# Autonomous localization and mapping using GraphSLAM and Scan Matching

*C. Laschet* *T. Pepels*

**Abstract**

The simultaneous localization and mapping (SLAM) problem, in which a robot has to create a map from its environment while simultaneously localizing itself on this map, has been a popular research topic in the robotics field for the past two decades. A well-known approach for solving this problem is the GraphSLAM technique, which represents the SLAM problem as a graph. Nodes of the graph represent robot positions and scan data, while edges represent constraints between these nodes. The map of the environment can then be computed by finding a configuration of the nodes such that they are as consistent as possible with the constraints between these nodes.

In this paper, a general comparison between various SLAM techniques is given and a detailed explanation of GraphSLAM is provided. Moreover, scan matching and the loop closure detection enhancement for GraphSLAM is discussed to improve the performance of GraphSLAM.

## 1 Introduction

A correct mapping of the environment of a robot is essential for performing complex actions, such as automated driving, robot vacuum cleaning and search and rescue operations. Moreover, an accurate localization of the robot on this map is also essential to perform any of these tasks. When external reference systems such as GPS are not available, which is often the case, this environment mapping and estimated robot pose has to be calculated from the robot's sensor readings. These sensor readings often include significant amounts of noise, which complicate this process even more.

The problem of simultaneous acquisition of the environment map and localization of the robot, which is also known as SLAM, has been a major research topic in robotics for the past two decades. SLAM is a process which tries to answer the questions of what the world surrounding the current position of the robot looks like, and where the robot is situated on this map. The problem however, is that a map is needed to determine the current location of the robot, and a localization of the robot is needed to create a map of the environment. In other words, the processes of localization and mapping are dependent on each other. Therefore, these questions can never be answered independently. This property is also the reason that most of the techniques developed for SLAM take an iterative approach, which try to repetitively improve the map and pose estimate of the robot.

From a probabilistic perspective, techniques for solving the SLAM problem can be divided into two categories: offline and online SLAM [8]. Offline SLAM, which is also known as full SLAM, tries to calculate the full posterior of the entire path taken by the robot during a period of time, along with the map, given the measurements that were observed and controls that were executed during this time frame. This approach can be formulated as follows:

$$\Pr(x_{1:t}, m \mid z_{1:t}, u_{1:t}) \tag{1}$$

Where $x_{1:t}$ is the estimate of the traversed path of the robot, $m$ is the map generated whilst traversing the path, $z_{1:t}$ are the measurements taken and $u_{1:t}$ the controls executed by the robot.

The online SLAM problem involves estimating the current pose of the robot along with the map, given the measurements taken and controls executed so far:

$$\Pr(x_t, m \mid z_{1:t}, u_{1:t}) \tag{2}$$

While both the online and offline variants are of equal practical importance, calculation of the full posterior is usually infeasible. SLAM is most often performed in high dimensionality environments with thousands of features, if not more. Therefore, practical solutions must rely on approximations calculated by an online SLAM technique.

A well-known online SLAM technique is the EKF SLAM algorithm [8]. The GraphSLAM technique [2, 8] is an example of an offline SLAM method. Between these two extremes are many other techniques, such as the Sparse Extended Information Filter (SEIF) [9] and FastSLAM [7]. Each technique has its own advantages and disadvantages. However, this will not be discussed here, since it lies out of the scope of this paper.

Multiple enhancements have been proposed for various SLAM methods. In this paper, scan matching [1, 6] is used as an enhancement for GraphSLAM to compensate for measurement and odometry errors of the robot. Moreover, loop closure detection [3, 2] is also used to enable recognition of previously seen features. Both of these enhancements are discussed in detail in this paper.

The outline of this paper is as follows. Section 2 provides a description of the scan matching method used for this paper. Next, section 3 explains the workings of exploring the environment given the current mapping of the robot. Section 4 then continues with a detailed discussion of GraphSLAM. Section 5 deals with combining these techniques as a working implementation, focussing on specific implementation details and added enhancements, such as the loop closure detection enhancement. Experiments and results are provided in section 6, and finally a short conclusion is provided.

