

AMERICAN UNIVERSITY OF BEIRUT

Map to Map: From SLAM to CAD Maps and
Back Using Generative Models

by

Rema George Daher

A thesis

submitted in partial fulfillment of the requirements
for the degree of Master of Engineering
to the Department of Mechanical Engineering
of the Maroun Semaan Faculty of Engineering and Architecture
at the American University of Beirut

Beirut, Lebanon
December 2020

AMERICAN UNIVERSITY OF BEIRUT

Map to Map: From SLAM to CAD Maps and Back Using Generative Models

by
Rema George Daher

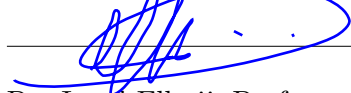
Approved by:



Dr. Daniel Asmar, Associate Professor

Advisor

Mechanical Engineering



Dr. Imad Elhajj, Professor

Member of Committee

Electrical and Computer Engineering



Dr. Elie Shammas, Associate Professor

Member of Committee

Mechanical Engineering

Date of thesis defense: December 15, 2020

Acknowledgements

I would like to acknowledge the support given by the University Research Board at the American University of Beirut. I would also like to thank Theodor Chakhachiro for his unwavering assistance in the brainstorming, implementation, and written part of this thesis.

Contents

Acknowledgements	v
1 Introduction	1
2 Related Work	3
2.1 Parametric Modeling	3
2.2 Machine Learning	5
3 Proposed System	7
3.1 Machine Learning	7
3.2 Parametric Modelling	9
3.2.1 Exteroceptive Error Model	11
3.2.2 Proprioceptive Error Model	13
4 Experiments	20
4.1 Data Generation	20
4.2 Training	25
5 Results and Discussion	27
5.1 Machine learning	27
5.2 Parametric Modelling	49
A	55

List of Figures

2.1	Color oriented image to image applications [1]	6
2.2	Anime to clothing application	6
3.1	Architecture of anime2clothing [2] includes a U-net structured generator G, a real/fake discriminator, and a domain discriminator. The real/fake discriminator takes as input either the real output from the dataset, the fake output generated from the generator, or another output from a different instance of the dataset (falsely associated). The domain discriminator also does this selection but checks if it is associated with the input to the generator. The domain discriminator is responsible for checking if the images are associated or not	8
3.2	Flowchart of the parametric modelling	10
3.3	An example of a probabilistic road map (Blue) with a specified trajectory (Orange)	11
4.1	An example of a 3D world environment in Gazebo	21
4.2	The figure on the right is from the HouseExpo dataset. To the right an example of a generated SLAM map	22
4.3	Results of the generated SLAM maps when the machine learning model was trained on partial maps	24
4.4	The training overview	26
5.1	The generator (G) and discriminator (D) losses for $Model_{AB}$. . .	28
5.2	The generator (G) and discriminator (D) losses for $Model_{BA}$. . .	28
5.3	Test results of $Model_{AB}$	30
5.4	Test results of $Model_{BA}$	31
5.5	Test results of $Model_{BA}$	32
5.6	Test results of $Model_{BA}$	33
5.7	Test results of $Model_{BA}$	34
5.8	Test results of $Model_{BA}$	35
5.9	The flowchart of the error calculation process adopted from [3] . .	37
5.10	The evaluation of the simulated SLAM map against the target CAD map	42

5.11	The evaluation of the generated CAD map from the machine learning method as compared to the target CAD map	47
5.12	The box plot visualization of the errors in Table 5.3	49
5.13	Comparing odometry to map conversion methods using using partial mapping	50
5.14	Comparing Machine Learning to Parametric modelling approaches using mapping of the complete floor	54
5.15	An example of an CAD map with a specific trajectory (left), the displacement field (middle), and its corresponding deformed image (right)	54

List of Tables

5.1	The batch size mapping according to the current resolution	27
5.2	The epoch to current resolution mapping	28
5.3	Evaluation results of the structure and area of the maps generated by <i>Mode_{BA}</i> using the evaluation method of [3]. Here, Error(sim) refers to the evaluation result of the simulated SLAM map with the CAD map. Moreover, Error(gen) refers to the error between the machine learning generated SLAM map and its corresponding CAD map.	
	48	
5.4	Time taken in seconds by the parametric modelling with different conversion of odometry to map error methods	
	51	

Chapter 1

Introduction

SLAM research is embarking on a new path, where focus is directed towards improving SLAM map quality and visual aesthetics. This new interest of improving maps is due to their various applications in localization and beyond. These applications can be found in many fields such as data science, navigation, search and rescue, architecture, intelligent transportation systems, and AR/VR. In this work, a framework will be introduced that automatically generates SLAM maps and another that automatically removes deformations and errors in these maps and generates an improved version.

Direct conversion from CAD maps to robotic maps can act as a faster and more efficient alternative to real time simulations. In robotics, simulations of SLAM maps are needed for research and experimentation on real-time navigation of an explored space, such as in search and rescue [4]. In addition, SLAM simulations are used in employee field training for controlling a robot remotely and map merging in multi robot tasks of known relative locations. Furthermore, these simulations can be used to create datasets for implementing various machine learning SLAM tasks.

In this thesis, we will be utilizing two generative modelling methods, parametric modelling and machine learning. The use of parametric modeling for SLAM map generation allows for a quantitative understanding of the distortions through the automatic generation of local deformation fields. This can be used to create ground truth for evaluating the performance of map alignment techniques as was done in our previous study [5]. However, in the previous work, the ground truth was generated in a more heuristic manner, leaving the ground truth metric less accurate than the one generated in this thesis. In addition, through parametric modelling, multiple possibilities of SLAM maps from one CAD map can be generated using a user defined trajectory. On the contrary, the learning based approach does not take the trajectory into consideration and output one possible hypothesis of the SLAM map without a specific robot trajectory. However, the learning technique allows for a deeper understanding and modeling of the effect of the environment shape on the output error.

In this work, machine learning is also used for the enhancement of SLAM maps to resemble CAD maps. This would be needed in any SLAM application to be able to correct the map after or during its generation. Improving the accuracy of the map would theoretically in turn improve localization and path planning.

The contributions of this work are:

- Creation of a dataset with SLAM maps and corresponding CAD maps
- Automatically transforming an CAD map into a SLAM map and a deformation field from a user defined trajectory using Parametric Modelling. This system adopts various literature models and molds them to be applied for map deformations
- Automatically generating SLAM maps from CAD maps using machine learning
- Automatically generating maps resembling CAD maps from SLAM maps using machine learning. This can be thought of as an "auto-correct" for SLAM maps.

Chapter 2

Related Work

Performing tasks that rely on preexisting robotic maps uses offline robot simulators. There exists a variety of robotic simulators for all sorts of applications. In the case of mobile robotics, simulators such as Gazebo [6] and Stage [7] are used. These particular simulators are available as standalone and as an integrated package with the robotic operating system (ROS). Simulators are used for tasks such as testing algorithms, performing experiments, and designing robots [8]. They offer an accurate and efficient way for research tasks without the need for physically operated robots nor an appropriate indoor/outdoor space to move. This makes it possible to advance in research in a more cost effective way.

The downside of simulators is that they are in need of high memory usage and processing power to be able to run a simulation smoothly. Another inconvenience is the need to actually wait for the simulation to happen in real time. What if we can get the output needed from the simulation without waiting for the robot to perform its task in real time and using very low processing powers?

This work will be focusing on an alternative for robotic simulators in the applications that need prior robotic maps. Such applications include navigation experiments, employee field training, and video-gaming. The point of this alternative is to be able to automatically generate a robotic map with no need for high performance graphics or processor and most importantly without the need to wait for the task to complete in real time. To do so, there are two ways to proceed, either formulate a parametric model or use machine learning. In this work, both options will be developed.

2.1 Parametric Modeling

The first goal of this thesis is to automatically generate a robotic map without simulators using only CAD maps. To do so parametrically, the modeling of SLAM error sources becomes of prime interest. In the literature, this is mainly done for calibration. In this review, an overview will be made on modelling methods

for calibration of exteroceptive and proprioceptive (2D LIDAR and odometry) sensors, which are the sensors that are used in this thesis to generate SLAM maps. The review will focus on calibration methods for differential drive robot, since this work’s simulated SLAM maps are created using Turtlebot3, which is a differential drive robot.

Starting with LIDAR error modelling, some papers focus on 3D LIDAR modelling [9] and on modeling LIDAR in an airborne system [10]. However, in this work, 2D ground LIDARs are used. Furthermore, some papers use a stochastic experimental approach [11] or a Kalman filter approach [12] for calibration. These kind of approaches make the LIDAR error modeling less parametric in nature with the usage of general parameters. That is why, in this thesis, the work of [13] will be adopted, which formulates a model for LIDAR sensors with errors of different sources and contributions.

As for odometry, this work will focus on wheel encoders as the main source of odometry data. [14] presents a literature review on calibration methods for odometry sensors. Calibration papers differ in three aspects, calibration technique and test path, error sources considered, and the simulations and experiments performed along with their performance. In this work, the purpose of researching calibration methods is to model the sensor errors as opposed to calibrating them, which makes the calibration technique and test path not of prime interest. This is why, the review will focus mainly on error sources used and the performance of the methods.

First, the error sources considered will be assessed. [15, 16, 17, 18, 19, 20] included in their formulations uncertainties of the wheelbase and the wheel diameters of the robot. Moreover, [15] didn’t include scaling error of actual average to nominal wheel diameter and assumed independence between wheelbase errors and wheel radius errors. In [16] the effect of non-systematic errors on the calibration is decreased by having the test path as a parameter. As for [18], a dependency between wheel base and wheel radius errors was introduced; however, only straight motion was included and some trigonometric simplifications were assumed as was also done in [15]. On the contrary, [19, 20] did not have a need for trigonometric considerations due to the use of final robot orientation. Furthermore, [17] took into consideration scaling, wheel diameters, and wheelbase errors.

Other methods rely on a more general modelling of the system making them of less importance to the case at hand; in this thesis, the parametric modelling of all possible error sources is of great importance. Such methods include optimization and Kalman filter based methods [21, 22, 23, 24, 25], methods that don’t identify the contribution of the different sources of error [26, 27, 28], and methods that generalize the errors by using corrective factors [29]. In this thesis, the methods with most detailed parametric modelling will be chosen. The search is now narrowed to [18, 19, 20], which consider a dependency between wheel base and wheel radius errors and do not have generalizations.

Moving on to simulations and experiments performed and the performance

in these systems. First, [19] shows that it is more accurate than [18] in terms of calibration. Moreover, the system in [20] uses same error equations as [19], but differs only in path used, which makes both error modelling the same. This is why, in this thesis the odometry error modelling will rely on [20, 19].

It is also worth mentioning that calibration and modeling sensor errors has also been done using machine learning. For example, [30] estimates odometry error using neural networks. However, in this work we will be using machine learning to directly generate a SLAM map from an CAD map.

2.2 Machine Learning

The use of deep learning in mobile robotics has been on the rise in recent literature [31, 32]. Machine learning has been utilized in SLAM for detecting features [33], simplifying maps through object detection [34], predicting shape of these objects (shape completion) [35], and predicting objects that should be used for localization and those that shouldn't (dynamic objects) [36]. It has also been used for understanding scenes through labeling and obtaining graph maps [37], estimating visual odometry (localization) [38], predicting monocular depth [39], predicting dense optical flow [40], and predicting loop closure [41]. An example of a SLAM system that uses deep learning throughout its processes (loop closure, tracking, 3D reconstruction) is DeepSLAM [42].

Specifically, this thesis will contribute to improving SLAM maps and through that open the door to the enhancement of localization and navigation after a better map estimation is made. This is achieved by automatically fixing SLAM maps using a machine learning model trained to generate maps resembling CAD maps from SLAM maps. To do so, a literature review on image to image translation in machine learning is in order.

Image to image translation methods can be divided into those that train on paired images and those that don't. For this application, the dataset generated is paired, narrowing the literature review to only paired training image to image translation methods. Specifically, the input and target images are both 2D occupancy grid maps, which means that in the image translation there is no color change. Color change is usually present in applications such as summer to winter, horse to zebra, labels to street scene, black and white to color, labels to facade, aerial to map, day to night, and edges to photo as shown in Fig. 2.1 used in pix2pix [1].

In the application carried out in this thesis — generating maps resembling CAD maps from SLAM maps — the main transformation that is being made between input and target is piece-wise deformations. One similar application is handwritten digits to printed digits [43]. However, the difference is that there are only 10 digits that need to be learnt, making the architecture much simpler, a classifier outputting a vector of length 10 and not an image. Another appli-

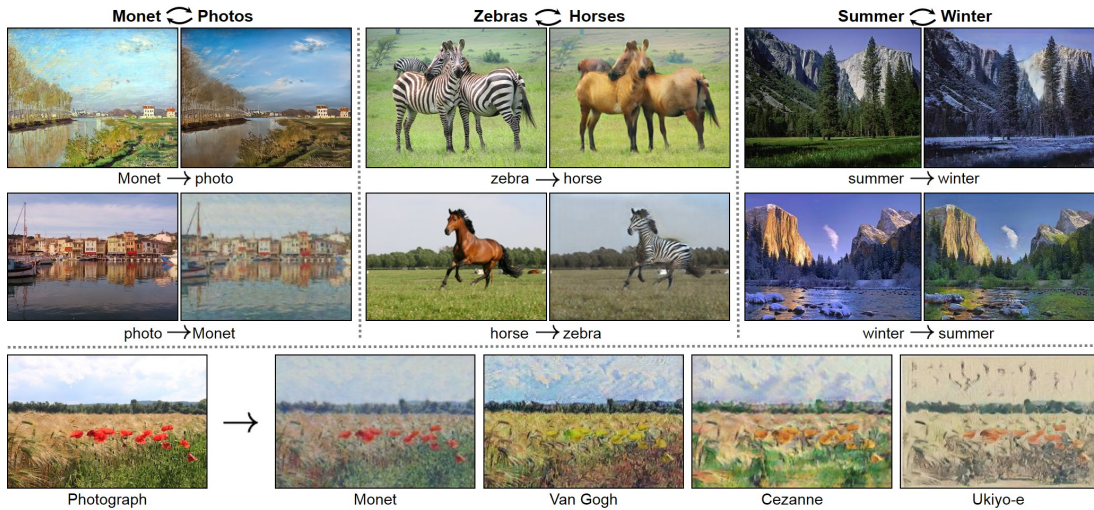


Figure 2.1: Color oriented image to image applications [1]

cation is rectifying fingerprints [44]. This technique needs the dataset to have the displacement field, which is not available in our dataset, since it is generated through real time experiments in simulators. Moreover, geometrically rectifying digital imagery is also similar, but digital imagery deformation is very simplistic and lacks local distortions and noise, which results in a simpler architecture than needed [45, 46].

On the contrary, the complexity of translating an anime image to real life clothing is appropriate to the application of this thesis [2]. In the translation of [2] colors mainly stay the same and local and global deformations are present as shown in Fig. 2.2. There might be a better application out there that is closer to the task at hand, but to the best of our knowledge this is the closest. The state of the art in anime to clothing is [2], which builds its network on the most widely used architecture in image to image translation pix2pix [1], but with a novel consistency loss.



Figure 2.2: Anime to clothing application

Chapter 3

Proposed System

The proposed system is divided into two sections. One section describes the Machine learning approach to model SLAM maps and fix them. The second describes the parametric modelling and generation of SLAM maps.

3.1 Machine Learning

The architecture used in this system is that of anime2clothing [2] shown in Fig. 3.1.

Anime2clothing relies on a GAN architecture. GANs are neural networks that are made up of two sub-models, the generator and the discriminator. The generator model learns to generate new data, while the discriminator classifies them as real or fake. To classify the output as real they have to be close to the input data (closeness is measured using a loss function). Otherwise, they are classified as fake or generated. Therefore, it can be said that these two sub models compete against each other to finally reach an optimal state were the output is new (generator role) but doesn't look fake (discriminator role) [47].

The anime2clothing GAN architecture in Fig. 3.1 is made up of a generator G — with a U-net structure [48] —, a traditional real/fake discriminator, and an additional domain discriminator D_d . The domain discriminator has a multi-scale architecture [49], which handles high-quality images. The multi-scale architecture uses multi discriminators, D_{d1}, D_{d2}, D_{d3} , which are similar except for the difference of image scales.

Spectral normalization [50] is also applied on the discriminators following the Lipschitz constraint. In addition, instances were augmented by adding false targets such as in [51]. False targets are simply targets paired with different inputs. The domain discriminator D_d is responsible for checking if the images are associated or not. On the contrary, the real/fake discriminator [51] D_R handles the quality of the image. Specifically, it is a multi-scale patch discriminator that can receive high-quality images and consists of discriminator $D_{r1}, D_{r2}, and D_{r3}$.

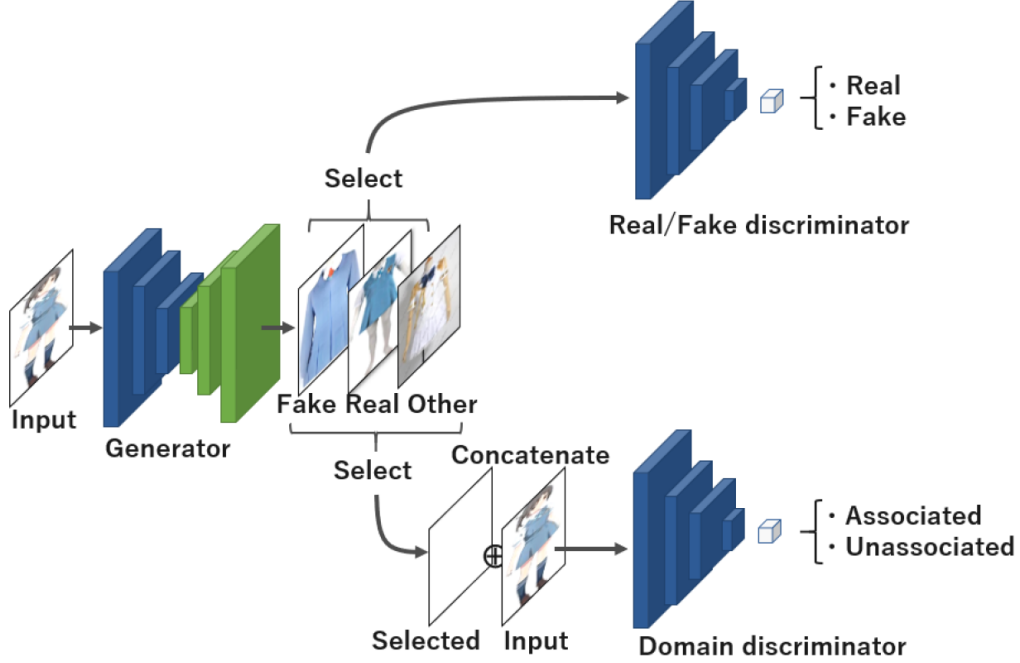


Figure 3.1: Architecture of anime2clothing [2] includes a U-net structured generator G , a real/fake discriminator, and a domain discriminator. The real/fake discriminator takes as input either the real output from the dataset, the fake output generated from the generator, or another output from a different instance of the dataset (falsely associated). The domain discriminator also does this selection but checks if it is associated with the input to the generator. The domain discriminator is responsible for checking if the images are associated or not

Using multiple patches allows for the modelling of the small and large scale shapes in the image. Finally, spectral normalization is also added to D_r .

Consequently, the structure of this network transforms the input image to the target image by relying on a minimax game [52]

$$\min_X \max_Y f \quad (3.1)$$

which is a combination of the input consistency loss, the feature matching loss, and the GAN objectives:

$$\begin{aligned} \min_G (\max_{D_d, D_r} (\sum_{k=1}^3 \mathcal{L}_{GAN_{domain}}(G, D_{d_k}) + \sum_{k=1}^3 \mathcal{L}_{GAN_{real/fake}}(G, D_{r_k})) \\ + \sum_{k=1}^3 \mathcal{L}_{FM_{domain}}(G, D_{d_k}) + \sum_{k=1}^3 \mathcal{L}_{input_{real/fake}}(G, D_{r_k}) + \lambda \mathcal{L}_{L_1}(G)) \end{aligned} \quad (3.2)$$

where λ is the importance weight and \mathcal{L} are the objective functions such that:

$$\mathcal{L}_{GAN_{domain}}(G, D_d) = \mathbb{E}_{(x,y)}[\log(D_d(x, y))] + \mathbb{E}_x[\log(1 - D_d(x, G(x)))] \quad (3.3)$$

$$\mathcal{L}_{GAN_{real/fake}}(G, D_r) = \mathbb{E}_y[\log(D_r(y))] + \mathbb{E}_x[\log(1 - D_r(G(x)))] \quad (3.4)$$

$$\mathcal{L}_{FM_{domain}}(G, D_d) = \mathbb{E}_{(x,y)} \sum_{i=1}^T N_i [\|D_d^{(i)}(x, y) - D_d^{(i)}(x, G(x))\|_1] \quad (3.5)$$

$$\mathcal{L}_{input_{real/fake}}(G, D_r) = \mathbb{E}_x \sum_{i=1}^T N_i [\|D_r^{(i)}(y) - D_r^{(i)}(G(x))\|_1] \quad (3.6)$$

$$\mathcal{L}_{L_1}(G) = \mathbb{E}_{(x,y)} [\|y - G(x)\|_1] \quad (3.7)$$

From these objective functions, the $\mathcal{L}_{GAN_{real/fake}}$ is the traditional GAN objective function. $\mathcal{L}_{GAN_{domain}}$ is also a minimax game for image associations using augmented unassociated data. In addition, the feature matching loss, $\mathcal{L}_{FM_{domain}}$, has the aim of generating an image closer to a corresponding real one, it is computed as the L1 loss between outputs of the intermediate layers of the domain discriminator. Furthermore, the input Consistency loss $\mathcal{L}_{input_{real/fake}}$, maintains shape and color, it is computed as the L1 loss between outputs of the intermediate layers of the real/fake discriminator. Moreover, the L1 loss, \mathcal{L}_{L_1} , maintains shape and color. Here, x and y are the input and output image respectively and T is the number of layers in the discriminator. Finally, N_i is the number of elements in each layer and $D^{(i)}$ is the i th layer of the discriminator.

Using a coarse-to-fine scheme [53], the generator and discriminators progressively grow by increasing systematically the resolution of the input image depending of the current number of epochs reached. This way, any gradient problems from high resolution images are alleviated, and the computation period is shortened. This model is denoted as $Model_{prog}$.

In this thesis, another model is created without progression of image resolution $Model_{32}$. This model trains the images at 32 resolution to capture only the global structure of the input image and not the details. After that, the output images are post-processed to increase their resolution and reduce blurriness.

3.2 Parametric Modelling

For the parametric modelling, the user defines a trajectory first. Then, the exteroceptive and proprioceptive error models are implemented with the use of the trajectory, CAD map, and the original pixel locations. The proprioceptive error model is later converted to a map specific error. The deformation fields, two matrices of pixel-wise location displacements in the x and y directions (d_x, d_y), from both proprioceptive and exteroceptive models are added up to generate the modelled SLAM map. This process is shown in Fig. 3.2. The contribution in this system is the use of the various equations from the different literature and

apply them to model the deformations of the map instead of the deformations in the trajectory.

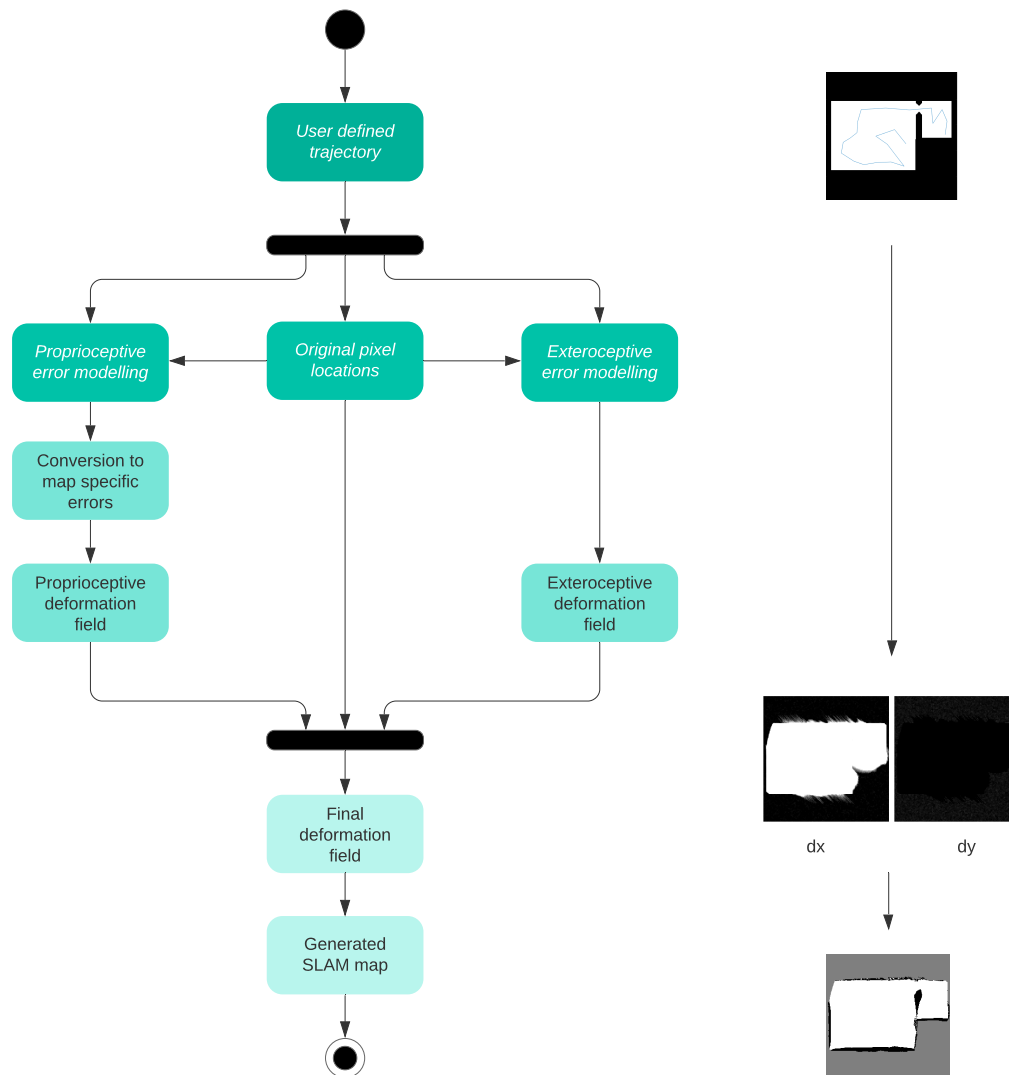


Figure 3.2: Flowchart of the parametric modelling

First, a probabilistic road map, which is a graph showing all possible trajectories in an occupancy grid map — the input 2D CAD map — is generated. An example of such a road map can be seen in Fig. 3.3 in blue. This is done using the method described in [54] for randomly placed nodes in the unoccupied space of the map. From this graph a path is chosen between two random points (or user defined path). An example of such a path is shown in red in Fig. 3.3. The orientation, velocity, and acceleration are then calculated using waypoints and time of arrival information.

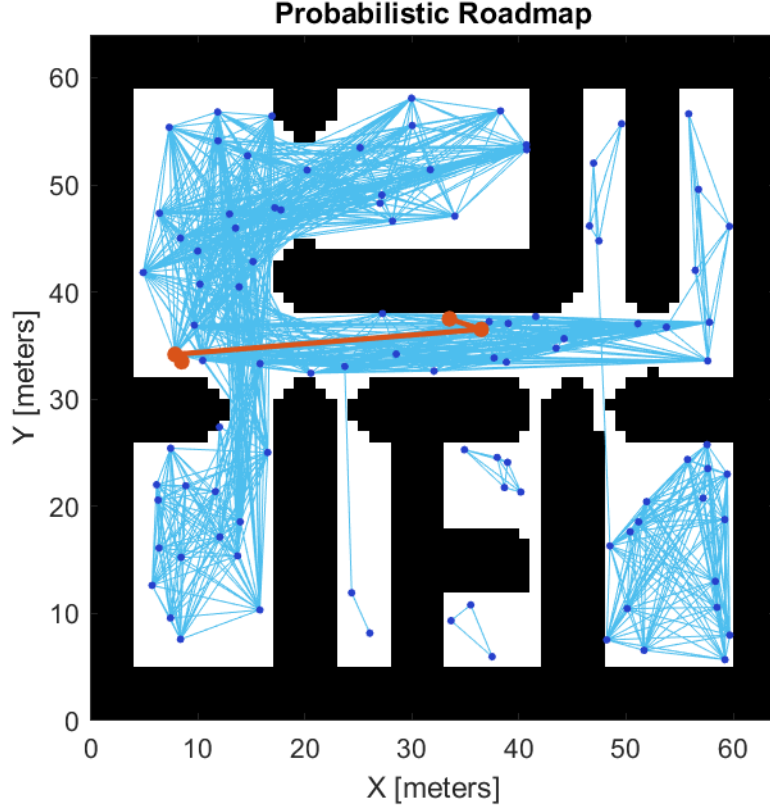


Figure 3.3: An example of a probabilistic road map (Blue) with a specified trajectory (Orange)

3.2.1 Exteroceptive Error Model

The exteroceptive sensor (2D LIDAR) error is modelled by first defining the landmark coordinates detected by the LIDAR using the following form:

$$X_L = \begin{bmatrix} X_l \\ Y_l \end{bmatrix} = D \begin{bmatrix} \cos\alpha \\ \sin\alpha \end{bmatrix} \quad (3.8)$$

where, D is the LIDAR measured slant range in meters, which is randomly assigned to every pixel in the image. Furthermore, α is the horizontal angle, which is assigned randomly to each pixel. D and α values are then smoothed using a median filter, which helps in giving neighboring pixels similar range and angle readings. However, D and α are both subject to errors. The error term for D is:

$$\Delta D = a_0 + a_1 D \quad (3.9)$$

$$\Delta \alpha = b_1 \alpha + b_2 \sin\alpha + b_3 \cos\alpha + b_4 \sin 2\alpha + b_5 \cos 2\alpha + b_6 D^{-1} \quad (3.10)$$

here, a_0 represents the measured origin's shift and a_1 represents the effect of counter frequency deviations as a scale error. In addition, b_1 is the scale error from the encoder, b_2 and b_3 depict the horizontal eccentricity, b_4 and b_5 represent the non-orthogonality, and b_6 models the collimation axis eccentricity. Given the values calculated in [13], we can impose a bound on these parameters.

