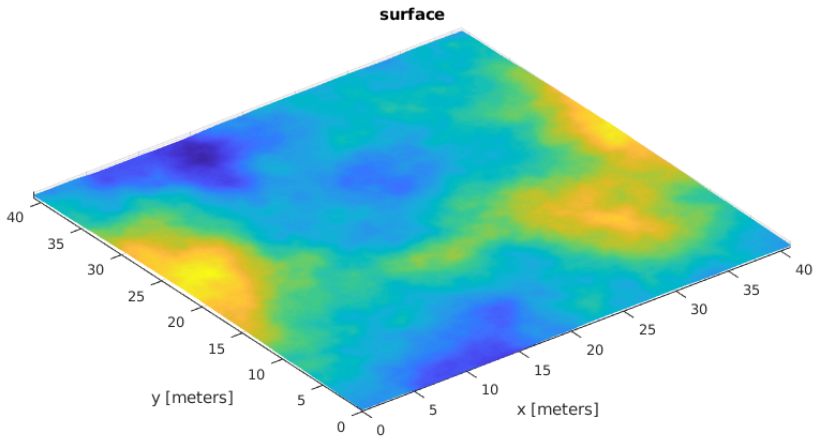


Figure 3.1: Randomly generated surface

The normal vectors to every point of the map are computed using a build-in function of MATLAB.
It takes as input only the height of the surface for all the x and y points.
The three outputs matrices (x, y and z components of the normal vector) must be rearranged in a vector form to be more easily readable from the odometry data and the measurements generator program.

3.1.2 Landmarks

The landmarks are selected as the local maximum of the surfaces.
Also these computations are done using a MATLAB function.
The adjustable parameters are:

- The surface generated above.
- The minimum distance between two local maxima.

After the computation, the local maxima nearer than the minimum distance from the borders of the map are excluded. Otherwise the points found in these four belts would be computed for a smaller local area, resulting in a more concentrated number of landmarks.

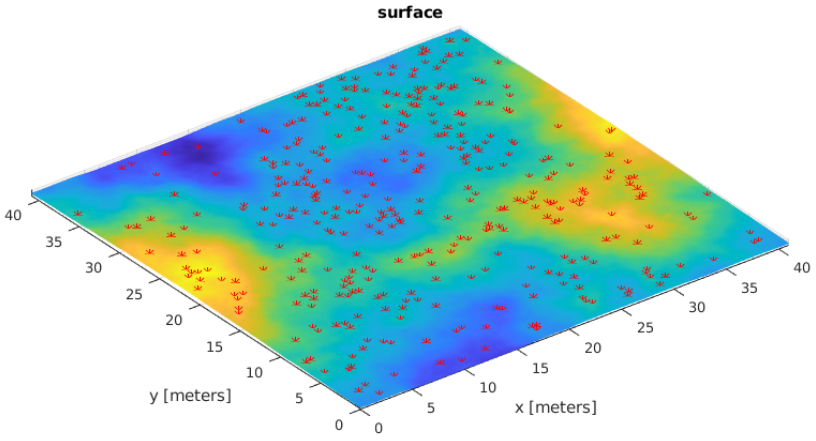


Figure 3.2: Landmarks on the surface

3.2 Measurements and odometry data generation

Four different measurements are acquired for every landmark in FOV and for every step of the algorithm:

- x coordinate in pixels of the landmark seen by the left camera
- y coordinate in pixels of the landmark seen by the left camera
- x coordinate in pixels of the landmark seen by the right camera
- y coordinate in pixels of the landmark seen by the right camera

To generate the measurements the following steps are followed:

3.2.1 Update the displacement and the velocity vector

The displacement is computed on the local plane $x - y$, considering a rotation angle α and a radius of curvature r_c, both fixed.

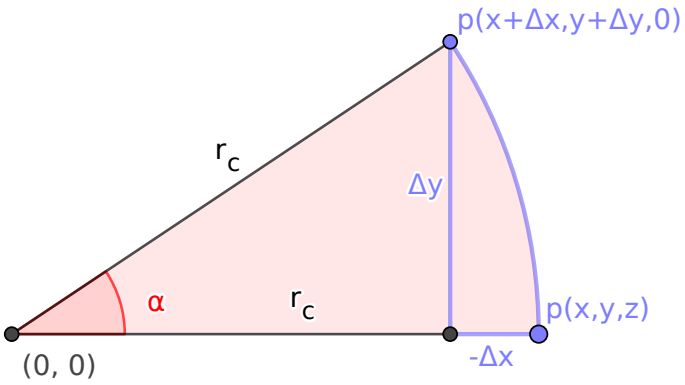


Figure 3.3: Odometry

Knowing r_c and α it's easy to compute the displacement as:

$$\Delta x = r_c * [1 - cos(\alpha)]$$
$$\Delta y = r_c * sin(\alpha)$$
$$\Delta z = 0$$

The new velocity vector, used in 3.2.4 to update the orientation, is computed simply rotating counterclockwise the local velocity vector (which is pointed as the local y axis) by an angle α.

The new velocity unit vector in the local frame will be:

$$v_x = -sin(\alpha)$$
$$v_y = cos(\alpha)$$
$$v_z = 0$$

To rotate the vector in the absolute frame the method explained in 2.6.1.5 is employed, using the vector v and the rotation quaternion q^{-1}(2.6.1.3).

3.2.2 Update the position

The x and y coordinates of the position of the rover on the surface are updated using the Δx and Δy computed in 3.2.1.

The z coordinate is obtained as the height of the surface in that point.

$$p_x^N = p_x^{N-1} + \Delta x$$
$$p_y^N = p_y^{N-1} + \Delta y$$
$$p_z^N = surface(x, y)$$

Then the position of the center of the rover c is computed as the sum of the position of the rover on the surface p and the normal to the surface multiplied by the height of the rover h.

$$c^N = p^N + normal * h$$

3.2.3 Save the odometry data

At each step the distance d_l traveled by the left wheel, d_r traveled by the right wheel and the three components of the normal to the surface *normal* in the current position are saved in a plain text file.

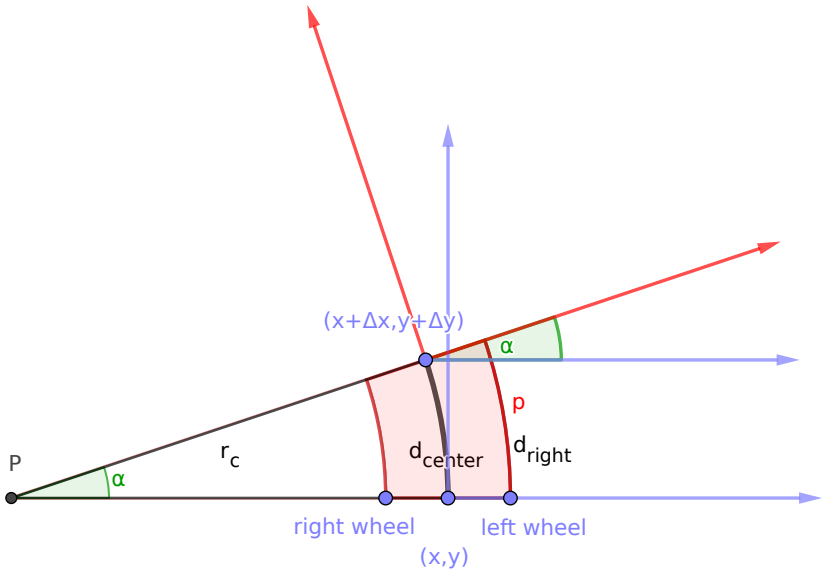


Figure 3.4: Distance traveled by the wheels

The distance traveled by the rover is called d_{center}, while the distance between the two wheels is called $d_{baseline}$.

Remembering that the arc length is equal to the radius times the angle ($d = \alpha * r$), the length of the arc d_{center} is [14]:

$$d_{center} = \frac{d_{left} + d_{right}}{2}$$

Subtracting $\alpha * r_{right} = d_{right}$ from $\alpha * r_{left} = d_{left}$:

$$\alpha(r_{right} - r_{left}) = d_{right} - d_{left}$$

Another known relation is that the sum of the radius of curvature for the left wheel and the distance between the wheels gives us the radius for the right wheel:

$$r_{left} + d_{baseline} = r_{right}$$
$$d_{baseline} = r_{right} - r_{left}$$

Applying the last relation to the previous one:

$$\alpha = \frac{d_{right} - d_{left}}{d_{baseline}}$$

So substituting both α and d_{center} in $\alpha * r_{center} = d_{center}$:

$$\frac{d_{right} - d_{left}}{d_{baseline}} * r_{center} = \frac{d_{left} + d_{right}}{2}$$

So the distance traveled by the wheels are computed as:

$$d_l = \frac{(2*r_c - d_{baseline})*\alpha}{2}$$
$$d_r = \frac{(2*r_c + d_{baseline})*\alpha}{2}$$

An error (generated by a Gaussian distribution of zero mean and with variance σ_o, $\mathcal{N}(o, \sigma_o)$, is summed to the two distances.

$$d_l = d_l + \mathcal{N}(o, \sigma_o)$$
$$d_r = d_r + \mathcal{N}(o, \sigma_o)$$

The Gaussian error is then added to the three components of the normal:

$$normal_x = normal_x + \mathcal{N}(o, \sigma_n)$$
$$normal_y = normal_y + \mathcal{N}(o, \sigma_n)$$
$$normal_z = normal_z + \mathcal{N}(o, \sigma_n)$$

3.2.4 Update the orientation

As explained in 2.6, the local frame is computed starting from the velocity and the normal vectors.

