



ARISTOTLE UNIVERSITY OF THESSALONIKI  
FACULTY OF ENGINEERING  
School of Electrical and Computer Engineering  
Department of Electronics and Computers

---

# Globally Adaptive SLAM for Autonomous Planetary Rovers using Map Matching

---

*Author:*

Dimitrios GEROMICHALOS

*Supervisors:*

Assoc. Prof. Loukas PETROU<sup>1</sup>

Dr. Emmanouil TSARDOULIAS<sup>1</sup>

Martin AZKARATE<sup>2</sup>

<sup>1</sup>*Aristotle University of Thessaloniki*

<sup>2</sup>*European Space Agency*

*A thesis submitted to the Faculty of Engineering in  
partial fulfillment of the requirements for the degree of*

Diploma  
in  
Electrical and Computer Engineering

Thessaloniki,  
February 2018



# Abstract

Rover autonomy is of paramount significance in a planetary exploration mission, since low supervision from the ground station can directly increase the scientific outcomes. However, the autonomous long range navigation of robotic vehicles raises several challenges, especially in such context where the communication to Earth is limited. An essential skill such vehicle must possess in order to navigate in an unknown environment safely is the accurate perception of its surroundings.

This thesis proposes an approach for real-time Simultaneous Localization and Mapping (SLAM) in extreme terrains, that takes into account the limited computational resources of planetary rovers. By employing scan matching, particle filter and elevation mapping techniques, we construct a system that is able to provide an accurate pose estimation and a high resolution representation of the environment which can be utilized for navigation purposes.

Moreover, the current thesis addresses the problem of absolute localization that emerges due to the accumulated drift in position and orientation. Building upon existing image matching techniques, we formulate a novel method that is capable of locating a local (robot-centric) map in an a priori global map with the aim to correct the robot's pose in a global reference frame. Additionally, the developed matching technique is evaluated against possible local and global map resolution values so as to provide a baseline for future applications.

Finally, we implement and integrate our approach on a planetary rover testbed as a means to validate the performance of the unified system. The experimental validation is performed in long range scenarios in a terrain that bears close resemblance to a planetary one.



# Contents

<b>Abstract</b>	iii
<b>Contents</b>	v
<b>List of Figures</b>	vii
<b>List of Tables</b>	ix
<b>List of Abbreviations</b>	xiii
<b>1 Introduction</b>	1
1.1 Motivation . . . . .	1
1.2 Problem Statement . . . . .	1
1.2.1 Presumptions . . . . .	2
1.3 Literature Review . . . . .	3
1.3.1 Simultaneous Localization and Mapping . . . . .	3
1.3.2 Planetary Absolute Localization . . . . .	5
Skyline-based . . . . .	5
Feature-based . . . . .	5
1.4 Thesis Objectives and Outline . . . . .	6
1.4.1 Research Objectives . . . . .	6
1.4.2 Outline . . . . .	7
<b>2 Globally Adaptive SLAM</b>	9
2.1 System Overview . . . . .	9
2.2 Data Registration . . . . .	10
2.2.1 Point Cloud Preprocessing . . . . .	11
2.2.2 Registration . . . . .	13
Structure of Elevation Map . . . . .	13
Map Prediction and Update . . . . .	14
2.2.3 Sensor Fusion . . . . .	15
2.3 Pose Estimation . . . . .	16
2.3.1 Initialization . . . . .	17
2.3.2 Prediction . . . . .	18
2.3.3 Update . . . . .	19
2.3.4 Resampling . . . . .	19

2.3.5	Estimation . . . . .	20
2.4	Pose Correction . . . . .	21
2.4.1	Global to Local Map Matching . . . . .	22
2.4.2	Criteria Checking . . . . .	25
	Traversed Distance Criterion . . . . .	25
	Environment Structure Criterion . . . . .	26
<b>3</b>	<b>Implementation and Tools</b>	<b>27</b>
3.1	Planetary Rover Testbed . . . . .	27
3.2	Overall System Architecture . . . . .	28
3.3	Robotic Software Framework . . . . .	29
3.4	Orbiter Data Preprocessing . . . . .	30
<b>4</b>	<b>Experimental Validation</b>	<b>33</b>
4.1	Scope of Experiments . . . . .	33
4.1.1	Data Collection . . . . .	33
4.1.2	Metrics . . . . .	34
4.2	Experiments on Pose Estimation . . . . .	35
4.2.1	Relative Localization Results . . . . .	35
4.3	Experiments on Global Map Matching . . . . .	38
4.3.1	Absolute Localization Results . . . . .	38
4.3.2	Map Resolution Viability . . . . .	40
<b>5</b>	<b>Conclusion</b>	<b>43</b>
5.1	Thesis Summary . . . . .	43
5.2	Contributions . . . . .	43
5.3	Directions for Future Extensions . . . . .	44
5.4	Applications . . . . .	44
	<b>Bibliography</b>	<b>47</b>

# List of Figures

1.1	Mars landing sites . . . . .	2
2.1	High level design diagram . . . . .	10
2.2	Pose estimation activity diagram . . . . .	18
3.1	Heavy duty planetary rover . . . . .	28
3.2	System component diagram . . . . .	29
3.3	ROCK component interface . . . . .	30
3.4	Mars view from HiRISE . . . . .	31
4.1	Aerial view of test field . . . . .	34
4.2	Terrain of first experiment . . . . .	35
4.3	Traverse of first experiment . . . . .	36
4.4	Particle filter parameter influence . . . . .	37
4.5	Terrain of second experiment . . . . .	38
4.6	Traverse of second experiment . . . . .	39
4.7	Global gradient map for pose correction . . . . .	39
4.8	Pose correction error over time plot . . . . .	40
4.9	Global map resolution comparison . . . . .	41



# List of Tables

4.1	Metrics on parameter sets for relative localization . . . . .	37
4.2	Results of pose correction for absolute localization . . . . .	40
4.3	Local and global map resolution viability . . . . .	42



# List of Algorithms

1	Registration of point cloud to map . . . . .	15
2	Initialization of particle filter . . . . .	17
3	Estimation of final pose from particle population . . . . .	21



# List of Abbreviations

<b>GA</b>	Globally Adaptive
<b>SLAM</b>	Simultaneous Localization And Mapping
<b>DEM</b>	Digital Elevation Map
<b>ICP</b>	Iterative Closest Point
<b>KF</b>	Kalman Filter
<b>PF</b>	Particle Filter
<b>PC</b>	Point Cloud
<b>SOR</b>	Statistical Outlier Removal
<b>CV</b>	Computer Vision
<b>TM</b>	Template Matching
<b>SDD</b>	Sum of Square Difference
<b>CCORR</b>	Cross Correlation
<b>CCOEFF</b>	Correlation Coefficient
<b>ROCK</b>	Robot Construction Kit
<b>RTT</b>	Real Time Toolkit
<b>ROS</b>	Robot Operating System
<b>ESA</b>	European Space Agency
<b>NASA</b>	National Aeronautics and Space Administration
<b>HIRISE</b>	High Resolution Imaging Science Experiment
<b>HDPR</b>	Heavy Duty Planetary Rover
<b>GPS</b>	Global Positioning System
<b>TOF</b>	Time Of Flight
<b>LIDAR</b>	Light Detection And Ranging
<b>IMU</b>	Inertial Measurement Unit
<b>SIL</b>	Software In the Loop
<b>MSE</b>	Mean Square Error
<b>RMSD</b>	Root Mean Square Deviation



*Dedicated to Fi*



## Chapter 1

# Introduction

### 1.1 Motivation

A space exploration vehicle designed to move across the surface of a planet, besides Earth, is called a planetary rover as of its ability to "rove over" an unknown terrain. Planetary rovers are used to collect valuable data and samples such as pictures, dust and rocks, which are later analyzed by scientists to enrich our understanding regarding the universe.

Past missions have sent robots to the Moon and Mars (Figure 1.1) with the NASA's Curiosity being the last rover to land on Mars in 2011. Future missions do not differ much, as new lunar rovers are about to launch in the next years and Mars rovers are currently under development scheduled to be sent to Mars in 2020.

Such missions utilize a great amount of time and effort to build, launch and finally operate. Our interest lies in the operation challenges that are caused by the fact that no constant communication can be achieved. As an example, commanding messages can take up to 21 minutes to travel between the Earth and Mars and bandwidth limitations can limit the number of messages that can be sent. This compounded to the unpredictable and unsafe conditions of an unknown terrain such as the Mars surface which is full of rocks, could lead to destructive system crashes of a costly mission.

Thus, we emphasize the need of a real time platform that will provide the capability of autonomous operation with low supervision from ground control as far as navigation is concerned.

### 1.2 Problem Statement

In order to navigate autonomously in extreme terrains, rovers need to accurately perceive the 3D environment around them based on sensory information. Terrains that classify as extreme are areas with high variance in elevation and great morphological anomalies, such as rocks and craters.

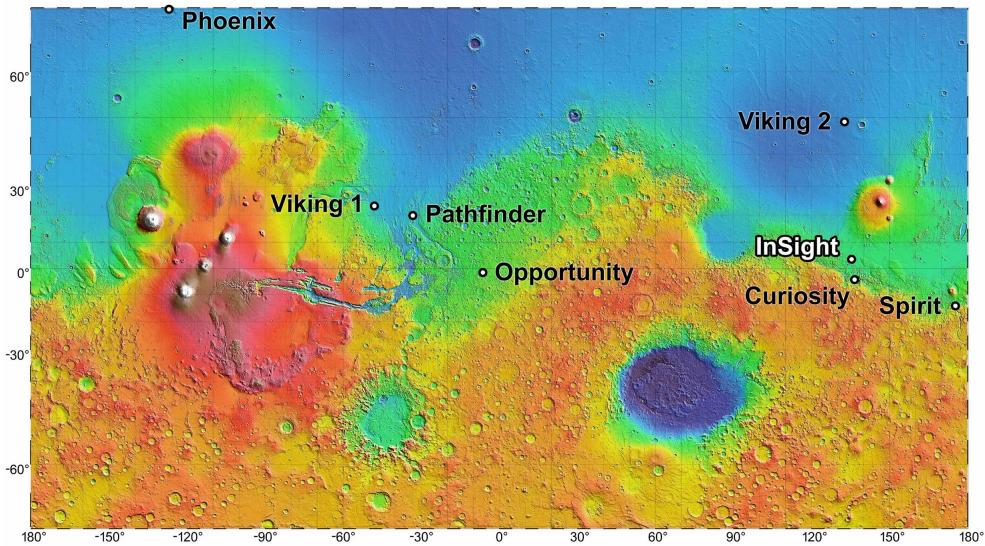


FIGURE 1.1: Mars landing sites of rovers and landers from NASA’s past missions.

This thesis actively researches the challenges of perception in such terrains in a planetary context and mainly focuses on two problems. The first is the relative localization of the robot and the local mapping of its surroundings in order to provide autonomous navigation capabilities to it. The second problem is the global localization of the robot, which is the elimination of the accumulated drift from relative localization in long range scenarios.

Specifically, given as inputs: (i) an initial prediction about the robot’s movement, (ii) a detailed 3D representation of the close surroundings of the robot and (iii) an a priori low resolution map of the greater area, the purpose is to output: (i) an accurate estimation of the position and orientation of the robot in a global reference frame and (ii) a local (robot-centric) map that represents the height of the environment.

### 1.2.1 Presumptions

In order to tackle the aspect of the problem that is of most interest, we first assume the following:

- a low resolution global map is available to the system at any time
- the initial global position of the robot is known
- the environment is static, i.e. is does not contain any moving objects
- the environment is unknown, i.e. there are no high resolution a priori maps available
- the terrain is single layered, i.e. there are no bridges or caves

## 1.3 Literature Review

### 1.3.1 Simultaneous Localization and Mapping

In early robotics research, localization and mapping were tackled independently. This is related to the fact that they have a cyclic inter-dependency. On the one hand, a map can only be created when the robot's pose is known. On the other hand, we need an accurate map representation in order to perform localization. Nevertheless, both tasks usually need to be performed simultaneously: in this case we speak of Simultaneous Localization and Mapping (SLAM).

SLAM problems can be distinguished in several ways:

- **Volumetric versus feature-based:** In volumetric SLAM, the map is sampled at a resolution high enough to allow for photo-realistic reconstruction of the environment. The map in volumetric SLAM is usually quite high-dimensional, with the result that the computation can be quite involved. Feature-based SLAM extracts sparse features from the sensor stream. The map is then only comprised of features. Feature-based SLAM techniques tend to be more efficient, but their results may be inferior to volumetric SLAM due to the fact that the extraction of features discards information in the sensor measurements.
- **Topological versus metric:** Some mapping techniques recover only a qualitative description of the environment, which characterizes the relation of basic locations. Such methods are known as topological. A topological map might be defined over a set of distinct places and a set of qualitative relations between these places. Metric SLAM methods provide metric information between the relation of such places.
- **Known versus unknown correspondence:** The correspondence problem is the problem of relating the identity of sensed objects to other sensed objects. The problem of estimating the correspondence is known as data association problem. It is one of the most difficult problems in SLAM.
- **Static versus dynamic:** Static SLAM algorithms assume that the environment does not change over time. Dynamic methods allow for changes in the environment. The vast literature of SLAM methods assumes static environments. Dynamic effects are often treated just as measurement outliers. Methods that reason about motion in the environment are more involved, but they tend to be more robust in more applications.
- **Small versus large uncertainty:** SLAM problems are distinguished by the degree of location uncertainty they can handle. The most simple SLAM algorithms allow only for small errors in the location estimate. They are good in situations where a robot goes down a path that does not intersect itself, and then returns along the same path. In many environments it is possible to reach

the same location from multiple directions. Here the robot may accumulate a large amount of uncertainty. This problem is known as the "loop closing problem". When closing a loop, the uncertainty may be large. The ability to close loops is a key characteristic of modern-day SLAM algorithms. The uncertainty can be reduced if the robot can sense information about its position in some absolute coordinate frame, e.g., through the use of a satellite-based global positioning receiver (GPS).