The deformation field (d_{xhit}, d_{yhit}) due to the latter error formulation becomes:

$$d_{xhit} = pm(D\cos(\alpha) - (D - \Delta D)\cos(\alpha - \Delta\alpha)) \quad (3.11)$$

$$d_{yhit} = pm(D\sin(\alpha) - (D - \Delta D)\sin(\alpha - \Delta\alpha)) \quad (3.12)$$

Here, pm is how many pixels there are per 1 meter in the images portraying the maps. The error formulated above can be considered as small measurement noise. However, it is shown in [55] that three other errors remain, which are errors due to random unexplained noise E_{rand} , unexpected dynamic objects E_{short} , and failures to detect objects E_{max} . The probability distributions of such measurements is:

$$P_{short} = \begin{cases} \frac{1}{1-e^{-\lambda z_k^{t*}}} \lambda e^{-\lambda z_k^t} & \text{if } 0 \leq z_k^t \leq z_k^{t*} \\ 0 & \text{otherwise} \end{cases} \quad (3.13)$$

$$P_{max} = \begin{cases} 1 & \text{if } z_k^t = z_{kmax}^t \\ 0 & \text{otherwise} \end{cases} \quad (3.14)$$

$$P_{rand} = \begin{cases} 1/z_{kmax}^t & \text{if } 0 \leq z_k^t \leq z_{kmax}^t \\ 0 & \text{otherwise} \end{cases} \quad (3.15)$$

λ is an intrinsic parameter by which the exponential curve's slope is obtained ($\lambda = 0.5$ [56]). P_{short} is assigned to have a random value for every landmark. z_k^{t*} is the true object range, and is assigned the values of D . From P_{short} we calculate z_k^t and E_{short} :

$$z_k^t = \frac{-\log\left(\frac{P_{short}(1-e^{-\lambda z_k^{t*}})}{\lambda}\right)}{\lambda} \quad (3.16)$$

$$E_{short} = z_k^{t*} - z_k^t \quad (3.17)$$

$$d_{xshort} = pm(D\cos(\alpha) - E_{short}\cos(\alpha - \Delta\alpha)) \quad (3.18)$$

$$d_{yshort} = pm(D\sin(\alpha) - E_{short}\sin(\alpha - \Delta\alpha)) \quad (3.19)$$

From the calculated z_k^t , P_{max} and P_{rand} are also calculated from equations 3.14 and 3.15. With Ran_{err} and Max_{err} assigned a random error between an experimentally calculated range, it follows that:

$$z_{kmax}^t = \max(z_k^t) \quad (3.20)$$

$$E_{max} = \begin{cases} Max_{err} P_{max} & \text{if } z_k^t = z_{kmax}^t \\ 0 & \text{otherwise} \end{cases} \quad (3.21)$$

$$d_{xmax} = pm(D\cos(\alpha) - E_{max}\cos(\alpha - \Delta\alpha)) \quad (3.22)$$

$$d_{ymax} = pm(D\sin(\alpha) - E_{max}\sin(\alpha - \Delta\alpha)) \quad (3.23)$$

$$E_{rand} = \begin{cases} Ran_{err} P_{rand} & \text{if } 0 \leq z_k^t < z_{kmax}^t \\ 0 & \text{otherwise} \end{cases} \quad (3.24)$$

$$d_{xrand} = pm(D\cos(\alpha) - E_{rand}\cos(\alpha - \Delta\alpha)) \quad (3.25)$$

$$d_{yrand} = pm(D\sin(\alpha) - E_{rand}\sin(\alpha - \Delta\alpha)) \quad (3.26)$$

Finally, the final displacement field becomes:

$$d_{xL} = Z_{hit}d_{xhit} + Z_{short}d_{xshort} + Z_{max}d_{xmax} + Z_{rand}d_{xrand} \quad (3.27)$$

$$d_{yL} = Z_{hit}d_{yhit} + Z_{short}d_{yshort} + Z_{max}d_{ymax} + Z_{rand}d_{yrand} \quad (3.28)$$

$Z_{hit}, Z_{short}, Z_{max}, Z_{rand}$ are the relative error weights and are equal to (0.4, 0.3, 0.2, 0.1), respectively [56]. Moreover, the recalculated measure of z_k^t and α , z_{meas} and α_{meas} becomes:

$$z_{meas} = D - d_{xL}/\cos(\alpha - \Delta\alpha) \quad (3.29)$$

$$\alpha_{meas} = \alpha - \Delta\alpha \quad (3.30)$$

3.2.2 Proprioceptive Error Model

As for the proprioceptive (odometry) error modelling, from the trajectory the nominal (without error) positions x_{nom}, y_{nom} are extracted. In addition, angle of rotation of the wheels γ , incremental displacement of the robot $\Delta d_{Rnom}, \Delta d_{Lnom}$, and distance traveled for every straight line in trajectory L are also calculated from the trajectory data. From the selected robot (Turtlebot3), the nominal wheelbase b_{nom} and the diameter of the right and left wheels are given D_{Rnom}, D_{Lnom} . D_a is the average wheel diameter and is assumed to be constant. Furthermore, E_b and E_d are the wheelbase and diameter error parameters, respectively. Moreover, $\Delta\theta$ and Δd are the heading angle and displacement, respectively, between successive poses. R is the radius of the curvature made by the trajectory due to various error sources. From [20, 19] the values of $\alpha_{E_b}, \alpha_{E_d}, \beta$ are assigned their experimental standard deviation. The parameters described can be modelled as follows:

$$E_b = \frac{90}{90 - (\alpha_{E_b} + \alpha_{E_d})} \quad (3.31)$$

$$b_{actual} = E_b b_{nominal} \quad (3.32)$$

$$R = \frac{L/2}{\sin(\beta/2)} \quad (3.33)$$

$$E_d = D_{Ract}/D_{Lact} = \frac{R + \frac{b_{actual}}{2}}{R - \frac{b_{actual}}{2}} \quad (3.34)$$

$$D_a = (D_{Rnom} + D_{Lnom})/2 \quad (3.35)$$

From equations of D_a and E_d :

$$D_{Lact} = \frac{2}{E_d + 1} D_a \quad (3.36)$$

$$D_{Ract} = \frac{2}{(1/E_d) + 1} D_a \quad (3.37)$$

$$\Delta d_{Ract} = \gamma D_{Ract} \quad (3.38)$$

$$\Delta d_{Lact} = \gamma D_{Lact} \quad (3.39)$$

$$\Delta d_{act} = \frac{\Delta d_{Ract} + \Delta d_{Lact}}{2} \quad (3.40)$$

$$\Delta \theta_{act} = \frac{\Delta d_{Ract} - \Delta d_{Lact}}{b_{actual}} \quad (3.41)$$

$$\Delta x_{act} = \Delta d_{act} \cos(\theta_{k-1} + \Delta \theta_{act}/2) \quad (3.42)$$

$$\Delta y_{act} = \Delta d_{act} \sin(\theta_{k-1} + \Delta \theta_{act}/2) \quad (3.43)$$

The same is derived for the nominal counterparts

$$\Delta d_{nom} = \frac{\Delta d_{Rnom} + \Delta d_{Lnom}}{2} \quad (3.44)$$

$$\Delta \theta_{nom} = \frac{\Delta d_{Rnom} - \Delta d_{Lnom}}{b_{nominal}} \quad (3.45)$$

$$\Delta x_{nom} = \Delta d_{nom} \cos(\theta_{k-1} + \Delta \theta_{nom}/2) \quad (3.46)$$

$$\Delta y_{nom} = \Delta d_{nom} \sin(\theta_{k-1} + \Delta \theta_{nom}/2) \quad (3.47)$$

In order to get the effect of the odometry error on the map, the covariance matrix has to be generated. The covariance terms are nothing but the error for its respective parameter. The error associated with x is the only one with a detailed

derivation. The other error terms are generated in a similar manner:

$$\sigma_x = \Delta x_{nom} - \Delta x_{act} \quad (3.48)$$

$$\sigma_y = \Delta y_{nom} - \Delta y_{act} \quad (3.49)$$

$$\sigma_\theta = \Delta \theta_{nom} - \Delta \theta_{act} \quad (3.50)$$

$$\sigma_{xy} = \Delta xy_{nom} - \Delta xy_{act} \quad (3.51)$$

$$\sigma_{x\theta} = \Delta x\theta_{nom} - \Delta x\theta_{act} \quad (3.52)$$

$$\sigma_{y\theta} = \Delta y\theta_{nom} - \Delta y\theta_{act} \quad (3.53)$$

Conversion to Map Specific Error

For the conversion of the odometry error to its effect on the map, the extended Kalman filter (EKF) SLAM equations are used [57]. However, it was calculated that using these equations may render the system time consuming and not advantageous over simulations. This is due to the use of the inverse of the covariance matrix that nonlinearly increases in size as the robot moves. That is why another method proposed in [58] was also used and later compared with the EKF method.

- **EKF Method**

The EKF method makes use of the matrix A , which is defined as the Jacobian of the prediction model, and J_x and J_z , the SLAM specific Jacobians. Δt is the change in thrust and R and Q are the range measurement and process Gaussian noise, respectively. Also, x, y, θ denote the position and orientation of the robot as measured by odometry. In addition, X is defined as the state vector made up of x, y, θ and landmark poses λ_x, λ_y . With that in mind, the following holds:

$$\Delta \theta = \sigma_\theta \quad (3.54)$$

$$\Delta thrust = \sigma_y / \sin(\theta) \quad (3.55)$$

$$\Delta x = \Delta thrust \times \sin(\theta) \quad (3.56)$$

$$\Delta y = \Delta thrust \times \cos(\theta) \quad (3.57)$$

Initially, the covariance matrix P includes only the robot covariance matrix P^{rr} :

$$P = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{x\theta} \\ \sigma_{yx} & \sigma_y^2 & \sigma_{y\theta} \\ \sigma_{\theta x} & \sigma_{\theta y} & \sigma_\theta^2 \end{bmatrix} \quad (3.58)$$

$$P^{rr} = P \quad (3.59)$$

Later, the covariance matrix P will be augmented to include robot to landmark cross covariance matrices for every landmark P^{ri} of size (2x3). these matrices are concatenated in the first 3 columns. Symmetrically, P^{ir} of size

(3x2) are situated in the first 3 rows. As for the diagonal, the landmark covariance matrices P^{ii} of size (2x2) are concatenated. In the remaining spots of the matrix P , the cross covariances between different landmarks are added.

- Step 1: Update robot state r . Each landmark instance is represented by i . The symbol P^{rr} is the top left 3x3 matrix of state covariance matrix P .

$$A = \begin{bmatrix} 1 & 0 & -\Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{bmatrix} \quad (3.60)$$

$$Q = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} & \sigma_{x\theta} \\ \sigma_{xy} & \sigma_y^2 & \sigma_{y\theta} \\ \sigma_{x\theta} & \sigma_{y\theta} & \sigma_\theta^2 \end{bmatrix} \quad (3.61)$$

$$P^{rr} = AP^{rr}A + Q \quad (3.62)$$

$$P^{ri} = AP^{ri} \quad (3.63)$$

- Step 2: For every reobserved landmark, the Kalman Gain K is calculated and the state vector X is updated. To do so, the following is also updated:

$$r = z_{meas} \quad (3.64)$$

$$\delta = \alpha_{meas} \quad (3.65)$$

$$z = (r, \delta) \quad (3.66)$$

$$c = a1 \quad (3.67)$$

$$d = b1 \quad (3.68)$$

$$R = \begin{bmatrix} rc & 0 \\ 0 & \delta d \end{bmatrix} \quad (3.69)$$

In addition, the change in range and bearing are reflected in the Jacobian of the measurement model $H \in \mathbb{R}^{2 \times (3+2k)}$, where k is the total number of observed landmarks. The first three columns of H are updated for each robot pose as follows:

$$H^{2 \times 3} = \begin{bmatrix} (x - \lambda_x) & (y - \lambda_y) & 0 \\ (\lambda_y - y) & (\lambda_x - x) & -1 \end{bmatrix} \quad (3.70)$$

Also for each reobserved landmark, the H matrix is updated in the following manner, where H^k denotes the k^{th} 2×2 matrix corresponding to the reobserved landmark:

$$H^k = -H^{2 \times 2} \quad (3.71)$$

It is worth noting that at each iteration over the number of reobserved landmarks, only the k^{th} term is updated while the other $k - 1$, 2×2 sub-matrices in H are set to zero.

$$K = PH^T(HPH^T + R)^{-1} \quad (3.72)$$

$$X = X + K(z - h) \quad (3.73)$$

where z and h are the range and bearing of the LIDAR and odometry measurement, respectively.

- Step 3: Add new landmarks to the current state and update the covariance to include the new landmark matrices. Here $N+1$ is used instead of i to denote that there is a new landmark, which would increase the matrix size by 1 with the following updated:

$$J_{xr} = \begin{bmatrix} 1 & 0 & -\Delta x \\ 0 & 1 & \Delta y \end{bmatrix} \quad (3.74)$$

$$J_z = \begin{bmatrix} \cos(\theta + \Delta\theta) & -\Delta t \sin(\theta + \Delta\theta) \\ \sin(\theta + \Delta\theta) & \Delta t \cos(\theta + \Delta\theta) \end{bmatrix} \quad (3.75)$$

$$P^{N+1N+1} = J_{xr} P J_{xr}^T + J_z R J_z^T \quad (3.76)$$

Robot – landmark covariance

$$P^{rN+1} = P^{rr} J_{xr}^T \quad (3.77)$$

$$P^{N+1r} = (P^{rN+1})^T \quad (3.78)$$

Landmark – landmark covariance

$$P^{N+1i} = J_{xr} (P^{ri})^T \quad (3.79)$$

$$P^{iN+1} = (P^{N+1i})^T \quad (3.80)$$

Finally, the difference between the original and final state of each observed pixel is saved in a matrix form as (d_{xO}, d_{yO}) , which is the odometry induced map deformation field.

- **Method of [58]**

As for the method proposed in [58], using experimental data collected in several samples, the linear and angular odometry errors are calculated using the following equations:

$$E_{lin}(\Delta l) = 0.09\Delta l + \sigma \quad (3.81)$$

$$E_{ang}(\Delta\theta) = 0.095\Delta\theta + \alpha \quad (3.82)$$

where $\Delta l = \sigma_p = \sigma_x / \cos(\sigma_\theta)$; and $\Delta\theta = \sigma_\theta$ are the odometry estimated linear and angular displacements respectively. $\sigma = mean(\sigma_p)$ is the average linear error from rotation, and $\alpha = mean(\sigma_\theta)$ the average angular error caused by a linear movement.

Then, at every robot pose a *mask* is created of the location of the pixels in the field of view of the robot.

$$\sigma_{mz} = z_{meas}(mask) \times \eta + E_{lin}(\Delta l) + \mathcal{N}(0, \epsilon_{lin}) \quad (3.83)$$

$$\sigma_{m\theta} = \beta/2 + E_{ang}(\Delta\theta) + \mathcal{N}(0, \epsilon_{ang}) \quad (3.84)$$

where σ_{mz} and $\sigma_{m\theta}$ are the map errors affected by LIDAR and odometry measurements. In addition, z_{meas} is the LIDAR measurements θ , respectively. η is an error factor of exteroceptive sensors typically about 1% and β is the laser beam's aperture angle. As for the normally distributed noise with variances ϵ_{ang} and ϵ_{ang} , it will be assumed to be zero. That is because they have already been included in the LIDAR modelling part.

For every reobserved landmark in iterative poses, the map error is added to generate the deformation fields of the observed pixels in the map d_{xO} and d_{yO} , such that:

$$\sigma_{mx} = \sigma_{mz} \cos(\sigma_{m\theta}) \quad (3.85)$$

$$\sigma_{my} = \sigma_{mz} \sin(\sigma_{m\theta}) \quad (3.86)$$

$$d_{mx} = \sigma_{mx} \quad (3.87)$$

$$d_{my} = \sigma_{my} \quad (3.88)$$

$$d_{xO}(mask) = d_{xO}(mask) + d_{mx} \quad (3.89)$$

$$d_{yO}(mask) = d_{yO}(mask) + d_{my} \quad (3.90)$$

The addition in equations 3.89 and 3.90 is considered an approximation. This is opposed to the detailed modeling of the effect of reobserved landmarks on the error in the EKF method. The EKF method includes the

effect of reobserving landmarks in its covariance matrix with its landmark to landmark and landmark to robot cross relations.

This method has a more general and less accurate parametric solution for the conversion than the EKF method. However, the EKF method uses SLAM equations and might take a long time since similar equations are used in simulations and that is what this work is trying to give an alternative to.

The deformation fields (d_{xL}, d_{yL}) and (d_{xO}, d_{yO}) are then added to form (d_x, d_y) which is used to deform the map accordingly.

Chapter 4

Experiments

For all simulations Gazebo and Turtlebot3 Waffle Pi robot were used with LDS-01 LIDAR, which is a 2D laser scanner capable of sensing 360 degrees of data to use for SLAM. However, for the purpose of showing more deformations in the maps we limited the sensing range to 180 degrees. To elaborate, with a limited field of view the observed features are limited and reobserving them is delayed until a large loop is closed and with a higher feature count the state estimation has a reduced error bound.

For the parametric modelling, Matlab [59] was used on a Toshiba Satellite Laptop with an Intel core i7-5500U CPU with 2.40GHz and 8GB RAM. As for the machine learning method, the training was done on Nvidia V100 PCI-E GPU with 128GB RAM for $Model_{AB}$ and 15GB RAM with $Model_{BA}$, which used half the batch rate.

For the machine learning method, the data was first generated and cleaned. Later the models were trained using two different techniques.

4.1 Data Generation

The data generation for machine learning relies on the HouseExpo dataset CAD maps (35,000 maps) provided by [60]. These maps were converted into 3D environments as shown in Fig. 4.1. This was done using stl tools [61]. After the STL files are generated, model and world files (files read by Gazebo) are automatically created with predetermined 3D environment variables. The whole process of 3D environment generation needed high memory to handle the 3D files. The platform used for the data generation was the American University of Beirut's (AUB) HPC cluster was used with AMD EPYC™ 7551P vCPU that has a maximum boost clock of up to 3.0GHz and 64GB RAM, 30GB of which were used to run the task at hand.

After generating the world files that are compatible with Gazebo, the same vCPU specifications are used to run parallel batch scripts for robotic exploration.

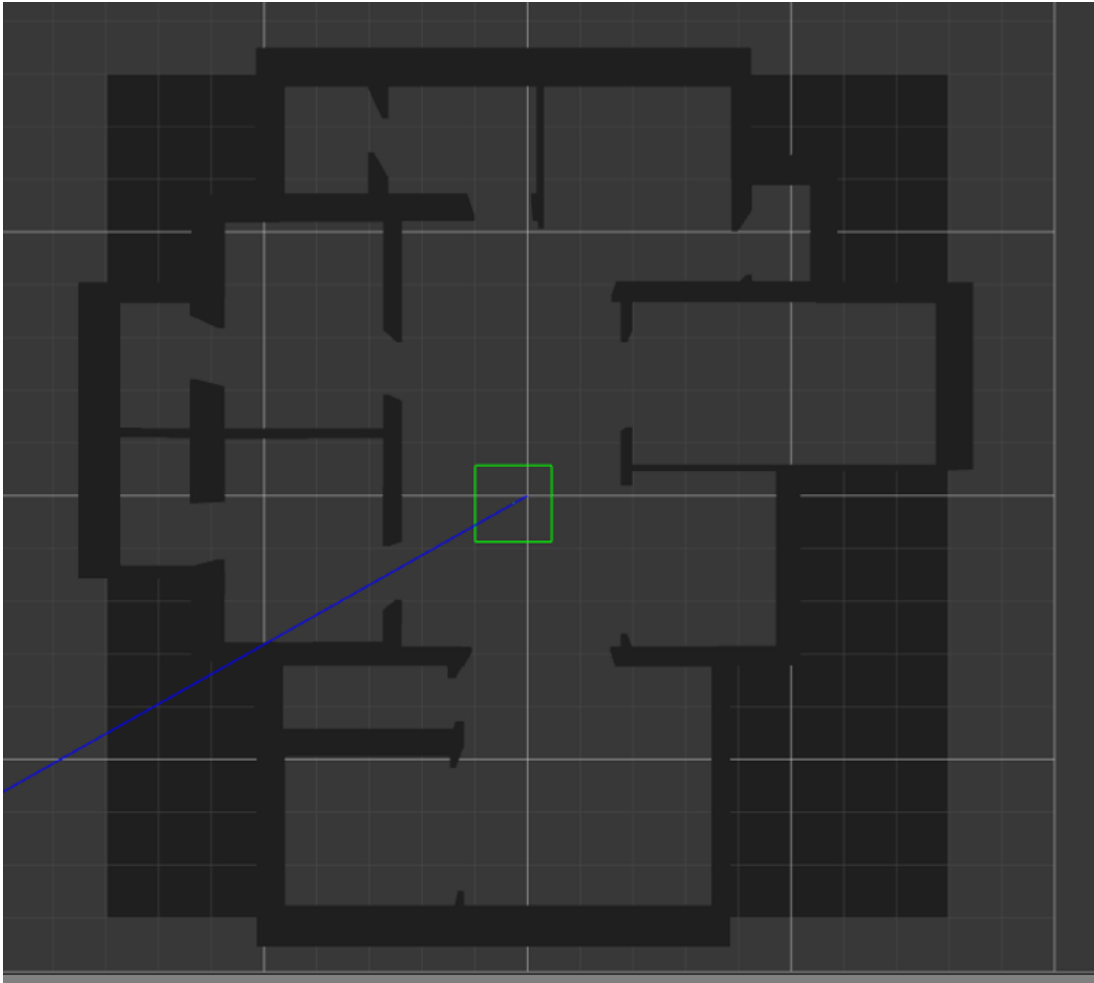


Figure 4.1: An example of a 3D world environment in Gazebo

These batch scripts contain code that performs an automatic robotic exploration of the generated world files in Gazebo and saves the resulting map. The script included launching Turtlebot3 with the world file in Gazebo, Gmapping SLAM for Turtlebot3, exploration from Explore Lite package [62], and saving the map with Map Server [63].

For every 3D environment, 7 minutes were allocated for the exploration. There are 35,000 maps, 7 minutes allocated time for each map, 1 day limit for each node, and an average of 10 job scripts running at the same time depending on other users and the queue position. Knowing that, 166 parallel scripts were run on different nodes making the total run time of generating SLAM maps approximately 17 days. An example of the generated maps is shown in Fig. 4.2.

Due to the constraint random movements applied in the map exploration stage, the SLAM maps generated do not have the same robot trajectory. That is why, when training the network to generate a SLAM map, it would be generating

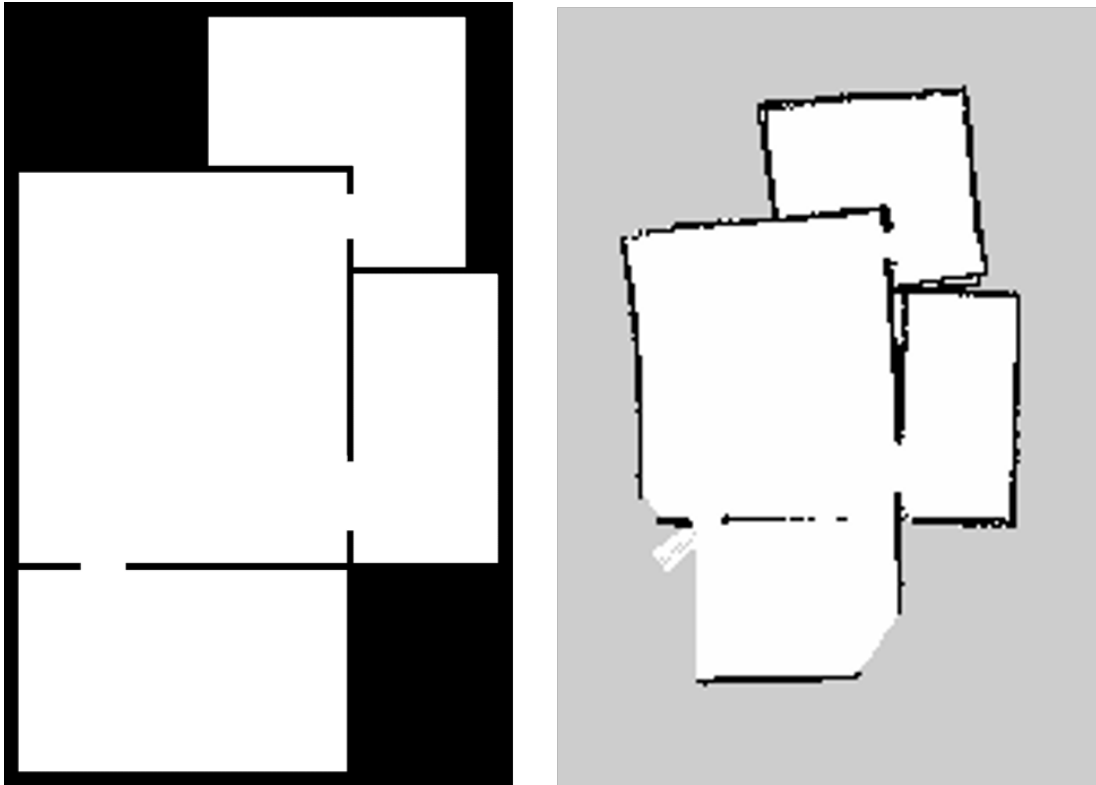


Figure 4.2: The figure on the right is from the HouseExpo dataset. To the right an example of a generated SLAM map

only "one possible hypothesis" of a SLAM map. Keeping this in mind, the model would be learning the effect of the shape of the CAD map on the output of the SLAM system.

At this point, the generated SLAM maps with their CAD map counterparts are in the form of images, which constitute the dataset for the training. However, before training the images were processed by:

- squaring and resizing to 256x256 resolution to achieve uniformity in the dataset size.
- changing the pixel values of the CAD maps to those used in the SLAM maps to avoid color differences between the input and output of the model; only deformation changes are of importance in this work.
- thickening the occupied edges of the CAD maps, since thick walls make it harder to lose data with a progressive architecture in terms of image resolution.
- removal of maps where the robot got stuck and no explorations were made, since getting stuck is not part of what is being modeled in this work. This

was done in 2 steps. The first step included an automatic deletion of generated maps that had a high black to white pixel count (occupied to unoccupied), which indicates that the robot got stuck at the beginning of the exploration near a wall. Whereas, if the exploration went smoothly, the black pixels (occupied), being mostly walls, are much less in count than the white (unoccupied) pixels. The second step included manual cleaning by visually assessing the maps and removing those where the robot got stuck and the map was very small compared to the CAD map and had a high black to white pixel count. After the cleaning, the dataset size became 24,991 instances.

Further cleaning was made to remove partially explored maps after results showed that the generated maps are not corresponding to the input maps as shown in Fig. 4.3. To expedite the cleaning process that could have taken up to 10 days, a small set of 1,468 instances were cleaned manually and 15 specific features of these images were tabulated including:

- black, grey, and white pixel count of SLAM maps
- black, grey, and white pixel count of CAD maps
- difference in black, grey, and white pixel count between SLAM and CAD maps
- difference in black to white, grey to white, black to grey pixels ratio between SLAM and CAD map maps
- difference in number of holes between SLAM and CAD maps calculated using Moore-Neighbor tracing algorithm [64], which was modified by Jacob Eliosoff’s stopping criteria [64]
- grey to white neighbor count in SLAM map
- grey to white neighbor count in CAD maps
- difference in grey to white neighbor count between SLAM and CAD maps
- complete or partial

After generating the 1,468 instance dataframe, an Extra Trees classifier [65] was trained with 11 as the max features and 100 as the number of estimators. All other parameters were left as default. Then the output with probability of being a complete image < 0.4 was discarded as partial (18,292 instances), > 0.8 were complete and there was no need to clean them (1,794 instances), and ≥ 0.4 and ≤ 0.8 were cleaned manually (3,905 instances). This process reduced the dataset to 4,225 image pairs.

