7-1-2024

Geo-SLAM: Using Geometric Hierarchies to Support Localization and Mapping in Forest Environments

Daniel Castellarin

Geo-SLAM: Using Geometric Hierarchies to Support Localization and Mapping in Forest Environments

by

Daniel Castellarin

A thesis submitted to the
B. Thomas Golisano College of Computing and Information Sciences
Department of Computer Science
in partial fulfillment of the requirements for the
Master of Science Degree in Computing and Information Sciences
at the Rochester Institute of Technology
on July 1, 2024

Abstract

Robust mapping and localization can be challenging in environments like forests where easily observable landmarks are virtually indistinguishable from each other. In such environments, GPS or robot odometry may also be compromised by the landscape, tree cover, or method of locomotion. Prior work discussed a method for creating geometric hierarchies to describe the relative positions of unlabeled landmarks. Notably, the polygons from these hierarchies were used to perform accurate place recognition from observations containing partial overlap and sensor noise. Here, we use geometric hierarchies to perform robust data association for collections of virtually identical 2D landmarks in direct support of SLAM. Our system, GeoSLAM, utilizes polygon matching to aggregate landmark positions across consecutive observations, reducing the noisiness of input data. We also maintain a global geometric hierarchy of the environment to enable fast and accurate robot pose estimation. Simulation results empirically demonstrate that this system facilitates accurate, real-time localization for robots experiencing significant sensor noise when exploring environments that contain varying densities of indistinguishable landmarks.

Committee Approval:

Date of Signature

Zach Butler
Interim Department Chair, Department of Computer Science
Thesis Advisor

Reynold Bailey
Professor, Department of Computer Science
Thesis Committee Reader

Fawad Ahmad
Assistant Professor, Department of Computer Science
Thesis Committee Observer

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Localization is the concept of understanding one's position relative to surrounding objects. It is required for autonomous systems to successfully implement localization so that they can accurately and precisely act on their environment. For robotics applications, this task is usually performed while the robot creates a map of its environment using a process called Simultaneous Localization and Mapping (SLAM).

For SLAM to work as intended, the robot must use its sensors to find frequent, easily detectable, and, most importantly, distinguishable landmarks in its environment [2]. In the absence of these landmarks, robots will often find distinct patterns (i.e. features) within its raw sensor data to describe unique locations in its environment. These functions are often called perception tasks. Regardless of how a robot performs these tasks, their success is vital for enabling the robot to maintain an accurate understanding of its position in the environment over time. Most importantly, robots must be precise when identifying unique locations across observations, because if it makes a mistake, it most likely will cause the system to crash.

Therefore, most of the development of autonomous systems goes towards robots that operate in highly controlled environments. Autonomous systems have become especially popular in the manufacturing and logistics industries [20] [16]. In these settings, engineers benefit from implementing infrastructure to improve the robot's understanding of the environment. For example, unique QR codes could be placed on various walls of a large warehouse. The position of each QR code would be known beforehand, so when the robot observes a QR code, it can easily confirm its location in the environment. Another common approach is implementing fixed active beacons in the environment, which also enable the robot to determine its own position relative to those beacons.

Even so, autonomous systems are being designed for a wide variety of environments. Robots that automatically perform cleaning and landscaping tasks are becoming more prominent on large campuses. Car manufacturers are in the process of designing and testing autonomous vehicles for use on public roadways. In indoor environments or on highways, there is typically a wide assortment of uniquely identifiable landmarks for autonomous systems to choose from.

Not all environments contain easily distinguishable landmarks, though. For example, forests are filled with assortments of trees, fallen logs, and other land formations. These objects are often hard to tell apart from each other and occur frequently enough that autonomous systems will not be able to accurately create associations between the landmarks they observe over time. On top of this, other methods of precise localization, like GPS, are not useful for robots that operate under the forest canopy due to interference caused by tree leaves and branches.

Recently, Nardari *et al.* [23] proposed a novel method for recognizing unique places in environments with virtually identical landmarks (like forests). They construct a single geometric hierarchy from the observed positions of landmarks in a 2D-plane. This hierarchy defines polygons from the spatial relationships between landmarks, which are ultimately used to identify unique locations in the environment. These polygons tend to be both unique and noise-resilient, making them ideal for localizing autonomous systems in environments where perception is difficult.

Although promising, their implementation of place recognition had certain weaknesses. First, their algorithm did not perform well in the presence of significant detection noise. In addition, its performance was only evaluated for observations of environments that have a density of landmarks similar to the density of trees in an average forest. Most importantly, when polygons from each observation's geometric hierarchy are stored separately, the computation time for place recognition will grow linearly with respect to the number of stored observations instead of the size of the map.

We believed that Nardari *et al.*'s methodology could be used to construct and maintain a global representation of a robot's environment filled with virtually identical landmarks. We sought to create an algorithm that would build a global geometric hierarchy, thus enabling real-time place recognition in environments where perception is difficult. In addition, we thought that by leveraging SLAM to improve the estimated positions of landmarks in the global map, we could ensure the map would continue to support accurate place recognition as the robot makes more noisy observations of its environment.

This thesis describes how we incorporated the place recognition technique developed by Nardari *et al.* into a complete SLAM algorithm called GeoSLAM. We explain what type of input this system expects, how we perform local bundle adjustment to reduce noise across observations, and how we maintain a global representation of the environment within SLAM. We also review various nuances found within our particular implementation. In order to demonstrate the stability and effectiveness of GeoSLAM, we evaluate its ability to accurately estimate the robot's pose and the positions of landmarks over time. We investigated the dependence of landmark density on GeoSLAM's localization accuracy and scalability. We also tested GeoSLAM across various detection and position noise levels to evaluate its adaptability when receiving input from low-precision sensing systems. Afterwards, we discuss the implications of GeoSLAM's performance and how it could be improved in the future.

Chapter 2

Related Work

GeoSLAM was developed after obtaining a deeper understanding of the concepts behind and applications of place recognition in the field of robotics. We first discuss the implementation of simultaneous localization and mapping (SLAM), how it utilizes place recognition through loop closure, and how loop closure detection was implemented to support the operation of systems belonging to forest environments.

2.1 Simultaneous Localization and Mapping

One of the most important developments in the field of autonomous robotics was simultaneous localization and mapping (SLAM). It enables robots that cannot make accurate individual observations of an unknown environment to build an increasingly accurate and cohesive map over time. Most importantly, it allows the robot to understand its own position in that environment. There are two major components to a SLAM system: a back-end and a front-end. Fig. 2.1 illustrates the interaction between these components in a SLAM system.

The SLAM back-end is designed to estimate the state of the robot and its environment. There are a few available approaches to performing this task. Filter-based methods which utilize concepts from probability theory, like ones that use an extended Kalman filter (EKF) or the FastSLAM algorithm, have classically been chosen for this component of the system [7]. Methods based on graph optimization are now the standard because they have been proven to have superior accuracy and efficiency [2].

The SLAM front-end is designed to take raw sensor input and extract important features from it so that the back-end can model the environment.

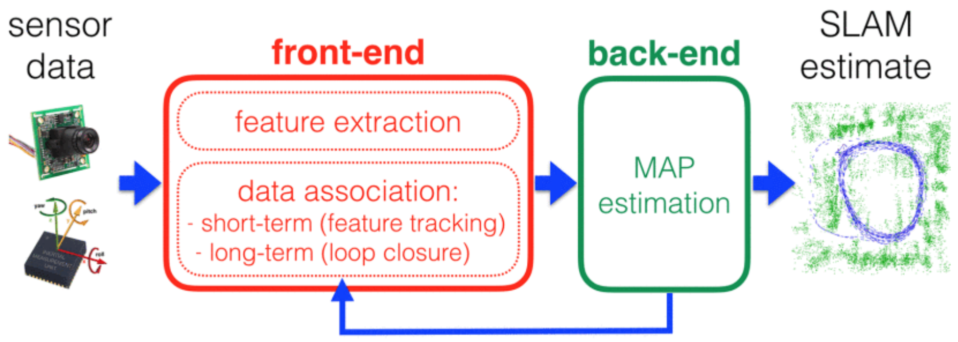


Figure 2.1: Front and back end of typical SLAM system. The back end can provide feedback to the front end for loop closure detection and verification [2].

The front-end is also tasked with associating measurements taken from these features with landmarks (i.e., unique locations in the environment) over multiple observations in the short-term (feature tracking) and long-term (loop closure). For example, if a single image was considered an observation for a simple Visual SLAM system, the front-end would identify pixels that represent a few distinguishable points (landmarks) in the environment and then extract measurements associated with those points to give to the back-end for processing.

The design of a SLAM system's front-end is often tied directly to what sensors the robot will be using. The two main contemporary methods are Visual SLAM and LiDAR-SLAM, developed around obtaining input data from cameras and LiDAR sensors respectively. Visual SLAM implementations can be further broken down into how they interact with the data, splitting them into feature-based methods [6] [22] and dense methods [24] [9] [8]. There are also implementations designed around the use of the RGB-D camera [33]. LiDAR SLAM techniques utilize point cloud data, which is a series of scattered points with accurate angle and distance information [15]. LiDAR sensors can interpret the environment in either two or three dimensions, and the front-end components of SLAM will differ depending on the dimensionality of the input [12] [32] [3]. In recent years, advances in deep learning have led to the integration of neural networks to solve specific tasks in front-end SLAM systems like extracting descriptors [19] and semantic interpretation of a scene [21].

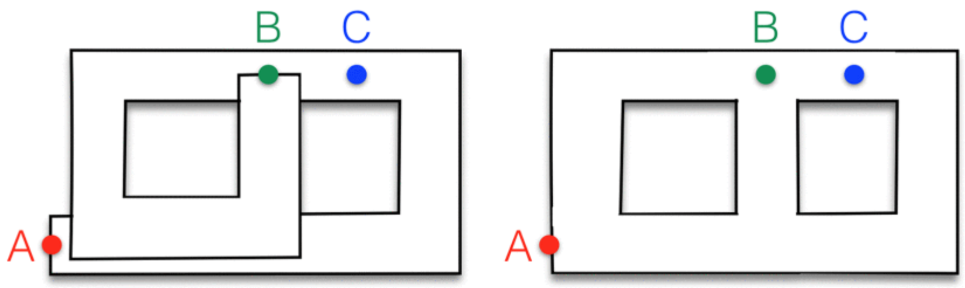


Figure 2.2: The robot explores an environment resembling two parallel hallways connected by three separate paths. Left: map built from odometry, which represents a single long corridor from positions A to B. This map fails to convey that points B and C are actually close in reality. Right: map built using SLAM; by leveraging loop closures, recognizes unique locations in the map to estimate the actual topology of the environment [2].

2.2 Loop Closure and Place Recognition

As part of the SLAM front-end, loop closure implementations must associate feature measurements from new observations with old landmarks. Practically, this is what allows loop closure to correct previously misunderstood notions about an environment (see Fig. 2.2). The most important aspect of a loop closure system is its ability to recognize when the robot has arrived at a location it has already seen (often called place recognition). Just like any other software algorithm in the field of robotics, it is expected that the method of loop closure is fast and computationally inexpensive.

The simplest possible approach to performing loop closure would be a brute-force analysis of eligible features, which is impractical for most robotics applications. One of the most popular techniques involves quantizing the feature space using bag-of-words models [27] to allow for more efficient searches. These can be improved using more complex vocabulary trees to organize descriptors hierarchically [25]. For systems that rely on visual information, accumulated words may not be matchable when illumination levels vary in the environment. Consequently, other techniques have been developed to account for this, such as gathering measurements for an object from different perspectives [4] and including spatial information in the descriptors [14]. Some descriptors are built for describing features that are given by different sensors, like FLIRT features for 2D LiDAR data [28].

There are a few things to consider when benchmarking loop closure systems. Detection outcomes can be categorized in four ways: true-positive, true-negative, false-positive, false-negative. Generally, loop closure detection systems attempt to achieve high precision across all recall values. The most common performance indicator of loop closure systems is the highest possible recall score with the least number of false-positive detections (as close to perfect precision as possible). This metric is very important since a single false-positive detection can often cause the SLAM system's back-end to break [30].

2.3 Loop Closure in Forest Environments

Solutions to the loop-closure task often rely on the environment having frequent, unique landmarks. However, in environments where the features that describe landmarks are virtually indistinguishable, like in a forest, methods like the bag-of-words approach are not viable [29]. Instead, systems would need to rely on spatial information to describe distinct places, which is traditionally acquired using LiDAR sensors. Methods like Geometrical Landmark Relations (GLARE) [13] transform 2D laser scans into pose invariant histogram representations. They compute a global descriptor for each observation and a local descriptor for each landmark. When matching observations, the global descriptors would be compared first, and then landmarks would be associated using the local descriptors. Although these methods were proven effective in some outdoor environments [13] [17], they can be susceptible to noise since they rely on space discretization, potentially causing a landmark to fall into different bins when matching observations.

Local methods of loop closure would not have this issue, though. Since they compute descriptors for regions around features, they can make successful landmark associations in the presence of partial overlap or occlusion. For example, Gawel *et al.* demonstrate how to use structural descriptors of the density of features' neighboring points to merge point cloud maps from different sensory systems [10]. Similarly, fast point feature histograms (FPFH) [26] compute surface normals of nearby points to create descriptors for each point. The normals are represented as angular features then stored in histogram bins.

