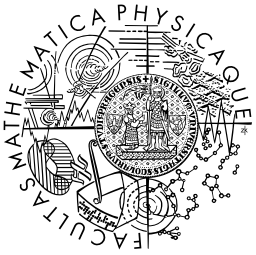

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Lukáš Jelínek

Graph-based SLAM on Normal Distributions Transform Occupancy Map

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the bachelor thesis:	RNDr. David Obdržálek, Ph.D.
Study programme:	Computer Science
Study branch:	Programming and Software Systems

Prague 2016

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

Title: Graph-based SLAM
on Normal Distributions Transform
Occupancy Map

Author: Lukáš Jelínek

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: RNDr. David Obdržálek, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: Recent advances in Normal distributions transform occupancy map (NDT-OM) representation have proven to be a viable option for mapping static as well as dynamic environments. Scan registration methods using NDT maps offer a fast and reliable way of registering two laser scans. In this work, we combine 2D NDT mapping and scan matching with the graph-based representation of simultaneous localization and mapping (SLAM). This novel approach uses NDT mini-maps for partial map storage inside the pose graph nodes. It also includes fast incremental scan matcher for odometry estimation. The scan matcher allows to create larger mini-maps which offer better loop closure validation. This work also presents a novel robust distribution to distribution (D2D)-NDT scan matching. It is used for loop closure registration and validation of correct matches. The implementation can operate as an online algorithm inside the Robot Operating System (ROS) framework. The algorithm was tested on MIT Stata Center datasets.

Keywords: SLAM NDT incremental scan matching ROS robot localization and mapping

I acknowlage my colleague Jiri Horner for sharing his knowledge in C++ programming. I would like to thank to my family and friends for all their support.

Contents

Introduction **3**

1 NDT SLAM problem analysis **4**
1.1 SLAM problem definition . . . 4
1.2 SLAM's position estimation categories . . . 5
1.3 Map representation . . . 5
1.4 Registration . . . 7
1.5 Graph-based SLAM on NDT maps . . . 8

2 Used algorithms and key concepts **10**
2.1 Graph-based SLAM . . . 10
2.1.1 Pose graph creation . . . 10
2.1.2 Loop closure creation . . . 11
2.1.3 Optimization . . . 12
2.2 NDT mapping algorithms . . . 13
2.2.1 NDT grid . . . 13
2.2.2 NDT-OM extension . . . 14
2.3 Registration algorithms . . . 15
2.3.1 NDT registration . . . 15
2.3.2 D2D-NDT registration . . . 16
2.3.3 ICP . . . 18
2.3.4 Correlative scan registration . . . 18

3 NDT graph-SLAM overview **20**
3.1 System composition . . . 20
3.2 Moving window . . . 22
3.3 NDT frame creation . . . 24
3.4 Loop closure detection . . . 24
3.5 Robust D2D-NDT registration . . . 25
3.5.1 Adaptation of correlative registration . . . 25
3.5.2 Algorithm overview . . . 26
3.5.3 Solution validation . . . 27

4 Implementation **29**
4.1 Used libraries . . . 29
4.1.1 Robot operating system (ROS) . . . 29
4.1.2 The Point Cloud library . . . 29
4.1.3 The G2O . . . 29
4.1.4 The Eigen . . . 29
4.2 Structure of the implementation . . . 30
4.2.1 NDTGrid2D . . . 30
4.2.2 VoxelGrid2D . . . 31
4.2.3 NDTCell . . . 31
4.2.4 Registration algorithms . . . 32

5 Evaluation of NDT graph based SLAM **33**
5.1 MIT dataset details . . . 33
5.2 ROS SLAM algorithm overview . . . 34
5.2.1 The Gmapping . . . 34
5.2.2 The Hector SLAM . . . 34
5.3 NDT-GraphSLAM evaluation . . . 35
5.3.1 NDT frame generation frequency . . . 35
5.3.2 Robust D2D score threshold . . . 36
5.3.3 Iterative mapping of room . . . 37
5.3.4 Long corridors . . . 39

6 Future works **42**

Conclusion **43**

Bibliography **44**

List of Figures **47**

List of Abbreviations **48**

Appendices **49**

Appendix A ndt_gslam package **50**
A.1 Overview . . . 50
A.2 Architecture . . . 50
A.3 Parameter specification . . . 50
A.4 ROS API . . . 51

Introduction

Humanity has envisioned many tasks which could be carried out by robots including transportation, health care, save and rescue and much more. Robots of the current world can efficiently operate only in very limited conditions. To solve problems of the future we need fast and reliable algorithms for our robots. The big question in the field of mobile robotics is how efficiently localize a robot and create the map as precise as possible. This problem is often referred to as Simultaneous localization and mapping (SLAM) problem.

Precise localization is a crucial part of any good navigation software. Generated map plays an important role in path planning and multi-robot coordination. SLAM algorithm should rely mostly on robots internal sensors like, e.g., sonars, cameras, wheel encoders. Using Global positioning system (GPS) is only possible in an outdoor environment. The precision of this localization is very often not good enough to successfully navigate robot.

The solution to the full problem of map building and robot positioning needs to combine algorithms for map representation, sensor measurement registration and position estimation. This work presents a novel approach in full SLAM problem based on Normal distributions transform (NDT) maps. In recent years NDT map building process has proven to be a reliable choice for scan registration. A map representation based on NDT can handle dynamic objects and updates occupancy. The pose estimation problem was in recent years solved mostly by graph-based SLAM optimizing engines. The graph-based method offers flexibility and speed even on big maps. Both techniques were studied separately and provide good results. The missing part is how to combine these approaches to improve robustness of full SLAM solution. To fulfill this goal, we will present a novel method for robust registration on top of NDT grids. The most challenging part of this fusion is how to represent the map. We use method based on small local mini maps which are easily used in the graph of the SLAM optimizer. Our algorithm has the additional robustness to odometry error by utilizing our NDT version of incremental scan matching. The combination of these part creates the whole system which can estimate its position without initial guess and robustly close errors caused by imprecise robot movement. On top of algorithm benefits, we wanted to make source code and implementation easily accessible and improvable. For this reason, we have decided to implement it in ROS, which is a current standard environment for robotic projects of all sizes.

This work has following structure. First chapter analyze full SLAM on NDT maps. The second chapter provides more information about algorithms used in this work. The third chapter describes the whole system of NDT SLAM. The fourth chapter makes the focus on implementation details behind the algorithms. In the last chapter, we wrap up results of this algorithm and compare it to existing ROS implementations.

1. NDT SLAM problem analysis

The full SLAM problem solution requires a combination of data association, mapping and pose estimation. In the first step, the algorithm needs to receive data from sensors. The standard SLAM requires information about the movement of the robot and robot's perception of an environment. The standard odometry tracking of the robot is done with wheel encoders or with the Integrated Measurement Unit (IMU).Perception of the environment can be obtained from 2D or 3D laser scanner. Another option is to use stereo cameras or Kinect[1].

SLAM algorithm based on laser scans are still frequently used in real life applications. The standard versions work with 2D scans [KMvSK11] [GSB07b]. Two scans can be used for a registration. It is a process which calculates relative transformation between two scans by aligning one scan on top of the other one. The registration is used for a variety of tasks, e.g. map building, odometry estimation, unique feature detection. Some registration techniques are described in the section 1.4.

The result of every SLAM solution should be a map which can be used in navigation and a trajectory planning. One possible map representation is the set of unique landmarks. The other option is to integrate measurement together and create a dense map of the environment. These methods will be analyzed in the section 1.3

Lastly, we need to estimate a position of the robot based on information from odometry and the map. Hence, we need to define what it is the position of the robot and how we will represent it.

1.1 SLAM problem definition

We describe the position estimation problem as a process which finds the location of the robot in every time step. We also need to estimate how the map will look like in every time step. In the real world, we deal with the sensors which always have some inherited noise. Therefore, we are not able to provide exact position of the robot. For this reason, we use a probabilistic definition of the problem. The robot moves through unknown space along trajectory expressed as variables $\mathbf{x}_{1:T} = \{\mathbf{x}_1, ..., \mathbf{x}_T\}$. While moving robot is taking the odometry measurements $\mathbf{u}_{1:T} = \{\mathbf{u}_1, ..., \mathbf{u}_T\}$ and the perception of environment $\mathbf{z}_{1:T} = \{\mathbf{z}_1, ..., \mathbf{z}_T\}$. The solution to position estimation is a probability of the robot's trajectory $\mathbf{x}_{1:T}$ and the map $\mathbf{m}$ of the local environment given all the measurements and the initial pose $\mathbf{x}_0$:

$$p(\mathbf{x}_{1:T}, \mathbf{m} \mid \mathbf{z}_{1:T}, \mathbf{u}_{1:T}, \mathbf{x}_0) \tag{1.1}$$

The odometry is represented as triple (x, y, θ) in 2D system. The initial pose can be interpreted as an origin of the coordinate system for the global map.

[1] `www.xbox.com/en-US/kinect`

1.2 SLAM's position estimation categories

Over the past decade researchers have developed three distinctive categories of SLAM position estimation.

The first type is the Extended Kallman Filter (EKF) variant. It is based on the Kalman filters (KF). The KF assume that probability density function is from Gaussian distributions and the position model is linear. This assumption is usually not correct for the robot movement model. The EKF solves the problem with non-linearity of the robot's pose model. The performance of EKF strongly depends on a quality of statistical model for noise in the sensors and the odometry. Unfortunately, these models are usually not available. A set of comparative tests for convergence and inconsistencies of EKF is in work of [HD07].

Another category is based on the Particle filters (PF). The set of weighted particles represents the current state of the robot. This representation has the advantage in modeling uncertainty through a multi-modal distribution and can deal with non-Gaussian sensor noise. The authors [MTKW02] proposed computationally efficient method based on the PF called FastSLAM. It uses the particles to represent posterior probability of the robot motion. Each particle also holds K Kallman filters representing landmark positions (unique features in the environment). The authors of this algorithm have demonstrated that it is possible to calculate high-precision maps utilizing FastSLAM. Inspired by FastSlam, a method based on Rao-Blackwellized Particle Filter is proposed in [GSB07a]. Derivations of this approach are still actively used in robotics today.[GSB07b]

The last category models positions of the robot with a graph representation. The least square optimization of the graph finds a possible robot trajectory over time. A graph node represents a possible pose of the robot and an edge between two nodes is a relative movement. The Nodes may also hold some information about current state of the map or a laser measurement. This representation was first time used in work of [LM97]. This technique was later improved by [OLT06]. They have presented an efficient optimization approach based on the scholastic gradient descent. It was able to correct even large graphs. Later, multiple authors have improved SLAM optimization by adding hierarchies to large graphs or adding robustness to the optimization process. The graph-based model of SLAM offers flexibility for adaptation of new improvements and can be reasonably fast even on large graphs. More details about a graph generation and optimization is in the section 2.1.

1.3 Map representation

A map of an unknown environment is a standard part of the SLAM problem solution. This map needs to be stored for local path planning and obstacle avoidance. The map precision is an important characteristic for an obstacle avoidance algorithms. The ideal map should keep low memory consumption because robots often have limited access to memory. The high-quality map is also beneficial for a precision of registration algorithms.

A point-cloud [RC11] is the map representation which stores measurements as simple points in space. It is the most accurate representation because no data from the sensor are lost. Scan-matching algorithms e.g. Iterative closest

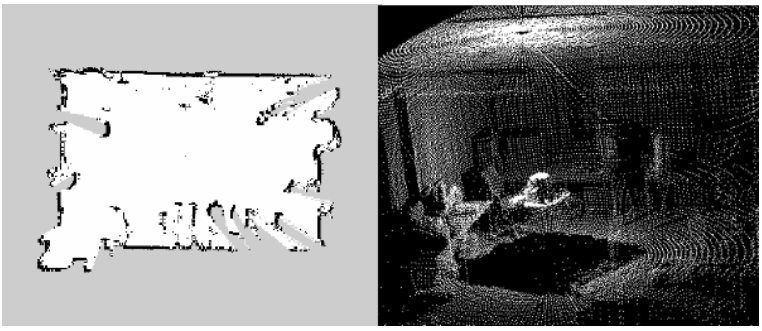

Figure 1.1: On the left is visualization of occupancy grid. On the right is visualized point cloud of a room.

point (ICP) use this data-structure. It is very easy to convert from this model to a different type of a map because it has all the information from the sensor measurement. A Problem is a memory consumption. If the robot runs for a long time with a higher frequency of the sensor data production, it is likely that robot will consume a significant amount of memory.

An occupancy map is a grid-based type of the map. It consists of the grid with cells. In every cell, it has just one value describing the likelihood that cell is occupied. This value becomes higher with more incoming data measurements. It has a constant memory consumption over time. It is also possible to use this representation for the registration [KMvSK11]. Furthermore, it can represent unoccupied spaces with a low likelihood value. This feature is used by many path planning and obstacle avoidance algorithms. The occupancy maps are the main output format for SLAM maps in ROS.

A quadtree is a tree data structure. Each node of the tree has exactly four children. The nodes are decomposing space into smaller sub-areas. Every node has its threshold. When it is reached, the cell is divided into a four smaller cells. This process dynamically changes a resolution of the grid. Therefore, we get higher precision in places where it matters more. The maximal precision is bounded by the minimal size of leaf nodes.