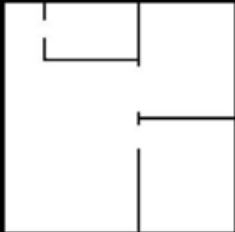


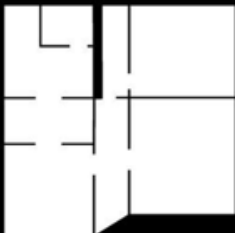

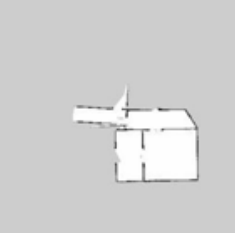
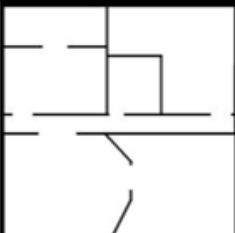
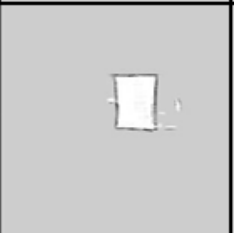
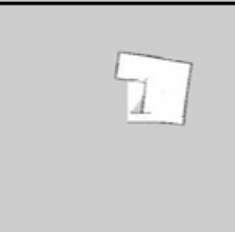
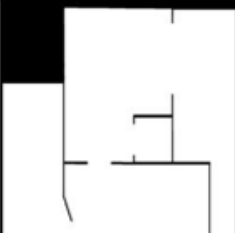
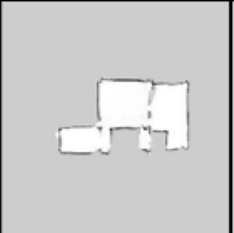
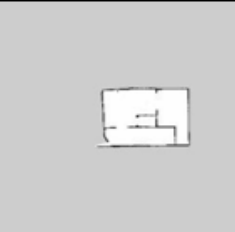
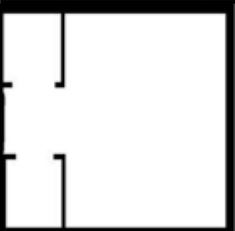
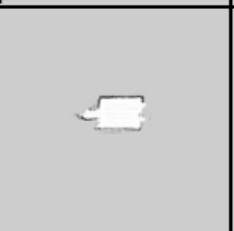
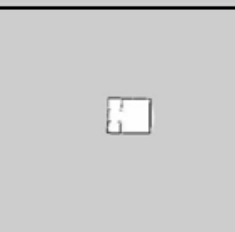
As-built floor plan	Generated SLAM map	Simulated SLAM map
		
		
		
		
		

Figure 4.3: Results of the generated SLAM maps when the machine learning model was trained on partial maps

The idea behind cleaning was that this reduction would make the training more accurate and shorter. One might argue that since the dataset is smaller, the training results might overfit. However, in many image to image applications, such a number is sufficient for learning. For example, when pix2pix was trained

on Google maps scraped image labelling in their Github repository, only 1,096 images were used [1].

As a final cleaning step, the 2D CAD map images were cropped to have less grey areas, which make the images easier to visually asses.

As a result, a dataset is obtained and consists of 2D CAD map images denoted as (A) and 2D SLAM occupancy grid map images denoted as (B). For the training step, the data was shuffled and split into 4,224 training pairs and 21 testing pairs.

4.2 Training

All training is performed using the Adam optimizer [66] with the initial learning rate of 0.0002 and with (0.55, 0.99) momentum parameter values. After 70% of the training epochs are completed, the learning rate starts decaying over the remaining 30% of the epochs.

The network is first trained on the generating B from A (input A, output B). In addition, the network is trained in a different direction, where the input is B and the target is A, making it possible to generate CAD maps from erroneous SLAM maps. The B to A direction is a way of fixing SLAM map errors; whereas the A to B direction is a way of modelling SLAM maps.

For the A to B direction $Model_{AB}$, $Model_{prog}$ was used. As for the B to A direction $Model_{BA}$, $Model_{32}$ was used, since the output that should resemble CAD maps should inherit only the global shape of the SLAM map B without the noise and the deformations. In addition, the preprocessing step, which includes cropping, color jitter, and flipping the dataset to augment it, is removed. This was done to avoid the deletion of global features from the maps. Finally, to further emphasize the global features, the occupied edges of the target images were thickened. The training is shown in Fig. 4.4.

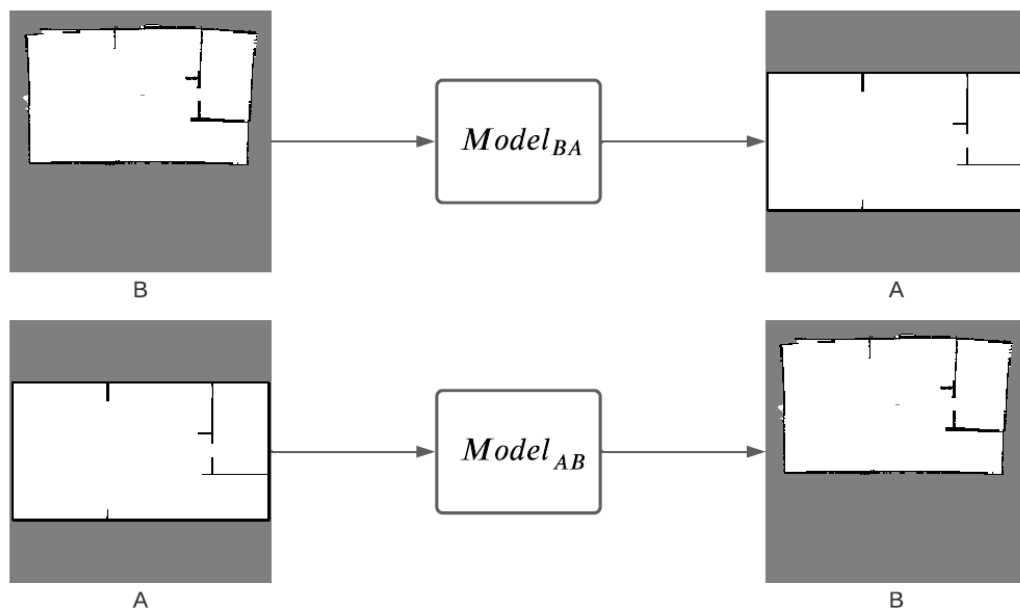


Figure 4.4: The training overview

Chapter 5

Results and Discussion

5.1 Machine learning

The *batch_rate* used for *Model_{AB}* was 1 and for *Model_{BA}* was 0.5. The batch size followed the mapping $batch_rate * batch_mapping$, where the mapping is shown in Table 5.1.

Table 5.1: The batch size mapping according to the current resolution

current resolution	batch_mapping
4	128
8	64
16	128
32	64
64	32
128	16
256	4

To assess the output, the model losses are first analyzed. As shown in Fig. 5.1 and 5.2, the generator and discriminator losses for both models reach a plateau at the end, which is the norm for GANs when they have reached an optimal equilibrium. This is the case since the generator and discriminator work against each other so as one increases the other decreases.

These losses follow different trends for *Model_{AB}* as compared to *Model_{BA}*, since *Model_{AB}* follows a progressive resolution change, while *Model_{BA}* has a constant image resolution. In *Model_{AB}*, the epoch to current resolution mapping is tabulated in Table 5.2. This mapping explains the jumps at the epochs where resolution changes in Fig. 5.1.

From Fig. 5.1 and Table 5.2, we can see that with every shift in resolution (for example at epoch 22, 37 and 67), the losses of *Model_{AB}* make a sharp increase

Table 5.2: The epoch to current resolution mapping

current resolution	epochs	total epochs
32	15	37
64	30	67
128	60	127
256	100	227

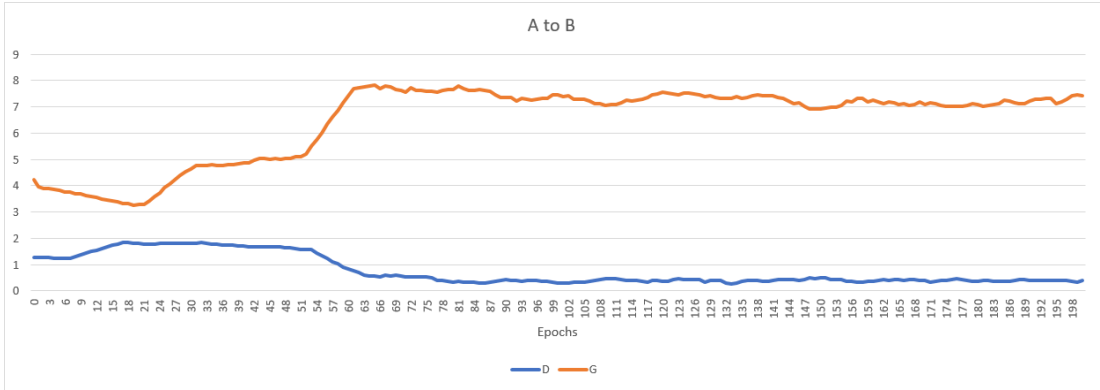


Figure 5.1: The generator (G) and discriminator (D) losses for $Model_{AB}$

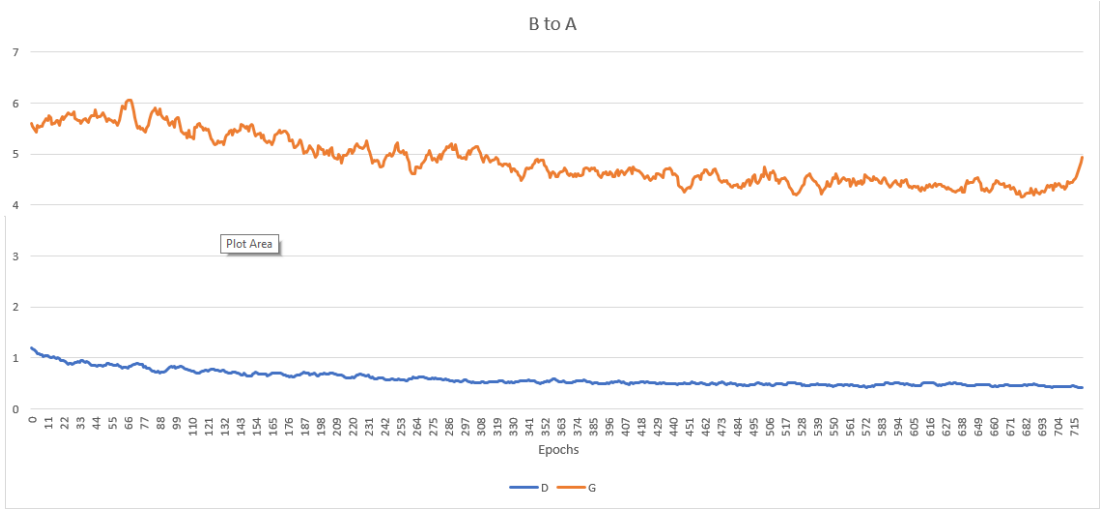


Figure 5.2: The generator (G) and discriminator (D) losses for $Model_{BA}$

followed by a slow decrease. Therefore, the losses didn't converge to low values as compared to the losses for $Model_{BA}$. However, this was beneficial since $Model_{BA}$ is generating maps resembling CAD maps that do not have any random deformations but one fixed target output. Therefore, the lower the loss the higher the accuracy of the output. Whereas, for $Model_{AB}$, there are many possible outputs

for the SLAM map due to the randomness in the errors. Thus, the generator losses should not be as low as that of $Model_{BA}$.

The test results of $Model_{AB}$ are shown in Fig. 5.3. Results of Epoch 155 were chosen because they gave the best results (visually assessed) and because this is where the plateau of the trend graph starts as shown in Fig. 5.1. The assessment of the output can only be made visually, since the output and the target will never be completely the same, but will follow the same error structure. It can be seen that the output errors are very similar to those of the target.

The similar features include the addition, removal, and deformation of occupied and unoccupied pixels. Specifically, the results mimic the common SLAM errors of failing to observe walls/obstacles as seen in the generated SLAM Map 4 Fig. 5.3. Another common error is misplacing observed landmarks and saving them multiple times in the map, which is also modelled in $Model_{AB}$ as seen in generated SLAM Map 3 of Fig. 5.3. This is due to the odometry error which makes the position of the robot with respect to these objects unstable.

Another aspect that contributes to the common SLAM errors is the curving of straight lines mainly due to angular error and curvature in linear motion. Such a case is seen very clearly in the results (ex. Map 5 of Fig. 5.3). Finally, random gaussian errors are also observed in both simulated and generated SLAM maps. This is due to LIDAR noise and the effect of the motion of the robot whether static, which can add additional wall thickness, or linear/angular, which affects the feature matching accuracy. In addition, the results are generated instantaneously as opposed to the simulated maps that are very time consuming and use high processing power.

However, there are some features that can still be learnt further. For one, the shape of the unbounded additional unoccupied (white) pixels occasionally forms a cloudy shape instead of the simulated scattered geometric shapes. For instance this can be seen in Map 6 of Fig. 5.3. In addition, the common SLAM errors mentioned above can be more exaggerated than in the simulated maps. These few limitations can be improved by increasing the dataset or modifying the architecture to include manually selected features as an addition to the input images. Later, a more detailed analysis will be done as a comparison with the parametric modelling results.

The results for $Model_{BA}$ are shown in Fig. 5.4, 5.5, 5.6, 5.7, and 5.8. These results are tested on the model trained till epoch 534, which was chosen by visually assessing the maps and checking the range where the loss plots start plateauing as shown in Fig. 5.1.

The area and structure of the map was first evaluated by utilizing the method proposed in my previous work [3] available in Appendix A. This method relied on finding the mean square error between corresponding edge points of the two maps being compared. For the purpose of illustration we will be denoting these maps with source and destination maps.

To get the correct error, the source map needs to be aligned through rotation,

	As-built floor plan	Generated SLAM map	Simulated SLAM map
1			
2			
3			
4			
5			
6			
7			
8			
9			

Figure 5.3: Test results of $Model_{AB}$

translation, and scale with the target at first. In [3], this similarity transform was generated by combining an automatic graph based method and a manual anchoring of the map. However, for the case at hand, we will use manual anchoring since the automatic similarity alignment method can sometimes give erroneous results.

	Simulated SLAM map	Generated as-built floor plan	Target as-built floor map
1			
2			
3			
4			

Figure 5.4: Test results of $Model_{BA}$

	Simulated SLAM map	Generated as-built floor plan	Target as-built floor map
5			
6			
7			
8			

Figure 5.5: Test results of $Model_{BA}$

	Simulated SLAM map	Generated as-built floor plan	Target as-built floor map
9			
10			
11			
12			

Figure 5.6: Test results of $Model_{BA}$













	Simulated SLAM map	Generated as-built floor plan	Target as-built floor map
13			
14			
15			
16			

Figure 5.7: Test results of $Model_{BA}$





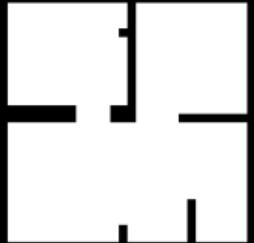
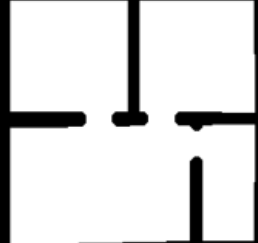




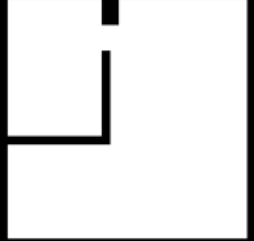
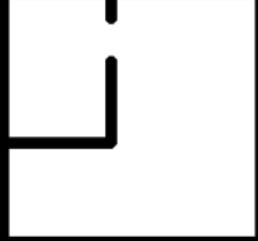



	Simulated SLAM map	Generated as-built floor plan	Target as-built floor map
17			
18			
19			
20			
21			

Figure 5.8: Test results of $Model_{BA}$

The error will be calculated between the edge points of the preliminary aligned source map and those of the target map. To do so, data association of the edge points is done. For this step, the maps were first aligned through a graph based nonlinear optimization method that generates a piecewise affine transformation. After alignment, correspondences of edge points are extracted through the closest point method. Now that the correspondences are extracted it is possible to get the error between the sets of edge points of the target and preliminary source map. The flowchart of calculating the error is shown in Fig 5.9.

In this thesis, this method was used to evaluate the simulated SLAM map against the target CAD map shown in Fig. 5.11. In addition, the generated CAD map from the machine learning method is evaluated with the target CAD map shown in Fig. 5.10. The two evaluation results were then compared by calculating the difference between the errors which are tabulated in Table 5.3. Map 6 was not evaluated with this method since the nonlinear alignment failed with it. We will consider Map 6 as an outlier.

On the one hand, the negative values (highlighted in green) indicate that the structure and area of the generated SLAM map is closer to the CAD map than the simulated SLAM map is. On the other hand, the positive values (highlighted in red) indicate the opposite. From these values we can infer that 75% of the simulated SLAM maps were improved by *Model_{BA}*. The negative values of the remaining 25% could be due to the contraction and expansion of rooms that is usually caused by SLAM errors. It can be inferred that the model did not learn this shrinkage and expansion perfectly. This can be due to the small size of the data or the use of GANs instead of a different deep learning architecture, which would be addressed in future work.

Further analysis was made by plotting a box plot shown in Fig. 5.12. In a boxplot, the first to third quartile (interquartile range) are represented by the blue box and the median is drawn as a horizontal line passing through the box. The interquartile range along with the median falls almost completely in the positive region with positive values ranging from 0 to 7 while the negative quartile only reaches -4. This shows that the *Model_{BA}* is far more likely to improve the simulated SLAM maps.

If this model were to be used in real time with SLAM, it would improve the localization. The errors in area can be corrected with loop closure, and the improvement in the noise and the deformation shown in this thesis can enhance the orientation and location of the robot, since in SLAM the accuracy of the localization depends on that of the map.

However, not all aspects of the generated CAD maps can be assessed quantitatively. That is why a qualitative evaluation is also in order. It can be seen that the errors of nonlinear deformations in the SLAM map decrease; most of the walls have become straight lines and most of the noise including additional occupied or unoccupied parts have been removed (ex: Map 19 in Fig. 5.8). In addition, the results show that the model learnt to remove additional walls as

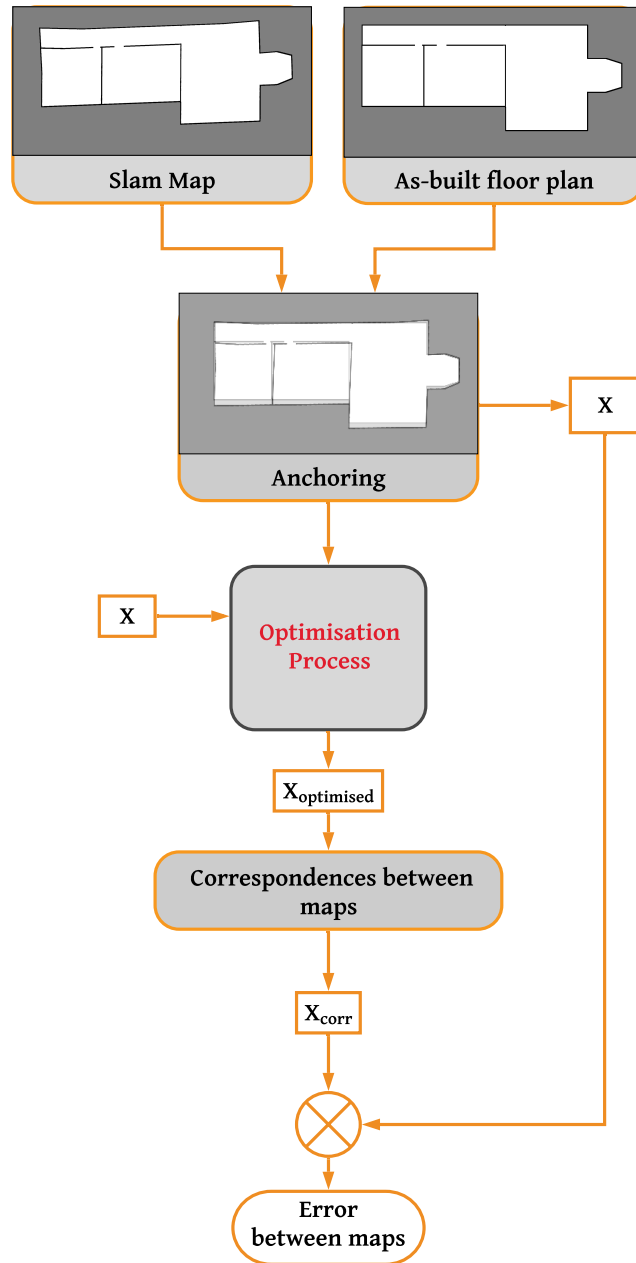
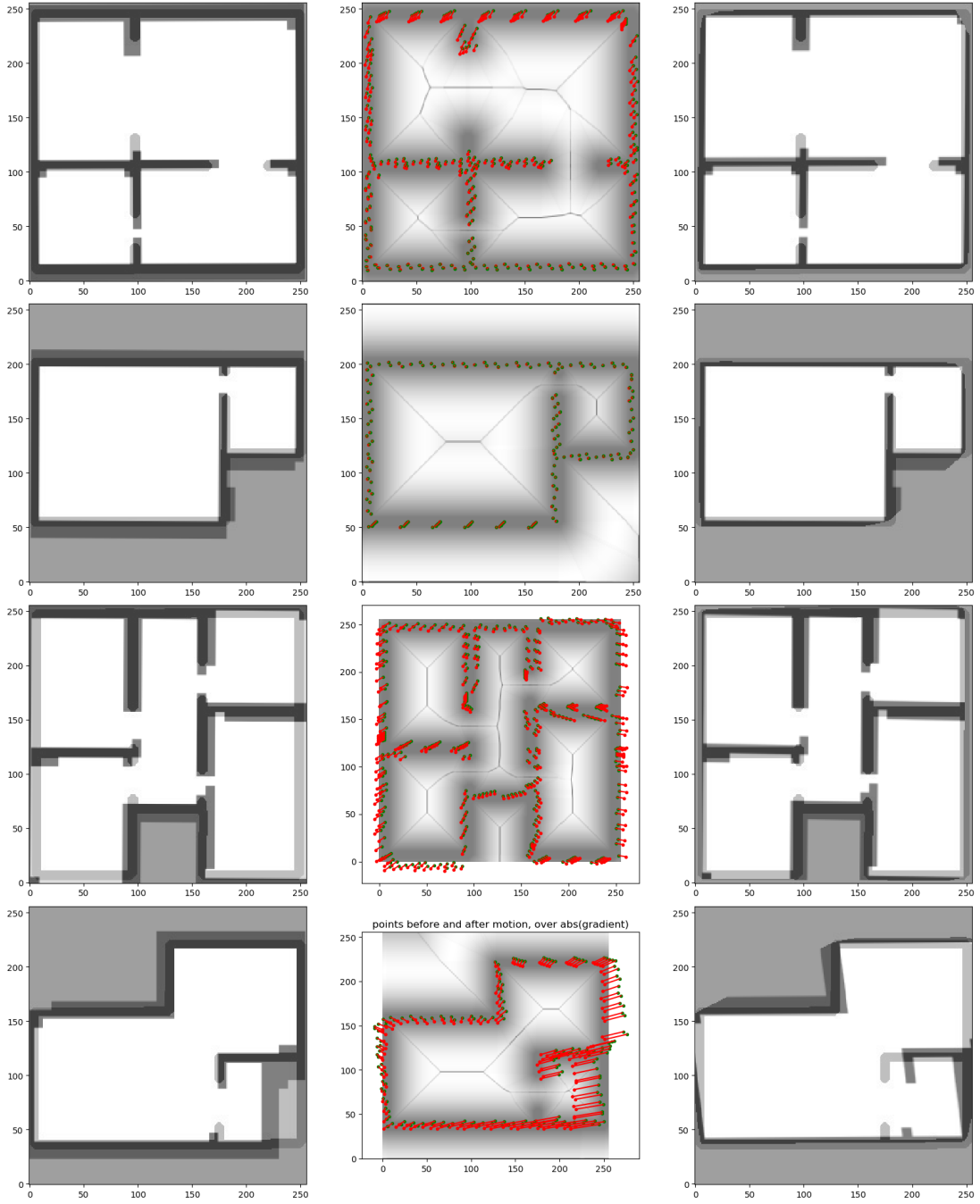
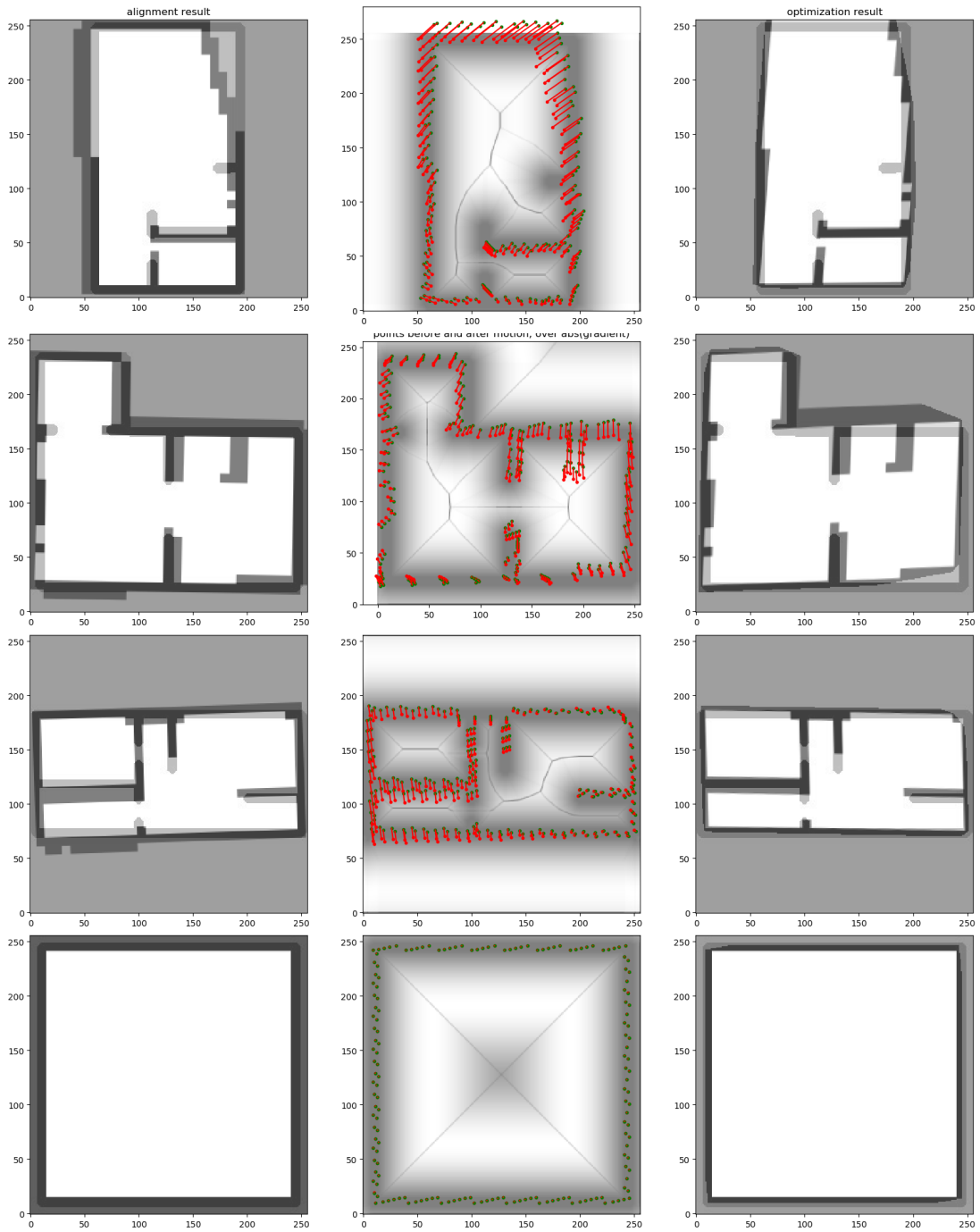
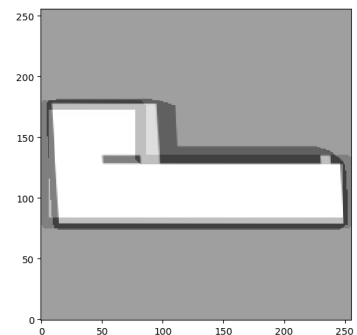
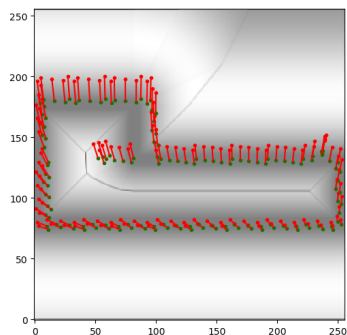
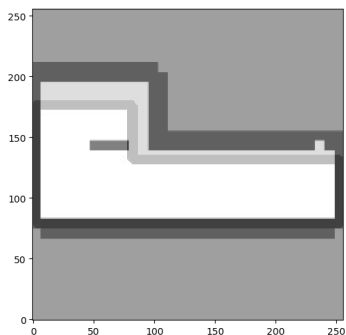
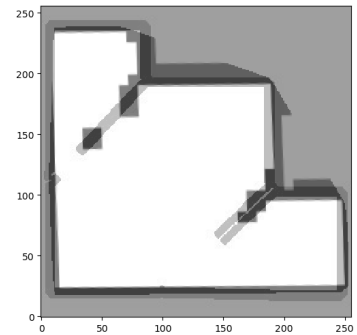
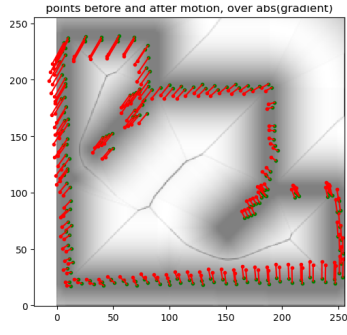
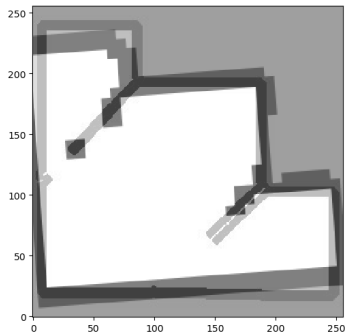
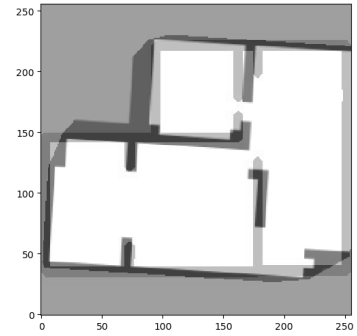
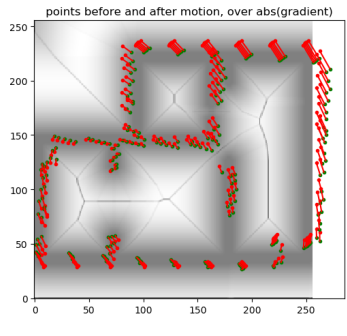
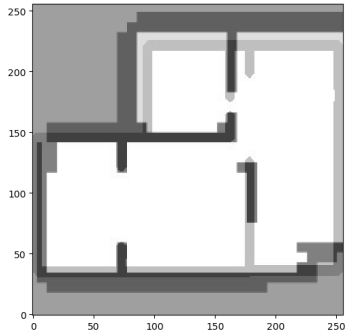
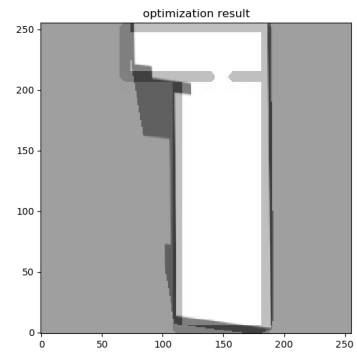
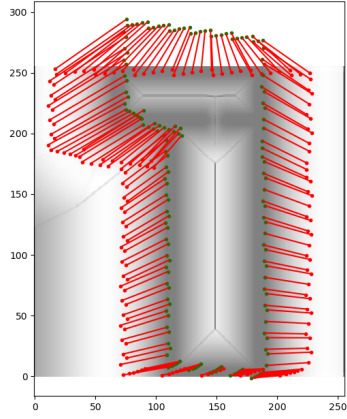
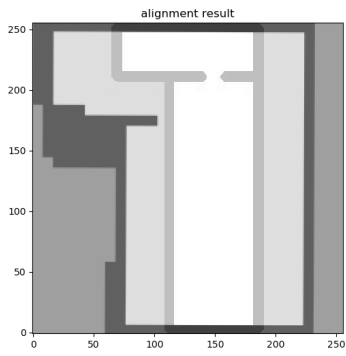