One novel approach proposed by Li *et al.* [18] represented local LiDAR point cloud observations using Delaunay Triangulations to match landmarks in a global map. For each triangle, they create a descriptor by concatenating the area and perimeter values of itself and the triangles that share its sides. Delaunay Triangulations of noisy point sets maximize the minimal angle for all vertices, reducing the noise in angular values. They also inherently create

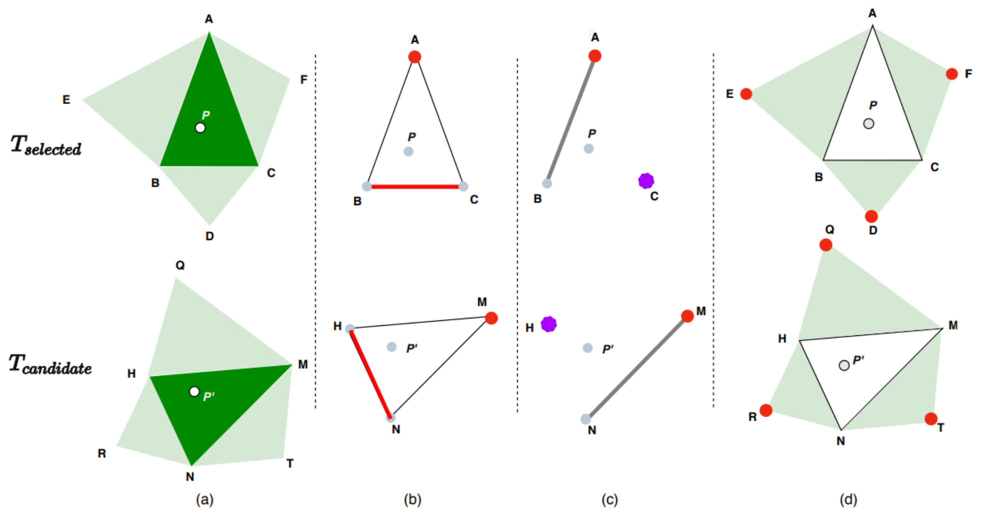


Figure 2.3: (a) The selected triangle $T_{selected}$ in the local observation with a candidate matched triangle $T_{candidate}$ from the global map. (b) Identify an initial point A in $T_{selected}$ with the corresponding point M in $T_{candidate}$ by comparing the triangles' side lengths. (c) Match the remaining points B, C with their corresponding points H, N. (d) Extrapolate correspondences to neighbor points E, D, F and Q, R, T [18].

local neighborhoods for every vertex (its set of connected vertices along the triangle edges). Notably, this method is relatively immune to point detection noise, because if some points are missed, most triangles will still be present so the system can recognize a previously seen location. The process of matching two descriptors is illustrated in Fig. 2.3.

The major disadvantage these local methods have is that they are limited by their computational complexity. The number of comparisons to be performed will grow with the number of points or landmarks in an observation. Since mobile robotics systems require real-time performance with limited resources, these solutions often become impractical.

In parallel to the work of Li *et al.*, Nardari *et al.* [23] proposed a local method that uses fewer descriptors than its contemporaries while being robust to noise and partial overlap. The details of this method will be further discussed in Section 3.2. Nardari *et al.* tested this methodology against GLAROT [17], a rotation invariant version of GLARE [13], and Li *et al.*'s [18] in map merging and simulated loop closure detection experiments, where it outperforms the others in both accuracy and robustness. The approach of Li *et al.* performed the best when only detection noise was introduced, but it did not hold up in the presence of position noise.

Despite displaying promising results, Nardari *et al.*'s methodology had some key weaknesses. Its performance has yet to be tested in denser environments, which would require the storage and comparison of more polygons per observation. More importantly, in worst case scenarios, the number of polygon comparisons would be quadratic (for two observations). Even though the number of comparisons should be less than similar methods of loop-closure (which need to compare two sets of landmarks together), in practical scenarios, the number of observations to compare against will scale linearly, too. Nardari *et al.* experimented with randomly sampling the polygons in each observation, but this decreased the accuracy of the method. Additionally, if some landmarks are not detected successfully in an observation, the performance of the method will drop. Altogether, implementing this method of loop closure by itself is not enough to guarantee adequate performance in practical robot applications.

Chapter 3

Background

The following sections describe fundamental concepts that are utilized in the implementation of GeoSLAM.

3.1 Frame Transformations

When mapping a robot's motion through 2D space, its instantaneous pose has three components: x_R, y_R, θ. This pose describes the robot's position and orientation relative to some fixed-reference frame, commonly known as the origin (see Fig. 3.1). In the domain of robot navigation, the origin of the robot's map is its starting position and orientation when it powers up.

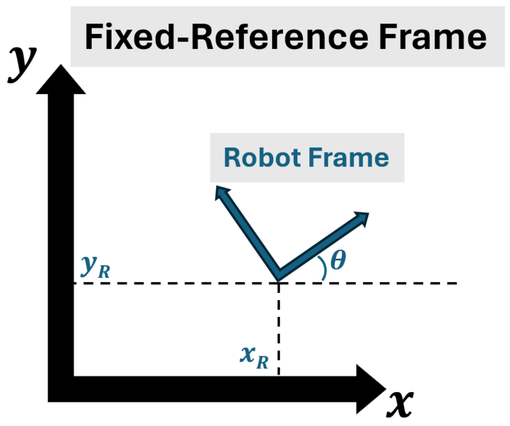


Figure 3.1: Depiction of a robot's pose (with components x_R, y_R, θ) relative to some fixed-reference frame.

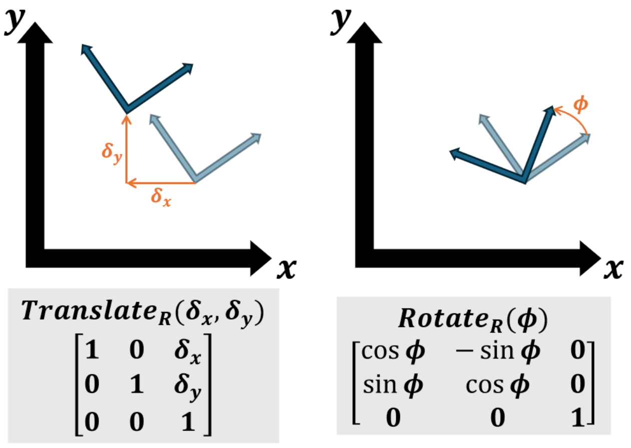

y
x
δ_y
δ_x
ϕ
$Translate_R(\delta_x, \delta_y)$
$\begin{bmatrix} 1 & 0 & \delta_x \\ 0 & 1 & \delta_y \\ 0 & 0 & 1 \end{bmatrix}$
$Rotate_R(\phi)$
$\begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Figure 3.2: Illustration of the translation (*left*) and rotation (*right*) of a rigid body.

In this context, the robot could also be described as a rigid body. When rigid bodies move through 2D space, they are capable of three degrees of freedom: translation along the x-axis, translation across the y-axis, and rotation about the normal of the 2D plane, as seen in Fig. 3.2. These are called pure transformations, and they can be defined mathematically using homogeneous transformation matrices.

A series of pure transformations can be combined to describe one complex movement between two poses. For example, in Fig. 3.3, robot R is initialized at pose (2,3,π). It then moves along forward by 3, turns $\frac{\pi}{2}$ to the left, and then moves forward again by 2. Intuitively, the robot's new pose would be (-1,1,$\frac{3\pi}{2}$). Similarly, we can represent this movement as a single homogeneous matrix by multiplying the pure transformations together (assuming +x is forward):

$$\begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 3 \\ 1 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

We can then multiply the combined transformation with the matrix representing the robot's previous pose to obtain its new pose:

$$\begin{bmatrix} -1 & 0 & 2 \\ 0 & -1 & 3 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & -1 & 3 \\ 1 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

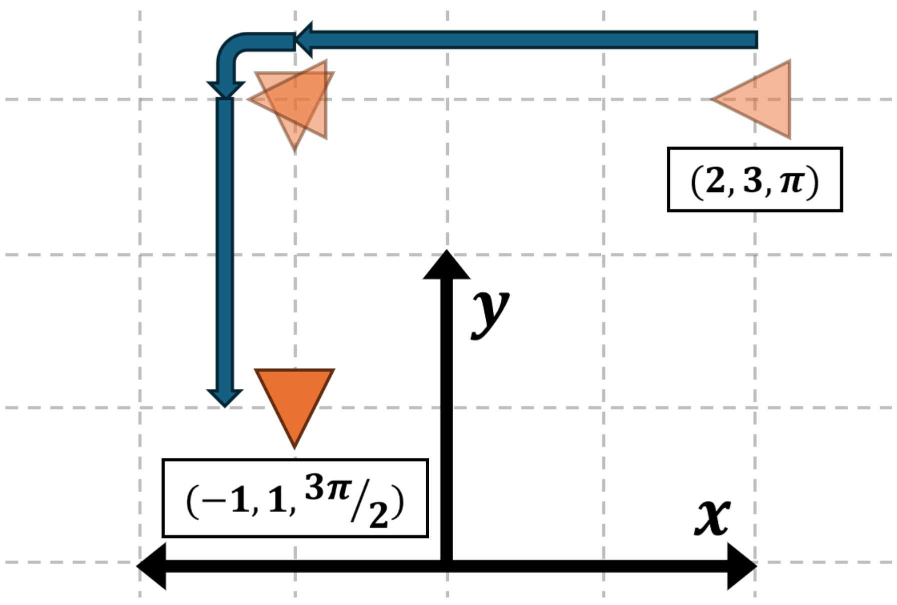


Figure 3.3: Example of robot movement.

3.2 Geometric Hierarchies

Nardari *et al.* [23] developed the methodology of using geometric hierarchies for place recognition in environments filled with identical landmarks. The important concepts are summarized in the following subsections.

3.2.1 Geometric Hierarchy Construction

The expected input to create a geometric hierarchy is a set of noisy 2D landmark locations. A Delaunay triangulation is then constructed from these points. This can be described as the set of triangles where the vertex of any triangle is not in the circumcircle of any other triangle. Triangles have the useful geometric property of being relatively robust to noise. In a Delaunay triangulation, the smallest angle of each triangle is maximized. Because of this, (assuming general position) there is only one valid Delaunay triangulation for a given set of points.

Despite these useful properties, using a Delaunay triangulation alone is not enough to accurately make associations between noisy observations. As the number of triangles grows, there is a greater possibility for similar triangles to appear in the same observation. These triangles must be grouped into regions

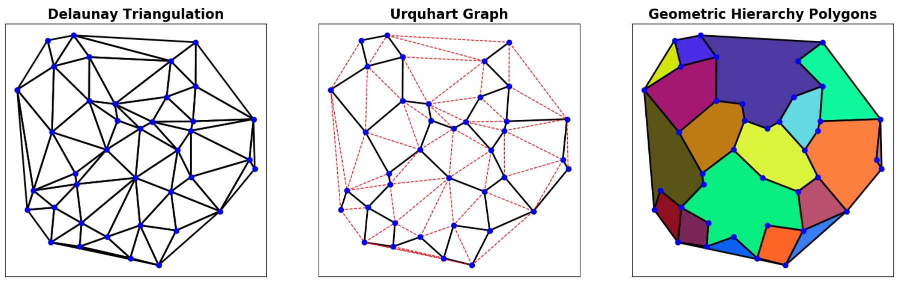


Figure 3.4: Given a Delaunay triangulation, Nardari *et al.* use Algorithm 1 to drop the longest edge for each triangle to produce an Urquhart graph, which also creates many-sided polygons representing the merged triangles across dropped edges [23].

Algorithm 1: Urquhart Graph with Cycle Detection [23]

```
  Data: Delaunay triangulation graph $G_D$, Set of triangles $T$
1 $C = T$            // cycle basis; accumulated polygons
2 $G_U = G_D$                    // Urquhart graph
3 foreach $t \in T$ do
4     Identify longest edge $e \in t$
5     Identify triangle $n$ that shares $e$ with $t$
6     Drop $e$ from $G_U$
7     $C_t = t \Delta n$              // symmetric difference
8 end
9 return $C, G_U$
```

to minimize the chance of finding false-positive correspondences between triangles. This can be executed deterministically by constructing an Urquhart graph from the Delaunay triangulation [31]. The prior work defines Algorithm 1, which summarizes the process of computing the Urquhart graph while also deriving a unique set of polygons, which serve as the top level of the geometric hierarchy (see Fig. 3.4). The asymptotic complexity of constructing a geometric hierarchy is bounded by the computation of the Delaunay triangulation, taking $O(n \log(n))$ time (where n is the number of landmarks).

3.2.2 Polygon Descriptors

Instead of using the Urquhart graph to find correspondences between observations, it is possible to match observations using the polygons derived from

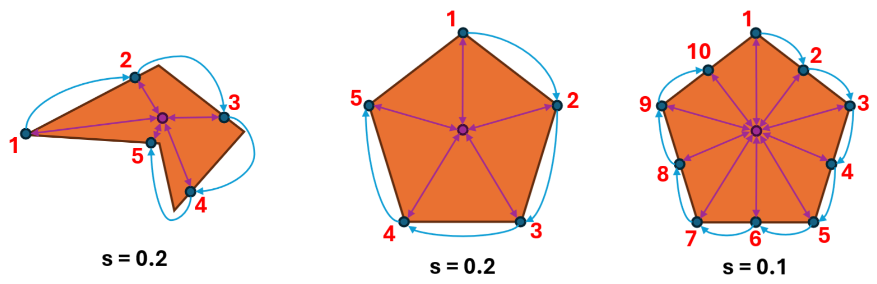


Figure 3.5: Sampling points around the perimeter of a polygon and computing its distance to the polygon's centroid. Sampling begins at a random vertex. The value of s determines how many points will be sampled to create the polygon's initial descriptor.

them. To do so, they must have descriptors that can be used to uniquely identify each other. When computing the descriptor, it is assumed that observations will not be sheared or have differences in scale, which should be true for LiDAR data.

A polygon's descriptor is initialized as the distances between some collection of points on the perimeter of the polygon and the centroid [5]. Because each polygon can have different numbers, a constant number of points are sampled relative to the perimeter of each polygon. The ratio of the perimeter between each point sampled for the descriptor is given by $0 < s < 1$. A large value for s will create lower-resolution descriptors for the polygon, while a small value will likely capture more details such as sharp corners in the polygon. An optimal value for s would balance these properties to maximize descriptor precision while also enabling matching to be robust to noise. This concept is illustrated in Fig. 3.5.

This initial descriptor is invariant to translation within the local observation since it only uses relative distances within the polygon. However, if the order of the descriptor elements is different between polygons (depending on where along the polygon's perimeter sampling begins), it would be possible for identical polygons to not successfully match with each other. This problem is addressed by applying a Discrete Fourier Transform to obtain a new descriptor [5]. Taken from the frequency domain, this new descriptor has the property that its magnitude will be the same regardless its order, making it invariant to where sampling was started.

