ARISTOTLE UNIVERSITY OF THESSALONIKI

SLAM for Autonomous Planetary Exploration using Global Map Matching

Author:
Dimitrios GEROMICHALOS

Supervisor:
Assoc. Prof. Loukas PETROU

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

Write abstract here...

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

Mention the scope, its state, the current issues and what is needed to solve them

- Scope: planetary exploration (for scientific purposes)
- State: past and current missions, their purpose and outcome/state
- Issues: no constant communication, manual command sequence, unsafe/upredictable conditions
- Needed: real-time and low-supervision system/platform

1.2 Problem Statement

Mention the environment and the robot's equipment and purpose
Mention that the problem is given X inputs to calculate Y outputs

- Environment: extreme planetary terrains with anomalies (rocks and craters) and high elevation variance
- Equipment: mechanical and sensor configuration
- Purpose: autonomous (low-supervision) navigation
- Inputs: odometry pose, sensor inputs, imu, global map
- Outputs: corrected pose, local elevation map

1.2.1 Presumptions

Explain what presumptions are made to the problem

- Initial global position is known

- Environment is static (not true for autonomous driving applications of algorithm)
- Environment is unknown (no high resolution a priori maps) (not true for autonomous driving applications of algorithm)
- Terrain is single layered (no bridges etc.)

1.3 Literature Review

1.3.1 SLAM

- What is SLAM
  - typical explanation in literature
  - categories
- How is SLAM implemented
  - volumetric/feature based approaches
  - grid-based fast slam
  - etc. etc. etc.

1.3.2 Planetary Absolute Localization

Mention how is absolute localization achieved in planetary applications

- Feature based approaches
- Skyline based approaches
- map based approaches

1.4 Thesis Objectives and Outline

1.4.1 Research Objectives

Mention what are the main objectives in terms of research

- Develop a novel technique for minimizing drift in global localization with global map matching and an execution strategy (criteria)
- Determine under what circumstances (i.e. resolution, local and global) can the orbital imagery be useful for SLAM

- Examine and quantify the gains (better localization, by providing a navigable map with the resolution and dense distribution restrictions that come with it) and loses (processing overhead)

1.4.2 Outline

Explain how this thesis is structured

Chapter 2

System Architecture

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 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 in order 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

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

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

$$p(x_{0:T},\, m_{1:T} \mid z_{1:T},\, u_{1:T}) = p(x_{0:T} \mid m_{1:T-1},\, z_{1:T},\, u_{1:T})\; p(m_{1:T} \mid x_{0:T},\, z_{1:T}) \tag{2.2}$$

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 accummulated 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 in order to find 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.

