FastSLAM on a TurtleBot3 Waffle Pi for Indoor Environments

Diogo Antunes *93042* | José Coelho *90124* | José Pedro *93109* | Pedro Taborda *93152*

I. Introduction and Motivation

One of the big challenges that robots face in becoming autonomous and ubiquitous is being able to comprehend their surroundings and being able to localize themselves in that environment. In the autonomous robotics literature this problem is named SLAM, Simultaneous Localization and Mapping, and is an active research topic. Formalizing the objective of solving SLAM, the robot should be able to determine its trajectory as well as create a map, which is composed of a set of static landmarks.

A popular approach to solving SLAM is the EKF-SLAM, based on Kalman Filtering. However, this approach has issues, namely the assumption of gaussianity and the quadratic computational cost on the number of landmarks [1].

FastSLAM helps solve both of these problems. To deal with the assumption of gaussianity, FastSLAM replaces the EKF on the robot pose with a particle filter. Then, due to a result on all landmarks being conditionally independent on each other, given the robot pose, each particle has one EKF per landmark. This reduces the complexity from quadratic to linear in the computations, although the total complexity is $\mathcal{O}(NlogN)$ due to the particle resampling [2]. These improvements mean that the FastSLAM algorithm allows for maps with more landmarks, which in turn improves localization results.

In this report, the algorithm is briefly explained and our implementation is documented, with results being shown.

II. Methods and Algorithms

The SLAM problem consists of determining the probability distribution function

$$p(s^t, \Theta | u^t, z^t, n^t), \tag{1}$$

where the superscript t denotes the sequence from 0 up to t and s is the robot pose, Θ is the map, which is the set of all landmarks, θ_n, u are the robot's controls, z are the measurements and n is the match between the observed landmark and the landmarks in Θ. Applying Bayes' rule, we can separate this estimation problem into two separate problems,

$$p(s^t, \Theta | u^t, z^t, n^t) = p(s^t | u^t, z^t, n^t) p(\Theta | s^t, u^t, z^t, n^t). \tag{2}$$

Succinctly, this means that first the pose is estimated from the robot's controls and measurements and then, knowing the pose, the map is estimated. This problem can be further subdivided due to the conditional independence between landmarks given the robot pose, which can be seen in the graphical model of the SLAM problem shown in Figure 1.

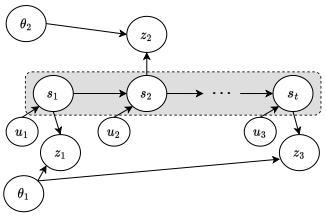


Fig. 1: SLAM Graphical Model.

Observing the graphical model, it can be concluded that if the pose is known, then the landmarks are independent. This means that the estimation can be further factorized into

$$p(s^t, \Theta | u^t, z^t, n^t) = p(s^t | u^t, z^t, n^t) \prod_{n=1}^{N} p(\theta_n | s^t, u^t, z^t, n^t). \tag{3}$$

Based on this observation, the FastSLAM algorithm uses a particle filter for estimating the posterior over the robot path. Each particle, assuming to have the correct path, has N EKF instances that estimate the position of the N landmarks.

A. Particle Filter

The particle filter contains, at each time instant t, a set of M particles which represent the robot pose's posterior, $p(s^t | z^t, u^t, n^t)$. Thus, each particle in the set, $s^{t,[m]} \in S_t$, is an estimate of the robot's trajectory,

$$S_t = \{s^{t,[m]}\}_m = \{s_1^{[m]}, ..., s_t^{[m]}\}_m. \tag{4}$$

Every time there is a new control action and measurement, the set S_t is built from the set S_{t-1}, u_t and z_t. Since the particle filter is meant to represent the robot pose's posterior, it would be desirable to sample from that distribution to draw new particles. However, given the difficulty in doing so, new particles are drawn from the probabilistic motion model which, under the Markov assumption, is $p(s_t | s_{t-1}^{[m]}, u_t)$. Since it is assumed that the particles in S_{t-1} are distributed according to $p(s^{t-1} | z^{t-1}, u^{t-1}, n^{t-1})$, the new particles are distributed according to $p(s^t | z^{t-1}, u^t, n^{t-1})$, which is the proposal distribution.

