Tutorial on Mutli-Robot Map Merging Using GraphSLAM

Vishnu Purushothaman Sreenivasan
University of Pennsylvania
Computer and Information Science Department, Robotics MSE
visp@seas.upenn.edu

Abstract— **Mapping large environments using mobile ground based robots is becoming increasingly popular with the recent improvements in the efficiency and accuracy of realtime Simulatenous Localizatin and Mapping (SLAM) algorithms. In most situations it is lucrative to have multiple robots simulaltaneously map different regions of the environment for faster exploration. The most popular algorithm for merging the individual maps is the Bundle Adjustment (BA) or the landmark based GraphSLAM. In this paper I present a comprehensive tutorial on using a simplified offline GraphSLAM algorithm for merging the individual 2D maps generated by any number of robots. The robots are assumed to have implemented some form of SLAM so that in a small region the maps generated are accurate. In this paper I delve into the details of utlizing scan matching with LIDAR sensors and replacing landmark features in the BA algorithm with smaller sub maps of the map created by each robot. The optimization problem is formulated as a least square error minimization by employing a graph structure. The optimization in this paper is performed into two parts: (1) the correction of the heading angle followed by (2) the correction of the X and Y positions of the local maps. The results show that though the problem is quadratic in time in the total number of smaller sub maps, it produces robust results and therefore is a very attractive option.**

I. INTRODUCTION

This paper is the report for the final project of the course, Learning In Robotics (ESE 650) at the University of Pennsylvania taught by Dr. Daniel D. Lee[1]. The project involves merging the individual 2D maps generated by robots exploring a large scale urban environment. The dataset chosen for this project is from the University of Pennsylvania, which was collected as a part of the Magic 2010 challenge [1]. The competition required the participants to design and deploy a team of field robots that could explore and map a large dynamic urban environment. The task also required the robots to identify and respond to threats in the environment. In this paper, I address the subproblem of merging the maps after the online SLAM employed by the robots. Each robot performs a SLAM algorithm to keep track of a probabilistic grid map[1]. The paper provides a specific framework for map merging using scan matching which can be used for one or many robots. The paper will explain the details of

1) Creating local sub maps for each robot.
2) Scan matching for finding correspondenses.
3) Yaw (heading angle) optimization from graph constraints using quadratic error minimization by employing regularized least squares.
4) Position optimization of the (x,y) positions of the local sub maps of all the robots in the global frame using the same technique as above.

I finally explain the results obtained from the above implementation. The goal of this paper is to elucidate the various steps involved in the implementation of the GraphSLAM algorithm using scan matching. Owing to the nature of the paper, I will cover the intuition behind the math and cover the equations superficially. But the paper provides references for more elaborate proofs and mathematical derivations. In the paper I finally discuss the results from implementing the algorithm and show that it provides robust outputs.

[1]Daniel D. Lee is with Faculty of Dept. of Electrical and Systems Engineering, Computer and Information Science (Secondary), Bioengineering (Secondary), University of Pennsylvania 200 S. 33rd Street, Philadelphia, PA 19104 ddlee@seas.upenn.edu

II. RELATED WORK

The GraphSLAM algorithm or its implementation for the purpose of multi robot map merging is not a new idea and has been successfully implemented many times [2], [3], [6] In the classical paper by Thrun et al. [2], the authors explain in detail the mathematical derivations and implementation of graph based SLAM algorithm for large scale mapping using landmark based features. They also explain techniques to exploit the sparsity of the graph constraints to improve of the speed of the linearized non linear optimization problem. A very similar algorithm is employed by Grisetti et al. [5] where they employ the Gauss-Newton Approach for error minization.

All the above papers though very comprehensive in the explanation of the mathematical proofs and the algorithms, do not explain the intricate nuances that appear when using scan matching for map correspondences in map merging problems. So, this paper addresses this issue by providing an elaborate step by step procedure for the above problem.

III. PROBLEM DEFINITION

In this section we will explore the problem in detail, with an explanation of the data set and the requirements of the project.