The updated velocity vector comes from 3.2.1. To make sure it is orthogonal to the normal to the surface only the perpendicular part is extracted from it:

$$v_\perp = v - (v \cdot normal) * normal$$

Using the normalized $v_\perp$and the normal to the surface, the DCM is build and then converted to rotation quaternion, following the procedure illustrated in 2.6.1.1.

3.2.5 Compute the spherical coordinates

To obtain the measurements in pixels for every landmark in the FOV the spherical coordinates θ_{left}, ϕ_{left}, θ_{right}, ϕ_{right} must be computed.

The Cartesian coordinates for the right and left camera for the i landmark can be found switching from the absolute frame to the local frames of the cameras.

To do that the position of the center of the rover are subtracted to the absolute position of the landmark ($l = l_{absolute} - c$) and then, using the inverse of the rotation quaternion q^{-1} (2.6.1.3), the position of the landmark can be rotated in the local reference frame of the rover (as explained in 2.6.1.5) [24].

The Cartesian coordinates of the landmark for the two cameras are found adding half of the distance between the two cameras ($\Delta stereo$) to the local x coordinate of the landmark for the left camera or subtracting it for the right one.

$$\begin{aligned} l_{left} &= l_x + \Delta stereo \\ l_{right} &= l_x - \Delta stereo \end{aligned}$$

The conversion from Cartesian coordinates to Spherical coordinates is made in this way:

$$\begin{aligned} \rho &= \sqrt{l_x^2 + l_y^2 + l_z^2} \\ \theta &= atan2(\sqrt{l_x^2 + l_y^2}, l_z) \\ \phi &= atan2(l_y, l_x^2) \end{aligned}$$

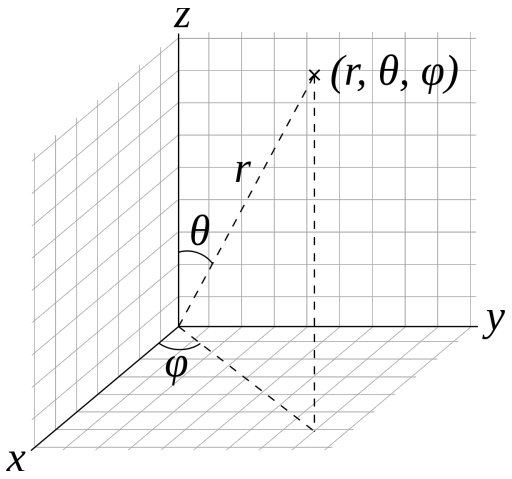


Figure 3.5: Spherical coordinates

3.2.6 Add the error

The error model is the exponentially correlated random noise, better suited for camera measurements than a simpler Gaussian noise, which is completely uncorrelated for different time steps [17].

As expected the error E^k is correlated to the error at the previous step E^{k-1} exponentially.

The first value E^1 is generated using a normal distribution with 0 mean and standard deviation σ_m at the first step or every time a landmark ends out of the FOV.

$$E^1 = \mathcal{N}(o, \sigma_m)$$

Starting from the second iteration the error is computed as:

$$E^k = K * E^{k-1} + \sqrt{1 - K^2} * \mathcal{N}(0, \sigma_m)$$
$$K = e^{-\frac{1}{f*\tau}}$$

This correlation decay with a time scale given by τ.

To better understand how this model affects the measurements an example plot error-time for this model is provided below.

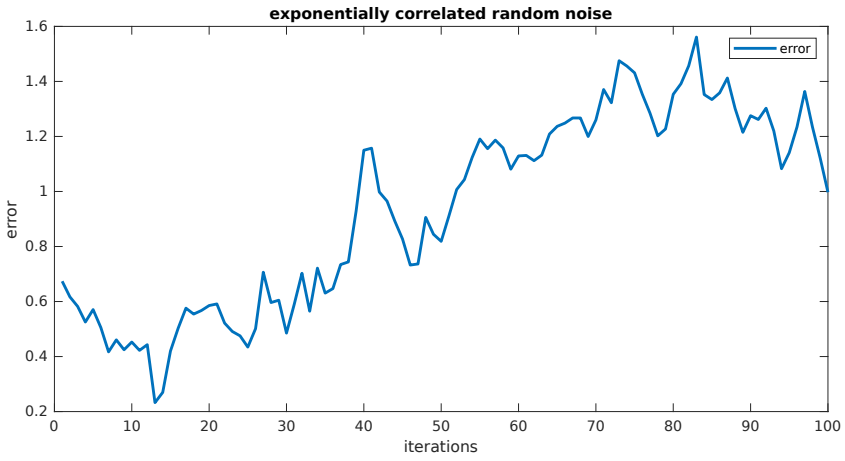


Figure 3.6: Error-time plot

This error is summed to the spherical coordinates θ and ϕ for both the cameras before saving the results.

3.2.7 Compute the measurements

To commute from spherical coordinates to pixels coordinates it is first assumed to work only with the x axis (the procedure is the same for y).

A right triangle whose hypotenuse goes from the camera to the edge of the image can be drawn. This triangle is represented in blue in the figure below. The angle made between the base of length L, perpendicular to the image, and the hypotenuse equals half of the horizontal FOV ($HFOV$).

Another triangle, this time with the hypotenuse that goes from the camera to the landmark, but with the same base L can be drawn. This triangle is represented in red. The angle between the hypotenuse and the base coincide with $\theta - \frac{pi}{2}$, where θ is the spherical coordinate. The side of the triangles opposite to this angle is the x pixels coordinate, called p_x.

Both the pixels coordinates, p_x and p_y, have zero value in the center of the captured image.

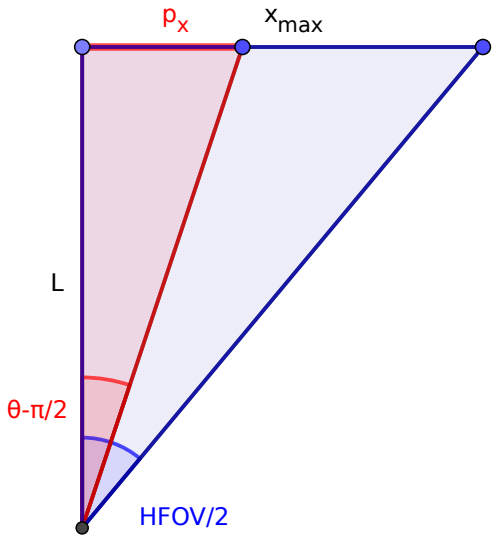


Figure 3.7: Similar triangles

It is now possible to write some trigonometry relations:

$$tan(\frac{HFOV}{2}) = \frac{x_{max}/2}{L}$$
$$tan(\phi - \frac{\pi}{2}) = \frac{p_x}{L}$$

Since the distance L is the same for both the triangles, it is possible to impose the equality for the two triangles to find out p_x:

$$L = \frac{x_{max}/2}{tan(\frac{HFOV}{2})}$$
$$L = \frac{p_x}{tan(\phi - \frac{\pi}{2})}$$

And finally the relation between p_x and the angle $\phi - \frac{pi}{2}$ can be computed:

$$p_x = \frac{x_{max} * tan(\phi - \frac{\pi}{2})}{2 * tan(\frac{HFOV}{2})}$$

While the inverse relation is:

$$cot(\phi - \frac{\pi}{2}) = \frac{x_{max}}{2 * p_x * tan(\frac{HFOV}{2})}$$

The same procedure can be done for p_y resulting in these relations:

$$p_y = \frac{y_{max} * tan(\theta - \frac{pi}{2})}{2 * tan(\frac{VFOV}{2})}$$
$$cot(\theta - \frac{\pi}{2}) = \frac{y_{max}}{2 * p_y * tan(\frac{VFOV}{2})}$$

Both the x and y coordinates are integer numbers, rounded to the nearest one. The measurements are saved in a plain text file.

Chapter 4

EKF-SLAM algorithm

The state vector s considered for the SLAM algorithm is composed by the three coordinates of the position of the rover (x, y, z), the four coordinates of the rotation quaternion of the local reference frame that coincide with the orientation of the rover (q_0, q_1, q_2, q_3), and three coordinates of the absolute position of every landmark (x, y, z). So the total state vector s length is $7 + 3 * number\,of\,landmarks$.

