# From SLAM to CAD Maps and Back Using Generative Models*

Rema Daher, Theodor Chakhachiro, and Daniel Asmar[1]

***Abstract*— In Simultaneous Localization and Mapping (SLAM), good quality maps could play an important role beyond the implicit benefits they have to the localization problem. For example, good quality SLAM maps could be used as 'pre-maps' for generating maps with a higher semantic meaning and geometric structure, such as in 2D CAD maps in which doors, windows, and other entities carry meaning. Unfortunately, in its raw form, the quality of a SLAM map one can obtain is limited by several factors, some due to software and others due to hardware. This paper proposes to address this problem by converting raw SLAM maps to CAD maps based on a generative adversarial network. The paper also investigates the inverse problem, that of generating SLAM maps from 2D CAD drawings. We do so by investigating two approaches, one based on an analytical model, the other on a generative adversarial network. Experiments demonstrate both qualitatively and quantitatively the success of the proposed approaches.**

## I. INTRODUCTION

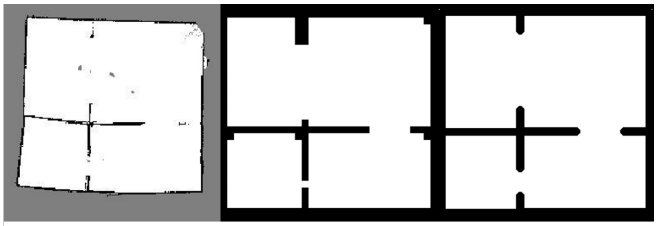
Autonomous mapping of an environment has become an important topic that some researchers have recently taken interest in. What they have been mostly focusing on is generating accurate and efficient maps. Examples include the mapping of civil engineering construction sites in order to survey the progress of work throughout, or to assess the conformity of as-is constructions to as-planned architectural drawings. Unfortunately, the low quality of maps generated by current SLAM solutions limits their use in their raw form to little beyond localization, and as such require considerable post-processing to transform them into maps possessing geometric and semantic meaning.

In this paper, we propose addressing this problem by directly transforming raw SLAM maps to 2D CAD maps, without the need to improve the SLAM algorithm itself nor endow the autonomous robot with high quality sensors. Instead, we propose learning the transformation between raw SLAM maps and 2D CAD maps using a generative adversarial network (Fig. 1).

To teach our proposed neural network, a dataset of corresponding SLAM and CAD maps is needed. We struggled to find any collection of SLAM maps in the literature that we could use for the sake of training. We therefore found it necessary to generate our own dataset of SLAM maps with corresponding CAD ground truth maps that we used to train our proposed machine learning algorithm.

*This work was supported by the University Research Board at the American University of Beirut (AUB)

Rema Daher, Theodor Chakhachiro, and Daniel Asmar are with the Vision and Robotics Lab at the Maroun Semaan Faculty of Engineering and Architecture, American University of Beirut, 1107-2020, Beirut, Lebanon rgd05@mail.aub.edu; tgc02@mail.aub.edu; da20@aub.edu.lb



(a) Simulated SLAM map (b) Generated CAD map (c) Target CAD map

Fig. 1. Automatically generating a CAD map using machine learning (middle) from a SLAM map (left); ground truth CAD map on the right.

Given the relatively high number of maps we needed for training, and the relatively high overhead of creating real SLAM maps using real robots, we resorted instead to a robotic simulator to create the desired maps. Although this solution was advantageous to using a real robot, the time necessary to run each of the simulations was also grueling, and the processing power and memory usage were also high. This experience led us to investigate and propose two solutions alternative to simulations to create SLAM maps, one based on parametric modelling and the other on machine learning. Through parametric modelling, multiple hypotheses of SLAM maps from one CAD map can be generated each using a different robot trajectory. This is opposed to the learning based approach where the trajectory is not taken into consideration, resulting in only one possible SLAM map hypothesis that is independent of any specific robot trajectory. On the flip side, the learning technique allows for a deeper understanding and modeling of the effect of the shape of the environment on the output error.

The contributions of this paper includes the following:

1) an automated system to transform SLAM maps to CAD drawings using machine learning,
2) an automated system to transform CAD maps into a SLAM maps using machine learning on one hand or parametric modelling on the other,
3) creation of a publicly available dataset of SLAM- and CAD-map pairs

These contributions are made possible through machine learning and parametric modeling, which requires us to delve into the literature for both.

### A. Machine Learning

To correct SLAM maps using machine learning, image to image translation needs to be performed. In our application, both the input and target images are 2D occupancy grid maps, in which piecewise deformations can occur such as in transformations of handwritten digits to printed digit

[1], or that of geometrically rectifying digital imagery [2]. However both of these applications are relatively simple requiring relatively simple architectures. Our application is more complex and requires a more convoluted architecture.

Another application is that of rectifying fingerprints [3]. This technique requires the training dataset to include displacement fields, which in our case is not available because the maps are generated in real time inside the simulators.