Since the proposal distribution, which the new particles represent, isn't the target distribution the distribution the filter is meant to represent, importance weights have to be

determined in order to correct this difference. The calculation of these weights is discussed further ahead.

These weights allow the particle set to be resampled. Resampling works by re-drawing particles from S_t according to their weights, with replacement. This means that particles with higher weights end up repeated and particles with lower weights disappear from the set. This concentrates particles on the higher probability regions of the target distribution, instead of the proposal distribution. However, resampling also reduces the filter's memory, since it reduces particle diversity.

B. Map Estimation

Since the factors due to the map in (3) are conditioned to the knowledge of the robot's pose, each particle has it's own map, $\Theta^{[m]}$. The estimation of each of these factors is done using an EKF for each landmark, which estimates the posterior $p(\theta_n^{[m]}|s^{t,[m]}, u^t, z^t, n^t)$. This means that for a particle filter with M particles, if each map has N landmarks then there are MN EKFs. The state model of the landmarks is described by

$$\begin{aligned} \theta_{n,t} &= \theta_{n,t-1} \\ z_{n,t} &= h(\theta_{n,t}, w_t) \end{aligned} \tag{5}$$

where h is the perceptual perceptual model and w_t is the measurement noise.

At each time step there is an association, n_t, which corresponds the current measurement, z_t, with a landmark. For all landmarks which are not being observed, $\theta_i, i \neq n_t$, there is no update

$$p(\theta_i|s^t, z^t, u^t, n^t) = p(\theta_i|s^{t-1}, z^{t-1}, u^{t-1}, n^{t-1}). \tag{6}$$

For the observed landmark, applying Bayes rule and using the Markov assumption,

$$\begin{aligned} &p(\theta_i|s^t, z^t, u^t, n^t) \\ &\quad \propto p(z_t|\theta_i, s^t, z^{t-1}, u^t, n^t) p(\theta_i|s^t, z^{t-1}, u^t, n^t) \\ &\quad = p(z_t|\theta_i, s_t, n_t) p(\theta_i|s^{t-1}, z^{t-1}, u^{t-1}, n^{t-1}). \end{aligned} \tag{7}$$

This update is done using the EKF, which linearises the probabilistic perceptual model, $p(z_t|s_t, \Theta, n_t)$.

In the cases where the data association is also dependent on the robot pose, then the associations are done per particle, $n_t^{[m]}$. This means that wrong associations might not significantly impact the filter's performance, if some particles make the correct association. In this case, the particles that made the wrong association will eventually have low weights and will be deleted from the filter through resampling.

C. LIDAR Feature Extraction

For a landmark based map, the LIDAR data isn't well suited to be used directly, since it complicates the data association problem given that each point is in most cases virtually indistinguishable from its neighbours. Thus, it is desirable to be able to extract features which are more unique from the data, like the walls reprented by clusters of points.

The features considered were lines. In indoor environments, straight lines are a useful landmark since walls are typically straight. For algorithm simplicity, the features are infinite lines, not line segments. Thus, two line segments along the same line, like two walls which are separated by an open door, are the same landmark. This presents some drawbacks, since the robot cannot distinguish the separate segments along the same line. In practice, this means that a robot in a corridor cannot correct its odometry estimates along the corridor's direction.

III. Implementation Details

The SLAM package developed was written in Python. Since the TurtleBot runs ROS, the data collection is done in ROS, recording the experiments in Rosbags. The topics which are extracted from the Rosbags are `/odom`, for the odometry, `/raspicam_node/image/compressd` and `/raspicam_node/camera_info` for the camera images and calibration parameters, and `/scan`, for the LIDAR scan data. To read the Rosbags in Python, the `rosbags` package is used.