## 2 Scan Matching

The general objective of scan matching is to compute the relative change in pose of a robot, by maximizing the overlap of two measurement data sets, observed at different poses. This estimate of relative change can be used in GraphSLAM for initial estimation of constraints between two nodes, and for correction of the robot pose due to small odometry errors.

Iterative Closest Point (ICP) [1, 6] is a scan matching technique that was used in this paper as an enhancement for GraphSLAM. The goal of ICP is to minimize the metric difference between two sets of points, called pointclouds. In this case, these pointclouds were created from laser sensor readings, where each point represented the end of one laser ray casted at an angle from the current robot pose. The Euclidian distance is usually used as a difference error to minimize. In this case however, a more advanced point to line metric was used [1], because Euclidian distance only takes translational change and no rotational change into account. The result of ICP is then a translational and rotational change which represents the change in poses between the two pointclouds that were provided.

ICP takes as input a reference scan $S_{ref}$, the current scan which will be compared with the reference scan $S_{current}$ and an initial estimation of relative change in pose, $q_0$. The output of ICP is then denoted by $q$, which represents the final estimation of relative change in poses of $S_{ref}$ and $S_{current}$. The ICP algorithm then iteratively repeats the following three steps until convergence of $q$:

1. In the first step, correspondences between individual points of laser data are found. For each point $p_i$ of $S_{current}$, the point $c_i$ is entered into $S_{ref}$ by using the current estimate $q_k$.

$$c_i = q_k(p_i) \tag{3}$$

2. Next, the best correspondence to $c_i$ is found by selecting the point from a range centred around the original point $p_i$ with the smallest distance measure. This results in a set $C$ of correspondence pairs $(p_j, c_j)$.

$$min\{d(c_j, [p_i, p_{i+1}])\} \tag{4}$$

3. Finally, compute a new estimation $q$ that minimizes the mean square error between each pair of $C$, which is shown below.

$$E_{dist}(q) = \sum_{j=1}^{n} d(p_j, q(c_j))^2 \tag{5}$$

Because a solution is computed iteratively, ICP can be used in real-time environments, and therefore is also applicable for online GraphSLAM. It should however be noted, that the initial estimate $q_0$ is essential for obtaining significant results. Without a decent initial estimate, ICP may become stuck in local minima [6]. Moreover, scan matching does not function well when the robot is situated in a large room with no objects in range of the sensor. In that case, there isn't any data available to match, which results in poor performance.

## 3 Exploration

To generate a map with SLAM, it is required that the robot drives around in its environment and receives measurement data of this environment. Without movement, the robot would only be able to process measurement data of the starting pose. In other words, SLAM calls for an exploration strategy.

An exploration strategy takes the current map created by SLAM and decides, based on this map, which position on the map the robot should move to next. Of course, large unknown areas are more interesting to explore than already mapped areas. In the case of this paper, Wave Frontier Detection (WFD) [4, 10] was used. WFD uses the notion of frontiers to decide where the robot should move to. A frontier is a border between the unoccupied known region, which has already been sensed by the robot and therefore been mapped by SLAM, and the unknown region, which is the area the robot has never seen before. Each frontier consists of frontier points, which are placed along this border.

An example of the result of WFD is shown in Figure 1, where the white dots represent frontier points. Each group of white dots form a frontier. Obstacles are shown in black, light grey areas represent the known region and dark grey areas show the unknown region.