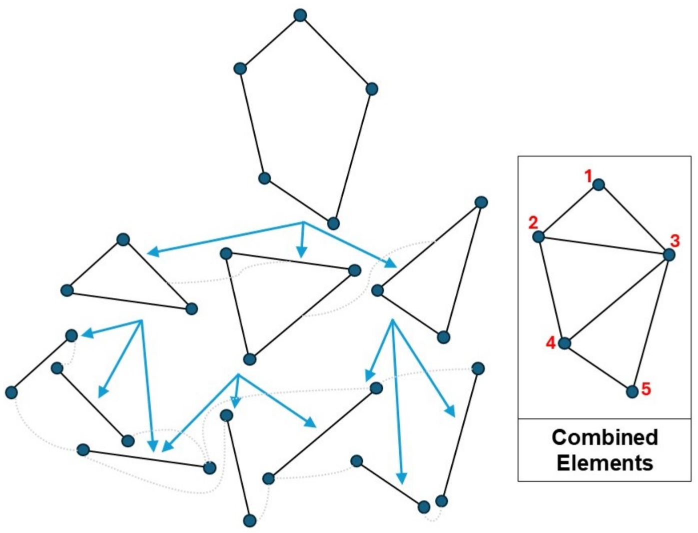


Figure 3.6: An example of the structure of a geometric hierarchy. Polygons are composed of a set of triangles, which are composed of a set of edges, which are defined from a set of vertices.

3.2.3 Matching Observations and Deriving Frame Transforms

The descriptor for each polygon and triangle is stored in the geometric hierarchy. Conceptually, the hierarchy is organized into three tiers: the polygons, the triangles, and the edges of the triangles (see Fig. 3.6). The matching process consists of finding similar elements within the same tier across two observations. To start, the polygon are matched together. To increase robustness and speed, polygons are only compared if the difference in number of points between polygons is less than or equal to Δ_N. If the difference between the magnitudes of their descriptors is less than τ, two polygons would be successfully matched.

For each pair of matching polygons, the set of triangles that composes each polygon are matched with each other. The process for matching two triangles is identical to the process used for polygons. However, successful triangle matches between two polygons are only accepted if the ratio of matched triangles exceeds η. Otherwise, any triangle matches found for the pair of polygons will not be considered valid.

For the final phase of matching, the edges from the remaining corresponding triangles are matched together. This is performed by finding the permutation of edges that minimizes the differences between the lengths of the triangles' edges. Ultimately, these edge correspondences are extended to the triangles' vertices, which represent the positions of individual landmarks in each observation.

Using these resulting landmark associations, a frame transformation is estimated between the two observations. Since these observations are made within the 2D plane and the data does not suffer from shear and scale variations, this transformation takes the form of a three-degree-of-freedom Euclidean (as seen in Section 3.1).

3.3 GraphSLAM Optimization

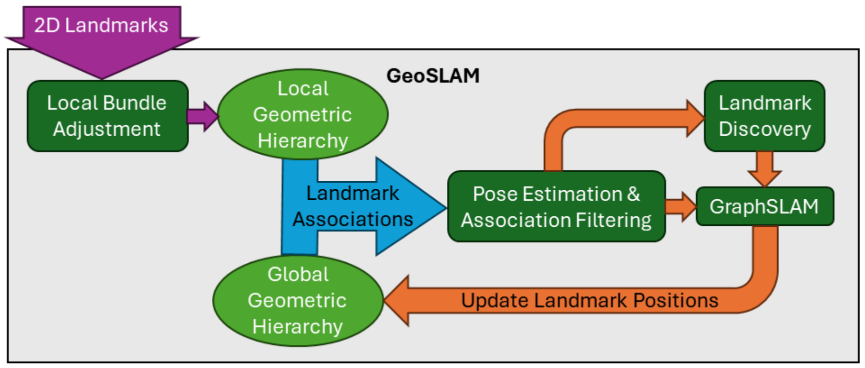
We chose to implement GraphSLAM to optimize the estimated state of the environment. Conceptually, the construction of the GraphSLAM problem is quite intuitive. The global map is defined as a graph. Every node in the graph represents the estimated position of an object in the environment. For example, nodes may represent either the estimated pose of a robot at a particular instance in time or the estimated position of a landmark. Every edge in the graph represents an observed (potentially noisy) measurement. These are usually referred to as "constraints". When a constraint links two robot pose nodes together, it represents an odometry measurement. When a constraint links a robot pose node to a landmark node, it represents the observed position of the landmark relative to the robot's pose. GraphSLAM utilizes every historical sensor measurement to reduce cumulative errors in its map. Therefore, as the robot makes more observations of its environment, more pose nodes constraints will be added to the graph.

Ultimately, GraphSLAM tries to find the position of each node that will minimize the error (or disagreement) between the constraints in the graph. This is achieved by constructing a linear system to compute corrections to the positions of nodes in the graph. This can be performed iteratively to further reduce the graph's error over time.

Chapter 4

System Architecture

The following sections will explore the design decisions and overall architecture of GeoSLAM. The first section explains what type of input this system expects. The next section describes how local bundle adjustment is implemented. The section after that explains how the SLAM problem is solved. The last section reviews GeoSLAM's various parameters and how the system was implemented. Fig. 4.1 depicts how observation data flows through GeoSLAM.



2D Landmarks
GeoSLAM
Local Bundle Adjustment
Local Geometric Hierarchy
Landmark Discovery
Landmark Associations
Pose Estimation & Association Filtering
GraphSLAM
Global Geometric Hierarchy
Update Landmark Positions

Figure 4.1: The structure of GeoSLAM. Raw sensor data is preprocessed into noisy 2D landmark positions for input. GeoSLAM initially aggregates a series of observations to reduce noise. Then, GeoSLAM localizes the robot by matching the geometric hierarchies of the less-noisy local observation and global map. Using the resulting landmark associations, GeoSLAM updates its understanding of the environment.

4.1 Input

GeoSLAM expects its input to be in the form of a noisy 2D point cloud. These points should represent the positions of static, repeatably observable landmarks in the environment with an average nearest-neighbor (NN) distance of at least λ meters. It is expected for some landmarks to be absent, have the wrong position, or not actually represent a real landmark. In other words, perfection is not required from the front-end of GeoSLAM.

Observations should be given to GeoSLAM proportional to the speed of the observer. In other words, GeoSLAM should receive multiple observations (or samples) of the landmarks in its vicinity. A key factor that enables this GeoSLAM to function in the presence of much noise is that it can recognize places even with partial overlap between observations. Since GeoSLAM relies solely on the spatial relationship between landmarks to recognize places, if there is a lot of detection noise in a given environment, GeoSLAM would expect more input frames to counteract the data loss per observation.

4.2 Local Bundle Adjustment

Before we can use our observations to update our global map of the environment, we want to remove as much of the noise from it as possible. Like all SLAM back-ends, our system will break if any incorrect data associations are committed to the graph. Our mapping algorithm is robust enough to filter out these bad associations, but if the local observations do not remotely represent what we have in the global map, then none of our observations will be used to update the global map in the first place. We utilize the place recognition technique developed by Nardari *et al.* to find transformations between observations (Section 4.2.1). We match these frames in a specific order maximize the potential of preserving the data from each observation while minimizing the amount of times we need to match (Section 4.2.2). Our system is going to require as many de-noisied observations as it can to have the most accurate and up-to-date robot position. Algorithm 2 defines the procedure for aggregating the data from sequential observations into a single keyframe.

4.2.1 Differential Pose Estimation

In order to compute an estimated transformation between observations, we build a geometric hierarchy for each input point cloud. Matches are found between hierarchy elements, starting with polygons, then the triangles that

Algorithm 2: Local Bundle Adjustment

```
Data: kfObsv, otherObsv, keyframeCloud, $s, \tau, \Delta_N, \eta, \gamma, \lambda, \sigma, n, f$
/* kfObsv is a set ordered by id, otherObsv is FILO list   */
Function LocalBundleAdjustment($frameID, pointCloud$):
    $notMatched \longleftarrow$ true
    $obsvToMatch \longleftarrow$ (kfObsv $= \emptyset$ ? otherObsv : kfObsv)
    struct {
        id $\longleftarrow frameID$
        cloud $\longleftarrow pointCloud$
        geoHier $\longleftarrow$ constructGeoHier($pointCloud$, $s$)
        tf $\longleftarrow$ NULL
    } $thisObs$

    while ($obs1 \longleftarrow obsvToMatch$.next()) $\neq \emptyset$ and $notMatched$ :
        ptMatches $\longleftarrow$ hierMatching($thisObs$, $obs1$, $\tau, \Delta_N, \eta$)
        Keep ptMatches with pairwise distance $< \lambda$
        if |ptMatches| $> \gamma$ then
            $notMatched \longleftarrow$ false
            if kfObsv $= \emptyset$ then                 // initialize keyframe
                $obs1$.tf = Identity()
                keyframeCloud = $obs1$.cloud
                Move $obs1$ from otherObsv into kfObsv
            $thisObs$.tf = getTf(ptMatches) $\times$ $obs1$.tf
            keyframeCloud += transformCloud($thisObs$)
            kfObsv.add($thisObs$)
            for each $obs2 \in$ otherObsv :            // ignore any repeats
                ptMatches $\longleftarrow$ hierMatching($obs2$, $thisObs$, $\tau, \Delta_N, \eta$)
                Keep ptMatches with pairwise distance $< \lambda$
                if |ptMatches| $> \gamma$ then
                    $obs2$.tf = getTf(ptMatches) $\times$ $thisObs$.tf
                    keyframeCloud += transformCloud($obs2$)
                    Move $obs2$ from otherObsv into kfObsv
    if $notMatched$ then otherObsv.append($thisObs$)
    if otherObsv.first().id $<$ $thisObs$.id $- f$ then otherObsv.pop()
    if |kfObsv| $\geq n$ or kfObsv.last().id $<$ $thisObs$.id $- f$ then
        Publish centroids of clustering(keyframeCloud, $\sigma$)
        Clear kfObsv and keyframeCloud
```

compose the matched polygons, and lastly individual landmarks which define each matched triangle (see full explanation in Section 3.2).

Even though the landmark associations resulting from polygon matching can be quite reliable in the presence of some noise, they are not perfect. There is the potential for false-positive associations to be included in the matching. Left unattended, these mistakes will cause us to calculate the wrong transformation between observations, which will ultimately mess up any attempt we make to combine landmark information across observations.

Given that the underlying data is quite noisy and that each landmark is only identifiable by its position relative to the observer, unless we want to perform a lot of work to validate each match, we need to leverage a couple of assumptions about our input to fix this issue. The first assumption is that the observer should be recording its observations relatively frequently, such that the observer will have only moved a small distance between each reported observation. The other assumption is that the environment's landmarks should be at least λ distance away from each other[1]. Given that the observer's differential distance between sequential observations is likely much less than λ, we can simply check whether a match is valid by ensuring that the positions of the matched landmarks (relative to the observer in their respective frames) are less than λ away from each other. In other words, since the observer should not have physically moved very far since its previous observation, we can quickly identify false-positive associations since their landmarks have significantly differing positions relative to the observer. We can speed up this process even further by evaluating only the maximum feature distance of the matched landmarks (instead of metrics like squared or Euclidean distance). See Fig. 4.2 for an illustration on how landmark associations are rejected.

It is important to remember that we only require two correct associations in order to compute a valid Euclidean transformation between observations (our method expects at least γ associations to minimize the position noise that potentially affects each landmark's position), but a single incorrect association will produce a significantly different matrix. Therefore, we only consider associations to be valid if we are almost certain they are correct. This can be adjusted by varying the value of λ in the system.

[1]If sequential observations are consistently depicting the robot moving a significant distance (due to some unavoidable latency), then another form of filtering these landmark association should be implemented to replace the oversimplified method explained above.

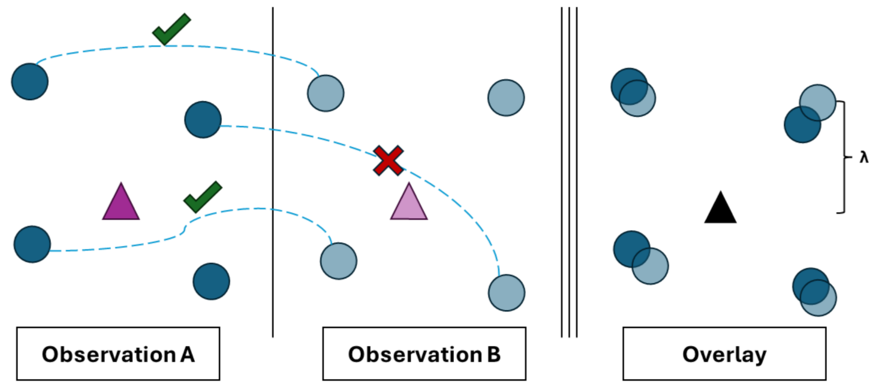


Figure 4.2: Demonstration of landmark association rejection during local bundle adjustment. When the distance between the observed positions of two matched landmarks exceeds λ, that correspondence is assumed to be incorrect and excluded when computing the transformation between observations.

4.2.2 Keyframe Construction

Since we expect the individual incoming observations to be fairly noisy, we want to cross-examine their contents across a small window of time. As new observations are received, each will be matched against each of the observations in the keyframe (starting with the most recent) until a valid association is found. When this occurs, we match it against the f most recent observations that were unable to be associated with the keyframe[2].

If the keyframe is uninitialized, observations will be matched with each other as they arrive until a valid association between two of them is found. The earliest observation will be used as the base frame for the keyframe. When other observations are matched with the keyframe, the positions of their landmarks will be transformed w.r.t. the base of the keyframe. These transformed positions get stored in a single point cloud. This process is similar to computing the forward kinematics of robotics arms, where the position of a robot's end-effector relative to the arm's base can be derived from the combination of the individual transforms which describe the state of each joint and link in the arm. In other words, regardless of which keyframe observation a new observation matches with, it is really easy to find the positions of the new observation's landmarks w.r.t. the base of the keyframe. See Fig. 4.3 for

[2] f should be relatively small. While our method for place recognition can still function when observations only exhibit a partial overlap, if observations are taken from a distance greater than λ, their landmark correspondences willbe rejected.