- **Active versus passive:** In passive SLAM algorithms, some other entity controls the robot and the SLAM algorithm is purely observing. The vast majority of algorithms are of this type; they give the robot designer the freedom to implement arbitrary motion controllers and pursue arbitrary motion objectives. In active SLAM, the robot actively explores its environment in the pursuit of an accurate map. Active SLAM methods tend to yield more accurate maps in less time, but they constrain the robot motion. There exist also hybrid techniques in which the SLAM algorithm controls only the pointing direction of the robot's sensors, but not the motion direction.
- **Single robot versus multi-robot:** Most SLAM problems are defined for a single robot platform, although recently the problem of multi-robot exploration has gained in popularity. In some, robots get to observe each other, in others, robots are told their relative locations.
- **Full vs online** In the full-SLAM problem the entire path is calculated and the map is constructed after data acquisition is done, while in the online-SLAM the previous poses are a continuous estimate and a map is built on the fly. Both approaches have their own merits. Full-SLAM can be said to be easier, as the data only needs to be processed and often no real-time requirements need to be fulfilled, while the online-SLAM is more useful if the map is also incorporated in the robot navigation system.

Popular approximate solution methods include particle filters, extended Kalman filters (EKF), and graph-based solutions.

Particle filters or Sequential Monte Carlo (SMC) filters, are a set of genetic type statistical approaches to solving the filtering problem. The particle filter is a tool for tracking the state of a dynamic system, even when the state is not fully observable. Particle filters is in practice a Bayes filter that uses a prediction and update cycle to estimate the state of a dynamical system from sensor measurements, and has similar applications as the Kalman filter, but handles large dimensionality better.

Interestingly this approach can be used to solve both the online and full-SLAM problem simultaneously using a grid-map based approach. Particle filters has become one of the most popular approaches to solving the SLAM problem in modern robotics.

Another popular approach for feature based maps is the Extended Kalman Filter, which applies the EKF to online-SLAM using maximum likelihood data association. The EKF assumes the noise to be Gaussian for motion and perception. EKF integrates out the robots pose as the robot moves, and therefore only keeps track of the current pose.

Finally, the full-SLAM problem can be solved using graph based approaches such as Graph-SLAM. In Graph-SLAM the robot poses are represented as a graph where the nodes correspond to the poses of the robot at different points in time, and the edges represent constraints between the poses. The latter are obtained from observations of the environment or from movement actions carried out by the robot. The whole problems boil down to solving a large optimization problem. The graph can be shown to be a sparse graph of nonlinear constraints. Graph-SLAM needs to keep track of all poses and measurements at all times, but since it is an offline approach, it does not need to perform computations while collecting data.

### 1.3.2 Planetary Absolute Localization

Precise localization in a planetary terrain without any communication assistance from the Earth is considered an essential feature in long-range scenarios (e.g. 10 km and up). A series of matching techniques have been examined in order to align the global and local maps.

#### Skyline-based

Skyline based methods attempt to estimate the global position of a rover by matching the skyline of the horizon, with skylines extracted from the global map. The VIPER algorithm (Cozman et al., 2000) is a global localization algorithm that is matching the horizon skyline signature captured by a panoramic picture acquired by the rover to predicted skylines signatures at various positions on the global map. The VIPER algorithm has been proven efficient in the "lost in space situation", where the rover's initial position is not known, and is rather quick because it is precomputes offline the skylines of the global map. It yields an estimation accuracy of 100 m to 150 m on a 30 m/pixel digital elevation map (DEM). However, this approach is not behaving smoothly in cases where the horizon is hardly detectable and when the DEM has not covered an area large enough to allow the prediction of the skyline.

#### Feature-based

MOGA, Carle et al. (2010), is another solution to the long-range localization problem with an approach similar to that of VIPER, but using 3D maps instead of 2D images.

Carle et al. (2010) found that capturing 3D peak features has better alignment performance. The methodology was tested with real data, including 37 LiDAR scans of terrain from a Mars-Moon analog site on Devon Island, Nunavut. LiDARs though, are hardly used in a planetary rovers due to the heavy weight and high power consumption, as well as their big data range, that results to heavy calculations.

Featured-based approaches, in general, extract at first interest points from the global and local maps and then they match them in search of global-local feature correspondences.

Li, Di, and Howard (2007), in order to autonomously localize a Mars rover, propose an integrated Visual Odometry and Bundle Adjustment method to automatically select tie-points, such as rocks, and link the ground images acquired at multiple rover sites into an image network. Test results using MER (Mars Exploration Rovers) data showed that the proposed method achieved promising results for medium-range (up to 26 m) traverse segments. However, significant differences in resolution between orbital and ground imagery has proven this approach unable to be fully automated.

Hwangbo et al. (2009) continue the work of Li et al. (2007) using orbital and ground images. They generate high-resolution orthographic photos and DEMs as soon as the orbital stereo images are acquired. A ground image network is also constructed using inter-stereo matching. From both types of imagery, a few landmarks are identified to be used as ground control points for the integration of the orbital and ground image networks. Finally, distribution pattern matching is implemented for rocks detected from orbital and ground imagery. The rover position is adjusted based on a 2D affine transformation obtained from rock pattern matching. Experiments in MER data achieve a success rate of up to 50%, since there are terrain restrictions: desert areas with no rocks, or on the contrary very rough areas, in which rocks are hardly defined are difficult to handle.

## 1.4 Thesis Objectives and Outline

### 1.4.1 Research Objectives

The main objectives of this thesis in terms of research are:

- to utilize SLAM techniques for the localization and mapping of a planetary rover
- to develop a novel technique for minimizing the localization drift with global map matching
- to determine under which circumstances (i.e. resolutions of local and global maps) can the orbital imagery be useful for solving the SLAM problem

- to examine and quantify the gains (i.e. high resolution map for navigation purposes and absolute localization) and loses (i.e. processing overhead) of such approach.

#### 1.4.2 Outline

The remainder of this thesis is organized as follows:

- **Chapter 2** introduces a detailed approach for solving the SLAM problem in a planetary context using global map matching,
- **Chapter 3** deals with the implementation details and the tools used to develop a working system overall,
- **Chapter 4** presents the results of the experiments that were carried out in order to validate the approach and
- **Chapter 5** concludes with a summary, thoughts about future directions as well as possible applications.



## Chapter 2

# Globally Adaptive SLAM

### 2.1 System Overview

In order for planetary robots to navigate autonomously in extreme and uncertain conditions where no GPS information is available, they need to perceive their surroundings with high precision and maintain this precision over time. A way of compensating for the lack of real-time GPS data, is to use an *a priori* global map that contains 3D information about the environment the robot is trying to navigate into. This map can be reconstructed using imagery taken from an orbiter and has a notably lower resolution compared to the robot's sensory data. To take advantage of this existing information, we have developed a system called *Globally Adaptive SLAM* (*GA SLAM*), which is based around two layers of data processing.

In the first tier, the objective is to perceive the environment and capture its 3D structure so as to enable the robot to navigate in it. This is achieved by constructing a local map and estimating the robot's current pose (position and orientation) w.r.t. its initial pose, using sensory inputs and odometry data. This process is commonly known as *Simultaneous Localization And Mapping* (SLAM) and can be expressed as

$$p(x_{0:T}, m_{1:T} | z_{1:T}, u_{1:T}) \quad (2.1)$$

where  $x$  are the estimated poses of the robot,  $m$  is the constructed map of the environment,  $z$  are the sensory inputs and  $u$  are the robot's controls. To reduce the problem's dimensions, the following factorization is performed

$$p(x_{0:T}, m_{1:T} | z_{1:T}, u_{1:T}) = p(x_{0:T} | m_{1:T-1}, z_{1:T}, u_{1:T}) p(m_{1:T} | x_{0:T}, z_{1:T}) \quad (2.2)$$

where the former belief represents the pose estimation problem, also called relative localization, and the latter belief represents the mapping problem.

In the second tier, the objective is to achieve absolute localization by eliminating the accumulated drift which occurs in the previous stage. This drift is present in long-range scenarios and comes from the fact that all the input information is coming

directly from the robot itself. To eliminate it, a matching technique is utilized with the aim of finding the position of the robot's local map inside a global map.

The separation of the system to two layers comes from the need to process the data in separate moments, depending on the conditions of the environment as well as on the robot's processing capabilities. In Figure 2.1, the high-level design diagram of the system shows the 4 main modules that comprise the entire system. Their functionality is explained in the following sections of this chapter.

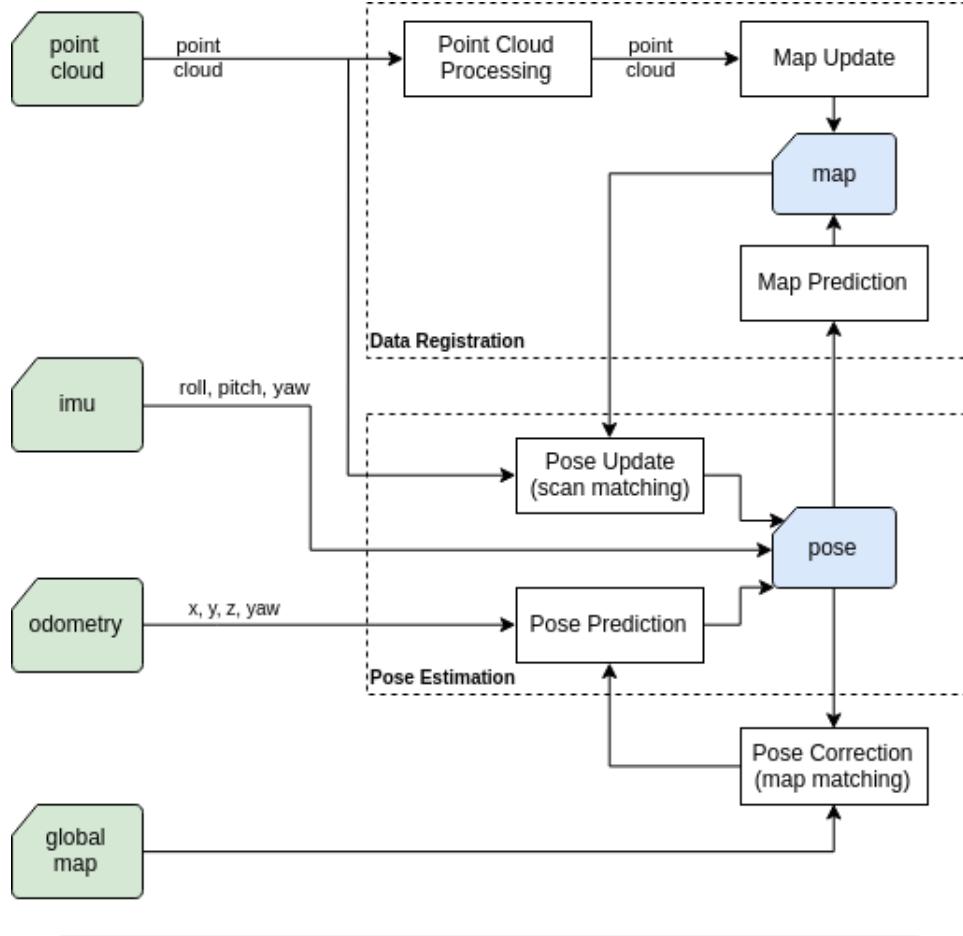


FIGURE 2.1: The high-level design diagram of the GA SLAM system.

## 2.2 Data Registration

An integral aspect of a robotic system, is the mapping process. In the previous chapter (Section 2.1) this process was expressed as

$$p(m_{1:T} | x_{0:T}, z_{1:T}) \quad (2.3)$$

where  $m$  is the posterior map of the environment,  $x$  are the estimated poses of the robot and  $z$  are the sensory inputs. The map is the model of the environment and can represent information about it in either two or three dimensions. In addition,

the information a map represents can be organized either in a grid-based manner (volumetric map) or in the form of landmarks (feature-based map) [Jefferies and Yeap, 2008]. Volumetric maps provide a more detailed and precise model of the environment compared to feature-based maps and are, therefore, the better choice for robots that aim to solve navigation tasks. Moreover, autonomous robots that need to navigate in non-flat surfaces, require a 3D model as a means to safely avoid the terrain's obstacles. Robotic applications with this requirement include outdoor, underwater, airborne or planetary missions.