Figure 5.9: The flowchart of the error calculation process adopted from [3]









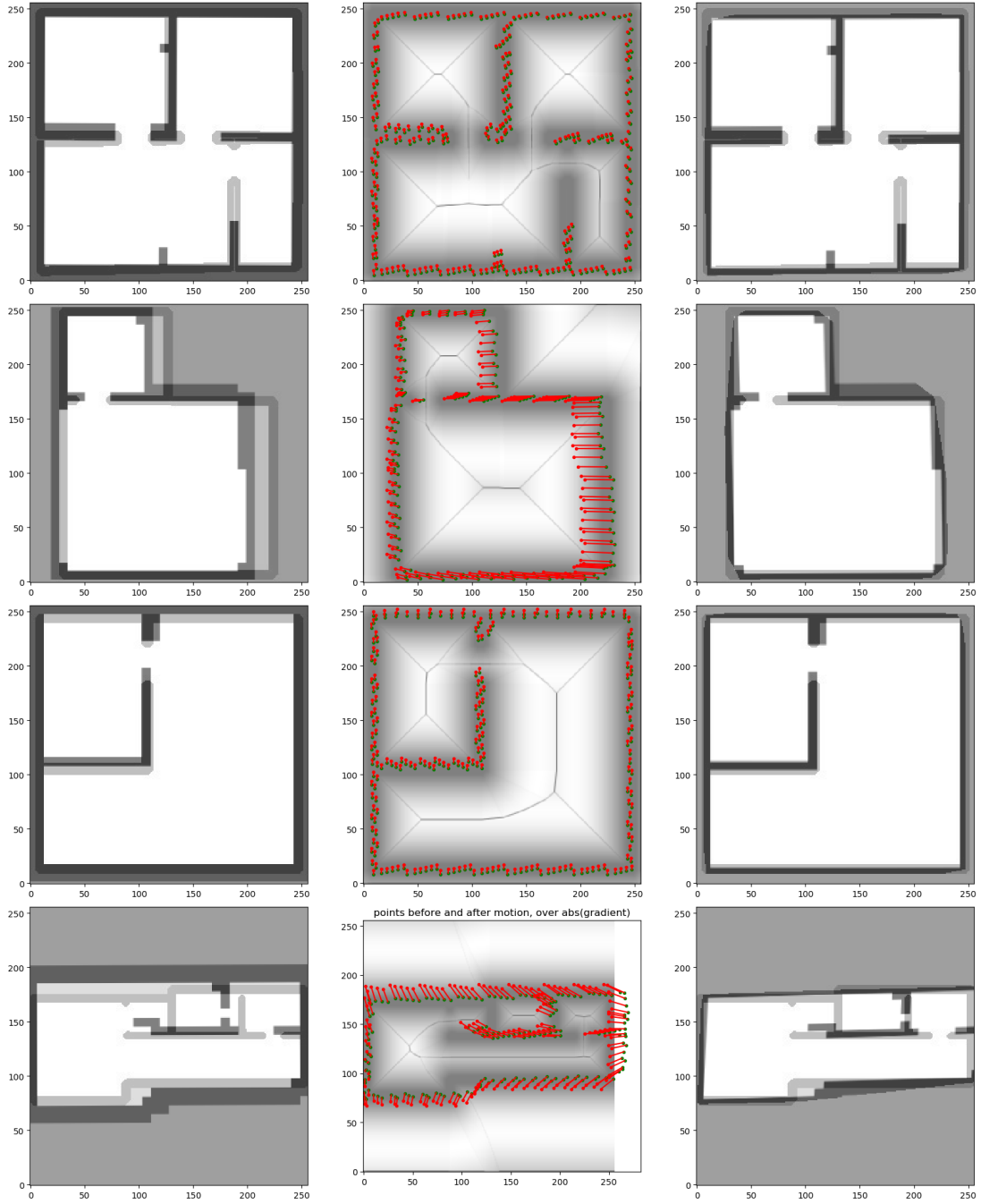
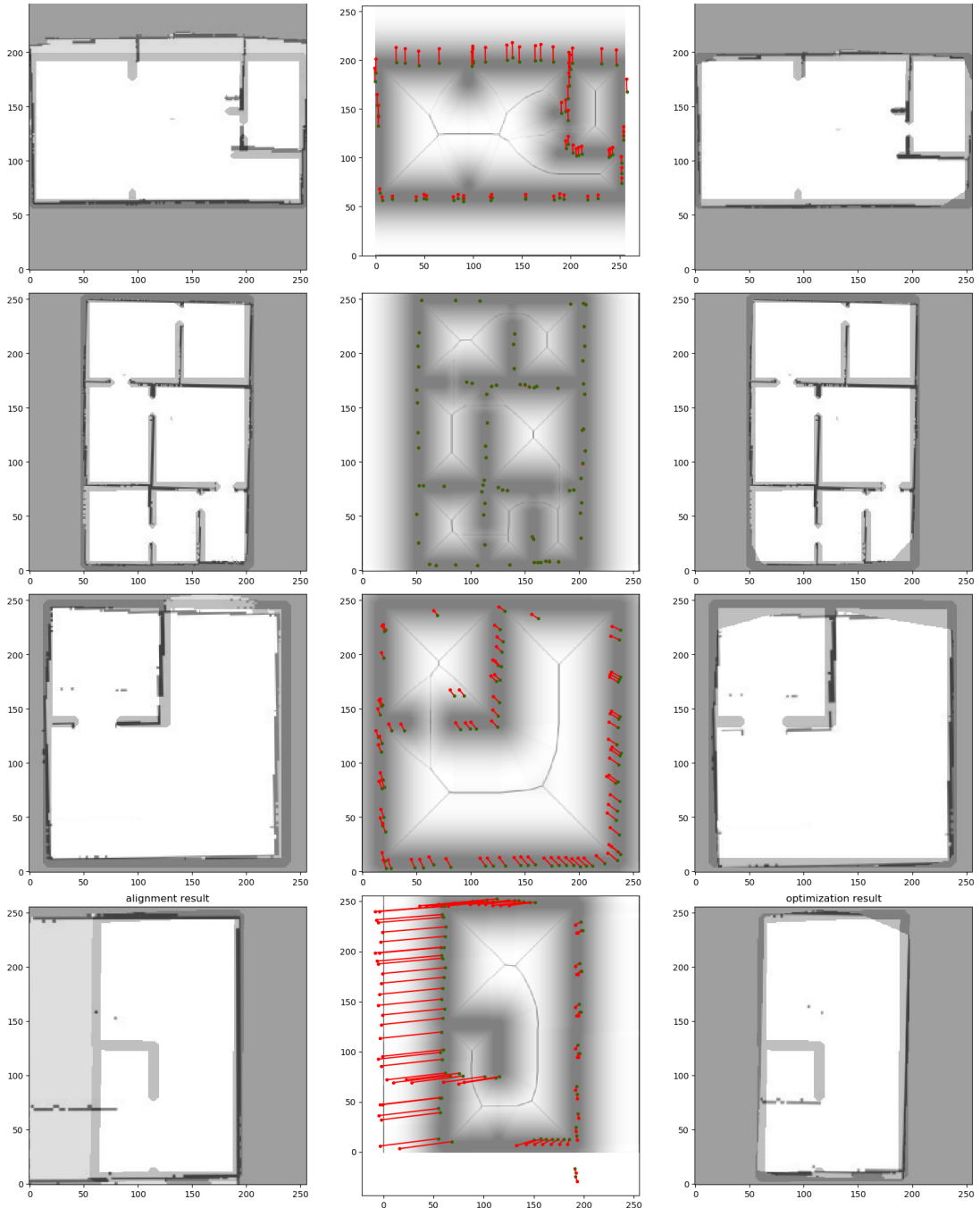
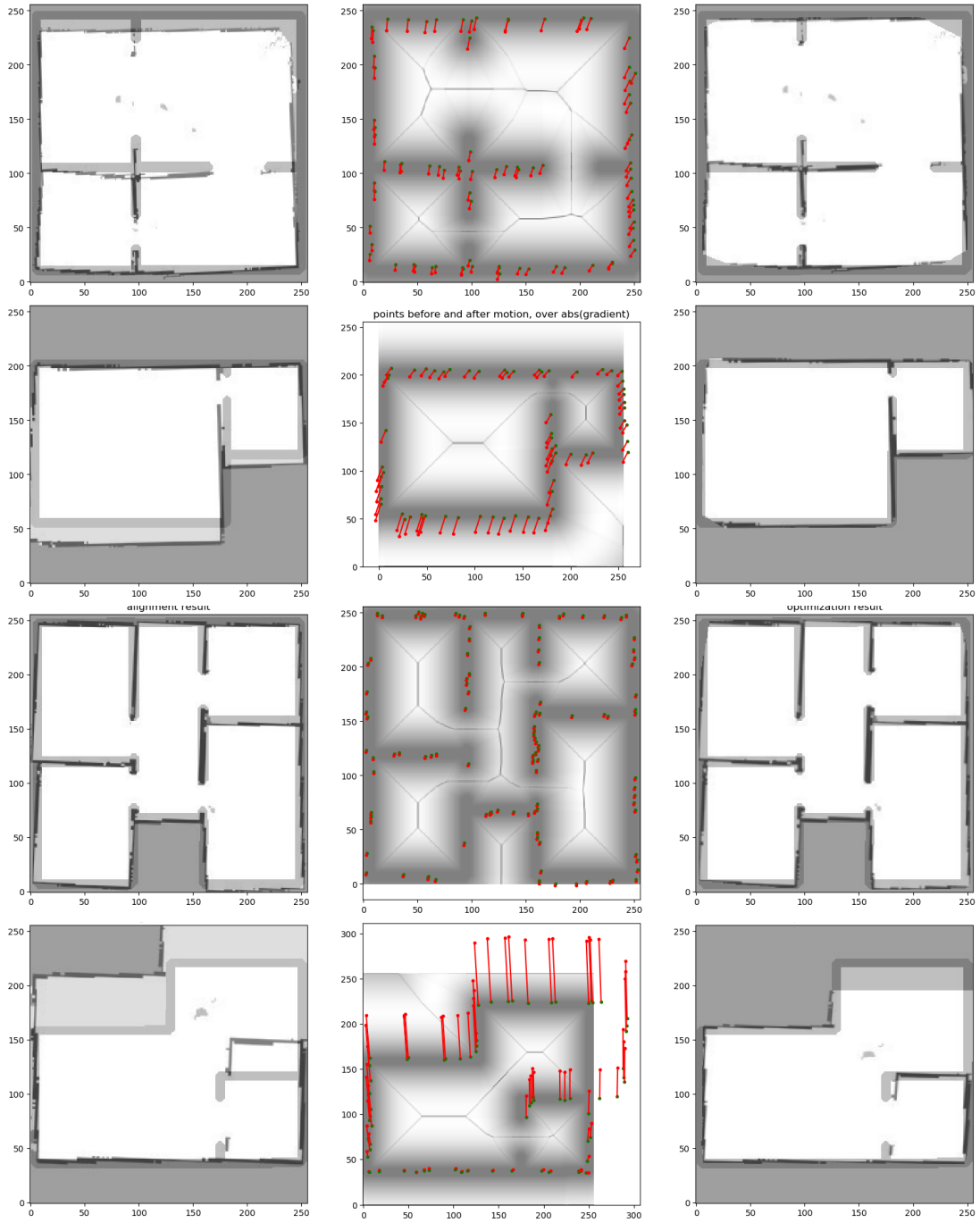
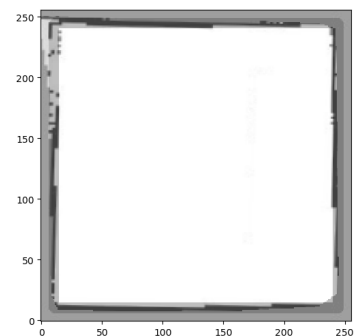
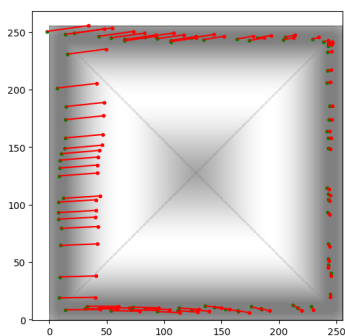
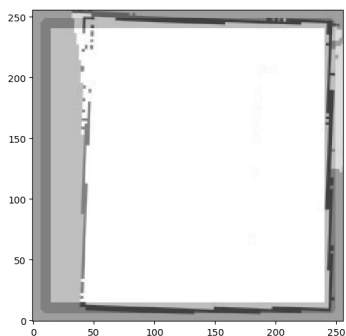
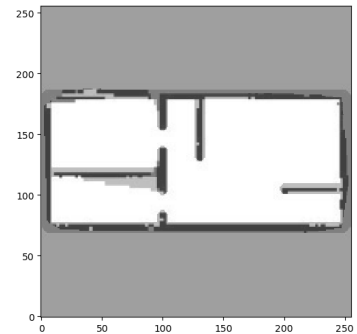
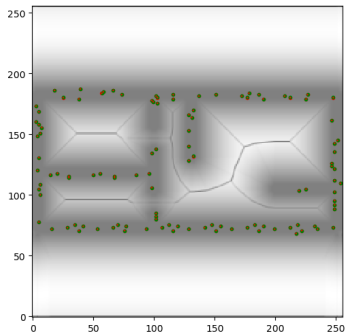
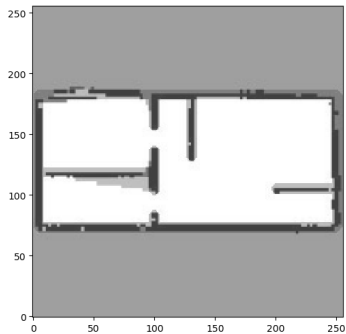
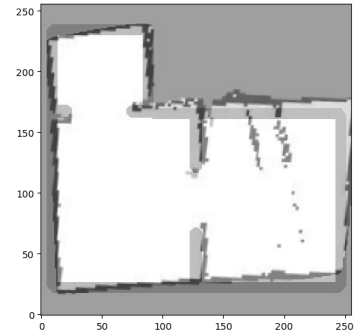
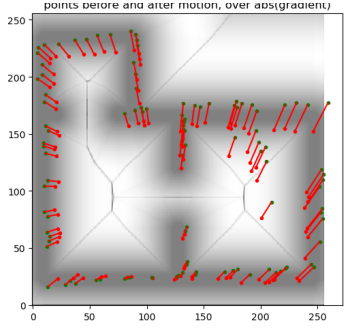
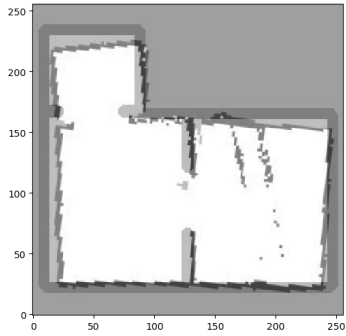
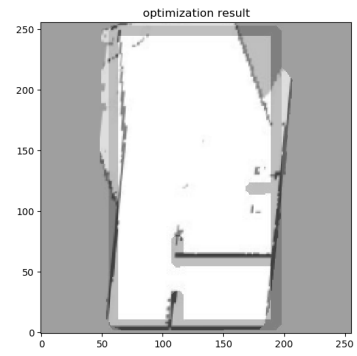
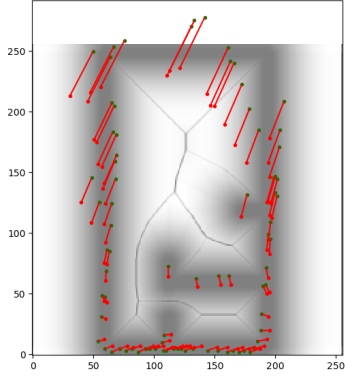
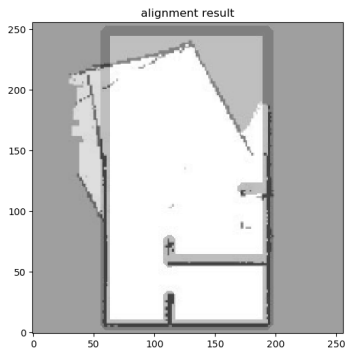
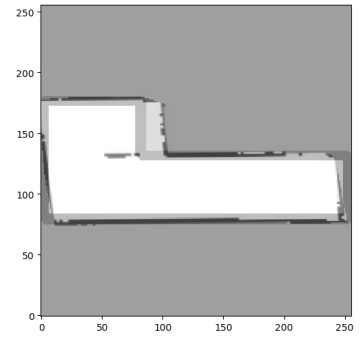
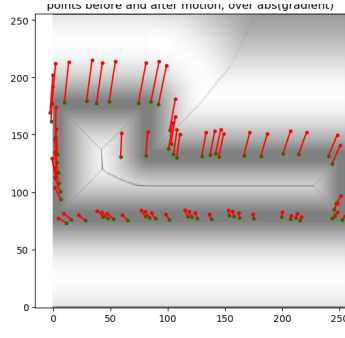
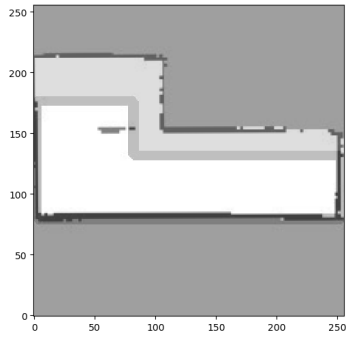
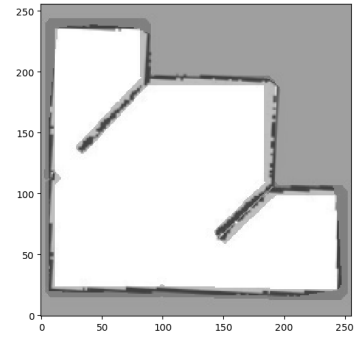
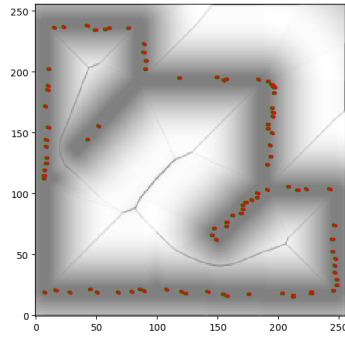
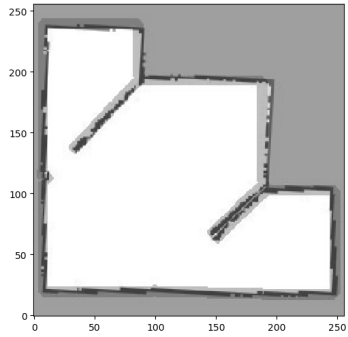
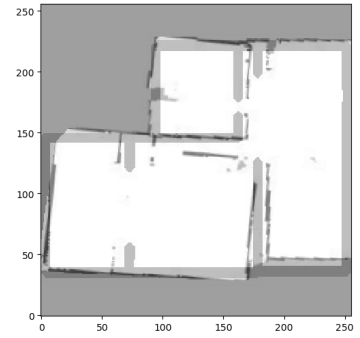
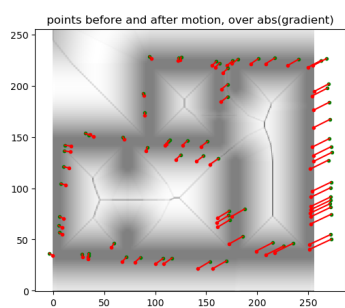
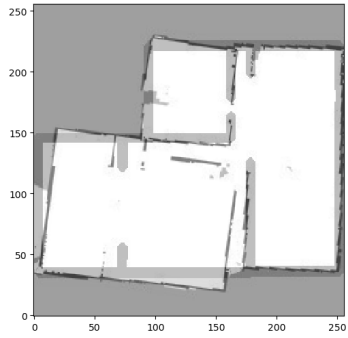
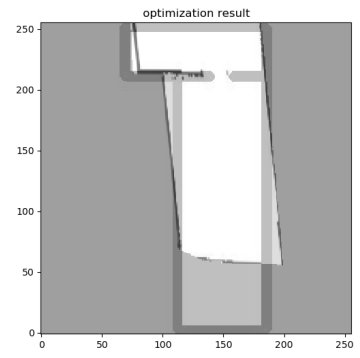
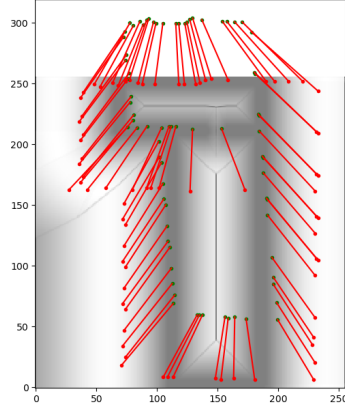
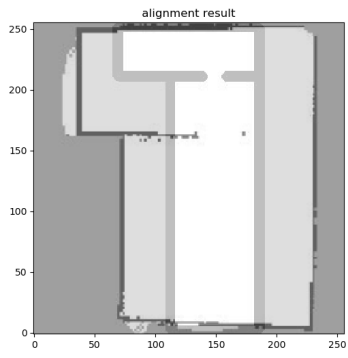


Figure 5.10: The evaluation of the simulated SLAM map against the target CAD map









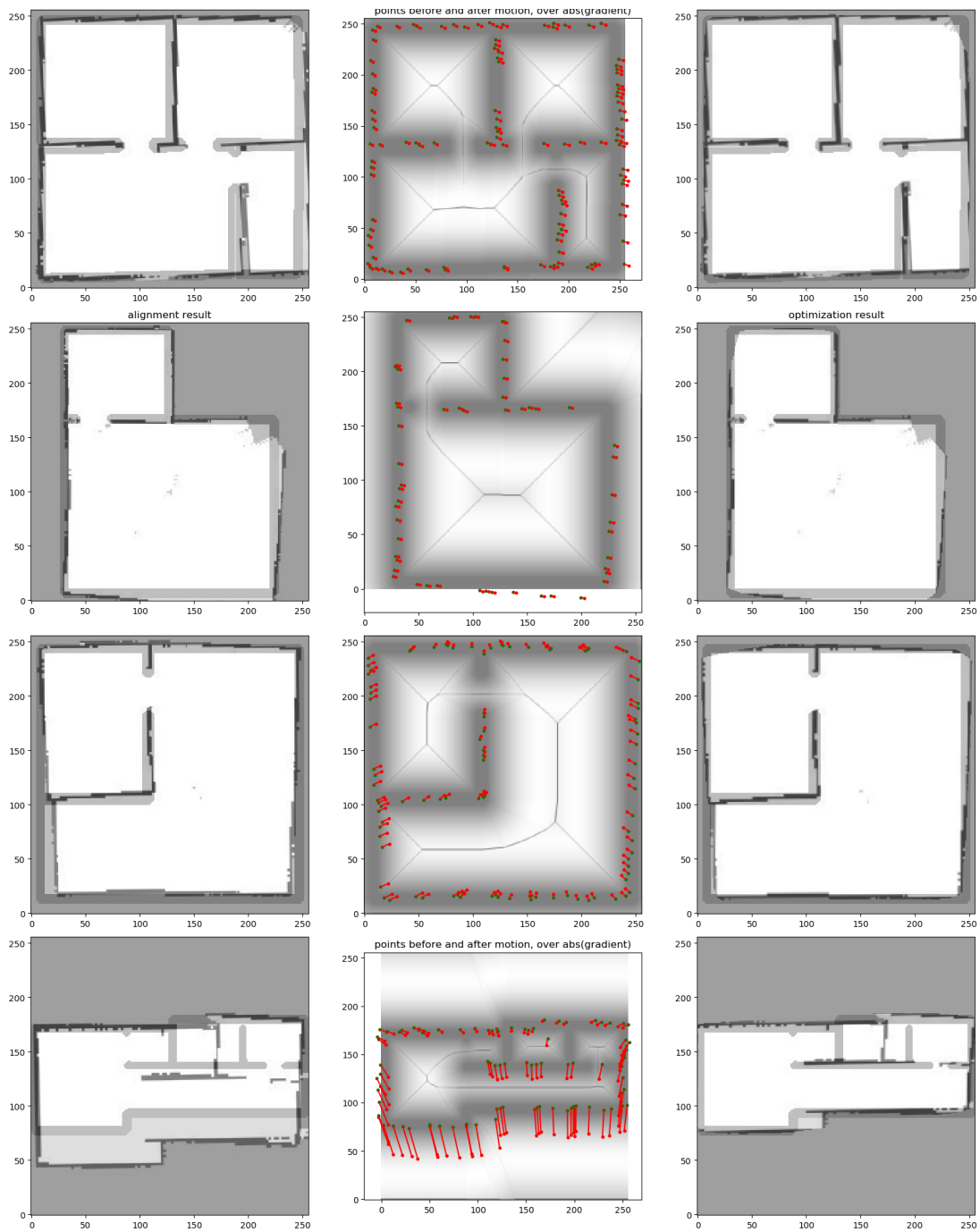


Figure 5.11: The evaluation of the generated CAD map from the machine learning method as compared to the target CAD map

Table 5.3: Evaluation results of the structure and area of the maps generated by $Mode_{BA}$ using the evaluation method of [3]. Here, Error(sim) refers to the evaluation result of the simulated SLAM map with the CAD map. Moreover, Error(gen) refers to the error between the machine learning generated SLAM map and its corresponding CAD map.