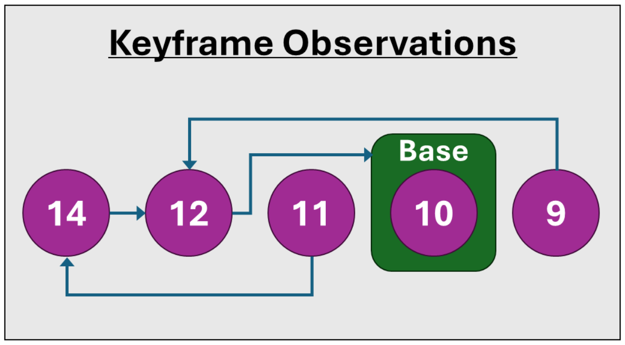


Figure 4.3: Case study on keyframe construction. Observations are sorted by the order of their arrival. The fixed reference frame of the keyframe is the robot's position in the observation used as the base frame. Every other observation in the keyframe stores a transformation to describe its landmark positions relative to the base frame. In this example, before the keyframe was initialized, observations {9, 10, 11} were not able to align with each other. Once observation 12 was created, the following sequence of events ensues to construct the keyframe shown above: **(1)** 12 matched with 10 **(2)** 10 established as keyframe base, **(3)** 12 linked to 10, **(4)** 9 matched and linked to 12 **(5)** 13 could not match with keyframe observations **(6)** 14 matched and linked to 12 **(7)** 11 matched and linked to 14.

a closer look at how observations can be arranged in the keyframe.

Ideally, the accumulation of our observation data should mask the presence of any detection noise. Once we have at least n observations-worth of data in our cumulative point cloud, we perform clustering on the aggregated landmark positions, keeping the centroids of the resulting clusters as our finalized landmark locations. Any form of clustering can be used for this purpose, but ultimately each cluster will represent a single point in space, and we want to try to ignore as many outlier landmark positions as possible when clustering. By clustering the landmark positions, we are attempting to remove as much of the position noise as possible. With these final landmark positions defined, we send them to the global map and start fresh with an empty keyframe.

4.3 Global Mapping

GeoSLAM is designed to continuously provide up-to-date robot pose and landmark position estimates in the global frame. We combine GraphSLAM, which incrementally improves our estimates regarding the state of the environment, with the place recognition technique of geometric hierarchy matching. Since we expect keyframe data to be a little noisy, it can still be challenging to make associations between the local and and global frames if both representations of the environment do not closely resemble the ground truth. However, if we can incrementally improve the global map over time, then it we could also improve our ability to find associations with the global map regardless of how noisy an incoming keyframe is.

As with any SLAM back-end, GraphSLAM requires essentially perfect data association to work properly. Since we expect some residual noise to remain in our data, this implementation utilizes a number of safeguards to prevent updating the map in a manner that would break the optimization process. Algorithm 3 summarizes the process of maintaining the global map using 2D landmark locations from incoming keyframes.

4.3.1 Map Initialization

Choosing the starting configuration of the map is important. There are currently two options regarding how the global map can be initialized. A user can either provide a map upon starting the program or the system will be expected to initialize one itself. A user-defined map is equivalent to a set of 2D landmark positions. Landmarks initialized in user-defined maps will not be included in graph optimization until they have been observed by the robot at least once. This will improve the speed of graph optimization especially when large maps are given to GeoSLAM.

When a map is not provided, the system will use the first keyframe it receives as its starting map. Since individual keyframes may have extra or missing landmarks relative to the robot's current position, this can be improved to collect multiple keyframes before establishing the starting global map. For example, we could probably store a small window of keyframes to be used find transformations between observations (similar to how local bundle adjustment is implemented). Then we could leverage our method for landmark discovery (see Section 4.3.4) to determine where landmarks belong in the global map. Once the set of global landmarks is considered to be dense enough for proper geometric hierarchy matching, we can forget the previous keyframes and only match new keyframes using the global map.

Algorithm 3: Global Map Update

```
Data: globalMap, newLandmarkGraph,
      $s, \tau, \Delta_N, \eta, \gamma, \psi, m, \alpha, \xi, \Lambda, \rho, \beta, r, \mu, k, \omega, \delta$
Result: up-to-date robot pose and landmark positions
/* globalMap contains (ldmks, globalGH, graph)            */
/* DiscoverLandmarks() can access newLandmarkGraph        */
Function GlobalMapUpdate($cloud$):
    $gH \leftarrow$ constructGeoHier($cloud$, $s$)
    if !exists(globalMap) then create globalMap using $cloud$ and $gH$
    else
        for $t \leftarrow \tau$ to $m$ by $\psi$ until $|ptMatches| > \gamma$ or $\tau > m$ do
            $ptMatches \leftarrow$ hierMatching($gH$, globalGH, $t, \Delta_N, \eta$)
        if $|ptMatches| \geq \gamma$
              and
            EstimatePose($cloud$, ldmks, $ptMatches, \alpha, \xi, \Lambda, \rho, \beta, r, \mu$)
         then
            Retrieve $landmarkAssoc, unmatchedLocals, bestTf$
            graph.addPose($bestTf$)
            for each $(lP, gP) \in landmarkAssoc$ do
                if $gP \notin$ ldmks then graph.addLandmark($gP$)
                graph.addConstraint($bestTf$, $gP$, $lP$)
            for each $(gP, obsv) \in$
             DiscoverLandmarks($unmatchedLocals, bestTf, k, \delta$) do
                graph.addLandmark($gP$)
                for each $(pose, position) \in obsv$ do
                    graph.addConstraint($pose$, $gP$, $position$)
            Optimize and update graph
            if graph error increased too much then rollback globalMap
            else
                if any new landmarks discovered then
                    globalGH $\leftarrow$ constructGeoHier(ldmks, $s$)
                else recalcPolygonDescriptors(globalGH)
                newLandmarkGraph drops observations older than $\omega$
```

4.3.2 Initial Landmark Association

As opposed to how geometric hierarchies are matched during local bundle adjustment, we attempt to match polygons using multiple threshold settings. The polygon descriptor threshold τ for local bundle adjustment will be set relatively high in order to find landmark associations between polygons that potentially are not very similar due to high amounts of noise. On the other hand, the global mapper's value for τ will likely be much lower because we expect the polygons to contain less noise and we do not want false-positive matches. However, finding matches only using this initial threshold is not guaranteed, so we repeatedly check for matches, incrementing τ by ψ until reaching some upper limit m. Since keyframes take multiple observations to produce, GeoSLAM will inherently have fewer keyframes to match against to update and optimize the robot's position. Therefore, we aim to utilize the data from every possible keyframe when updating the global map. The only reason a keyframe should be ignored is when it contains information that could potentially disturb the stability of the map.

4.3.3 Global Pose Estimation & Data Association Filtering

As mentioned in Section 4.2.1, the landmark associations that result from matching geometric hierarchies are not perfect. Therefore, we need to validate that the given associations are correct before we can use them to calculate a pose estimate for our robot. Incidentally, we have the opportunity to create even more landmark associations while finding the best possible pose estimate for the robot. Algorithm 4 describes this process.

We will repeatedly use samples of ξ-many landmark associations to obtain a variety of global pose estimates for the robot. Using each pose estimate, we can transform the landmarks in the keyframe into the global map. If a transformed point is closer than Λ to the location of a global landmark, then those points most likely represent the same landmark. If the transformed point is farther away than ρ to any global landmarks, then it most likely represents a landmark not yet depicted in the global map. To improve scalability, if the maximum observation range of the robot (r) is known, we can reduce the number of global landmarks that the keyframe points will be compared against.

Ultimately, we will keep the estimated robot pose that resulted in the highest number of landmark associations derived after transforming the keyframe into that pose. We will store those newly derived landmark associations as well as the candidate points to represent new global landmarks.

Algorithm 4: Pose Estimation and Landmark Association Filtering

```
   Data: localLandmarks, globalLandmarks, pointMatches
   Result: landmarkAssocations, unmatchedLocals, bestTf,
           hasEstimate
1  Function EstimatePose($\alpha, \xi, \Lambda, \rho, \beta, r, \mu$):
2      hasEstimate $\longleftarrow$ false
3      repeat
4          $goodMatchesMap, problematicGlobals, matchless \longleftarrow \emptyset$
5          $tf \longleftarrow$ getTf(random subset (size $\xi$) of pointMatches)
6          $nearbyGlobals \longleftarrow$ {globalLandmarks within range $r$ of $tf$}
7          foreach $gP \longleftarrow (idx, position \in (tf \times$ localLandmarks$))$ do
8              Find $closestDist$ from $gP$ to some $gL \in nearbyGlobals$
9              if $closestDist < \Lambda$ then
                   /* landmark observable once per frame        */
10                 if $gL \in goodMatchesMap$ then
11                     $problematicGlobals$.add($gL$)
12                     $goodMatchesMap$.remove($gL$)
13                 else if $gL \notin problematicGlobals$ then
14                     $goodMatchesMap[gL] \longleftarrow gP$
15                 end
16             else if $\rho <= closestDist$ then $matchless$.add($gP$)
17         end
18         if $|goodMatchesMap| >$ |landmarkAssocations| $\geq \mu$ then
19             landmarkAssocations $\longleftarrow goodMatchesMap$
20             bestTf $\longleftarrow tf$
21             unmatchedLocals $\longleftarrow matchless$
22             hasEstimate $\longleftarrow$ true
23         end
24     until $\alpha$ iterations or $\beta \cdot$ |localLandmarks| $\leq$ |landmarkAssocations|
25 end
```

While it could be argued that we only need the landmark associations resulting from matching geometric hierarchies, an important reason to extend landmark associations beyond those matches is to promote the representation of a diverse selection of observed landmarks per pose estimate. In general, each geometric hierarchy consists of an assortment of generic and unique polygons. Every polygon in the geometric hierarchy is eligible for matching, which is especially important when the geometric hierarchy formed from an observation does not contain many geometrically distinct polygons. It is important to improve the positional accuracy of every landmark in the global map so that we can localize the robot as consistently and as accurately as possible. If the positions of only a certain subset of landmarks are optimized across subsequent observations, then when those optimized landmarks are not visible at certain positions in the environment, there would be a higher chance to make incorrect associations or estimate incorrect poses with incoming noisy keyframes.

Conversely, it is not necessary for every landmark in each keyframe to be matched with its corresponding landmark in the global map. In fact, it may be harmful to try to do so, since perfect landmark association is difficult to obtain using only positional information to distinguish individual landmarks from each other. Across consecutive keyframes, there should be some amount of overlap in their observed landmarks. Even if an association is not found for a landmark in one keyframe, there will be more opportunities to improve its estimated position soon afterwards.

When not enough ”reliable” landmark associations (μ) are found using a certain pose estimate, that pose estimate was most likely incorrect. In other words, not enough of our prior knowledge about the robot's immediate proximity translates to the representation of the environment that we currently see. It is possible that none of the estimated poses will provide enough associations to update the global map confidently. Therefore, when this occurs, the keyframe will be discarded. We can assume that the landmark associations resulting from strictest geometric hierarchy matching settings included enough incorrect associations to significantly affect pose estimation, thus trying to perform matching again is not worth the time or effort trying to fix.

4.3.4 New Landmark Discovery

Once we are highly confident with our correspondences between landmarks in the keyframe and the global map, we must also determine whether new landmarks should be added to the map. It is important to remember that the keyframes are still somewhat noisy, and their landmark positions will probably not be perfect. If we were to accidentally introduce a false landmark into the global map, we would disrupt the state of the global geometric hierarchy, making it much more difficult to accurately find matches between the keyframe and the global map. Therefore, we implement a procedure that will prevent mistakes like this from occurring.

Algorithm 5: New Landmark Discovery

```
Data: newLandmarkGraph
Result: global landmarks and their observed local positions
1  Function
    DiscoverLandmarks($unmatchedLocals, currentPose, k, \delta$):
2      foreach $node \longleftarrow (localPt, globalPt) \in unmatchedLocals$ do
3          newLandmarkGraph.add($currentPose, node$)
4          Create edges to existing nodes with global positions closer
             than $\delta$
           /* Assume $\delta <$ (avg distance between NN landmarks)  */
5          if $k$-clique(newLandmarkGraph, $node$) then
6              Include the average global position and the local positions
                 of the k-clique elements in the result
7              Remove k-clique elements from newLandmarkGraph
8          end
9      end
10 end
```

Our strategy, like seen in the upstream methods, is to only add a landmark to the global map once we have found enough keyframes within a short period of time that agree on its approximate position. Algorithm 5 describes how this process works. We maintain a sliding window of observed landmark locations across keyframes, and we use them as the nodes for a graph (or network) data structure. The edges of this graph represent that two landmarks are within distance δ of each other[3]. A "consensus" between keyframes can be defined as a k-clique in the graph. k is pre-defined by the user, representing the minimum number of observations required to discover a landmark. We can discover k-cliques while individual observed landmark positions are added to the graph, making this operation very fast. Landmark positions that have been in the network for more than ω keyframes are removed after the global geometric hierarchy is updated (Section 4.3.5).

[3]See Section 4.4.3 for more details regarding the value of δ.

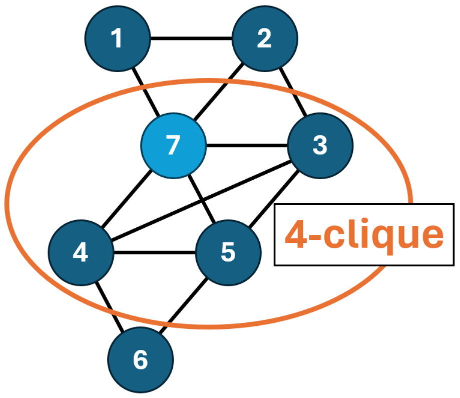


Figure 4.4: A clique is a subgraph where each vertex is connected to every other vertex. In this example, if GeoSLAM was looking for a 4-clique to find a consensus in the association network, it would find one after landmark 7 is added.

4.3.5 GraphSLAM and Geometric Hierarchy Maintenance

This system uses the online version of GraphSLAM to improve the estimated position of landmarks and the pose of the robot over time. In this version, the only one optimization is computed after the data from a given observation is added to the graph. In particular, the graph directly retrieves and updates the positions of landmarks directly from the global geometric hierarchy.