As we have already mentioned, planetary rovers have an important constraint in on-board computational power. This limitation deems the use of pure 3D maps inappropriate. To overcome this problem, we can represent the environment with a 2.5D map, known as elevation map [Cremean and Murray, 2005]. An elevation map is essentially a 2D grid-map composed of cells that hold the value of the terrain's elevation. It is important to note that this mapping method does not model completely the 3D environment, but can provide a sufficient approximation even for the most demanding applications.

To make use of this mapping technique, a robot must be equipped with a sensor that is capable of providing measurements in the 3D space. Such measurements are usually stored in a structure called point cloud - a set of points  $P$  that contain information about their position in space ( $x, y, z$ ) [Rusu and Cousins, 2011]. A point cloud can be constructed from sensors that provide depth information, with the most popular being stereo cameras, time-of-flight (ToF) cameras and LiDARs.

In this section, the functionality of the Data Registration module will be explained. Its main purpose is to preprocess the input sensor data (point cloud) and register it to create an elevation map or update an existing one.

### 2.2.1 Point Cloud Preprocessing

The importance of preprocessing a point cloud before registering its data in a map, comes from the fact that point clouds contain large amount of points and hence process them is a time-consuming task. Additionally, the points represented in the cloud may be referenced in a different frame (usually the sensor's frame) than the frame of the map.

For this purpose, we implemented a point cloud preprocessing submodule, that consists of the following four steps:

1. **Voxelization:** In this step, we reduce the volume of the point cloud by down-sampling it using a discrete 3D grid. Each cell of the grid, called *voxel*, is mapped to  $\{0, 1\}$  and it therefore represents the existence or absence of an obstacle in that position. This method is called *voxelization* and results in a more sparse cloud of 3D points [Hinks, Carr, Truong-Hong, et al., 2013].

2. **Transformation:** A point  $P$  sampled from the sensor is in the sensor's reference frame  $S$  and need to be transformed in the map's frame  $M$  in order to be registered. This is achieved by

$$\mathbf{t}_{MP} = \mathbf{R}_{SM}\mathbf{t}_{SP} - \mathbf{t}_{SM} \quad (2.4)$$

where  $\mathbf{t}_{MP}$  is the position of  $P$  in the map frame,  $\mathbf{t}_{SP}$  is the position of  $P$  in the sensor frame,  $\mathbf{R}_{SM}$  is the rotation between the sensor frame and the map frame and  $\mathbf{t}_{SM}$  is the translation between them.

3. **Cropping:** Since a point cloud can span a wide area in space depending on the sensor's field of view and orientation, we can further reduce the volume of the cloud by discarding (cropping) points that fall out of the map. This is done by defining a box in space and removing from the point set all the points that are not inside it. In our case, we define the minimum and maximum cutoff points that represent the diagonal of the crop box as

$$\begin{aligned} min\_cutoff\_point &= \left( -\frac{l_x}{2} - t_x, -\frac{l_y}{2} - t_y, z_{min} \right) \\ max\_cutoff\_point &= \left( \frac{l_x}{2} + t_x, \frac{l_y}{2} + t_y, z_{max} \right) \end{aligned} \quad (2.5)$$

where  $l_x$  and  $l_y$  are the lengths of the map in meters in the  $x$  and  $y$  axis,  $t_x$  and  $t_y$  are the positions of the map in meters in the  $x$  and  $y$  axis and  $z_{min}$  and  $z_{max}$  are the minimum and maximum elevation of the map.

4. **Uncertainty calculation:** In this last step, we use the sensor's model to calculate the variance of each point in the point cloud,  $\sigma_z^2$ . As a result, each point will contain in addition to a  $x$  and  $y$  position, a one-dimensional Gaussian distribution  $\{z, \sigma_z^2\}$  for the height. The  $z$  value is calculated by applying the projection matrix  $\mathbf{P} = [0 \ 0 \ 1]$  to the transformed point using the Equation 2.4:

$$z = \mathbf{P}(\mathbf{R}_{SM}\mathbf{t}_{SP} - \mathbf{t}_{SM}) \quad (2.6)$$

This equation can also be written as

$$z = f(\mathbf{t}_{SP}) \quad (2.7)$$

where  $f : \mathbb{R}^3 \rightarrow \mathbb{R}$  is a non linear function.

To obtain the variance value of the transformed point, we propagate the uncertainty [Ochoa and Belongie, 2006] of the initial point as

$$\sigma_z^2 = \Sigma^f = \mathbf{J} \Sigma^{\mathbf{t}_{SP}} \mathbf{J}^\top \quad (2.8)$$

where  $\Sigma^f$  is the variance-covariance matrix of  $f$ ,  $\Sigma^{t_{SP}}$  is the covariance matrix of the initial point and  $\mathbf{J}$  is the Jacobian matrix of  $f$  defined as

$$\mathbf{J} = \frac{\partial f}{\partial \mathbf{t}_{SP}} = \mathbf{P} \mathbf{R}_{SM} \quad (2.9)$$

The covariance matrix for a point in the point cloud is defined as

$$\Sigma^{t_{SP}} = \begin{bmatrix} \sigma_{xx}^2 & \sigma_{xy}^2 & \sigma_{xz}^2 \\ \sigma_{yx}^2 & \sigma_{yy}^2 & \sigma_{yz}^2 \\ \sigma_{zx}^2 & \sigma_{zy}^2 & \sigma_{zz}^2 \end{bmatrix} \quad (2.10)$$

and is different for every sensor, since it depends on the sensor's model and characteristics. For example, for a stereo camera we approximate the variance of the point in three dimensions with a quadratic equation:

$$\sigma^2 = \frac{c \cdot \tan(0.5 \cdot f)}{0.5 \cdot b \cdot w} d^2 \quad (2.11)$$

where  $c$  is the search range or disparity precision,  $f$  is the field of view,  $b$  is the baseline,  $w$  is the horizontal resolution of the camera and  $d$  is the range of the point [Huesing et al., 2014].

## 2.2.2 Registration

### Structure of Elevation Map

The elevation map, also referred to as *local map* or simply *map*, is a robot-centric grid-based map. This means that it is centered in the robot's position in the global reference frame. When the robot moves, previous cells that now fall out of the map are reset and values that belong to the new explored area take their place.

The map models the elevation of the environment as well as the uncertainty of it. As a result, in every position of the 2D grid, the elevation is represented by a one-dimensional Gaussian distribution. This means that the map consists of two layers

- the layer of *mean* elevation ( $\mu_z$ ) and
- the layer of elevation *variance* ( $\sigma_z^2$ )

Apart from its structure, a map is also characterized by a few other properties. The most important are

- the *resolution*, which is the dimension of a cell in meters,
- the *length*, which is the length of the map in meters,
- the *minimum* and *maximum* elevation values that the map can hold,

- the 2D *position* of the map in the global reference frame and
- the *orientation* of the map in the global reference frame

Regarding the last property, it is worth mentioning that the orientation of the map is fixed w.r.t. to the global frame. This decision was made with a focus on reducing the complexity when performing a prediction of the map (see next section).

## Map Prediction and Update

Even though the map prediction and update steps are grouped in the same submodule, they implement two separate functionalities. The map prediction is triggered when a new odometry pose is received and processed by the Pose Estimation module (Section 2.3). The map update is triggered as soon as a new point cloud arrives from the sensor.

### • Prediction

This step is responsible for moving the map when the robot assumes a new position. This is essentially a simple 2D transformation of the map's position in the  $x$  and  $y$  axes. The robot's orientation is not taken into account, since as we said earlier (Section 2.2.2), the orientation of the map about the  $z$  axis is fixed. To perform the prediction, we simply

1. apply the delta translation  $(t_x, t_y)$  of the new estimated pose to map and
2. keep the height Gaussian  $\mathcal{N}(\mu_z, \sigma_z^2)$  of each cell unchanged

When a translation is applied to the map, cells that fall out of the map have their values reset and elevation values of the new explored area take their place instead. To avoid unnecessary data copying in memory, the data storage can be implemented as a two-dimensional circular buffer.

### • Update

This is the core step of the Data Registration module, since this step implements the registration of the input point cloud into the map. The algorithm of the point cloud projection is shown in Algorithm 1.

It uses a probabilistic approach to take advantage of the fact that both the point cloud (Section 2.2.1) and the map (Section 2.2.2) represent the elevation as a one-dimensional Gaussian distribution. The last operation of the algorithm (Operation 22) is the actual fusion, and can be implemented with different methods. A popular method is the one used in a Kalman filter [Cremean and Murray, 2005], which given two Gaussian distributions  $\mathcal{N}(\mu_1, \sigma_1^2)$  and  $\mathcal{N}(\mu_2, \sigma_2^2)$ , can be implemented as

$$\text{innovation} = \mu_2 - \mu_1 \quad (2.12)$$

$$gain = \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2} \quad (2.13)$$

$$\mu_1 = \mu_1 + gain \cdot innovation \quad (2.14)$$

$$\sigma_1^2 = (1 - gain) \cdot \sigma_1^2 \quad (2.15)$$

---

**Algorithm 1** Registration of point cloud to map
 

---

```

1:  $M_\mu$ : mean elevation layer of the map
2:  $M_{\sigma^2}$ : variance elevation layer of the map
3:  $P$ : point cloud - set of  $\{x, y, z\}$ 
4:  $V$ : point cloud variances - set of  $\{\sigma_z^2\}$ 
5:
6: for each  $p$  in the  $P$  do ▷ iterate through all points
7:   if position of  $p$  is outside of map then
8:     continue
9:
10:   $\mu_{point} \leftarrow p[z]$  ▷ projection of the point
11:   $\sigma_{point}^2 \leftarrow V[p]$ 
12:
13:   $index \leftarrow find\_map\_index(p[x], p[y])$  ▷ corresponding cell in grid
14:
15:   $\mu_{cell} \leftarrow M_\mu[index]$ 
16:   $\sigma_{cell}^2 \leftarrow M_{\sigma^2}[index]$ 
17:
18:  if  $\mu_{cell}$  or  $\sigma_{cell}^2$  is not initialized then
19:     $\mu_{cell} \leftarrow \mu_{point}$ 
20:     $\sigma_{cell}^2 \leftarrow \sigma_{point}^2$ 
21:  else
22:     $\mathcal{N}(\mu_{cell}, \sigma_{cell}^2) \leftarrow \mathcal{N}(\mu_{cell}, \sigma_{cell}^2) \cdot \mathcal{N}(\mu_{point}, \sigma_{point}^2)$ 
23:
24:   $M_\mu[index] \leftarrow \mu_{cell}$ 
25:   $M_{\sigma^2}[index] \leftarrow \sigma_{cell}^2$ 
  
```

---

### 2.2.3 Sensor Fusion

The goal of this submodule is to fuse data from different sources and increase the overall quality of the map. This is achieved by taking advantage the already computed uncertainty estimates in the map and applying fusion techniques to improve the existing data.

Using sensor fusion we can exploit the multiple sensors equipped in a robot to increase the covered area in the map. In addition to increased coverage, fusing sensor data that are overlapping in the map can increase its quality.

The way a point cloud is registered (fused) in the map, was already mentioned in a previous section (Section 2.2.2). Using this method we can combine information

from different sensors, as long as the input data is in the form of a point cloud, by simply updating the map and taking advantage of the uncertainty estimates that are calculated in the preprocessing of the point cloud.

A drawback of this technique is that large calibration errors and inaccuracies in the transformations ( $\mathbf{t}_{SM}$  and  $\mathbf{R}_{SM}$ ) between a sensor and the map can have the opposite effect i.e. deteriorate the map quality. This is due to the fact that fusion can over-smooth obstacles since their correspondence in the map will be slightly different for each sensor. Errors like these can easily produce false negative obstacles and have a detrimental effect in the navigation of the robot.

## 2.3 Pose Estimation

The pose, or state, of a robot navigating in a 3D environment, consists of its position ( $x, y, z$ ) and its orientation ( $roll, pitch, yaw$ ) w.r.t. a global reference frame. Hence, the goal of pose estimation, also referred to as relative localization, is to accurately track the robot's pose in the environment. This is usually accomplished by either employing a probabilistic filter, e.g. a Kalman filter or a particle filter, or by utilizing methods such as scan matching [Fox et al., 2001]. A filter's inputs are usually expressed in the form of position or velocity and can be extracted using odometry techniques. In comparison, the input of a scan matching method is by definition a scan, i.e. measurement, received by a sensor. For our case, we approached the localization problem by making use of a combination of scan matching and particle filtering techniques.

First, we assume the state of the problem to be

$$\mathbf{x} = [x \ y \ z \ roll \ pitch \ yaw]^\top \quad (2.16)$$

where *roll*, *pitch* and *yaw* are the rotations about the *x*, *y* and *z* axes respectively. However, we cannot obtain accurate estimates for the *roll* and *pitch* states from odometry. To overcome this limitation, we use an *inertial measurement unit* (IMU) sensor and receive these states directly from it, so the problem state becomes

$$\mathbf{x} = [x \ y \ z \ yaw]^\top \quad (2.17)$$

By making the assumption that the *z* estimation from odometry is accurate in an area near the robot, the state can be further simplified as

$$\mathbf{x} = [x \ y \ yaw]^\top \quad (2.18)$$

which is also referred to as

$$\mathbf{x} = [x \ y \ \theta]^\top \quad (2.19)$$

In the case where this assumption is false, the result will be a deformed map, since the elevation values of the map will be directly affected by the wrong  $z$  estimate of the robot.