The application in which the transformation best lends itself to our problem is that of anime to real-life clothing, where colors mainly stay the same and local and global deformations are present. The state of the art in anime to clothing is [4], which builds its network on the most widely used architecture in image to image translation known as pix2pix [5], but with a novel consistency loss.

### B. Parametric Modeling

To automatically generate a robotic map from CAD maps in a parametric manner, the modeling of SLAM error sources becomes of prime interest. This review will focus on modelling methods for calibration of exteroceptive and proprioceptive (2D LIDAR and odometry) sensors mounted on a differentially driven robot, such as the one used in our work.

Previous work in the literature exist on the modeling of 3D LIDARs [6] and on airborne systems [7], as opposed to this work's focus on 2D ground LIDARs. Other papers use a stochastic experimental approach [8] or a Kalman filter [9] for calibration. These approaches are not useful for us since they are experimental and don't rely on meaningful physical parameters. Conversely, we adopt the approach of Mader *et al*. [10], who formulate a model for LIDAR sensors and include errors of different sources and contributions.

On the odometry side, we consider wheel encoders as the onboard sensors and model their errors. Borenstein *et al*. [11] did not include scaling errors and assumed independence between wheelbase and wheel radius errors. In Abbas *et al*. [12], the effect of non-systematic errors on calibration is decreased by having the test path included as one of the parameters. As for the work of Lee *et al*. [13], dependency between wheelbase and wheel radius errors was introduced; however, only straight motion was included and some trigonometric simplifications were assumed as was done in [11]. Furthermore, [14] took into consideration scaling, wheel diameters, and wheelbase errors.

Other methods rely on a more general modelling of the system. Such methods include optimization and Kalman filter based methods [15], methods that don't identify the contribution of the different sources of error [16], and methods that generalize the errors by using corrective factors [17].

In our work, we adopt a more detailed parametric modelling approach such as in the methods of [13], [18], [19], which consider a dependency between wheel base and wheel radius errors. Given the stated superior performance, we adopt the method of Tomasi *et al*. [19].

## II. PROPOSED SLAM-TO-CAD MAPPING SYSTEM

The architecture used in our SLAM to CAD system is that of Tango *et al*. [4] (Fig. 2), which relies on a generative adversarial network (GAN). GANs are neural networks that are made up of two sub-models—a generator and a discriminator—, which compete against each other to eventually reach an optimal state where the output is new (generator role) but doesn't look fake (discriminator role) [20]. The anime2clothing GAN architecture in Fig.

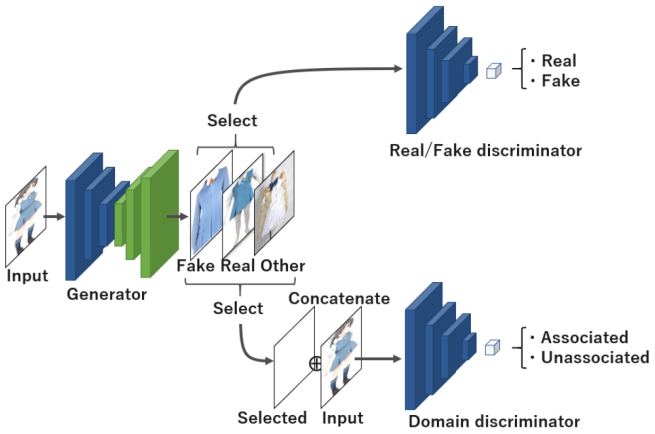



Fig. 2. Architecture of anime2clothing [4]

2 is made up of a generator $G$ with a U-net structure, a traditional real/fake discriminator, and an additional domain discriminator $D_d$. The domain discriminator has a multi-scale architecture, 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 in image scales. Spectral normalization [21] is also applied on the discriminators following the Lipschitz constraint. In addition, instances were augmented by adding false targets, which are simply targets paired with different inputs.

The domain discriminator $D_d$ is responsible for checking if the images are associated or not. The real/fake discriminator $D_r$ 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 $D_{r3}$. Using multiple patches allows for the modelling of the small and large scale shapes in the images. 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 $\min_X \max_Y f$. The objective function is a combination of the input consistency loss, the feature matching loss, and the GAN objectives:

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

$\omega$ is the importance weight and $\mathscr{L}$ the objective functions:

$$\mathscr{L}_{GAN_{domain}}(G,D_d) = \mathbb{E}_{(x,y)}[log(D_d(x,y))] + \mathbb{E}_x[log(1-D_d(x,G(x)))] \quad (2)$$

$$\mathscr{L}_{GAN_{real/fake}}(G,D_r) = \mathbb{E}_y[log(D_r(y))] + \mathbb{E}_x[log(1-D_r(G(x)))] \quad (3)$$

$$\mathscr{L}_{FM_{domain}}(G,D_d) = \mathbb{E}_{(x,y)} \sum_{i=1}^{T} N_i[||D_d^{(i)}(x,y) - D_d^{(i)}(x,G(x))||_1] \quad (4)$$

$$\mathscr{L}_{input_{real/fake}}(G,D_r) = \mathbb{E}_x \sum_{i=1}^{T} N_i[||D_r^{(i)}(y) - D_r^{(i)}(G(x))||_1] \quad (5)$$

$$\mathscr{L}_{L_1}(G) = \mathbb{E}_{(x,y)}[||y - G(x)||_1] \quad (6)$$

From these objective functions, $\mathscr{L}_{GAN_{real/fake}}$ is the traditional GAN objective function. $\mathscr{L}_{GAN_{domain}}$ is also a minimax game for image associations using augmented unassociated data. In addition, the feature matching loss, $\mathscr{L}_{FM_{domain}}$, 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 $\mathscr{L}_{input_{real/fake}}$, maintains shape and color locally, it is computed as the L1 loss between outputs of the intermediate layers of the real/fake discriminator. The L1 loss, $\mathscr{L}_{L_1}$, also maintains shape and color but on a global scale. 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 ith layer of the discriminator.

This model (*i.e.* $Model_{32}$) trains the images at 32 resolutions to capture only 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.

## III. Proposed CAD-to-SLAM Mapping System

In this section, the proposed AI- and parametric-based CAD to SLAM methods are explained.

### A. Machine Learning

The same architecture as that of the proposed SLAM-to-CAD mapping system is utilized but by switching the inputs for the outputs, and adopting a coarse-to-fine scheme. The generator and discriminators progressively grow by increasing systematically the resolution of the input image depending on the current number of epochs that are reached. This way, any gradient problems from high resolution images are alleviated, and the computation period is shortened. This model is denoted as $Model_{prog}$ and used for generating SLAM maps from CAD maps.

### B. Parametric Modelling

Our second approach to generate synthetic SLAM maps which relies on parametric modeling is shown in Fig. 3. First, the user defines a trajectory or a probabilistic roadmap is generated using the method described in [22] with a path between two random points. The orientation, velocity, and acceleration are then calculated using waypoints and time of arrival information. Then, the exteroceptive and proprioceptive error models are implemented on the CAD map. The deformation fields, which are 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.

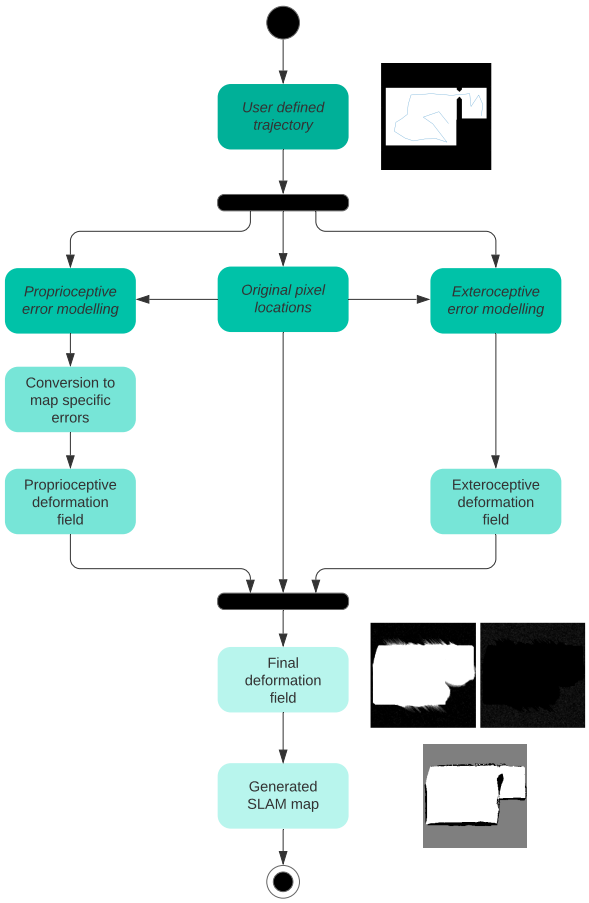



Fig. 3. Flowchart of the parametric modelling

*1) Exteroceptive Error Model:* The exteroceptive sensor (2D LIDAR) error is modelled by first defining the landmark coordinates $X_L = \begin{bmatrix} X_l \\ Y_l \end{bmatrix} - D \begin{bmatrix} cos\alpha \\ sin\alpha \end{bmatrix}$. $D$ is the measured slant range in meters, which is randomly assigned to every pixel in the image. Furthermore, $\alpha$ is the horizontal angle, which is assigned randomly to each pixel. $D$ and $\alpha$ values are then smoothed using a median filter, which helps in giving neighboring pixels similar range and angle readings.