A. Landmark Detection

The algorithm developed allows the use of two types of landmarks: ArUco markers and the previously mentioned straight lines. The sensor used for the detection of each of these landmarks is different. The ArUco markers are detected using a single, monofocal, RGB camera. The straight lines are identified from the LIDAR data.

1) ArUco Detection: The ArUco markers are robust, being reliably distinguished from each other even in images with multiple markers, and computationally inexpensive to detect from a 30fps video, as the one obtained from the TurtleBot. Since their physical dimensions are known beforehand, given that its the user who populates the environment with the markers, and their corners are reliably detected from the image, their position relative to the camera can be determined, even from just a monofocal camera. This detection is done using the `cv2` package, which produces the marker pose relative to the camera from the image and camera calibration data. ArUco detection using the TurtleBot's camera was only reliable at distances smaller than $2\,\mathrm{m}$.

2) Straight Line Detection: Each LIDAR scan is a point cloud of measurements from which straight lines are extracted. This is done by using the following algorithm:

Algorithm 1 Line extraction from LIDAR data.

Input: LIDAR data, $data$
Output: Set of lines, L

$L \leftarrow \{\}$
success $\leftarrow$ True
while size($data$) $> min_p$ **and** success $=$ True **do**
  $line$, success $\leftarrow$ RANSAC($data$)
  Remove $line$ inliers from $data$
  $L \leftarrow L + line$
end while

min_p is the minimum number of inliers for the algorithm to consider the line as a true line and RANSAC is a well

established algorithm to fitting models to data with outliers [3]. The lines identified are described by the angle of their normal direction in the robot's frame, ϕ, and the distance along that normal to the intersection with the line, r. This is exemplified in Figure 2, where both the measurement in the robot frame and in the world frame are shown for the line L.

B. Action Model

The robot's dynamics are approximated by an odometry model affected by Gaussian multiplicative noise in distance travelled and change of orientation. The odometry parameters are the change in s given in the robot's frame, called Δs_r, and the change in θ, called $\Delta\theta$. The change in s given in the world frame, called Δs, is obtained by rotating Δs_r by θ. The Gaussian noise in the distance traveled is called v_s and has s.t.d. σ_r. The Gaussian noise in θ is called v_θ and has s.t.d. σ_θ. With these definitions, the odometry model is defined algebraically by

$$\begin{cases} s_{t+1} = s_t + \Delta s(1 + v_s) \\ \theta_{t+1} = \theta_t + \Delta\theta(1 + v_\theta) \end{cases}. \tag{8}$$

This noise model was chosen under the consideration that the odometry data has relative noise, not absolute, which justifies the multiplicative nature of the Gaussian noise model.

C. Perceptual Model

Since there are two diferent types of landmarks: ArUco markers, θ_n^A, and lines, θ_n^L, there are two different perceptual models, one for each. In Figure 2 one example measurement of each landmark can be seen, as well as the landmarks world coordinates.

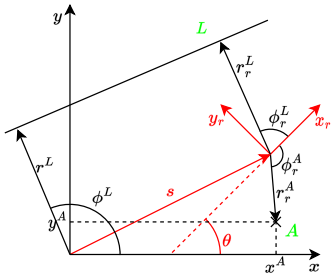


Fig. 2: Diagram of the robot measurements and the landmarks. The black frame represents the world frame and the red one represents the robot frame. The s red vector and the angle θ are the translation vector and the rotation angle from world frame to robot frame. An ArUco A at world coordinates $[x^A, y^A]^T$ is perceived by the robot by a measurement in polar coordinates $[r_r^A, \phi_r^A]$. A lidar line L $[r^L, \phi^L]$ in world frame is perceived by the robot at $[r_r^L, \phi_r^L]$.

Both the camera and the LIDAR data are on a different frame than the robot, although they have the same orientation. Since this means a simple reference frame change has to be done, it is omitted from the perceptual models here to simplify the derivations, although it is taken into account by the algorithm.