We employ a particle filter that samples particles in a continuous state space to estimate the posterior of the robot's state

$$p(x_{0:T} \mid m_{1:T-1}, z_{1:T}, u_{1:T}) \quad (2.20)$$

where  $m$  is the constructed map of the environment,  $z$  are the sensory inputs and  $u$  are the odometry inputs. This is done by using the odometry inputs to predict the state of each particle and the sensory inputs, i.e. point clouds, for the weight update. Additionally, the uncertainty of each particle is modeled using a Gaussian distribution. Finally,  $yaw$  measurements from the robot's IMU sensor are fused with the final estimate to provide more accurate measurements.

An activity diagram illustrating the algorithm's steps is shown in Figure 2.2. These steps are explained in detail in the following sections.

### 2.3.1 Initialization

This step is executed only during the initialization of the algorithm, or when the particle filter is reset and the objective is to sample each particle using a Gaussian distribution. The probability density function of the distribution  $\mathcal{N}(\mu, \sigma^2)$  is

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (2.21)$$

where  $\mu$  is the mean of the distribution,  $\sigma$  is the standard deviation and  $\sigma^2$  is the variance. Using this function, the state of each particle is sampled, as shown in Algorithm 2.

---

**Algorithm 2** Initialization of particle filter

---

```

1:  $P$ : set of particles with  $\{\{x, y, \theta\}, weight\}$ 
2:
3: for each  $p$  in the  $P$  do
4:    $x \sim \mathcal{N}(\mu_x, \sigma_x^2)$             $\triangleright$  sample the states from their distributions
5:    $y \sim \mathcal{N}(\mu_y, \sigma_y^2)$ 
6:    $\theta \sim \mathcal{N}(\mu_\theta, \sigma_\theta^2)$ 
```

---

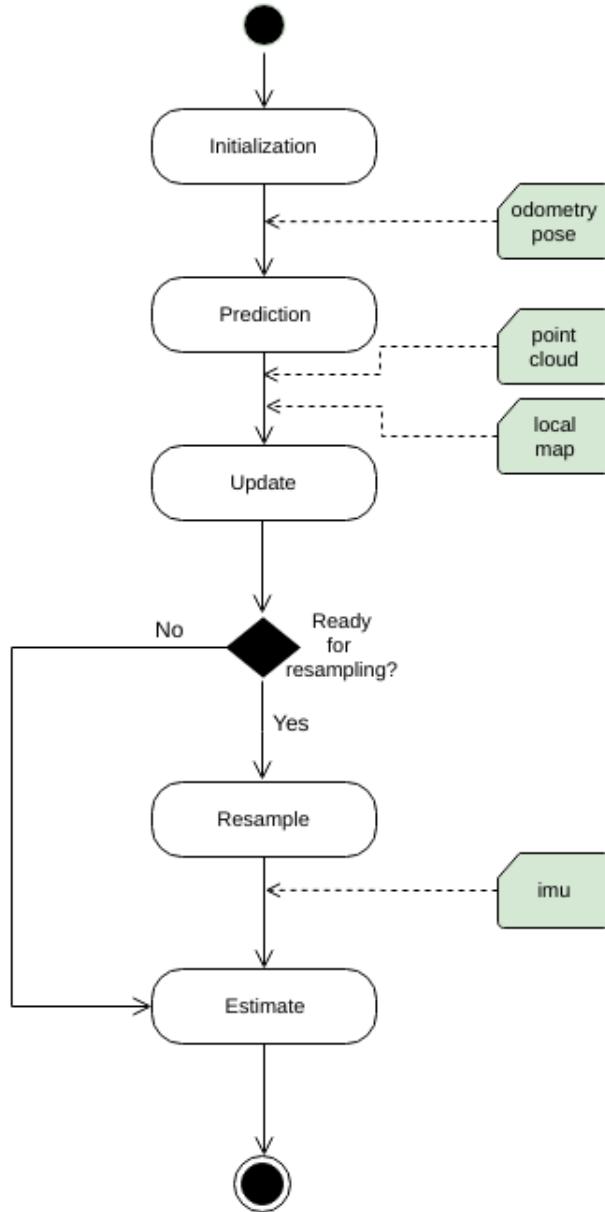


FIGURE 2.2: Activity diagram of the pose estimation module.

### 2.3.2 Prediction

When a new odometry input  $u_t$  is received, the state of each particle  $p_i$  is sampled using the prediction as the mean value of a Gaussian distribution as such

$$x_t^i \sim \mathcal{N}(x_{t-1}^i + \Delta x_t, \sigma_x^2) \quad (2.22)$$

$$y_t^i \sim \mathcal{N}(y_{t-1}^i + \Delta y_t, \sigma_y^2) \quad (2.23)$$

$$\theta_t^i \sim \mathcal{N}(\theta_{t-1}^i + \Delta \theta_t, \sigma_\theta^2) \quad (2.24)$$

where  $x_i$ ,  $y_i$  and  $\theta_i$  represent the state of the particle,  $\Delta x$ ,  $\Delta y$  and  $\Delta\theta$  are the delta inputs from odometry and  $\sigma_x^2$ ,  $\sigma_y^2$  and  $\sigma_\theta^2$  are the variances of the states that define the Gaussian noise that will be added to the particle.

### 2.3.3 Update

When a new point cloud measurement  $z_t$  is received, the weight of each particle  $p_i$  is updated using scan matching. In particular, for every particle, the input point cloud is matched with a point cloud extracted from the robot's local map using an Iterative Closest Point (ICP) variant [Zhang, 1994].

For the purpose of achieving a fast convergence of the ICP, we first preprocess the inputs as such: (i) crop the raw point cloud (sensor scan) to the local map's size (ii) convert local map to a point cloud using the elevation values of the grid map (iii) downsample both point clouds using a voxel grid (in a similar way as in Section 2.2.1).

In addition, the point cloud extracted from the local map is transformed to each particle's state (pose) and as a result each particle holds a representation of the environment according to its hypothesis. Since the weight of each particle is proportional to the uncertainty of its hypothesis, it can be updated by measuring the degree of alignment between the raw point cloud and the particle's point cloud.

One way to measure the alignment between a source point cloud  $S = \{s_1, s_2, \dots, s_{N_s}\}$  and a reference point cloud  $R = \{r_1, r_2, \dots, r_{N_r}\}$  is by computing the sum of the squared error

$$e = \frac{1}{N_r} \sum_{i=1}^{N_r} \|s_i - r_i\|^2 \quad (2.25)$$

where  $s_i$  and  $r_i$  are the points which correspond to each other.

An easy method of performing this computation is to run one iteration of the ICP algorithm and using the output fitness score as the point cloud alignment error. Therefore, the weight  $w_i$  of a particle  $p_i$  is calculated as

$$w_i = \frac{1}{icp\_fitness\_score} \quad (2.26)$$

### 2.3.4 Resampling

The act of resampling the set of particles of the filter is done to ensure that the particle population models the uncertainty of the problem as close as possible. Although there are several variants [Douc and Cappé, 2005] that can be exploited to solve the resampling problem depending on the situation, we make use of a fairly common technique called multinomial resampling.

Given a particle set of  $N$  particles, a cumulative distribution is initially formed by normalizing the weight of each particle as

$$w_i = \frac{w_i}{\sum_{k=0}^N w_k} \quad (2.27)$$

where  $w_i$  is the weight of a particle  $p_i$  from the particle set. From the cumulative distribution,  $N$  new particles are sampled uniformly while maintaining the state of each particle.

It is obvious that particles which were associated with a high weight in the previous population have a higher chance of being selected. Thus, a new population is created that possibly contains duplicate particles. These particles will be scattered again in the next iteration by adding Gaussian noise in the prediction step (Section 2.3.2).

An important aspect of resampling is the resampling strategy used, i.e. the moment that resampling is performed in the population. Depending on the parameters of the particle filter as well as on the states of the problem, different strategies yield different results. In our approach, we adopted a simple strategy that resamples every  $K$  iterations of the algorithm. This parameter depends on the robot's controls as well as on the magnitude of the Gaussian noise added to each particle and can be configured accordingly for each application.

### 2.3.5 Estimation

The last step of the algorithm is to extract the final pose estimate from the particle filter. This is accomplished in two steps: (i) estimate pose from the particle cloud (ii) fuse pose with measurements from an IMU sensor.

In many applications of the particle filter, the first step is achieved by simply picking the best particle, i.e. the particle with the highest weight in the particle set. However, using this method, when the best particle is removed from the population during resampling, choosing the next best particle will cause the robot to "teleport" in the continuous space, hence deforming the constructed map. To overcome this issue, we have implemented a technique (Algorithm 3) that extracts the pose by thresholding particles with low weight and averaging the remaining.

Finally, to further improve the estimation, we fuse the  $yaw$  measurement from the IMU sensor with the  $yaw$  estimate from the particle filter. The fusion is done by means of one-dimensional Gaussian distributions, hence we need to associate a variance to each value.

The IMU Gaussian is represented as  $\mathcal{N}(\mu_{imu}, \sigma_{imu}^2)$  where  $\mu_{imu}$  is the measurement value from the sensor and  $\sigma_{imu}^2$  is the variance which is a parameter that represents how reliable the sensor is. The second Gaussian is represented as  $\mathcal{N}(\mu_\theta, \sigma_\theta^2)$ , where

---

**Algorithm 3** Estimation of final pose from particle population

---

```

1:  $P$ : set of particles
2:  $p$ : particle - set of  $\{states, weight\}$ 
3:
4:  $S \leftarrow \{\}$  ▷ candidate particles
5: for each  $p$  in  $P$  do
6:   if  $p[w] > w_{threshold}$  then ▷ remove uncertain particles
7:      $S \leftarrow S + p$ 
8:
9: Sort  $S$  by weight
10:  $M \leftarrow S[1 : k]$  ▷ pick top k particles
11:
12:  $w_{sum} \leftarrow \sum_{i=1}^k w_i$ 
13:  $x \leftarrow \sum_{i=1}^k w_i x_i / w_{sum}$  ▷ weighted average of each state
14:  $y \leftarrow \sum_{i=1}^k w_i y_i / w_{sum}$ 
15:  $\theta \leftarrow \sum_{i=1}^k w_i \theta_i / w_{sum}$ 

```

---

$\mu_\theta$  is the *yaw* estimate that was extracted in the previous step and  $\sigma_\theta^2$  is the variance of the distribution. It can be calculated by applying the formula for a weighted standard deviation as

$$\sigma_\theta = \sqrt{\frac{\sum_{i=1}^N w_i (\theta_i - \mu_\theta)^2}{\sum_{i=1}^N w_i}} \quad (2.28)$$

where  $N$  is the number of particles,  $w_i$  is the weight of each particle and  $\theta_i$  is the *yaw* hypothesis of each particle. The fusion of the two distributions is performed in a similar fashion as in the Data Registration module (Section 2.2.2).

## 2.4 Pose Correction

Although the robot's pose is constantly tracked using a particle filter in the Pose Estimation module (Section 2.3), the approach is still based on dead-reckoning due to the fact that all the inputs of the process arrive from internal sensors. As a result, a localization error is accumulated in long-range traverses and the robot's pose in the global reference frame is unknown. With the aim of bounding this error and achieve absolute (global) localization, we have developed an extra component in the system that eliminates the drift by matching the local map to an *a priori* global one.

The global map can be reconstructed using imagery taken from an orbiter (satellite), although its resolution cannot reach the level of the local one. Furthermore, in contrast to the local map which is always centered around the robot, the global map is referenced in the global frame, i.e. the same coordinate system as the robot's pose.

This enables us to match the two maps and apply the delta transformation to correct the drifted pose.

### 2.4.1 Global to Local Map Matching

The matching of 2D images (frames) is a topic that has been extensively researched in the field of computer vision and several techniques have been developed for that purpose. Since grid-based 2D maps are essentially images, it is possible to apply these techniques to maps as well. The act of matching two images consists of either finding the absolute position of the first image inside the second one or finding a certain correspondence between the two.

Popular approaches vary from simple and statistical-based ones such as comparing the histogram of the images to find similarities, to more complex and efficient ones such as template matching and feature matching [Jayanthi and Indu, 2016]. The former, template matching, is a method that convolves a search image (template) with the one being searched into (source) and is usually used to find a smaller image in a bigger one. The latter, feature matching, is based on the idea of extracting a feature set from each image and comparing the two sets in order to find correspondences among them.

Although feature matching is usually the most efficient method, it can yield sub-optimal results in scenarios where features are almost indistinguishable and their extraction contains uncertainties. For this reason, we chose to base our matching approach on a template matching method and exploit the feature-sparse environment of our planetary application.

The proposed algorithm consists of the following three steps:

- **Preprocessing**

Template matching requires the two images to be of the same resolution. Since, as we mentioned earlier, the global image (source image) has a lower resolution than the local image (template image), it is necessary to downsample the local one to match the resolution of the global. To achieve this, we scale the template image with the local to global resolution ratio using a nearest-neighbor interpolation method.

As we discussed in the implementation of the Pose Estimation component (Section 2.3), the long-term drift of the z estimation which is received directly from odometry, causes an elevation offset in the local map. To overcome this offset, it is possible to match the gradients (first derivatives) of the two images instead of directly matching the normal ones, by applying the Sobel operator in the two images.