The NDT representation [BS03] uses grid-based data-structure. Each cell has normal distribution parameters stored inside. The normal distribution is calculated from scan points which are mapped into the cell. This model offers constant memory consumption over time. The NDT has a better internal representation of the mapped points than octree (3D case of quad-tree) which was proven in the work of [SAS$^+$13]. They have shown that a coarser NDT grid can have a better map precision than finer octree map. The standard NDT representation of the map is fully explained in the section 2.2.1. The NDT maps can also include occupancy information. This extension is called NDT-Occupancy mapping (NDT-OM) and it is presented in the section 2.2.2. Furthermore, this extension can remove dynamic objects from the map which is crucial for SLAM in dynamic environment.

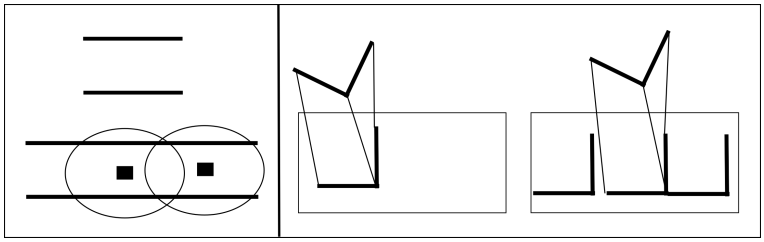

Figure 1.2: Local and global ambiguities in scan registrations. On the left is local ambiguity. On the right is global ambiguity with two maps. First map is original map without all information about environment. Second map is reality with all features. Registration wrongly associated matching based on information only from first map.

1.4 Registration

A scan registration is a key concept in the full SLAM solution. The SLAM algorithm can use scan matching between two scans to determine a transformation. It tells us how far a robot moved between two scans. Unfortunately, these scans might not offer enough information for successful registration. Imagine a robot which is standing in the corner of a room with the sensor facing the wall. Scan from this robot has only information from a very limited field of view which may lead to alignment errors. Therefore, it is usually necessary to combine individual scans to operate with more data.

One of the algorithms which use this process is called incremental scan-matching. It takes arriving scan and tries to match it against the map built from previous measurements. By doing so, it can be used as a replacement for robot odometry. The section 2.3.2 provides the NDT based algorithms which are suitable for incremental scan-matching. Another popular approach is the ICP [LM97]. All these algorithms use optimization methods (e.g. Newton's method) which require a good initial guess. Otherwise, they converge to some local minimum.

The scan registration also verifies loop closures in the graph based SLAM. The Loop closure is an edge which close the loop (creates a cycle) of robot's movement in the graph. More details about loop closure generation are in the section 2.1.2.

The scan matcher needs two scans to perform registration. The graph-based SLAM stores these measurements inside of the nodes. In case that two nodes physically overlap they share the same measurement of the environment. The registration finds this similarity and calculates transformation between nodes. The biggest problem with this alignment is that we have no valid prior information about positions of these nodes. These two scans can overlap, or they can be from completely different parts of the world. The registration needs to estimate the transformation. Additionally, it needs to correctly identify if two scans overlap. We present one such an algorithm in the section 2.3.4

However, even scan matcher with correct validation can fail to identify the overlap. This is caused by ambiguities in the environment [Ols09a].

The first is a local ambiguity. Imagine a robot which moves in a long corridor similar to one in the figure 1.2 on the left. This environment does not have many distinctive features. One of the nodes has a measurement of two straight lines

shown in top part. The second node has a measurement of the whole corridor shown on the bottom. The ellipses represent two out of an infinite number of correct alignments which would result in a perfect match. In reality, only one of them may be correct. Unfortunately, registration is not able to recognize the correct answer.

The second one is a global ambiguity. This ambiguity usually happens when the algorithm does not have enough information about the whole environment. One of the nodes has a measurement in the top part of the figure 1.2 on the right. The second node has only information in the first rectangle. From this perspective, it looks like there is only one possible match. Unfortunately, based on reality in the environment these two nodes do not overlap at all because the correct match is shown in the second rectangle. Once again registration algorithm had no chance of figuring this out without prior knowledge about the whole environment.

1.5 Graph-based SLAM on NDT maps

After initial research, we have noticed benefits of the NDT mapping. The NDT maps have a good memory consumption. They can hold occupancy information and reject dynamic objects with the use of NDT-OM extension 2.2.2. The registration algorithms for the NDT grids already exist. Unfortunately, they need initial guess for correct convergence.

The Graph based pose estimation currently represent a flexible way how to find robot's position. It is also possible to extend it to work on large scale maps. Additional topological information from the graph can be beneficial for the detection of registration ambiguities.

Further, in the work of [SSAL13b], authors have described that scan matching based on the NDT grids can provide precise result in mapping process with use of the NDT-OM extension. They have proven this by mapping large area with the incremental scan matching. This process resulted in the precise map. They have noted in the conclusion that even though results are very accurate, there is a need for a solution with loop closure mechanism to improve results. They have also tested the reliability of dynamic object rejection. Their results have proven that the NDT-OM is a really good option for a dynamic environment.

The loop closures can be created in graph based SLAM and additionally tested by robust scan matcher. This work presents robust registration method for loop closure registration and validation on NDT maps 3.5.

In the previous works, there was always one global NDT map. The iterative scan matching then used this map for the alignment of incoming scans. In this work, we use the pose graph which is optimized by SLAM's back-end. The optimization makes changes to the location of the nodes which needs to update the global map. Therefore, we present a way how to represent the map which can be updated after graph optimization 3.3.

Incremental scan matching on NDT grids was proven to get good results. Therefore, it should be included in this work as well and combine it with rest of the proposed system 3.2.

Lastly, it needs to be implemented in a way that on-line processing of real datasets is possible. It needs to have standard ROS interface commonly found in

other SLAM packages. It should use standard libraries available in ROS.

The final result of this works should be an implementation of 2D graph based SLAM on NDT maps with easy use inside of ROS ecosystem.

2. Used algorithms and key concepts

This chapter offers an introduction to multiple state-of-the-art algorithms used in this work. It starts with an explanation of a graph-based SLAM variant. The next Section 2.2 describes NDT based map representations. The section 2.3 is dedicated to NDT based registration algorithms. We also include a basic introduction to ICP and Correlative registration algorithm.

2.1 Graph-based SLAM

A graph-based SLAM constructs a graph representation of the pose estimation problem. This graph is called a pose graph. Nodes in the graph represent potential poses of a robot at certain time stamp T. Therefore, the nodes are representing our trajectory $\{\mathbf{x}_1, ..., \mathbf{x}_T\}$ Additionally, they also hold current state of the map. Edges in the graph represent possible transformation between the nodes. They also include a covariance matrix representing noise from odometry sensor. The process of edge creation is executed in algorithm's front-end. It creates them either from odometry $\mathbf{u}_T$ or by measurement data $\mathbf{z}_T$ registration. Once the graph is completed, it is optimized by algorithms back-end. Result of this process is the most likely position of all nodes in the graph.

2.1.1 Pose graph creation

Process of graph creation operates in SLAM's front-end. First step is to receive robot's movement. This transformation may come from wheels' encoders, visual odometry from camera or IMU. Front-end also receives a covariance of the transformation based on noise model of source sensor. From transformation and covariance we can create an edge for the graph. This edge type is usually called odometry edge. Consecutive odometry measurements creates long chain of edges in graph.

Nodes represent current robot position. Therefore, they should have some initial estimate. This initial guess may come from concatenation of transformations in odometry edges. Another method is to use propagation of transformation through minimum spanning tree constructed out of full graph.

Second type, represents edges from nodes to landmarks. A landmark is and unique descriptors of the place. When landmark is detected, front-end creates node representing this place and landmark edge connecting it with graph. Edge carries transformation between current node and landmark. If landmark already exists than created edge might help to optimize correct pose estimate of other nodes.

Third common type of edges are loop closure edges. These edges usually connect two nodes, which share same perception of the world. Aligning these perceptions yields virtual transformation between these nodes. A covariance needs to be provided from alignment process and depends on used technique. Loop closure

edge usually exists if we have revisited same place again. This is crucial information for SLAM's back-end. Based on it optimalization finds out if odometry edges reliably represent reality and adjust pose estimates.

2.1.2 Loop closure creation

First step of correct loop closure creation is to identify all nodes ,which might have overlapping measurements. Given pose a we find all nodes $b_1 ... b_n$ from graph whose sensor measurements overlap pose a. This could be determined by finding relative position of nodes a and b_i. One possible method how to determine is to use Dijkstra projection mentioned in [Ols09a]. Dijkstra projection starts at node a and concatenate covariances and transformation along the minimum uncertainty path. This path is selected based on determinant of covariance matrix. Small covariance matrix has lower determinant than covariance matrix with large numbers. Minimum uncertainty selection guaranties that algorithm will get to the target b_i with maximum precision. Concatenation of covariances is done based on equation:

$$P_{a+b} = J_a P_a J_a^T + J_b P_b J_b^T \tag{2.1}$$

$$J_a = \begin{pmatrix} 1 & 0 & -x\sin\theta - y\cos\theta \\ 0 & 1 & x\cos\theta - y\sin\theta \\ 0 & 0 & 1 \end{pmatrix} J_b = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.2}$$

where P_a is acumulated covariance, P_b is aditional covariance, Jaccobian J_a use parameters from transformation $(x, y, \theta)_a$ and J_b from $(x, y, \theta)_b$ Concatenation of transformations is defined as:

$$\begin{pmatrix} x \\ y \end{pmatrix}_{a+b} = \begin{pmatrix} x \\ y \end{pmatrix}_a + R(\theta_a) \begin{pmatrix} x \\ y \end{pmatrix}_b \tag{2.3}$$

$$\theta_{a+b} = \theta_a + \theta_b \tag{2.4}$$

where $R(\theta_a)$ is rotation matrix created from angle θ_a.

After successful generation of overlapping nodes, every potential pair needs to be tested by registration algorithm. This algorithm needs to be robust enough to reject as many incorrect pairs as possible. If matching is possible it should align measurements and return best transformation. More about this type of algorithms can be found in section 1.4.

Even the best registration algorithm may fail and return erroneous measurement. Loop closure process needs to reject these errors. One solution is to use method proposed by [Ols09a]. In this approach we first group loop closure edges into groups based on their topological distance from each other. Later we validate every cluster against internal inconsistencies. Edges marked as inconsistent are deleted from system.

Other option is to use robust optimization engines, witch can identify outliers in the form of error edges. Comparison of known outliers rejection methods was done by [SP13].

2.1.3 Optimization

Back-end receives graph with odometry edges and loop closure edges. The main task of back-end is to optimize this graph and return the most likely position of nodes. Popular method of optimization is to use the Gauss-Newton or the Levenberg-Marquardt algorithms.

To utilize these methods we first need to define our error function. We will use notation similar to one presented in section 1.1. Let $\mathbf{x} = (\mathbf{x}_1, ..., \mathbf{x}_T)^T$ be a vector of graph's nodes positions. Let $z_{i,j}$ to be a registration algorithm transformation between nodes x_i and x_j. Let $\Omega_{i,j}$ be a information matrix of this transformation (information matrix is an inverse of covariance). Lastly let $\hat{z}_{i,j}$ be a estimate of registration transform received from initial configurations of nodes i and j.

The log-likelihood of measurement $z_{i,j}$ is than defined as:

$$l_{i,j} = (\hat{z}_{i,j} - z_{i,j})^T \Omega_{i,j} (\hat{z}_{i,j} - z_{i,j}) \tag{2.5}$$

where $(\hat{z}_{i,j} - z_{i,j})$ is a difference between expected measurement and real measurement. Now we can define out error function as

$$F(\mathbf{x}_{1,T}) = \sum_{<i,j> \in G} (\hat{z}_{i,j} - z_{i,j})^T \Omega_{i,j} (\hat{z}_{i,j} - z_{i,j}) \tag{2.6}$$

Our goal is to calculate such a $\mathbf{x}$ that this function is minimal. More formaly we wan to find solution to

$$\bar{x}_{1,T} = argmin_{\mathbf{x}} \; F(\mathbf{x}) \tag{2.7}$$

Information on how to minimize this function, calculate derivatives and how to exploit structure of the problem to get significant speed gains continue in reading in this tutorial [GKSB10].

2.2 NDT mapping algorithms

2.2.1 NDT grid

NDT grid representation was first time used by [BS03] in their scan registration process. Central idea was to convert laser scan into grid with cells containing normal distributions. Points in space from laser scanner are first separated into corresponding cells. From points in single cell we approximate normal distribution (μ_i, P_i) by calculating mean and covariance:

$$\mu_i = \frac{1}{n}\sum_{k=1}^{n} x_k \tag{2.8}$$

$$P_i = \frac{1}{n-1}\sum_{k=1}^{n}(x_k - \mu_i)(x_k - \mu_i)^t \tag{2.9}$$

NDT grid was than used for registration.Originally proposed grid could be updated with new laser scans only by keeping used points and recalculating all cells again. This has changed with proposed recursive covariance update step by [SAS+13]. Their update step offers way how to fuse in new measurements. First it calculate normal distributions for added points. In second step, it merges old covariances with new one.