As mentioned in the introduction, the dataset used in this project was obtained from the General Robotics, Automation, Sensing and Perception Lab (GRASP) team of the University of Pennylvania from the data collected in their participation in the Magic 2010 challenge [1]. The 2010

Multi Autonomous Ground-robotic International Challenge (MAGIC 2010) was jointly organized by the Defence Science and Technology Organisation (DSTO) in Australia and the Research Development and Engineering Command (RDECOM) in the United States. The competition required the participants to design and deploy a team of field robots that could explore and map a large dynamic urban environment while simulataneously localizing, identifying and responding to threats in the environment.

The data is obtained from five robots exploring different regions of the environment with a fair amount of overlap in the regions explored. This enables merging of the maps using loop-closures (when the robots comes back to the same position it has or another robot has been before). The robots are equipped with a vertical and a horizontal lidar sensor, an IMU and a GPS tracking system. The details of the specific type of sensors are available in [1]. The robots performed a highly optimized scan matching technique for SLAM similar to [4], to build a probabilistic grid map.

The data provided include the poses of the robots in the global reference frame including their x,y positions and the yaw, pitch and roll angles. Also included is the horizontal and the vertical LIDAR hits in the global reference frame, along with a specification of which LIDAR hit corresponds to an obstacle. This obstacle information is important especially in the case of the vertical LIDAR. This information could be used to find ditches or pits which could otherwise be confused with obstacles. The GPS information provided has a low resolution and could not be used in the SLAM algorithm with any suitable benefits. For the purposes of this project we are only interested in the x,y positions and the LIDAR data, other informations are irrelevant.

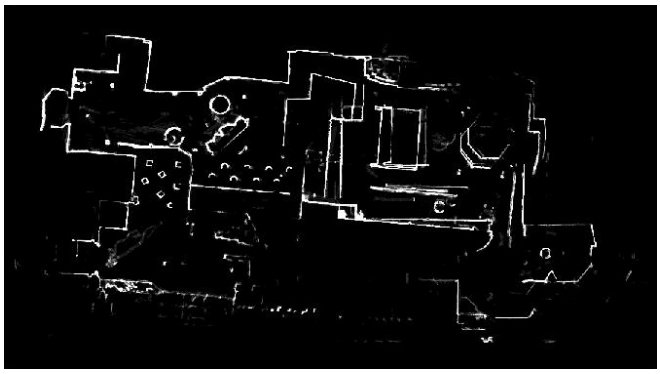

Fig. 1. The log map generated from robot 2 LIDAR and pose data. As we can see in most regions the map is quite perfect but somewhere along the end the robot lose track of its heading angle and starts producing double walls.

As it is clear from Figure 1, though the SLAM implementation for the individual robots is very good, it is not perfect. Some individual maps of robots have imperfections. Also all the robots do not start at the exact same location and there is an inherent difference in the orientation of the different individual maps as seen in Figure 2. The goal of this work is to construct an error free single unified global map from the individual maps offline.

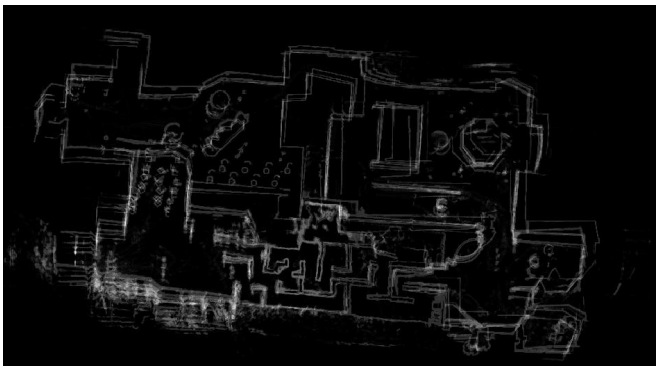

Fig. 2. The log map generated using all the robots LIDAR and pose data. It is quite apparent that the robots have a relative shift in their initial positions and in their estimate of their initial yaw.

IV. Mathematical Formulation

Before we can explain the mathematical formulation of the map merging problem, it would be useful to define some terminologies, and a brief overview of the algorithm of GraphSLAM. I will equivalently switch between the terminology of poses and states. Figure 3 is taken from [2].