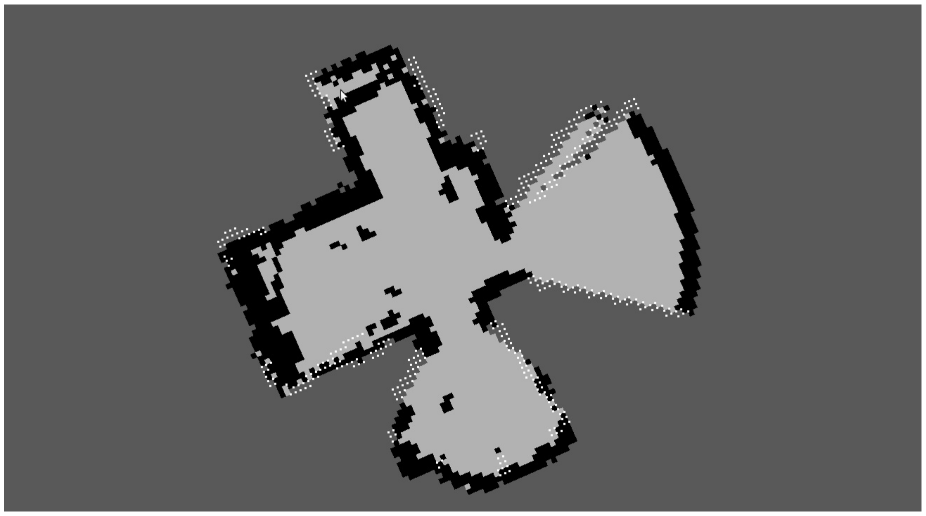

Fig. 1: Frontiers generated by WFD are denoted by the groups of white dots.

Figure 2 depicts the process of finding frontiers with WFD. WFD starts with a breadth first search in the known region, starting from the current robot pose, searching for frontier points (1). When a frontier point is found (2), a second breadth first search is performed in the known region, starting from the previously found frontier point (3). This second breadth first search searches for new frontier points which are connected to the original frontier point found in (2), creating a collection of connected frontier points. The resulting collection of found frontier points (4) is then added as a new frontier, or added to an already existing frontier when the collection is connected to this frontier. A more detailed explanation can be found in [4, 10].

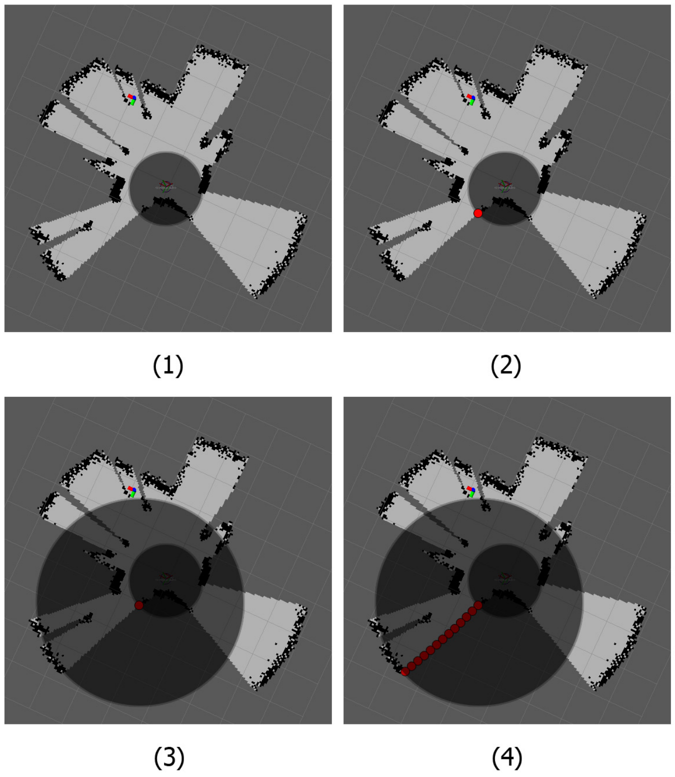

Fig. 2: WFD approach of finding frontiers.

Because frontiers are never situated in the unknown region, the key advantage of WFD is that it prevents scanning any parts of the unknown region. However, WFD does search the complete known region. The disadvantage of this technique is that frontier detection takes more time as more exploration has been performed.

After calculation of the frontiers, the robot picks one of these frontiers to move to. By moving to this new frontier, unknown space is explored and new frontiers are generated. As the robot moves around in its environment, SLAM provides new maps, which can then be used again by WFD. This process repeats until no more frontiers are found, which means that the environment is completely explored and the mapping process is completed.

## 4 GraphSLAM

As was discussed before, GraphSLAM is an offline SLAM method. It calculates the estimate of a complete path taken by a robot and a map of the environment seen during the traversal of the path, based on the laser range data and controls that were observed and executed during that time frame.

