Intelligent Robotics: Simultaneous Localisation and Mapping, with Accompanying Exploration Algorithm

Matthew Broadway, Joe Faulls, Nishanth Ganatra, Ben Johnson, Michael Oultram and Charlie Street

I. Introduction

A robot can use its on-board sensors to generate a map of an unknown environment, while keeping track of its location within its internal representation of the world; this is known as Simultaneous Localisation and Mapping (SLAM). This report presents an implementation of SLAM written in C++ with an accompanying algorithm that utilises the map to explore autonomously with incomplete knowledge. The explore algorithm and SLAM collaborate to create a complete global map of the environment using an efficient route. This report also provides performance evaluations, based on experimental evidence, of both the individual components and the system as a whole. Different implementations of each component are compared to determine which configurations perform best overall. The robot used for testing was a Pioneer 3-DX with a Hokuyo URG-04LX laser sensor, providing 512 distance readings across $180°$, up to a distance of ≈ 5.75m.

Other than the Robot Operating System (ROS), this implementation uses minimal dependencies. External libraries were utilised where a component was outside of the project's scope, such as for efficient linear algebra (`Eigen`) and the ICP algorithm (`libpointmatcher`).

ROC analysis of the overall system shows that the highest performing SLAM configuration performs well with regards to finding 'free' and 'unknown' space. However, it performs poorly in classifying 'blocked' cells leading to highly sub-optimal global maps in certain situations.

The 'standard' method of local map generation performed better than the other two methods. In particular, the local maps generated by a neural network were unexpectedly poor.

Experimental evidence found that none of the stitching methods performed well in correcting rotational noise. However, with regards to translational noise, the 'Edge' and 'Gradient' methods performed significantly better corrections than 'Brute Force' stitching.

Several exploration methods were tested; when presented with an empty square map, the 'explore with hoover' algorithm explored the largest proportion of the environment in the given time. In a more realistic map, 'wall hugger' was found to explore the largest amount of the environment.

II. System Background

A SLAM system running on a mobile robot, as first popularised by Thrun (1998), works by first instructing the robot to move through the world, occasionally triggering the generation of 'local maps'. One possible form of local maps are occupancy grids: crude representations of obstacles immediately visible to the robot's sensors, with some amount of uncertainty associated with them due to imperfect sensing. In the case of this implementation, the local map is generated using a rotating laser sensor (LiDAR); other techniques exist for creating occupancy grid local maps using different sensors, such as sonar (Moravec and Elfes, 1985) or a depth camera for 3D occupancy volumes (Oliver *et al.*, 2012). Methods utilising non-grid based local maps are also possible, for example 'Visual SLAM' (Karlsson *et al.*, 2005).

Local maps tend to drift over time so that they no longer line up perfectly with older local maps, due to the accumulation of small errors in odometry. This is called the 'closed loop problem' (Newman and Ho, 2005) and to combat it, local maps are instead used to maintain a single 'global map', which is also an occupancy grid. The global map is updated by each local map in turn, resulting in an accurate and consistent record of the surrounding environment. The current local map can be lined up with the global map prior to updating; this is referred to as 'stitching'.

Crucially, this system can be run during exploration, allowing for simultaneous map construction and localisation within that map, hence the name: SLAM. The partially constructed map can be used to influence the planned route through the world to quickly explore the entire reachable area, rather than exploring without a map.

III. System Design

The robot is intelligently instructed to move through the world by an exploration algorithm (see section VII). When sufficient movement (0.0625m or $20°$) has occurred, the SLAM implementation proceeds to update its map. First, the most recent sensor data is recorded from the `base_scan` and `odom` ROS channels. The local map is generated from the laser data and then stitched onto the global map. The cells of the global map are then updated using the new data. A first estimate for the robot pose is calculated using the odometry data and is then refined using the results of the local map stitching. The new global map is published to guide the exploration algorithm.

IV. Local Map

A. Background

In order to generate a global map, the robot frequently gives a description of the current state of the environment visible to its sensors. The local map covers a small area directly in front of the robot, with each cell containing an estimated probability of being occupied given a set of sensor readings: $\mathrm{P}(\textit{cell is occupied} \mid \textit{sensor readings})$. Calculating this probability is the act of building an 'inverse sensor model'. The prior probability (output of the inverse model for cells which are 'unknown'), is $\mathrm{P}(\textit{cell is occupied}) = 0.5$.