However, $D$ and $\alpha$ are both subject to errors such that $\Delta D = a_0 + a_1 D$ and $\Delta\alpha = b_1\alpha + b_2 sin\alpha + b_3 cos\alpha + b_4 sin2\alpha + b_4 cos2\alpha + b_6 D^{-1}$. 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 [10], we can impose a bound on these parameters. The deformation field $(d_{xhit}, d_{yhit})$ due to the latter error formulation becomes: $d_{xhit} = pm(Dcos(\alpha) - (D - \Delta D)cos(\alpha - \Delta\alpha))$ and $d_{yhit} = pm(Dsin(\alpha) - (D - \Delta D)sin(\alpha - \Delta\alpha))$. Here, $pm$ is the pixels per meter ratio.

The error formulated above is that of small measurement

noise. However, it is shown in [23] 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:

$$P_{short} = \begin{cases} \frac{1}{1-e^{-\lambda z_k^{t*}}} \lambda e^{-\lambda z_k^t} & if\ 0 \le z_k^t \le z_k^{t*} \\ 0 & otherwise \end{cases} \quad (7)$$

$$P_{max} = \begin{cases} 1 & if\ z_k^t = z_{kmax}^t \\ 0 & otherwise \end{cases} \quad (8)$$

$$P_{rand} = \begin{cases} 1/z_{kmax}^t & if\ 0 \le z_k^t \le z_{kmax}^t \\ 0 & otherwise \end{cases} \quad (9)$$

$\lambda = 0.5$ is an intrinsic parameter by which the exponential curve's slope is obtained [24]. $P_{short}$ is assigned a random value for every landmark. $z_k^{t*}$ is the true object range, and has the values of $D$. From $P_{short}$ we calculate $z_k^t$ and $E_{short}$:

$$z_k^t = \frac{-log(\frac{P_{short}(1-e^{-\lambda z_k^{t*}})}{\lambda})}{\lambda};\ E_{short} = z_k^{t*} - z_k^t \quad (10)$$

$$d_{xshort} = pm(Dcos(\alpha) - E_{short} cos(\alpha - \Delta\alpha)) \quad (11)$$

$$d_{yshort} = pm(Dsin(\alpha) - E_{short} sin(\alpha - \Delta\alpha)) \quad (12)$$

From the calculated $z_k^t$, $P_{max}$ and $P_{rand}$ are also calculated from (9). With $Ran_{err}$ and $Max_{err}$ assigned a random error between an experimentally calculated range, $d_{xmax}, d_{ymax}, d_{xrand}, d_{yrand}$ are calculated similarly to $d_{xshort}, d_{yshort}$. Lastly, the final displacement field is a weighted sum of these displacement fields. $Z_{hit}, Z_{short}, Z_{max}, Z_{rand}$ are the relative error weights and are equal to (0.4, 0.3, 0.2, 0.1) respectively [24]. Moreover, the recalculated measure of $z_k^t$ and $\alpha$ becomes $z_{meas} = D - d_{xL}/cos(\alpha - \Delta\alpha)$ and $\alpha_{meas} = \alpha - \Delta\alpha$.

*2) Proprioceptive Error Model:* As for the odometry error model, from the trajectory, the nominal positions $x_{nom}, y_{nom}$ are extracted as well as angle of rotation of the wheels $\gamma$, incremental displacement of the robot $\Delta d_{Rnom}, \Delta d_{Lnom}$, and distance traveled for every straight line in trajectory $L$. From the selected robot, 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 $\Delta d$ 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 [19], [18] the values of $\alpha_{E_b}, \alpha_{E_d}, \beta$ are assigned their experimental standard deviation. The parameters described can be modelled as follows:

$$E_b = \frac{90}{90 - (\alpha_{E_b} + \alpha_{E_d})},\ b_{actual} = E_b b_{nominal} \quad (13)$$

$$R = \frac{L/2}{sin(\beta/2)},\ E_d = D_{Ract}/D_{Ract} = \frac{R + \frac{b_{actual}}{2}}{R - \frac{b_{actual}}{2}} \quad (14)$$

$$D_a = (D_{Rnom} + D_{Lnom})/2 \quad (15)$$

From equations of $D_a$ and $E_d$:

$$D_{Lact} = \frac{2}{E_d + 1} D_a,\ D_{Ract} = \frac{2}{(1/E_d) + 1} D_a \quad (16)$$

$$\Delta d_{Ract} = \gamma D_{Ract},\ \Delta d_{Lact} = \gamma D_{Lact} \quad (17)$$

$$\Delta d_{act} = \frac{\Delta d_{Ract} + \Delta d_{Lact}}{2},\ \Delta\theta_{act} = \frac{\Delta d_{Ract} - \Delta d_{Lact}}{b_{actual}} \quad (18)$$

$$\Delta x_{act} = \Delta d_{act} cos(\theta_{k-1} + \Delta\theta_{act}/2) \quad (19)$$

$$\Delta y_{act} = \Delta d_{act} sin(\theta_{k-1} + \Delta\theta_{act}/2) \quad (20)$$

The same is derived for the nominal counterparts. 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 similarly $\sigma_x = \Delta x_{nom} - \Delta x_{act}$.