A. Overview

On a high level the algorithm involves building a graph with the poses or the states of the robot and map features as the nodes of the graph. The arcs of the graph signifies the constraints between the poses and the map features or between the poses themselves. There are two types of graph arcs which are called by different names in the literature.

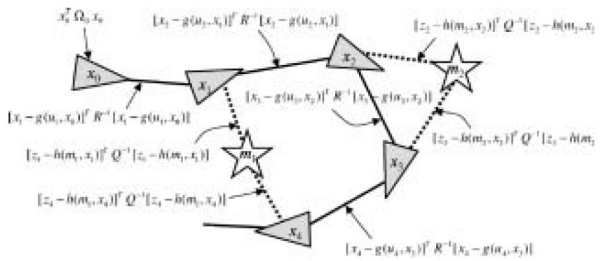

Fig. 3. GraphSLAM illustration. The triangles represent poses or the states of the robot. The stars represent the landmarks or the map features. The solid arcs correspond to motion contraints and the broken arcs correspond to map constraints.

- Edge corresponding to motion update, also known as the odometry update (sometimes referred to as temporal constraint). This contraint holds the immediate neighbors (previous state and/or next state) of the graph together. This is usually obtained from the wheel odometry and IMU. This constraint is shown in Figure 3 taken from [2].
- Edge corresponding to map features which can also be formulated as virtual measurements (sometimes referred to as spatial constraint). The map features also called

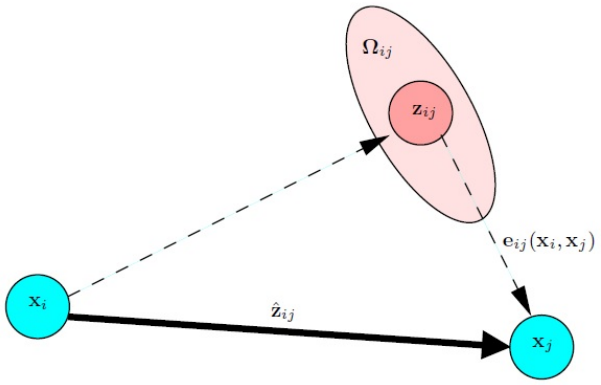

Fig. 4. Virtual measurement illustration. $\mathbf{x_i}$ and $\mathbf{x_j}$ refer to states. The measurement $\hat{\mathbf{z}}_{\mathbf{ij}}$ correspond to the relative position of $\mathbf{x_j}$ as seen from $\mathbf{x_i}$ in the current graph configuration and $\mathbf{z_{ij}}$ is the estimate of the same relative position but based on a common landmark or map feature that both the nodes $\mathbf{x_i}$ and $\mathbf{x_j}$ of the graph see. $\mathbf{\Omega_{ij}}$ represents the inverse of the covariance matrix representing the estimate of the noise in the sensor. $\mathbf{e_{ij}}(\mathbf{x_i}, \mathbf{x_j})$ is the error vector, between the current relative postion in the graph and the one corresponding to the measurement.

as landmarks are unique objects chosen from the environment such as trees, stationary vehicles and building features. This constraint is obtained when a state sees a landmark. Often the map features are marginalized out of the contraint optimization by adding additional constraint between poses. This is called a virtual measurement meaning that two states are constrained in their position based on how they observe the same landmark feature. This constraint is shown in Figure 4 taken from [5].

This graph is constructed by using the odometry information and finding correspondences in the landmarks. In our particular application the odometry is already done and therefore we can neglect the error due to it and instill a constraint with zero error. This is a valid assumption as the robot does a local SLAM and it is acceptable to assume that the consecutive states do not possess any relative error. For the map features, we use a virtual measurement based approach, that is we check each of the local sub maps which correspond to a mean estimate of state or pose with every other for correspondence using scan matching. We compute the error in rotation or position and use this error in the optimization. This will be explained in detail in the later sections.

B. Equations

Lets treat the problem formally now. The equations below are directly obtained from [5]. As I mentioned before, I highlight only the important details required for the implementation, please refer to the papers [5], [2] for a more formal treatment of the math.