GraphSLAM uses observed data and executed controls for generation of a sparse graph. This sparse graph represents the complete set of data known by GraphSLAM, which is then used for calculation of the path estimate and map generation. Nodes in this sparse graph represent either a previous robot pose, the current robot pose or map features. In this case, these map features are previously observed laser data. Arcs between nodes represent a dependency between these two nodes, which can be divided into two categories. Motion arcs link any consecutive robot pose nodes and therefore represent a constraint between these two poses, which is generated from odometry data. Measurement arcs link poses to features that were measured at that pose. A measurement arc therefore represents a constraint which is derived from laser data.

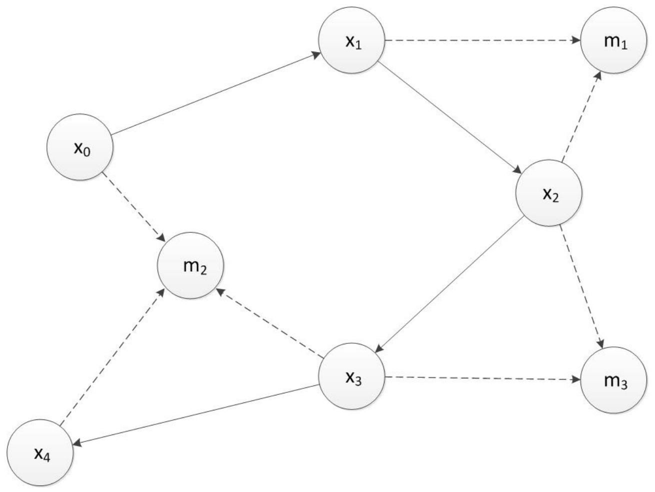

Fig. 3: A graph generated by GraphSLAM.

Figure 3 shows an example of a graph generated by GraphSLAM, in which the robot has traversed five consecutive poses $x_{0:4}$ and observed three features $m_1$, $m_2$ and $m_3$. Solid line arcs represent motion arcs; dashed line arcs represent measurement arcs.

While the above description of GraphSLAM is visually appealing, most of this information is internally stored in matrices. GraphSLAM utilizes two matrices, the information matrix $\Omega$ and the information vector $\psi$. The information matrix is a square matrix of size $n$, where $n$ is the number of nodes in the graph, and holds the measurement and motion constraints between these nodes. The information vector has $n$ rows and contains the actual values that the constraints of $\Omega$ should satisfy.

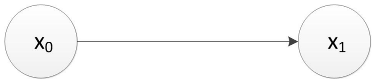

Fig. 4: Example graph of a robot moving from $x_0$ to $x_1$.

For example, suppose a robot moves from pose $x_0$ to $x_1$, and does so by moving 2 units forward. The resulting graph is shown in Figure 4. In that case, $x_1 = x_0 + 2$ and therefore $x_0 - x_1 = -2$ and $x_1 - x_0 = 2$. The resulting information matrix $\Omega$ and information vector $\psi$ are shown in 6. Note that the first row refers to node $x_0$, while the second row refers to node $x_1$.

$$\begin{array}{c} x_0 \\ x_1 \end{array} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} -2 \\ 2 \end{bmatrix} \tag{6}$$

To continue this example, suppose a laser scan was performed at position $x_0$, resulting in measurement data $m_1$ and that this measurement data returned a range of 7, i.e. an obstacle was seen 7 units away from the current pose. Then a constraint exists between the pose node $x_0$ and the measurement node $m_1$, which is denoted by $x_0 = m_1 + 7$ and therefore $m_1 - x_0 = -7$ and $x_0 - m_1 = 7$. The resulting graph is shown in Figure 5.

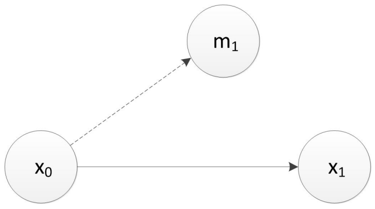

Fig. 5: Example graph of a robot moving from $x_0$ to $x_1$ and measuring $m_1$ at $x_0$.