This report explores three algorithms for local map generation. Two methods are derived from *Probabilistic Robotics* (Thrun, Burgard and Fox, 2005, pp. 221-244): a hard-coded 'standard' algorithm and a method to learn the model using an Artificial Neural Network. The third is a novel, non-deterministic method which utilises Axis-Aligned Boundary Boxes (AABB).

1) Standard: The 'standard' method for calculating the inverse sensor model, as described by Thrun, Burgard and Fox (2005, p. 230), considers the distance reading closest to each cell. A test is performed for every cell to determine whether the cell centre is within $\pm\alpha/2$ of the closest laser reading, where α controls the wall thickness. Within this range, the cell is considered 'blocked'. If the cell is closer to the robot than the laser reading, then the cell is considered 'free'. All other cells are considered 'unknown'. Fixed values are used for the probability of each class. This is a crude approximation which does not model the uncertainty of the real-world sensor. However, the noise of modern sensors is minimal such that the approximation is sufficient.

2) Neural Network: An effective inverse sensor model is difficult to determine algorithmically. It can be derived by applying Bayes' rule to an existing forward sensor model, frequently used for localisation. However, Ellore (2002, p. 36) found that maps generated by a neural network 'are of better quality than the maps produced by the [forward] sensor model'. Furthermore, Van-Dam, Kröse and Groen (1996, p. 6) discovered that 'Gaussian shaped sensor models are rather inaccurate' and so proposed the use of neural networks for learning the inverse sensor model.

This method involves training a neural network to mimic the inverse sensor model. The network outputs the probability of a single cell being occupied, given its nearby sensor data and location. Two appropriate network topologies for this task are Multi-Layer perceptrons (Thrun, 1998) and Radial Basis Function Networks (Van-Dam, Kröse and Groen, 1996).

Large quantities of training data are required to train a neural network for this task. Sense-and-Drive as described in Van-Dam, Kröse and Groen (1996, p. 9) can be used to collect such data. The training samples are obtained by taking a sensor reading, followed by crashing the robot into an obstacle to determine the 'occupied' 'ground truth'. Each traversed cell corresponds to a training sample consisting of the input data to the neural network and a boolean value for a cell's occupancy.

The inverse sensor model must produce probabilities rather than the discrete boolean values found in the training set. It was shown by Mitchell (1997) that a neural network can learn the maximum likelihood hypothesis for $f'(x) = \mathrm{P}(f(x) = 1)$, given a function $f : X \to \{0, 1\}$ described empirically by the training data.

3) AABB: Axis-Aligned bounding boxes (AABBs) are commonly used in computer vision and video games to perform simple intersection tests, such as the point-rectangle intersections required here. This method considers each laser reading once in turn, whereas the others described consider each cell once. For each laser reading, the single cell intersecting with the point at the end of the reading must be determined. This is done using an AABB test with every cell until an intersection is found. The intersecting cell is considered 'blocked' and the cells between the 'blocked' cell and the robot are considered 'free', again using AABB tests to determine which cells lie in-between. All other cells are considered 'unknown'.

B. Design

Some design choices were made independent of the inverse sensor algorithms. The robot is placed at the middle-bottom of the local map. The chosen cell resolution $\mathrm{res} = 0.25\text{m/cell}$ and the laser range $\mathrm{maxr} = 5.75\text{m}$, combine to give the local map dimensions of:
$\text{width} = \left\lfloor 2 \times \frac{\mathrm{maxr}}{\mathrm{res}} \right\rfloor = 46 \;\&\; \text{height} = \left\lfloor \frac{\mathrm{maxr}}{\mathrm{res}} \right\rfloor = 23$

The value -1 is used to specify 'padding'. Local map cells that need to be excluded from the update are given this value. These cells are either outside the sensor's range or part of a border used when rotating the map.