*a) 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 [25]. However, in EKF SLAM, the need to calculate the inverse of the Covariance makes it computationally expensive. As for the method proposed in [26], the linear and angular odometry errors are calculated using empirical data collected in two phases to model the effect of both linear and angular displacements.

After the conversion of the odometry error to a map specific error, the deformation fields $(d_{xL}, d_{yL})$ and $(d_{xO}, d_{yO})$ are added to form $(d_x, d_y)$, which is used to deform the map accordingly.

## IV. EXPERIMENTS

For all simulations Gazebo and Turtlebot3 Waffle Pi robot were used with LDS-01 LIDAR of 180 degree sensing range. For the machine learning method, the training was done on Nvidia V100 PCI-E GPU with 128GB RAM for the CAD-to-SLAM model and 15GB RAM for the SLAM-to-CAD model, which used half the batch rate. As for the parametric modelling, Matlab was used on a Toshiba Satellite Laptop with an Intel core i7-5500U CPU with 2.40GHz and 8GB RAM.

### A. Data Generation

The data generation for machine learning relies on the HouseExpo dataset CAD maps (35,000 maps) [27]. These maps were converted into 3D environments using [28] and then automatically converted into Gazebo world files. The platform used was the American University of Beirut's (AUB) HPC cluster with AMD EPYC™ 7551P vCPU with 64GB RAM, 30GB of which were used. Using the same vCPU specifications, parallel batch scripts were executed for 7 minutes robotic exploration, which utilized (Turtlebot3, Gazebo, Gmapping, Explore Lite [29], and Map Server for map saving [30]. The total run time of generating SLAM maps was approximately 17 days.

Due to the constraint random movements applied in the map exploration stage, the generated SLAM maps do not all

have the same robot trajectory. That is why when training the network to generate a SLAM map, it is actually generating 'only one' possible hypothesis of a SLAM map. Keeping this in mind, the model would be learning the effect of the map structure on the SLAM map errors.

To achieve uniformity, the generated SLAM and CAD map images were processed by squaring, resizing, and ensuring consistent pixel value scheme and edge thickness. In addition, maps in which the robot got stuck were removed. This was done in 2 steps: (1) generated maps were automatically deleted if they 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; and (2) a manual cleaning was performed by visually assessing the maps.

Further processing to remove partially explored maps after training results showed that the generated maps did not corresponding to the input maps. To expedite the cleaning, a small set of 1,468 instances were cleaned manually and 15 specific features of these images were tabulated including pixel intensity (black, white, grey) count and ratios, number of holes using Jacob Eliosoff's stopping criteria [31], and grey to white neighboring pixel count.

The Extra Trees classifier [32] was trained and the output with probability of being complete in the range of 0.4 to 0.8 was cleaned manually (3,905 instances). This process reduced the dataset to 4,225 image pairs. Finally, the 2D CAD map images were cropped to have less grey areas, which makes it easier to visually asses them. 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 for training in the B to A direction, and 9 test pairs for training in the A to B direction.

### B. Training

All training is performed using the Adam optimizer [33] 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 trained on generating B from A and A from B. The B to A direction is done to correct SLAM map errors; whereas the A to B direction is done to model 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. To further emphasize the global features, the occupied edges of the target images were thickened.

## V. RESULTS AND DISCUSSIONS

The results of this work include the output of SLAM-to-CAD maps using $Model_{BA}$. In addition, the results of generating SLAM maps using both $Model_{AB}$ and parametric modelling are presented.

### A. SLAM-to-CAD Mapping

To asses the output, the model losses are first analyzed. As shown in Fig. 4, 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.

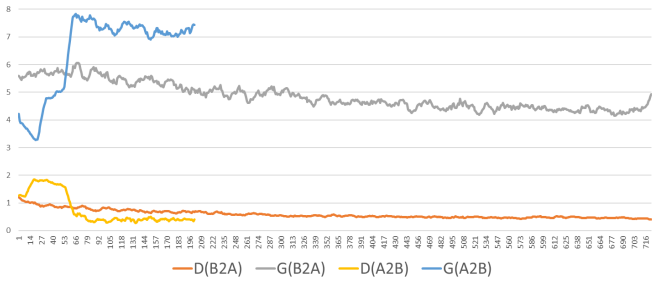



Fig. 4. Generator G and discriminator D losses for $Model_{AB}$ and $Model_{BA}$

The results for $Model_{BA}$ are shown in Fig. 5. These results are tested on the model trained until epoch 534, which was chosen visually and graphically from the start of the plateau in Fig. 4.

The mean square error of overlaid image intensities was used to evaluate the generated CAD maps. The alignment of the images was done using rotation, translation and scale only and then the MSE was tabulated in Table I.