The length of the state vector s varies depending on the number of seen landmarks in the current iteration, but a maximum number of landmark is imposed.

The same goes for the state error covariance matrix P (called from now on simply covariance matrix P). The covariance matrix P shows the error associated with the robot and the landmark state estimations. From the covariance matrix P, it is possible to monitor the error and the uncertainties of the estimation. Therefore the study on its behavior is one of the most important issues of SLAM [15].

The covariance matrix P total size is $(7 + 3 * number\,of\,landmarks) * (7 + 3 * number\,of\,landmarks)$ and varies likewise the state vector s.

$$P = \begin{bmatrix} P_{rover} & P_{rover-landmark} \\ \\ P^T_{rover-landmark} & P_{landmark} \end{bmatrix}$$

The covariance matrix P is divided in its three sub-matrices:

- The covariance of the rover P_{rover}, of fixed size $(7) * (7)$.
- The cross-covariance rover and landmark $P_{rover-landmark}$, of variable size $(7) * (3 * number\,of\,landmarks)$.

- The covariance matrix of the landmark $P_{landmark}$, of variable size $(3 * number\,of\,landmarks) * (3 * number\,of\,landmarks)$.

4.1 Initialization

The following steps are executed only one time at the beginning of the algorithm:

The covariance matrices are initialized full of zeros. The starting values of the state vector s are the correct ones used in the simulation environment. This is not initialized with an error since the map is build with respect to the starting position.

4.2 Prediction

The fist step of the iterative part of the algorithm is the prediction step. In this step a guess on the new position is made using the odometry data and the information of the previous step .

The following calculations are iterated multiple times before going on with the algorithm (in this specific case three times). This is done because the odometry data is integrated at a smaller time step with respect to the SLAM algorithm itself.

Other than the odometry generation (3.2.3), in which the displacement is consistent with the surface height, the prediction step in the SLAM algorithm hypothesizes a displacement in the $x - y$ local plane. This assumption has to be done since the surface shape is unknown and therefore the rover doesn't have other information to exploit.

4.2.1 Mean

The first thing that has to be done is computing the angle α using the values of the distance traveled by the left and right wheels. As already seen in 3.2.3:

$$\alpha = \frac{d_{right} - d_{left}}{d_{baseline}}$$

Then it is possible to obtain is the distance traveled by the rover and the radius of curvature:

$$\begin{aligned} d_{center} &= \tfrac{d_{left}+d_{right}}{2} \\ r_c &= \tfrac{d_{center}}{\alpha} \end{aligned}$$

And finally the displacement in the local reference frame:

$$\Delta x = r_c * [1 - cos(\alpha)]$$
$$\Delta y = r_c * sin(\alpha)$$
$$\Delta z = 0$$

This displacement vector can be rotated in the absolute reference frame using the inverse if the rotation quaternion q^{-1}, as explained in 2.6.1.5.

The last thing to do is the update of the current position adding the displacement to the position at the previous step (referred to the absolute frame):

$$x = x + \Delta x$$
$$y = y + \Delta y$$
$$z = z + \Delta z$$

Then the orientation of the rover must be updated.

Only in the first iteration of the prediction step also the normal to the surface, detected by the gravity sensor and with its relative error, is used to compute the correct orientation of the rover. This is done building the DCM matrix of the rotation from the absolute frame to the local frame. In a way very similar to the odometry generation already explained in 3.2.3, the rotation matrix is built using the updated velocity unit vector and the normal to the surface.

The velocity unit vector is updated rotating the local axis y of α counterclockwise:

$$v_x = -sin(\alpha)$$
$$v_y = cos(\alpha)$$
$$v_z = 0$$

The updated velocity vector is then rotated in the absolute frame using the inverse of the rotation quaternion q^{-1}, as previously explained in 2.6.1.5.

To take only the orthogonal part to the normal to the surface only the perpendicular component is extracted from the vector:

$$v_\perp = v - (v \cdot normal) * normal$$

Using the normalized $v_\perp$ and the normal to the surface it is now possible to build the DCM and then convert it to rotation quaternion, following the steps written in 2.6.1.1.

$$DCM = \begin{bmatrix} x_x & x_y & x_z \\ v_x & v_y & v_z \\ normal_x & normal_y & normal_z \end{bmatrix}$$

Otherwise, if it is not the first iteration of the prediction step, the velocity unit vector v is rotated around the local z axis of the angle α with this method:

First the local z axis vector expressed in the absolute reference frame is found rotating $[0, 0, 1]$ with the inverse of the rotation quaternion (2.6.1.5).

Then the rotation quaternion z_q is composed in this way:

$$\begin{aligned} z_{q0} &= cos(-\tfrac{\alpha}{2}) \\ z_{q1} &= z_x * sin(-\tfrac{\alpha}{2}) \\ z_{q2} &= z_y * sin(-\tfrac{\alpha}{2}) \\ z_{q3} &= z_z * sin(-\tfrac{\alpha}{2}) \end{aligned}$$

Finally using the Hamiltonian Product (2.6.1.4) the new rotation quaternion is found:

$$q^N = z_q * q^{N-1}$$

4.2.2 Covariance

The uncertainties rises after this step, due to the errors introduced by the movement of the rover. Indeed the movement is only partially known (since the out of plane contribution is unknown due to unpredictable surface shape) and the known components are not exact.

To update the covariance sub-matrices the Jacobian of the prediction model A and the noise matrix Q are needed.

The Jacobian of the prediction model A is computed using DA (1.4). Adding an independent DA variable to each of the first 7 variables of state (x, y, z, q_0, q_1, q_2, q_3) before entering the prediction step it is possible to obtain the Jacobian as the derivative of each updated variable of state with respect to the 7 DA objects:

$$A = \begin{bmatrix} \frac{\partial x}{\partial x} & \frac{\partial x}{\partial y} & \cdots & \frac{\partial x}{\partial q_3} \\ \frac{\partial y}{\partial x} & \frac{\partial y}{\partial y} & \cdots & \frac{\partial y}{\partial q_3} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial q_3}{\partial x} & \frac{\partial q_3}{\partial y} & \cdots & \frac{\partial q_3}{\partial q_3} \end{bmatrix}$$

The precess noise matrix Q is fixed and the variables $noise_{translation}$ and $noise_{rotation}$ are decided before starting the algorithm depending on the odometry uncertainties. The matrix Q is diagonal with the first three diagonals values equal to $noise_{translation}$ and the last four equal to $noise_{rotation}$:

$$Q = \begin{bmatrix} noise_{translation} & 0 & \cdots & 0 \\ 0 & noise_{translation} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & noise_{rotation} \end{bmatrix}$$

Lastly the covariance sub-matrix P_{rover} is updated using the Jacobian of the prediction model A and the process noise matrix Q [19, 16, 21]:

$$P^N_{rover} = A * P^{N-1}_{rov} * A^T + Q$$

While the cross-covariance sub-matrix of rover-landmark $P_{rover-landmark}$ is updated in this way [19, 16, 21]:

$$P^N_{rover-landmark} = A * P^{N-1}_{rover-landmark}$$

4.3 Removal of the old landmarks

Since there is a limit in the maximum number of landmarks in the state vector s and in the covariance matrix P, the landmarks that has not been observed are deleted to make room for new ones. This process is not common in all the SLAM algorithms.

This procedure starts with the reading of the cameras measurements.

After all the measurements have been read, it is time to check if there is some landmark that is saved in the state vector s but has not been measured.

If this is the case the landmark must be removed from both the state vector s and the covariance sub-matrices. Thus the size of these changes according to the new number of slots taken by the landmarks.

If the landmark that must be removed is not in the last taken slot of the state vector s, then all the landmark saved in the following slots must be translated in the preceding one.

The same procedure must be done for the covariance sub-matrices, except for P_{rover} which has fixed size.

4.4 Correction

The following procedure, that exploits the so called *direct observation model* [19], has to be applied for every landmark that has been measured in this iteration and that was also measured in the previsions one. Incorporation of new landmarks in the next step decreases the computational cost [16]. With this step the uncertainties introduced by the movement of the rover decreases (not only for the rover position and orientation but also for the known landmarks), thanks to the cameras measurements of the surrounding mapped landmarks.

So, once a known landmark has been observed, his 4 measurements (p_x^{left}, p_y^{left}, p_x^{right}, p_y^{right}) are stored in a vector called $z_{measured}$.