Next, the new node and its constraints are added to the already existing $\Omega$ and $\psi$ matrices that were constructed in previous steps. The result of adding the new node and the according measurement constraint is shown in 7. Note that the size of both matrices has increased to accommodate for the new node $m_1$.

$$\begin{array}{c} x_0 \\ x_1 \\ m_1 \end{array} \begin{bmatrix} 2 & -1 & -1 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} -9 \\ 2 \\ 7 \end{bmatrix} \tag{7}$$

This process of expanding the matrices $\Omega$ and $\psi$ and adding new measurement and motion constraints, is repeated for every measurement observed or motion control executed by the robot. Essentially, a system of linear equations is created over time, by mapping the graph's edges to constraints in matrix form.

The size of the matrices increases as more features are observed and longer time spans with more controls are used as input. Moreover, most of the constraints are non-linear. This calls for an optimization process. Fortunately, GraphSLAM has access to the full data when building the map, and therefore can apply improved linearization and data association techniques, such as linearization of constraints [8] and scan matching [1, 6]. These improvements generally cause GraphSLAM to be more accurate than EKF approaches. GraphSLAM does however have limitations on how much input can be provided, since the size of the matrices increases linearly over time.

After these linearization and data association procedures are applied, the linear system of matrices created by GraphSLAM can be solved. The final path estimation of the robot $\mu$ is given by the equation shown in 8, where the inverse of $\Omega$ is multiplied by $\psi$.

$$\begin{bmatrix} x_0 \\ \vdots \\ x_n \\ m_1 \\ \vdots \\ m_k \end{bmatrix} = \Omega^{-1}\psi \tag{8}$$

## 5 Implementation

This section provides an insight into how the above explained techniques were combined into a working implementation. The internal structure of the graph and the process of localization are discussed. Moreover, the loop closure detection enhancement and the solving of the graph are explained.

### 5.1 Graph Structure

The graph used by GraphSLAM consists of a collection of nodes and a collection of edges.

- A node holds sensor data of the laser scan that was made at the pose of that node, an estimation of the pose based on the odometry of the robot, a corrected estimation of the pose provided by scan matching, and a local occupancy grid created from the laser sensor data as follows:
  - Unkown and empty space is determined by raycasting from the robot-pose along the scans.

    - Blocked space is determined by the endpoints of the laser-scans.

- An edge holds references to a parent and child node, along with a mean and covariance, which is provided by the scan matching process. The mean and covariance are eventually used for solving the graph.

Together, these lists of nodes and edges represent the complete graph. Each time a node is inserted into the graph, the current scanmatched pose is inserted and scan matching is performed between the last inserted node and the current node, which provides the covariance between the current and previous node. Edges are inserted with the corresponding mean and covariance provided by scan matching. Finally, the node itself is inserted into the collection of nodes. The mean and covariance values of the edges, along with the scan matching estimated pose of each node is used when solving the graph. After solving the graph, the local occupancy grids at each node are used to generate the final map. This is done by overlapping the local grids into a large occupancy grid. A simple threshold on each global grid determines if the grid will be occupied/empty/unknown in the map.

## 5.2 Localization

Each node in the graph holds an estimated pose. Because the odometry of the robot will be corrupted with noise, the pose estimate based on odometry could be off by a large margin. Therefore, scan matching is used to improve this pose estimation. Scan matching re-aligns the estimated pose, by matching its laser data with the laser data of the previously inserted node. This technique ensures a higher accuracy of the node's pose estimation, which results in better optimization during the solving of the graph.

Moreover, to ensure a constant update of the current robot-pose. Scan-matching is performed continuously, generating an estimated pose based on laser data. Each time laser data is received, an estimated change in pose is computed. This estimation is used by scanmatching to generate a new pose. The new laserdata is compared to data from a reference pose. This reference pose is updated discretely, after either 5cm of movement or 5 degrees of rotation.