Consider a set of state vectors $\mathbf{x} = (\mathbf{x_1}, \dots \mathbf{x_T})^\mathbf{T}$. We will follow the terminology explained in Figure 4, which is reiterated below for clarity, if $\mathbf{x_i}$ and $\mathbf{x_j}$ refer to two states, the measurement $\hat{\mathbf{z}}_{\mathbf{ij}}$ correspond to the relative position of $\mathbf{x_j}$ as seen from $\mathbf{x_i}$ in the current graph configuration and $\mathbf{z_{ij}}$ is the estimate of the same relative position but based on a common landmark or map feature that both the nodes $\mathbf{x_i}$ and $\mathbf{x_j}$ of the graph see. $\mathbf{z_{ij}}$ corresponds to the mean of the random variable corresponding to the measurement with $\mathbf{\Omega_{ij}}$ representing the inverse of the covariance matrix representing the estimate of the noise in the sensor. $\mathbf{e_{ij}}(\mathbf{x_i}, \mathbf{x_j})$ is the error vector, between the current relative postion in the graph and the one corresponding to the measurement.

$$\mathbf{e_{ij}}(\mathbf{x_i}, \mathbf{x_j}) = \mathbf{z_{ij}} - \hat{\mathbf{z}}_{\mathbf{ij}}(\mathbf{x_i}, \mathbf{x_j}) \tag{1}$$

If we try to find the negative log-likelihood $\mathbf{l_{ij}}$of the observed virtual measurement based on our current graph configuration,

$$\mathbf{l_{ij}} = [\mathbf{z_{ij}} - \hat{\mathbf{z}}_{\mathbf{ij}}(\mathbf{x_i}, \mathbf{x_j})]^\mathbf{T} \mathbf{\Omega_{ij}} [\mathbf{z_{ij}} - \hat{\mathbf{z}}_{\mathbf{ij}}(\mathbf{x_i}, \mathbf{x_j})] \tag{2}$$

So our optimization problem becomes, to minimize this negative log-likelihood,

$$\mathbf{F_x} = \sum_{<\mathbf{i},\mathbf{j}>\in\mathbf{C}} \mathbf{e_{ij}^T \Omega_{ij} e_{i,j}} \tag{3}$$

So we need to find $\mathbf{x}$ (the states) such that

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \mathbf{F_x} \tag{4}$$

The optimization can be approached in many ways but we use the equations from [5] which used Gauss-Newton or Levenberg-Marquardt algorithms. First we need to linearize the negative log likelihood function by linearizing the error function using Taylor series. This requires an initial guess $\tilde{x}$, *the quality of the initial guess is very important to avoid local minima.* This is not an issue for our problem because each robot has ran a SLAM algorithm so our initial guess is very close to the global optimum.

The linearization and derivation of the Gauss-Newton algorithm is well explained in [5]. I will only write down the final equations.

$$\mathbf{e_{ij}}(\tilde{\mathbf{x}}_\mathbf{i} + \mathbf{\Delta x_i}, \tilde{\mathbf{x}}_\mathbf{j} + \mathbf{\Delta x_j}) = \mathbf{e_{ij}}(\tilde{\mathbf{x}} + \mathbf{\Delta x}) \simeq \mathbf{e_{ij}} + \mathbf{J_{ij}} \mathbf{\Delta x} \tag{5}$$

where, $\mathbf{J_{ij}}$ is the Jacobian of $\mathbf{e_{ij}}(\mathbf{x})$ at $\tilde{\mathbf{x}}$

$$\mathbf{H \Delta x}^* = -\mathbf{b} \tag{6}$$

where H is called the information matrix and b is called the information vector and are defined as below,

$$\mathbf{H_{ij}} = \mathbf{J_{ij}^T \Omega_{ij} J_{ij}} \tag{7}$$

$$\mathbf{H} = \sum \mathbf{H_{ij}} \tag{8}$$

$$\mathbf{b_{ij}} = \mathbf{e_{ij}^T \Omega_{ij} J_{ij}} \tag{9}$$

$$\mathbf{b} = \sum \mathbf{b_{ij}} \tag{10}$$

The information matrix $\mathbf{H}$ is a $d*n$ by $d*n$ matrix where d is the number of dimensions of the state vector and n is the number of states being optimized. The information vector $\mathbf{b}$ is of dimension $d*n$ by 1. The Jacobian $\mathbf{J}$ is a d by $d*n$ matrix. We will go into the details of how to compute each of these terms for our problems in detail in the later sections. Some important things to note is the, structure of the $\mathbf{J_{ij}}$ and $\mathbf{H_{ij}}$, which are mostly sparse as explained in [5]