Since the computations explained here include a sparse matrix (the Jacobian of the measurement model H, defined below), only the affected part of the state vector s and covariance matrix P are extracted from the full ones and saved respectively in s_c (with length 10) and P_c (with size $10 * 10$). For example, considering the landmark l^N:

$$s_c^N = \begin{bmatrix} c_x \\ c_y \\ \vdots \\ q_3 \\ l_x^N \\ l_y^N \\ l_z^N \end{bmatrix} \qquad P_c^N = \begin{bmatrix} P_{rover} & P_{rover-l^N} \\ \\ P_{rover-l^N}^T & P_{l^N l^N} \end{bmatrix}$$

The first things that must be computed are the predicted measurements. These data is what it is expected to measure given the new predicted position and orientation of the rover. The measurement generation part has been already addressed in 3.2.

Starting from the coordinates of the absolute position of the landmark (they can be found at the end of the state vector s_c) it is possible to convert them to the local reference frame. This can be done subtracting the position of the rover to the absolute position of the landmark and then rotating the resulting vector using the inverse rotation quaternion q^{-1} (2.6.1.5).

At this point the coordinates of the landmark with respect to the local reference frame must also consider the position of the camera. So the local coordinates with respect to the two cameras can be computed adding (or subtracting) half of the distance between the two cameras $\Delta stereo$ to the local x coordinate for the left (or right) camera:

$$\begin{aligned} l_{left} &= l_x + \Delta stereo \\ l_{right} &= l_x - \Delta stereo \end{aligned}$$

As explained in 3.2.5, the Cartesian coordinates can be transformed to spherical coordinates:

$$\begin{aligned} \rho &= \sqrt{l_x^2 + l_y^2 + l_z^2} \\ \theta &= atan2(\sqrt{l_x^2 + l_y^2}, l_z) \\ \phi &= atan2(l_y, l_x^2) \end{aligned}$$

Finally it is possible to get the predicted measurements. This is possible using the same relation used for the measurement generation derived in 3.2.7. To get the x and y pixels coordinates (p_x, p_y) this relation can be applied (once for each camera):

$$p_x = \frac{x_{max} * tan(\phi - \frac{\pi}{2})}{2 * tan(\frac{HFOV}{2})}$$

$$p_y = \frac{y_{max} * tan(\theta - \frac{pi}{2})}{2 * tan(\frac{VFOV}{2})}$$

The vector that contains the 4 predicted measurements (p_x^{left}, p_y^{left}, p_x^{right}, p_y^{right}) computed above for a specific landmark is called $z_{predicted}$.

Also the Jacobian of the observation model H is computed using DA (1.4). Adding an independent DA variable to each of the first 7 variables of state (x, y, z, q_0, q_1, q_2, q_3) and to the 3 coordinates of the considered landmark (l_x, l_y, l_z) before the computation of the predicted measurements it is possible to obtain the Jacobian as the derivative of each measurement (p_x and p_y for both the cameras) with respect to the the 10 DA objects mentioned above:

$$H = \begin{bmatrix} \frac{\partial p_x^{left}}{\partial x} & \frac{\partial p_x^{left}}{\partial y} & \cdots & \frac{\partial p_x^{left}}{\partial l_z} \\ \frac{\partial p_y^{left}}{\partial x} & \frac{\partial p_y^{left}}{\partial y} & \cdots & \frac{\partial p_y^{left}}{\partial l_z} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial p_x^{right}}{\partial x} & \frac{\partial p_x^{right}}{\partial y} & \cdots & \frac{\partial p_x^{right}}{\partial l_z} \end{bmatrix}$$

The measurement noise matrix R changes for every correction step and it's value is based on the distance between the rover and the considered landmark.

The variables $fixed\,noise$ and $proportional\,noise$ are decided before starting the algorithm depending on the cameras performances. The matrix R is diagonal with all the diagonals values equal to:

$$noise = fixed\,noise + distance * proportional\,noise$$

The measurement noise matrix R looks like this:

$$R = \begin{bmatrix} noise & 0 & \cdots & 0 \\ 0 & noise & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & noise \end{bmatrix}$$

For the computation of the Kalman Gain matrix K, it is first needed the measurement matrix Z. The latter is obtained in this way:

$$Z = H * P * H^T + R$$

The mentioned Kalman gain matrix K is a measure of how much the observed landmarks can be trusted. It is obtained as follows:

$$K = P * H^T * Z^{-1}$$

Now it is possible to update both the state vector s_c and the covariance matrix P_c:

$$s_c^N = s_c^{N-1} + K * (z_{measured} - z_{predicted})$$
$$P_c^N = P_c^{N-1} + K * Z * K^T$$

The last thing to do is to update the full state vector s and the covariance matrix P with the updated values of s_c and P_c.

This process must be repeated for every measured known landmark.

4.5 New landmarks initialization

The last step is the new landmarks initialization. During this process all the unknown observed landmarks are appended to the state vector s and to the covariance matrix P.

4.5.1 Mean

The initialization starts with the deduction of the landmarks position in the local reference frame using the cameras measurements, exploiting the so-called *inverse observation model* [19].

For the computation of the Cartesian coordinates (x, y, z) in the local reference frame, starting from the measurements of both the cameras, it is first considered only the x pixels coordinate of the new landmark.

Consider the following figure, which is in the plane $x - y$ of the local reference frame. The red triangle vertices are the two cameras (on the base) and the new landmark at the top [9].

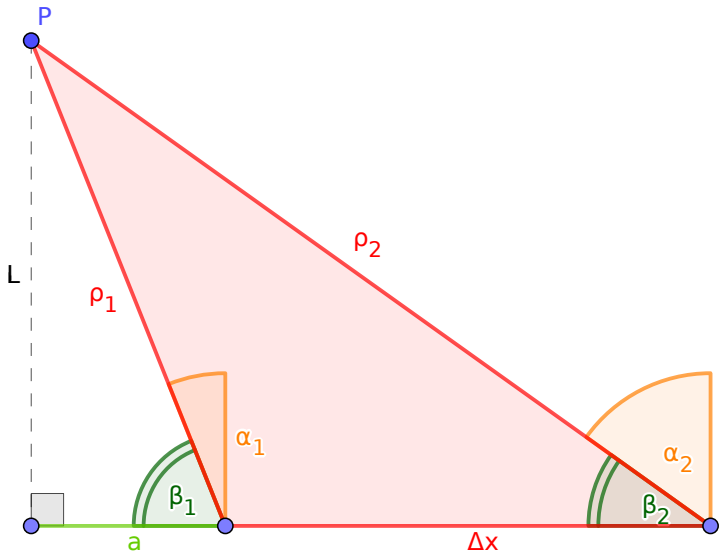


Figure 4.1: Trigonometry of the cameras setup

It has been already explained how to switch from pixels coordinates to spherical coordinates in 3.2.7:

$$cot(\phi - \frac{\pi}{2}) = \frac{x_{max}}{2 * p_x * tan(\frac{HFOV}{2})}$$

Defining the external angles α_1 and α_2 (considered positive like in the figure above) as function of the spherical angle ϕ (3.2.5):

$$\alpha_1 = \phi_1 - \frac{\pi}{2}$$
$$\alpha_2 = \phi_2 - \frac{\pi}{2}$$

It then is possible to obtain the more convenient internal angles β_1 and β_2:

$$\beta_1 = \frac{\pi}{2} - \alpha_1 = \pi - \phi_1$$
$$\beta_2 = \frac{\pi}{2} - \alpha_2 = \pi - \phi_2$$

So the it is possible to write the following relation, valid for both the cameras:

$$tan(\beta) = cot(\alpha)$$

Let's consider the segment L. It can be obtained in two different ways:

$$L = a * tan(\beta_1) \tag{4.1}$$
$$L = (a + \Delta x) * tan(\beta_2) \tag{4.2}$$

Where Δx is the distance between the cameras, so it is two times $\Delta stereo$:

$$\Delta x = 2 * \Delta stereo$$

Dividing (4.1) by (4.2):

$$\frac{a * tan(\beta_1)}{(a + \Delta x) * tan(\beta_2)} = 1$$

Reorganizing the equation:

$$\frac{a}{a + \Delta x} = \frac{tan(\beta_2)}{tan(\beta_1)}$$

To make these equations more readable let's define a new variable r as:

$$r = \frac{tan(\beta_2)}{tan(\beta_1)}$$

The equation becomes:

$$a(1 - r) = r * \Delta x$$

In the end it is possible to get a:

$$a = \frac{r * \Delta x}{1 - r}$$

Applying this equivalence to (4.1) or (4.2) two equivalent relations for L are obtained:

$$L = \frac{r*\Delta x}{1-r} * tan(\beta_1)$$
$$L = (\frac{r*\Delta x}{1-r} + \Delta x) * tan(\beta_2) = \frac{\Delta x}{1-r} * tan(\beta_2)$$