Map ID	Error(sim) - Error(gen)
1	3.605497191
2	-1.213548915
3	1.091363786
4	0.411571549
5	1.740986643
7	4.656658349
8	-1.012509943
9	7.317520375
10	0.670650887
11	1.861008056
12	-1.476125407
13	6.222065703
14	1.210678083
15	0.507280654
16	-2.556146321
17	2.899847685
18	0.084914512
19	-3.870882335
20	0.579974187
21	6.350456144

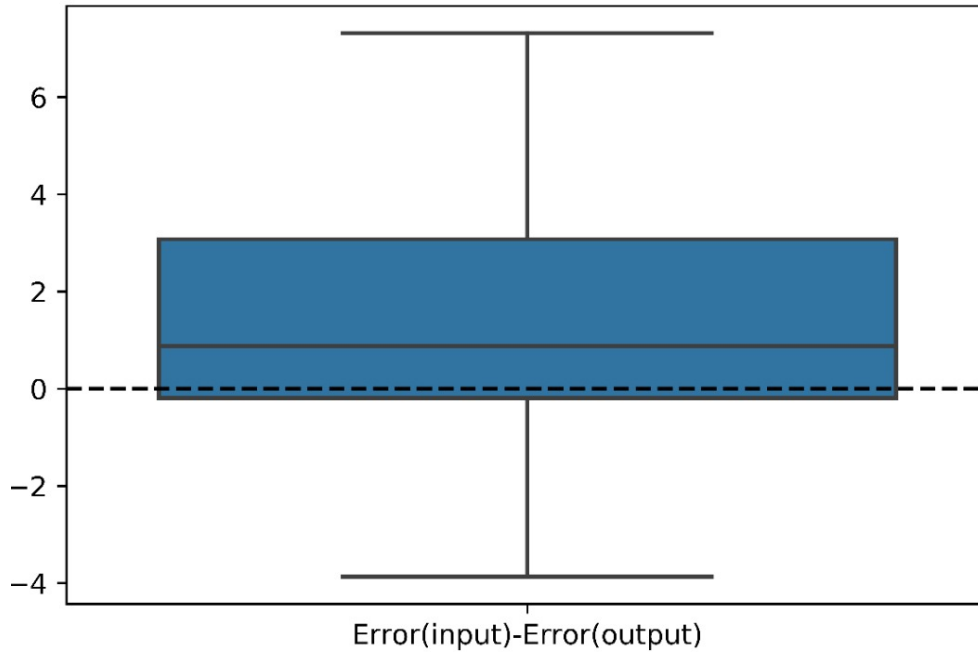


Figure 5.12: The box plot visualization of the errors in Table 5.3

seen in Map 11 of Fig. 5.5. Finally, in general wall positions were improved as seen in Map 10 of Fig. 5.6.

5.2 Parametric Modelling

To assess the parametric model approach, the model is implemented with the two odometry-to-map error conversion methods (EKF and Method of [58]). When running on high resolution images, the EKF method ran for 7 hours and then gave a memory problem. This means that EKF with high resolution can't be used since typical SLAM simulations would be a better alternative in that case. After noticing that the time and processing of the EKF method increased nonlinearly, experiments were conducted on lower resolution images (64x64). The resulting map was then resized to reach a higher (unblurry) resolution (256x256). The resizing was done using bilinear interpolation. With this technique the time was decreased to reach a magnitude of seconds.

As for method of [58] implemented on (256x256) resolution images, time was not an issue, the run time was in seconds even for very high resolutions (ex: 1600x1600). The output of the system with the various conversion methods is shown in Fig. 5.13 with the corresponding time of each run shown in Table 5.4.

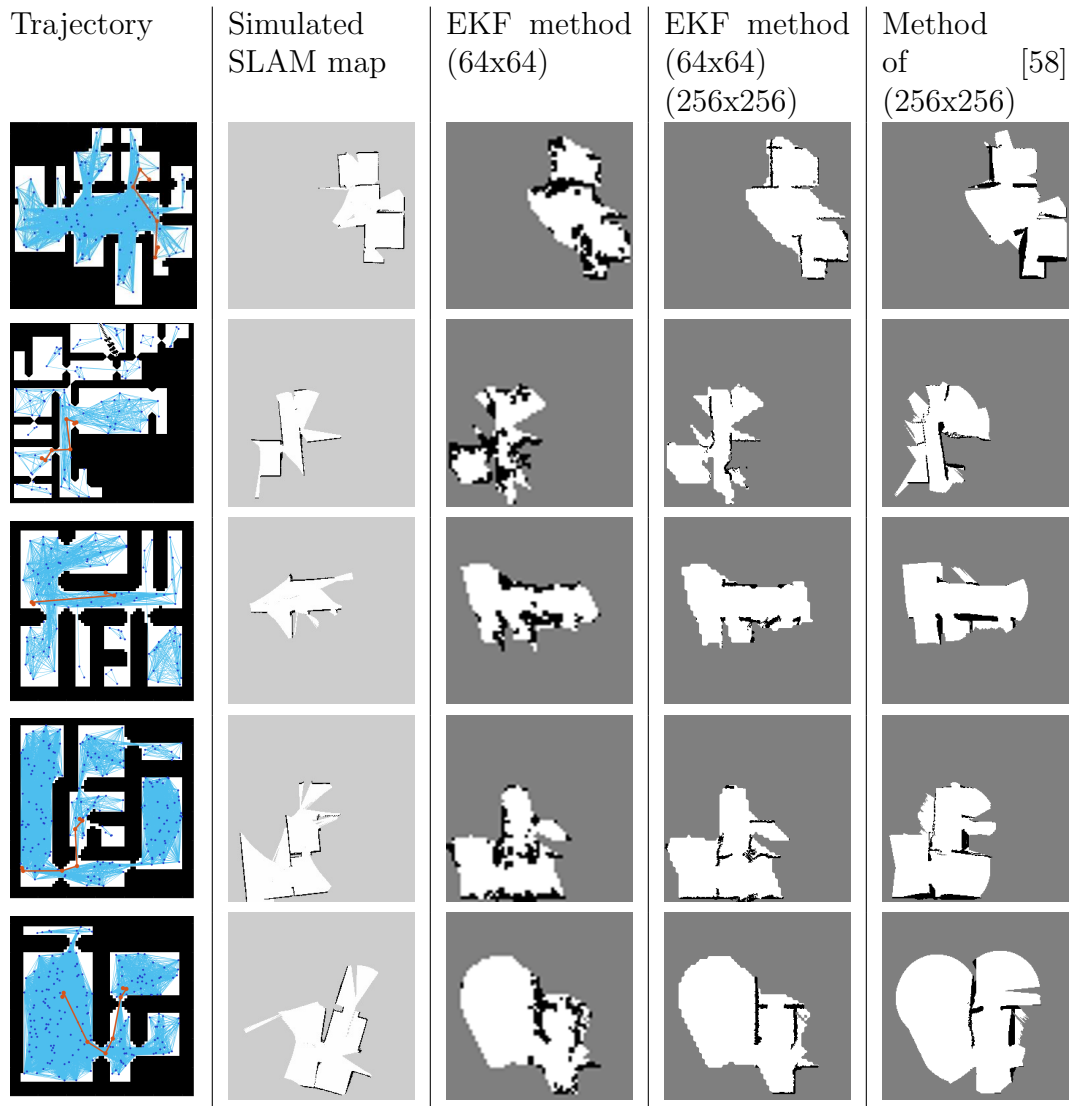


Figure 5.13: Comparing odometry to map conversion methods using using partial mapping

Table 5.4: Time taken in seconds by the parametric modelling with different conversion of odometry to map error methods

Map ID	Low Resolution EKF method	High Resolution EKF method	Method of [58]
1.2	74.8616	44.6421	3.1995
2.2	15.1272	15.5864	2.62
3.2	20.2189	19.3533	2.7361
4.2	34.1854	34.1854	3.6313
5.2	94.4141	94.4141	3.225

The EKF method has a more detailed parametric model, where most importantly the error of reobserved landmarks is not a mere addition as is done in [58]; it relies on a complex covariance matrix that includes all landmarks and robot relations. However, due to this complexity, this method uses almost all equations of SLAM, which makes it time consuming and in need of higher memory.

The method proposed by [58] is (1) faster and (2) can handle high resolutions. Thus, it (3) avoided the bilinear interpolation, which can cause loss of data. Finally, the results in Fig. 5.13 show that the deformations are similar in both methods. Therefore, for the comparison of Parametric modelling with the Machine learning method, the conversion of [58] will be used.

The results of the parametric modelling are shown in Fig. 5.14, which also includes the corresponding simulated SLAM maps with the specific trajectory (Trajectory A) used for both. Even though $Model_{AB}$ gives one possible hypothesis of a SLAM map with no specific trajectory, it is also included in Fig 5.14 to be able to compare both methods. Having the advantage of a user defined trajectory, the parametric modelling approach shows a closer resemblance to the simulated SLAM map of the same trajectory than the random result of $Model_{AB}$.

From the results of Fig. 5.14, we can deduce that while both methods give very realistic results, $Model_{AB}$ incorporates features not taken into consideration in the parametric model. For instance, removal of walls and the shrinkage and expansion of rooms is portrayed better with the learning approach as shown in Instance i of Fig. 5.14. Whereas the parametric approach doesn't remove any structural elements; for example, doors in the parametric approach are better preserved. Wall removal is present in SLAM errors; however doors are mostly preserved in SLAM.

In addition, From the results we can deduce that the parametric approach does not model the simulated SLAM map's sudden brokenness and large room shifts $Model_{AB}$ as shown Instance (a) and (e) in Fig. 5.14. These large errors usually occur mainly due to vast spaces in the environment where there are no special features and the data association fails. This makes sense because feature

association is mainly an environment specific aspect which is the domain in which the machine learning model $Model_{AB}$ was specifically trained.

To conclude, the advantages of modelling using machine learning are:

- The ability to generate CAD maps from SLAM maps, which could help improve generated SLAM maps in any application
- Allows for further research of the effect of environment shape on the errors by studying the important features used in these models
- Incorporate aspects not present in parametric modelling such as removal of walls, room specific scaling, sudden brokenness, and large room shifts

As for the advantages of parametric modeling:

- Quantitative understanding of the deformations through displacement fields shown in Fig. 5.15. These fields can act as a ground truth local transformation field for assessing alignment methods as was done in my previous work [5].
- Multiple possibilities of SLAM maps can be generated from one CAD map as opposed to the one hypothesis of $Model_{AB}$.
- Random or user selected trajectory as opposed to the random trajectory generated by $Model_{AB}$.
- Preservation of doors in the map



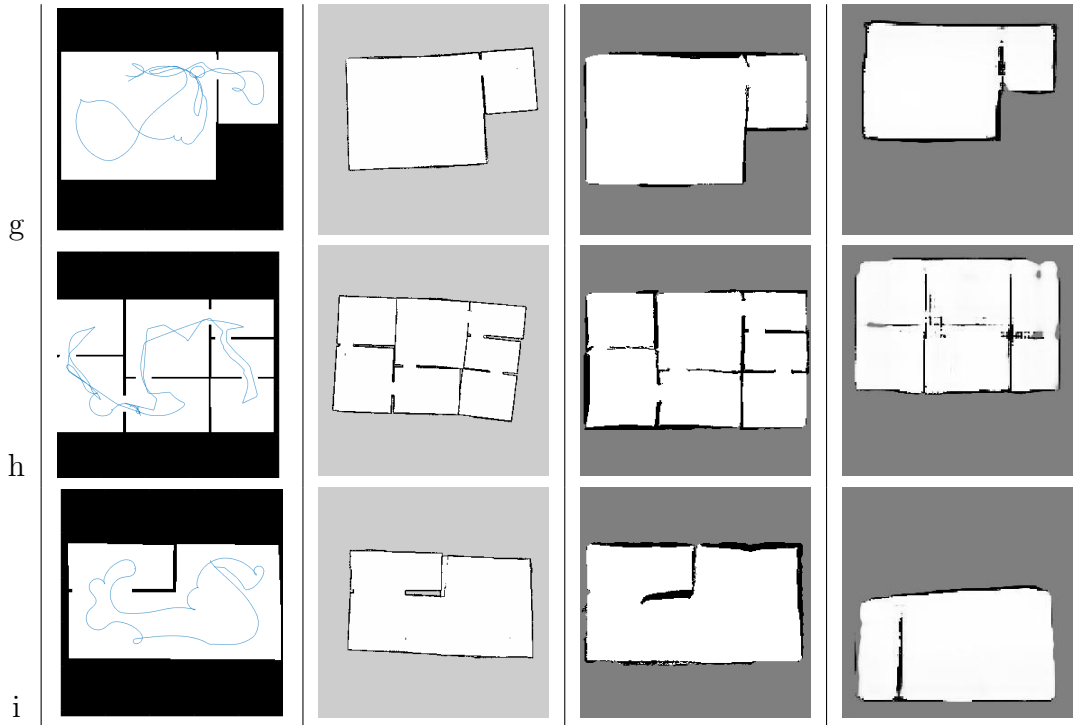


Figure 5.14: Comparing Machine Learning to Parametric modelling approaches using mapping of the complete floor



Figure 5.15: An example of an CAD map with a specific trajectory (left), the displacement field (middle), and its corresponding deformed image (right)

Appendix A

Automated Robotic Assessment of 2D As-built Floor Plans

Daniel Asmar¹ · Rema Daher¹ · Yasmine Hawari¹ ·
Hiam Khoury² · Imad H. Elhajj¹

Received: date / Accepted: date

Abstract Site inspection is a notably tedious, time-consuming, and error-prone process when carried out manually by construction inspectors. It therefore is a prime candidate for automation and would reduce the effort and time incurred while improving and organising the obtained data. Incorporating recent contributions to the field of laser-based measurement systems, this paper presents a system aimed at an infrastructure-less automation of one portion of the construction-site inspection task: the comparison of as-built drawings to their as-designed counterparts. The proposed system relies on Simultaneous Localisation and Mapping (SLAM) to build a dense map of the environment, which is then used to automatically validate the original 2D as-planned drawings. The components of the proposed system have been tested through proof of concept ex-

periments and results highlight the potential of using laser measurement data as proposed for automating and improving the inspection processes in the field of construction engineering.

Keywords Site surveying · Automated inspection · Robotics · Simultaneous localization and mapping

1 Introduction

If a construction project were comprised of an absolutely flawless set of processes, then its input would be the set of as-designed documentation approved beforehand, and its output would be the final structure, which matches the original design intent perfectly. In reality, however, this is not the case. During and after a construction operation, the current structure cannot be truthfully represented by the as-planned documentation. Therefore, a documentation of the current structure must be generated and updated throughout the construction process so that it may provide a more accurate representation, not only while it is being built, but long after it is completed as well. This documentation is referred to as the "as-built map". In the United States, for instance, existing buildings are expected to re-

¹ Vision and Robotics Lab,
Maroun Semaan Faculty of Engineering and Architecture, American University of Beirut, Riad El Solh, 1107-2020, Beirut, Lebanon
E-mail: da20@aub.edu.lb, rgd05@mail.aub.edu, yde01@mail.aub.edu, ie05@aub.edu.lb

² Civil and Environmental Engineering,
Maroun Semaan Faculty of Engineering and Architecture, American University of Beirut, Riad El Solh, 1107-2020, Beirut, Lebanon
E-mail: hk50@aub.edu.lb

main operational for roughly 30 to 50 years, so about 87% of the current building stock will maintain operation in the year 2050 [23]. In other words, a large percentage of buildings in existence will remain so until the middle of this century, and their physical state is represented by none other than as-built documentation. Despite being so essential, as-built data remains poorly collected, communicated, processed, and/or revised [23, 22]. Again, using the United States as an example, the result is a whopping estimate of approximately \$1.5 billion incurred annually, due to missing, incomplete, or otherwise flawed as-built documentation, and a further \$4.8 billion incurred annually, in order to validate any available as-built documentation, according to a 2004 report by the National Institute of Standards and Technology (NIST) [12].

The generation and verification of as-built documentation has warranted particular interest within the multi-disciplinary fields of automation and robotics in construction. Concepts of automation and robotics should not be confounded; however, the two fields benefit to different extents from the present prevalence of laser measurement systems. On the one hand, these sensors, coupled with advances in the field of robotics, contribute to the production of robots that are increasingly effective at mapping and scanning unstructured and dynamic environments [26], which are the type associated with construction projects. However, the restrictions placed on robots at the construction site arise from their current inefficiency at autonomously traversing the cluttered, dynamic, and uncontrolled terrain inherent to that category of environment [29]. On the other hand, advances in the field of automation in construction may truly benefit from laser measurement systems' accuracy and resolution. After all, small and light incarnations of these sensors can be carried by a remote controlled robot instead of the myriad personnel who perform as-of-yet manual data collection tasks in a construction site environment [23]. Furthermore, the hardware trends

are supplemented by similar trends in software; the accuracy of laser navigation algorithms continues to be refined as the size of the hardware continues to be miniaturised. It is at this point where the development of more effective, less expensive laser measurement sensors meets advances in localisation and mapping, with application to construction environments that the content of this paper finds its calling.

In brief, the system herein proposed aims to automate a specific construction site inspection task, which is the localisation, measurement, and assessment of as-built floor plans with the use of data obtained by LIDARs mounted on a mobile robot. The proposed solution is a cost-effective, rapid, accurate, and infrastructure-less one that (1) does not require a stationary on site device to perform the as-built floor plan inspection task, (2) uses a 2D laser scanner in lieu of bulky 3D ones, (3) introduces an accurate anchoring technique to enhance map alignment and (4) proposes a new error metric for quantitatively evaluating the difference between two corresponding maps.

2 Background and Related Work

The duty of a construction inspector is to guard the standards of quality on-site, monitor construction work for conformity with specifications and drawings, and maintain the agreed-upon workmanship standards. The inspector must perform all these tasks impartially and independently, without being influenced by the overarching costs and deadlines. In doing so, construction inspectors can consequently reduce the defects, improve the quality, and ensure the acceptability of the finished product, and thus avoid the potential direct and indirect costs that could otherwise be incurred [40]. It is by this rationale that construction inspectors are found to be cost-effective, a quality that can be significantly augmented by the introduction of automation to the construction inspection task.

Incorporating some degree of automation to construction tasks not only increases cost-effectiveness, but also generally increases productivity and enhances safety as well. In this context, the automation of construction work is defined as the increase of the workload on machines, systems, and robots while decreasing that on human beings and manual labour. However, not all tasks are well-suited for automation. Given the current state of the art, robots [31,16] and automated systems [4,23] are relatively more reliable and cost-effective at tackling the levels of elemental motions and basic tasks. These two levels of the construction industry are still quite broad and, although a breakdown of the basic tasks comprising all construction work is detailed by [8], only one in particular is relevant to the discussion at hand: the inspection task, which can be defined as the process of critical examination with the intent to detect flaws and verify the correctness of as-built structures [4]. However, this simple and concise definition belies a multifaceted set of activities that sprawls both spatially across the construction site and temporally across the construction process. Systems such as [17] deal with the temporal aspect of the inspection task, whereas the system proposed herein focuses on the spacial inspection and specifically on that of as-built floor plans.

In the current state of affairs, when verification of as-built floor plans is required, the inspector must first collect and become acquainted with any as-built documentation available. Numerous case studies attest to the fact that acquiring as-built documentation is no prompt or efficient activity, mainly due to insufficient on-site quality inspectors, pressure to cut costs or meet target dates, and ineffectual or disorganised communications among the project's departments [22]. Once all the data has been acquired, it is dubbed the current as-built state (a.k.a. the ground truth, in this instance), and it is used to validate and update any existing as-built documentation [23]. If the as-built state differs from as-planned documentation, or from previous as-built documentation if avail-

able, by a margin exceeding the project-specific tolerance, then corrections are in order. If the construction of the inspected area is still in its early stages, then efforts may be made and resources may be channelled towards reducing the error to within tolerable limits. If these errors are found too late in the construction process, then a whole range of potential revisions to the as-built documentation could be made. Some of these revisions involve multiple departments that are separated both horizontally and vertically in the construction management hierarchy. As a result, the expected revisions may not be communicated accurately or effectively, may not be assigned enough consideration or resources, and may not be implemented correctly or efficiently. This is due to the multiple data dissemination issues that plague communication in the field of construction [22].

As the cost of computational power decreases, the feasibility of automating as-built map assessment tasks increases. Far from being identical to manual assessment, automated assessment using sensors and robots is more effective in multiple ways. The manual techniques employed by current construction inspectors are subjective and prone to qualitative human errors, due in no small part to the human tendency to become fatigued, inattentive, and less focused as time wears on [31]. Automated systems are objective and, although they are prone to their own types of errors, those errors may be remedied by future developments in the field. The bottom line is that the time, the effort, and by association, the costs of assessment are inherently reduced by introducing automation.

In recent years, various methods have been used for geometric data acquisition in the construction field. A comparison of these methods is shown in Table 1. However, they all rely on manual procedures. Consequently, even if some methods are fast relative to other manual methods, they are all rendered time-consuming as compared to automated systems. Another disadvantage of some of the most popular manual methods is operation complexity.

Table 1: Geometric data acquisition methods used for as-built map generation [33]

Map Data Acquisition Methods			
Method	Data treatment	Equipment Cost	Data Retrieval Speed
Traditional Methods	Simple/ time consuming	Low	Slow
Photogrammetry	Simple/ time consuming	Low	Non-real time retrieval
Videogrammetry	Complex	Low	Real time retrieval
3d camera ranging	Complex	Medium	Real time retrieval
3d laser scanning	Complex	High	Non-real time retrieval
Topographic methods	Complex/ time consuming	High	Non-real time retrieval

In practice, the most popular methods to manually reconstruct environments are the cameras 3D laser scanners [33]. Indeed, the relevant literature is teeming with examples of Computer Vision technologies or laser scanning with as-built floor plan documentation, observation, and assessment applications [32]. Some applications involving extraordinarily large structures or complex environments have even successfully employed both technologies simultaneously [23].

Computer vision technologies, such as photogrammetry and videogrammetry, deal with reconstructing 3D space by use of images and video frames respectively, [33] and proved promising in the application of comparison of as-built drawings to their as-designed counterparts. These techniques [45,24,2] rely on feature matching between key frames or sequences of images, and are therefore prone to failure when the reconstructed space is void of features; this is the case with construction sites that mostly consist of empty walls and repetitive columns. Additionally, the data obtained from these techniques is comprised of dense or semi-dense 3D point clouds that require lengthy processing times and a considerable amount of cleaning [28]. It is also important to note the difference in depth information that either sensor can provide. While vision sensors provide rich 3D data containing information about textures

and colours, it is more difficult to retrieve accurate distance measures given its risk of mismatching large amounts of information [13,20].

As such, for the reasons mentioned above, several systems have been introduced for the purpose of mapping and assessing as-built floor plans using laser scanners, which can directly provide more accurate distance measures. However, most of these systems use 3D laser scanners which are often bulky, relatively expensive, and require a large amount of hard disk space to store the 3D data points that they collect [32,37,48]. In addition, these 3D laser scanners are positioned inside the construction site to form 3D maps.

For instance, Boshce *et al.* [3] proposed an approach for comparing 3D CAD objects with the data obtained by a 3D laser scanner positioned within a construction site. The comparison is performed using object recognition, in which parts of the construction site, such as columns, are recognised and matched to the as-designed 3D CAD drawings. However, this approach reveals only part of the construction site since the laser scanner holds a stationary position within a dynamic environment. Xiong *et al.* [46] presented a system for automating the reconstruction of virtual 3D maps from point clouds, extracted via 3D lasers. In [1] an automatic 3d reconstruction is also presented, but with the use of a combination of sensors

including 2D Laser, 3D laser, and RGB camera. Although these systems successfully provide rich 3D maps, the focus of their work does not include any validation of as-planned drawings. Finally, their system is of a more holistic nature, requiring large processing times, and not applicable to a task of quick inspection.

On the other hand, when it comes to estimating the error between as-built and as-planned drawings, the majority of existing systems such as Macher *et al.* [27] have relied on manual alignment and validation of as-built plans with ground truth. Others, such as Jung *et al.* [21] have matched critical points manually to calculate the root mean square error. In [24], RGBD data is used for reconstruction then as-built and as-planned drawings are aligned by detecting the floor and walls, after which objects are categorized as built or not using raycasting. In the latter method, only a binary categorization is made to compare as-built drawings to their as-planned counterparts. In addition, this system relies on object detection from RGBD reconstruction which relies on features matching. As discussed previously, due to the lack of features on construction sites, this method would not be able to be used for an accurate quantitative assessment. In this case, Intersection Over Union (IOU) [25] metric is a possible option. However, IOU acts as a global metric, and fails to accurately estimate local disparity, which is needed to differentiate between maps that have similar global errors. Alternatively, correspondences can be estimated using Hough/Radon transform, but it turns out this method is limited to rigid transformations. Another method is a graph-based approach, where maps are represented as vertices and edges, and their solutions produce affine transformations by capturing similarities related to topology and geometry, which are prominent features in building layouts. However, as in the case of IOU, rigid and affine transformations, that target global parameters, cannot achieve perfect alignment nor flawless data association due to their disregard of local defects. To account for these

defects, data association can be performed using optimisation, such as point set registration and Iterative Closest Point (ICP) [47]. However, the success of such methods is limited to small image displacements, which is not the case of the proposed study. Finally, image registration is usually used for pixel correspondences by relying on pixel intensity information; however, occupancy grid maps have a limited range of intensity values, and have many repetitive patterns. Hence, image registration solutions [34] for occupancy maps are susceptible to local minima and have not yielded good results.

The main contributions of the presented work thereby lies in the development and evaluation of an infrastructureless robotic system which (1) does not require the time-consuming procedures, complex operation, and large amount of data storage of a 3D laser or any other form of infrastructure, (2) opts for a 2D laser scanner or LiDAR and compensates for its readings' errors by using an occupancy grid-based map and incorporating a stochastic model into the algorithm's framework, and (3) has the 2D laser scanner mounted on a mobile robotic platform that can be also deployed as a form of wearable computing and used by an inspecting agent, thereby allowing the task at hand to be performed in a cost-effective, rapid, and accurate manner. Another major contribution of this paper lies in adopting a combined approach to resolve the correspondence problem, where the maps are first aligned before the data is corresponded. In this case, the proposed robotic system incorporates an anchoring technique which renders alignment more accurate. As anchoring acts as an initial guess, accuracy increases which, in turn, increases the possibility of finding a global optimal solution instead of a local one in the optimization process and reduces the number of iterations needed. The proposed solution is inspired by [35], which uses a graph-based method to produce an affine alignment coupled with nonlinear optimisation of a fitness function that obtains a piece-wise affine alignment (*i.e.*, local alignment). Once

the maps are aligned, edge points in the SLAM map are corresponded with those in the as-planned map, and then the original edge points before alignment are used to calculate the corresponding error.

The overarching goal of the research described in this paper is thereby to study the effectiveness of the proposed robotic inspection system in mapping and assessing as-built information in construction applications. The following section in this paper presents the underlying developed algorithms of the proposed system.

3 Proposed System

We propose an automated robotic assessment of 2D as-built floor plans, which consists of both map generation and map matching and validation as shown in Figure 1. The flowchart depicts a synthetic SLAM map (a) and its corresponding ground truth as-built map (b). From (a), X_{GFT} are extracted, which represents "Good Features to Track" [36]. X_{GFT} are then fed into the preliminary map alignment process (c) with anchoring, which outputs a nonrigid alignment (d) and a rigid alignment (e). The optimisation process (f) is performed on (d) to transform the edge points X into $X_{optimised}$. Similarly, (f) is performed on (e), to transform edge points, X_{rigid} , after rigid alignment into $X_{optimised,rigid}$. Correspondences are then made (g) to acquire the closest occupied points in the ground truth map, X_{corr} and $X_{corr,rigid}$, to $X_{optimised}$ and $X_{optimised,rigid}$, respectively. If the SLAM map is globally sheared or scaled (h), the error is calculated between $X_{corr,rigid}$ and X_{rigid} (j); otherwise the error is calculated between X_{corr} and X_{rigid} (i). All the flowchart sections except those in red are part of our contribution.

3.1 Map generation using SLAM

SLAM is the problem of localising an agent in a setting, while concurrently building a map

of that environment [7]. LIDAR-based SLAM can be implemented using 3D or 2D laser sensors, respectively producing 3D or 2D maps. 3D LIDARs are typically much more expensive than 2D LIDARs, but for applications such as the one proposed in this paper, the 2D data extracted by the 2D LIDAR is sufficient. More specifically, since in the proposed task, the validation of the 2D as-built maps is the primary goal, a 2D LIDAR profile recorded at a distance close to the ground can accurately report the extents of the floor dimensions.

To map an environment in 2D, one can resort to different SLAM flavors: while the comparison in [11] puts Cartographer [18] as the most accurate and robust 2D LIDAR-based SLAM system, the comparison made by [10] puts GMapping [5] as the best performer. In this paper, we opted for GMapping given how easy it was to set up on our mobile platform.