V. Details of Implementation and Results

In this section, I will provide an in depth tutorial on implementing the offline GraphSLAM algorithm for map merging using scan matching. I will also go into the details of the values I used for my implementation but the caveat here is that the optimal values can differ greatly depending on the quality of the sensors used, the quality of the online SLAM algorithm employed and the size of the global map etc. I also discuss the results obtained.

As mentioned in the introduction the pipeline for this post mapping, map merging problem is iterated in the following steps,

1) Creating local sub maps for each robot.
2) Scan matching for finding correspondenses and finding the yaw error.
3) Yaw (heading angle) optimization from graph constraints using quadratic error minimization using regularized least squares.
4) Scan matching for finding correspondenses and finding the error in (x,y) position after correcting the yaw angles.
5) Position optimization (x,y) positions of the local sub maps of all the robots in the global frame using the same technique as above.
6) Post-processing to create a cleaner map.

Please note the additional steps of finding correspondenses post yaw correction and post-processing of the map which was deliberately left out in the introduction for brevity.

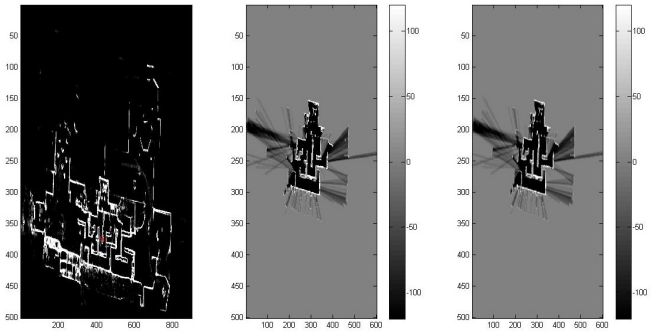

Fig. 5. The first subplot shows the map for the robot 1. The second subplot shows a sub map of robot 1 at the position indicated by the red circle in subplot 1. The third sub plot shows the gaussian smoothed sub map.

A. Getting Local Sub Maps

Here we discuss the procedure to obtain local sub maps of each robot which I will hence just refer to as sub maps. First depending on the size of the global map and the implementation of the online SLAM algorithm we choose a distance value based on which we will create the sub maps for each robot. If the quality of the implementation of the SLAM is very good, then it is reasonable to assume that over a large region the map will be accurate. Similarly if the global map size is very large it is advisable to take a lower resolution for the map as well grouping several poses together. In the UPenn dataset, each robot has approximately 2000 pose information along with the other sensor data. I first trim the initial 400 packets during which none of the robots move. Then I choose a distance threshold of 10 metres. Starting from a pose x_n, I group x_n to x_{n+k} poses together, if x_{n+k} is the first pose after x_n which is more than the distance threshold away. I also give a 20 percent overlap between successive maps. The sub maps are occupancy grid maps with each grid value indicating the log odd probability of being an obstacle. I use a local map of size -30m to 30m along the x direction and -25m to 25m along the y direction, with a resolution of 0.1m. If the horizontal LIDAR hits cell in the map I update it by 20 (log odds update). I then use the Bersenham's line drawing algorithm to find the empty cells and decrement them by 10. I also decrement the cells where the vertical LIDAR says its empty by 10, to avoid pits and ditches. I upper and lower bound the log odds value to 120 and -120 respectively. I then gaussian smooth the map with a window of size 2 by 2 and sigma of 1.3.