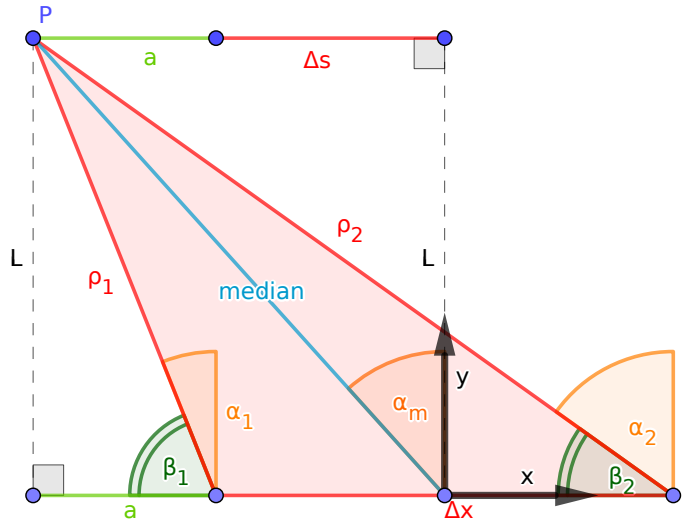


Figure 4.2: Inverse observation model

As it can be seen from the figure, a is linked with the Cartesian coordinate x in the local reference frame, indeed:

$$l_x = -a - \frac{\Delta x}{2}$$

While L coincide with the coordinate y:

$$l_y = L$$

Both x and y Cartesian coordinates have been derived, the z coordinate is still missing. Using the Pythagorean theorem it is possible to compute the spherical coordinates ρ_1 and ρ_2:

$$\rho_1 = \sqrt{a^2 + L^2}$$
$$\rho_2 = \sqrt{(a + \Delta x)^2 + L^2}$$

And using the median theorem it is finally possible to get the distance between the landmark and the rover:

$$median^2 = \frac{1}{2} * (\rho_1^2 + \rho_2^2 - \frac{\Delta x^2}{2})$$

Using the relations for ρ_1 and ρ_2:

$$median = \sqrt{a^2 + L^2 + \frac{\Delta x^2}{4} + a * \Delta x}$$

To make the situation clearer for each camera a new angle ω can be defined, starting from the spherical coordinate θ, as:

$$\omega_1 = \theta_1 - \tfrac{\pi}{2}$$
$$\omega_2 = \theta_2 - \tfrac{\pi}{2}$$

Using the relations obtained in 3.2.7 the tangent of ω can be derived:

$$tan(\omega) = \frac{2 * p_y * tan(\frac{VFOV}{2})}{y_{max}}$$

The tangent of ω is useful to project the median and get the missing Cartesian coordinate z, in the local reference frame:

$$l_z = -median * tan(\omega)$$

Since this relation is valid for both the cameras (and so for both ω_1 and ω_2) to reduce the impact of the measurement noise the algebraic average is considered.

The coordinates referred to the the local reference frame can be switched to the absolute one using the inverse of the several times repeated procedure:

- Rotate the position vector of the new landmark using the inverse of the rotation quaternion, as explained in 2.6.1.5.
- Sum the position vector of the rover to the resulting vector.

The absolute position of the new landmark must be appended at the end of the state vector s:

$$s^N = \begin{bmatrix} s^{N-1} \\ l_x \\ l_y \\ l_z \end{bmatrix}$$

4.5.2 Covariance

To update the covariance sub-matrices with the new sub-matrices $P_{\mathcal{L}\mathcal{L}}$ and $P_{\mathcal{L}x}$ the SLAM specific Jacobians G_R and $G_{y_{n+1}}$ are needed [19].

G_R is made-up from the derivatives of the absolute position of the new landmark with respect to the rover position and orientation ($G_R = \frac{\partial l}{\partial s}$) while $G_{y_{n+1}}$ is made up from the derivatives of the absolute position of the new landmark with respect to its measurements ($G_{y_{n+1}} = \frac{\partial l}{\partial p}$).

Both the Jacobians are computed using DA (1.4), in a similar way to the computation of A (4.2.2) and H (4.4).

Adding an independent DA variable to each of the first 7 variables of state (x, y, z, q_0, q_1, q_2, q_3) and to the 4 measurements of the new landmark (p_x^{left}, p_y^{left}, p_x^{right}, p_y^{right}) before starting the initialization of a new landmark it is possible to obtain the Jacobians as the derivative of the computed absolute position of the new landmark with respect to the first 7 DA objects to get G_R and to the last 4 to get $G_{y_{n+1}}$.

$$G_R = \begin{bmatrix} \frac{\partial l_x}{\partial x} & \frac{\partial l_x}{\partial y} & \cdots & \frac{\partial l_x}{\partial q_3} \\ \frac{\partial l_y}{\partial x} & \frac{\partial l_y}{\partial y} & \cdots & \frac{\partial l_y}{\partial q_3} \\ \frac{\partial l_z}{\partial x} & \frac{\partial l_z}{\partial y} & \cdots & \frac{\partial l_z}{\partial q_3} \end{bmatrix}$$

$$G_{y_{n+1}} = \begin{bmatrix} \frac{\partial l_x}{\partial p_x^{left}} & \frac{\partial l_x}{\partial p_y^{left}} & \frac{\partial l_x}{\partial p_x^{right}} & \frac{\partial l_x}{\partial p_y^{right}} \\ \frac{\partial l_y}{\partial p_x^{left}} & \frac{\partial l_y}{\partial p_y^{left}} & \frac{\partial l_y}{\partial p_x^{right}} & \frac{\partial l_y}{\partial p_y^{right}} \\ \frac{\partial l_z}{\partial p_x^{left}} & \frac{\partial l_z}{\partial p_y^{left}} & \frac{\partial l_z}{\partial p_x^{right}} & \frac{\partial l_z}{\partial p_y^{right}} \end{bmatrix}$$

The sub-matrices $P_{\mathcal{L}\mathcal{L}}$ and $P_{\mathcal{L}x}$ must be appended to the current covariance matrix P in this way:

$$P^N = \begin{bmatrix} P^{N-1} & P_{\mathcal{L}x}^T \\ \\ P_{\mathcal{L}x} & P_{\mathcal{L}\mathcal{L}} \end{bmatrix}$$

And they can be computed as reported below:

$$P_{\mathcal{LL}} = G_R * P_{rover}^{N-1} * G_R^T + G_{y_{n+1}} * R * G_{y_{n+1}}^T$$

$$P_{\mathcal{L}x} = G_R * \begin{bmatrix} P_{rover}^{N-1} & P_{rover-landmark}^{N-1} \end{bmatrix}$$

Where R is the measurement noise matrix defined in 4.4.

Chapter 5

Numerical results

In this chapter two different cases are considered: first a reconstruction of the position and the orientation of the rover using only the wheels encoders and the gravity sensor (so with a simple integration of the odometry) and then using the full SLAM algorithm. An high slip surface is simulated with a Gaussian error $\mathcal{N}(o, \sigma_o)$ with 0 mean and a variance $\sigma_o = 0.01$ on the distance measured by the left and right wheels. The numerical results for both the cases are reported and analyzed below.

5.1 Odometry only

First let's analyze the trajectory reconstructed by the integration of the odometry. As said before, the odometry is based on the distance traveled by the left and right wheels, measured with wheels encoders. Every 3 steps the orientation is recovered also thanks to the gravity sensor, which measures the normal to the surface. All the measurements are subjects to Gaussian errors.

The correct trajectory (the circular one) is plotted in red while the reconstructed one is plotted in cyan. The x and y axes are expressed in meters.

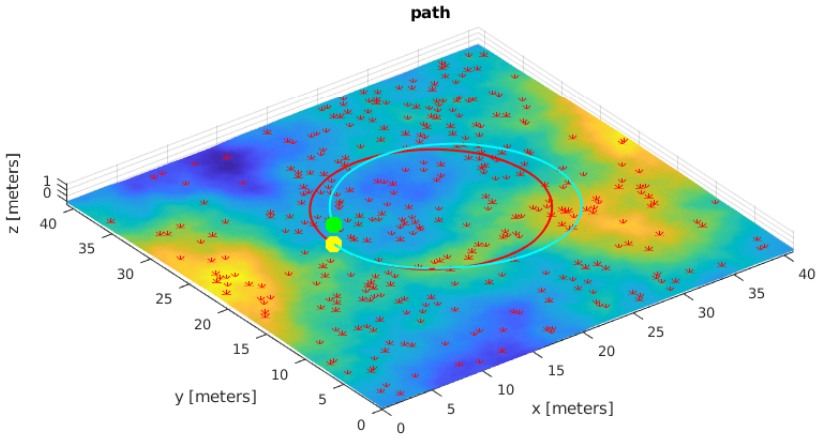


Figure 5.1: Trajectory reconstructed with odometry only

The error, computed as the distance between the reconstructed position and the correct one, is plotted below for every iteration. The error is expressed in meters.