The computation of the gradient of an image  $I$ , requires first the computation of the gradients in the x and y axes by convolving  $I$  with a Sobel odd-sized kernel. For a kernel size of three, the kernel in the horizontal direction is

$$K_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad (2.29)$$

and in the vertical direction is

$$K_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} \quad (2.30)$$

Thus, the two gradients are computed as

$$G_x = K_x * I \quad (2.31)$$

$$G_y = K_y * I \quad (2.32)$$

and the final magnitude of the gradient is calculated by combining both gradients:

$$G = \sqrt{G_x^2 + G_y^2} \quad (2.33)$$

Finally, we replace unknown cell values, i.e. cells that do not contain an elevation value, with zero since the matching method we use cannot handle unknown values. It is also possible to mask these values during matching. In any case, zero values do not contribute to the outcome of the match as it is explained in the next step.

- **Matching**

A drawback of template matching compared to feature matching is that it cannot match rotated images, and therefore it cannot find a *yaw* correction in the robot's global pose. To overcome this limitation, we warp (rotate) the template image in a discrete angle space, e.g. from  $-10^\circ$  to  $10^\circ$  with a step of  $1^\circ$ . As a result, we run template matching for every individual angle (e.g. twenty times for the previous example) and choose the *yaw* correction angle that provides the best result.

Template matching works by sliding the template image  $T$  pixel-by-pixel in the source image  $I$  and comparing at each location the two images using a metric (score). Then for each location of  $T$  over  $I$ , the score is stored in a result matrix  $R$ . Consequently, the location in  $R$  where the score is highest, is the location of the best match [Hashemi et al., 2016].

The metric used to compare the images, greatly influences the final result. The most popular metrics used are:

1. the Sum of Square Difference (SDD), which is expressed as

$$R(x, y) = \sum_{x', y'} (T(x', y') - I(x + x', y + y'))^2 \quad (2.34)$$

where  $x'$  and  $y'$  correspond to the pixel (location) of a specific pass.

2. the Cross Correlation, which is expressed as

$$R(x, y) = \sum_{x', y'} (T(x', y') \cdot I(x + x', y + y')) \quad (2.35)$$

3. the Correlation Coefficient, which is expressed as

$$R(x, y) = \sum_{x', y'} (T'(x', y') \cdot I'(x + x', y + y')) \quad (2.36)$$

where

$$T'(x', y') = T(x', y') - \frac{\sum_{x'', y''} T(x'', y'')}{width \cdot height} \quad (2.37)$$

$$I'(x + x', y + y') = I(x + x', y + y') - \frac{\sum_{x'', y''} I(x + x'', y + y'')}{width \cdot height} \quad (2.38)$$

Since, the invalid cells of the template image have been converted to zero, the last method cannot be used because these cells would affect the result. After experimental evaluation of the first and the second metrics, we came to the conclusion that the second method (Equation 2.35) yields more optimal results for our application.

In addition, it is practical to normalize the metric to produce a metric that scales with the size and intensity of the template and source images. The new metric becomes:

$$R(x, y) = \frac{\sum_{x', y'} (T(x', y') \cdot I(x + x', y + y'))}{\sqrt{\sum_{x', y'} T(x', y')^2 \cdot \sum_{x', y'} I(x + x', y + y')^2}} \quad (2.39)$$

Another advantage of the normalization of the metric, is that it provides an output score in the fixed range  $[0 \dots 1]$ , with 1 being a perfect match. Using that, we can provide certain guarantees to the matching by selecting the best match only when the score is above a specific threshold (e.g. 95%).

#### • Correction

The last step consists of converting the position  $(x, y)$  of the match from image

coordinates to map coordinates:

$$x_{map} = \left\lfloor \frac{h}{2} \right\rfloor - x_{image} \quad (2.40)$$

$$y_{map} = \left\lfloor \frac{w}{2} \right\rfloor - y_{image} \quad (2.41)$$

where  $h$  and  $w$  are the height and width of the template image respectively.

Finally, we apply the correction  $(x, y, \theta)$  to the particle filter as a control input, i.e. all the particles are updated using this delta transformation. Doing that, we ensure that Gaussian noise will be added to each particle, as specified in Section 2.3.2.

#### 2.4.2 Criteria Checking

Absolute map-based localization techniques are known to be time consuming, since they require the processing of a large amount of data. In our case, template matching works by comparing the local map to a patch of the global map for every pixel of the latter. Moreover, this process is executed multiple times for a discrete space of angles values with the aim to correct the drift in  $yaw$  in addition to the 2D position correction. It becomes obvious that a proper execution strategy has to be adopted so as to avoid the use of unnecessary processing power.

For planetary applications, since energy efficiency is of utmost importance, it is common for rovers to focus only on one task at each moment. Thus, an absolute localization task cannot be run in the middle of a traverse, where the robot's motors are at full power. An approach that takes this limitation into consideration, is to execute the task in the end of the day, when the rover recharges and only secondary tasks are scheduled.

Nevertheless, as energy-efficient task scheduling is out of the scope of this work, we have proposed a practical solution that can be utilized alongside the rest of the system. For this purpose, two criteria are defined that have to be checked before a global to local map matching is initiated. The first criterion defines the minimum distance the robot must cover and the second criterion defines the structure of the environment that the pose correction module works in.

##### Traversed Distance Criterion

To minimize the execution overhead of the map matching process, it would be sensible to correct the localization's accumulated drift only when it surpasses a certain level. For this purpose, we defined a threshold distance value that the robot must

traverse before a new correction can take place:

$$\sqrt{(x - x_0)^2 + (y - y_0)^2} > d_{threshold} \quad (2.42)$$

where  $\{x, y\}$  is the current position of the robot,  $\{x_0, y_0\}$  is the position of the last pose correction and  $d_{threshold}$  is the traversed distance threshold value. It is important to notice that the robot's traversed distance is calculated in two dimensions, since the correction is also expressed in terms of  $\{x, y\}$ .

By taking into account the fact that the delta pose that results from the map matching process is in the same order of magnitude as the global map's scale, the threshold value should be proportional to it as well. In addition, the threshold depends on the accuracy of the pose estimation as well as on the speed of the robot, and should be chosen accordingly for each application.

### Environment Structure Criterion

With the exception of localization techniques that rely on global navigation satellite systems (e.g. GPS), the existence of some form of surrounding features is necessary for a robot to localize itself. As a result, we defined a second criterion that takes into account the structure of the environment by checking for elevation features.

This is achieved in four steps: (i) create a gradient image from the local elevation map by calculating the gradient in elevation (similarly to the map matching preprocessing step of Section 2.4.1) (ii) threshold the gradient image to remove low gradient values (iii) add remaining values to get total gradient value (iv) compare total value to threshold.

The threshold compared to the total gradient value is a parameter that mainly depends on the morphological characteristics of the terrain. Furthermore, it should be tuned according to the resolution of the local map since the sum of the map's gradients is proportional to the map's scale.

## Chapter 3

# Implementation and Tools

### 3.1 Planetary Rover Testbed

In order to benchmark our system in real scenarios, we implemented it on a testbed rover and integrated it with the rest of its components. The rover we used is named Heavy Duty Planetary Rover (HDPR) and is capable of traversing different types of terrain with a speed of up to 1 m/sec. It was developed by the Automation and Robotics Lab (ARL/TEC-MMA) of the European Space Agency (ESA) and is currently used for validating software and hardware components in long range scenarios.

The HDPR rover [Boukas et al., 2016], depicted in Figure 3.1, bears close resemblance to the camera setup of the ExoMars mission rover [Vago et al., 2017] and allows the evaluation of algorithms that have a focus on autonomous navigation. Specifically, the sensor suite of the rover comprises of the following components:

- Exteroceptive:
  - LocCam stereo camera (PointGrey Bumblebee 2)
  - PanCam stereo camera (PointGrey Grasshopper 2)
  - Kinect time of flight camera
  - MESA SR4500 time of flight camera
  - Velodyne LiDAR (VLP-16)
- Proprioceptive:
  - Trimble GNSS receiver (BD970)
  - Inertial Measurement Unit (Sensonor STIM300)
  - Shaft Encoders and Potentiometers

From these sensors, we exploit (i) the LocCam for visual odometry inputs and (ii) the PanCam for SLAM inputs. The LocCam has a baseline of 120 mm, is positioned about 1.1 m above the ground and is tilted at an angle of 18°. The PanCam has a



FIGURE 3.1: The Heavy duty planetary rover (HDPR) [Boukas et al., 2016].

baseline of 500 mm, is positioned about 1.9 m above the ground, is tilted at an angle of 19° and has a field of view (FoV) of 40°.

In addition, the Inertial Measurement Unit (IMU) provides measurements about the orientation and acceleration of the robot. Although the rest of the sensors are not actively used by our system, they can be exploited for validation purposes.

## 3.2 Overall System Architecture

As we mentioned previously, the GA SLAM system was designed to function with point cloud based inputs in addition to odometry inputs.

To utilize the sensor suite of the HDPR rover, we can extract point clouds using stereo processing techniques and odometry estimation using visual odometry methods. For the former we employ the PanCam stereo camera and for the latter we make use of the LocCam stereo camera. Furthermore, we exploit the measurements received from the IMU to augment the estimation of the robot’s pose. The utilization of each sensor of the HDPR rover w.r.t. to our system, is shown in the system component diagram of Figure 3.2.

As depicted, the local elevation map of the GA SLAM subsystem can be used in order to extract a map that contains traversability features about the surrounding environment of the rover, i.e. features that determine if the rover is able to traverse a specific area. At a next layer, the traversability map and the estimated pose

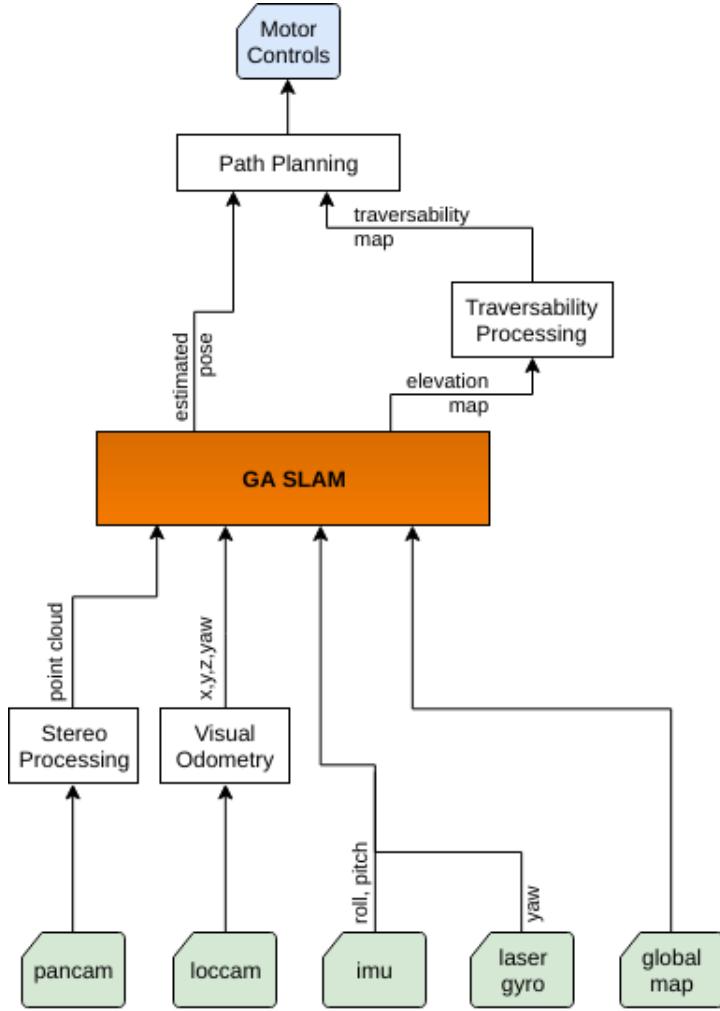


FIGURE 3.2: The components of the system that enable the HDPR to perform autonomous navigation tasks.

from SLAM can be utilized by a path planning module, with the aim to provide autonomously control the rover in an unknown environment. Finally, It should be noted that the choice of visual odometry instead of plain wheel odometry, comes from the fact that a robot is prone to slippage in outdoor environments, and more specifically planetary terrains.

### 3.3 Robotic Software Framework

The GA SLAM system<sup>1</sup> was implemented as a framework-independent library for C++ and depends only on libraries for linear algebra, point cloud processing and image processing. However, with the aim to provide integration with the rest of

<sup>1</sup>GA SLAM: [https://github.com/geromidg/GA\\_SLAM](https://github.com/geromidg/GA_SLAM)

HDPR's software modules and tools, we developed a component to bridge the library using the Robot Construction Toolkit (ROCK)<sup>2</sup>.

ROCK is a software framework for the development of robotic systems, similar to Robot Operating System (ROS). It differs from ROS in the sense that the development of applications is achieved using a model driven approach. It is implemented on top of the Orococos Real-Time Toolkit (Orococos RTT) component layer and can provide real-time capabilities to a system. In addition, it contains a rich collection of reliable and extensible modules that are ready to use as well as tools for visualization and task monitoring.