Consider two sets of measurement $\{x_i\}_{i=1}^{m}$ and $\{y_i\}_{i=1}^{n}$ than formula for mean calculation is in equation (2.11). Recursive update for covariance (RCU) is in equation (2.14)

$$T_x = \sum_{i=1}^{m} x_i \quad T_y = \sum_{i=1}^{n} y_i \quad T_{x\oplus y} = T_x + T_y \tag{2.10}$$

$$\mu_{x\oplus y} = \frac{1}{m+n} T_{x\oplus y} \tag{2.11}$$

$$S_x = \sum_{i=1}^{m}(x_i - \frac{1}{m}T_x)(x_i - \frac{1}{m}T_x)^T \quad S_y = \sum_{i=1}^{n}(y_i - \frac{1}{n}T_y)(y_i - \frac{1}{n}T_y)^T \tag{2.12}$$

$$S_{x\oplus y} = S_x + S_y + \frac{m}{n(m+n)}(\frac{n}{m}T_x - T_{x\oplus y})(\frac{n}{m}T_x - T_{x\oplus y})^T \tag{2.13}$$

$$P_{x\oplus y} = \frac{1}{m+n-1} S_{x\oplus y} \tag{2.14}$$

Proof and further explanation for these equations can be found in work of [SAS+13] and later improved in [SSAL13a].

In addition to fusing in new laser measurements we can also easily generated coarser grid by merging cells from higher resolution grid to grid with lower resolution. This mechanism is useful in path planning where we can plan on coarser grid which could be faster. Also, we can use multi-level scan matching approaches, which will be discussed in next section 2.3. Small disadvantage of this method is that we need to keep number of points used in every cell.

It is worth noting that in continual integration of scans calculated mean and covariance grow unbounded with increasing number of points added. This could lead to numerical instabilities. Second problem is that cell's distribution contains measurements from all scans. This is problem in dynamic environment where some objects might disappear. These problems are solved by restricting maximal number of points in cell with parameter M

$$N_{x \oplus y} = \begin{cases} n+m, & n+m < M \\ M, & n+m \geq M \end{cases} \tag{2.15}$$

Parameter M modifies how fast we let RCU replace old measurements by new one. Small value of M makes adaptation faster and big M keeps weight of older data higher. This cause to have new data making smaller impact on result of process.

2.2.2 NDT-OM extension

NDT grids offers good compromise between space and precision, but it lacks information about occupied space and unoccupied space. This is crucial for planning algorithms. This functionality was added to NDT by [SAS+13] and later improved by same authors in later work [SSAL13a]. Every cell in NDT-OM is represented with parameters $c_i = \{\mu_i, p_i, N_i, p_i\}$, where μ_i and P_i are parameters of estimated normal distribution, N_i is number of points in cell and p_i is probability of the cell being occupied.

Calculation of occupancy parameter is done by ray-tracing. Consider that we have current map m_x. We have calculated new NDT map m_y from incoming distance measurements. Both maps needs to be in the same coordinate system. Ray-tracing starts at current robot position in map m_x. End point of ray-tracing is value of mean from one of the cells in new map m_y. Program visits every cell along the line and updates covariance. It is important to visit every cell just once. When is ray-tracing over we merge in all cells from m_y into m_x with RCU update rule.

The main idea in occupancy update calculation is that not all cells are occupied fully. Normal distribution usually occupies only part of the cell. A ray tracing through this cell might not intersect bounds of normal distribution at all. In order to consistently update occupancy the update value should not be a constant. Better option is to choose a function describing difference between map m_y and m_x. This function with explanation might be found in [SSAL13a].

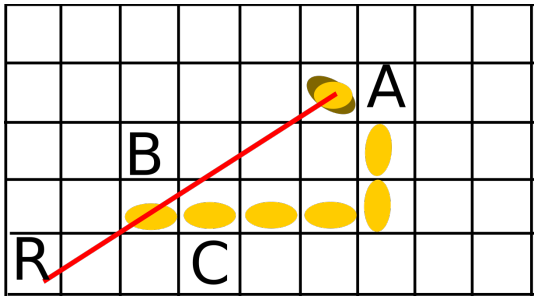

Figure 2.1: Image describing raytracing update. Yellow elipses represent normal distributions. Letter R represent robot position and red line ray tracing line. RCU will be applied to the cell marked A. A distribution in cell marked with letter B will get updated as unoccupied. Cell C will stay without any update.

2.3 Registration algorithms

2.3.1 NDT registration

NDT registration process was first time explained by [BS03]. They have explained how to make 2D registration between older scan (target scan) and newer scan (source scan). Target scan was converted to NDT grid by technique mentioned in section 2.2.1. Result of registration should be transformation defined in 2D:

$$T : \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} \tag{2.16}$$

where $(t_x, t_y)^T$ represents translation and θ represents rotation. Transformation is used for transforming source scan. At the beginning of program parameters of transformation are initialized either by zero or from initial guess. For each point of transformed scan cost function is computed This function is defined as:

$$score(\mathbf{p}) = \sum_i \exp(-\frac{1}{2}((T(x_i, \mathbf{p}) - \mu_i)^T P_i^{-1} (T(x_i, \mathbf{p}) - \mu_i)) \tag{2.17}$$

where $\mathbf{p} = (t_x, t_y, \theta)$ are parameters of transformation, $N(\mu_i, P_i)$ are parameters of normal distribution where point x is transformed by transformation T.Goal of the NDT scan-matching is to find parameters $\mathbf{p}$ which maximize this function. This maximization problem is changed to minimization problem by searching for minimal value of -score. Newton's algorithm finds minimizing parameters in p by iteratively solving equation

$$H\Delta p = -g \tag{2.18}$$

Representation of hessian, gradient and all derivations might be found in work of [Mag09]. Magnusson also introduced new scaling parameters into score function in order to reject possible outliers. probability distribution function (PDF) inside of cells of target NDT grid may not be always from normal distribution. In practice any representation which approximates structure of the element is valid. Outliers are points far from the mean of distribution and cause unbounded growth of PDF.

At the beginning, algorithm created discrete NDT grid out of target scan. This introduces discretization problems. These problems are cause by points

generating PDF which are larger than their cells. In the original work of [BS03] this was solved by creating 4 target grids where each grid is translated by half of the cell size in single direction. This process made this algorithm inefficient. Introduction to multi-layer NDT grid structure, presented by [UT11], solved this problem. Multi-layer approach consists of several grids with different resolution. Grids are ordered from coarser grid to finer grid. Algorithm starts with coarse grid and estimates parameters of transformation. Calculated transformation is used as initial guess at lower level. This principle practically eliminated need for four overlapping grids. It also offered better convergence time and increase to robustness. Algorithm is able to converge when matched scans are farther away. Good configuration is 4 layers with cell sizes 2, 1, 0.5 and 0.25 meters.

Another improvement to algorithm is usage of concept of linked cells. In practical registration very often part of the source scan lie far from any target cells. This causes only small portion of points contribute to score function. It can cause algorithm failure or just increase time of convergence. Linked cells prevent this by providing cells in target scan, which are close to the point from source scan. Implementation of this technique is possible with use of kD-tree with means of all cells as input points. Every point or source scan finds k-nearest cells and execute score calculation on them.

Algorithm 1 NDT algortihms with muilti layer and linked cell enhancements

Require: source scan, target scan, parameters (x, y, θ) of initial transformation, cell resolution for each layer

1: **function** NDTREGISTRATION($scan_s$, $scan_t$, p_{init},$resolutions$)
2: $\mathbf{p} \leftarrow p_{init}$
3: **for all** res from $resolutions$ **do**
4: $ndt_t \leftarrow$createNDTGrid(res,$scan_t$) ▷ described in section 2.2.1
5: transform each point $x_i \in scan_{trans}$ with $T(x_i, \mathbf{p})$
6: $\mathbf{p} \leftarrow$ computeSingleGrid($scan_{trans}$, ndt_t, $\mathbf{p}$)
7: **end for**
8: **return** calculated parameters $\mathbf{p}$ of transformation
9: **end function**

2.3.2 D2D-NDT registration

Distribution to distribution (D2D)-NDT is variant of NDT registration algorithm proposed by [SMAL12]. It is extension of original algorithm presented in section 2.3.1. Instead of using only one grid for target scan. This aproach uses two grids. One for source scan and second for target grid. Algorithm than minimize the sum of L_2 distances between pairs of PDF's from both grids. Formally, transformation between two sets of cells X and Y is defined as:

$$f(\mathbf{p}) = \sum_{i=1,j=1}^{n_X,n_Y} -d_1 \exp\left(-\frac{d_2}{2}\mu_{ij}^T(R^T P_i R + P_j)^{-1}\mu_{ij}\right) \tag{2.19}$$

$$\mu_{ij} = R\mu_i + t - \mu_j \tag{2.20}$$

where $\mathbf{p} = (t_x, t_y, \theta)$; $X(\mu_i, P_i)$ and $Y(\mu_j, P_j)$ are PDF's of individual cells in pair; a pair (R, t) represents rotation matrix from parameter θ and translation

Algorithm 2 Computing transformation on with single target NDT grid and source point cloud

Require: source scan, target NDT grid, parameters (x, y, θ) of initial transformation

function COMPUTESINGLEGRID($scan_s$, ndt_t,p_{init})
  while not converged **do**
    $\mathbf{p} \leftarrow p_{init}$
    $(score, g, H) \leftarrow (0, 0, 0)$
    for all points $x_i \in scan_{trans}$ **do**
      $\bar{x_i} \leftarrow T(x_i, \mathbf{p})$
      $cells \leftarrow$find k-closest cells to $\bar{x_i}$
      for all cells $c_i \in cells$ **do**
        {based on [Mag09]}
        $(score, g, H) \leftarrow (score, g, H)+$ calcNewtonParameters(c_i,$\bar{x_i}$)
      end for
    end for
    solve $H\Delta p = -g$
    $\mathbf{p} \leftarrow \mathbf{p} + \Delta p$
  end while
  return p
end function

vector $t = (t_x, t_y)$. Regulation parameters d1 and d2 are set to values $d_1 = 1$ and $d_2 = 0.05$. Equation 2.20 represents difference in means where mean u_i is transformed to new position.

Optimization of this function is done in similar way to 2.3.1 by utilizing Newtons method and solving $H\Delta\mathbf{p} = -g$. Derivations for calculation of hessian and gradient are presented in work of [SMAL12].

This algorithm is also possible to improve by iterating over multiple layers with different resolutions similar to NDT registration in previous section.

In comparison, with NDT registration this algorithm needs only NDT grids for registration. Point cloud can be thrown away after successful creation of grid. This allow saving memory and efficiently represent maps in SLAM. In addition, D2D is almost ten times faster than standard NDT registration on same dataset. This was proven in comparative study from [MVS+15]. The main cause of this speed up is smaller number of calls for score calculation. In point to distribution (P2D)-NDT mentioned in last section we need to calculate score for each point in source point cloud. In case of D2D we just calculate score function for each cell of source grid. This is done by generating only pairs between cell from source grid and closest cell from target cell. Closest cell can be easily found by using kD-tree with values of target grid's means.

2.3.3 ICP

The iterative closest point (ICP) algorithm was first introduced by [CM92] and it is still very popular method for registering point clouds. To briefly summaries algorithm: ICP iteratively refines position of two point clouds by optimizing the sum of square distances between corresponding pair of points from two clouds. This approach is usually called point-to-point registration. Class of algorithms based on ICP has developed many modifications. Surrvey of base type of ICP algorithms and their comparison on well designed datasets is in work of [PCSM13].

2.3.4 Correlative scan registration

Correlative scan registration is algorithm presented by [Ols09b]. This method was developed to robustly solve registration problem. It does not require any initial guess. Therefore, it is possible to use it for loop closure registration.

The algorithm requires two point clouds. Target point cloud is used for generation of fast look up table filled with bit values. It is created by separating points from target cloud into individual cells. Every cell which has some points in it is marked as occupied. After this step we have a table with value 1 in cells with some points and value 0 in cells without points. In next step we add sensor noise to the table. As a function of our noise we use radially symmetric kernel.

$$K_{i,j} = \exp\left(\frac{-1}{2}\left(\frac{\sqrt{(ir)^2 + (jr)^2}}{\sigma}\right)^2\right)\eta \tag{2.21}$$

$$K = \begin{pmatrix} 2 & 14 & 2 \\ 14 & 100 & 14 \\ 2 & 14 & 2 \end{pmatrix} \tag{2.22}$$

where $K_{i,j}$ is one element of kernel; $\sqrt{(ir)^2 + (jr)^2}$ is euclidean distance from center of the kernel to the element i, j with cell size parameter r. Standard deviation of sensor nose is abbreviated by σ and η is kernels max value.

The Kernel overlaps over every occupied cell in the table. If value of kernel is higher than value in table. Table is updated with the kernels value. Generated smoothing can be seen in figure **??**.

This algorithm is avoiding initial guess by trying all possible rotations and translations of source cloud. Every point of transformed source cloud is mapped into certain cell of look up table. The total score of transformed cloud is sum of all mapped cells scores. Algorithm usually tries rotations and translations from selected range. Transformation with the best score is the most probable transformation.

This brute force process might take long time if we select small cell size to achieve good registration. To speed up this process we first need to avoid computationally expensive calculation of goniometric functions in transformation. This can be achieved by first generating all possible rotations of point cloud. For each rotation we try all translations from selected range with step size selected based on cell size of look up table.