## 5.3 Loop closure detection

As the robot moves around in its environment, pose uncertainty increases due to noise in odometry data. These errors add up while performing actions over time. Moreover, this increases the uncertainty of measured data, because measurement data is linked to the pose at which this data was observed. As a result, pose and sensor data uncertainty increases over time.

When the robot reaches a pose that is closely located to a pose it has seen before, a loop is created in the graph created by GraphSLAM, by adding an edge between these nodes. The constraints of this edge are provided by performing scan matching between the poses of these two nodes. If the result generated by scanmatching is similar to the actual distance between the nodes, we conclude that the nodes are in fact closeby and within visual range of eachother. Otherwise, although these nodes are closeby, if their scanmatching results are too dissimilar, it is likely they are not in visual range of eachother, i.e. there is an obstacle between them. In this case a new constraint is not added to the graph. The added constraint results in a more certain estimation of the poses and measurement data of these nodes and a better posterior of the robot path in general. This is especially true when the pose of the related node is relatively well known. Moreover, the coherency of the complete graph is improved by adding additional links between nodes.

The loop closure detection enhancement detects these opportunities to create loops. When GraphSLAM inserts a node that is positioned close to another node, the previously discussed loop is created by adding an edge between these nodes. An example is shown in Figure 6. The node that holds pose $x_5$ is linked to pose $x_0$ due to its proximity to this pose. Because $x_0$ is the starting node, its pose is known with great certainty. Linking $x_5$ to this node increases the certainty of the estimation of pose $x_5$.

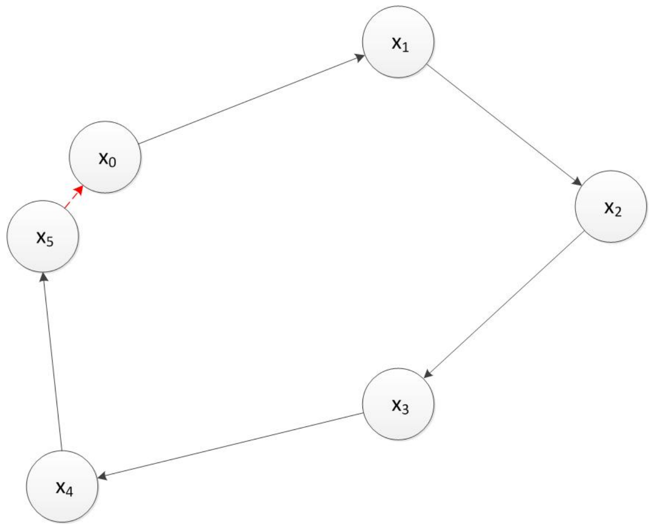

Fig. 6: WFD approach of finding frontiers.

## 5.4 Solving

Solving of the graph created by GraphSLAM is done by the g2o library [5], an open-source framework for optimizing graph-based nonlinear systems. This library has proven to offer stable performance while still being suitable for a wide range of problems. The g2o library takes as input the previously described nodes and edges, which hold the by scan matching corrected estimated pose, mean and covariance, and returns for each node the optimized pose. The existing nodes are then updated with the results and a map generated.

## 6 Experiments

A mapping was made of Maastricht University's Swarmlab using the methods described in this paper. The results are shown in Figure 7. Note that the final pose is shown as the green/red axis and corresponds to the pose in the simulator. For this experiment rotational and translational odometry error were enabled and set to 10% and 40% respectively. The map was generated by moving the robot using tele-operation controls through the environment.