1) ArUco Markers: The measurement of the marker's position relative to the robot is given in polar coordinates. Given the landmark's world coordinates, $\theta_n^A = [x^A, y^A]^T$, and the robot pose, $s = [s_x, s_y, \theta]$, the measurement is given by

$$\begin{aligned} r_r^A &= \sqrt{(x^A - s_x)^2 + (y^A - s_y)^2} \\ \phi_r^A &= \operatorname{atan2}(y^A - s_y, x^A - s_x) - \theta. \end{aligned} \tag{9}$$

Assuming that the Gaussian measurement noise, w_t, is linear with the measurement obtained, the perceptual model h is given by

$$h^A(\theta_n^A, w_t, s_t) = \begin{bmatrix} r_r^A \\ \phi_r^A \end{bmatrix} + w_t. \tag{10}$$

2) LIDAR Lines: After the line extraction from the LIDAR data, the line measurements are given in (r_r, ϕ_r), which represent the intersection with the line along its normal, as seen in Figure 2. Since the line landmarks are represented by the intersection along the normal from the world frame origin, the measurement is given by

$$\begin{aligned} r_r^L &= \begin{bmatrix} \cos\phi_r^L & \sin\phi_r^L \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} r^L\cos\phi^L - s_x \\ r^L\sin\phi^L - s_y \end{bmatrix} \\ \phi_r^L &= \phi^L - \theta. \end{aligned} \tag{11}$$

Since RANSAC is an algorithm, it is not simple to determine how LIDAR sensor noise would be mapped to line noise. Thus, a simple noise model for the lines was assumed, with the Gaussian noise w_t being linearly added to the measurement. Thus, the perceptual model for the lines is simply

$$h^L(\theta_n^L, w_t, s_t) = \begin{bmatrix} r_r^L \\ \phi_r^L \end{bmatrix} + w_t. \tag{12}$$

D. Landmark EKF

To estimate each landmark's position from the successive measurements an EKF is used. The classical Kalman filter cannot be used in this case due to the non-linear perceptual models for both kinds of landmarks. Since the landmarks are static there is no predict step, only the update step due to the measurements. Let $\boldsymbol{H}_{\theta,t}$ be the Jacobian of the perceptual model, h, and defining the sensor uncertainty covariance matrices as

$$\boldsymbol{R} = \begin{bmatrix} \sigma_r & 0 \\ 0 & \sigma_\phi \end{bmatrix}^T \begin{bmatrix} \sigma_r & 0 \\ 0 & \sigma_\phi \end{bmatrix}, \tag{13}$$

where σ_r and σ_ϕ are the standard deviations of the uncertainties in the sensors distance and angle measurements, respectively. The EKF updates each landmark's expected value, $\mu_{n,t}$, and covariance, $\boldsymbol{\Sigma}_{n,t}$, through

$$\begin{aligned} \boldsymbol{S}_{n,t+1} &= \boldsymbol{H}_{\theta,t+1}\boldsymbol{\Sigma}_{n,t}\boldsymbol{H}_{\theta,t+1}^T + R \\ \boldsymbol{K}_{n,t+1} &= \boldsymbol{\Sigma}_{n,t+1}\boldsymbol{H}_{\theta,t}^T\boldsymbol{S}_{n,t+1}^{-1} \\ \mu_{n,t+1} &= \mu_{n,t} + \boldsymbol{K}_{n,t+1}(z_{n,t+1} - h(\mu_{n,t}, s_{t+1})) \\ \boldsymbol{\Sigma}_{n,t+1} &= (\boldsymbol{I} - \boldsymbol{K}_{n,t+1}\boldsymbol{H}_{\theta,t})\boldsymbol{\Sigma}_{n,t}. \end{aligned} \tag{14}$$

$\boldsymbol{S}_n$ and $\boldsymbol{K}_n$ are the innovation covariance and the Kalman gain, respectively.