GMapping is a LIDAR-based SLAM implementation of a Rao-Blackwellized Particle Filter (RBPF) [15]. Here, a brief summary of the derivations used by [15] is made for completeness. GMapping estimates the trajectory, $x_{1:t} = x_1, \dots, x_t$, and the joint posterior, $p(x_{1:t}, m | z_{1:t}, u_{1:t-1})$, which makes use of the observations $z_{1:t} = z_1, \dots, z_t$, the odometry input $u_{1:t-1} = u_1, \dots, u_{t-1}$, and the map m . In order to make these estimations, the RBPF formulates the posterior as

$$p(x_{1:t}, m | z_{1:t}, u_{1:t-1}) = p(m | x_{1:t}, z_{1:t}) \cdot p(x_{1:t} | z_{1:t}, u_{1:t-1}),$$

where $p(m | x_{1:t}, z_{1:t})$ is computed analytically as described in [30]. On the other hand, $p(x_{1:t} | z_{1:t}, u_{1:t-1})$ is estimated using a particle filter, where prospective trajectories are represented as particles. The steps for Gmapping are as follows:

1. Each particle is sampled from the previous particle and an initial pose using a proposal distribution π .
2. Each particle is given an importance weight w_t according to the following:

$$w_t^i = \frac{\eta p(z_t | m_{t-1}^i, x_t^i) \cdot p(x_t^i | x_{t-1}^i, u_{t-1})}{\pi(x_t^i | x_{1:t-1}^i, z_{1:t}, u_{1:t-1})} w_{t-1}^i,$$

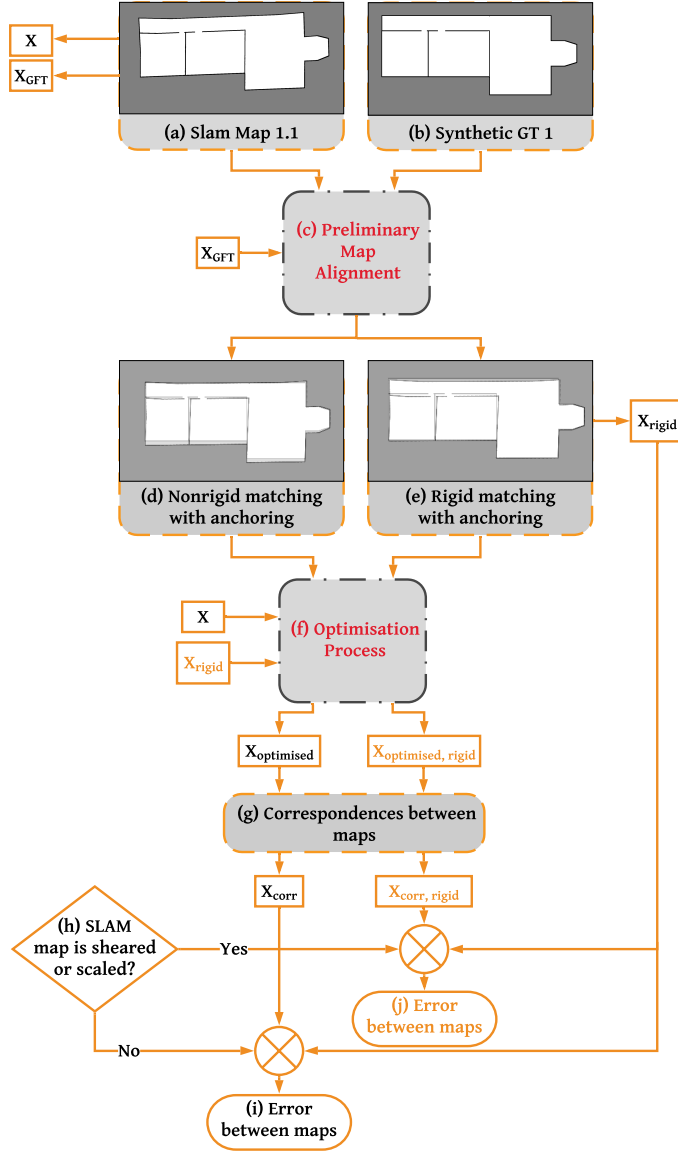


Fig. 1: Flowchart of our proposed system with a synthetic SLAM map

where the normalisation factor

$\eta = 1/p(z_t|z_{1:t-1}, u_{1:t-1})$ is the same for all particles.

3. If samples are far from the target distribution (i.e. sample weight lower than a certain threshold), other particles with higher weights are re-sampled to replace them. At this point, the particles have equal weights.

This is necessary due to the finite particle count used and the difference between the target distribution and the proposal.

4. Finally, $p(m^i|x_{1:t}^i, z_{1:t})$ is based on the sample's trajectory $x_{1:t}^i$ and all previous observations $z_{1:t}$.

In addition, GMapping uses occupancy grid mapping, which lies among the most popular probabilistic techniques used for map generation given a particular trajectory [41]. Generally, each cell in the occupancy grid represents a sector in the two-dimensional $x - y$ plane and the probability $p(m_{x,y})$ of that cell being occupied is represented by the intensity of its colour on the map. For the occupancy grid algorithm [42], the posteriori probability of the occupancy grid is calculated given the set of observations, and a Bayesian filter is applied to update this occupancy probability at each observation. In the proposed system, each generated two-dimensional map of the inspected site is represented by an occupancy grid, in which each cell exists in one of the following three distinct states: *occupied*, *unoccupied*, or *indeterminate*. The accuracy of the generated map can be gauged by its error with the ground truth, the Computer-Aided Design (CAD) map of the inspected site.

3.2 Map matching and validation

The proposed map matching and validation algorithm is illustrated in the upper part of Figure 1. In order to align both the SLAM and as-planned maps, they have to be represented in the same format. Therefore, the as-planned CAD drawing is first exported as an image at the same resolution as that of the SLAM map. Then, the pixels of the image are quantified, where cells are set to either occupied = 0, open = 255, or unexplored=127. This step is necessary to translate the as-planned CAD drawing into the standard format of the occupancy grid before comparing the two.

3.2.1 Preliminary alignment

Once the two maps are reduced to the same resolution and format, they are aligned using the method of Shahbandi and Magnusson [14]. In the first step, the maps are transformed into line representations using radiography (a variation of the Radon Transform [14]). The de-

tected lines, which represent wall structures in a map, are segmented such that each group of lines enclosing a face F represents an enclosure or room (Figure 2 (a) and (b)). To improve the segmentation of rooms, redundant lines are removed, the shape of each face is undistorted and represented as *Oriented Minimum Bounding Box* (OMBB) as depicted in Figure 2 (c) and (d).

Once the faces in each map are segmented, the maps are aligned by first matching faces from the SLAM maps to those in the as-planned maps, and then finding the most probable affine transformation between these two maps. Improbable transformations are rejected using a constraint of uniform scale in both coordinates (*i.e.*, similarity). The net result is the elimination of $\sim 90\%$ of possible alignment hypotheses. Finally, to select the optimal alignment, we rely on a fitness score, which is calculated using SLAM map occupied cells X_{GFT} representing "Good Features to Track" [36]; as a result, the optimal affine transformation $tform_{align}$ that maps X_{GFT} to its closest edge points in the as-planned map is selected.

Since this is an affine alignment, differences still exist between the two maps; an example of such differences can be seen in Figure 2 (e).

3.2.2 Anchoring

To align the two maps, one needs to choose two reference points on each map and align them further to those identified in another map using what we call anchoring. In each map, the two points are manually identified as corners that are distant from each other. Points should not be chosen in proximity to each other to avoid the effect that any local deformation could have on the entire map.

Once the points are identified, the source points P_{s1} and P_{s2} in the SLAM map are transformed using the final transformation from Section 3.2.1, $tform_{align}$, to obtain P_{s1a} and P_{s2a} as described in Equation (1). $tform_{anch}$ is defined in Equation (2), where ET represents a

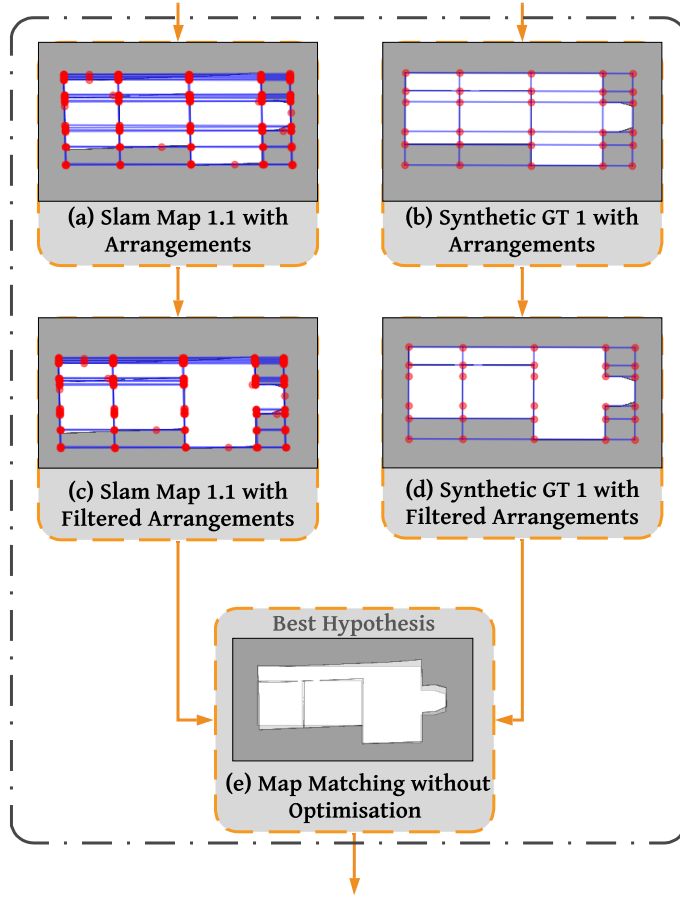


Fig. 2: Flowchart adapted from [14] for the preliminary map alignment, and performed on a synthetic SLAM map and its corresponding ground truth as-built map

Euclidean Transformation, using "least square estimation" [44], between P_{s1a}, P_{s2a} and the as-planned map anchor points P_{d1}, P_{d2} . The two transformations, $tform_{align}$ and $tform_{anch}$, are combined in (3) to form $tform_{align,anch}$. An example of the alignment of maps using $tform_{align,anch}$ can be seen in Figure 1 (d).

In addition to the above transformation, a combined rigid transformation is also used to calculate the error between the SLAM map and its corresponding as-planned map. This transformation is computed by repeating the same anchoring process but with a rigid varia-

tion of $tform_{align}$ and $tform_{anch}$, denoted by $tform_{align,rigid}$ and $tform_{anch,rigid}$ as shown in (4) and (5). Equation 6 combines these rigid transformations resulting in $tform_{align,anch,rigid}$. As an example, a rigid alignment after anchoring is depicted in Figure 1 (e). Following are the equations governing the anchoring process:

In order to correct the global inconsistencies and the local deformities of the SLAM map, edge points are deformed by minimising local errors between the maps as portrayed in Figure 1 (f). At this stage, X represents all

$$(P_{s1a}, P_{s2a}) = tform_{align}(P_{s1}, P_{s2}) \quad (1)$$

$$tform_{anch} = ET(Euclidean, (P_{s1a}, P_{s2a}), (P_{d1}, P_{d2})) \quad (2)$$

$$tform_{align,anch} = tform_{anch}.tform_{align} \quad (3)$$

$$(P_{s1a,rigid}, P_{s2a,rigid}) = tform_{align,rigid}(P_{s1}, P_{s2}) \quad (4)$$

$$tform_{anch,rigid} = ET(Euclidean, (P_{s1a,rigid}, P_{s2a,rigid}), (P_{d1}, P_{d2})) \quad (5)$$

$$tform_{align,anch,rigid} = tform_{align,rigid}.tform_{anch,rigid} \quad (6)$$

occupied cells using Canny edge detection [6]. Using edge points instead of "Good Features to Track" as in [35] to increase the accuracy of the calculated map errors.

3.2.3 Optimisation

In this section, the nonrigid case is adopted to explain the optimisation. However, before any optimisation is performed, the as-planned map M_o is transformed into a distance map through a distance transform, $M_d = DT(M_o)$, using the method described in [9]. The distance map is then normalised through a Gaussian function $M_f = [\exp \frac{-d_i^2}{2\sigma_f^2} \mid \forall d_i \in M_d]$, where M_f is lower for unoccupied cells that are farther from edges and higher for those that are closer. Here, σ_f represents the neighbourhood of the fitness map. As an example, for homes and offices $\sigma_f = 1 \pm 0.4$ meters. Finally, the Gaussian map is obtained as $M_g = \frac{\partial M_f}{\partial x} + i \frac{\partial M_f}{\partial y}$.

Once the as-planned map is transformed, the optimal motion matrix, dX of the SLAM map edge points X , is selected which maximises the Gaussian function:

$$dX = \operatorname{argmax}_{dX} \sum_{i=1}^K M_f(x_i + dx_i) \mid x \in X, dx \in dX \quad (7)$$

where the number of X cells is denoted by K . In order to align the SLAM map with the as-planned map, the convex hull of the points, X , is first tessellated into smaller triangles. Afterwards, for every triangle, an affine transformation is assigned based on the motion model matrix dX . This process is called a piece-wise

transformation. Moreover, to ensure coherency of the motion models within every neighbourhood dX is modified: $dx'_i = \frac{1}{K} \sum_{j=1}^K dx_j \cdot w_{ij}$, where

each dx_j is obtained from the gradient map. w_{ij} is a weight that represents the correlation of pairs of points, x_i and x_j :

$w_{ij} = \exp \frac{-\|x_i, x_j\|^2}{2\sigma_n^2} \mid x_i, x_j \in X$, where $\| \cdot \|$ is used to represent the Euclidean distance, and σ_n is between 0 and ∞ (*i.e.*, between absent and strict coherency). According to [35], $\sigma_n = 8 \pm 4$ meters. Figure 3 (a) shows the representation of the motion dx_i of the edge points.

The SLAM map edge points are now optimised to align with the as-planned map and are denoted as $X_{optimised}$. The resulting optimised alignment is shown in Figure 3 (b), and since the alignment is very accurate, the maps appear to be as one map. Similarly, for the rigid case, the motion of X_{rigid} is optimised, resulting in $X_{optimised,rigid}$.

Now that the type of maps have been defined, the fitness score mentioned in Section 3.2.1 can be formulated as follows:

$$fitness = \operatorname{mean}([M_f(x, y) \mid \forall (x, y) \in X]_{N \times 1}) \quad (8)$$

where $M_f \leftarrow$ as-planned map, $X \leftarrow$ sensor map, and $\sigma_f = 0.1$.

3.2.4 Error metric

Algorithm 1 lists the necessary steps to estimate the errors between the SLAM and as-

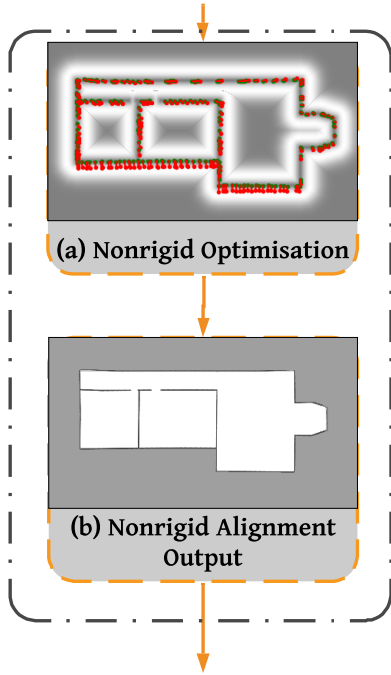


Fig. 3: Optimisation process adapted from [35]

planned maps. With each distance map is associated an array called *label*, where each pixel and its closest occupied pixel are labelled with the same number. Then, the locations of pixels with a unique *label* are extracted and denoted as $Coords_{all}$, which are transformed to image coordinates and represented as $Coords$. Pixels in the as-planned map having coordinates = $Coords$ are labelled as either (*Zero*) for edge pixels having a value of zero or (*NonZero*) for all other nonzero pixels.

Each *NonZero* pixel (x, y) corresponds to its closest *Zero* pixel (x_0, y_0) , and these correspondences are saved in an array called *Matches*, where every row represents (x, y, x_0, y_0) in the as-planned map. Lines 1 to 7 in Algorithm 1 show the transformation from *label* to *Matches*.

In order to calculate the error between the SLAM map and the as-planned map, edge pixels in the optimised SLAM map, $X_{optimised}$,

need to be matched with their closest edge pixels in the as-planned map, X_{corr} . Although *Matches* is generated for the as-planned image, it can also be utilised to find matches for $X_{optimised}$ since the optimised SLAM map and the as-planned map have the same size and resolution. To elaborate, the correspondence data in *Matches*, which references the as-planned map edge points, can be cross-referenced with $X_{optimised}$ to correspond them to the edge points in the as-planned map.

It is important to note that $X_{optimised}$ and X_{rigid} (*i.e.*, X after $tform_{align,anch,rigid}$ is applied) are ordered such that $X_{optimised,i}$ corresponds to $X_{rigid,i}$. Thus, $X_{rigid,i}$ corresponds to $X_{corr,i}$.

Then, the Euclidean distance between X_{rigid} and X_{corr} is calculated. The resulting distance represents the localised error vector (in pixels) between maps.

In fact, to modify this error to unit length, the error is multiplied by the resolution of the map and is represented as *Error*. Afterwards, to analyse these errors in the 5 section, the mean of the local error vector *Error* is calculated along with the maximum local error; which describe the average error and the maximum deformation.

In addition to the above, a similar procedure is performed where instead of using X to get $X_{optimised}$, X_{rigid} is used and $X_{optimised,rigid}$ is generated along with $X_{corr,rigid}$, which is portrayed in Figure 1 (g), along with its output in Figure 1 (j). This allows for an alternative algorithm for the case in which nonrigid alignment fails. Such nonrigid alignment may fail when there is a lack of sufficient rooms (faces) in the map which the alignment depends on. Namely, a limited amount of faces causes the number of alignment hypotheses to reduce; therefore, increasing the probability of getting an incorrect match. To correct for that, $X_{optimised,rigid}$ relies on the initial anchor points introduced earlier to optimise the alignment without nonrigid global transformations. At this stage, visual assessment of the image align-

Algorithm 1: Error Evaluation

```

1 ( $M_d$ , label) = D T( $M_o$ )
2 for  $iteration \in unique(label)$  do
3   Coords_all = where(label=iteration);
4   Coords = TransformtoImageCoordinates(Coords_all);
5   Zero=where(image(Coords)=0);
6   NonZero=where(image(Coords)!=0);
7   Matches =append(Matches, (NonZero, Zero));
   /* match is of the form ( $x, y, x_0, y_0$ ) where unoccupied cell  $x, y$  corresponds
   to occupied cell  $x_0, y_0$  */
8 for  $iteration \in X_{optimised}$  do
9   row = where(Matches(NonZero) =  $X_{optimised}$ );
10  if  $size(row)=0$  then
11     $Dist[i]$  = EuclideanDistance( $X_{rigid}, X_{optimised}$ );
12  else
13     $Dist[i]$  = EuclideanDistance( $X_{rigid}, Matches(row, 2 : 4)$ );
14 MeanError=mean(Dist)
15 MaxError=max(Dist)

```

ment results is made in order to choose between results using $X_{optimised}$ or $X_{optimised,rigid}$.

In cases where there is no global shear or scale distortion in the sensor SLAM map, and the induced nonrigid alignment forces an invalid transformation, the nonrigid alignment does not fail, but the rigid alignment is preferred. To elaborate, when it is known that the map does not have global shear and scale, the rigid method would be a better fit.

4 Experiments

In this section, we first propose the hardware used, the data that was experimented on, along with the procedures taken to obtain our results.

4.1 Proposed hardware

In the experiments, the Clearpath Husky (see Figure 4) equipped with a 2D Sick LIDAR and wheel encoders was used to estimate ego

motion. A robotic platform is needed to automate the system and its on-board sensors providing odometry information that improve the SLAM output. It is worth noting that any other robotic platform with a 2D LIDAR can be used. The system is driven by a Dell Latitude E6420 with an Intel Core i5-2520M CPU running at 2.50 GHz with a 16 GB RAM and Intel HD Graphics 3000. GMapping is available as a free library on Robot Operating System (ROS) [39].

4.2 Experimental Settings

To test and validate our proposed automated robotic assessment of 2D as-built floor plans, we prepared two sets of experiments, including the corridors in (1) the fourth floor of the Scientific Research Building (SRB) at the American University of Beirut (AUB), and (2) the fourth floor of the IOEC Building at AUB. These locations resemble construction sites, since they are void of furniture except for some manually added obstacles, such as boxes. In these sets of experiments, the trial runs traced the



Fig. 4: The Huskey robot

inner walls of the hallway which runs around the area. Afterwards, these maps were processed as seen in Figure 5 (a) and (d).

The ground truth of the as-built maps was collected through manual measurements using a laser meter [19] and an adjustable set square. However, after drawing the measurements, the loop didn't close due to imperfections in the walls and the angles (up to 1degree shifts). Errors could also have resulted from the inaccuracy of the set square in measuring angles. In addition to the ground truth as-built maps shown in Figure 5 (b) and (e), the as-planned drawings were also collected to assess our system shown in Figure 5 (c) and (f). In order to verify if the map generated from the SLAM system can be used in place of the as-built map (generated with traditional methods), the proposed system is not only performed on the SLAM map and as-planned map as described in Section 3, but also on the as-built map resulting in the following comparisons:

1. Compare the SLAM map to the as-built map that was obtained by manually taking measurements.
2. Compare the as-built map to the as-planned drawing to obtain the actual difference between the reality and the as-planned drawing.

3. Compare the SLAM map to the as-planned drawing to check if this comparison is similar to (2).

In addition to the generated SLAM maps, synthetic data is also used for further validation and analysis of the system. First, Map 1 in Figure 6 (a) and Map 2 in Figure 6 (e) from the HouseExpo dataset [43] are used as ground truth as-built floor plans. In addition, synthetic SLAM maps are generated by performing affine transformations and piece-wise local shearing, as described in Section 5.

5 Results and Discussion

In this section, the results and the validation of our system are analysed for the synthetic and SLAM data.

5.1 Synthetic data results

For each ground truth map, three synthetic maps were generated to represent SLAM maps; each map was generated using the following transformations:

1. Rotation, translation and low local shear resulting in Map 1.1 and 2.1 in Figure 6 (b) and (f).
2. Rotation, translation and high local shear resulting in Map 1.2 and 2.2 in Figure 6 (c) and (g).
3. Scale, shear, rotation, translation and low local shear resulting in Map 1.3 and 2.3 in Figure 6 (d) and (h).

These different variations are used to test the effect of local shear, and global shear and scale on the resulting errors between these maps and their ground truths.

The rigid and nonrigid alignments of the synthetic maps, which are followed by an optimization are shown in Figure 7, 8, 10, and 11. Subsequently, the errors between the synthetic maps are calculated and tabulated in Table 2. The results show that the error increases as local shear increases; this is evident from Maps

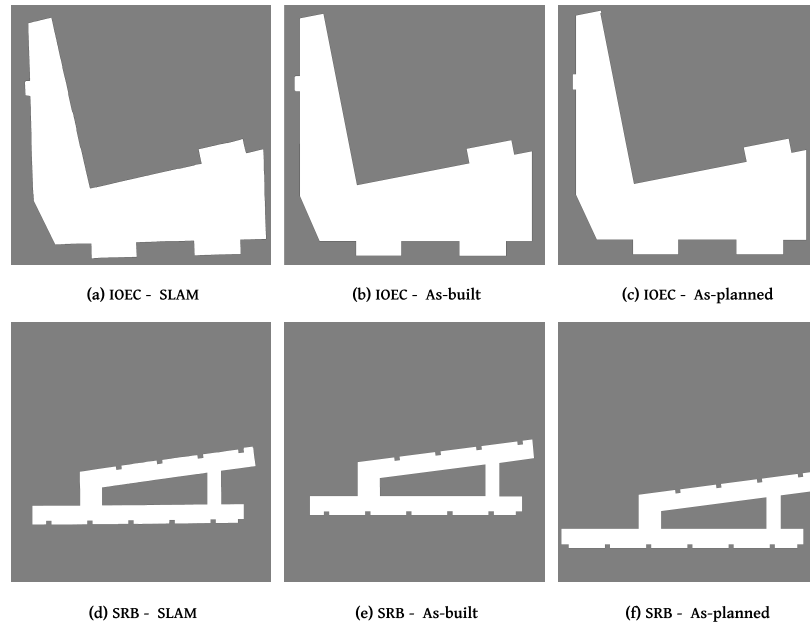


Fig. 5: Processed maps

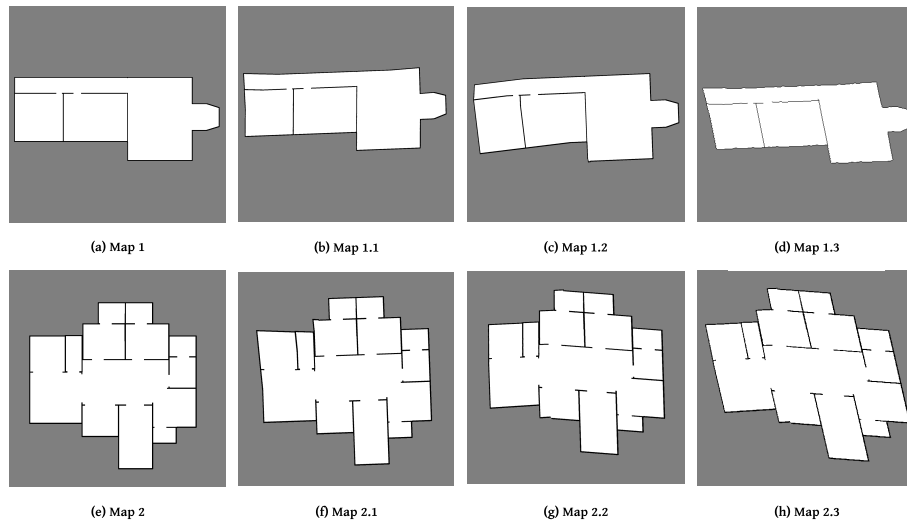


Fig. 6: Processed synthetic maps

1.1 and 2.1 in Figure 6 (b) and (f), where the errors are lower than those of Maps 1.2 and 2.2. Some of the maps have better optimisation re-

sults using rigid alignment due to the lack of faces in the maps as described in Section 3.2.

Table 2: Synthetic map errors according to their respective ground truths

Synthetic Map Error Values					
Synthetic SLAM Map	Mean Error $E_{nonrigid,opt}$ (pixels)	Max. Error $E_{nonrigid,opt}$ (pixels)	Mean Error $E_{rigid,opt}$ (pixels)	Max. Error $E_{rigid,opt}$ (pixels)	Validation Error (pixels)
Map 1.1	13.35	25.374	13.37	25.83	25.81
Map 1.2	18.24	74.03	18.76	73.65	79.84
Map 1.3	193.3	371.99	192.79	361.65	371.77
Map 2.1	3.34	19.14	3.28	19.14	21.97
Map 2.2	53.95	96.66	54.03	95.25	97.70
Map 2.3	120.56	252.12	80.77	179.20	256.90

Whereas, Map 2.3 and 1.3 fail visually with rigid alignment due to their global shear and scale, as shown in Figure 7 (c) and Figure 8 (c); these results are also further elaborated on in Section 3.2. In this case, the error values of the nonrigid alignment are used.

Afterwards, in order to ensure that the alignment process is producing acceptable error results, the maximum error is validated manually. Specifically, the maximum error from the rigid alignment was used for Map 1.1, 2.1, 1.2, and 2.2, Figure 6, since it is slightly more accurate than the nonrigid error given that there is no global shear and scale. For Map 1.3 and 2.3, nonrigid scores were used due to the presence of global shear and scale.

In most cases, the manual validation of the maximum error is very close to the algorithm output, with an average variation of $\pm 1.15\%$; with the exception of Synthetic Map 1.2 and 2.1, whose validation errors ensue an 8.40% and 14.80%, respectively. These results of Map 1.2 and 2.1, shown in Figure 7 (b) and 8 (a), respectively, are further investigated in Figure 9, which shows the correspondences used in calculating the error. The slightly shifted correspondences, shown in blue in Figure 9 (b) and (d), are due to the local shear in the map and its resulting imperfect optimisation. Specifically, high local shear renders a neighbourhood incoherent, and since optimisation works by trying to maintain the coherency between points, shifted correspondences can oc-

cur. Therefore, it can be concluded that these results are due to the few outliers of incorrect correspondences whose effect on the mean error is insignificant.

Another comparison was performed between the vanilla ICP method [47] and our proposed system. This additional step was made to further elaborate on the effectiveness of the algorithm versus typical error metrics. Map 2.3 was selected since it has several prominent types of deformations. The result of the ICP method shows a 57% variation from the actual maximum error; whereas, our algorithm gave a 1.9%. Thus, this system succeeds where ICP fails to give good correspondences.

5.2 SLAM results

Table 3 and Table 4 show the GMapping results obtained from the SRB fourth floor and the IOEC fourth floor, respectively. Since these maps, Figure 5, were generated using a LIDAR, we can disregard global scale. Global shear can also be disregarded, since the loop closure performed by GMapping did not result in any misalignment, or rather "brokenness", in the map. Therefore, it is sufficient to rely on $X_{optimised,rigid}$ to compare results, shown in Figure 12 and Figure 13. The nonrigid alignment and optimisation results are slightly less accurate and misaligned. This is visually clear in Figure 14 and Figure 15.