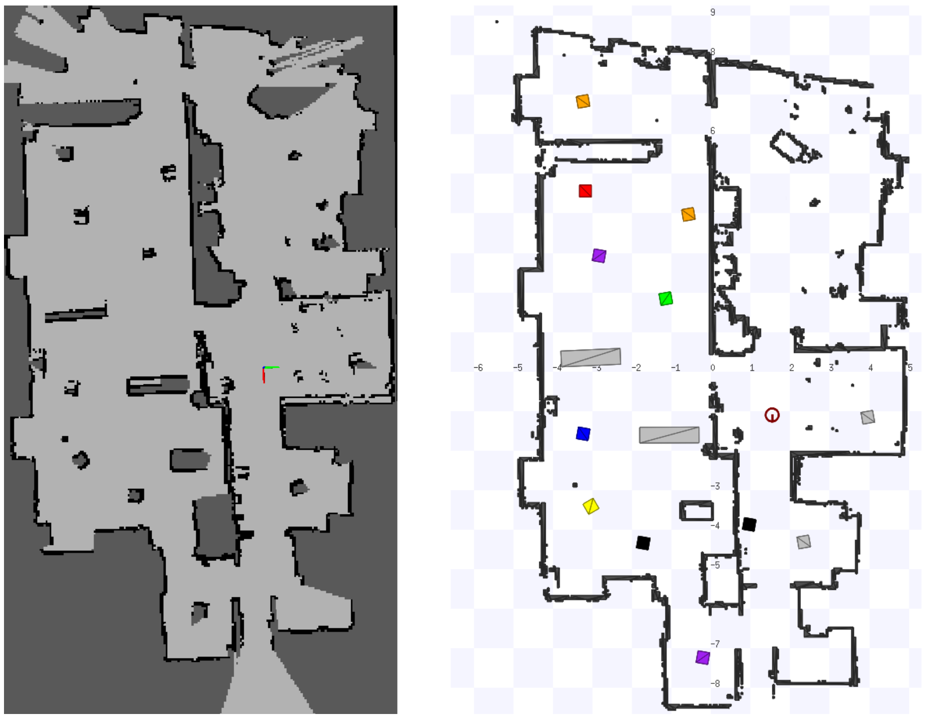

Fig. 7: **Left**: Map of Maastricht University's Swarmlab generated.
**Right**: Actual map in stage simulator

## 7 Conclusion

In this paper, GraphSlam was combined with scan matching and the loop closure detection enhancement. Moreover, an exploration strategy was implemented to provide new sensor data to GraphSLAM. Specific implementation details were provided, describing how the offline GraphSLAM technique was used as an online SLAM method with scan matching and loop closure detection.

By adding scan matching, pose estimates were improved greatly, resulting in better performance of GraphSLAM. The g2o library provided a stable and reliable solution for solving the graph. Loop closure detection increased coherence of the graph, which in turn resulted in better pose estimations.

Experiments were conducted to test the improvement in performance of GraphSLAM when combined with scan matching and the loop closure detection enhancement. Results of these experiments showed that the scan matching was unable to handle large open rooms with few obstacles, due to a lack of reference information provided by the sensor data. In this case, the robot should not perform scan matching and rely on odometry data. In other cases, scan matching did improve the performance of GraphSLAM significantly.

Future work could be done by implementing a detection technique that can decide whether or not scan matching should be performed, given the corresponding sensor data. Another interesting topic would be to combine scan matching with GraphSLAM in three dimensional environments.

## References

[1] A. Censi. An ICP Variant Using a Point to Line Metric. *ICRA, Robotics and Automation*, pages 19–25, 2008.

[2] G. Grisetti, R. Kümmerle, Stachniss C., and Burgard W. A Tutorial on Graph-Based SLAM. *Intelligent Transportation Systems, IEEE*, 2(4):31–43, 2010.

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

[4] Matan Keidar and Gal A. Kaminka. Robot exploration with fast frontier detection: theory and experiments. In *AAMAS*, pages 113–120, 2012.

[5] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A General Framework for Graph Optimization. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, Shanghai, China, 2011.

[6] J. Minguez, Lamiraux F., and Montesano L. Metric-Based Scan Matching Algorithms for Mobile Robot Displacement Estimation. *ICRA, Robotics and Automation*, pages 3557–3563, 2005.

[7] 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.

[8] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents).* The MIT Press, 2005.

[9] Matthew R. Walter, Ryan M. Eustice, and John J. Leonard. Exactly sparse extended information filters for feature-based slam. *Proceedings of the IJ-CAI Workshop on Reasoning with Uncertainty in Robotics*, 26:17–26, 2001.

[10] B. Yamauchi. A frontier-based approach for autonomous exploration. In *Proceedings of the 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, CIRA, pages 146–. IEEE Computer Society, 1997.