E. Importance Weights

In order to correct for the fact that the particle filter samples from a distribution which is not the one it is meant to represent, importance weights are calculated through

$$w_t^{[m]} = \frac{\text{target distribution}}{\text{proposal distribution}} = \frac{p(s^t|z^t,u^t,n^t)}{p(s^t|z^{t-1},u^t,n^{t-1})}. \quad (15)$$

Applying Bayes' rule and the Markov assumption, it can be shown [2] that these importance weights are given by

$$w_t^{[m]} \propto \int p(z^t|\theta_n,s_t,n_t)p(\theta_n|s^{t-1},z^{t-1},u^{t-1},n^{t-1})d\theta_n. \quad (16)$$

The second term is the landmark estimate, which is a normal distribution estimated by the EKF. The first term is the observation likelihood which, due to the observation model non-linearity, is not a normal distribution. However, since the EKF already linearises the model to estimate the landmark location it is a consistent approximation to linearise the model for the likelihood as well, which yields a normal distribution. Since, under these approximations, the integral is the convolution of the p.d.f.s of two normal distributions, the weights come as

$$w_t^{[m]} \propto \exp(-\frac{1}{2}(z_t - h(\theta_n,s_t))^T \boldsymbol{Q}_t^{-1}(z_t - h(\theta_n,s_t)), \quad (17)$$

where Q_t is given by

$$\boldsymbol{Q} = \boldsymbol{R} + \boldsymbol{H}_{\theta,t}\boldsymbol{\Sigma}_{n,t-1}\boldsymbol{H}_{\theta,t}^T. \quad (18)$$

F. Actions, Measurements and Resampling

Although the algorithm is derived assuming that every action is accompanied by a measurement, and vice-versa, that needn't be the case. In the case of the TurtleBot, the odometry has a higher sampling rate, 25 Hz, than either the camera, 15 Hz, or the LIDAR, 5 Hz. The algorithm deals with this by treating each new odometry reading as a new action. This causes a new set of particles to be generated by sampling from the action model. However, since there isn't an observation, all the particles keep their previous weights. When an observation arrives, importance weights are determined for each particle. Resampling happens before sampling from the action model, generating a set of particles with weight 1 to which the probabilistic model can be applied. Since the minimum variance sampler is used, when all particles have weight 1, as happens in a period of no observations, all of them are kept.

IV. Results

The datasets which were collected to test the algorithm are all from the 5th floor of the North Tower at IST, which has a square corridor with a side length of 15.78 m and the corridor width is 1.70 m.

Given that the map produced by the algorithm is composed of lines and ArUco markers, the figures of merit extracted from the algorithm results are the corridor width and the corridor length. These allow quantitative analysis of the algorithm performance since these environment characteristics can be determined with high precision. Since, when using real datasets, the ground truth for the robot trajectory cannot be determined easily, this is done in simulation.

A. Experimental Setup

Two datasets were gathered, one where the robot makes a round trip around only a quarter of the hallway and another where a full loop around the corridor is done. The parameter space which is considered in the tuning of the algorithm is composed of the number of particles, N, the action model uncertainty expected value, μ_s and μ_θ, and standard deviation, σ_r and σ_θ, as well as the LIDAR line measurement uncertainty standard deviations, σ_{r^L} and σ_{ϕ^L}. The ArUco measurement uncertainty isn't considered since, as will be shown, the ArUco's do not improve the algorithm's performance and their utility was mostly as an initial landmark which was easy to detect, to validate the algorithm's functioning.

Given that the parameter space is 7-dimensional, the analysis of the effect of each parameter on the algorithm's performance has to be done independently, due to computational constraints. For this, sensible default values were defined for the parameters in the cases where they are not varying. The default number of particles chosen was $N = 50$, the action model uncertainty has $\mu_r = \mu_\theta = 1$, $\sigma_r = 0.1$ and $\sigma_\theta = 0.5$ and the LIDAR uncertainty is characterized by $\sigma_{r^L} = 0.1\,\text{m}$ and $\sigma_{\theta^L} = 10°$.