Real speed improvements offers usage of two layer architecture of look up tables. The first table has coarse resolution. This table is used for initial estimation

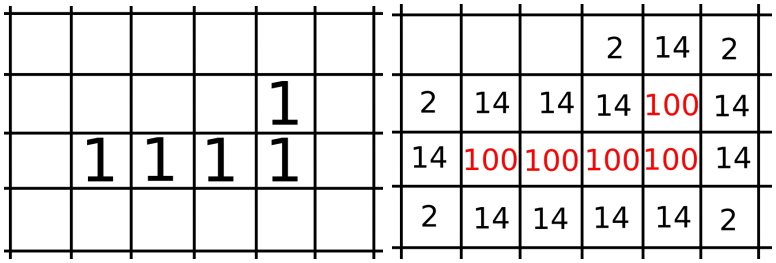

Figure 2.2: Image on the left shows look up table before applying smoothing kernel. Image on the right is after application of kernel.

on the whole range of selected rotations and translations. The transformations with best score are used in the second round. From every good transformation is generated search space voxel. Origin of voxel is taken from transformation. Size of voxel is cell size from coarse table. Search voxels are evaluated on look up table with fine resolution. Search space is this time limited to search voxel and initial transformation is taken from origin of voxel. The best result is our solution. By this process computation time drops rapidly as show in work of [Ols09b].

3. NDT graph-SLAM overview

In this chapter we will present our solution to 2D version of the graph based SLAM on the NDT maps. This chapter starts with complete overview of the algorithm. In the next sections we explain how each part of the system is designed.

3.1 System composition

The standard input of many SLAM algorithms is an odometry. In our case, we do not require any prior information about the robot movement. Our source of odometry is a fast incremental scan matching. The only mandatory input is a point cloud extracted from the robot's laser measurement. The scan matcher calculates relative transformation based on received point cloud and a map from previous iterations of the incremental scan matcher. We will call this map a moving window. Details are in the section 3.2.

The resulting transformation is used in the NDT frame creation process. The NDT frame is a small map which is created out of couple consecutive scans. A precise transformation is needed to merge these scans into a single frame. In our system, we use transformation from incremental scan matcher. The pose graph stores the NDT frame inside the node. The NDT frames integrate multiple scans to reduce the problem with the limited field of view. Each frame carries more information which gives a better outline of the world. More information about the world also helps to reduce a chance of ambiguous loop detections 1.4 because larger frames have a higher chance to include some unique features. Additionally, we also want to utilize advantages of NDT-OM occupancy update rule. It can detect dynamic objects with ray-tracing. The detection is done by merging multiple scans and re-observing the same cell multiple times. More information about design choices behind NDT frames is in the section 3.3.

The next phase of the algorithm creates a node in the pose graph when the NDT frame is created. An odometry edge connects two consecutive nodes. The odometry received from scan matching process was used to create NDT frames. Therefore, odometry edge has a transformation between origins of consecutive frames. In the next step, pose graph generates possible loop closure edges. The algorithm traverses a graph with Dijkstra projection and applies our radius based metric described in section 3.4.

The potential matches need to be registered and validated. It is the most difficult problem. We need an algorithm which can perform 10s of registration per second. At the same time, it needs to reject matches which are not from the same part of the environment. Some errors caused by local and global ambiguities 1.4 will not be avoided. We propose a solution to these problems by improving version of D2D-NDT. In this adaptation, we use a robust initial pose estimation from the correlative scan registration 2.3.4 and fine alignment from D2D-NDT 2.3.2. The full description is in 3.5.

The loop closure edges need to be validated against possible outliers caused by ambiguities. We have decided to use a robust optimization engine with switchable constraints. We have made a decision based on the comparative study by [SP13], where this method offered the best results. An important factor in optimization

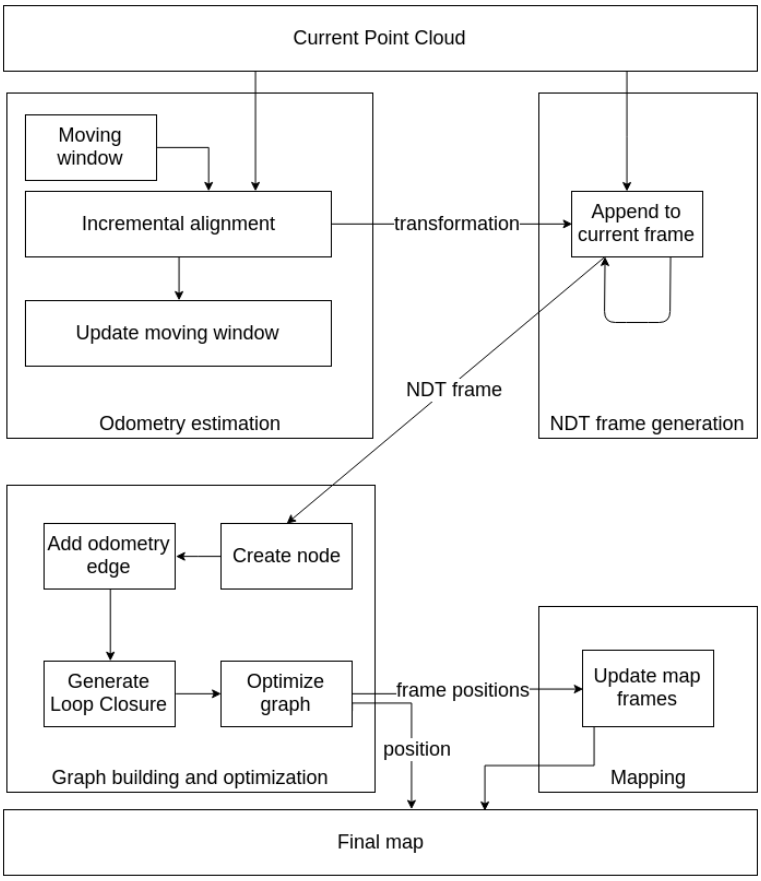

Figure 3.1: Diagram of graph based SLAM on NDT maps

process is a number of nodes and edges in the graph. The computation time grows with increasing number of elements in the graph. We limit the number of nodes by using NDT frames. Two consecutive frames can be farther away from each other because they represent a bigger part of the environment.

A Smaller number of nodes in the graph also mean less work to the NDT mapper. In the case of successful loop closure, we need to regenerate map based on the new position of nodes in the graph. In this version of the algorithm, we just iterate over all frames and merge them to the new map based on the new origins. In the future NDT frames allow to generate only a part of the map based on a request from a user. It could also be possible to load and save individual frames and save memory in the long run of the algorithm.

A combination of these parts together creates the graph-base SLAM on the NDT maps.

3.2 Moving window

A moving window is a special type of the NDT grid. It uses all features of the NDT-OM including the occupancy update and the dynamic object rejection. The main idea behind the moving window is to offer a small map which can be used by incremental scan matcher in order to efficiently align incoming scans against a longer history. In the standard incremental scan matching approach, we need to know the whole map. This map is then used to correct small errors when revisiting the same place again. A problem with this method arises when an alignment fails. In this case, the incorrectly aligned scan is merged into the map. It creates the same feature multiple times in the map. The next registration can use the wrong feature, and the error might never be corrected. The NDT-OM can solve some of the degeneration by cleaning occupied cells which are on the way between robot position and the new measurement. In order for this mechanism to work, the next scan needs to converge to the original correct position. This is very unlikely with a corrupted map and the standard NDT technics for registrations, which can end up in the local optimum during the scan refinement step.

In our system, we do not need to know the whole map of environment because loops closures errors are fixed by pose graph optimization. We need incremental scan matcher to provide us a good local estimate of the robot movement. For this purpose, we only need to know the part of the environment overlapped with a current scan. It strongly depends on the type of a sensor and an environment. In our setup, robot operates in the indoor environment with laser sensor ranging up to 20m in long corridors. In this scenario window size of 20m should incorporate all information which can help in scan registration. If it is possible to select smaller window size based on the structure of the environment, the algorithm may save time and memory.

The mowing window also needs to follow a movement of the robot to incorporate new measurements. It can be done in two ways. The first, we might rotate window based on exact changes of the robot global position. The algorithm must transform the whole window after each small movement of the robot. This approach tries to transform every single normal distribution inside grid in every step of the algorithm. Transformed cell's distribution may suddenly overlap multiple fields. We would need to develop a mechanism how to split original distributions into the multiple cells. A better way is to keep windows orientation fixed and only translate the window based on the robot's movement. It prevents a rotation of distributions in the cells but still, suffers from the same splitting problem. The final solution is to move the window only in multiples of the cell size. It does not affect parameters of normal distribution inside of original grid when the window moves multiple fields in any direction. After the movement, some cells may get out of the scope of the current window. These cells are destroyed which help to reduce accumulated error in the window.

To minimize any alignment issues we have decided to perform fast scan matching. We achieve it by processing as many laser scans as possible. The high-frequency scan matching does not need initial guess because a valid result is reasonably close to the initial position of the source and the target measurements. The registration algorithm which is capable of this performance needs to work in order of milliseconds. Two algorithms developed for fine registration on top of

the NDT grid are the P2D-NDT and the D2D-NDT. We use the standard D2D algorithm because it offers ten times better run time than P2D. A comparative study by [MVS+15] shows that even though P2D is usually more precise, it needs significantly more computation time. Skipping multiple measurements from the sensor may cause that we will not be able to estimate robustly transformation and the whole process can converge to a local minimum.

Algorithm 3 Moving window processing loop

Require: point cloud X, move window's NDT grid M, transformation T_o to the origin of moving window, transformation T_r unused from move in last call of function, transformation P last known absolute pose of moving window.

1: **function** CALCULATETRANSFORM(X)
2: $X_o \leftarrow$ transformPointCloud(X,$T_o * T_r$)
3: $N_o \leftarrow$ createNDTGrid(X_o)
4: $T_{ox} \leftarrow$ alignD2D(N_o,M)
5: M.mergeIn(X_o,T_{ox}) ▷ applies transformation on point cloud and merge it into moving window
6: $T_{diff} \leftarrow P^{-1} * (T_o * T_{ox})$
7: $P \leftarrow T_o * T_{ox}$ ▷ update of absolute pose of window for next call
8: $T_r \leftarrow M$.moveWindow(T_{ox})
9: **return** T_{diff}
10: **end function**

add visualization of running window

3.3 NDT frame creation

The NDT frame is created by merging multiple point clouds based on transformation received from odometry estimation. The important question is how many scans should we combine? This algorithm uses consecutive addition of transformation as in equation 2.3. Afterward, it calculates a total displacement done by a robot. If it is more than a threshold we close down the old NDT frame and start to add scans into the new empty frame. The new frame is assigned its coordinate system based on current robot position. Every new scan is transformed into the coordinate system of currently opened frame and merged in. The closed frame is sent to the pose graph generation where it is transformed into the node.

A Selection of good displacement parameter is important for a run of the algorithm. A small value will create many nodes in the pose graph. Every node will reflect an only small portion of the environment. This will make loop closure computationally expensive by a need to evaluate too many possible loop closure nodes. At the same time, loop closing algorithm will work with only limited information. This may cause a bigger number of local and global ambiguities in registration. A large value of displacement will generate fewer nodes with more information in each node. This is less computationally dependent. On the other hand, it creates an ambiguous environment inside of the NDT frame. The loop closure registration may not correctly deduce which part of the same environment in the frame is correct for registration. The registration algorithm is forced to identify this situation and solve it. At the same, it wastes an optimizer's potential in ambiguity rejection based on topological information of the whole environment.

3.4 Loop closure detection

A loop closure detection is done on top of the pose graph. The loop detector can use current positions of the graph nodes and relative transformations stored in the odometry and loop closure edges. With this information, we need to find all nodes which can with current node create a loop closure edge. The process starts by Dijkstra projection mentioned in the section 2.1.2 from the current node. A part of the projection is also a calculation of the relative displacement along the edge. The sum of displacements is used as a parameter for rejection of nodes which are too close to our current position. These nodes are certainly overlapping with our start node and therefore it is not necessary to check them again. All the nodes passing the previous test are used in one of two rejection models.

The first model tests all nodes against selected radius. The second mechanism is using cumulative transformation and covariance calculated by Dijkstra projection. In validating if two nodes overlap we use same metric as presented by [Ols09a].

$$\Delta c = (c_b - c_a) \tag{3.1}$$

$$s = \max(0, \|\Delta c\| - r_a - r_b)\frac{\Delta c}{\|\Delta c\|} \tag{3.2}$$

$$mahl = s^T P_{a,b}^{-1} s \tag{3.3}$$

where c_a and c_b are the centroids of start and currently compared NDT grids; r_a and r_b are radii of the respective NDT grids and $P_{a,b}^{-1}$ is an inverse of the

accumulated covariance.

The selected nodes are registered by robust D2D. Those matches with high score are inserted into the graph. The edges added by this mechanism may still include some errors or ambiguities. Rejection of these edges is done in the optimizer.

3.5 Robust D2D-NDT registration

Construction of a robust D2D registration needs to be fast and precise. Also, it needs to have a mechanism how to reject invalid association. It can use only information present in NDT grids because a loop closure mechanism is working only with this data. We knew that the D2D offers quick and reliable registration on the NDT grids with a good initial guess. The correlative scan matching algorithm 2.3.4 can provide registration without a knowledge of the initial guess. Unfortunately, in the standard version, it is not possible to operate with NDT grids. The performance of this algorithm is also slower than D2D. To solve these problems we have developed modified version of the correlative algorithm which can work on top of NDT grids.