After the graph is successfully optimized, the updated positions of landmarks are saved in the geometric hierarchy. Since these positions are stored in a single location, updating these values is very straightforward. However, at this point, it is possible for new landmarks to have been added to the map. Additionally, because the landmarks have changed position, it is possible for the structure of the geometric hierarchy to have changed. In the best-case scenario, no new landmarks have been added, the triangulation still satisfies the Delaunay constraint, and none of the triangles changed which of their edges is longest (causing the arrangement of polygons to change). In the worst-case scenario, the triangulation would contain many non-Delaunay triangles and the arrangement of polygons would have changed.

For this problem, we assume that even if the changes in landmark positions caused a major change in the Delaunay triangulation or polygon arrangement, there would still be some amount of partial overlap in the robot's observations that would enable matches between keyframes and the global map. Therefore, we elect to not recompute the structure of the geometric hierarchy when new landmarks have not been added. Considering the current state of the system, this decision is made to reduce computational requirements and ensure the system can operate practically in real-time. However, we also want to ensure that polygons from incoming keyframes will continue match with the global map over time. Therefore, we compromise by choosing to always update the edge lengths of the triangulation and the descriptors of the polygons.

When landmarks are added to the global map, it is guaranteed some portion of the Delaunay triangulation. In these cases, we find that it is necessary to reconstruct the geometric hierarchy, as the new landmarks should improve future polygon matching capability. Specifically, this should only occur when the robot is exploring the frontiers its map. If the robot is operating within a pre-mapped area, the computational cost to reconstruct the geometric hierarchy should never occur.

4.4 Parameter Tuning

Throughout Section 4, there are many references to the various parameters that must be set correctly so that GeoSLAM will work as intended. The following section compiles those parameters into Tables 4.1, 4.2, and 4.3, and explains the rationality behind choosing the value for each parameter. The values for certain parameters are more likely to vary depending on the implementation: **bold** parameters are sensitive to detection noise, *italicized* parameters are sensitive to position noise, and <u>underlined</u> parameters are sensitive to the environment's landmark density.

4.4.1 Setting Parameters for Polygon Matching

Regarding the settings for the polygon matching parameters in Table 4.1, we elect to use the same values as the prior work for s, Δ_N, and η (5%, 3, 50%). These values are unlikely to change due to differences in sensor noise and the environment's landmark density. The threshold γ was not defined in the prior work, and is primarily designed to be an extra counteract uncertainty regarding landmark associations.

In prior work, optimal value for τ was determined to be 5 for the data they tested against. However, that setting was desirable for the general use-case of map-merging and loop-closure detection, whereas for this system, we assume that there should always be a match between geometric hierarchies since we can almost guarantee partial overlap between observations. There are also two different use-cases for τ within GeoSLAM. For local bundle adjustment, we may want to set τ to a high value, such as 9, because we might expect the

Table 4.1: Polygon Matching Parameters

s	Ratio of Perimeter between Sampled Points for Descriptor
τ	***Maximum Descriptor Distance for Valid Polygon Match***
Δ_N	Differing Number of Sides for Polygons Matching Eligibility
η	Minimum Ratio of Matched Triangles in a Polygon
γ	Minimum Landmark Associations for Valid Frame Association

Table 4.2: Local Bundle Adjustment Parameters

λ	*Maximum Distance between Landmarks for Valid Association*
σ	*Cluster Tolerance (radius)*
n	**Standard Number of Observations per Keyframe**
f	Number of Unmatched Observations before Flushing Keyframe

incoming frames to be very noisy, and we want to quickly find any matches we can. In the global mapper, we may want to set τ to be much lower (for example 2), because we want to find a small set of very reliable landmark associations if possible (to get the best samples for our pose estimate), and if that low threshold does not work for a given keyframe, then the system will automatically increase the threshold to try to find valid matches anyway. Ultimately, the user should consider how noisy they expect the incoming data to be when setting τ.

4.4.2 Setting Parameters for Local Bundle Adjustment

Table 4.2 lists the parameters utilized specifically for local bundle adjustment. λ (measured in meters) should be set relative to the speed of the robot and the rate of incoming observations, such that the robot moves a distance much smaller than λ between each observation. λ should also be smaller than what the average nearest-neighbor distance between landmarks in the environment is to reduce the likelihood of the system using incorrect landmark associations resulting from geometric hierarchy matching to compute the differential transformation between two observations (for more information regarding λ, refer to Section 4.2.1).

n should be set depending on the amount of expected sensor noise, especially detection noise. As demonstrated in prior work [23], detection noise had a much greater effect on loop-closure detection via geometric hierarchy matching than position noise. This is because the presence of landmarks (or lack thereof) significantly changes the construction of the geometric hierar-

Table 4.3: Global Mapping Parameters

ψ	Increment Amount for τ
m	Maximum Value for τ
α	Maximum Pose Estimate Attempts
ξ	Landmark Association Sample Size for Estimating Pose
β	Ratio of Valid Landmark Associations to Exit Estimation Early
Λ	*Maximum NN Landmark Distance for Valid Filtered Association*
ρ	Minimum NN Landmark Distance for Landmark Discovery Election
r	Maximum Sensor Range
μ	Minimum Filtered Landmark Associations for Valid Pose Estimate
k	*Minimum Agreeing Observations to Establish New Landmark*
ω	*Number of Observations in Active Window*
δ	Maximum Feature Distance for Landmark Discovery Association

chy. With more noise, some frames may not match, and more frames may be required in order to collect enough information about the local landmarks to construct an accurate and useful keyframe.

σ may or may not be present depending on the type of clustering used. Its usage in this implementation is discussed in Section 4.5.5. Lastly, f should be set lower if runtime efficiency is important, as fewer observations that have yet to be added to the current keyframe will be kept in memory to be matched against.

4.4.3 Setting Parameters for Global Mapping

Table 4.3 lists the user-defined parameters for the global mapping component. ψ and m are used to define how many matching attempts should be made between incoming keyframes and the global map. α, ξ, β, Λ, ρ, r, and μ all pertain to estimating the robot's global pose and finding landmark associations with high confidence. k, ω, and δ dictate the behavior of how new landmarks are established in the global map.

α, ξ, and β resemble traditional RANSAC parameters. Our implementation has α set relatively low because the estimation procedure we implemented is not optimized very well. ξ is set very small to reduce the potential for a bad data association from messing up the pose estimate while also forcing sample variety to be high even though α is so low. β is a setting that allows for early estimation exits in cases with low environment noise.

Λ and ρ are thresholds for discovering filtered landmark associations and potential new landmarks respectively. The value for Λ should be very small (at most 20% of λ), because the number of landmark associations that pass this threshold will determine the best pose estimate to provide to GraphSLAM. Similarly, when more landmark associations pass under this tight threshold, we can be more certain that our pose estimate is correct. The value for ρ should be lowered as the average nearest-neighbor distance between the environment's landmarks decreases.

r should be equivalent to the sensor range of the robot this system is supporting. It is specifically designed to improve scalability of the landmark association filtering process.

μ is useful for identifying keyframes that will likely provide a bad pose estimate and get rejected after graph optimization is attempted. In other words, its more likely for the pose estimate to be wrong when we are only able to find a small number of landmark associations from it. This also counteracts the effects of having the value of ξ being so low, where the only remaining landmark associations may be the original samples. Set it high if ξ is low or you are concerned that the mapper is rejecting bad graph optimizations, but it could be instead exiting early and saving time/computations.

The remaining parameters k, ω, and δ pertain to the landmark discovery procedure. Like the other association distance thresholds, δ should be smaller than the average distance between NN landmarks in the environment, in this case to prevent multiple landmarks collapsing into a single point during discovery. k and ω are less trivial to set. When setting k, we want to balance (the need to establish landmarks in the map quickly to improve our ability to match keyframe geometric hierarchies with the global map) with (our desire to prevent fake landmarks to the global map, which would disrupt geometric hierarchy matching). In general, k will likely need to be increased depending on the amount of residual noise we are expecting in the incoming keyframes. ω must always be larger than k, and it determines how consistent consecutive frames must be when coming to a consensus regarding the location of new landmarks. A bigger ω value will allow more opportunities for landmarks to be discovered, while a value that is slightly bigger than k will make sure that landmarks are only discovered when the sequence of frames agree on its position.

4.5 Practical Implementation Details

Although briefly mentioned in previous subsections, the following will provide details regarding how the proposed system was physically coded.

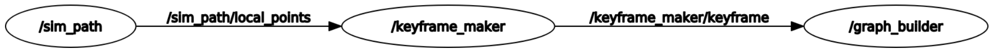


Figure 4.5: GeoSLAM ROS Node Architecture.

4.5.1 Overall System Architecture

In order to test the efficiency and effectiveness of GeoSLAM, we created a full C++17 implementation using ROS and associated libraries. The algorithms found in this project were developed using Nardari *et al.*'s open-source geometric hierarchy code [23] (discussed in Section 4.5.4), guidelines for implementing GraphSLAM [11] (discussed in Section 4.5.6), or were otherwise conceived independently[4]. GeoSLAM utilizes a newer version of Eigen (3.4+) to implement efficient data storage and manipulation, and therefore requires the use of PCL 1.14.0 instead of the PCL version that ROS provides to successfully compile the code.

Fig. 4.5 illustrates the ROS network architecture of GeoSLAM. The first node simulates a robot travelling through a given forest environment. Each noisy observation is stored as a 2D point cloud and published. The next node subscribes to these observations and performs local bundle adjustment with them, publishing the resulting keyframes as 2D point clouds. The final node uses each keyframe to update a global map. In addition to passing information over ROS Topics, each of these nodes logs important data. This information is processed offline using separate Python scripts.

4.5.2 Simulated Forest Generation

The method for generating a simulated forest environment is the same as was used by Nardari *et al.* [23] Forests are initialized by generating a set of 2D landmarks using Poisson-Disc sampling through Bridson's algorithm [1] (see Fig. 4.6). The minimum distance between points is varied depending on the desired forest density.

This initial configuration will have trees arranged in a regular pattern, which does not realistically represent the distribution of trees in a real forest. Thus, the position of each tree is perturbed with zero-mean Gaussian noise. Table 4.4 depicts the parameters used to create each type of simulated forest.

[4]https://github.com/danielcastellarin/urquhart

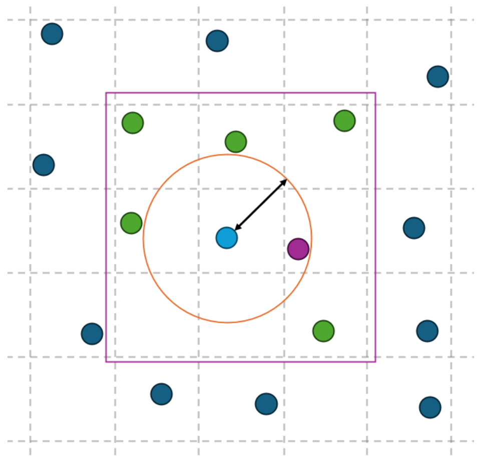


Figure 4.6: Illustration of Bridson's algorithm. All points are stored in a grid, where each tile is assigned (at most) one point. When sampling a new point, the program first checks if there is a point assigned to the tile where it will be placed. If that tile is available, the program will evaluate the distances to the points assigned to the surrounding eight tiles to ensure that no point is within the user-defined minimum distance between points. In the above example, one of the points in an adjacent tile violates the minimum distance constraint, meaning that the sampled point would not be kept in the grid.

Minimum Distance	Standard Deviation	Forest Type	Landmark Count	Average Distance between Landmarks
4	2	Dense	42011	2.733m
7	3	Intermediate	13770	4.968m
9	3.5	Sparse	8419	6.560m

Table 4.4: Parameters used to generate each type of simulated forest. Each forest is a 2D point cloud representing 1km^2 of trees.

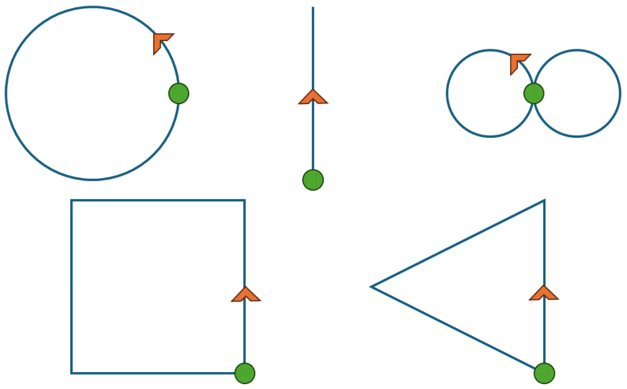


Figure 4.7: Five path options for the simulated robot. The robot starts at the green circle and follows the path in the direction indicated by the orange arrow.

4.5.3 Robot Observation Simulation

For all experiments, we simulated a robot observer travelling a distance of 200m. Fig. 4.7 illustrates the path variations that could be simulated for the robot. We randomly sample starting poses in the environment until we find one where no landmark will be within 30cm of the robot's chosen path. We assume that the observer has 360° vision of its surroundings. That being said, GeoSLAM should function identically no matter where the landmarks are relative to the observer.

The robot records the position of the landmarks within 40m of itself at set intervals along its given path. When travelling straight or along a curve, this interval is defined as a distance of 20cm (by default) around the shape's perimeter. When the robot reaches a corner, it will rotate 0.2 radians per interval until it aligns with the next segment or it will stop if it has already travelled the full 200m. When transitioning between forward and turning maneuvers, the robot will always exhaust its full movement capability. For example, if the robot is 15cm from a corner, it will arrive at the corner and then turn 0.5 radians before recording its next observation. Therefore, using the default interval, for the circle, line, and figure-eight paths, the robot makes 1000 observations along its path, whereas it makes 1022 and 1020 observations for the square and triangle paths respectively.

Before performing local bundle adjustment using the observed landmark positions, the simulation applies detection and position noise to each landmark. Detection noise describes the robot's probability of detecting any given landmark in its sensor range. If detected, we then apply noise to the landmark's sensed position using a Gaussian distribution with a given standard deviation. Unless specified otherwise, our simulations use 90% and 20cm for detection and position noise respectively.