In ROCK [Joyeux and Albiez, 2011], each distinct functionality of the system is encapsulated in an RTT component (Figure 3.3) using a tool called *oroGen*. This tool, auto-generates the C++ code of the component, given the specification of the component's interface. The communication with the rest of the ROCK universe, is implemented through a network of connected components. This is achieved using the framework's Ruby modules which support the explicit interconnection of a component's input port(s) with the output port(s) of another component.

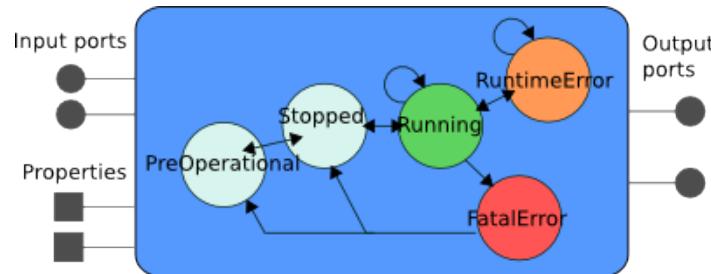


FIGURE 3.3: The interface and the state machine of a component in the Orococos Real-Time Toolkit. The output port(s) of a component can be explicitly connected to the input port(s) of another component. The properties are used to configure the component internally.

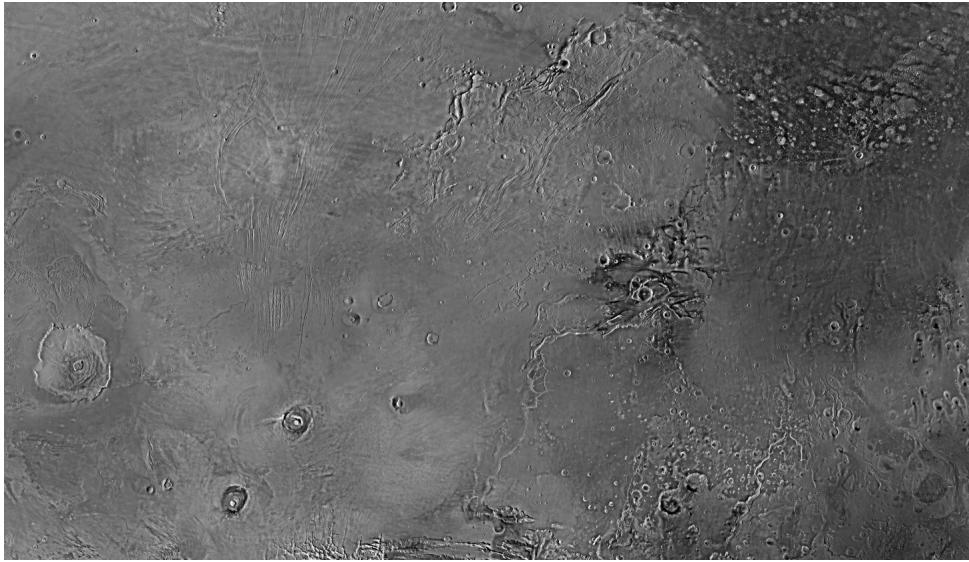
### 3.4 Orbiter Data Preprocessing

As a means to validate the map matching capabilities of the system, an orbiter (global) map is necessary. A real example of such map has been reconstructed using orbital imagery, depicted in Figure 3.4, that was captured using the High Resolution Imaging Science Experiment (HiRISE) camera mounted on NASA's Mars Reconnaissance Orbiter (MRO) [Gallagher et al., 2005].

It is possible to emulate an orbiter map using drone imagery. The aerial images can then be used to reconstruct and provide a 3D representation of the captured scene in

<sup>2</sup>ROCK: <https://www.rock-robotics.org>

the form of a densified point cloud. At a later stage, the constructed point cloud can be projected in a 2D grid and create an elevation map.



---

FIGURE 3.4: Close view of Mars that was captured using the HiRISE camera mounted on the Mars Reconnaissance Orbiter.

Prior to the projection, it is necessary to preprocess the global point cloud in order to simplify it and transform it to a known reference system. This is achieved in 4 steps: (i) voxelize point cloud to the desired resolution (ii) crop the voxel grid in the order of magnitude as the local map's size instead of using the whole information (e.g. for a local map of  $10\text{ m} \times 10\text{ m}$  crop the grid to  $100\text{ m} \times 100\text{ m}$ ) (iii) smooth the voxel grid using a Statistical Outlier Removal (SOR) filter which rejects points that are far from their neighbors (iv) transform the voxel grid to the robot's initial absolute pose.



## Chapter 4

# Experimental Validation

### 4.1 Scope of Experiments

The purpose of the experiments is to validate the proposed system as well as its robustness in various terrains and configurations. Since the autonomous navigation of the robot is out of the scope of this thesis, the mapping procedure is only indirectly validated as part of the SLAM system. Thus, the main focus of the experiments is the relative and absolute localization techniques that were developed.

#### 4.1.1 Data Collection

The performed experiments were executed by means of *Software In the Loop* (SIL) tests using precollected data. The data were collected by the Planetary Robotics Lab team of the European Space Agency as part of a two week test campaign that took place in the Minas de San Jose desert in Tenerife, Spain with the aim to validate the integration of the HDPR rover (Section 3.1) and provide data for future research projects. The aerial view of the local is presented in Figure 4.1.

For that reason, the collected dataset contains samples from: (i) three stereo camera sensors (ii) IMU sensor (iii) point clouds reconstructed from drone imagery (iv) GPS sensor. These samples are used both as input to our system and as ground truth information.

The environment in which the test data were collected, was chosen with the aim to bear close resemblance to rough planetary terrains, specifically to the one of Mars. In particular, various geological properties such as the distribution and the shape of the rocks is similar to the Martian terrain.

Finally, the speed of the robot in these datasets was chosen to be close to the 10 cm/sec value, considering the long-range traverses it was required to cover. As such, for the first two experiments, the size of the local map is fixed at 20 m × 20 m and its resolution at 10 cm/cell, while for the global map we presumed the resolution to be fixed



FIGURE 4.1: Aerial view of test field that was reconstructed using drone imagery.

at 50 cm/cell. In the third experiment, the performance of our approach is validated against these parameters, hence they are variable.

#### 4.1.2 Metrics

To quantify the experimental results and evaluate the outputs of the system, it is necessary to introduce certain metrics:

- **Mean square error (MSE) of pose graph:** This is a straightforward metric for benchmarking SLAM algorithms [Kümmerle et al., 2009] and can be defined as

$$\varepsilon_{MSE}(\hat{x}_{1:T}) = \frac{1}{T} \sum_{t=1}^T (\hat{x}_t \ominus x_t)^2 \quad (4.1)$$

where  $\hat{x}_{1:T}$  is the estimated pose graph,  $x_{1:T}$  is the ground truth pose graph,  $T$  is the number of samples and  $x_i \ominus x_j$  is the distance between two poses.

- **Root mean square deviation (RMSD) of pose graph:** Similarly to the MSE metric, this metric expresses the sample standard deviation of the differences between the estimated and the ground truth poses and can be defined as

$$\varepsilon_{RMSD}(\hat{x}_{1:T}) = \sqrt{\frac{1}{T} \sum_{t=1}^T (y_t - \varepsilon_{MSE})^2} \quad (4.2)$$

where  $y_t$  is defined as  $\hat{x}_t \ominus x_t$ .

## 4.2 Experiments on Pose Estimation

### 4.2.1 Relative Localization Results

The purpose of this experiment is to test the accuracy of the Pose Estimation module (see Section 2.3) and evaluate its capabilities in a relative localization (short-range) scenario. To accomplish that, we will carry out each experiment with different parameter sets with the aim of examining the system’s sensitivity as well as finding an optimal parameter configuration.

The main parameters that we will tune during the experiments are all related to the particle filter and include (i) the number of particles used (ii) the resampling frequency (iii) the standard deviation of the Gaussian noise added to each particle. Finally, the expected error of the estimated pose should ideally be in the same order of magnitude as the size of one cell of the local map, so as to ensure that the robot can distinguish the environment’s obstacles with a high-level of certainty.

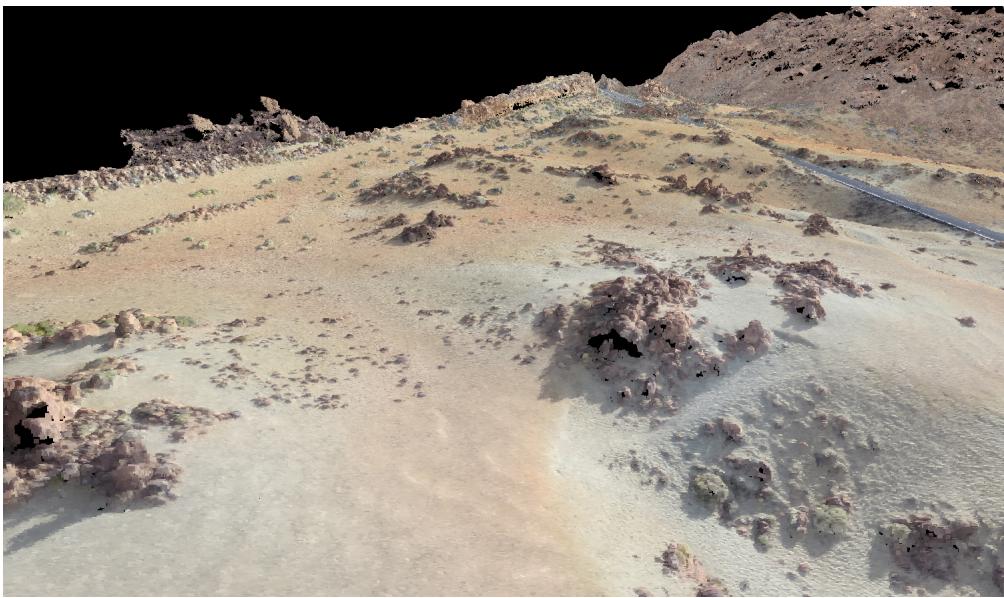


FIGURE 4.2: 3D view of the terrain evaluated in the first experiment.

Figure 4.2 presents a view of the evaluated terrain of the first scenario. The terrain is depicted in the form of one dense point cloud which was reconstructed from drone imagery. The bird’s eye view in Figures 4.3b and 4.3c shows the traverse of the robot in this scenario and compares the ground truth path, the one from raw visual odometry and the one from the pose estimation. In contrast to the visual odometry estimate (blue path), the pose estimate of our system (red path) follows closely the ground truth (black path). The richness of the terrain in features, which is visible in Figure 4.3a, is an advantage for relative localization since the weights of the particles in the particle filter are updated using scan matching.

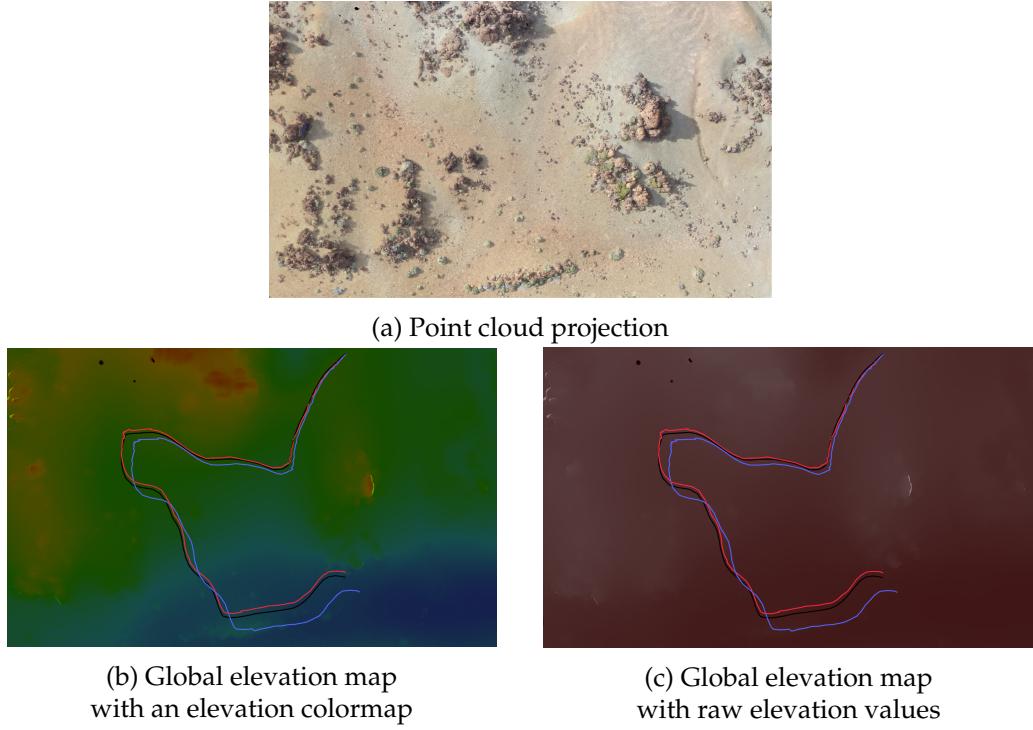


FIGURE 4.3: Orthographic views of the environment and the robot’s traverses in it. The black, red and blue colors correspond to the ground truth, SLAM and visual odometry paths accordingly.