3.5.1 Adaptation of correlative registration

We have started with the base algorithm described in the section 2.3.4. It is sufficient for our needs when this algorithm provides only a rough initial guess. For this reason, we use only one layer architecture. Our single grid has double cell size in comparison with the original size of the NDT grid. It offers faster execution time. In the first part of the algorithm, we need to go over large search space because we cannot expect any prior information from the graph. Larger grid size limits the number of translation because we always try translations in multiple of the cell size as mentioned in the 2.3.4.

Secondly, we need to transfer original NDT grid into a reasonable point cloud. In our implementation, we have decided to recreate point cloud out of grid by taking a mean from every cell with distribution. A collection of these means makes our mean cloud. In addition, we use information about how many points were used to create a normal distribution. This information is used in our algorithm as a weight for every mean value. Original algorithm uses two point clouds.

First is called target cloud and is used for the creation of look up table. This table is created by projecting all points to individual cells. When is a cell occupied by at least one point it is marked with value 1. In our scenario, we use a cloud of means from the target grid to construct a look-up table. Use of means is more robust to outliers than original look-up table from a point cloud. The original implementation marked every cell occupied regardless on the number of points mapped into it. Our grids need at least 4 points to create a normal distribution. This limits an influence of the single point spread in a space and also emphasizing dominant structures in the environment. Target grid conversion to mean cloud does not loose any information in comparison to the original cloud. This is because the look-up table and the target grid are aligned. On top of that double step size of the look-up table makes four cells from the target NDT contribute to a single look-up cell.

The second source cloud is used for scoring in the look-up table. Every point of a point-cloud contributes to total score based on the value from the look up cell it belongs to. In our case, the single point represents information about the mean center of multiple points. In order to keep all information, algorithm maps mean into the correct cell in the look-up table. By doing this, the mean only contributes once. Fortunately, a score generated by mean can be scaled with the use of weight associated with the mean. This makes a weighted mean point contribute the same amount to the system as standard points from point cloud. The score function is defined as:

$$score(T, C) = \frac{1}{d} \sum_{p \in C} v(T, p) w(p) \tag{3.4}$$

where T is transformation which should be applied to point p of cloud C. A function $v(T, p)$ applies transformation T to point p maps it to look up table and return score value for single point. A function $w(p)$ return weight of current mean point. Scaling factor d is defined as

$$d = m \sum_{p \in C_t} w(p) \tag{3.5}$$

where m is the maximal value one point can receive from look up table after application of smoothing kernel in equation 2.21; C_t represents target point cloud.

The last problem with conversion of source cloud to mean cloud is to handle discretization errors. These errors happen when we need to transform NDT grid. In this situation, one original PDF may overlap multiple cells. The original point cloud would contribute into multiple cells. Our mean formulation would contribute only to one cell based on mean location. To minimize this effect, we map every mean value into the target look-up table which has double cell size in comparison with source NDT. This process is similar to multi-layer discretization removal in multi-layer NDT registration [UT11]. The Target look up table also include a smoothing kernel, which assigns some value to cells surrounding occupied cell in the table. This also makes mean which could potentially slip out of occupied cell's boundaries contribute to the total score.

By executing these approximations, we were able to create a version of the correlative registration on top of NDT grids. Coarser resolution improved performance and allowed us to search larger search space. Approximation of the input cloud into means reduce the number of a point we need to test in every iteration of the algorithm loop. This effectively lowered number of calls to the look-up table, which speeds up the whole process. In addition, mean cloud removes outliers from the points spread in space.

3.5.2 Algorithm overview

With the coarse initial guess estimate, we can construct the algorithm. The first step is to run correlative estimation algorithm on a pair of grids. The result is the best initial guess it could find in the selected search space. The correlative estimator uses a coarse look-up table which means that grid still needs to be transformed up to two NDT cell. The next step is to run the D2D algorithm.

The multi-layer definition of the D2D can converge to the right solution if there is one. The problem arises if two matched grids are from different locations and do not share same environment features, e.g., lines, corners. In this case, correlation registration finds the best possible solution, which means that it rotates grid in a way that maximalizes a score. The D2D than try to find the best alignment and usually falls to the first local minimum it can find. To solve these situations we propose solution validation process. Example of bad alignment is in figure 3.2.

3.5.3 Solution validation

Robust alignment offers us the best transformation between the source and the target NDT grid. This alignment can fail and not provide a successful registration at all. We need to validate if this registration succeeded or failed. In this algorithm, we again use correlative scan matcher. In this case, we use a cell size of the target look-up table matching the cell size of the NDT target grid. We map every point from mean source cloud into a look-up table and receive a total score based on contributions of each weighted mean point. In this case, discretization is helping us to provide better results. Some means may stay out of the target grid this means that registration was less successful which result in a lower score. This method can reject scans based on their overlap. It is not able to distinguish the wrong alignment in case that two scans look similar but originate in two different parts of the environment. This ambiguity is resolved in the graph.

Algorithm 4 Robust D2D registration algorithm

Require: source NDT grid G_s and target NDT grid G_t. Resolution of NDT grids r. Validation threshold v

1: **function** ALIGN(G_s, G_t, r)
2: transformation T is identity
3: (T,$score$) ← correlativeEstimater(G_s, G_t,T, $2 * r$)
4: T ← alignD2D(G_s, G_t,T)
5: (T,$score$) ← correlativeEstimater(G_s, G_t,T, r)
6: **if** $score \geq v$ **then**
7: **return** (T,$true$)
8: **else**
9: **return** (T,$false$)
10: **end if**
11: **return** T
12: **end function**

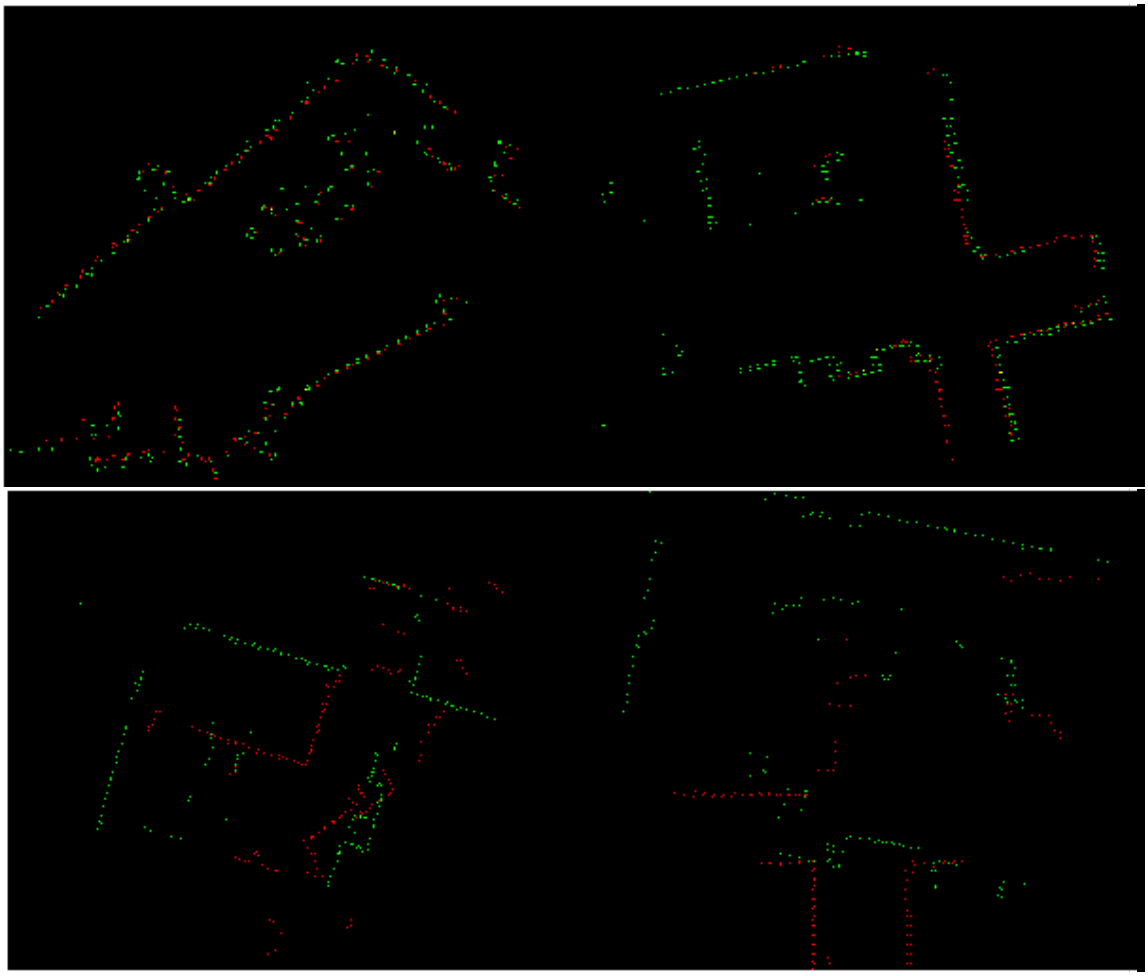

Figure 3.2: Images with alignment. Red dost represent target scan and green dots source scan. The first row shows valid alignment marked with high score. Second row shows two alignments which were rejected by validation.

4. Implementation

In this chapter, we will present implementation details of our system. First, we present all libraries used in this project. Later we will introduce outcome in the form of the ROS package. Also, we will briefly present the structure of the program and key components.

4.1 Used libraries

4.1.1 ROS

The ROS [QCG+09] is a popular robotic framework. It offers a flexible way how to combine existing tools, libraries, and algorithms to make a full robotic solution from drivers up to the higher logic of planning and mapping. The communication between individual programs (nodes) is done through the subscriber-publisher model. A configuration of programs is stored in the parameter server. This server also takes care of managing communication between nodes.

4.1.2 The Point Cloud library

The Point Cloud Library (PCL) is a standard ROS library for manipulation with point clouds [RC11]. The library includes state-of-the-art algorithms in registration, filtering, segmentation, and feature extraction. It also contains tools for visualization and manipulation with point clouds. In our project, we use mostly point cloud class which is the most basic data structure in the library. We also use registration base class for implementation of our scan matching algorithms.

4.1.3 The G2O

The G2O is a pose graph optimization library presented by [KGS+11]. It is currently the most used library for the pose graph optimization. It offers well designed extendable interface which makes it easy to add a new definition of pose graph optimization. New optimization methods often have an implementation for this library. In our program, it is used as main optimization engine for our pose graph.

4.1.4 The Eigen

The Eigen [GJ+10] is a templated C++ library for linear algebra. It includes modules for dense and sparse matrix representations, numerical solvers and transformation representation. This project mostly uses geometry module with affine transformation. We also utilize numerical solvers in our implementation of registration algorithms. We have selected this library because it is considered a standard library for linear algebra in the ROS. Many packages use it and offer API's designed with this library.

4.2 Structure of the implementation

The architecture of the whole system can be divided into three parts. The first part is theROS interface. In our implementation, this interface expects only laser scanner data. However, it is also possible to provide odometry information. The interface uses standard names for topics. This interface includes all inputs and outputs which can be found in other SLAM packages. Additionally, it provides a map in the form of a point cloud. The full documentation of this interface is in Appendix A.

The second part is the SLAM algorithm interface implemented in C++. It offers the same functionality as the ROS interface. We have decided to have this double interface because it is convenient to use our SLAM also without the ROS subscribe-publish interface. It was mainly used for debugging and testing purposes. This interface also offers some flexibility if we decide to do a different version of our algorithm. In this case, we do not have to rewrite node's source code. In this layer of abstraction, we take care of an initial estimation of the odometry and the NDT frame building process. A map generation also takes place in this part of the architecture.

Thirds part is graph SLAM interface. This interface makes abstraction around graph creation and optimization process. This section is using our custom pose graph implementation. On top of this graph, we developed a loop closure detection and validation. This graph is synchronized with the graph inside of optimization engine G2O. We carry two graphs for the reason of easier switching between different optimization engines in the future. Our graph representation also includes additional information about state and type of the edge. Implementing it into G2O would require rewriting this code with every new optimizer and with every new G2O edge and vertex type.