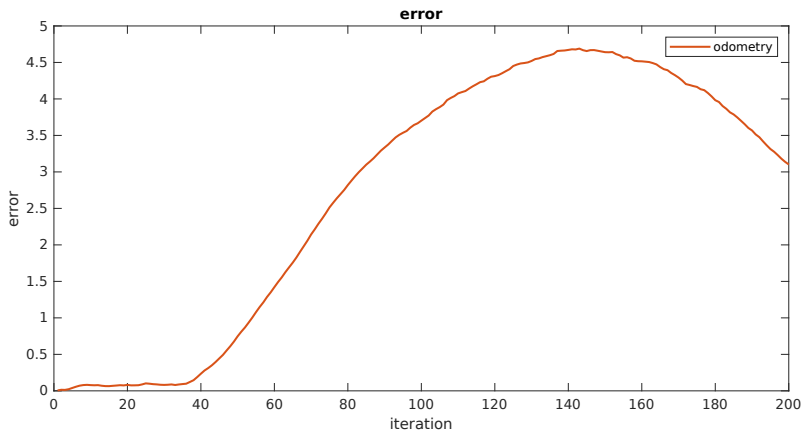


Figure 5.2: Error with odometry, step-by-step

To quantify the drift error in the estimation of the position it is possible to observe the maximum value of the error. In this case the maximum error is about $4.69\,m$.

5.2 SLAM algorithm

Now let's analyze the same trajectory with the use of the SLAM algorithm. As written before, the SLAM algorithm has access, every 3 odometry integration steps, to the cameras measurements (left and right camera) in addition to the odometry data. A maximum number on the landmarks stored in the state vector is imposed, in this case 10, to reduce the computational cost of the algorithm.

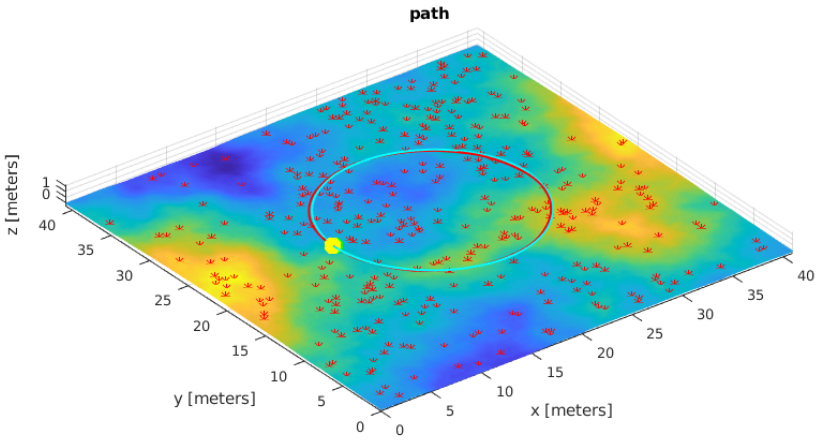


Figure 5.3: Trajectory reconstructed with the SLAM algorithm

Also in this case the error is plotted below for every iteration. Like before, the error coincide with the distance between correct position and recovered one and is expressed in meters.

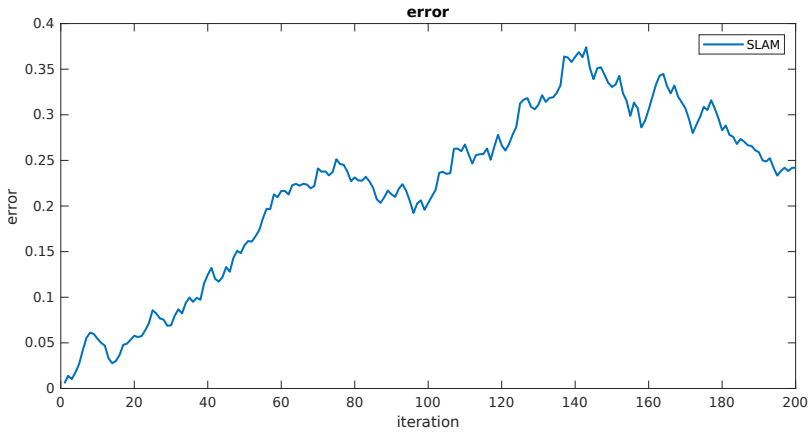


Figure 5.4: Error with the SLAM algorithm, step-by-step

	Maximum error in m
SLAM algorithm	0.374
Odometry integration	4.69

Table 5.1: Comparison of the maximum error

The maximum error is now more than an order of magnitude smaller. Indeed it is about $0.374\,m$. To visualize better this difference a comparison of the two errors plot, one for the SLAM algorithm and one for the integration of odometry is reported below.

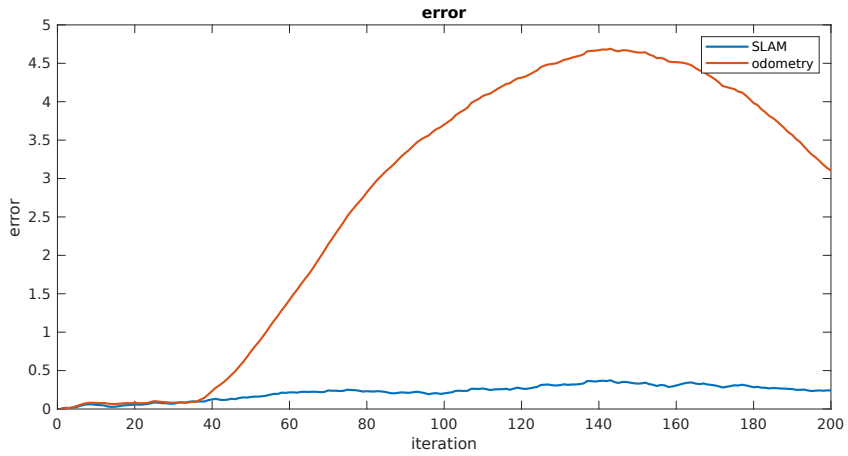


Figure 5.5: Comparison of the errors with SLAM and with odometry

The error for the odometry only case is very similar to the SLAM algorithm one for about 35 iterations and then the two errors gradually get away. This happens because, in the odometry only case, an erroneous orientation generates a drift which is not corrected by any feedback. This can also be observed in the trajectory plot. The error of the odometry only case mainly derives from an inaccurate orientation. Indeed the shape of the reconstructed trajectory is not very different from the correct one, which is circular, but there is not overlap between them.

To ensure that not only the mean value but also the uncertainties for both the rover and the landmarks are correctly estimated, it is possible to study the sub-matrices of covariance. Indeed using their value, the error ellipse can be plotted, to check if the mean values are inside this region [18]. In a 3D case the error ellipse is actually an ellipsoid. The error ellipsoid can be plotted for different confidence values [10]. In the figures below

it is computed for a confidence of 70%. Thus this error ellipsoid defines the region that contains 70% of all samples computed from a Gaussian distribution.

It is first checked the position of the rover. Below there are provided two plots of the ellipsoid in three different points of the trajectory: at $\frac{1}{3}$, at $\frac{2}{3}$ and at the end. The yellow point is the correct position, while the green one is the estimated one. The ellipsoid is centered around the estimated position.

If the uncertainties are computed correctly the correct position should be inside the ellipsoid. To compute the ellipsoid the covariance of the rover P_{rover} has been used.

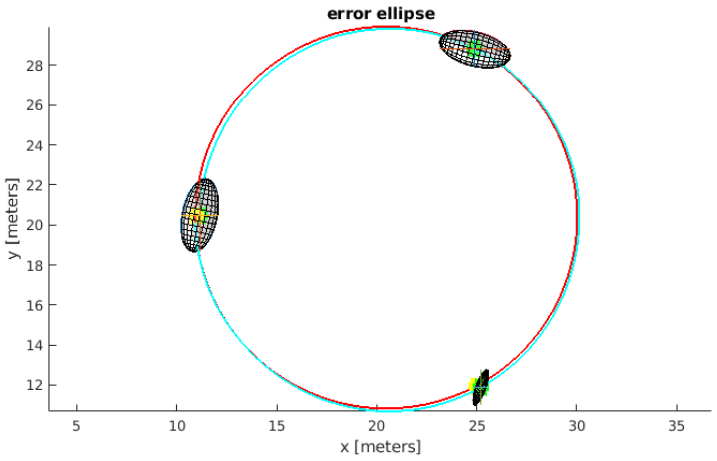


Figure 5.6: Error ellipsoids of the position of the rover, plane $x - y$

Two things stand out: first that the size of the ellipsoid is growing going on with the trajectory and then that the ellipsoid has a z axis much smaller than its x and y axes. This is coherent with the trajectory, which has only small variations in the z direction, due to the uneven surface.