4.5.4 Geometric Hierarchy

The basic structure of code is maintained from the open-source version[5] provided by Nardari *et al.* [23], which was translated into C++ after the original work (developed with Python) was published. In terms of third-party libraries, they utilized QHull to compute the Delaunay triangulation for a given set of points and OpenCV to perform Discrete Fourier Transforms to compute the descriptor for each polygon.

We implemented three major changes to this version. The first was that we changed how polygons store their vertex positions and edge lengths. Instead of each polygon storing separate copies of the positions of vertices and the lengths of edges, for the entire hierarchy, we store the values in a single location, and we make the polygons store references to those values. This change enables the state of the geometric hierarchy to be updated quickly and easily, which is critical when optimizing the global map.

The second major change we implemented was that we refactored most of the standard library containers into Vector and Matrix objects from the Eigen library. This was intended to improve the overall speed and scalability of the code. For example, vectors in the C++ standard library are designed to be easy-to-use dynamically-allocated lists. On the other hand, Eigen's dense vectors and matrices store their data contiguously in memory are optimized to perform math on its elements very quickly.

The last major change was that we rewrote the tree data structure which linked the polygons of the hierarchy together. The original implementation utilized adjacency lists from Boost's graph library. However, that implementation was traversing the tree inefficiently (e.g. using breadth-first-search for everything), which we noticed was causing performance issues at scale. Since this tree only has a maximum depth of three, running breadth-first-search does not actually improve runtime. Instead, we implemented the tree as three dictionaries: polygon lookup, parent lookup, and children lookup. Utilizing these maps, our version of the geometric hierarchy ran much faster at scale than the original.

[5]https://github.com/gnardari/urquhart

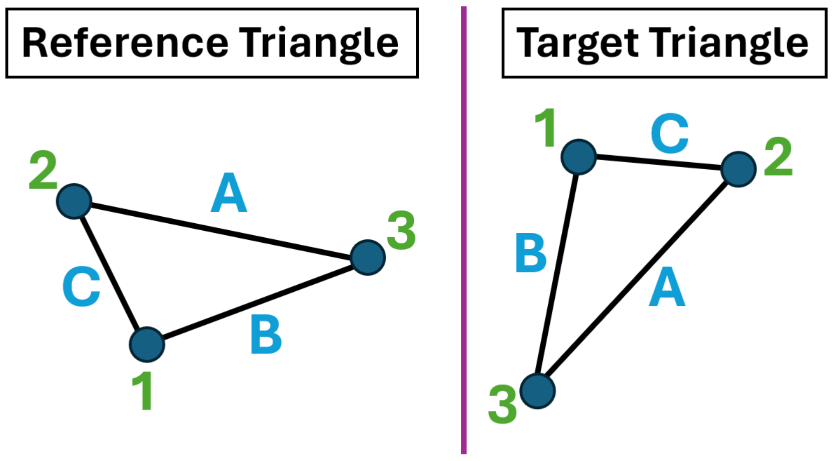


Figure 4.8: An example of how the current implementation matches vertices of triangles. After the edges A,B,C are matched between two triangles, the program will extend those matches to the vertices opposite to each edge. For instance, when the ”A” edges match, the ”1” vertices match, too.

There were also a couple of minor changes we implemented to improve the accuracy of landmark association. The first was we use a non-greedy approach for polygon matching. This is especially useful for matching against the global map, where it is much more important to find the best possible polygon matches for the purpose of landmark association. we also fixed how the vertices of triangles are matched based on edge lengths. Previously, after finding the best permutation of edges for the participating triangles, the code would match the starting vertex of the matched edges. This method will only work if the vertices of both triangles both ordered either clockwise or counter-clockwise. However, this is not guaranteed for the triangles computed by QHull, so sometimes the landmark associations would be wrong even though the edge associations were correct. Instead of matching the starting vertex of the matched edges, we match the vertices opposite to the matched edges, officially making this process invariant to the participating triangles' order of vertices. See Fig. 4.8 for a closer look at an example.

4.5.5 Local Bundle Adjustment

The major portions of the local bundle adjustment functionality is implemented using PCL functions. Specifically, this includes transforming entire point clouds into new frames and clustering the cumulative keyframe cloud.

In regards to how the point clouds are transformed into new frames, PCL does not provide a built-in function to do this for 2D point clouds. Therefore, we store these point clouds in 3D (z always set to 0), derive a 3D transformation matrix, and use the transformation function for 3D point clouds to perform the operation we need.

In regards to how the cumulative keyframe cloud is clustered, we use a greedy method defined within the PCL library (EuclideanClusterExtraction). A caveat to this approach is that clustering accuracy heavily relies on setting the cluster tolerance σ correctly. Too low could cause clusters to either not form or a single landmark represented as multiple landmarks. Too high and multiple landmarks could collapse into a single point. As long as these edge cases happen infrequently/rarely (1 in k keyframes), the global map should remain stable. We have also set the minimum and maximum cluster size to 2 and n respectively. The lower bound is 2 to ensure that single outliers are not included in the keyframe. The upper bound is n to ensure that multiple landmarks will not collapse into a single point in the keyframe when they happen to be clustered together. This has the side effect of potentially not representing those the landmarks in the keyframe at all, which could be undesirable. However, due to the robustness of the system, this flaw does not drastically impact performance.

4.5.6 GraphSLAM

We implemented GraphSLAM from scratch following the guidelines from an introductory GraphSLAM article [11] and a Python implementation which demonstrates the concepts in code[6].

Compared to the prior work, we implemented three modifications to the version of GraphSLAM. The first was that we configured map optimization to run online. Instead of ingesting the observations across all robot poses and running multiple optimizations with GraphSLAM (solving the Full-SLAM problem), our implementation reads one set of observations at a time and performs an optimization with that data added to the map state. This results in GraphSLAM performing quicker computations, which produce smaller (less computationally expensive) changes in the state of the map, enabling the process to run practically in real-time as the robot is actively observing the environment.

Another major change we made is that the positions of all global landmarks are physically stored and updated in the geometric hierarchy. As mentioned

[6]https://github.com/StefanoFerraro/Graph-SLAM

previously in Section 4.5.4, we modified the geometric hierarchy implementation to have polygons store references to their vertex positions and edge lengths, such that there will be one copy of each vertex position and edge length. Having these landmark positions stored in one location makes it much easier to maintain data consistency after the graph is optimized.

The last modification we implemented in the system was the additional ability for the graph to detect poor data associations and rollback the state of the graph after attempting a graph optimization using those bad associations. This is often featured in practical applications of GraphSLAM, but usually not discussed in literature for the underlying theory. It is possible to detect mistaken associations by computing the error between the newly optimized robot pose and landmark positions and the landmark positions relative to the robot given by the latest keyframe. When this error is significantly higher than the errors calculated after previous poses were optimized, it is safe to assume that GraphSLAM's system of equations was built incorrectly because it included some bad landmark associations.

After a false data association is detected, we elect to rollback the state of the graph, geometric hierarchy, and landmark discovery network to the state of the map before the latest keyframe was received. Similar to how the latest keyframe is dropped when the system cannot find enough landmark associations to validate any estimated robot poses, attempting to isolate the mistaken association or correct the estimated pose would not be worth the effort.

While GraphSLAM can suffer from computational complexity issues at scale, we use it because GraphSLAM maintains global consistency in the map and does not make assumptions about the distribution and observed positions of landmarks in the environment. Even though we used GraphSLAM in our implementation, any formulation of online-SLAM can be used to optimize this system's global map.

4.5.7 Visualizations

We utilized many visualizations when designing the components of GeoSLAM. All of them were built using Matplotlib plots in Python. They could either read data from files or receive data in real-time using ROS messages. Matplotlib is optimized for simple, static plotting, and is not considered thread-safe. However, we still used it alongside ROS to rapidly refresh visualizations in real-time. Although ROS has built-in, scalable visualization support through the Rviz package, we elected not to use it because utilizing Rviz would require much more development time set up and update as GeoSLAM evolved over

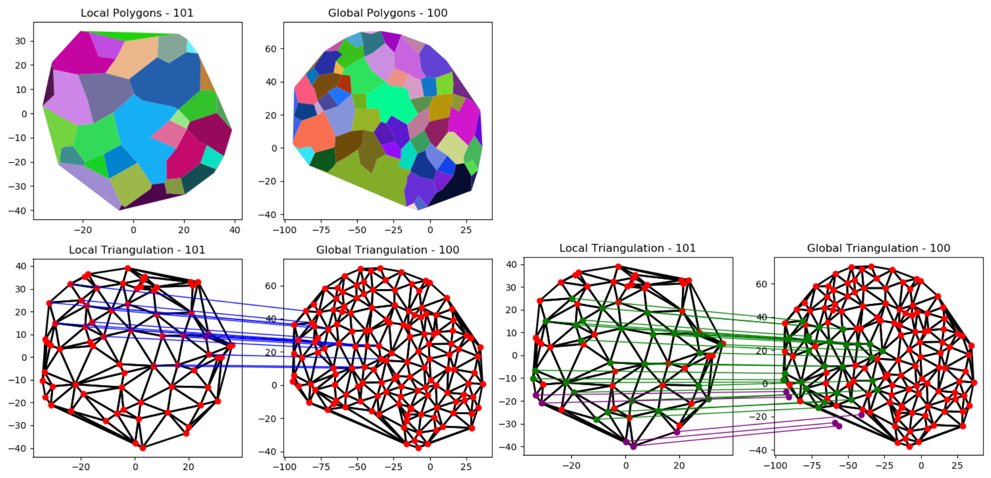


Figure 4.9: Offline display of initial polygon matching results between a keyframe and the global map, depicted as blue lines. This display could be toggled to show the improved landmark association results (green) and the positions of potentially new landmarks (purple) instead of the initial polygon matching results.

time.

Fig. 4.9 depicts a visualization of the landmark associations between the keyframe and global map after the a new global pose has been estimated (section 4.3.3). It displays the geometric hierarchies of a keyframe and the global map side-by-side and draw lines to demonstrate that two landmark positions have been associated. Initially, this visualization only depicted the landmark associations resulting from initial polygon matching, but we added a feature which enables the user to instead show the improved landmark association results. Matplotlib takes too long to draw these graphics in real-time, so this visualization can only be run once a simulation is complete. New landmarks have been added to the graph (section 4.3.4) prior to optimizing the graph state (section 4.3.5).

Fig. 4.10 depicts a visualization for the state of GraphSLAM after new landmarks have been added to the graph (section 4.3.4) prior to optimizing the graph state (section 4.3.5). GeoSLAM would serialize this data (for ease of development) in a ROS message, transmit it to another ROS node, which would display this visualization in real-time (one keyframe at a time). Dots represent landmarks and triangles represent the robot's poses. Previously observed landmarks are red, currently observed landmarks are green, newly established

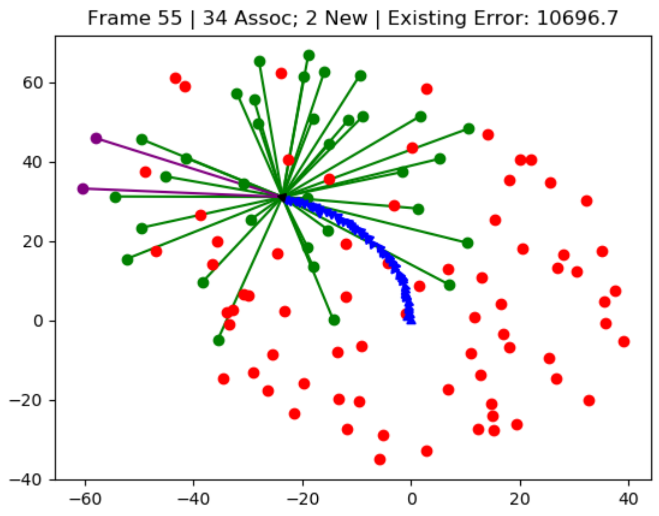


Figure 4.10: Real-time display of GeoSLAM's graph state. In this example, the robot has travelled more than one-quarter of a circular path. From its most recent keyframe, GeoSLAM was able to find 34 associations with previously mapped landmarks (green) and establish the position of 2 new landmarks (purple). The black triangle, blue triangles, and red dots represent the robot's current estimated pose, the robot's previous poses, and the other existing landmark positions respectively. The error statistic represents the cumulative error, or "disagreement", between the constraints in the graph, which can signal when the graph should be rolled back (see Section 4.5.6).

landmarks are purple, previous robot poses are blue, and the current robot pose is black. This visualization was vital for helping us troubleshoot issues with the GraphSLAM implementation and identify when GeoSLAM accepted an incorrect landmark association, causing the map to become unstable.

There were two other types of visualizations we developed. The first were visualizations that illustrated frame transformations within the simulated environment, specifically from the global frame to the robot's local frame to the robot's ground truth odometry frame. This was especially helpful when we needed to verify that simulated robot paths and observations were working as intended. The second were a set of live visualizations that illustrated the formation of keyframes from various observations.

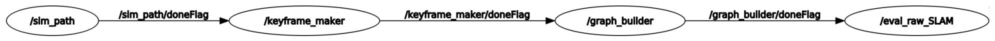


Figure 4.11: GeoSLAM ROS Node Architecture modified for scalable experimentation. Data that was originally passed over the network is instead directly read from log files. Strings containing the path to the run's log files are transmitted along the chain of nodes to indicate when the run's data is ready for further processing.

4.5.8 Computing Results

GeoSLAM required the executions of many simulations in order to adequately evaluate its accuracy and stability. However, the ROS framework does not facilitate experiment automation in its framework. In addition, because our experiments assume that GeoSLAM will be processing every observation and keyframe, this information must be stored somewhere instead of only communicating it via network connection. Therefore, we had to modify our original ROS Node architecture to enable scalable experimentation on GeoSLAM. Fig. 4.11 depicts the new architecture built for running the experiments described in Section 5.