An important part of the architecture is handling of NDT frames. A created frame is stored inside shared pointer. The same pattern is used in PCL's point cloud data type. The shared pointer is then passed to the graph creation process and also to the NDT map building process. Nodes of the pose graph include this pointer as their representation of the world. The NDT frames in nodes are used for loop closure registration. This means that registration algorithms use the shared pointers in their API as well. This approach is also a standard for registration algorithm in the PCL library

4.2.1 NDTGrid2D

The NDTGrid2D is the main class for all operations in our approach. It offers basic functionality for grid creation. It can be merged with or without a use of ray-tracing (occupancy update). It is used for dynamic entity update from NDT-OM. It also offers grid translation which is needed for the moving window implementation. Another group of functionality is for registration algorithms. They require radius search and k-nearest neighbor search. The odometry estimator also needs to use means from cells. The last group is output format methods. Grid can create a coarser instance of itself. It is also able to be printed to standard console output. We have implemented methods for conversion into our custom type of occupancy and NDT map messages. These messages are used

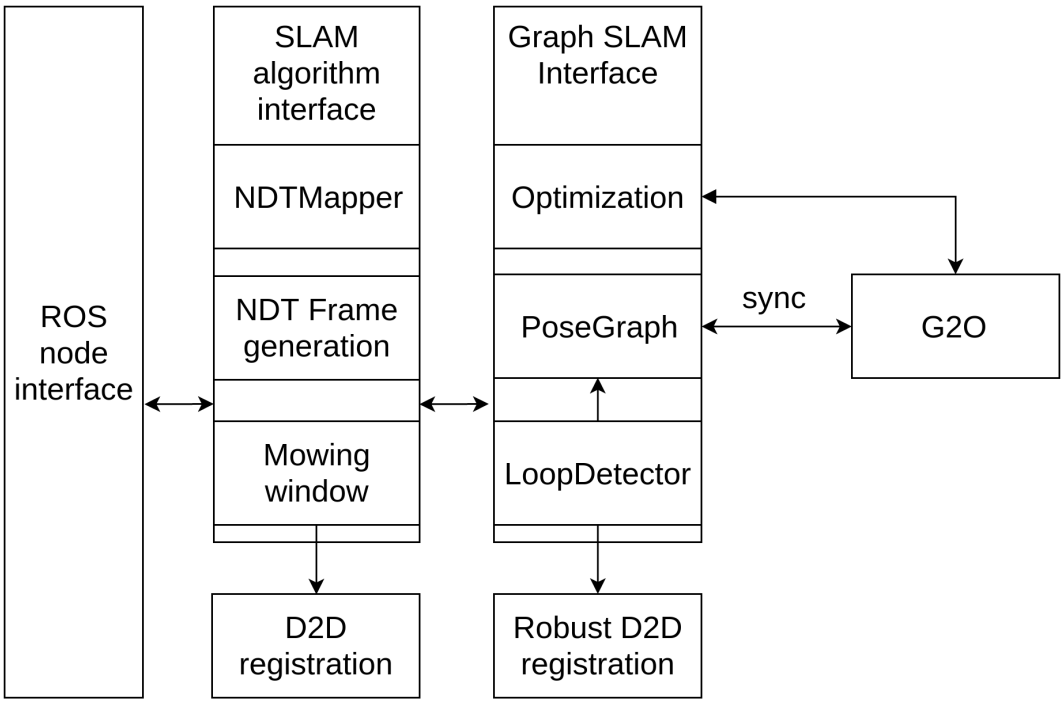

Figure 4.1: Overview of individual parts of architecture and their relationships.

only internally and can be transformed into ROS variants.

In order to fulfill all these needs implementation of NDTGrid2D is just a higher abstraction layer on top of the VoxelGrid2D. The voxel grid is taking care of memory layout, resizing, element lookup and ray-tracing. The NDTGrid2D has two template parameters. The first parameter is the type of the cell. Grid is initialized from a point cloud, for this reason, it needs to have second template parameter representing a type of the point. The second parameter is standardly used in PCL related algorithms.

Core algorithm logic for merging and updating cells is stored in every cell. This allows developing new cells without any changes to the grid.

4.2.2 VoxelGrid2D

It is a generic grid-like structure with one template parameter. The type used in the template is required to have implemented operator plus and copy assignment. This data structure is intended to use with larger cell types in the sparse environment. Based on these requirements we designed the memorry model. Grid is represented by single vector which holds pointers to cells. In the case of the unoccupied cell, it uses null pointers. The grid is initialized empty with no cells inside. It allows dynamic resizing either manual or automatic based on inputted cells. It offers base functionality for ray-tracing and radius search.

4.2.3 NDTCell

The NDT cell is the core of all calculations on top of the grid. In case of NDT-OM implementation it holds covariance and mean estimation, occupancy update rule and RCU update rule for merging of cells with Gaussian inside. In the future

experiments we can easily design a new type of the cell with different calculation model and keep NDTGrid2D and the VoxelGrid without modifications.

4.2.4 Registration algorithms

When designing registration algorithms we have decided to use same interface as PCL's registration algorithms. By extension of their base class our programs can be used standard way inside of PCL. This makes it easy to use our algorithms alongside PCL implementations. It also possible to use all visualization and io tools provided by PCL. Our algorithms have option to run in multiple threads which boost their performance on multi-core processors.

5. Evaluation of NDT graph based SLAM

In this chapter we will demonstrate functionality of our algorithm. We compare it with two well known SLAM approaches implemented in ROS. We also explain what parameters are needed to fully use potential of this algorithm. Whole experiment is conducted by running prerecorded data files from PR2 robot operating in Massachusetts Institute of Technology (MIT) Sata Center [FJKL13]. We have selected this dataset because laser scanner provides sufficient number of points to produce NDT fields with normal distribution. It is also recorded in form of "bag file" which is standard format in ROS. It offers very challenging situations for robust testing. It is not uncommon that algorithms fail on many of recorded data sets. The problem is even more difficult when using only 2D laser data information.

5.1 MIT dataset details

This dataset offers fine laser data with 1130 points per scan. The sensor's field of view is 260 degrees.The 2D laser scans have maximum range of 60 m with publishing frequency around 20 Hz. The dataset was recorded on multiple floors of the Sata Center. This means that robot enters an elevator and is transported to different floors. Therefore, we have selected only data sets which stay on the same floor. Our experiments were conducted only on the second floor because it has information about ground truth.

We have selected two datasets which we think showcase well possibilities of this algorithm. First dataset runs in small loop inside of one room. This is challenging because robot needs to correct its position multiple time. It is also computationally difficult for loop closure mechanism. In every iteration it needs to compare its position with all previous measured data. Iterative movement at the same place can also demonstrate robustness of incremental scan matcher again any shift caused by registration error.

Second dataset starts in the long corridor and moves in direction towards room from the first selected dataset. It again do multiple loops and then it moves through corridor to new room. It maps this room and returns through the same corridor. In this type of setup odometry information or incremental scan matching can accumulate error over long corridor and last room, which should be visible on the returning trip. This tests loop closure mechanism over long distance.

In the figure 5.1 is a map of used datasets built with NDT-OM with transformation between scan originating in ground truth measurements provided with dataset.

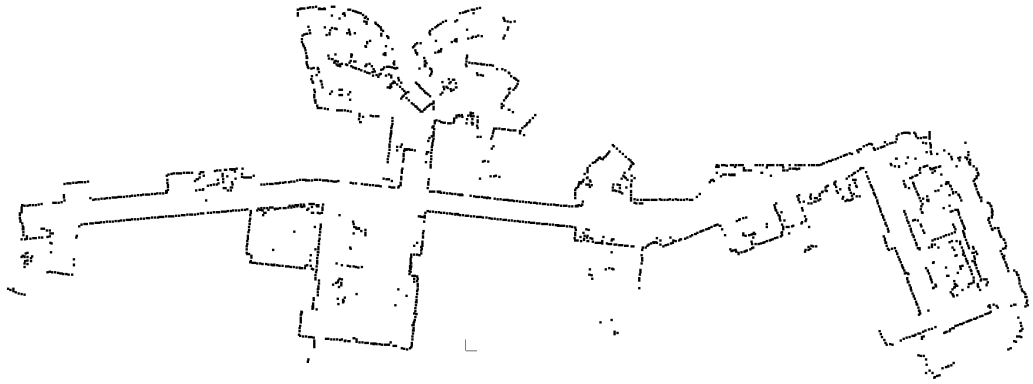

Figure 5.1: The NDT map generated out of selected datasets with use of ground truth odometry information.

5.2 ROS SLAM algorithm overview

5.2.1 The Gmapping

The Gmapping proposed by [GSB07b] is the popular SLAM algorithm in ROS. It is based on Rao-Blackwellized particle filters. In this algorithm each particle is carrying representation of the map. In this set up number of particles is rapidly increasing memory requirements. For this reason authors used not only odometry to estimate robot movement but also the most recent measurements. This reduces number of samples by providing better estimate of robot's movement. The Gmapping was used in dozens of projects in ROS. It is usually first SLAM used by every novice user in ROS. It is well documented and tested. Output representation is in form of occupancy grid with fine resolution 0.05m.

5.2.2 The Hector SLAM

The Hector SLAM proposed by [KMvSK11] uses fast and robust scan matching to estimate robots movement and build map. The algorithm also do not use any odometry which makes it ideal for aerial robots. Registration is done by optimalization of laser end points with map built in previous iterations. Registration equation is solved using a Gaussian-Newton minimization method. This process returns transformation between laser endpoints with resulting map. This method may converge into local minimum. Authors prevent this by using multiple grids each with coarser resolution. This method operate on top of occupancy grid with fine resolution around 0.05m.

In comparison to our method it is very similar to our front end odometry estimator. Our method of running window uses fast incremental scan matching. It also uses several layers to avoid local minimum. The biggest difference is used underling map model. In our algorithm we use map with coarse cell size 0.25m with PDF inside. In addition, we also have loop closure engine with pose graph which should resolve more difficult localization errors.

5.3 NDT-GraphSLAM evaluation

In all our experiments we have used our SLAM algorithms as was described in section 3.1. We have decided to set moving window size to max range of the sensor which is 60 m in our dataset. We have also set fixed values for radius search for loop detection to 20m. We selected this value so we can test as many loop closures as possible. During our mapping and localization test we also record all loop closure measurements. These are saved to the disk as point cloud file (.pcd). We also save results of loop closure registration with resulting score for evaluation of loop closure algorithm based on changing parameters. Files from experiments are available in an attachment of this work. Initial value for loop closure registration threshold was set to 0.6. Every loop registration with score higher than this value will be inserted into the graph as loop closure edge.

Output from our method is in point clouds. Each point represents position of mean value inside of the cell. Resolution of this map is same as for all NDT grids (0.25m). Our representation is different in comparison to output methods of Gmapping and hector mapping which use occupancy map. Representation of the output map does not change characteristics of reconstructed maps. It is important that map has correct shape. It is also important that empty places like hallways or centers of the rooms stay unoccupied with as little noise as possible. Result should not include any phantom walls. These are walls present on the map but they do not exist in reality. They are usually caused by wrong pose estimation. In our representation we also output pose graph visualization which is only for debugging and demonstration purposes.

5.3.1 NDT frame generation frequency

In this experiment we wanted to test what is optimal euclidean distance between two consecutive NDT frames. We use same representation of frames as mentioned in section 3.3. Based on design of the system this parameter should influence quality loop closure detection and evaluation. In order to test this parameter we have decided to test it on second dataset with distances 1m 2m and 4 meters.

One meter range has generated frame every 1 meter of robot's trajectory. This has created high amount of nodes with small map representation of environment inside. For mapping purposes this created nice map because there was small odometry error inside of frame. Error may be caused by wrong odometry estimate from moving window. This can be observed in the first picture of figure 5.2. Its also important to note that it has generated the biggest amount of loop closure edges. This mainly thanks to fact that it is easier for two frames get high overlapping score from robust D2D if they have no errors inside. On the other hand it is more probable that these scans will have problems with ambiguities. This has happened in total 4 times in the second dataset with registration threshold set to 0.6.

The second variant with two meters long distance between frames offered optimal results. It has generated fewer nodes than the first variant. Loop closure edges added to the graph were able to repair errors from odometry estimation and still keep same quality of map.

The third variant failed to find loop closures. Every frame had data with

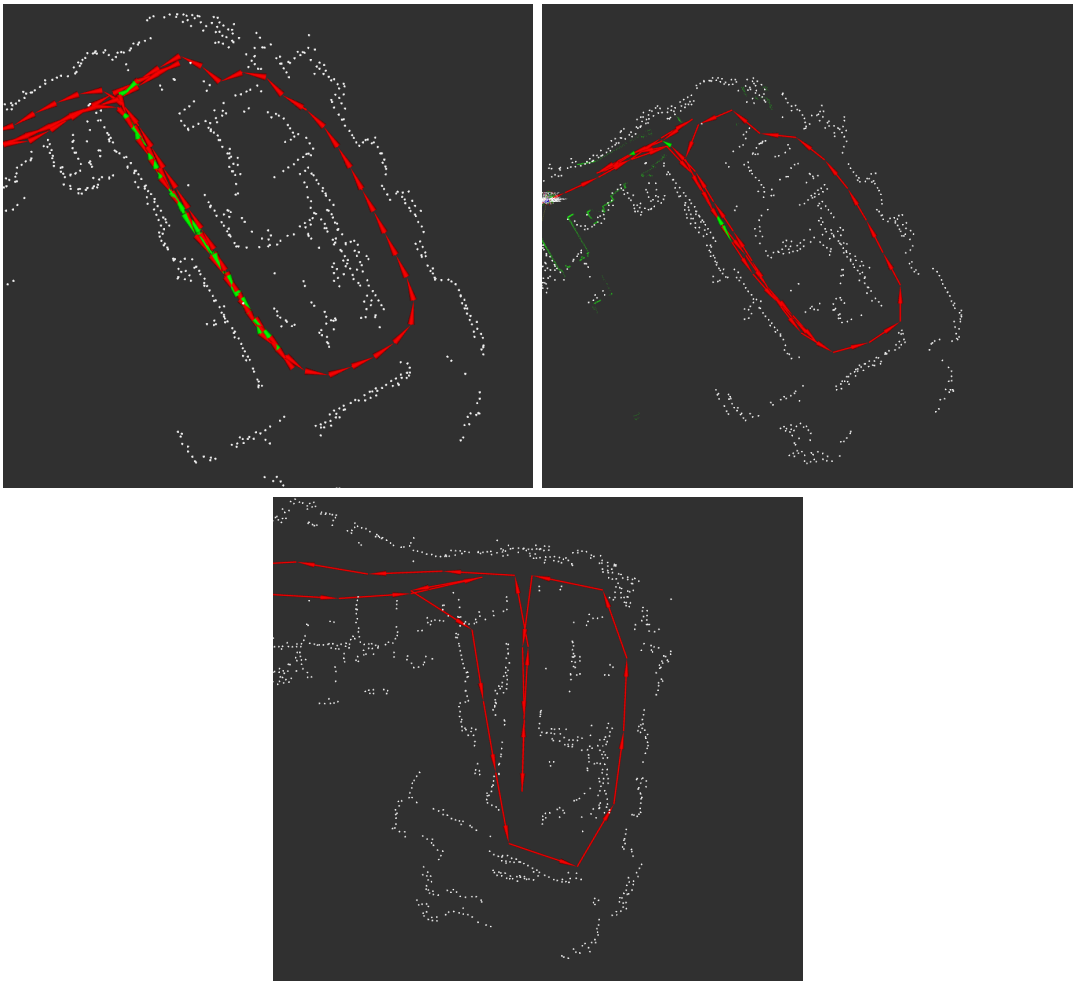

Figure 5.2: Comparison of effect of different frequency of frame creation. From left pictures of 1m, 2m and 4 meter distance between frames. Red arrows represent odometry edges, green edges are loop closure edges, white dots represent point cloud of means from each cell.

heavy noise inside. This caused that non of the loop closing tests received score more than 0.1.