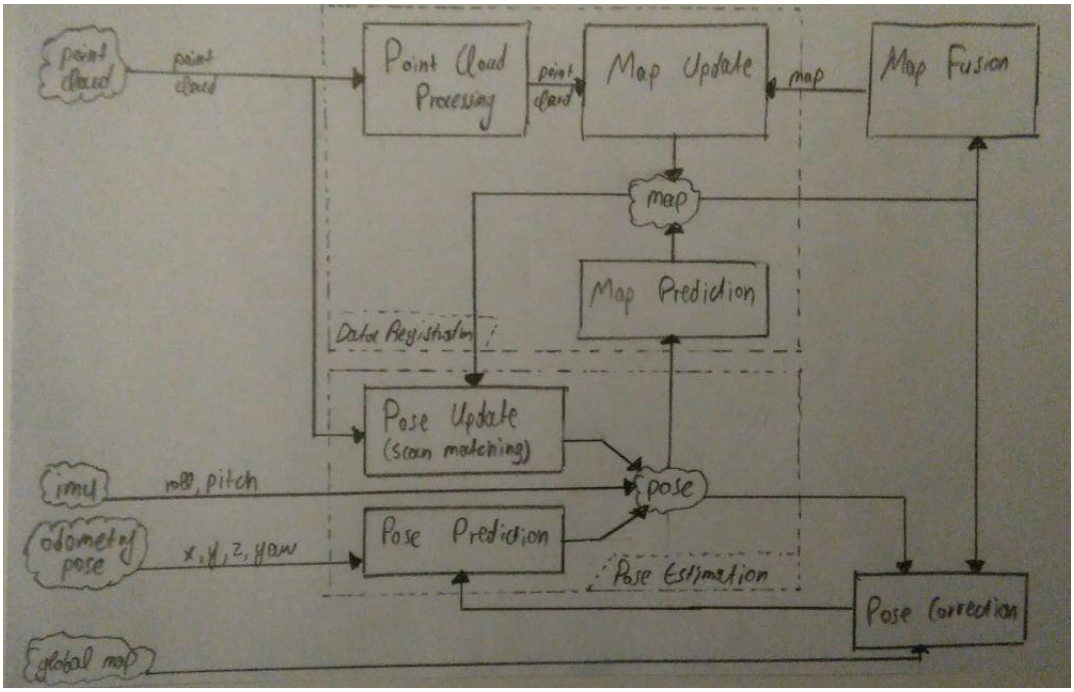

FIGURE 2.1: The high-level design diagram showing the submodules that comprise the system as well as the data flow.

2.2 Data Registration

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

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

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). 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 in order 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. 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). 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 downsampling 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.

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 be achieved by

$$\mathbf{t}_{MP} = \mathbf{R}_{SM}\mathbf{t}_{SP} - \mathbf{t}_{SM} \tag{2.4}$$

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

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

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, σ_z^2. As a result, each point will contain in addition to a x and y position, a one-dimensional Gaussian distribution for the height. This calculation is different for every sensor, since it depends on the sensor's characteristic. For a stereo camera, we approximate the variance as

$$\sigma_z^2 = \dots \tag{2.6}$$

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 the map consists of two layers

- the layer of *mean* elevation (μ_z) and
- the layer of elevation *variance* (σ_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 in order to reduce 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 2.4). 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, which given two Gaussian distributions $\mathcal{N}(\mu_1, \sigma_1^2)$ and $\mathcal{N}(\mu_2, \sigma_2^2)$, can be implemented as

$$innovation = \mu_2 - \mu_1 \tag{2.7}$$

$$gain = \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2} \tag{2.8}$$

$$\mu_1 = \mu_1 + gain \cdot innovation \tag{2.9}$$

$$\sigma_1^2 = (1 - gain) \cdot \sigma_1^2 \tag{2.10}$$

Algorithm 1 Registration of point cloud to map

M_μ: mean elevation layer of the map
M_{σ^2}: variance elevation layer of the map
P: point cloud - set of $\{x, y, z\}$
V: point cloud variances - set of $\{\sigma_z^2\}$

for each p in the P **do** ▷ iterate through all points
    if position of p is outside of map **then**
        continue

    $\mu_{point} \leftarrow p[z]$ ▷ projection of the point
    $\sigma^2_{point} \leftarrow V[p]$

    $index \leftarrow find\_map\_index(p[x], p[y])$ ▷ corresponding cell in grid

    $\mu_{cell} \leftarrow M_\mu[index]$
    $\sigma^2_{cell} \leftarrow M_{\sigma^2}[index]$

    if μ_{cell} or σ^2_{cell} is not initialized **then**
        $\mu_{cell} \leftarrow \mu_{point}$
        $\sigma^2_{cell} \leftarrow \sigma^2_{point}$
    else
        $\mathcal{N}(\mu_{cell}, \sigma^2_{cell}) \leftarrow \mathcal{N}(\mu_{cell}, \sigma^2_{cell}) \cdot \mathcal{N}(\mu_{point}, \sigma^2_{point})$

    $M_\mu[index] \leftarrow \mu_{cell}$
    $M_{\sigma^2}[index] \leftarrow \sigma^2_{cell}$

2.3 Data Fusion

The goal of this module 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. In particular, we developed two submodules to perform fusion in separate stages of the pipeline. These are explained in the following sections.

2.3.1 Sensor Fusion

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 oversmooth 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.