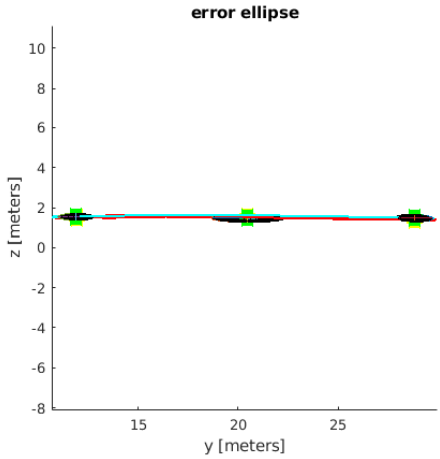


Figure 5.7: Error ellipsoids of the position of the rover, plane $y - z$

It can be concluded that for all the three cases the correct point is inside the error ellipsoid for a confidence level of 70%. This is a good indicator that the covariance of the rover P_{rover} is estimated correctly.

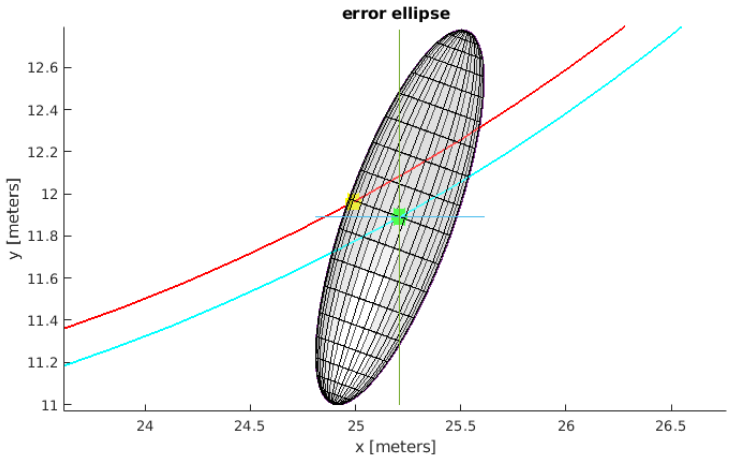


Figure 5.8: Error ellipsoid of the position of the rover at $\frac{1}{3}$ of its trajectory

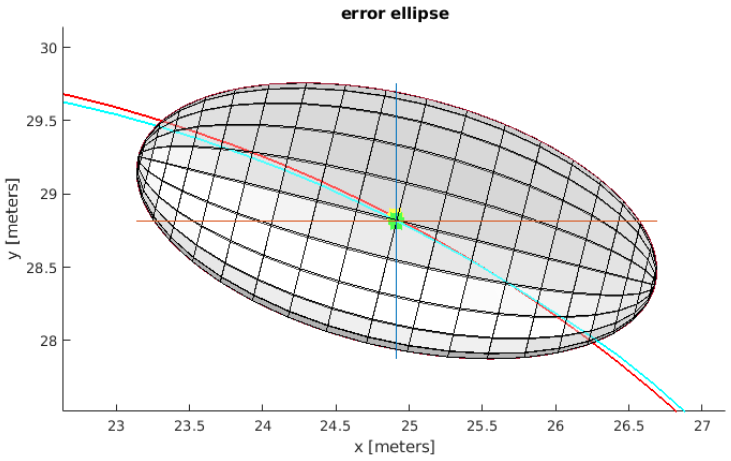


Figure 5.9: Error ellipsoid of the position of the rover at $\frac{2}{3}$ of its trajectory

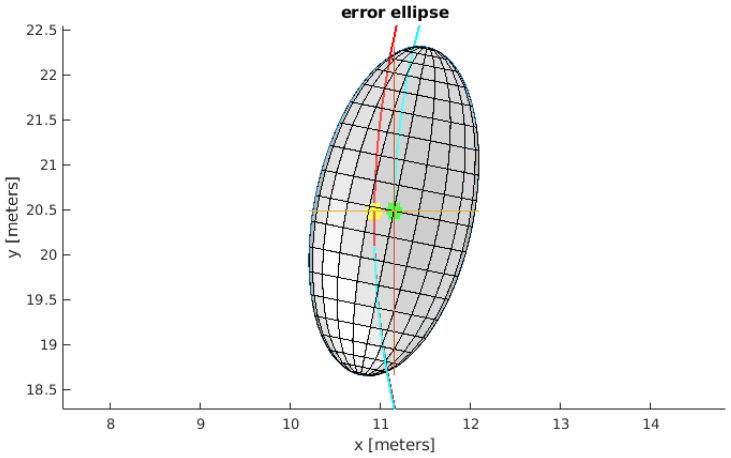


Figure 5.10: Error ellipsoid at the end of the trajectory

It is a good practice to check also if the landmarks are inside their error ellipsoid. Only the last step of the trajectory is selected for this check. To compute this ellipsoid the covariance of the landmark $P_{landmark}$ has been used.

The landmarks are plotted as little stars in red. They can be viewed better in the third figure below.

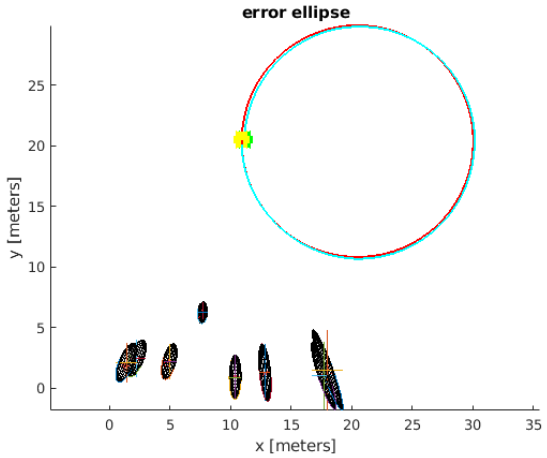


Figure 5.11: Error ellipsoids of the landmarks, plane $x - y$

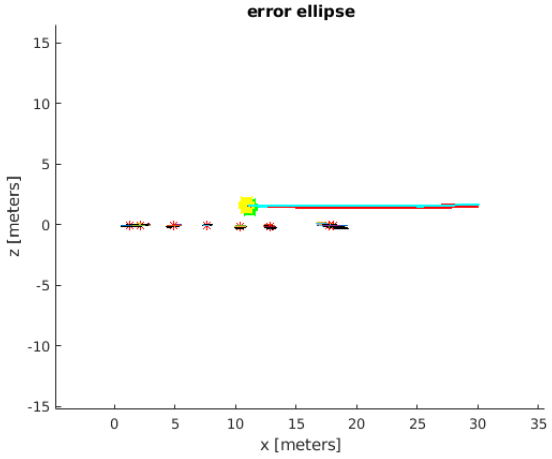


Figure 5.12: Error ellipsoids of the landmarks, plane $x - z$

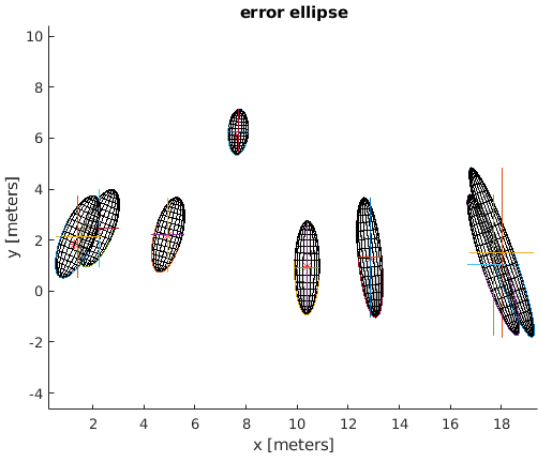


Figure 5.13: Error ellipsoid of the landmarks, closer view

Also in this case the landmarks are inside their ellipsoids. It can be observed how the shape of the ellipsoids strongly depends on the position of the rover. Moreover just like the ellipsoid of the position of the rover, also these have a much smaller z axis compared to the other two.

Since the orientation of the rover is not directly visible from the figures above, a plot of the position (x, y, z) and orientation (q_0, q_1, q_2, q_3) for every correction step of the SLAM algorithm is provided below.

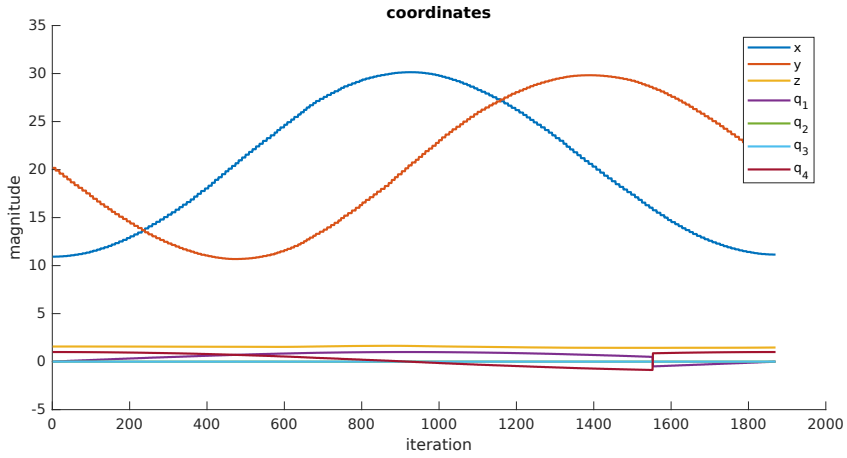


Figure 5.14: Position and orientation of the rover, step-by-step

Chapter 6

Conclusions

6.1 Conclusions

As expected, the SLAM algorithm gives a big advantage with respect to an odometry only based approach. The error of the first is, indeed, about an order of magnitude smaller compared to the last one on a high slip surface, thanks to smaller drift error in the estimate. The SLAM algorithm gives also the advantage of reconstructing a map with all the observed landmarks.