For every experiment, every parameter combination was tested 5 times. Then, the result was manually classified as acceptable or not, by analysing if the corridor was the same on both parts of the round trip and if the walls were parallel. If there were more unacceptable results than acceptable, then the given parameters would not be considered. In the plots shown, these areas are crossed off with red highlights to indicate that tests were ran and the results were not acceptable. If the results are adequate, then the average of the performance metrics is taken as the value for that combination.

B. Number of Particles

In Figure 3 the results of an experiment where the number of particles is varied between 10 and 200 is shown.

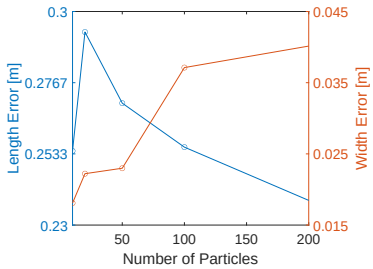


Fig. 3: Effect of the number of particles on the algorithm's performance.

Contrary to intuition, the increased number of particles doesn't improve the performance of the algorithm significantly. In the case of the corridor width, it appears to converge to a

value different from the manual measurement of the corridor's width. This indicates that there may have been an error in the measurement of the reference width.

Simulated Trajectory RMSE: A segment of the corridor in the 5th floor of the North Tower, IST was copied (using manual measurements) into a simulated environment. Several trials, consisting of running FastSLAM on the simulated robot 20 times for a certain number of particles N, were performed. For each trial and each simulation, the root mean squared error between the robot's simulated pose and its estimate (mean pose of all particles) is measured, and the mean along x, y and θ is computed, yielding a vector containing the RMSE of the run. These vectors are averaged over all the runs in a trial (varying the random number generator's initialization seed), yielding a mean RMSE for a given number of particles. In the simulation, the robot traverses the aforementioned corridor, the XY coordinate system's x direction is orthogonal to the corridor's walls, and the y direction parallel to them.

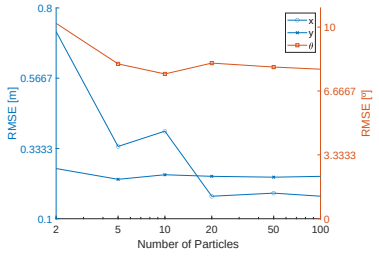


Fig. 4: Effect of the number of particle's on the error between the estimated trajectory and the simulated trajectory.

As it is possible to observe, as the number of particles increases, the error along the x direction decreases. However, the error in the y direction does not follow the same trend. This is probably due to the corridor being along this direction, so the particles have no way to discern position along the corridor.

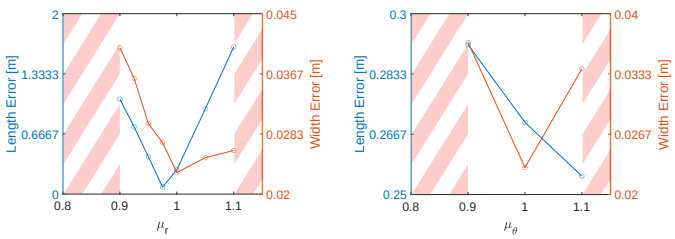
The error on the orientation (θ) also decreases, but at a lower rate. This may be due to the lack of landmarks which heavily rely on orientation.

C. Odometry Bias

Although the odometry bias isn't being estimated online, tuning the expected value of the odometry noise can give insight into the existence of bias in these readings. In Figure 5a and 5b, the effect of these parameters is shown.

For the bias in θ, the results indicate that there is none, given that it does not affect the error in the corridor length determined and the error in the corridor's width has a minimum for $\mu_\theta = 1$. However, since the resolution of the data is low, it cannot be accurately concluded whether the true minimum is actually 1 or some value near it.

For the bias in r however, the data clearly shows that the minimum in the corridor length error is associated with $\mu_r = 0.975$. This allows us to conclude that the odometry of the robot used, incorrectly overestimates the distance traveled and by correcting the odometry noise with a scale factor of less than 1, the results are improved.