2.4 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. 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

$$\mathbf{x} = [x \quad y \quad z \quad roll \quad pitch \quad yaw]^\intercal \tag{2.11}$$

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

$$\mathbf{x} = [x \quad y \quad z \quad yaw]^\mathsf{T} \tag{2.12}$$

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

$$\mathbf{x} = [x \quad y \quad yaw]^\mathsf{T} \tag{2.13}$$

which is also referred to as

$$\mathbf{x} = [x \quad y \quad \theta]^\mathsf{T} \tag{2.14}$$

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

$$p(x_{0:T} \mid m_{1:T-1},\, z_{1:T},\, u_{1:T}) \tag{2.15}$$

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 modelled 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 X. These steps are explained in detail in the following sections.

2.4.1 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

$$f(x \mid \mu,\, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2.16}$$

where μ is the mean of the distribution, σ is the standard deviation and σ^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: $x \sim \mathcal{N}(\mu_x,\ \sigma_x^2)$ ▷ sample the states from their distributions
5: $y \sim \mathcal{N}(\mu_y,\ \sigma_y^2)$
6: $\theta \sim \mathcal{N}(\mu_\theta,\ \sigma_\theta^2)$

2.4.2 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

$$x_t^i \sim \mathcal{N}(x_{t-1}^i + \Delta x_t,\ \sigma_x^2) \tag{2.17}$$

$$y_t^i \sim \mathcal{N}(y_{t-1}^i + \Delta y_t,\ \sigma_y^2) \tag{2.18}$$

$$\theta_t^i \sim \mathcal{N}(\theta_{t-1}^i + \Delta \theta_t,\ \sigma_\theta^2) \tag{2.19}$$

where x_i, y_i and θ_i represent the state of the particle, Δx, Δy and $\Delta\theta$ are the delta inputs from odometry and σ_x^2, σ_y^2 and σ_θ^2 are the variances of the states that define the Gaussian noise that will be added to the particle.

2.4.3 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.

In order to achieve 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 $S = \{s_1, s_2, \ldots, s_{N_s}\}$ and a reference point cloud $R = \{r_1, r_2, \ldots, r_{N_r}\}$ is by computing the sum of the squared error

$$e = \frac{1}{N_r} \sum_{i=1}^{N_r} ||s_i - r_i||^2 \tag{2.20}$$

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

$$w_i = \frac{1}{icp\_fitness\_score} \tag{2.21}$$

2.4.4 Resampling

The act of resampling the set of particles of the filter is done in order to ensure that the particle population models the uncertainty of the problem as close as possible. Although there are several variants that can be exploited to solve the resampling problem depending on the situtation, 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

$$w_i = \frac{w_i}{\sum_{k=0}^{N} w_k} \tag{2.22}$$

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 2.4.2).

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.

2.4.5 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.

Algorithm 3 Estimation of final pose from particle population

P: set of particles
p: particle - set of $\{states, weight\}$

$S \leftarrow \{\}$ ▷ candidate particles
for each p in P **do**
    if $p[w] > w_{threshold}$ **then** ▷ remove uncertain particles
        $S \leftarrow S$ + p

Sort S by weight
$M \leftarrow S[1:k]$ ▷ pick top k particles

$w_{sum} \leftarrow \sum_{i=1}^{k} w_i$
$x \leftarrow \sum_{i=1}^{k} w_i x_i \ / \ w_{sum}$ ▷ weighted average of each state
$y \leftarrow \sum_{i=1}^{k} w_i y_i \ / \ w_{sum}$
$\theta \leftarrow \sum_{i=1}^{k} w_i \theta_i \ / \ w_{sum}$

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 $\mathcal{N}(\mu_{imu},\ \sigma^2_{imu})$ where μ_{imu} is the measurement value from the sensor and σ^2_{imu} is the variance which is a parameter that represents how reliable the sensor is. The second Gaussian is represented as $\mathcal{N}(\mu_\theta,\ \sigma^2_\theta)$, where μ_θ is the yaw estimate that was extracted in the previous step and σ^2_θ is the variance of the distribution. It can be calculated by applying the formula for a weighted standard deviation as

$$\sigma_\theta = \sqrt{\frac{\sum_{i=1}^{N} w_i(\theta_i - \mu_\theta)^2}{\sum_{i=1}^{N} w_i}} \tag{2.23}$$

where N is the number of particles, w_i is the weight of each particle and θ_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).