Since our original code was designed to run one instance of GeoSLAM and then close, we needed to modify the code to include a new run option. The key to our new option is that each node would erase its memory to prepare for incoming data from a new simulation. First, the robot simulation node would record multiple random paths (one at a time) in log files. After a given path was simulated, a flag would be sent to the local bundle adjustment node, telling it to look at the log files in a given directory to generate a sequence of keyframes for the global map. When all of the keyframe data is ready, the flag is passed to the next node to process all of the keyframes into a global map. After this, the flag would be passed to a new node which will compare each state of the global map against the ground truth of the simulation. The new node aggregates the data across the simulations, awaiting a shutdown command cascading from the simulation node to know when all simulations have been completed and the aggregated results can be displayed.

Comparing the ground truth robot and landmark positions was also nontrivial. The ground truth must be transformed into the reference frame of the robot's map to enable valid comparison. Assuming that the observations are indexed, we need to track which observation serves as the base of each keyframe. With this knowledge, we need to transform the ground truth tree positions and robot poses w.r.t the robot's pose in the observation used as the base of the first keyframe. The observation used here may differ depending on how many observations it takes to construct the first keyframe.

At this point, we can find the error in each robot pose across all map states. Pose error is separated into two values: position error (Euclidean distance) and orientation error (absolute difference between angles). We also find the Euclidean distance from each landmark in each map state to the closest transformed ground truth landmark position. This nearest-neighbor approach is necessary because landmarks can only be described by their position. The error in the estimated robot poses (i.e. its path through the environment) and landmark positions is averaged across all unique states of the global map to obtain three metrics: robot position error (RPE), robot orientation error (ROE), and landmark position error (LPE). RPE and LPE is measured in cm, whereas ROE is measured in degrees. Our evaluation does not depict ROE because the error was below 1° across all experiments.

Chapter 5

Evaluation

In order to demonstrate the stability and effectiveness of GeoSLAM, we evaluate its ability to accurately estimate the robot's pose and the positions of landmarks over time. Experiments are performed using the simulation software described in Sections 4.5.2, 4.5.3, and 4.5.8.

Tables 5.1, 5.2, and 5.3 provide the parameter values used throughout the evaluation of GeoSLAM. While GeoSLAM is designed to be adaptable for various environments and conditions, we utilize this one configuration of parameters to demonstrate its inherent flexibility. Please refer to Section 4.4 for insight on setting the values in Tables 5.2 and 5.3. The values of parameters marked with an * in Table 5.1 indicate that it will act as an independent variable in one of the following experiments. In addition, i was lowered to 0.1 for certain runs that simulated high noise levels in the experiment from Section 5.3.

Table 5.1: Default Simulation Parameter Values (*experiment-dependent)

Parameter	Value	Description
F	intermediate*	forest density
D	200	robot travel distance (m)
P	circle*	robot path type
i	0.2	observation interval (m || rad)
C	0.3	minimum distance to any landmark (m)
r	40	robot sensor range (m)
κ	90%*	probability of successful landmark detection
ϵ	0.2*	st. dev. of position estimation error (m)
j	10	number of paths simulated

Table 5.2: Local Bundle Adjustment Parameter Values

Parameter	Value	Description
s	0.05	polygon perimeter step for descriptor computation
Δ_N	3	tolerance side count for polygon comparison
η	0.5	triangle match ratio
τ	9	polygon descriptor distance threshold
λ	1	landmark association distance threshold (m)
γ	7	minimum associations for valid alignment
n	8	expected observations per keyframe
f	3	stale frame threshold
σ	1.0	cluster tolerance (m)

Table 5.3: Global Mapping Parameter Values

Parameter	Value	Description
s	0.05	polygon perimeter step for descriptor computation
Δ_N	3	tolerance side count for polygon comparison
η	0.5	triangle match ratio
τ	2	initial polygon descriptor distance threshold
ψ	0.5	increment amount for τ
m	10	maximum value for τ
γ	16	minimum associations to begin pose estimation
α	75	maximum attempts to estimate robot pose
ξ	2	landmark association sample size for estimating pose
Λ	0.2	landmark association threshold (m)
ρ	2.5	landmark discovery threshold (m)
μ	6	association threshold for valid pose estimate
β	0.9	landmark association ratio to accept pose estimate
k	4	observations to establish new global landmark
ω	5	stale frame threshold
δ	0.5	landmark discovery association threshold (m)

Table 5.4: Path Type Experiment Results

Path Type	Landmark Count	LPE (cm)	RPE (cm)	Map Updates
circle	224.8	21.34	21.21	124.9
linear	289.0	24.87	25.26	124.7
figure-eight	138.0	14.45	15.22	124.7
square	211.3	17.89	18.49	126.8
triangle	206.4	23.54	22.26	124.9

5.1 Path Type

This first experiment attempts to uncover how the robot's exploration pattern affects the accuracy of its map. The simulation runs observations along paths arranged randomly within the intermediate-density forest. Ten simulations are run for each type of path seen in Fig. 4.7. Table 5.4 depicts the average number of unique landmarks, LPE, RPE, and number of keyframes used to update the global map across each path type.

Before discussing these results, we will consider the important implications of some of the path types. Every path spans the same distance (200m). The linear path covers the most ground, since the robot only moves forward. This explains why the linear path observes more unique landmarks than the other paths. The figure-eight path consists of two circular paths 100m in length each. Therefore, the robot will be taking more overlapping observations of a smaller set of landmarks. Lastly, the square and triangle paths feature the robot rotating in place. Because of this, the robot will be taking an extra 22 and 20 observations of its environment respectively. This ultimately means that the square path allows for the global map to be updated a maximum 127 times, the triangle path 126 times, and the circle, figure-eight, and linear paths 125 times.

The only useful result from this data indicates a subtle positive correlation between the number of unique landmarks observed across a robot's path and the global map's historical landmark position error.

5.2 Forest Density

The next batch of simulations was designed to evaluate how the number of landmarks per observation affects the performance of GeoSLAM. Observations were simulated around ten circular paths for each type of forest: dense, intermediate, and sparse. Table 5.5 depicts the average number of landmarks

Table 5.5: Landmark Density Experiment Results

Forest Density	Landmarks Detected per Observation	Map Updates	RPE (cm)	LPE (cm)
Dense	189.6 / 210.8	125.0	11.37	12.18
Intermediate	61.7 / 68.6	124.9	21.21	21.34
Sparse	37.4 / 41.5	122.1	22.77	21.19

detected per observation (out of how many within sensing range), number of global map updates (125 maximum), RPE, and LPE across the ten simulations for each forest density type.

Besides demonstrating that GeoSLAM can adapt to mapping environments with varying amounts of observable landmarks, this experiment shows that GeoSLAM can estimate the positions of the robot and the landmarks in its map more accurately when it observes more landmarks per keyframe (i.e. having more samples improves its approximations).

More importantly, GeoSLAM is able to successfully update its map more often when observing more landmarks per keyframe. When more landmarks are observed, there will be more polygons available for matching. Even if some landmarks are not observed, GeoSLAM only needs one matching pair of polygons to align two frames and extend associations to other landmarks in each observation. Therefore, when each observation has less landmarks, less polygons will be created, and there will be a lower probability that any two polygons will match between observations, especially when omitting a landmark from the observation impacts the configuration of the polygons that were built using the other landmarks in its proximity.

5.3 Noise Levels

Perhaps the most important experiment evaluated how sensor noise impacts GeoSLAM's performance. We simulated the robot exploring the intermediate-density forest along 10 different circular paths with different amounts of detection and position noise. The average RPE and LPE across the runs for each noise profile are given in Tables 5.6 and 5.7 respectively. Although the parameters of GeoSLAM could be further tuned to improve its performance in mapping against more difficult noise profiles, these results demonstrate that GeoSLAM can maintain an accurate map of the environment even when incoming observations contain varying amounts of noise.

As was mentioned earlier (Section 5), the interval between observations (i)

Table 5.6: Noise Level Experiment: RPE Results. Noise profiles with results in gray cells indicate that observations were taken every 10cm to sample landmarks more frequently. Accuracy values in bold indicate that some runs resulted in noticeably fewer successful map updates.

Detection Position	100%	95%	90%	85%	80%
0cm	0.41	0.64	1.20	1.23	1.00
10cm	2.97	4.16	8.44	12.63	13.56
20cm	7.24	12.46	17.95	26.32	31.39
30cm	17.62	18.93	37.62	38.98	**30.54**
40cm	27.47	37.79	43.05	**41.47**	**43.54**

Table 5.7: Noise Level Experiment: LPE Results. Noise profiles with results in gray cells indicate that observations were taken every 10cm to sample landmarks more frequently. Accuracy values in bold indicate that some runs resulted in noticeably fewer successful map updates.

Detection Position	100%	95%	90%	85%	80%
0cm	0.26	0.41	1.11	0.94	0.39
10cm	3.05	4.01	8.33	11.35	12.17
20cm	8.56	12.31	17.01	25.70	30.12
30cm	22.66	18.42	40.26	40.56	**24.95**
40cm	40.39	37.65	43.52	**40.48**	**56.46**

was lowered to from 20cm to 10cm for runs that simulated high noise levels. This applies to the noise profiles whose results are in gray cells. The noise from these runs were so intense that GeoSLAM required more overlapping observations to confidently build its global map. This should not be perceived as a weakness for GeoSLAM; there is some threshold of noise where information becomes unintelligible, especially when the information is not very descriptive (i.e. landmarks are indistinguishable besides their observed position). Instead, it should be seen as a strength that GeoSLAM can utilize those extra observations to build an accurate map despite high noise levels.

The accuracy values from Tables 5.6 and 5.7 in bold indicate that some runs associated with these noise profiles resulted in noticeably fewer successful map updates. When a run consists of fewer map updates, its accuracy values are more likely to be higher or lower than the other results because fewer position estimates factor into the average accuracy. These runs have fewer map updates because GeoSLAM will not add the new keyframe data if certain requirements are not met. First, at least 16 landmark associations between the keyframe and global map result from initial polygon matching using a polygon descriptor threshold of at most 10. Second, at least one robot pose estimate must correspond to at least 6 improved landmark associations. Third, after optimizing the global map using GraphSLAM, the error associated to the newest robot pose node in the graph must not exceed 10000 (this number is arbitrary and is primarily used to indicate when constraints originating from this node disagree significantly from other constraints in the graph). In all of these cases, GeoSLAM ignores the keyframe when it cannot recognize the robot's current location or it has reason to doubt the landmark associations it defined when localizing the robot in the global map.

This problem may be occurring for a few different reasons. First, noise levels could be so high that individual observations are not able to align with each other to create keyframes. In addition, when keyframes are created, they may contain enough noise such that it cannot meet the global map's strict landmark association requirements. If this is the case, then it could indicate that the association thresholds were not generous enough given the amount of noise in each observation. Lastly, as the robot explores new areas of the environment, GeoSLAM will not be able to perform place recognition with incoming keyframes if it cannot quickly and consistently establish new landmarks in the global map. Even though any single one of these mistakes could cause GeoSLAM to mess up the global map, GeoSLAM demonstrates that it can maintain a stable representation of the environment regardless of how unreliable its observation data is.

5.4 Localization in Large Environments

As opposed to the previous experiments, which were concerned with evaluating the accuracy and stability of GeoSLAM over time, this next experiment was designed to evaluate GeoSLAM's ability to perform one-shot localization in large scale environments. Here, GeoSLAM is provided with an initial map of the intermediate-density forest covering 1km^2, containing 13770 landmarks forming 5110 polygons. We simulated the robot observing this forest along 10 circular paths with 90% landmark detection success rate and 20cm position noise. Although not representative of practical use-cases, to provide the most challenging test, each keyframe is localized in the global map independently, which means it does not account for previous pose estimates. In the end, all but 6 out of 1250 keyframes resulted in localizing the robot within 0.5m of the ground truth, where 4 keyframes localized the robot within 1m of the ground truth and 2 keyframes were skipped due to insufficient matching. In all 10 runs, GeoSLAM completed the path successfully. The average runtime to perform initial polygon matching was 129ms, and the average runtime to estimate the robot pose and obtain landmark associations for GraphSLAM was 179ms.

5.5 GeoSLAM Efficiency

Although not an experiment by itself, this section provides insight into the efficiency of each GeoSLAM component. Fig. 5.1 illustrates the median time each GeoSLAM component required to process each incoming keyframe. Landmark Discovery is not depicted because all runtimes were under 1ms. In this case, GeoSLAM was tested using our default simulation parameters provided in Table 5.1.

Polygon matching between the keyframe and the global map is very fast, especially since there are far fewer polygons than there are individual landmarks. In addition, the number of polygons in the global map grows much slower than the number of landmarks as more of the environment is explored. Polygon matching also includes many opportunities to short-circuit comparisons, which reduces the time spent processing associations that do not likely match.

The slowest component of this system is GraphSLAM, whose runtime scales with number of poses estimated and landmarks observed. Even if the robot does not move, GraphSLAM's runtime will constantly increase over time since the landmarks around it will still be repeatedly observed.

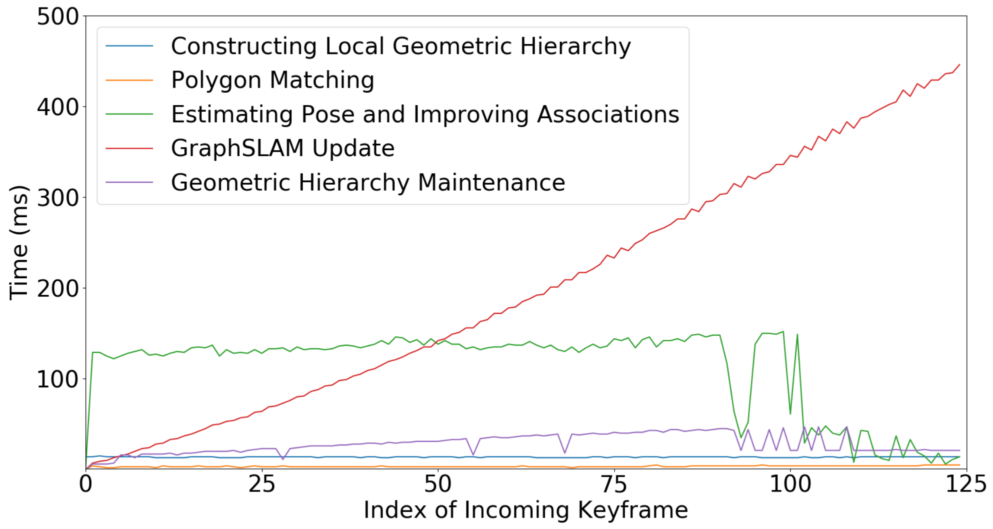


Figure 5.1: GeoSLAM Runtimes per Incoming Keyframe

Global pose estimation is relatively slow because it usually performs all 75 attempts before exiting. Towards the end of each run, global pose estimation usually terminates early since incoming keyframes are no longer discovering new landmarks as the robot nears its starting position.