In addition to the mean value, also the uncertainties are correctly estimated. This has been proven for both the position of the rover and of the landmarks.

6.2 Future developments

In the proposed SLAM algorithm the growing computational cost needed for a large area has been worked out with the removal of old landmarks. Even if it is a simple solution, it is not the best one since it induces a drift error due to the new initialization (with bigger uncertainties) of an already measured landmark.

A more complex solution is what is called a segmented SLAM [8]. These algorithms can divide the explored area into smaller parts and, depending on the position and orientation of the sensors, use only a convenient subset of the full set of measured landmarks [1].

As written in 1.2, an advantage of building a map and keeping stored all the observed landmarks is that the SLAM algorithm may have global consistency. This means that the algorithm can recognize a previously mapped landmark and close a loop, reducing the drift error.

Also, the DA implemented in this work can be used to develop an high order SLAM algorithm using a truncated Taylor expansion representation for the covariance and the Isserlis formula for its propagation. This has already been done for an EKF in [23], [22] and [5].

The main problem with this approach is that, to be convenient, it needs a more complex and precise non-linear prediction of the position and orientation of the rover than the one used in this work. This is an hard task since for a low speed rover the dynamic is almost absent and also because the explored surface is unknown a priori.

Appendix A

Run the code

This appendix is a small guide on how to run the code used in this work.

The code has been written on a machine running a Linux distribution (Arch Linux) updated to the latest package available in the repositories during the development (late 2018 / early 2019).

The following installation tutorial is based on this setup but should work flawlessly on every updated Linux machine.

A.0.1 Install the DACE library

The DACE library can be download from `https://github.com/dacelib/dace` cloning the GitHub repository:

```
git clone "https://github.com/dacelib/dace.git"
```

Actually in the repository there is an `ArchLinux` folder containing a `PKGBUILD` file to easily install the library on an Arch Linux machine, but this approach has not been followed because of an experimental feature that must be enabled.

Indeed, before installing, the `CMakeLists.txt` file has to be changed to enable the experimental Algebraic Matrix feature. This can be done changing this line:

```
option(WITH_ALGEBRAICMATRIX "Include the AlgebraicMatrix
type (experimental)" OFF)
```

To this one:

```
option(WITH_ALGEBRAICMATRIX "Include the AlgebraicMatrix
type (experimental)" ON)
```

The installation process is very simple and the steps are reported here:

```
cd dace
mkdir build
cd build
cmake ..
make
sudo make install
```

The last thing to do is to make sure that the path to the library (`/usr/local/lib`) is present in the `/etc/ld.so.conf` file. If it is not, it must be appended.

That process can be also done running the following BASH script:

```
if grep -q /usr/local/lib "/etc/ld.so.conf"; then
  echo 'nothing to do' > /dev/null
else
  echo '/usr/local/lib' | sudo tee --append \
  /etc/ld.so.conf > /dev/null
fi
```

A.0.2 Generate the surface and the landmarks

The surface and the landmarks can be generated with the MATLAB script `surface_generator.m`. The inputs used (like the dimension of the map, its number of pixels and the steepness) are the same described in 3.1 and can be changed in the preamble of the script.

Before the execution it is required to change the MATLAB current folder with the folder containing the file.

The script automatically plots the generated surface with the landmarks and saves a picture of them. The inputs and the data of the environment (such as the markers position, the surface height for every point and the normal vectors) are saved in plain text files so that they can be read by the C++ programs.

Also the workspace is saved to be then used by the scripts that plots and analyze the results.

A.0.3 Generate the odometry data and the measurements

The odometry and measurement generator has been written in C++ and it's called `simulator.cpp`.

Also in this case the inputs are found in the preamble and can be changed before the compilation of the program.

`simulator.cpp` can be compiled with GCC (which is required) and then executed with this command:

```
g++ -ldace ./simulator.cpp && ./a.out
```

The program reads the inputs and the data of the environment written by `surface_generator.m`.

Its results and inputs also useful for the SLAM algorithm are saved in plain text files.

A.0.4 Run the SLAM algorithm

Also the SLAM algorithm `SLAM.cpp` has been written in C++, so its execution is similar to the program above.

The inputs are found in the first part of the code and it can be compiled and executed with this command:

```
g++ -ldace ./SLAM.cpp && ./a.out
```

The results, together with other data used to analyze the performance of the code, are saved in plain text files.

A.0.5 Plot and analyze the results

Before the execution of both the script it is required to change the MATLAB current folder with the folder containing the files.

To plot the trajectory computed by the SLAM algorithm the MATLAB script `plots.m` can be used.

Likewise `surface_generator.m`, also `plots.m` saves the resulting plot in the current folder.

To analyze the position error and the most relevant matrices at the various time-steps the MATLAB script `data.m` can be used.

Before its execution it is required to manually update the maximum number of landmarks to match the input used in `SLAM.cpp`. This script automatically plots 5 different figures.

Bibliography

[1] Tim Bailey and Huch Durrant-White. Simultaneous localisation and mapping (slam): Part ii state of the art, 2006.

[2] Martin Berz. Differential algebraic techniques, entry in handbook of accelerator physics and engineering, 1998.

[3] Martin Berz. *Advances in Imaging and Electron Physics*, volume 108. Academic Press, 1999.

[4] Martin Berz and Kyoko Makino. Cosy infinity version 9 reference manual, 2006.

[5] Francesco Cavenago, Mauro Massari, Pierluigi Di Lizia, and Alexander Witting. Assessment of onboard da state estimation for spacecracft relative navigation, 2017.

[6] Peter Corke, Dennis Strelow, Sanjiv Singht, L.H. Matthies, and S.I. Roumeliotis. Omnidirectional visual odometry for a planetary rover, 2004.

[7] Huch Durrant-White, Fellow, IEEE, and Tim Bailey. Simultaneous localisation and mapping (slam): Part i the essential algorithms, 2006.

[8] Nathaniel Fairfield, David Wettergreen, and George Kantor. Segmented slam in three-dimensional environments, 2010.

[9] Sanja Fidler. Csc420: Intro to image understanding, 2018.

[10] Michael John Romer Healy. Drawing a probability ellipse, 1972.

[11] D.M. Helmick, Yang Cheng, and D.S. Clouse. Path following using visual odometry for a mars rover in high-slip environments, 2004.

[12] Niklas Karlsson, Enrico Di Bernardo, Jim Ostrowski, Luis Goncalves, Paolo Pirjanian, and Mario E. Munich. The vslam algorithm for robust localization and mapping, 2005.

[13] NASA Jet Propulsion Laboratory. Mars science laboratory: Curiosity rover, 2013.

[14] Edwin Olson. A primer on odometry and motor control, 2005.

[15] Nur Aqilah Othman and Hamzah Ahmad. The analysis of covariance matrix for kalman filter based slam with intermittent measurement, 2013.

[16] Søren Riisgaard and Morten Rufus Blas. Slam for dummies, 2005.

[17] Simone Servadio. High order filters for relative pose estimation of an uncooperative target. Master's thesis, Politecnico di Milano, 2017.

[18] Randall C. Smith and Peter Cheeseman. On the representation and estimation of spatial uncertainty, 1986.

[19] Joan Solà. Simultaneous localization and mapping with the extended kalman filter, 2014.

[20] Byron D. Tapley, Bob E. Schutz, and George H. Born. *Statistical Orbit Determination*. Academic Press, 2002.

[21] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. The MIT Press, 2005.

[22] Monica Valli, Roberto Armellin, Pierluigi Di Lizia, and Michèle Lavagna. Nonlinear filtering methods for spacecraft navigation based on differential algebra, 2012.

[23] Monica Valli, Roberto Armellin, Pierluigi Di Lizia, and Michèle Lavagna. Nonlinear mapping of uncertainties in celestial mechanics, 2013.

[24] Dae Hee Won, Sebum Chun, Sangkyung Sung, Taesam Kang, and Young Jae Lee. Improving mobile robot navigation performance using vision based slam and distributed filters, 2008.

[25] Khalid Yousif, Alireza Bab-Hadiashar, and Reza Hoseinnezhad. An overview to visual odometry and visual slam: Applications to mobile robotics, 2015.