2.5 Pose Correction

Although the robot's pose is constantly tracked using a particle filter in the Pose Estimation module (Section 2.4), 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. In order to bound 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 contast 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.

2.5.1 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 correspondece 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. 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 be 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 2.4), 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

$$K_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \tag{2.24}$$

  and in the vertical direction is

$$K_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} \tag{2.25}$$

  Thus, the two gradients are computed as

$$G_x = K_x * I \tag{2.26}$$

$$G_y = K_y * I \tag{2.27}$$

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

$$G = \sqrt{G_x^2 + G_y^2} \tag{2.28}$$

  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° to 10° with a step of 1°. 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.

  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

  $$R(x,y) = \sum_{x',y'} (T(x',y') - I(x+x',y+y'))^2 \tag{2.29}$$

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

  2. the Cross Correlation, which is expressed as

  $$R(x,y) = \sum_{x',y'} (T(x',y') \cdot I(x+x',y+y')) \tag{2.30}$$

  3. the Correlation Coefficient, which is expressed as

  $$R(x,y) = \sum_{x',y'} (T'(x',y') \cdot I'(x+x',y+y')) \tag{2.31}$$

     where

  $$T'(x',y') = T(x',y') - \frac{\sum_{x'',y''} T(x'',y'')}{width \cdot height} \tag{2.32}$$

  $$I'(x+x',y+y') = I(x+x',y+y') - \frac{\sum_{x'',y''} I(x+x'',y+y'')}{width \cdot height} \tag{2.33}$$

  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 2.30) 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:

$$R(x,y) = \frac{\sum_{x',y'}(T(x',y') \cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}} \tag{2.34}$$

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:

$$x_{map} = \left\lfloor \frac{h}{2} \right\rfloor - x_{image} \tag{2.35}$$

$$y_{map} = \left\lfloor \frac{w}{2} \right\rfloor - y_{image} \tag{2.36}$$

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

  Finally, we apply the correction (x, y, θ) 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 2.4.2.

2.5.2 Criteria Checking

TODO

Elevation Features Checking

Although the aforementioned matching technique can provide accurate results, it highly depends on the terrain the robot is navigating in. The reason is that ...

Explain the steps and the threshold parameter selection

(i) Create slope map from local elevation map by calculating the elevation gradient (ii) Remove values in slope map below a specific threshold (iii) Add remaining values to get absolute slope of map (iv) Compare to threshold

Traversed Distance Checking

Explain the step and the threshold parameter selection

(i) calculate traversed distance since last pose correction step (ii) compare to threshold

Chapter 3

System Implementation

3.1 Library

Mention GA SLAM library and tools used to create it etc.

3.1.1 Concurrency

Explain the algorithm's processing limitations and the circumstances under which it can run
Explain how concurrency can help and how it's implemented
Add timing diagram explaining the issues and how they are solved

3.1.2 Robotic Software Framework

Explain briefly ROCK, its tools, workflow and advantages/disadvantages
Explain briefly ROS etc.
Mention that the standalone library has interface to both frameworks

3.1.3 Orbiter Data Preprocessing

Explain that orbital imagery is emulated using drone imagery
Explain quickly how a 3D point cloud is reconstructed from the orbital imagery
Mention NASA's HiRISE (High Resolution Imaging Science Experiment)

Explain the steps for creating a global map from orbital point cloud:

1. Voxelize point cloud to the desired resolution
2. Crop the point cloud in the order of magnitude as local map's size
3. Smooth the point cloud using a SOR filter
4. Transform point cloud to robot's pose (using the initial absolute position)

Add figures of orbiter point cloud (raw & processed) and the orbiter map
Add figures from NASA's HiRISE depicting actual Mars point clouds

3.2 System Architecture

Explain how the algorithm can be integrated in a system and what components are needed (stereo, odometry etc.)
Add component diagram showing the architecture and how the different components are integrated