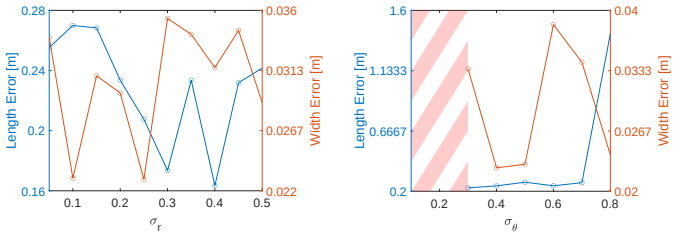
(a) Effect of μ_r on the algorithm's performance.

(b) Effect of μ_θ on the algorithm's performance.

Fig. 5: Effect of the odometry bias on performance.

D. Odometry Uncertainty

In Figure 6a and 6b, the effect of the odometry uncertainty on the algorithm is shown. Since the odometry uncertainty directly determines the action model distribution, it plays a key role in the algorithm performance since it determines how much the particles are spread when there is an action. Thus, the importance of determining adequate uncertainty parameters cannot be understated.



(a) Effect of σ_r on the algorithm's performance.

(b) Effect of σ_θ on the algorithm's performance.

Fig. 6: Effect of odometry uncertainty on performance.

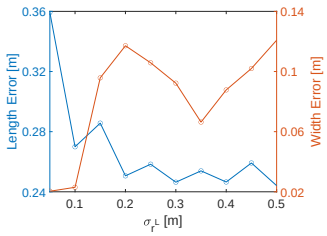
The results relative to σ_r suggest that, in the interval over which the parameter was varied, the uncertainty in the odometry distance measured has little effect on performance. This means that the interval of interest is probably shorter and closer to 0. However, the previous results indicate that the odometry suffers from biasing and taking into consideration that the default parameters do not compensate for it, it is important not to choose an uncertainty which is too low, since it might compromise the ability of the particle filter of generating particles in areas with high observation likelihood.

Observing the results on σ_θ, conclusions can be drawn on how to choose the adequate value. It can be seen that too high of an uncertainty causes severe degradation in the algorithm's performance, while for lower values the effect of the corridor length error seems minimal. In this interval, $\sigma_\theta = 0.4$ seems to strike the best compromise between minimizing corridor length and corridor width error.

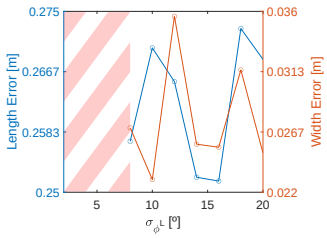
E. Sensor Uncertainty

Much like odometry uncertainty, the sensor uncertainty heavily impacts the quality of the results in two distinct

manners. Too less uncertainty and the chance of having a particle with relevant observation likelihood is too low to have adequate performance. Too high uncertainty and the ability of algorithm to distinguish between similar, yet clearly different, measurements is impaired. Thus, in Figure 7a and 7b results are shown for the variation of the sensor's uncertainties in order to determine where the optimal middle point lies.



(a) Effect of σ_{r_L} on the algorithm's performance.



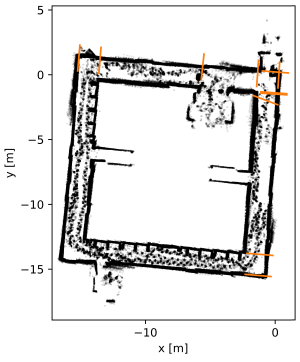
(b) Effect of σ_{ϕ_L} on the algorithm's performance.

Fig. 7: Effect of sensor uncertainty on performance.