B. Map Correspondence and yaw error

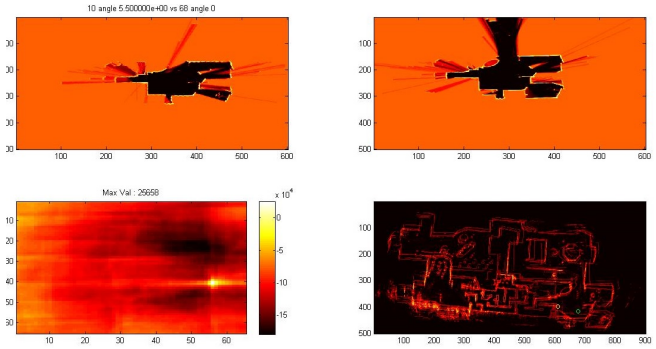

Fig. 6. The left hand top figure shows a rotated sub map of robot 1 and the right hand top figure shows a rotated sub map of robot 4. The bottom left image shows the correlation map where the x axis indicates the x shifts and the y axis indicates the y shifts. The bottom right image gives the robot 1 position as a yellow circle dot and the robot 4 as a green circle dot in the global map.

In this section we will explore the technique to find map correspondenses, that is sub maps which are near to each other and share common features. But we limit the search to sub maps which are at a maximum radius of 10m from each other. We use a standard scan matching technique, where we extract the LIDAR points of one sub map and add the cell values of the grid points it corresponds to in the other sub

map. We also shift the x and y of the LIDAR points so that we find a good match at a particular shift. We choose the maximum value in the correlation map and if it is above a threshold we accept it as a correspondence. For this part I chose a threshold of 17000. The different shift values I chose for x and y were between -8m to 8m with a grid spacing of 0.25m. But we here have a dual task of estimating the relative difference in the yaw of the two maps. In a general case this should match with the pose information given. In the chosen data set since the maps are in the global frame, the difference in the yaw becomes the error itself(the actual difference should be zero) which we need to minimize. In order to ascertain the yaw angle, I computed the edge map of the sub map. Using the edge map, I found the peak angle of the map using Hough transform. I chose the top two peaks and have a threshold that the second peak should be atleast 0.5 times the first peak. Then I rotate both the maps with the 4 different combinations of the angles and then compute the map correlation. If the peak is above the threshold for one combination, then it is accepted as a correspondence and the difference in the angles corresponding to the peak value is taken as the error in yaw.

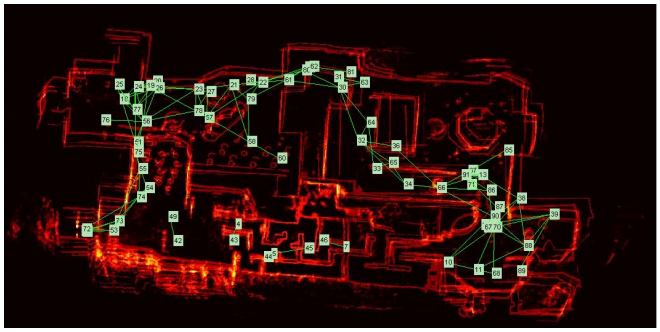

Fig. 7. Spatial graph constraints between the different sub maps. The numbers indicate the pose number ordered from robot 1 to 5.

C. Yaw Optimization

Once we have all the graph constraints and the corresponding yaw error we can build the so called information matrix $\mathbf{H}$ and the information vector $\mathbf{b}$. But we realize we do not have the $\mathbf{\Omega_{ij}}$ matrix or the inverse of the covariance matrix (here it is just the inverse of the variance of yaw). If one has a problem which involves sensors whose errors which can experimentally determinable then one could use that to estimate the correct $\mathbf{\Omega_{ij}}$. For the problem given I choose an arbitrary $\mathbf{\Omega_{i,i+1}}$ for the temporal constrainsts and a larger $\mathbf{\Omega_{ij}}$ for spatial constraints multiplied by the maximum value of the correlation matrix which would strengthen the constraints of the most correlated sub maps. Since it was a one dimensional case I reformulated the problem as below. I simplified the construction procedure which effectively produces the same information matrix and information vector as the algorithm mentioned before. I created a square tridiagonal matrix with the value 4 along the main diagonal and -2 along the other two diagonals with the dimensions equal to the number of poses or the sub maps. This enforced the temporal constraint between the current state with the previous and/or the next state. I removed the constraints between the last pose of one robot $\mathbf{x_i}$ with the first pose of another robot $\mathbf{x_{i+1}}$ by subtracting 2 from the main diagonal elements i, i and $i+1, i+1$ in the information matrix. Simultaneously I added 2 to $i, i+1$ and $i+1, i$. Then to all the spatial constraints from map correlation $< i, j >$, I add 100 times the maximum value of the map correlation matrix to i, i and j, j elements, and I subtract the same amount from i, j and j, i in $\mathbf{H}$. The larger value for spatial constraints is to give a higher penalty for getting them wrong. $\mathbf{b}$ is constructed by adding 100 times the error multiplied by the maximum value of the correlation matrix for each constraint $< i, j >$ at j and subtracting the same value from i. The information vector is updated with zeros for the temporal constraints.