TABLE I
THE MSE RESULTS OF THE GENERATED CAP MAPS.
$MSE_1$=MSE(GENERATED CAD MAPS, TARGET CAD MAPS) AND
$MSE_2$=MSE(SIMULATED SLAM MAPS, TARGET CAD MAPS)

| Map | $MSE_2$ | $MSE_1$ | $MSE_1 < MSE_2$ | Map | $MSE_2$ | $MSE_1$ | $MSE_1 < MSE_2$ |
|---|---|---|---|---|---|---|---|
| 1 | 0.092 | 0.031 | 1 | 12 | 0.038 | 0.059 | 0 |
| 2 | 0.032 | 0.064 | 0 | 13 | 0.153 | 0.000 | 1 |
| 3 | 0.079 | 0.051 | 1 | 14 | 0.339 | 0.319 | 1 |
| 4 | 0.250 | 0.089 | 1 | 15 | 0.116 | 0.127 | 0 |
| 5 | 0.122 | 0.047 | 1 | 16 | 0.028 | 0.110 | 0 |
| 6 | 0.107 | 0.221 | 0 | 17 | 0.058 | 0.039 | 1 |
| 7 | 0.104 | 0.022 | 1 | 18 | 0.092 | 0.048 | 1 |
| 8 | 0.058 | 0.112 | 0 | 19 | 0.058 | 0.081 | 0 |
| 9 | 0.225 | 0.120 | 1 | 20 | 0.151 | 0.054 | 1 |
| 10 | 0.117 | 0.073 | 1 | 21 | 0.094 | 0.091 | 1 |
| 11 | 0.086 | 0.060 | 1 | - | - | - | - |

On the one hand, in the fourth and eighth column of Table I the ones indicate that the structure and area of the generated CAD map is closer to the target CAD map than the simulated SLAM map is. On the other hand, the zeros indicate the opposite. From this table, Map 15 of Fig. 5 shows that it was not improved; however, through visual assessment it is clear that Map 15 was indeed improved. This sometimes happens due to the reliance of the evaluation method on pixel intensity values, which are limited features in the occupancy grid maps where there are repetitive structures. Thus, 70% of the simulated SLAM maps were improved by $Model_{BA}$. The

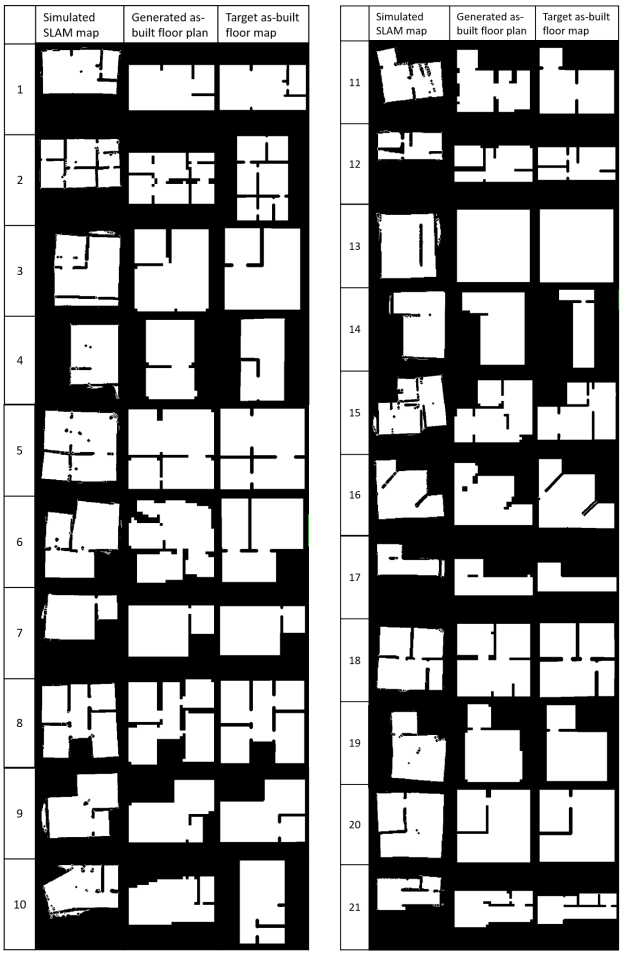



Fig. 5. Test results of $Model_{BA}$

remaining 30% could be due to the contraction and expansion of rooms that is usually caused by SLAM errors. Thus, the model did not perfectly learn this shrinkage and expansion. This can be due to the small size of the data or the use of GANs instead of a different deep learning architecture.

Further analysis was made by plotting a box plot shown in Fig. 6. The interquartile range along with the median falls almost completely in the positive region with positive values ranging from 0 to 0.080, while the negative quartile only reaches -0.022. This shows that the $Model_{BA}$ is far more likely to improve the simulated SLAM maps. This model would improve localization since in SLAM the accuracy of the localization depends on that of the map. Any remaining errors can be corrected with loop closure.

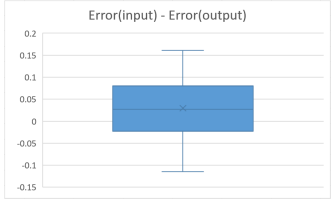



Fig. 6. The box plot visualization of the errors in Table I

However, not all aspects of the generated CAD maps can be assessed quantitatively. 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 has been removed (*e.g.* Map 19 in Fig. 5). In addition, the model learned to remove additional walls and improve wall positions (*e.g.* Map 11 and 10 of Fig. 5).