1) Standard: This method follows the approach by Thrun, Burgard and Fox (2005, p. 230), but with some modifications. If any of the four closest laser readings to the cell's angle fall within $\pm\alpha/2$ (with $\alpha = 0.4\text{m}$), then the cell is considered 'blocked'. This approach prioritises false positives over false negatives. The original method used a single laser reading per cell. The two closest readings are used to determine whether a cell is behind a wall. The β parameter (beam width) is not required due to the chosen shape of the local map used with this method. Additionally, padding is not used in the original method as the prior (0.5) is used instead.

2) Neural Network:

Algorithm 1 Sense-and-Drive

```
1:  while samples collected $< 10{,}000$ do
2:      Take laser reading
3:      Rotate randomly through $-\frac{\pi}{2} \le \theta \le \frac{\pi}{2}$
4:      Move forward until a 'collision' occurs
5:      Calculate which cells were traversed
6:      for each traversed cell $c$ do
7:          if stopped_in($c$) then $c := 1$
8:          else $c := 0$
9:          Store a training sample for the cell $c$
10:     Move backwards by 0.5m
11:     Rotate randomly through $\theta \in \{0, 120°, -120°\}$
```

a) Sense-and-Drive: To obtain quality training data, high accuracy in translation and rotation is required. The robot's velocity varies over time based on several factors: a chosen initial velocity $v_0 = 0.2\text{m/s}$, the current smallest distance to *any* obstacle $d_t = \min(\text{Lasers}_t)$ and the initial distance reading in the direction of travel d_0. The velocity at time t is calculated by $v_t = v_0\, d_t/d_0$. If d_t drops below a chosen threshold (0.2m) then v_t is set to 0 to allow the robot

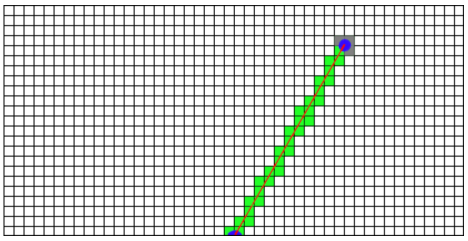

Fig. 1: A single iteration of Sense-and-Drive along a path (red) with traversed cells (green) and blocked cells (grey) highlighted.

to roll to a stop and avoid a collision. This allows the training data to be gathered with minimal human supervision.

During rotation, the angular velocity ω is initially set to 50% of the maximum until the turn has $< 20.1°$ remaining, at which point it switches to 10%. When the destination is within a threshold of $2.86°$, ω is set to 0. This allows the robot to ease to a halt, increasing the rotational accuracy.

To determine the traversed path, shown in Figure 1, the odometry records the start and end positions of the movement. The robot stops just before a collision since it was not equipped with a touch sensor. The cell at the end of the path is considered 'blocked', which gives a 'ground truth' value of $\mathrm{P}(\mathrm{occ}_{xy}) = 1$ for the training set. All other traversed cells (ones which intersect the path) are considered 'free'. These cells are given a ground truth of $\mathrm{P}(\mathrm{occ}_{xy}) = 0$ since the robot passed through them without colliding. Each training sample consists of: the 'ground truth' of the cell, the four sensor readings closest to the cell and the polar vector between the robot and the cell. After reaching an obstacle, the robot moves backwards and rotates away from the wall. This should give move variety to the distance readings in the training set.

b) ANN: The chosen network topology is a multi-layer perceptron with a single output neuron and one hidden layer. Cruz *et al.* (2002, p. 160) found that using a combination of several distance readings, along with the angle and distance to the cell, produced 'the smallest errors, with good adaptability and robustness'. Four readings were chosen based on the approach by Thrun (1998).

The chosen output activation function is the sigmoid/logistic function (σ). This is a clear choice since the network is tasked with a binary classification problem with a single output neuron. The chosen hidden activation function is tanh. This is a common choice, as it has been shown to enable gradient descent to learn faster (LeCun *et al.*, 2012, p. 19). The chosen error function is 'cross entropy loss'. The network was trained with online stochastic gradient descent with back-propagation.