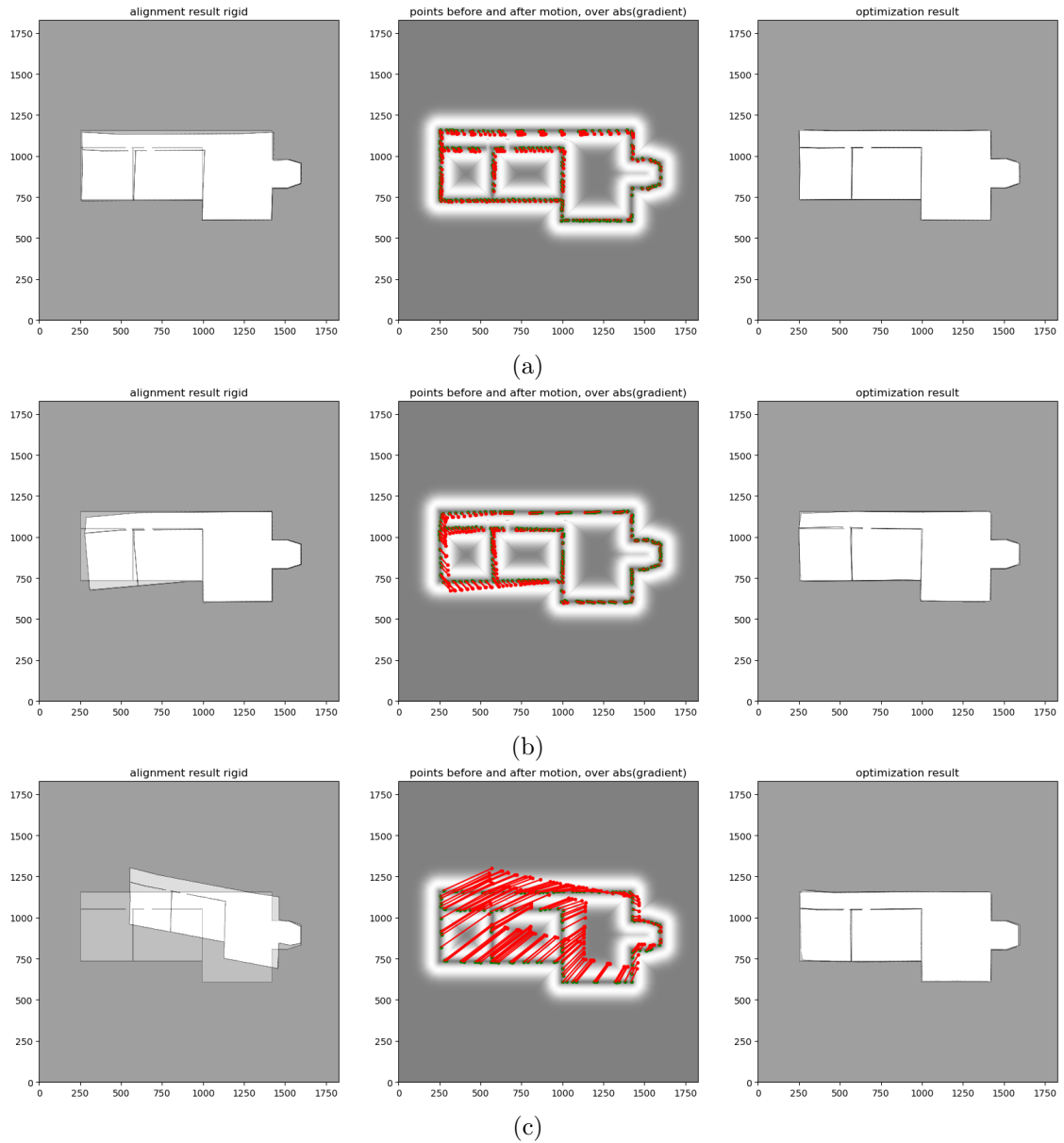


Fig. 7: Synthetic Map 1 rigid alignment and optimisation with (a) Map 1.1; (b) Map 1.2; (c) Map 1.3

Comparing SLAM and as-built maps, the mean errors for SRB and IOEC are 3.925 and 3.854 cm, respectively. These errors are due to the 2D LIDAR errors along with the SLAM system inaccuracies. The ANSI standards [38] for house measurements state that measure-

ments are to be rounded to the nearest one tenth of a foot, which equates to 3.048 cm. Thus, the errors from the SLAM maps that should be taken into consideration are the difference between the mean error and the tolerable error, equating to 0.877 and 0.806 cm

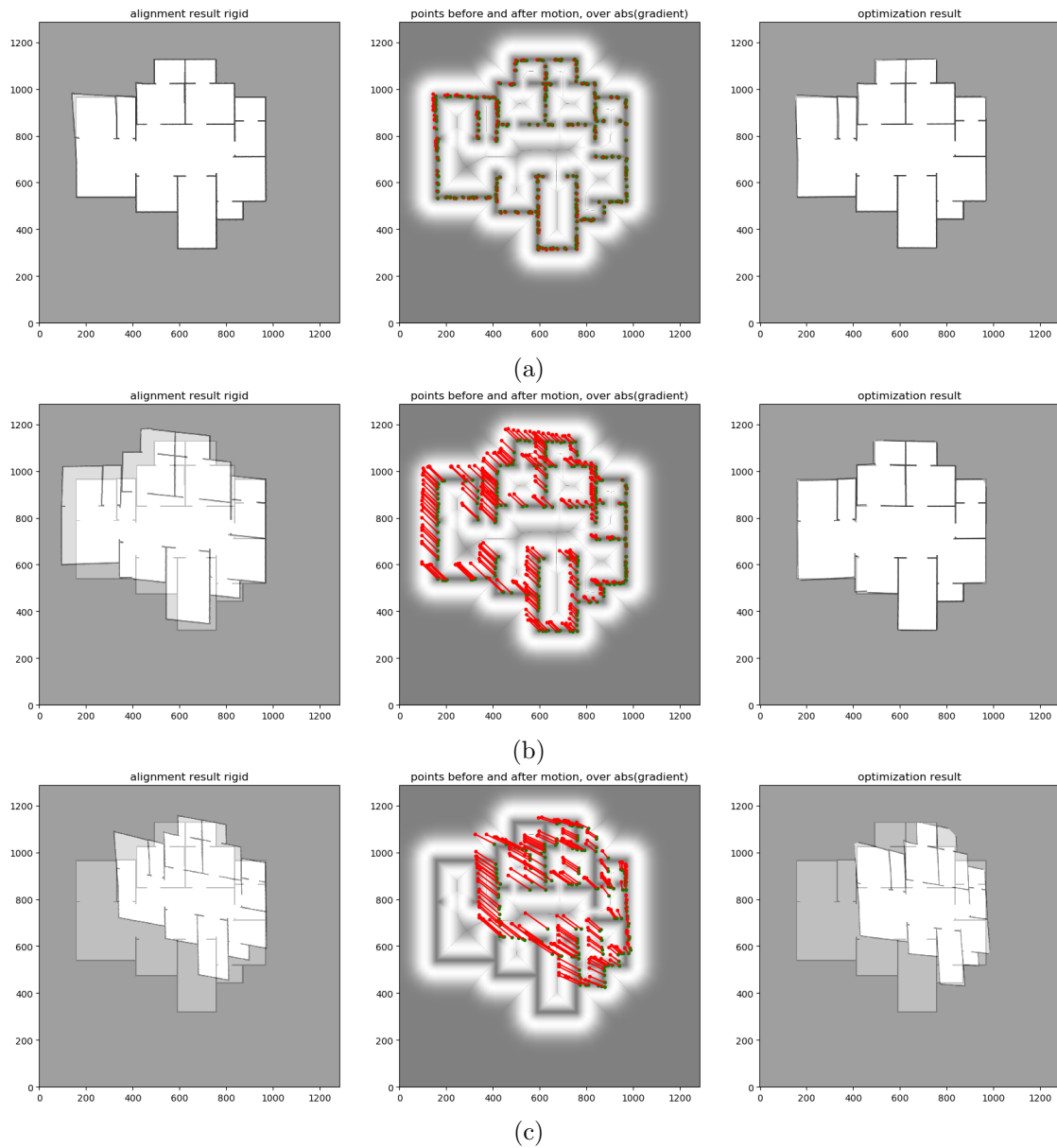


Fig. 8: Synthetic Map 2 rigid alignment with (a) Map 2.1; (b) Map 2.2; (c) Map 2.3

for SRB and IOEC, respectively. These are negligible errors and could be accounted for when using our system for obtaining the as-built map.

For further validation of the effectiveness of the system, the difference between the results of the as-built and as-planned maps versus the

results of the SLAM and as-planned maps provides practical insight. The SRB results show a difference of 0.329 cm, and IOEC shows 1.77 cm. These results are therefore small enough (less than 3.048 cm) to confirm that the SLAM map can be used in practice instead of the manually measured as-built map.

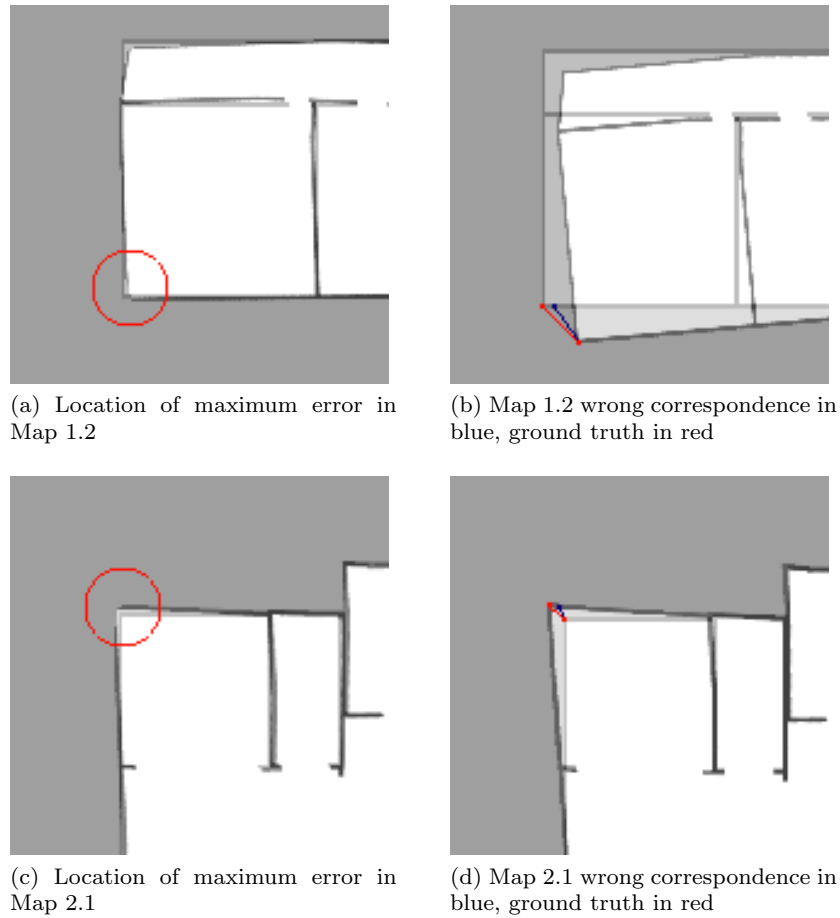


Fig. 9: Maximum error in synthetic maps

Overall, the proposed system is meant to generate an as-built map using SLAM to substitute traditional methods. This SLAM map is then compared to the as-planned map to check for errors that may have occurred throughout the construction process. In our results, we use a two-step verification process to verify the validity of using a robot as a surveyor: first, by comparing the SLAM maps to the as-built maps; and second, by comparing the error of the as-built maps versus as-planned maps to the error of the SLAM maps versus the as-planned maps. The first step resulted in an average of 3.890 cm, an error which is tolerable under ANSI standards mentioned previously.

The second step resulted in an average of 1.05 cm, an error that is negligible under the same standards.

5.3 Performance

It took approximately 30 minutes to perform an average run along the test bed hallways, where the localisation and mapping was implemented in C++ in real time. The data was then cleaned and fed to the map matching and error algorithm which was implemented in Python. Run-time increased as the size of the input images and the map deformations increased. For example, in spite of the fact that

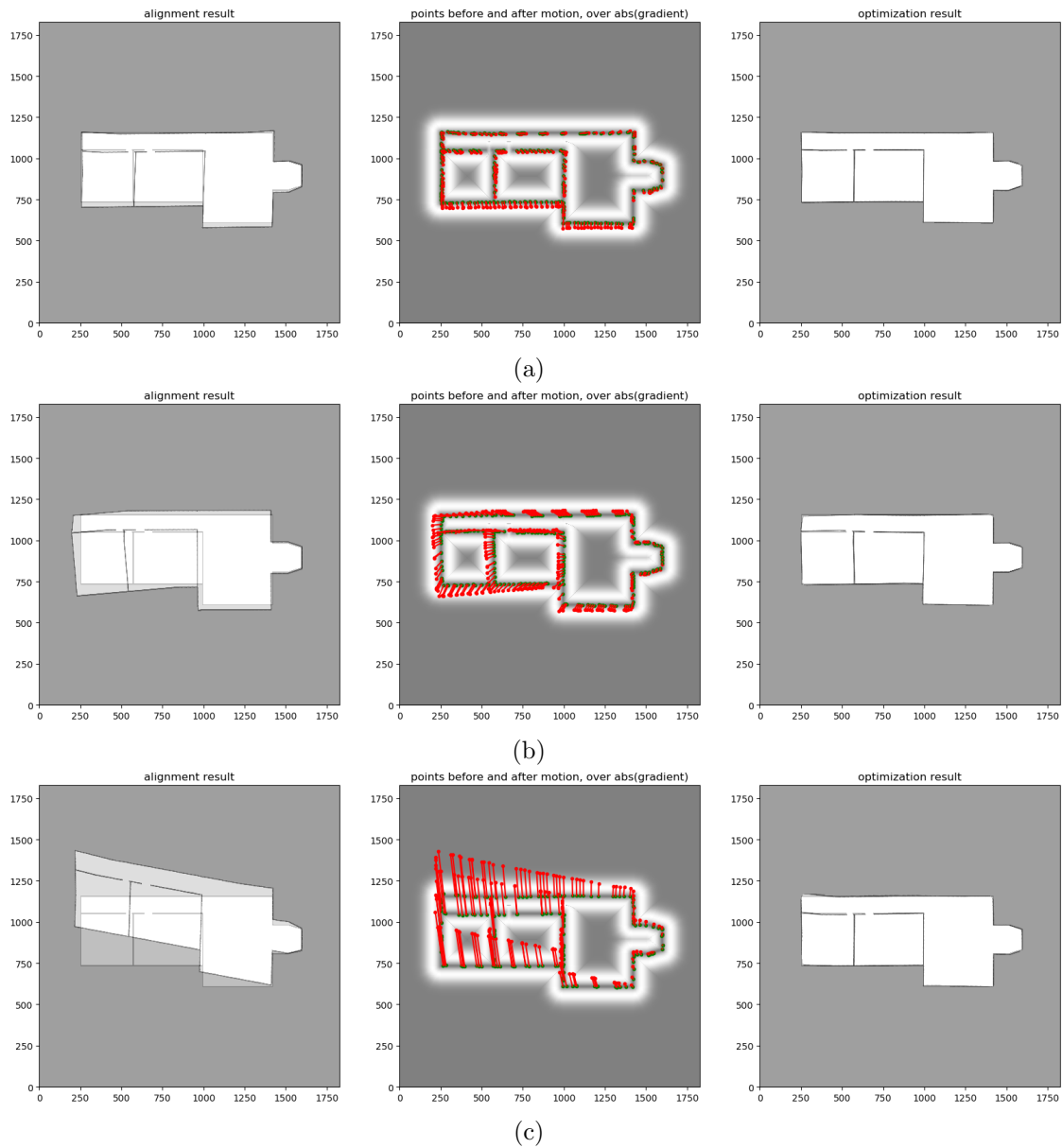


Fig. 10: Synthetic Map 1 nonrigid alignment and optimisation with (a) Map 1.1; (b) Map 1.2; (c) Map 1.3

three of the maps (Figure 5 (d), (e) and (f)) all had the same sizes (3228×3228 pixels), their run time ranged from 8.67 hrs to 16.94 hrs. All the processing was implemented on a Toshiba Satellite Intel Core i7-5500U running

at 2.3/3.0 Turbo GHz with an 8GB RAM and Intel HD Graphics 5500.

These findings indicate that the system has the potential to greatly reduce the effort, time, and cost incurred by the as-built mapping and verification task.

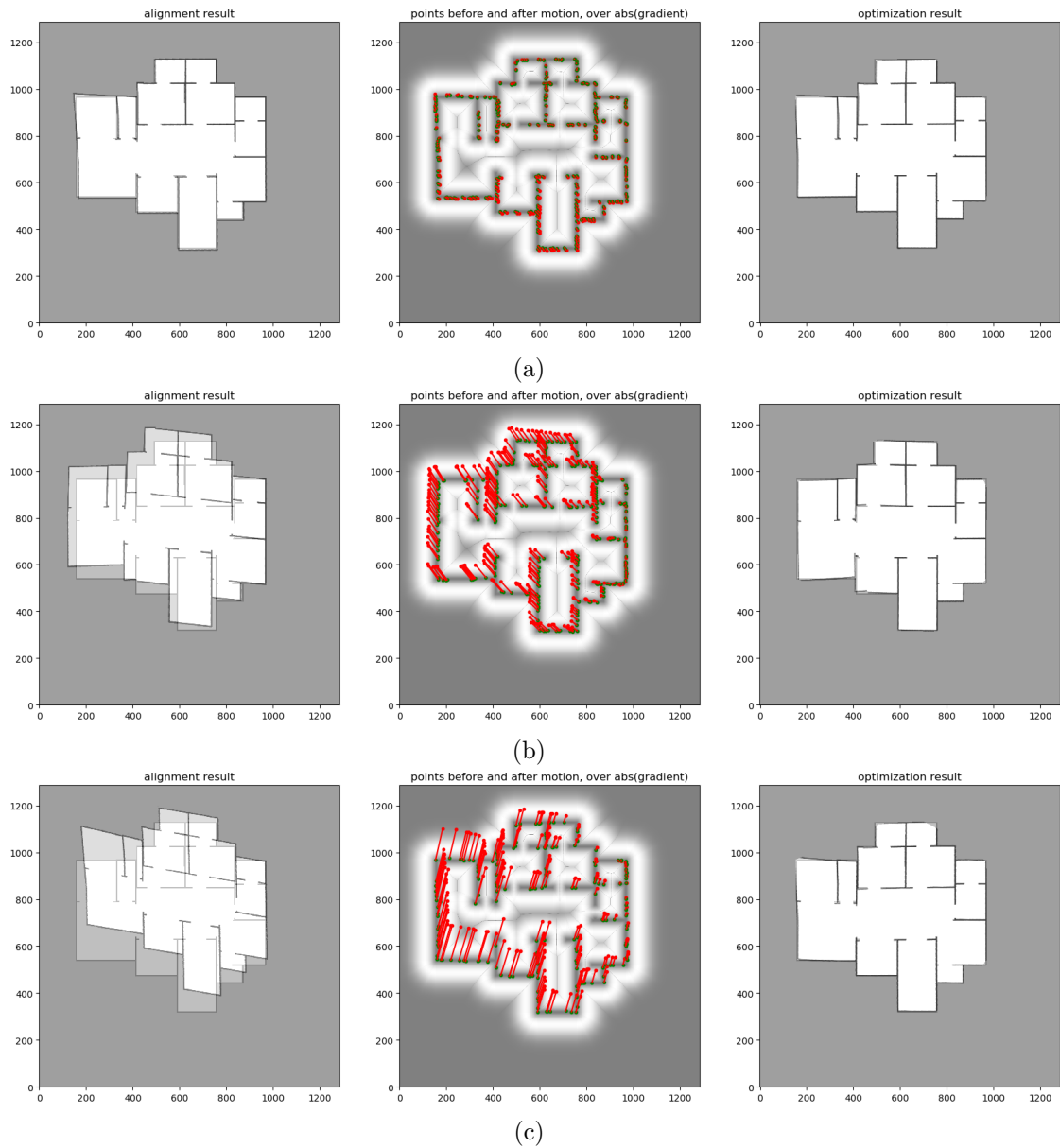


Fig. 11: Synthetic Map 2 nonrigid alignment and optimisation results with (a) Map 2.1; (b) Map 2.2; (c) Map 2.3

6 Conclusions, Limitations, and Future Work

This paper demonstrated how an infrastructure-less system can be effectively employed to automate the correctness verification of as-built

floor plans in a construction environment. Throughout the paper we showed how to automatically generate a 2D as-built floor plan of the site and quantify the errors in the generated map, how anchoring renders alignment more accurate, and how the proposed error metric accu-

Table 3: SRB fourth floor map error values. As-planned map (Apm); as-built map (Abm); mean error (m.error); max error (x.error)

SRB Fourth Floor Map Error Values					
Map 1	Map 2	m.error $E_{nonrigid,opt}$ (cm)	x.error $E_{nonrigid,opt}$ (cm)	m.error $E_{rigid,opt}$ (cm)	x.error $E_{rigid,opt}$ (cm)
SLAM Map	Abm	5.294	16.901	3.925	12.574
SLAM Map	Apm	5.240	120.302	5.204	119.984
Abm	Apm	8.689	139.878	4.875	123.923

Table 4: IOEC fourth floor map error values. As-planned map (Apm); as-built map (Abm); mean error (m.error); max error (x.error)

IOEC Fourth Floor Map Error Values					
Map 1	Map 2	m.error $E_{nonrigid,opt}$ (cm)	x.error $E_{nonrigid,opt}$ (cm)	m.error $E_{rigid,opt}$ (cm)	x.error $E_{rigid,opt}$ (cm)
SLAM Map	Abm	3.736	8.347	3.854	8.963
SLAM Map	Apm	12.39	29.319	12.538	29.319
Abm	Apm	10.721	27.101	10.767	24.407

rately evaluates the generated as-built maps. In addition, the applicability of this system to real construction sites is also demonstrated, where the system errors were within the acceptable limits. The proposed system is versatile, in that the laser measurement system can be mounted on a mobile robot or on a human construction site inspector. Since it is based on a 2D instead of a 3D laser, it is simple to apply, relatively cheap, and verification can be achieved automatically with low processing overhead.

Despite the widespread availability of laser measurement systems, the intended application of this research in construction inspection tasks is still in its infancy. It is therefore crucial to develop systems, such as that proposed in this paper. In the future we will work on addressing the limitations of this system through (1) conducting further tests to tackle the lack of experiments in construction sites at their different stages. This will add extra mapping challenges such as occlusions, uncontrolled ter-

rain, edge detection, and movements in the environment. (2) Another direction of the future work will be towards addressing the manual aspect of the anchoring method through devising an automated alternative.

7 Acknowledgements

Funding the research for this publication was provided by a grant from the University Research Board (URB) at the American University of Beirut.

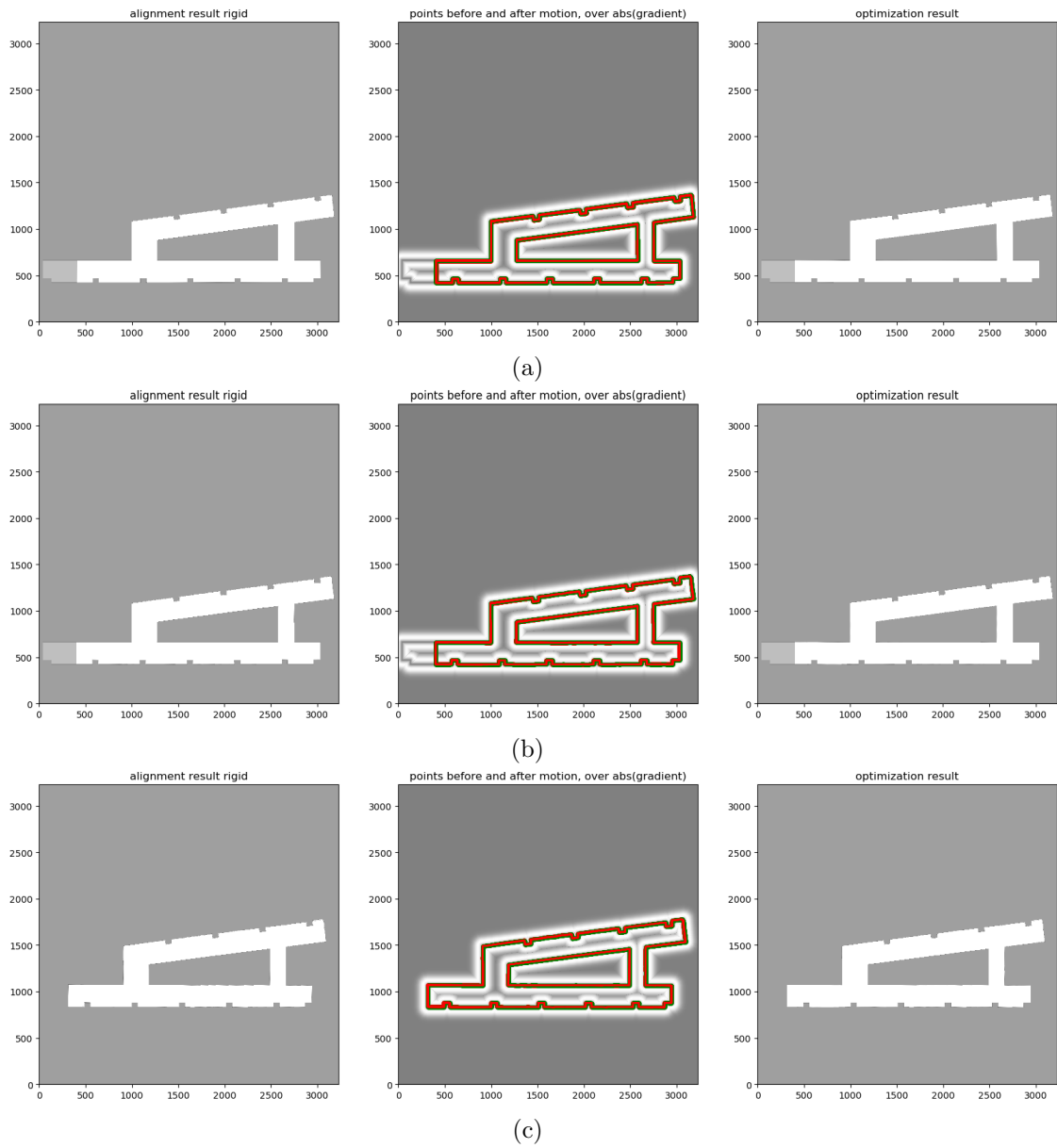


Fig. 12: SRB fourth floor rigid alignment between: (a) as-built and as-planned; (b) SLAM and as-planned; (c) SLAM and as-built

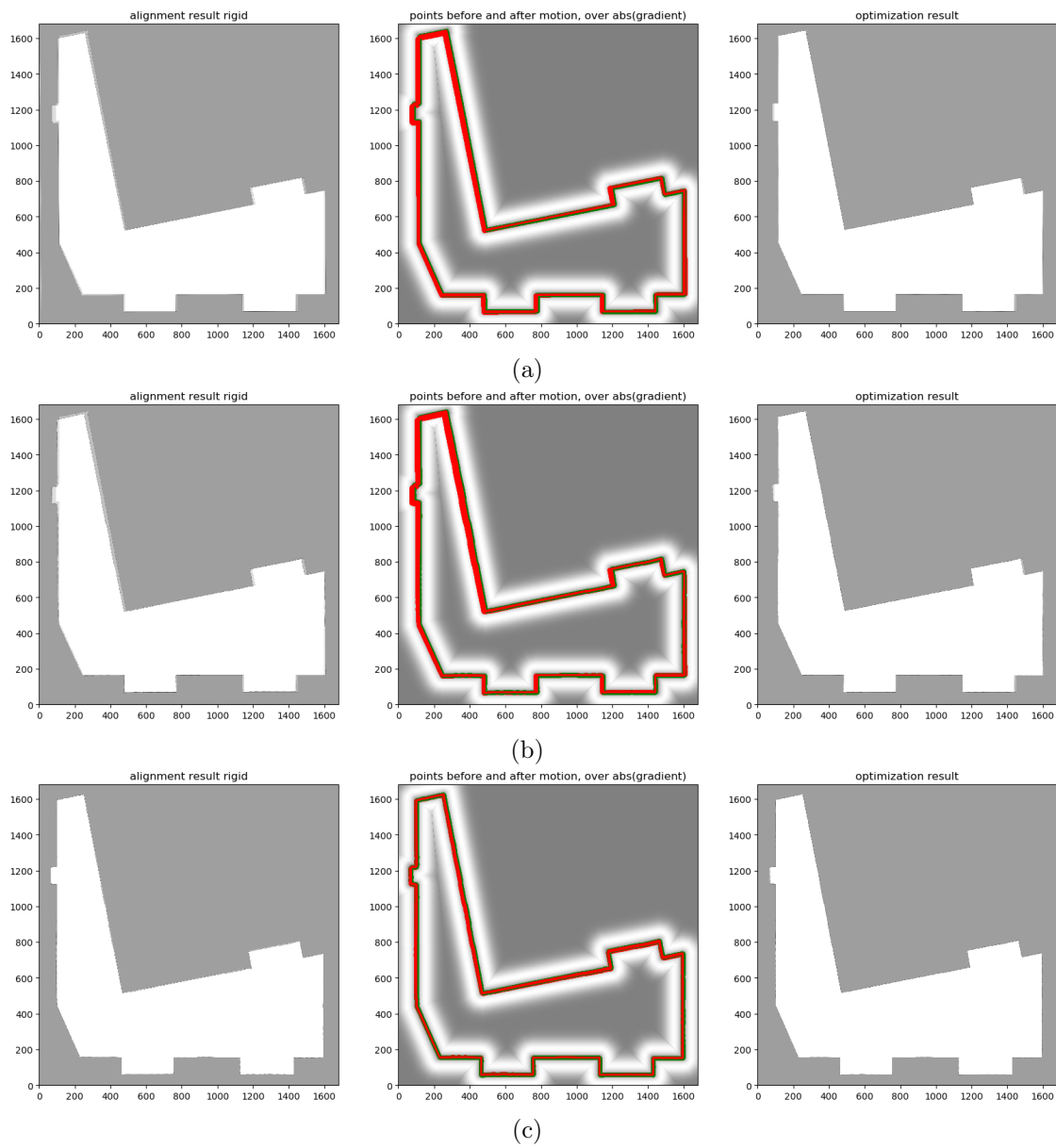


Fig. 13: IOEC fourth floor rigid alignment between: (a) as-built and as-planned; (b) SLAM and as-planned; (c) SLAM and as-built