In particular, the influence the number of particles used has on the outcome is shown in the comparison plot in Figure 4.4a. We observe that the pose error decreases as we increase the number of particles used in the particle filter, since the continuous space of the problem is modelled more effectively. However, increasing this parameter, brings a proportional increase to the computation needs for every iteration of the filter. As it is visible in the plot, a justified choice would be to use 100 particles for a balance of error and processing power. Furthermore, it is important to notice that the drift in pose (20 cm in the case of 100 particles), is proportional to the robot’s speed (10 cm/sec in this experiment).

Regarding the second parameter, the plot in Figure 4.4b represents the same error as the previous plots, with the difference that the number of particles is fixed to the value of 100 and the resampling frequency is the variable parameter. It is visible that resampling too often or too rare has an impact on the pose error. This is due to the fact that by resampling, the particle population is gathered near the best belief and as a result other possible solutions are eliminated.

For the purpose of further quantifying the results of this scenario altogether, we calculate the *MSE* and *RMSD* metrics (defined in Section 4.1.2) for the parameter sets that were used to generate the above plots. The values of these metrics are presented in Table 4.1. The conclusion we drew from the plots are further confirmed using the two metrics, since it is visible that the combination of a low number of particles and a very low or very high resampling frequency increases the mean square error and

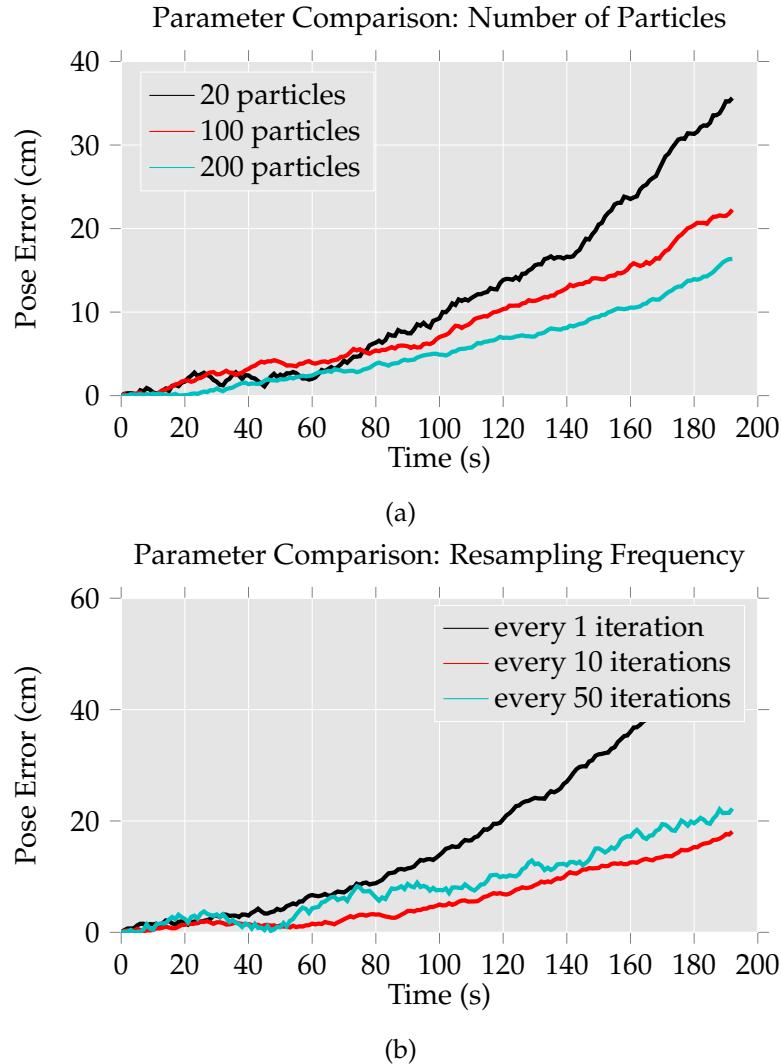


FIGURE 4.4: (a) The influence the number of particles has on the pose error. (b) The influence the resampling frequency parameter has on the error.

deviation by more than a factor of 4.

Particle Number	Resampling frequency	MSE	RMSD
20	10	301.6	287.3
100	10	61.1	55.7
200	10	46.4	41.3
100	1	252.0	240.8
100	10	61.1	55.7
100	50	168.4	157.5

TABLE 4.1: Values of the *MSE* and *RMSD* metrics on different sets of parameters.

## 4.3 Experiments on Global Map Matching

### 4.3.1 Absolute Localization Results

Contrary to relative localization, which is the ability to track the robot's pose, absolute localization is the ability to maintain the tracked pose which diverges due to accumulated errors over time. For this purpose, we will test the system in a long-range scenario where localization drifts are manifested and assess the robustness of the Pose Correction module (see Section 2.4). Similarly to the pose estimation experiments, the expected error of the corrected pose should ideally be in the same order of magnitude as the size of one cell of the global map, in consideration of the fact that the global map's resolution is higher than the resolution of the local map.

In the first scenario we evaluate the pose correction functionality in a terrain with average feature richness, i.e. a terrain that contains an average amount of elevation features such as rocks. Figure 4.5 presents the view of such terrain as a dense point cloud and Figure 4.6 shows the traverse inside it. It is visible that the SLAM estimation (red path) has a notable drift from the ground truth values (black path). Near the end of the traverse, we can observe the correction of the robot's pose due to a successful match of the local to global elevation map.

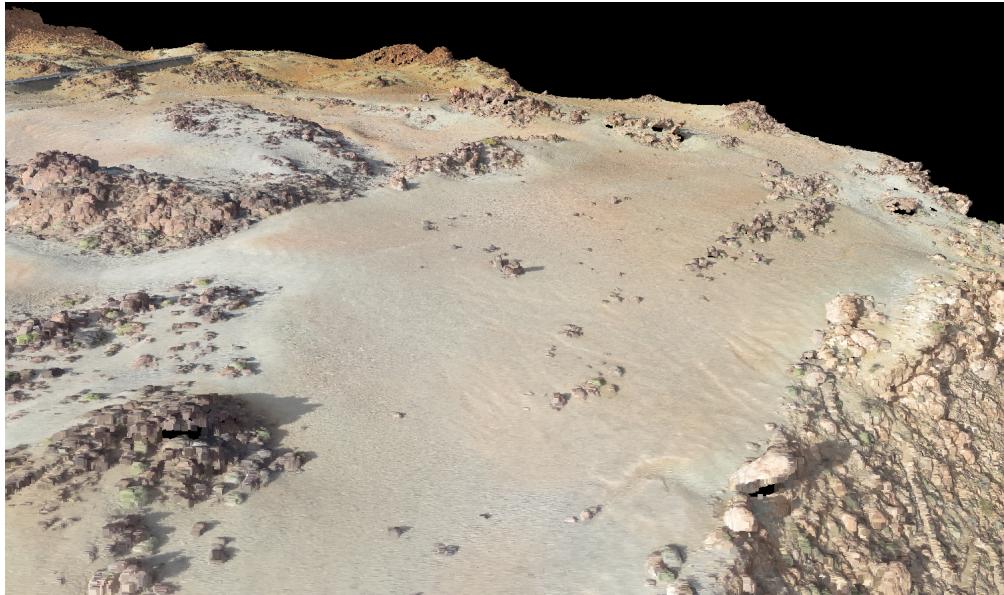


FIGURE 4.5: 3D view of the terrain evaluated in the second experiment.

In addition, we present the orthographic projections of the gradient of the global elevation map (Figure 4.7) where the elevation features are more visible.

The correction of the accumulated error is more visible in the plot of Figure 4.8. The score of the yielded result from the global to local map matching, as well as the rate of correction, are visible in Table 4.2. We can notice that the score of the match is

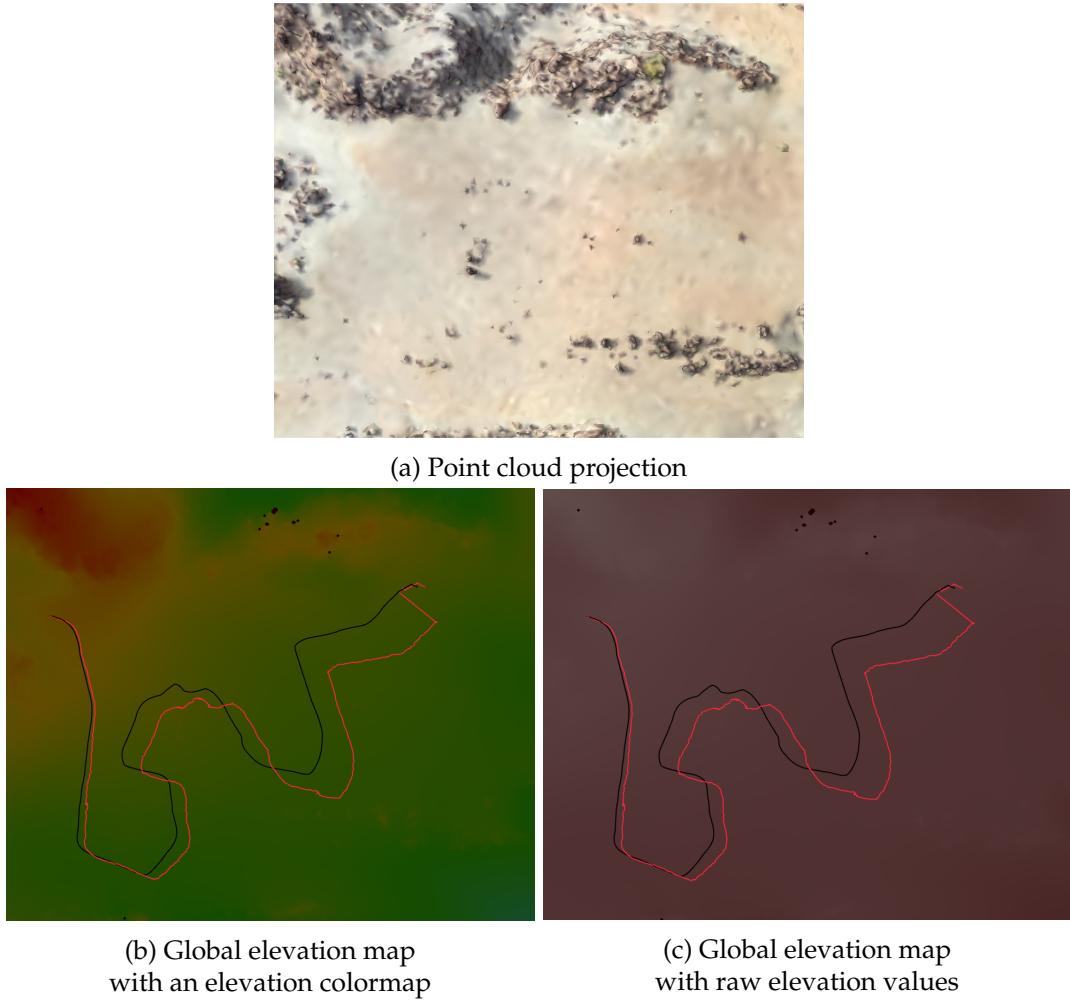


FIGURE 4.6: Orthographic views of the environment and the robot’s traverses in it. The black and red colors correspond to the ground truth and SLAM paths accordingly. The pose correction is visible in the end of the traverse.

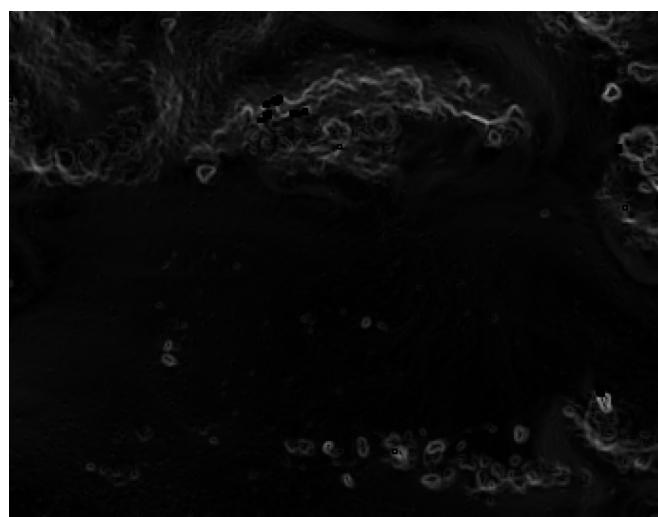


FIGURE 4.7: The global gradient map that the local map is matched to for pose correction.

97.1% (where 100% is a perfect match - as defined in Section 2.4.1) and since the matching threshold was set to 95%, the match is accepted and used to correct the robot's pose by 98.6%.