Fig. 8. This shows the log map of all the sub maps combined after rotating by the computed yaw from the optimization.

I then add a small value (10^{-9}) to the main diagonal elements to make the matrix non singular, this can also be thought as a ridge regularization. A value of 1000000 is added to the $\mathbf{H}(\mathbf{1}, \mathbf{1})$ element to anchor the first element(else the entire map can shift as there are only relative constraints in the optimization problem). We now compute the change in yaw angle using Equation (6) and rotate all the sub maps.

D. Map Correspondence and (x,y) position correction

We use the same technique as for yaw for finding map correlation using scan matching but this time we do not need to rotate the maps. We just extract the LIDAR points from the second map and scan match it with the first map. We evaluate the maximum value of the correlation matrix and if it is above a threshold of 20000 we accept it as a match. Here I chose a finer resolution for x and y shift, varying between the same -8m to 8m but with a grid spacing of 0.15m. I also limit the search to the same 10m radius. I first obtain the difference in the x and y positions of the matched sub maps in the global reference frame in the current graph and then find the difference map correlation gives. The error is the disagreement between these two values. We need to compute $\mathbf{\Omega_{ij}}$ matrix as here we have a 2 dimensional error vector and the optimization would depend on their individual confidence and their interaction. $\mathbf{\Omega_{ij}}$ is here the Hessian matrix at the peak point of the map correlation matrix obtained from scan matching for different values of x and y shift. The Hessian is defined as below

$$\mathbf{H}' = \begin{vmatrix} C_{xx} & C_{xy} \\ C_{xy} & C_{yy} \end{vmatrix} \tag{11}$$

where C_{xx}, C_{yy} and C_{xy} represents the second differential of the map correlation matrix at the peak point along the x direction, along the y direction and along the x direction followed by y the direction respectively. This can be numerically computed. For a more accurate result I use bilinear interpolation to estimate the above differentials at a distance of 0.015m in both the dimensions from the peak point. An important caveat is that sometimes we may not be able to compute the bilinear interpolation resulting in garbage values (in Matlab it may yield a NaN). In such situation we can either ignore the constraint, take a larger distance when computing the bilinear interpolation or choose an arbitrary small $\mathbf{\Omega}$ matrix.

E. Position (x,y) optimization

For this optimization, I exactly follow the algorithm explained in the previous section. To the spatial correspondences and the errors computed in the previous step, I add the temporal constraints between every consecutive robot poses but assuming zero error. I assume that the error vector to take the following form

$$\mathbf{e_{ij}} = \mathbf{z_{i,j}} - (\mathbf{x_j} - \mathbf{x_i}) \tag{12}$$

Since the Jacobian is the first differential of the error for a constrain $< i, j >$, the Jacobian would be

$$\mathbf{J_{ij}} = \begin{vmatrix} \dots & 1 & 0 & \dots & -1 & 0 & \dots \\ \dots & 0 & 1 & \dots & 0 & -1 & \dots \end{vmatrix} \tag{13}$$