Based on these results we have decided to use fixed distance of 2 meters between two frames in the next tests.

5.3.2 Robust D2D score threshold

In a previous section we have use fixed registration threshold to value 0.6. In this section we will test if this is really optimal value. This test is executed by first setting threshold to value 0.6. Then running second dataset. All measurement data received from all loop closure registration are sorted based on score value into groups. The first group has score range from 0.4 to 0.49. The second group from 0.5 to 0.59. The third group starts at 0.6 and ends in 0.69. Last group include all loop closures with higher value. We will look at number of edges in each category which are have wrong alignment. These edges are usually created on failure of algorithm to register correctly frames. The other reason might be ambiguity in the environment. We want to minimize number of incorrect edges

	correct	error	total
[0.7,1]	21	0	21
[0.6, 0.7)	33	1	34
[0.5, 0.6)]	18	26	44
[0.4, 0.5)]	13	28	31

Figure 5.3: Number of correct and incorrect registrations in score groups

in our graph. Result of this experiment is in 5.3.

Based on result we can conclude that algorithm is able to securely identify valid loop closure on this dataset if score is above 0.6. One error in this category was caused by ambiguity in long corridor. This error is not possible to correct by usual registration algorithms. 2.3 Other two categories equally include more errors than correct results. Errors can be divided into two types. Some incorrect registrations are caused by matching unrelated places. These places are different but it is possible to match them in a way which yields good score. Score assigned by matcher is usually lower than 0.55. Some errors are also registration failures. In this type of error mostly depends on the structure ambiguity inside of the frame. This happens if two frames include same area but each has dominant number of cells mapping different feature of environment. This ambiguity makes robust estimator connect these two parts. This increases score in these likely part to higher levels. On the other hand parts of the frames not matching each other lower the score. As a result score of these errors ranges is in whole range from 0.4 up to 0.6.

Based on this experiment we can set to 0.6 or higher and get enough high quality loop closures to fully correct graph.

5.3.3 Iterative mapping of room

This dataset represents single room. In order to fully map it robot moved multiple time around the room. Every movement carries some error. It is necessary to correctly align consecutive scans. This well demonstrates coordination of moving window with loop closure mechanism. Whole room has fitted inside the moving window and registration provided robust transformation for NDT frame building process. Loop closures were correctly identified all above threshold 0.6. Distance between frames is 2m as discussed previously. Map is compact and without any defects.

Hector mapping has not converged into correct output. It was not able to cope with rotations of the robot in this dataset. We have also tested slowing down dataset with rate 0.6. This has not helped to hector recover correct data.

The Gmapping offers solution with similar quality to our result.

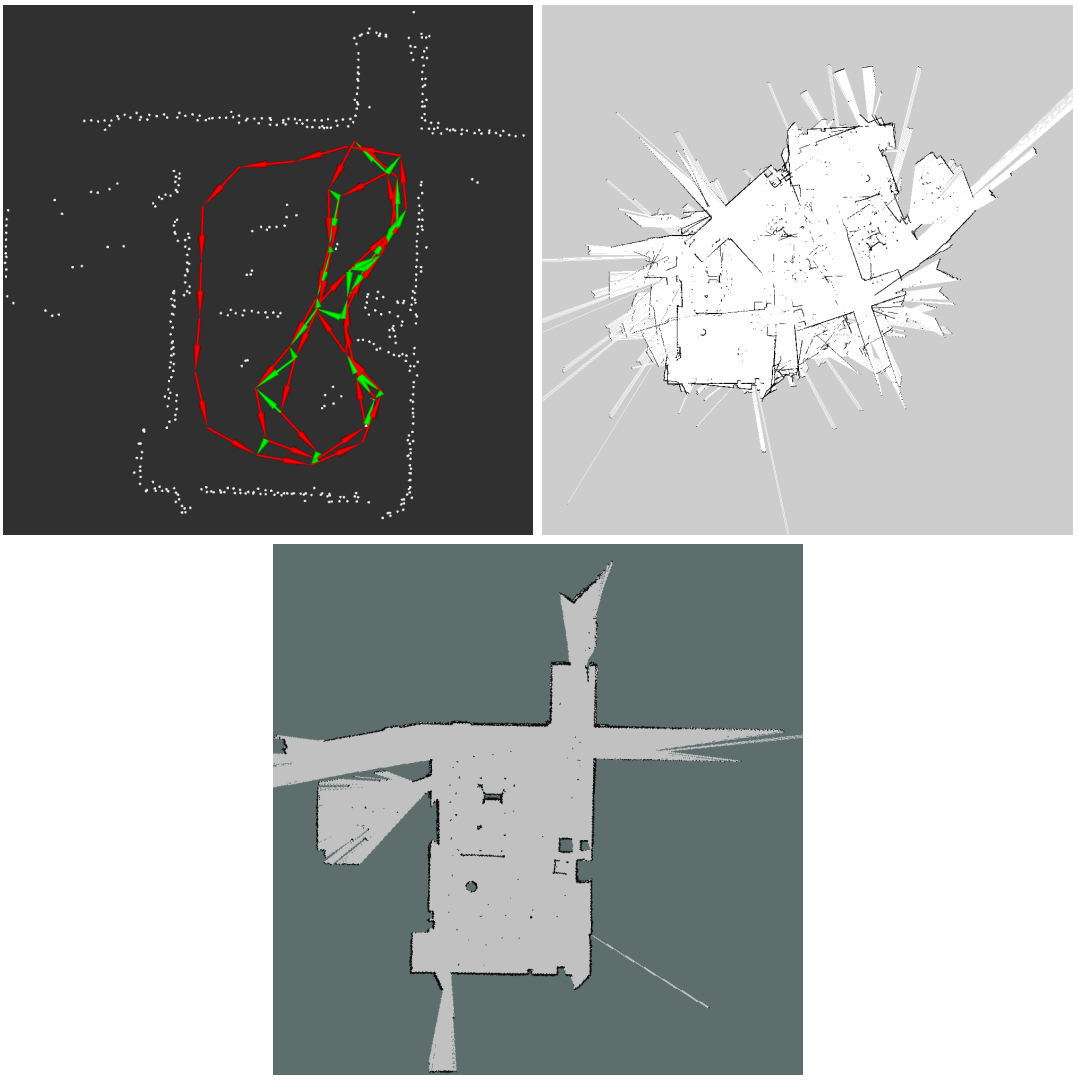

Figure 5.4: Map of the room mapped by our NDT aproach, Hector mapper and Gmapping. Visualization of our aproach includes visualization of pose graph with red odometric edges and green loop closure edges. White spots represent means of each cell's normal distribution.

5.3.4 Long corridors

Long corridor dataset was selected because it maps two main rooms plus it adds mapping of top part of the map. This part with its irregular shapes proven to be very difficult mostly for hector mapping. It has failed mapping process as you can see on the middle image in figure 5.5. The Gmapping algorithm offered accurate result.

Our approach has recovered main shape of the map correctly. Small difference is in noisiness of the walls. Our algorithm has higher noise. Main reason is different mapping model. The Gmapping and hector mapping are both using extremely fine map with resolution 0.05m. Our approach is using coarser 0.25m grid for visualization. Our map is coarser but still covers and represent free space correctly without noise. Coarser grid has also advantage in path planning or ray-tracing which is faster. Finer grids often needs to be converted into lower resolutions to work with them efficiently.

Second difference is length of the corridors. Our approach has shortened its length. The main reason is data alignment ambiguity. Robot passing through these corridors do not see the end of hall. This makes him observe only two straight walls on the right and on the left. Without prior information about robot movement, this is correctly understood as robot standing still without movement. The way our algorithm deals with this type of errors is by closing loop closure when returning on the same place through different path. In this case robot used same trajectory, which lead to same error only in the opposite direction. The only other solution how to sove this problem is to integrate movement of robot into moving window incremental scan matching. Resulting cooridor than look like in figure 5.6.

Figure 5.5: Map of the corridors mapped by our NDT approach, hector mapper and Gmapping. Visualization of our approach includes visualization of pose graph with red odometric edges and green loop closure edges. White spots represent means of each cell's normal distribution.

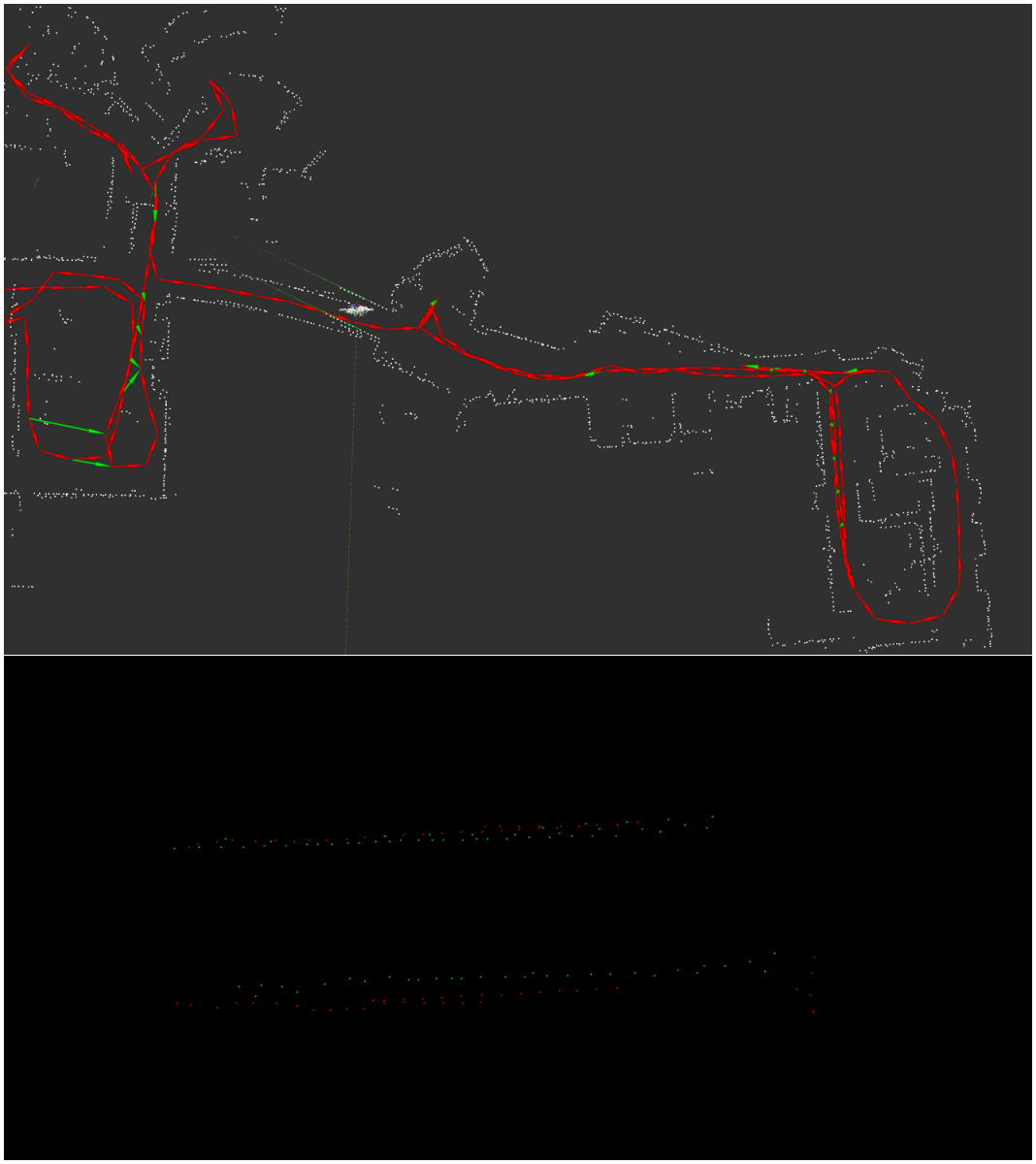

Figure 5.6: Top picture shows portion of map with corridor when incremental scan matcher may use initial guess from odometry. Bottom picture shows ambiguous registration in corridor area.