### *B. CAD-to-SLAM Mapping*

As for generating SLAM maps from CAD maps, we present the results of the learning and parametric based approaches.

*1) Machine Learning:* due to the fact that $Model_{AB}$ follows a progressive resolution change, the epoch and batch size change according to the current resolution, which explains the jumps at the epochs where resolution changes in Fig. 4. As a result of these jumps, the losses did not converge to low values as compared to the losses for $Model_{BA}$.

The test results of $Model_{AB}$ are shown in Fig. 7. Results of Epoch 155 were chosen because they gave the best results visually and graphically where the plateau of the trend graph starts (Fig. 4). 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.

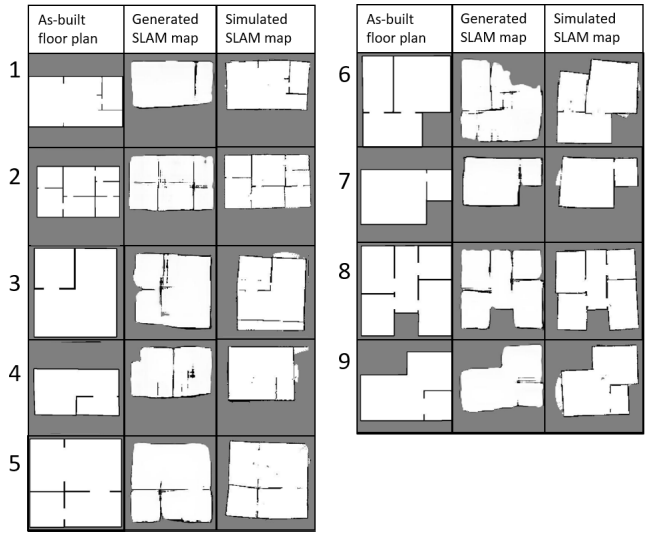



Fig. 7. Test results of $Model_{AB}$

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. 7. 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. 7. 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 common SLAM errors is curving of straight lines mainly due to angular error and

curvature in linear motion. Such a case is modelled very clearly in the results (e.g. Map 5 of Fig. 7). 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, 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 be further learned. For one, the shape of the unbounded additional unoccupied pixels occasionally forms a cloudy shape instead of the simulated scattered geometric shapes (Map 6 of Fig. 7). 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.

*2) Parametric Modelling:* To assess the parametric model, it is implemented with the two odometry-to-map error conversion methods (EKF and [26]). When running on high resolution images, EKF was very time consuming and thus low resolution images were used. Bilinear interpolation was then performed to upsize the output images to (256x256).

As for the [26] method implemented on (256x256) resolution images, time was not an issue, the run time was in seconds even for very high resolutions (e.g. 1600x1600). The output of the system with the various conversion methods is shown in Fig. 8 with the corresponding time of each run shown in Table II.

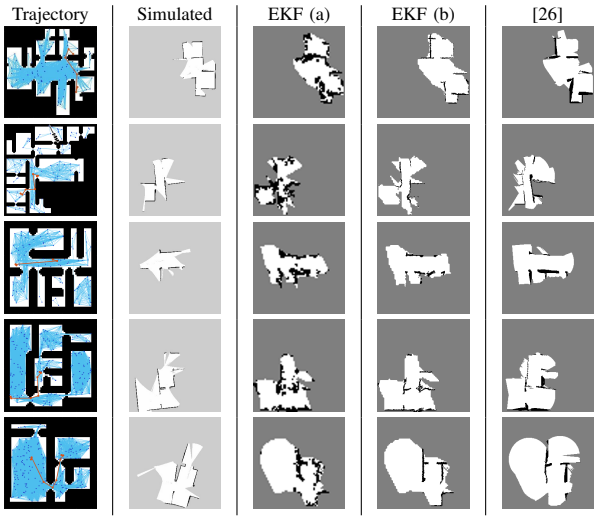



Fig. 8. Comparing odometry-to-map conversion methods using partial mapping. The columns represent (1) the trajectory, (2) the simulated SLAM map (3) the EKF method using (64x64) resolution, (4) the EKF method using (64x64) resized to (256x256), and (5) the [26] method using (256x256) images

On the one hand, the EKF method has a more detailed parametric model, where most importantly the error of re-observed landmarks is not a mere addition as is done in [26]; it relies on a complex covariance matrix that includes all landmarks and robot relations. However, this makes the method time consuming and in need of higher memory. On the other hand, [26] is: (1) faster and (2) can handle high resolutions. Thus, it avoided the bilinear interpolation for resizing, which can cause loss of data. Finally, the results in Fig. 8 show that the deformations are similar in both methods. Therefore, for the comparison of Parametric modelling with the Machine learning method, the conversion of [26] will be used.