where the identity matrix corresponds the ith and the negative of the identity matrix corresponds to the jth postion. Here for the temporal constraints I use a identity matrix as the covariance matrix and for the spatial constraints I use the absolute value of the Hessian matrix multiplied with the maximum of the map correlation for the constraint as the $\mathbf{\Omega}$ matrix. I use the equation (7), (8), (9), (10) to compute the information matrix $\mathbf{H}$ and the information vector $\mathbf{b}$. I add a small value (10^{-9}) to the diagonal of $\mathbf{H}$ to avoid rank defeciency and I add a large value 1000000 to $\mathbf{H}(\mathbf{1},\mathbf{1})$ and $\mathbf{H}(\mathbf{2},\mathbf{2})$ to anchor the first pose at the origin. I then use equation (6) to compute the shifts in the positions of the submaps and shift them by shifting their centres. This completes the optimization procedure. This procedure can be iterated for better accuracy. The entire optimization takes about 7 minutes on a computer with a intel i-5 2nd generation 2.1GHz processor with 8GB RAM and a 2GB GPU card.

F. Post Processing

I use a simple sliding window which scales the values so that the minimum value in the window becomes -100 and the maximum value becomes 100. The window size 50m in x direction and 100m in y direction. It helps in bringing out the darker regions but produces a pixelated image of the map. This step can skipped or changed as necessary.

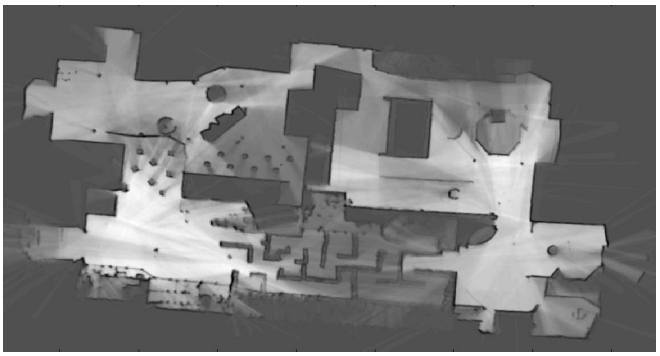

Fig. 9. This shows the negative log map of all the sub maps combined after rotating by the computed yaw and shifting by the (x,y) positions obtained from the optimization.

Fig. 10. Post-processed map.

VI. CONCLUSIONS

In this paper I have presented a step by step tutorial for implementing an offline GraphSLAM algorithm for merging individual 2D maps from any number of robots using scan matching. The results obtained on the University of Pennsylvania dataset produced more than convincing results. There is not any specific novel endeavors in this paper except the scaling of the covariance matrix based on the strength of the correspondences. This is because the goal of the paper was to provide an extensive discussion on the implementation of the algorithm explained in [5]. Right now the algorithm performs very well but the time taken for the optimization is quite a burden and future work would involve optimizing the algorithm for more efficient computations.

ACKNOWLEDGMENT

I would like to thank my professor Dr. Daniel D. Lee for all the support and the inspiration he provided. His classes were the most enlightening and helped me in understanding the most difficult concepts in robotics very easily. I would also like to extend a special thanks to Zhou (Jimmy) Wang, TA for the course Learning in Robotics (ESE 650). The completion of this project would not have been possible without his timely support and aid.

REFERENCES

[1] Butzke, Jonathan, et al. "The University of Pennsylvania MAGIC 2010 multi-robot unmanned vehicle system." Journal of Field Robotics 29.5 (2012): 745-761.

[2] Thrun, Sebastian, and Michael Montemerlo. "The graph SLAM algorithm with applications to large-scale mapping of urban structures." The International Journal of Robotics Research 25.5-6 (2006): 403-429.

[3] Chang, H. Jacky, et al. "Multi-robot SLAM with topological/metric maps." IROS. 2007.

[4] Olson, E. (2008). Robust and efficient robotic mapping. Ph.D. thesis,MIT, Cambridge, MA

[5] Grisetti, Giorgio, et al. "A tutorial on graph-based SLAM." Intelligent Transportation Systems Magazine, IEEE 2.4 (2010): 31-43.

[6] Fox, Dieter, et al. "Distributed multirobot exploration and mapping." Proceedings of the IEEE 94.7 (2006): 1325-1339.