6. Future works

This work has focused on 2D graph based SLAM on NDT maps. In the future work it should be extended into 3D space. Performance of NDT mapping and registration depends on amount of data which can be inputted into cell creation. Therefore, couple layers from 3D laser point cloud merged together could represent better 2D information for registration.

Different methods for loop closure registration should tested in the future. One of the option is creation of feature descriptor and utilizing well know registration algorithms used in computer vision. This work may be also expanded by fusion of 3D vision color information with 3D point cloud. This would add more information to individual cells. It could increase precision and convergence time in 3D case.

This work is possible to extend on the side of pose graph as well. The graph can be improved by fusion of similar nodes. This would allow using this algorithm over long period without increase on memory usage. Another improvement could implement multi-layer graph representation for mapping purposes of multiple floors.

Conclusion

NDT graph-based SLAM algorithm presented in section 3.1 can reliably solve robot localization problem as well as create map representation of the world. The algorithm is suitable for use on robotic systems equipped with a 2D laser scanner. The algorithm does not require odometry information. Therefore, it is particularly useful for robots lacking odometry sensors (e.g. drones).

The whole process starting with parsing of input data and ending with providing location and map can run in an online matter. The combination of NDT scan matcher for fast odometry estimation and pose graph map optimization proved to be a good combination. While incremental scan matching was not able to create correct map in challenging environment because pose errors were too large, correct generation of loop closure allowed for valid map creation and was able to prevent introducing scan matching errors into the map.

The proposed solution of loop closure validation can correctly identify sufficient number of loop closing constrains. It also offers fast processing time [1].

The algorithm is implemented as ROS package ndt_gslam. It uses similar interface to other SLAM algorithms in ROS, threfore it can be used as their replacement with no additional effort needed. On top of that, it offers the maps also in the form of point clouds. All registration algorithms were implemented with the use of standard PCL APIs which makes them viable option for the use in the PCL ecosystem.

[1] in our installation on oridinary laptop with only two-core processor, we processed up to 50 loop closures per second

Bibliography

[BS03] P. Biber and W. Strasser. The normal distributions transform: a new approach to laser scan matching. In *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, volume 3, pages 2743–2748 vol.3, Oct 2003.

[CM92] Yang Chen and Gérard Medioni. Object modelling by registration of multiple range images. *Image and vision computing*, 10(3):145–155, 1992.

[FJKL13] Maurice Fallon, Hordur Johannsson, Michael Kaess, and John J Leonard. The mit stata center dataset. *The International Journal of Robotics Research*, 32(14):1695–1699, 2013.

[GJ+10] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[GKSB10] Giorgio Grisetti, Rainer Kummerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, 2010.

[GSB07a] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *Trans. Rob.*, 23(1):34–46, February 2007.

[GSB07b] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, Feb 2007.

[HD07] S. Huang and G. Dissanayake. Convergence and consistency analysis for extended kalman filter based slam. *IEEE Transactions on Robotics*, 23(5):1036–1049, Oct 2007.

[KGS+11] R. Kuemmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A general framework for graph optimization. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3607–3613, Shanghai, China, May 2011.

[KMvSK11] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, November 2011.

[LM97] Feng Lu and Evangelos Milios. Globally consistent range scan alignment for environment mapping. *Autonomous robots*, 4(4):333–349, 1997.

[Mag09] Martin Magnusson. The three-dimensional normal-distributions transform: an efficient representation for registration, surface analysis, and loop detection. 2009.

[MTKW02] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. Fastslam: A factored solution to the simultaneous localization and mapping problem. In *In Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 593–598. AAAI, 2002.

[MVS+15] M. Magnusson, N. Vaskevicius, T. Stoyanov, K. Pathak, and A. Birk. Beyond points: Evaluating recent 3d scan-matching algorithms. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3631–3637, May 2015.

[Ols09a] Edwin Olson. Recognizing places using spectrally clustered local matches. *Robot. Auton. Syst.*, 57(12):1157–1172, December 2009.

[Ols09b] Edwin B Olson. Real-time correlative scan matching. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 4387–4393. IEEE, 2009.

[OLT06] Edwin Olson, John Leonard, and Seth Teller. Fast iterative alignment of pose graphs with poor initial estimates. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2262–2269. IEEE, 2006.

[PCSM13] François Pomerleau, Francis Colas, Roland Siegwart, and Stéphane Magnenat. Comparing icp variants on real-world data sets. *Autonomous Robots*, 34(3):133–148, 2013.

[QCG+09] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[RC11] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[SAS+13] J. Saarinen, H. Andreasson, T. Stoyanov, J. Ala-Luhtala, and A. J. Lilienthal. Normal distributions transform occupancy maps: Application to large-scale online 3d mapping. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2233–2238, May 2013.

[SMAL12] Todor Stoyanov, Martin Magnusson, Henrik Andreasson, and Achim J Lilienthal. Fast and accurate scan registration through minimization of the distance between compact 3d ndt representations. *The International Journal of Robotics Research*, 31(12):1377–1393, 2012.

[SP13] N. Sünderhauf and P. Protzel. Switchable constraints vs. max-mixture models vs. rrr - a comparison of three approaches to robust pose graph slam. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5198–5203, May 2013.

[SSAL13a] J. Saarinen, T. Stoyanov, H. Andreasson, and A. J. Lilienthal. Fast 3d mapping in highly dynamic environments using normal distributions transform occupancy maps. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4694–4701, Nov 2013.

[SSAL13b] T. Stoyanov, J. Saarinen, H. Andreasson, and A. J. Lilienthal. Normal distributions transform occupancy map fusion: Simultaneous mapping and tracking in large scale dynamic environments. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4702–4708, Nov 2013.

[UT11] Cihan Ulas and Hakan Temeltas. A 3d scan matching method based on multi-layered normal distribution transform. *IFAC Proceedings Volumes*, 44(1):11602–11607, 2011.

List of Figures

1.1 On the left is visualization of occupancy grid. On the right is visualized point cloud of a room. 6
1.2 Local and global ambiguities in scan registrations. On the left is local ambiguity. On the right is global ambiguity with two maps. First map is original map without all information about environment. Second map is reality with all features. Registration wrongly associated matching based on information only from first map. . . 7

2.1 Image describing raytracing update. Yellow elipses represent normal distributions. Letter R represent robot position and red line ray tracing line. RCU will be applied to the cell marked A. A distribution in cell marked with letter B will get updated as unoccupied. Cell C will stay without any update. 15
2.2 Image on the left shows look up table before applying smoothing kernel. Image on the right is after application of kernel. 19

3.1 Diagram of graph based SLAM on NDT maps 21
3.2 Images with alignment. Red dost represent target scan and green dots source scan. The first row shows valid alignment marked with high score. Second row shows two alignments which were rejected by validation. 28

4.1 Overview of individual parts of architecture and their relationships. 31

5.1 The NDT map generated out of selected datasets with use of ground truth odometry information. 34
5.2 Comparison of effect of different frequency of frame creation. From left pictures of 1m, 2m and 4 meter distance between frames. Red arrows represent odometry edges, green edges are loop closure edges, white dots represent point cloud of means from each cell. 36
5.3 Number of correct and incorrect registrations in score groups . . . 37
5.4 Map of the room mapped by our NDT aproach, Hector mapper and Gmapping. Visualization of our aproach includes visualization of pose graph with red odometric edges and green loop closure edges. White spots represent means of each cell's normal distribution. . . 38
5.5 Map of the corridors mapped by our NDT approach, hector mapper and Gmapping. Visualization of our approach includes visualization of pose graph with red odometric edges and green loop closure edges. White spots represent means of each cell's normal distribution. 40
5.6 Top picture shows portion of map with corridor when incremental scan matcher may use initial guess from odometry. Bottom picture shows ambiguous registration in corridor area. 41

List of Abbreviations

D2D Distribution to distribution. 15, 16, 19, 21, 24–26

EKF Extended Kallman Filter. 5

GPS Global positioning system. 3

ICP Iterative closest point. 5, 7, 9, 17

IMU Integrated Measurement Unit. 4, 9

KF Kalman filters. 5

NDT Normal distributions transform. 3, 6–9, 12–16, 19–26

NDT-OM NDT-Occupancy mapping. 6–8, 13, 19, 21

P2D point to distribution. 16, 21, 22

PDF probability distribution function. 14, 15, 25

PF Particle filters. 5

RCU Recursive update for covariance. 12, 14, 33

ROS Robot operating system. 3, 8

SLAM Simultaneous localization and mapping. 3–10, 16, 19, 20

Appendices

A. ndt_gslam package

A.1 Overview

This package is used for simultaneous localization and mapping (SLAM) of an unknown environment. It creates a 2D map of the environment. This package is possible to use with or without information from odometry. This algorithm includes fast incremental scan matcher for precise odometry estimation. It also uses a graph-based representation of robot motion. It gives an advantage in recovering robot's map and position after significant drift or scan matcher's error. It provides two maps. The first map is from incremental scan matcher. It represents only the local area around the robot. It may be used for obstacle avoidance. The second map has information about the whole environment. Therefore, it is ideal for planning algorithms.

A.2 Architecture

This package includes three major parts. The first part is iterative scan matcher. It uses fast registration based on D2D-NDT alignment process. It can register incoming scans up to 70 Hz. The second part is pose graph holding small mini-maps inside the nodes. Edges represent relative transformation between two nodes. The graph also includes loop closure edges. These edges are created by observing the same place in the map from two different nodes. These loop edges can correct the map by using g2o graph optimization library.

A.3 Parameter specification

It is important to set up parameters correctly, to get maximum out of this package. The first parameter is a size of scan matching map. This parameter should be set based on a range of laser scanner. Window size parameter can be used for limiting the maximal range of laser scanner.

The second important parameter is a radius of search. This parameter sets how many nodes in surroundings of the last node will be checked for potential loop closure. Large radius will increase usage of computational resources.

The third parameter is a minimal distance for loop closure detection. It is calculated by going backwards over odometry edges in the graph. This process concatenates traveled distance on each edge. Resulting distance is checked against this parameter. If it is bellow, the limit node is not checked for loop closures. The idea behind this is that it is not necessary to check last couple nodes in the graph because they would not introduce any new information.

A.4 ROS API

Subscribed Topics

`scan` (sensor_msgs::LaserScan)
Laser measurements.
`odom` (nav_msgs::Odometry)
Robot's odometry information. Used if selected `subscribe_mode` == `ODOM`.
`pose` (geometry_msgs::PoseWithCovarianceStamped)
Robot's pose estimation. Used if selected `subscribe_mode` == `POSE`.

Published Topics

`map` (nav_msgs/OccupancyGrid)
Map of the environment.
`graph` (visualization_msgs::MarkerArray)
Visualization of pose graph.
`win_ndt` (ndt_gslam::NDTMapMsg)
Incremental scan matchers map.
`map_pcl` (pcl::PointCloud¡pcl::PointXYZ¿)
Point cloud of the map with points representing mean values from NDT cells.
`win_pcl` (pcl::PointCloud¡pcl::PointXYZ¿)
Point cloud of the scan matchers map with points representing mean values from NDT cells.

Parameters

`robot_base_frame_id` (string, default: `base_link`)
robot's base frame name in tf tree.
`odom_frame_id` (string, default: `odom`)
tf frame provided by odometry system.
`map_frame_id` (string, default: `map`)
frame id used in published maps. Algorithm creates tf transformation between `odom_frame_id` and `fixed_frame_id`.
`subscribe_mode` (string, default: `NON`)
three posible options are `NON`, `ODOM` and `POSE`. Based on selection of mode this node subscribes to correct topic. `NON` will not subscribe to any topic. `ODOM` will subscribe to `odom` and received odometry will be used in incremental scan matching. `POSE` will subscribe to `pose` and use pose estimate for incremental scan matching.
`scanmatch_window_radius` (double, default: `40`)
radius of incremental scan-matcher's map. Should be in meters.
`node_gen_distance` (double, default: `2`)
euclidean distance between two consecutive nodes in the pose graph.
`loop_max_distance` (double, default: `30`)
maximal search radius for loop closure edges detection. Higher values are computationally more demanding, but can recover map from bigger errors. Should be in meters.
`loop_min_distance` (double, default: `14`)

selects how many meters from current node may not be detected any loop closure. Distance is measured by concatenation of previous odometry edges' transformations. Example: if selected default `node_gen_distance` than last 8 nodes in graph will not be tested for loop closures.

`loop_score_threshold` (double, default: `0.6`)
loop closure rejection threshold. All potential loop closure edges with higher score than selected will be inserted into the pose graph. Value should be in range [0,1].

`serialize_graph` (bool, default: `true`)
turn on or off publishing of the pose graph visualization.

Required tf Transforms

`laser_frame` → `robot_base_frame`
This transformation is used for transforming laser scans to robot coordinate frame.

`robot_base_frame` → `odom_frame`
This transformation is necessary for correct calculation of provided tf transform. It is usually provided by odometry system.

Provided tf Transform

`map_frame` → `odom_frame`
Transformation localizing robot inside of calculated map.