The Geometric Hierarchy Maintenance component will either update the polygon descriptors or reconstruct the global geometric hierarchy. The faster runtimes can be attributed to when the polygon descriptors are updated. This tends to occur more frequently towards the end of each run because new landmarks are no longer being established on the map. Constructing a geometric hierarchy requires more time as the number of landmarks increases. This is why Constructing the Local Geometric Hierarchy takes roughly the same amount of time across all keyframes (since the robot is not moving into an area with a higher density of landmarks) while reconstructing the global geometric hierarchy takes longer over time.

Chapter 6

Discussion

Overall, our results indicate that GeoSLAM can accurately and efficiently localize a robot using unlabeled landmarks across a variety of realistic scenarios. We are able to show that, using a single parameter configuration, GeoSLAM can maintain a stable and accurate representation of environments with various landmark densities when they are observed at varying levels of noise. As expected, GeoSLAM can update its global map more accurately and reliably when it observes more landmarks per keyframe. In addition, we have shown that the polygons created from one keyframe remain distinguishable even when matching against the entirety of a large environment. We have also demonstrated that even when an environment contains thousands of landmarks, by grouping landmarks into polygons, we can efficiently find candidate landmark associations between observations.

That being said, GeoSLAM does not necessarily produce more accurate results compared to other SLAM implementations. In addition, although it was designed to solve the online-SLAM problem, there are a couple glaring performance and scalability issues that must be addressed to ensure consistent real-time capability for the collective system. Most importantly, this design does not immediately demonstrate much practical value. The majority of SLAM systems are designed specifically with a perception system in mind. Since GeoSLAM does not interface directly with raw sensor data, it is difficult to compare this system with other SLAM implementations.

What sets GeoSLAM apart from other implementations of SLAM is that it only utilizes the positions of landmarks to perform place recognition quickly and accurately. Other SLAM implementations will often utilize unique descriptors to clearly identify locations, which makes it easier to perform data association correctly. Since SLAM back-ends can only maintain a stable map state through perfect data association, other systems may elect to spend more computation time producing their unique descriptors with as much accuracy as possible. This can be problematic when such systems are used in environments that require quick decision making (e.g. autonomous driving scenarios).

GeoSLAM exemplifies the potential for a paradigm shift in the design of place recognition systems. Instead of spending so much computational resources on perfectly understanding which unique objects are within the robot's proximity, practical implementations could derive a vague, imperfect abstraction of the environment so that they can save resources for other downstream processes. Reliable place recognition systems, like what is demonstrated within GeoSLAM, can still utilize this abstraction to effectively perform SLAM despite how much noise it may contain.

Chapter 7

Future Work

Although the results that GeoSLAM produced were quite promising, there are many components that could be improved upon and explored further. Changing the SLAM implementation and how the global geometric hierarchy is updated can improve GeoSLAM's scalability. GeoSLAM can also be more reliable through the implementation of robust initialization and correction methods for its global map. Lastly, there is the opportunity to evaluate GeoSLAM's effectiveness when processing more realistic data.

7.1 Incremental Geometric Hierarchy Updates

Reconstructing the entire hierarchy every time a new landmark is added to the global map will not be sustainable for mapping larger environments. Instead, a method should be implemented to incrementally modify the geometric hierarchy based on local changes in the global map.

A few problems must be solved so that this approach works as intended. First, there must be a method to quickly find the Delaunay triangles that will be added or changed depending on which landmarks changed positions and where new landmarks are located. Utilizing data structures that are designed to organize geometric data would be especially useful here. Second, triangle and polygon descriptors only need to be recomputed when a triangle's longest edge changes, causing the configuration of polygons to change, too. Third, any changes to the configuration of the geometric hierarchy should occur in-place within its physical data structure.

7.2 Global Map Initialization

When not provided with a user-defined map, GeoSLAM uses the first keyframe as its initial global map. GeoSLAM does not validate this keyframe, so if the keyframe contains incorrect information, the map cannot be easily fixed. A more robust method of initializing GeoSLAM's global map is required to prevent a single initial bad keyframe from disrupting the robot's understanding of the environment.

One option would be to utilize GeoSLAM's landmark discovery functionality when initializing the global map. GeoSLAM could save the first several keyframes separately and estimate transformations between each of them. As the data from each keyframe is transformed into a single reference frame, GeoSLAM's landmark discovery procedure could be followed to find agreement between each keyframe and establish individual landmarks in the global map. Once enough landmarks have been established in the global map (such that a certain amount of polygons can be defined), the previous keyframes can be forgotten, and the original method for localizing incoming keyframes in the global map will commence.

7.2.1 Correcting the Global Map

GeoSLAM can prevent data association mistakes from propagating into the global map. However, when a false landmark is saved to the map or the robot can no longer localize itself using its surroundings, it is not easy for GeoSLAM to recover. The following subsections describe a couple of new features that would be capable of fixing the map state.

7.2.2 Deleting False Landmarks

GeoSLAM currently cannot remove false landmarks from its global map. It simply does not have enough information to know when a newly established landmark should not have been created in the first place. However, it may be possible to analyze GeoSLAM's raw place recognition results to determine how helpful each landmark is towards localizing incoming keyframes.

When the robot is estimated to be within sensing range of a given landmark in the global map, the place recognition statistics associated with that landmark will be tracked. During initial polygon matching, we count how many successfully-matched polygons and triangles contained this landmark and whether this landmark had a correspondence in the keyframe. We also identify whether this landmark was associated after estimating the robot's new pose. If the landmark is false, its polygons will not likely match with anything in the keyframe and it will likely not have any associations itself. After some amount of keyframes where the landmark does not contribute to the place-recogntion, it would be safe to assume that the landmark is false and can be removed from the global map.

7.2.3 Merging Disjoint Maps

In regards to how GeoSLAM initializes global maps, there is also the possibility that it could maintain multiple disparate global maps. This would especially be useful if there is a temporary localization failure in the primary global map, for example due to a lower density of landmarks in a certain region. When a new keyframe is received, it would be matched against each global map. If a keyframe matches with at least two global map, then those maps can be merged. This concept is similar to how other SLAM systems implement loop closure, and is also comparable to how keyframes are constructed for GeoSLAM in Section 4.2.

7.3 Other SLAM Back-ends

The actual implementation of SLAM is not a key component in the design of GeoSLAM. In this current implementation, GraphSLAM dominates the computation time of the system as more observations are recorded. GraphSLAM inherently has poor scalability, but various approaches are available that can alleviate this issue. For example, GraphSLAM could maintain submaps for different portions of the environment. This would ultimately structure the graph hierarchically, which would enable quicker SLAM computations for long-term explorations of large environments. There is also the option to try implementing a probablistic SLAM solution. These methods are characteristically built for solving SLAM in real-time, but may suffer from poor accuracy in some situations.

7.4 Realistic Data

This work only exhibits GeoSLAM processing simulated observations of landmarks. While we attempted to make our simulations as realistic as possible, many concessions were made to enable easier implementation. For example, we assume that the robot has the ability to observe any landmark within a certain distance of itself. In reality, landmarks like trees could occlude each other. The simulation could be improved by assigning a width to each tree and realistically implementing line-of-sight to accurately portray which landmarks are physically visible to the robot. There is also the potential to feed real data into GeoSLAM. Future work could utilize a front-end perception system to observe an environment, turning real sensor data into 2D landmark positions for GeoSLAM. Different sensor equipment configurations and feature extraction methods can be used to vary the quality of landmark positioning for these observations.

Chapter 8

Conclusion

This work has explored how place recognition for virtually identical landmarks can be implemented to successfully maintain a stable and accurate understanding of a robot's environment regardless of situational conditions, culminating in the creation of GeoSLAM. We utilize geometric hierarchies to find transformations between local observations of the environment and ultimately localize the robot. We aggregate data across observations into keyframes to reduce the noisiness of landmark positions. We safeguard the optimization of our global map against incorrect data associations using multiple verification techniques. Through our simulations, we demonstrate that, using a single parameter configuration, GeoSLAM can maintain a stable and accurate representation of environments with various landmark densities when they are observed at varying levels of noise. Future work can improve GeoSLAM's scalability, but there is also the potential that the concepts featured in GeoSLAM's design could be applied to existing algorithms to improve their performance as well.

Bibliography

[1] Robert Bridson. Fast poisson disk sampling in arbitrary dimensions. In *ACM SIGGRAPH 2007 Sketches*, SIGGRAPH '07, page 22–es, New York, NY, USA, 2007. Association for Computing Machinery.

[2] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

[3] Steven W. Chen, Guilherme V. Nardari, Elijah S. Lee, Chao Qu, Xu Liu, Roseli Ap. Francelin Romero, and Vijay Kumar. Sloam: Semantic lidar odometry and mapping for forest inventory. *IEEE Robotics and Automation Letters*, 5(2):612–619, 2020.

[4] Winston Churchill and Paul Newman. Experience-based navigation for long-term localisation. *The International Journal of Robotics Research*, 32(14):1645–1661, 2013.

[5] Stephen A. Cook. *The Complexity of Theorem-Proving Procedures*, page 143–152. Association for Computing Machinery, New York, NY, USA, 1 edition, 2023.

[6] Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1052–1067, 2007.

[7] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics & Automation Magazine*, 13(2):99–110, 2006.

[8] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(3):611–625, 2018.

[9] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part II 13*, pages 834–849. Springer, 2014.

[10] Abel Gawel, Titus Cieslewski, Renaud Dubé, Mike Bosse, Roland Siegwart, and Juan Nieto. Structure-based vision-laser matching. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 182–188, 2016.

[11] Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, 2010.

[12] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2d lidar slam. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278, 2016.

[13] Marian Himstedt, Jan Frost, Sven Hellbach, Hans-Joachim Böhme, and Erik Maehle. Large scale place recognition in 2d lidar scans using geometrical landmark relations. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5030–5035, 2014.

[14] Kin Leong Ho and Paul Newman. Loop closure detection in slam by combining visual and spatial appearance. *Robotics and Autonomous Systems*, 54(9):740–749, 2006.

[15] Leyao Huang. Review on lidar-based slam techniques. In *2021 International Conference on Signal Processing and Machine Learning (CONF-SPML)*, pages 163–168, 2021.

[16] Mohd Javaid, Abid Haleem, Ravi Pratap Singh, and Rajiv Suman. Substantial capabilities of robotics in enhancing industry 4.0 implementation. *Cognitive Robotics*, 1:58–75, 2021.

[17] Fabjan Kallasi and Dario Lodi Rizzini. Efficient loop closure based on falko lidar features for online robot localization and mapping. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1206–1213, 2016.

[18] Qingqing Li, Paavo Nevalainen, Jorge Peña Queralta, Jukka Heikkonen, and Tomi Westerlund. Localization in unstructured environments: Towards autonomous robots in forests with delaunay triangulation. *Remote Sensing*, 12(11):1870, 2020.

[19] Zhe Liu, Shunbo Zhou, Chuanzhe Suo, Peng Yin, Wen Chen, Hesheng Wang, Haoang Li, and Yunhui Liu. Lpd-net: 3d point cloud learning for large-scale place recognition and environment analysis. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2831–2840, 2019.

[20] Zhihao Liu, Quan Liu, Wenjun Xu, Lihui Wang, and Zude Zhou. Robot learning towards smart robotic manufacturing: A review. *Robotics and Computer-Integrated Manufacturing*, 77:102360, 2022.

[21] John McCormac, Ankur Handa, Andrew Davison, and Stefan Leutenegger. Semanticfusion: Dense 3d semantic mapping with convolutional neural networks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4628–4635, 2017.

[22] Raúl Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[23] Guilherme V. Nardari, Avraham Cohen, Steven W. Chen, Xu Liu, Vaibhav Arcot, Roseli A. F. Romero, and Vijay Kumar. Place recognition in forests with urquhart tessellations. *IEEE Robotics and Automation Letters*, 6(2):279–286, 2021.

[24] Richard A. Newcombe, Steven J. Lovegrove, and Andrew J. Davison. Dtam: Dense tracking and mapping in real-time. In *2011 International Conference on Computer Vision*, pages 2320–2327, 2011.

[25] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 2161–2168, 2006.

[26] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (fpfh) for 3d registration. In *2009 IEEE International Conference on Robotics and Automation*, pages 3212–3217, 2009.

[27] Sivic and Zisserman. Video google: a text retrieval approach to object matching in videos. In *Proceedings Ninth IEEE International Conference on Computer Vision*, pages 1470–1477 vol.2, 2003.

[28] Gian Diego Tipaldi and Kai O. Arras. Flirt - interest regions for 2d range data. In *2010 IEEE International Conference on Robotics and Automation*, pages 3616–3622, 2010.

[29] Gian Diego Tipaldi, Luciano Spinello, and Wolfram Burgard. Geometrical flirt phrases for large scale place recognition in 2d range data. In *2013 IEEE International Conference on Robotics and Automation*, pages 2693–2698, 2013.

[30] Konstantinos A. Tsintotas, Loukas Bampis, and Antonios Gasteratos. The revisiting problem in simultaneous localization and mapping: A survey on visual loop closure detection. *IEEE Transactions on Intelligent Transportation Systems*, 23(11):19929–19953, 2022.

[31] RB Urquhart. Algorithms for computation of relative neighbourhood graph. *Electronics Letters*, 14(16):556–557, 1980.

[32] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. In *Proceedings of Robotics: Science and Systems (RSS '14)*, July 2014.

[33] Shishun Zhang, Longyu Zheng, and Wenbing Tao. Survey and evaluation of rgb-d slam. *IEEE Access*, 9:21367–21387, 2021.