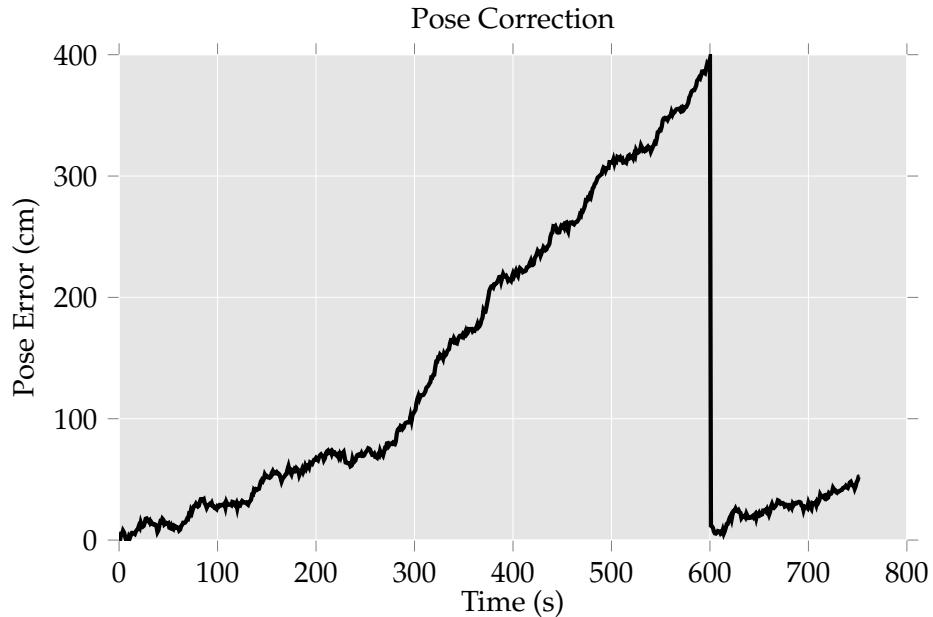


FIGURE 4.8: Pose correction results of second experiment.

Execution time of one iteration of matching	630 ms
Total execution time of matching	12.6 sec
Percentage of correction in position	98.6%
Percentage of correction in orientation	82.9%
Score of the matched location	97.1%
Acceptance threshold of match	95%

TABLE 4.2: Resulting time, correction and score of the match found.

As we already mentioned, the exact moment of correction is determined by the two criteria that were defined in the implementation of the Pose Correction module (see Section 2.4.2). However, since both of them are straightforward, their experimental validation is left out.

### 4.3.2 Map Resolution Viability

In this experiment, we validate and measure the performance of our map matching technique for absolute localization by performing the same scenarios with different combinations of resolutions for the local and global maps. This is done with the aim to draw conclusions regarding the viability of our approach w.r.t. the available global map data as well as the processing capabilities of a robot.

Figure 4.9 presents the local and global elevation maps with distinct resolution values. Upon matching these maps, we observe the score of each match for every combination of resolutions (Table 4.3). We can notice that a low resolution global map has more impact in the map matching score than a low resolution local map. In fact, a global map with a resolution of 2.0 m/pixel yields poor results for every resolution of the local map. On the other hand, a global map with a resolution of 1.0 m/pixel has the possibility to yield a high score only when the resolution of the local map is high enough.

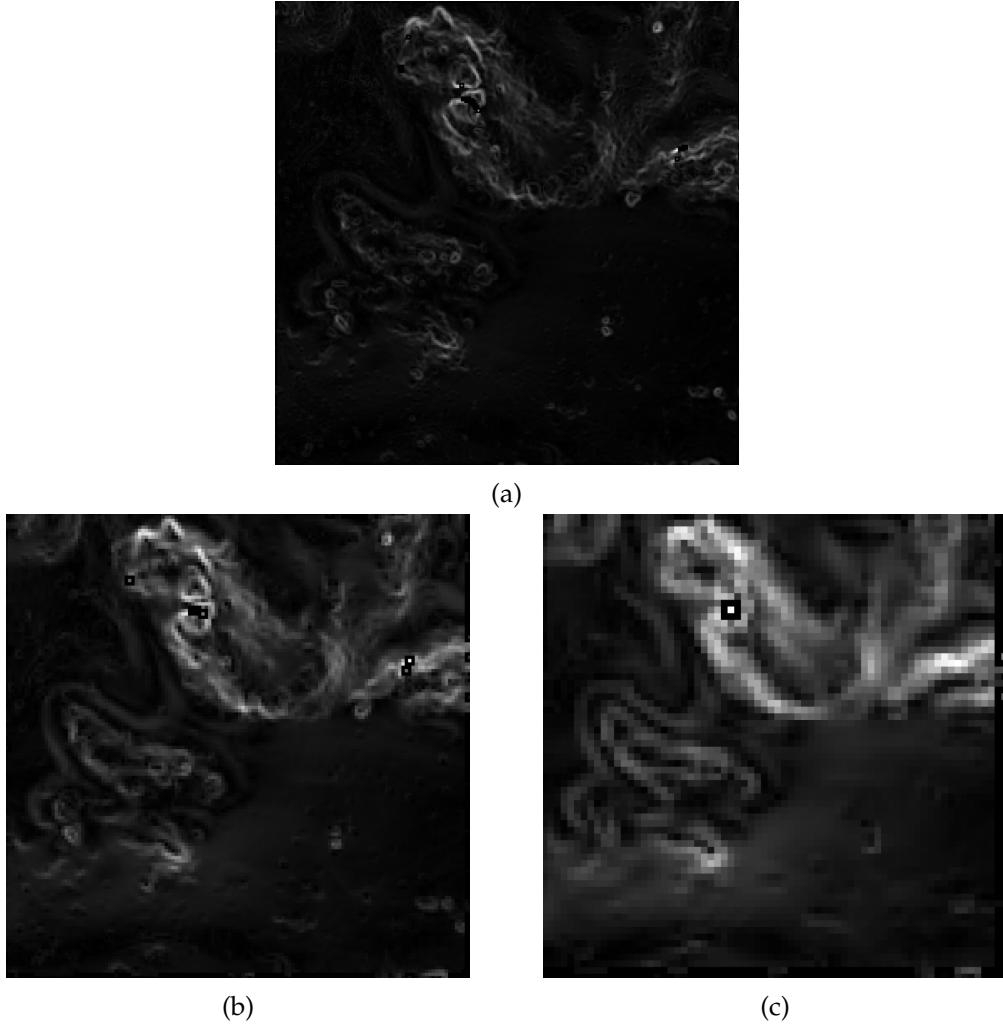


FIGURE 4.9: Comparison of global gradient map resolution. (a) Resolution of 0.5 m/pixel. (b) Resolution of 1.0 m/pixel. (c) Resolution of 2.0 m/pixel.

Although the size and resolution of the local map are parameters that we determine, they are highly limited by the on board processing power of the robot, since the higher the resolution the more computations must be performed. On the other hand, the resolution of the global map is a parameter that is determined by the equipped camera sensors of the orbiters and the reconstruction techniques applied to the imagery received from those. As the technology of the space industry advances, we

Local map resolution (m)	Global map resolution (m)	Matching score
0.1	0.5	97%
0.1	1.0	76%
0.1	2.0	48%
0.2	0.5	91%
0.2	1.0	67%
0.2	2.0	32%
0.5	0.5	72%
0.5	1.0	36%
0.5	2.0	25%

TABLE 4.3: Score of match for combinations of local and global map resolutions.

can expect to observe in the next few years an increase in the aforementioned resolution parameters and, consequently, an increase in efficiency of the map matching techniques. As an example, the HiRISE on board camera of the *Mars Reconnaissance Orbiter* can currently achieve a resolution of 30 cm/pixel with the possibility to acquire stereo image pairs with a precision better than 25 cm/pixel over locations of high priority, i.e. potential landing site of future missions to Mars.

## Chapter 5

# Conclusion

### 5.1 Thesis Summary

This thesis has considered the problem of perception for rovers that aim to navigate autonomously in a planetary terrain. We constructed an approach that is able to tackle the SLAM problem in a relative context by making use of elevation mapping techniques using 3D point clouds and by employing a particle filter which utilizes a scan matching method. In addition, we extended the approach with the aim to correct errors in position and orientation that accumulate over time using a template matching method that locates the local elevation map in a globally referenced one.

The proposed system was implemented as a standalone library and was integrated with the software components of ESA's HDPR rover. The integration with the HDPR's system was achieved using the ROCK robotics framework. This allowed the utilization of a rover with a sensor setup that bears close resemblance to the one of the ExoMars mission.

Moreover, we demonstrated the developed system using collected data from a Mars analogue (Chapter 3.4) and proved its advantages and weaknesses in short and long range scenarios. Specifically, the proposed method appeared to perform efficiently in terrains that contained an average amount of elevation features (e.g. rocks). However, in terrains that are characterized by the absence of such features, the global pose correction mechanism fails to provide accurate matches. Finally, we examined the influence the resolutions of the local and global maps have on the matching technique.

### 5.2 Contributions

The main contribution of this thesis in terms of research, is the development of a novel technique for minimizing the localization drift with global map matching in addition to solving the SLAM problem for a planetary rover.

For this purpose, we developed an open-source and research-oriented C++ library with the following characteristics:

- suitable for rough terrain navigation by utilizing 2.5D (elevation) maps
- ability to correct the robot's global pose using map matching of local (robot-centric) and global (orbiter) elevation maps
- sensor-agnostic data registration using point clouds (enables the support for sensor such as lidar, stereo camera, ToF camera etc.)
- automatic sensor fusion when multiple point cloud inputs are provided without prior configuration
- adaptive design to fit the needs of different robots and applications.

### 5.3 Directions for Future Extensions

As we already mentioned, SLAM approaches on planetary rovers have an important constraint which is the limited on board processing power. Hence, it is important to design a lightweight system that requires as low computations as possible.

With that in mind, a possible extension of the proposed particle filter algorithm would be to model the space of the problem, i.e. localization, in a discretized way instead of a continuous one. This can be achieved by sampling particles in a 2D grid that has the same resolution as the local map. As a result, particles that fall in the same cell of the grid are eliminated and are replaced by new particles.

Another direction for future work would be to exploit the uncertainty of the robot's pose to improve the quality of the local map. This can be accomplished using a neighborhood fusion method [Fankhauser et al., 2014] by fusing nearby cells for each cell of the map according to the variance of the robot's translation. The variance values could be extracted directly from the particle filter, by calculating the distribution of the particles, or by using the motion model of the robot.

Finally, the global elevation map could be further utilized by using it to initialize the cells of the local map. This could provide a starting point for the fusion of new point cloud data in the local map and increase the overall quality of the map.

### 5.4 Applications

Besides planetary scenarios, the proposed approach of this thesis can also be used in Earth applications, with the obvious one being autonomous driving vehicles. Similarly to a planetary rover, self-driving cars require high precision in terms of perception.

Although the usage of GPS data in such vehicles helps in the global localization problem, its availability cannot be guaranteed at every possible location, especially in non-urban areas. A way to utilize the technique we developed in such applications, would be to exploit global maps that have been *a priori* reconstructed to contain information about the terrain's elevation. This could provide self-driving vehicles with a mechanism to correct possible drifts in position and orientation.



# Bibliography

- Boukas, Evangelos et al. (2016). "HDPR: A Mobile Testbed for Current and Future Rover Technologies". In: *International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*.
- Creman, L. and R. M. Murray (2005). "Uncertainty-based Sensor Fusion of Range Data for Real-time Digital Elevation Mapping (RTDEM)". In: *ICRA*.
- Douc, Randal and Olivier Cappé (2005). "Comparison of Resampling Schemes for Particle Filtering". In: *Image and Signal Processing and Analysis*.
- Fankhauser, Péter et al. (2014). "Robot-Centric Elevation Mapping with Uncertainty Estimates". In: *International Conference on Climbing and Walking Robots (CLAWAR)*.
- Fox, D. et al. (2001). "Particle Filters for Mobile Robot Localization". In: *Statistics for Engineering and Information Science*.
- Overview of the Optical Design and Performance of the High Resolution Science Imaging Experiment (HiRISE)* (2005).
- Hashemi, Nazanin Sadat et al. (2016). "Template Matching Advances and Application in Image Analysis". In: *American Scientific Research Journal for Engineering, Technology, and Sciences*.
- Hinks, Tommy, Hamish Carr, Linh Truong-Hong, et al. (2013). "Point Cloud Data Conversion into Solid Models via PointBased Voxelization". In: *Journal of Surveying Engineering*.
- Huesing, J. et al. (2014). *CDF Study Report: MarsFAST Rover*.
- Jayanthi, N. and S. Indu (2016). "Comparison of image matching techniques". In: *International Journal of Latest Trends in Engineering and Technology*.
- Jefferies, Margaret E. and Wai-Kiang Yeap (2008). "Robotics and Cognitive Approaches to Spatial Mapping". In: *Springer Tracts in Advanced Robotics*.
- Joyeux, Sylvain and Jan Albiez (2011). "Robot development: from components to systems". In: *6th National Conference on Control Architectures of Robots*.
- Kümmerle, Rainer et al. (2009). "On measuring the accuracy of SLAM algorithms". In: *Autonomous Robots*.

- Ochoa, Benjamin L. and Serge J. Belongie (2006). "Covariance Propagation for Guided Matching". In:
- Rusu, Radu and Steve Cousins (2011). "3D is here: Point cloud library (PCL)". In: *IEEE International Conference on Robotics and Automation 2011 (ICRA 2011)*.
- Vago, Jorge L. et al. (2017). "Habitability on Early Mars and the Search for Biosignatures with the ExoMars Rover". In: *Astrobiology*.
- Zhang, Zhengyou (1994). "Iterative point matching for registration of free-form curves and surfaces". In: *International Journal of Computer Vision*.