3.3 Planetary Rover Testbed

Explain lab's main rovers (HDPR & Exoter) and their mechanical/sensor configuration
Add figure depicting and explaining the rover(s)

Chapter 4

Experimental Validation

4.1 Scope of Experiments

Mention why mapping experiments are out of the scope
Explain briefly the purpose of the following experiments
Explain the type of the experiments

- Purpose: validate the algorithm and its robustness in various terrains and configurations
- Type: SIL tests (precollected data)

4.1.1 Environment

Explain the environment of the collected data (location, traversed paths etc.)
Add figures showing the said environment/traverses

4.1.2 Metrics

Mention the metrics used

- MSE of pose graph
- mean variance of pose graph,
- (execution times)
- etc.

4.2 Experiments on Pose Estimation

4.2.1 Relative Localization Results

Explain what the experiment is (test the accuracy of the particle filter)
Explain what is the expected accuracy of the estimation (1 cell size of local map)

Add bird's eye view of XY plot comparing:

- ground truth 2D position
- (Odometry)
- PF estimate

Add plot showing the error vs the number of particles used
Add plot showing the mean variance of the estimate vs the number of particles used
Add table showing the execution time vs the numner of particles used
Add plot showing the error vs the resampling frequency

Discuss the above results and what are the advantages/drawbacks while tuning each parameter

4.3 Experiments on Global Map Matching

4.3.1 Absolute Localization Results

Explain what the experiment is (test the drift correction on long range traverses)
Explain what is the expected accuracy of the correction (1 cell size of global map)

Add bird's eye view of XY plot comparing:

- ground truth 2D position
- (Odometry)
- PF estimate without global correction
- PF estimate with global correction

Repeat test in environment with different distribution of features
(Repeat test in environment without features to show the limitation of the approach)

Add table with the matching accuracy of the different experiments

Discuss the threshold value that should be chosen on such environments

4.3.2 Map Resolution Viability

Explain what the experiment is (test under what resolutions can we expect a drift correction)

Add figure showing local & global maps with different resolution
Add table comparing global vs local map resolutions using the correction error (actual offset - matched location)

Discuss the global and local map resolution of the current Mars missions and how these will change in the future (e.g. NASA is planning to have 0.25m orbiter map resolution by 2020)

Chapter 5

Conclusion

5.1 Thesis Summary

Mention briefly what was discussed in this thesis and what the experiments proved

5.2 Contributions

Mention the research and the development contribution of this thesis

- Research contributions (same as objectives of Introduction)
- Development contributions (system features):
  - Suitable for rough terrain navigation by utilizing 2.5D (elevation) maps
  - Global pose correction using map matching of local (robot-centric) and global (orbiter) elevation maps
  - Sensor-agnostic data registration using point clouds (support for lidar, stereo camera, ToF camera etc.)
  - Automatic sensor fusion when multiple inputs are provided (without prior configuration)
  - Adaptive design to fit the needs of different robots and applications

5.3 Directions for Future Extensions

Mention future work and possible extensions to the algorithm

- Use a discretized particle filter to gain advantage of the grid nature of the map
- Use global map for complete (all initial cells) or partial (new unknown cells at the map's edge) of the local map

- Use the motion model in addition to using the particle cloud distribution to determine the estimated pose's uncertainty
- Use a method (e.g. SOR) to minimize outliers in input point cloud
- Apply a threshold to points in projected point cloud that fall in the same map cell
- Implement a strategy for selecting a match in template matching when matches with similar score are a. close to each other b. far from each other

5.4 Applications

Talk about other possible applications except for the planetary one

- Autonomous driving vehicles in urban environment using a priori 3D reconstructed maps (i.e. Google Maps) - although GPS data helps in avoiding the drift of dead reckoning, global map can still be used for local map initialization
- Autonomous driving in rough non-urban areas were no predefined paths exists and GPS usage is limited
- Lunar teleoperated robot with SLAM for augmenting the operator's perception - processing is done on the station (i.e. ISS) since stereo images are sent anyway