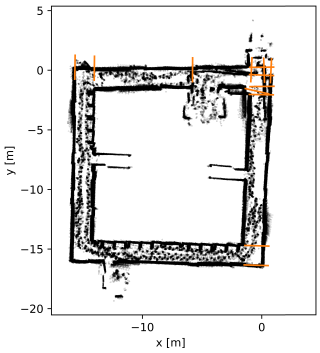
From the results it can be concluded that, due to the sensor characteristics and the algorithm used to extract lines from the data, there is a minimum amount of angle uncertainty required for the algorithm to function. From there, the results vary unpredictably with the change in uncertainty. For the length uncertainty, there is again a compromise between corridor length and corridor width error.

F. Tuned Algorithm

Choosing, based on the experiments ran, "optimal" parameters, with $N = 50$, $\mu_r = 0.975$, $\mu_\theta = 1.1$, $\sigma_r = 0.3$, $\sigma_\theta = 0.4$, $\sigma_{r^L} = 0.1\,\mathrm{m}$ and $\sigma_{\phi^L} = 10°$, the algorithm is ran on the dataset which loops around the corridor. These results are shown, along with the result obtained with default parameters, in Figure 8a and 8b.



(a) Better algorithm parameters.

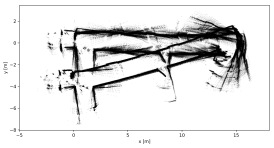


(b) Default algorithm parameters.

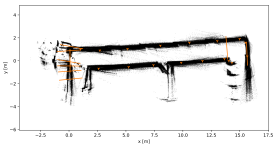
Fig. 8: Results on a data set with a loop. For visualisation convenience, the point cloud data is rendered by centering each LIDAR scan on the estimated pose at the scan instant. The points inside the corridor are the legs of curious bystanders following the robot during the test.

Besides the improvement of the map, which can be confirmed visually, since with the new parameters the loop can be adequately closed and previously couldn't, the corridor length error went from $0.64\,\mathrm{m}$ to $0.29\,\mathrm{m}$ and the width error from $0.06\,\mathrm{m}$ to $0.005\,\mathrm{m}$. The substantial error improvement in both the corridor length and width, coupled with the closing of the loop, provide validation of the determined parameters.

In Figure 9b and 9a show results using only the odometry data, and the odometry data aswell as ArUco and LIDAR landmarks. The results show that using ArUco landmarks do not have a great effect on the estimated map, which can be seen by the quality of the results in Figure 8a which use only the LIDAR.



(a) Results produced with only the odometry of the robot.



(b) Results produced with both LIDAR and ArUco landmarks.

Fig. 9: Results on a data set with the loop around the side of the corridor. For visualisation convenience, the point cloud data is rendered by centering each LIDAR scan on the estimated pose at the scan instant.

V. Conclusions

Given the quality of the results, since very good map estimates were produced, the algorithm is deemed to be effective. A map of the floor was produced, without any set up work like adding ArUco markers and despite the environment including moving people, which obstruct the LIDAR measurements. The choice of lines for landmarks provides the robustness to people.

Further improvements could be made to improve the results. The proposal distribution used in FastSLAM 2.0 [4] could be implemented, which takes the observation into account when sampling new particles, improving the algorithm performance with fewer particles. Using line segments as landmarks instead of using infinite straight lines would help with long corridors which have interruptions, like open doors, since it would add longitudinal information. An attempt could be made to estimate the odometry bias online, which would help correct wrong corridor lengths or the difficulty in closing loops. Although the bias was tuned, since it not estimated online it only works for the robot on which the tests were ran.

References

[1] Thrun, Sebastian, et al. "Fastslam: An efficient solution to the simultaneous localization and mapping problem with unknown data association." Journal of Machine Learning Research 4.3 (2004): 380-407.

[2] Montemerlo, Michael, et al. "FastSLAM: A factored solution to the simultaneous localization and mapping problem." Aaai/iaai 593598 (2002).

[3] Fischler, Martin A. and Bolles, Robert C. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography." Association for Computing Machinery 24 (1981): 381–395.

[4] Montemerlo, etal. ”FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges.” IJCAI 3 (2003): 1151-1156.