TABLE II

TIME TAKEN IN SECONDS BY THE PARAMETRIC MODELLING WITH DIFFERENT CONVERSION OF ODOMETRY TO MAP ERROR METHODS

| Map ID | EKF (a) | EKF(b) | [26] |
|---|---|---|---|
| 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 results of the parametric modelling are shown in Fig. 9, which also includes the corresponding simulated SLAM maps with the specific 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 9 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 $Model_{AB}$.

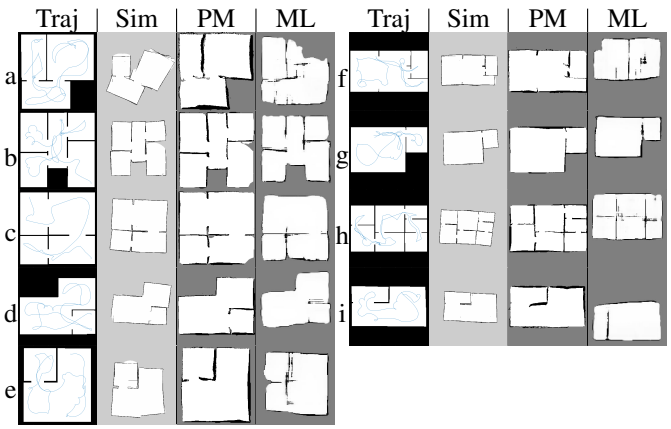



Fig. 9. Comparing Machine Learning to Parametric modelling approaches using mapping of the complete floor. The columns represent (1) the trajectory A, (2) the simulated SLAM map using trajectory A, (3) the parametrically modelled SLAM map using trajectory A, and the machine Learning generated SLAM map, which is not trajectory specific

From the results of Fig. 9, 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. 9. Whereas the parametric approach doesn't remove any structural elements; for example, doors in the parametric approach are better preserved. This can be an advantage of the parametric approach, since even though wall removal is present in SLAM errors, doors are mostly preserved.

In addition, from the results we can deduce that the

parametric approach does not model the simulated SLAM maps' sudden brokenness and large room shifts as shown in Instance (a) and (e) in Fig. 9. These large errors usually occur mainly due to vast spaces in the environment where there are no special features and the data association fails. Thus, it makes sense to observe better results with $Model_{AB}$ when it comes to these large errors because feature association is mainly an environment specific aspect, which is the domain on which $Model_{AB}$ was specifically trained.

## VI. CONCLUSIONS

Robotic maps are generated and used by SLAM. However, these maps are of low quality from a semantic or geometric point of view. In this paper, we were able to automatically generate CAD maps from SLAM maps using GANs. The inverse problem was also performed to create an alternative to robot map simulations and generate SLAM maps automatically. This was achieved using either a learning- or parametric-based modelling approach. The parametric modelling proved more readily usable due to its ability to generate multiple possible SLAM maps and deformation fields; however, the learning approach is a promising start for new research in the field due to its ability to learn features not modelled parametrically. Future work may include introducing different architectures with larger datasets and additional features.

## REFERENCES


[1] M. A. Hossain and M. M. Ali, "Recognition of handwritten digit using convolutional neural network (cnn)," *Global Journal of Computer Science and Technology*, 2019.

[2] 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*, 2019, pp. 4855–4864.

[3] 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)*. IEEE, 2018, pp. 1–8.

[4] 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.

[5] 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.

[6] 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.

[7] 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. IOP Publishing, 2016, p. 012030.

[8] 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)*. IEEE, 2018, pp. 1–8.

[9] 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.

[10] 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.

[11] 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.

[12] 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*. IEEE, 2006, pp. 2073–2078.

[13] 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.

[14] 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*. IEEE, 2008, pp. 000 897–000 900.

[15] G. Goronzy and H. Hellbrueck, "Weighted online calibration for odometry of mobile robots," in *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 2017, pp. 1036–1042.

[16] 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.

[17] 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.

[18] 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*. IEEE, 2012, pp. 4533–4538.

[19] 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)*. IEEE, 2017, pp. 1–6.

[20] 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*, 2014, pp. 2672–2680.

[21] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," *arXiv preprint arXiv:1802.05957*, 2018.

[22] 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.

[23] S. Thrun, "Probabilistic robotics," *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.

[24] L. Zhang, "Self-adaptive markov localization for single-robot and multi-robot systems," Ph.D. dissertation, 2010.

[25] 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.

[26] 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.

[27] 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.

[28] T. Hearn, "Stl tools," https://github.com/thearn/stl_tools, 2013.

[29] J. Horner, "Explore lite," http://wiki.ros.org/explore_lite, 2010.

[30] B. Gerkey and T. Pratkanis, "Map server," http://wiki.ros.org/map_server, 2009.

[31] 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.

[32] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine learning*, vol. 63, no. 1, pp. 3–42, 2006.

[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.