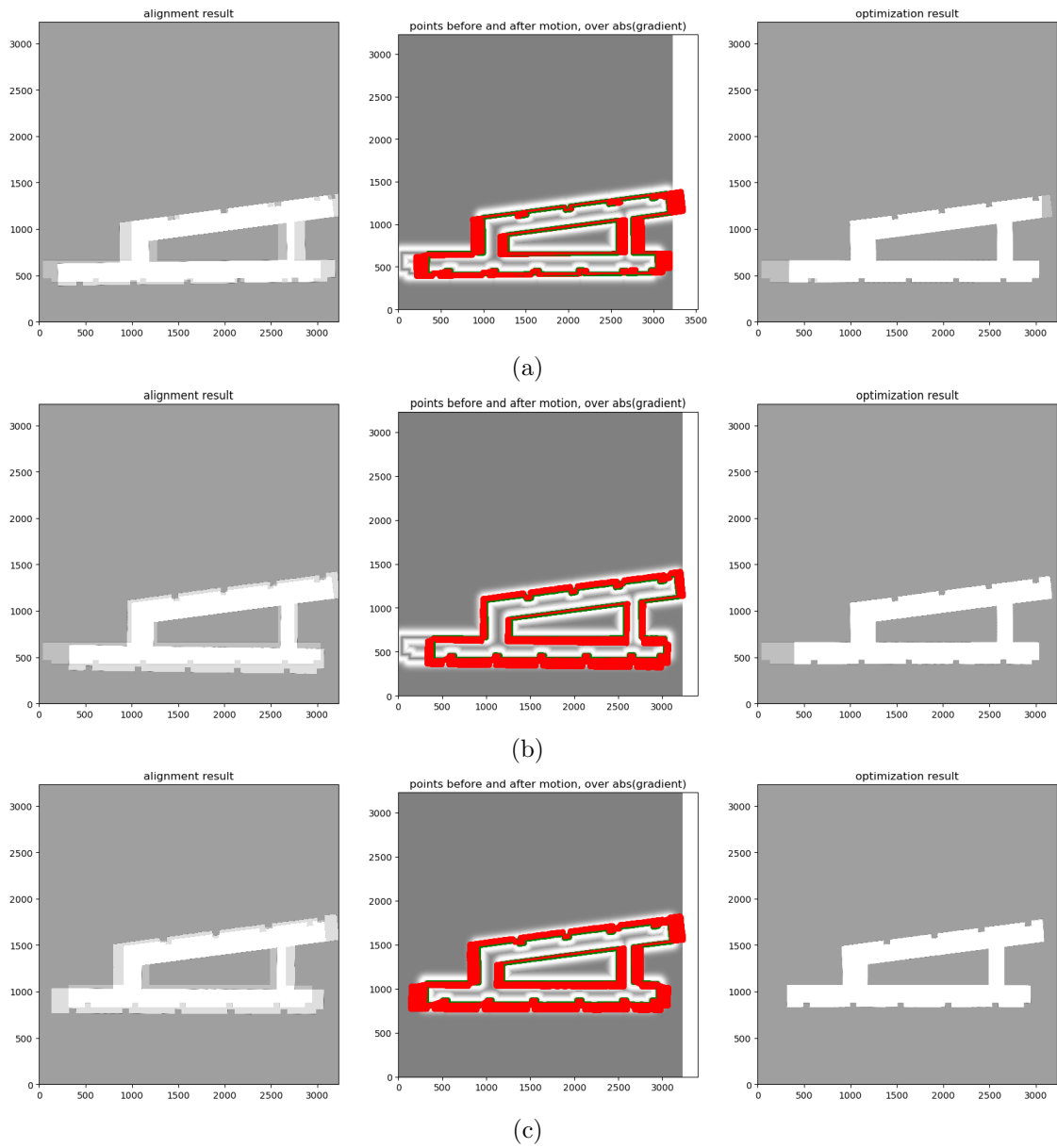


Fig. 14: SRB fourth floor nonrigid alignment between: (a) as-built and as-planned; (b) SLAM and as-planned; (c) SLAM and as-built

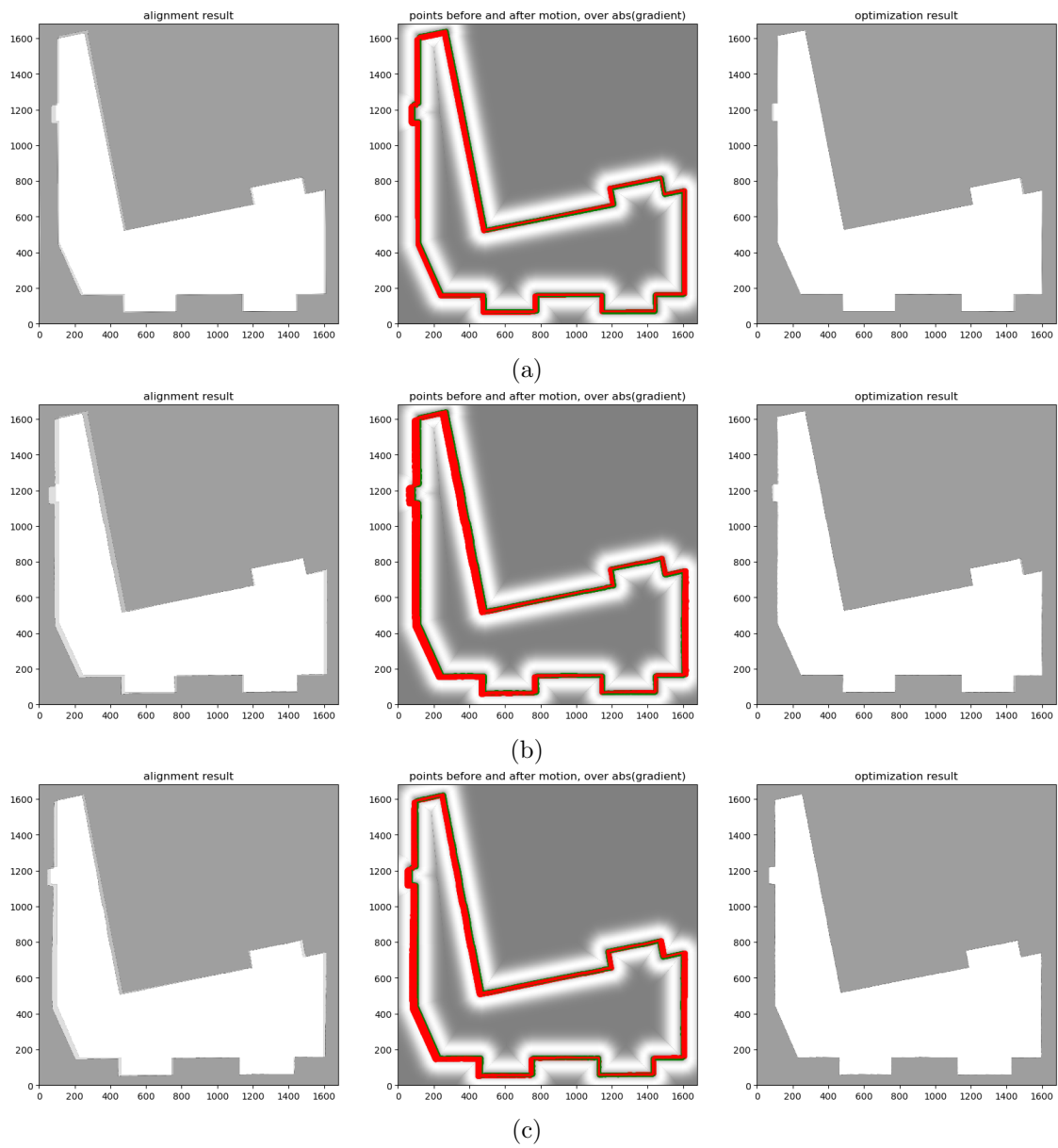


Fig. 15: IOEC fourth floor nonrigid alignment between: (a) as-built and as-planned; (b) SLAM and as-planned; (c) SLAM and as-built

References

1. Adán, A., Quintana, B., Prieto, S., Bosché, F.: An autonomous robotic platform for automatic extraction of detailed semantic models of buildings. *Automation in Construction* **109**, 102963 (2020)
2. Asadi, K., Ramshankar, H., Noghabaei, M., Han, K.: Real-time image localization and registration with bim using perspective alignment for indoor monitoring of construction. *Journal of Computing in Civil Engineering* **33**(5), 04019031 (2019)
3. Bosché, F.: Automated recognition of 3d cad model objects in laser scans and calculation of as-built dimensions for dimensional compliance control in construction. *Advanced Engineering Informatics* **24**(1), 107–118 (January, 2010)
4. Boukamp, F., Akinici, B.: Automated processing of construction specifications to support inspection and quality control. *Automation in Construction* **17**, 90–106 (2007)
5. Brian Gerkey ROS Wiki: gmapping. <https://wiki.ros.org/gmapping> (2007). [Online; accessed 4-October-2019]
6. Canny, J.: A computational approach to edge detection. In: *Readings in computer vision*, pp. 184–203. Elsevier (1987)
7. Durrant-Whyte, H., Bailey, T.: Simultaneous localisation and mapping: part 1. *Robotics and Automation Magazine* **13**(2), 99–110 (2006)
8. Everett, J.: Back to basics in construction automation. In: *International Association for Automation and Robotics in Construction*, pp. 583–590 (1990)
9. Felzenszwalb, P.F., Huttenlocher, D.P.: Distance transforms of sampled functions. *Theory of computing* **8**(1), 415–428 (2012)
10. Filatov, A., Filatov, A., Krinkin, K., Chen, B., Molodan, D.: 2d slam quality evaluation methods. In: *2017 21st Conference of Open Innovations Association (FRUCT)*, pp. 120–126. IEEE (2017)
11. Filipenko, M., Afanasyev, I.: Comparison of various slam systems for mobile robot in an indoor environment. In: *International Conference on Intelligent Systems* (2018)
12. Gallaher, M.P., O'Connor, A.C., Dettbarn, J.L.J., Gilday, L.T.: Cost analysis of inadequate interoperability in the u.s. capital facilities industry. Tech. rep., National Institute of Standards and Technology (2004)
13. Garrido, G.G.: Development of a tightly-coupled composite vision/laser sensor for indoor slam. Ph.D. thesis, École Nationale Supérieure des Mines de Paris (2011)
14. Gholami Shahbandi, S., Magnusson, M.: 2d map alignment with region decomposition. *Autonomous Robots* (2017). DOI 10.1007/s10514-018-9785-7
15. Grisetti, G., Stachniss, C., Burgard, W., et al.: Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE transactions on Robotics* **23**(1), 34 (2007)
16. Hähnel, D., Burgard, W., Thrun, S.: Learning compact 3d models of indoor and outdoor environments with a mobile robot. *Robotics and Autonomous Systems* **44**, 15–27 (2003)
17. Han, K., Degol, J., Golparvar-Fard, M.: Geometry-and appearance-based reasoning of construction progress monitoring. *Journal of Construction Engineering and Management* **144**(2), 04017110 (2018)
18. Hess, W., Kohler, D., Rapp, H., Andor, D.: Real-time loop closure in 2d lidar slam. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1271–1278. IEEE (2016)
19. Hilti: Laser meters. URL https://www.hilti.com/c/CLS_MEA_TOOL_INSERT_7127/CLS_LASER_METERS_7127. [Online; accessed 4-October-2019]
20. Jiang, G., Yin, L., Jin, S., Tian, C., Ma, X., Ou, Y.: A simultaneous localization and mapping (slam) framework for 2.5d map building based on low-cost lidar and vision fusion. *Applied Sciences* **9**(10) (2019). DOI 10.3390/app9102105
21. Jung, J., Hong, S., Yoon, S., Kim, J., Heo, J.: Automated 3d wireframe modeling of indoor structures from point clouds using constrained least-squares adjustment for as-built bim. *Journal of Computing in Civil Engineering* **30**(4), 04015074 (2015)
22. Kim, Y.S., Oh, S.W., Cho, Y.K., Seo, J.W.: A pda and wireless web-integrated system for quality inspection and defect management of apartment housing projects. *Automation in Construction* **17**, 163–179 (2008)
23. Klein, L., Li, N., Becerik-Gerber, B.: Imaged-based verification of as-built documentation of operational buildings. *Automation in Construction* **21**, 161–171 (2012)
24. Kopsida, M., Brilakis, I.: Real-time volume-to-plane comparison for mixed reality-based progress monitoring. *Journal of Computing in Civil Engineering* **34**(4), 04020016 (2020)
25. Kosub, S.: A note on the triangle inequality for the jaccard distance. *Pattern Recognition Letters* **120**, 36–38 (2019)
26. Lakaemper, R., Madhavan, R.: Towards evaluating world modeling for autonomous navigation in unstructured and dynamic environments. In: *Proceedings of the 10th Performance Metrics for Intelligent Systems Workshop*, pp. 355–360 (2010)
27. Macher, H., Landes, T., Grussenmeyer, P.: From point clouds to building information models: 3d semi-automatic reconstruction of indoors of existing buildings. *Applied Sciences* **7**(10), 1030 (2017)

28. Michael Schwind: Comparing Lidar and Photogrammetric Point Clouds. <https://www.gim-international.com/content/article/comparing-lidar-and-photogrammetric-point-clouds>. [Online; accessed 10-October-2019]
29. Moltke, I., Frambøll, C.K.: Quick guide to construction automation and robotics. <http://adaptablehouse.vtt.fi/> (2006)
30. Moravec, H.P.: Sensor fusion in certainty grids for mobile robots. In: *Sensor devices and systems for robotics*, pp. 253–276. Springer (1989)
31. Paterson, A.M., Dowling, G.R., Chamberlain, D.A.: Building inspection: Can computer vision help? *Automation in Construction* **7**, 13–20 (1997)
32. Pătrăucean, V., Armeni, I., Nahangi, M., Yeu-ung, J., Brilakis, I., Haas, C.: State of research in automatic as-built modelling. *Advanced Engineering Informatics* **29**(2), 162–171 (2015)
33. Sanhudo, L., Ramos, N.M., Martins, J.P., Almeida, R.M., Barreira, E., Simões, M.L., Cardoso, V.: Building information modeling for energy retrofitting—a review. *Renewable and Sustainable Energy Reviews* **89**, 249–260 (2018)
34. Santos, J.M., Portugal, D., Rocha, R.P.: An evaluation of 2d slam techniques available in robot operating system. In: *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 1–6. IEEE (2013)
35. Shahbandi, S.G., Magnusson, M., Iagnemma, K.: Nonlinear optimization of multimodal two-dimensional map alignment with application to prior knowledge transfer. *IEEE Robotics and Automation Letters* **3**(3), 2040–2047 (2018)
36. Shi, J., Tomasi, C.: Good features to track. Tech. rep., Cornell University (1993)
37. Son, H., Na, J., Kim, C.: Semantic as-built 3d modeling of buildings under construction from laser-scan data based on local convexity without an as-planned model. In: *ISARC. Proceedings of the International Symposium on Automation and Robotics in Construction*, vol. 32, p. 1. IAARC Publications (2015)
38. SRA, B.M.: The ansi z765 standard for calculating square footage. *The Appraisal Journal* **81**(4), 300 (2013)
39. Stanford Artificial Intelligence Laboratory et al.: Robotic operating system. <https://www.ros.org> (2018)
40. Thomas, B.: Icwgb. <http://www.icwgb.org/> (2012)
41. Thrun, S.: Learning occupancy grid maps with forward sensor models. *Autonomous Robots* **15**, 111–127 (2003)
42. Thrun, S., Bucken, A.: Integrating grid-based and topological maps for mobile robot navigation. In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pp. 944–951 (1996)
43. Tingguang, L., Danny, H., Chenming, L., De-long, Z., Chaoqun, W., Meng, M.Q.H.: House-expo: A large-scale 2d indoor layout dataset for learning-based algorithms on mobile robots. arXiv preprint arXiv:1903.09845 (2019)
44. Umeyama, S.: Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis & Machine Intelligence* **4**, 376–380 (1991)
45. Wei, Y., Akinci, B.: A vision and learning-based indoor localization and semantic mapping framework for facility operations and management. *Automation in Construction* **107**, 102915 (2019)
46. Xiong, X., Adan, A., Akinci, B., Huber, D.: Automatic creation of semantically rich 3d building models from laser scanner data. *Automation in Construction* **31**, 325–337 (2013)
47. Yagfarov, R., Ivanou, M., Afanasyev, I.: Map comparison of lidar-based 2d slam algorithms using precise ground truth. In: *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pp. 1979–1983. IEEE (2018)
48. Yoon, S., Jung, J., Heo, J.: Practical implementation of semi-automated as-built bim creation for complex indoor environments. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* **40**(4), 143 (2015)

Bibliography

- [1] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, “Image-to-image translation with conditional adversarial networks,” in *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, 2017.
- [2] K. Tango, M. Katsurai, H. Maki, and R. Goto, “Anime-to-real clothing: Cosplay costume generation via image-to-image translation,” *arXiv preprint arXiv:2008.11479*, 2020.
- [3] R. Daher, Y. Hawari, and D. Asmar, *Automated Robotic Assessment of 2D As-built Floor Plans*.
- [4] E. Daniel, *Path Planning and Optimization on SLAM-Based Maps*. PhD thesis, University of Stuttgart, 2016.
- [5] R. Daher, T. Chakhachiro, and D. Asmar, *A Comparative Assessment Method for Map Alignment Techniques*.
- [6] C. Agüero, N. Koenig, I. Chen, H. Boyer, S. Peters, J. Hsu, B. Gerkey, S. Paepcke, J. Rivero, J. Manzo, E. Krotkov, and G. Pratt, “Inside the virtual robotics challenge: Simulating real-time robotic disaster response,” *Automation Science and Engineering, IEEE Transactions on*, vol. 12, pp. 494–506, April 2015.
- [7] R. Vaughan, “Massively multi-robot simulation in stage,” *Swarm intelligence*, vol. 2, no. 2-4, pp. 189–208, 2008.
- [8] I. G. Alonso, M. Fernández, J. M. Maestre, and M. d. P. A. G. Fuente, *Service robotics within the digital home: applications and future prospects*, vol. 53. Springer Science & Business Media, 2011.
- [9] L. Chang, X. Niu, T. Liu, J. Tang, and C. Qian, “Gnss/ins/lidar-slam integrated navigation system based on graph optimization,” *Remote Sensing*, vol. 11, no. 9, p. 1009, 2019.
- [10] H. Ren, Q. Yan, Z. Liu, Z. Zuo, Q. Xu, F. Li, and C. Song, “Study on analysis from sources of error for airborne lidar,” in *IOP Conference Series: Earth and Environmental Science*, vol. 46, p. 012030, IOP Publishing, 2016.

- [11] M. Segata, R. L. Cigno, R. K. Bhadani, M. Bunting, and J. Sprinkle, “A lidar error model for cooperative driving simulations,” in *2018 IEEE Vehicular Networking Conference (VNC)*, pp. 1–8, IEEE, 2018.
- [12] W. Liu, “Lidar-imu time delay calibration based on iterative closest point and iterated sigma point kalman filter,” *Sensors*, vol. 17, no. 3, p. 539, 2017.
- [13] D. Mader, P. Westfeld, and H.-G. Maas, “An integrated flexible self-calibration approach for 2d laser scanning range finders applied to the hokuyo utm-30lx-ew,” *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences*, vol. 45, 2014.
- [14] R. B. Sousa, M. R. Petry, and A. P. Moreira, “Evolution of odometry calibration methods for ground mobile robots,” in *2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pp. 294–299, IEEE, 2020.
- [15] J. Borenstein and L. Feng, “Measurement and correction of systematic odometry errors in mobile robots,” *IEEE Transactions on robotics and automation*, vol. 12, no. 6, pp. 869–880, 1996.
- [16] T. Abbas, M. Arif, and W. Ahmed, “Measurement and correction of systematic odometry errors caused by kinematics imperfections in mobile robots,” in *2006 SICE-ICASE International Joint Conference*, pp. 2073–2078, IEEE, 2006.
- [17] A. Bostani, A. Vakili, and T. A. Denidni, “A novel method to measure and correct the odometry errors in mobile robots,” in *2008 Canadian Conference on Electrical and Computer Engineering*, pp. 000897–000900, IEEE, 2008.
- [18] K. Lee, C. Jung, and W. Chung, “Accurate calibration of kinematic parameters for two wheel differential mobile robots,” *Journal of mechanical science and technology*, vol. 25, no. 6, p. 1603, 2011.
- [19] C. Jung and W. Chung, “Accurate calibration of two wheel differential mobile robots by using experimental heading errors,” in *2012 IEEE International Conference on Robotics and Automation*, pp. 4533–4538, IEEE, 2012.
- [20] D. L. Tomasi and E. Todt, “Rotational odometry calibration for differential robot platforms,” in *2017 Latin American Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR)*, pp. 1–6, IEEE, 2017.
- [21] E. Ivanjko, I. Komsic, and I. Petrovic, “Simple off-line odometry calibration of differential drive mobile robots,” in *Proceedings of 16th Int. Workshop on Robotics in Alpe-Adria-Danube Region-RAAD*, 2007.

- [22] G. Antonelli, S. Chiaverini, and G. Fusco, “A calibration method for odometry of mobile robots based on the least-squares technique: theory and experimental validation,” *IEEE Transactions on Robotics*, vol. 21, no. 5, pp. 994–1004, 2005.
- [23] S. Mondal, Y. Yun, and W. K. Chung, “Terminal iterative learning control for calibrating systematic odometry errors in mobile robots,” in *2010 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pp. 311–316, IEEE, 2010.
- [24] A. Censi, A. Franchi, L. Marchionni, and G. Oriolo, “Simultaneous calibration of odometry and sensor parameters for mobile robots,” *IEEE Transactions on Robotics*, vol. 29, no. 2, pp. 475–492, 2013.
- [25] G. Goronzy and H. Hellbrueck, “Weighted online calibration for odometry of mobile robots,” in *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*, pp. 1036–1042, IEEE, 2017.
- [26] A. Martinelli, N. Tomatis, A. Tapus, and R. Siegwart, “Simultaneous localization and odometry calibration for mobile robot,” in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, vol. 2, pp. 1499–1504, IEEE, 2003.
- [27] D. Caltabiano, G. Muscato, and F. Russo, “Localization and self-calibration of a robot for volcano exploration,” in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA’04. 2004*, vol. 1, pp. 586–591, IEEE, 2004.
- [28] L. Cantelli, S. Ligama, G. Muscato, and D. Spina, “Auto-calibration methods of kinematic parameters and magnetometer offset for the localization of a tracked mobile robot,” *Robotics*, vol. 5, no. 4, p. 23, 2016.
- [29] Y. Maddahi, N. Sepehri, A. Maddahi, and M. Abdolmohammadi, “Calibration of wheeled mobile robots with differential drive mechanisms: An experimental approach,” *Robotica*, vol. 30, no. 6, pp. 1029–1039, 2012.
- [30] H. Xu and J. J. Collins, “Estimating the odometry error of a mobile robot by neural networks,” in *2009 International Conference on Machine Learning and Applications*, pp. 378–385, 2009.
- [31] A. Pablo, “Slam + machine learning ushers in the ”age of perception” - robotics business review,” 2020.
- [32] A. Davison, “Augmenting slam with deep learning,” May 2019.

- [33] D. Li, X. Shi, Q. Long, S. Liu, W. Yang, F. Wang, Q. Wei, and F. Qiao, “Dxslam: A robust and efficient visual slam system with deep features,” *arXiv preprint arXiv:2008.05416*, 2020.
- [34] M. Antoun, D. Asmar, and R. Daher, “Towards richer 3d reference maps in urban scenes,” in *2020 17th Conference on Computer and Robot Vision (CRV)*, pp. 39–45, IEEE Computer Society, 2020.
- [35] L. Hu, W. Xu, K. Huang, and L. Kneip, “Deep-slam++: Object-level rgbd slam based on class-specific deep shape priors,” *arXiv preprint arXiv:1907.09691*, 2019.
- [36] M. S. Bahraini, A. B. Rad, and M. Bozorg, “Slam in dynamic environments: A deep learning approach for moving object tracking using ml-ransac algorithm,” *Sensors*, vol. 19, no. 17, p. 3699, 2019.
- [37] X. Qi, S. Yang, and Y. Yan, “Deep learning based semantic labelling of 3d point cloud in visual slam,” in *IOP Conference Series: Materials Science and Engineering*, vol. 428, p. 012023, 2018.
- [38] K. R. Konda and R. Memisevic, “Learning visual odometry with a convolutional network,” in *VISAPP (1)*, pp. 486–490, 2015.
- [39] K. Tateno, F. Tombari, I. Laina, and N. Navab, “Cnn-slam: Real-time dense monocular slam with learned depth prediction,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6243–6252, 2017.
- [40] Z. Qin, M. Yin, G. Li, and F. Yang, “Sp-flow: Self-supervised optical flow correspondence point prediction for real-time slam,” *Computer Aided Geometric Design*, vol. 82, p. 101928, 2020.
- [41] X. Chen, T. Läbe, A. Milioto, T. Röhling, O. Vysotska, A. Haag, J. Behley, C. Stachniss, and F. Fraunhofer, “Overlapnet: Loop closing for lidar-based slam,” *Proceedings of the Robotics: Science and Systems (RSS), Freiburg, Germany*, pp. 12–16, 2020.
- [42] R. Li, S. Wang, and D. Gu, “Deepslam: A robust monocular slam system with unsupervised deep learning,” *IEEE Transactions on Industrial Electronics*, 2020.
- [43] M. A. Hossain and M. M. Ali, “Recognition of handwritten digit using convolutional neural network (cnn),” *Global Journal of Computer Science and Technology*, 2019.

- [44] A. Dabouei, H. Kazemi, S. M. Iranmanesh, J. Dawson, and N. M. Nasrabadi, “Fingerprint distortion rectification using deep convolutional neural networks,” in *2018 International Conference on Biometrics (ICB)*, pp. 1–8, IEEE, 2018.
- [45] X. Li, B. Zhang, P. V. Sander, and J. Liao, “Blind geometric distortion correction on images through deep learning,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4855–4864, 2019.
- [46] V. Rengarajan, Y. Balaji, and A. Rajagopalan, “Unrolling the shutter: Cnn to correct motion distortions,” in *Proceedings of the IEEE Conference on computer Vision and Pattern Recognition*, pp. 2291–2299, 2017.
- [47] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in neural information processing systems*, pp. 2672–2680, 2014.
- [48] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241, Springer, 2015.
- [49] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro, “High-resolution image synthesis and semantic manipulation with conditional gans,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8798–8807, 2018.
- [50] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, “Spectral normalization for generative adversarial networks,” *arXiv preprint arXiv:1802.05957*, 2018.
- [51] D. Yoo, N. Kim, S. Park, A. S. Paek, and I. S. Kweon, “Pixel-level domain transfer,” in *European Conference on Computer Vision*, pp. 517–532, Springer, 2016.
- [52] D.-Z. Du and P. M. Pardalos, *Minimax and applications*, vol. 4. Springer Science & Business Media, 2013.
- [53] T. Karras, T. Aila, S. Laine, and J. Lehtinen, “Progressive growing of gans for improved quality, stability, and variation,” *arXiv preprint arXiv:1710.10196*, 2017.
- [54] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

- [55] S. Thrun, “Probabilistic robotics,” *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.
- [56] L. Zhang, *Self-Adaptive Markov Localization for Single-Robot and Multi-Robot Systems*. PhD thesis, 2010.
- [57] S. Riisgaard and M. R. Blas, “Slam for dummies,” *A Tutorial Approach to Simultaneous Localization and Mapping*, vol. 22, no. 1-127, p. 126, 2003.
- [58] A. Souza, A. Medeiros, L. Gonçalves, and A. Santana, “Probabilistic mapping by fusion of range-finders sensors and odometry,” *Sensor Fusion and Its Applications*, pp. 423–442, 2010.
- [59] MATLAB, *9.9.0.1467703 (R2020b)*. Natick, Massachusetts: The MathWorks Inc., 2020.
- [60] T. Li, D. Ho, C. Li, D. Zhu, C. Wang, and M. Q.-H. Meng, “Houseexpo: A large-scale 2d indoor layout dataset for learning-based algorithms on mobile robots,” *arXiv preprint arXiv:1903.09845*, 2019.
- [61] T. Hearn, “Stl tools.” https://github.com/theearn/stl_tools, 2013.
- [62] J. Horner, “Explore lite.” http://wiki.ros.org/explore_lite, 2010.
- [63] B. Gerkey and T. Pratkanis, “Map server.” http://wiki.ros.org/map_server, 2009.
- [64] P. R. Reddy, V. Amarnadh, and M. Bhaskar, “Evaluation of stopping criterion in contour tracing algorithms,” *International Journal of Computer Science and Information Technologies*, vol. 3, no. 3, pp. 3888–3894, 2012.
- [65] P. Geurts, D. Ernst, and L. Wehenkel, “Extremely randomized trees,” *Machine learning*, vol. 63, no. 1, pp. 3–42, 2006.
- [66] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.