The weight update equations to train this network using back propagation were derived:

$$\Delta w_{ij}^{(2)} = -\eta\, a_i^{(1)} \delta_1^{(2)}$$
$$\Delta w_{ij}^{(1)} = -\eta\, x_i^p \left(1 - (a_j^{(1)})^2\right) \delta_1^{(2)} w_{j1}^{(2)}$$

Where:

- $\delta_1^{(2)} = \sigma(s_j^{(2)}) - t^p$ where the superscript refers to the network layer, and the subscript, the neuron.
- $w_{ij}^{(l)}$ is an element of the weight matrix $\mathbf{w}^{(l)}$: the weight between layer $l-1$, neuron i and layer l, neuron j.
- $f^{(l)}(x)$ is the activation function for layer l.
- $a_i^{(l)} = f^{(l)}(s_i^{(l)})$ is the activation of layer l, neuron i.
- $s_j^{(l)} = \mathbf{w}_{*j}^{(l)} \cdot \mathbf{a}^{(l-1)}$ is the signal to layer l, neuron j.
- x_i^p is the i-th element of some input vector $\boldsymbol{x}^p$.
- t^p is the target label for input vector $\boldsymbol{x}^p$.
- η is the learning rate.

Other network and training parameters include: the learning rate, $\eta = 0.1$; the number of training epochs = 30,000 and 6 : 100 : 1 neurons in the input, hidden and output layers respectively.

3) AABB: In algorithm 2, a cell can be one of the following categories: Unsure, LikelyBlocked$[X]$, LikelyFree$[X]$ and Untouched. Where applicable, X denotes the likelihood value for the cell, which is converted to a probability at the end of the algorithm. This is done using the curves $y = e^{-kx}$ and $y = 1 - e^{-kx}$, where k influences the gradient (with $k = 0.25$).

Every cell is initialised to Untouched. Each laser reading is iteratively shrunk, 'dragging' it back towards the robot. The first five cells that the laser reading hits are considered LikelyBlocked with a decreasing probability. Subsequent cells are set to LikelyFree unless already LikelyBlocked.

C. Empirical Study

1) Neural Network Training and Optimisation:

a) Sense-and-Drive: To collect accurate training samples, execution of an arbitrary rotation followed by an arbitrary forward movement should finish in the desired location accurate to one cell (0.25m square). This requires accuracy in translation up to half the cell size and accuracy in rotation up to the cell size at a distance of 5.75m (the maximum sensor range), calculated by $\tan^{-1}\frac{0.25}{5.75} = 2.49°$. The rotation accuracy calculation assumes that the robot starts the rotation orthogonal to the grid and between two cells, which holds during Sense-and-Drive.

Translation accuracy was tested by instructing the robot to move a known distance and recording the discrepancy between the 'ground truth' and odometry. The translation accuracy was tested through a range of 1–5m. The absolute error results of the translational accuracy tests fell within the range 0.003–0.073m, with a mean of 0.0384m and a standard deviation of 0.0218m. The maximum recorded error was within the allowed margin.

Rotational accuracy was tested by placing the robot orthogonal to a wall at a known distance y. A laser pointer fixed to the robot is used to indicate its heading. The robot rotates on the spot by a specified angle. The distance x along the wall between the laser dots from before and after the rotation is measured. The 'ground truth' angle was calculated using $\theta = \tan^{-1}\frac{y}{x}$, which was then compared with the odometry to obtain the error. The accuracy was tested through a range of $-90°$–$90°$. The absolute error results of the rotational accuracy tests fell within the range

Algorithm 2 AABB Local Map

```
1:  **Input:** Lasers $\in$ [minr, maxr]$^{512}$
2:  Let res be the cell resolution (m/cell)
3:  $k = 0.25$
4:  bad $= \{\text{NaN}\} \cup [-\infty, \text{minr})$
5:  occ be a $w \times h$ occupancy grid initialised to Untouched
6:  occ$_{r,\theta}$ be the cell $(r,\ \theta)$ from the robot's location
7:  **for** $l \in$ Lasers **do**
8:      **if** $l \in$ bad **then skip**
9:      $l := \min(l, \text{maxr})$
10:     **if** $l \neq$ maxr **then**                ▷ not blocked if maxr
11:         Let $\theta$ be the angle of reading $l$
12:         occ$_{l,\theta}$ += LikelyBlocked[0.8]
13:         **if** $(X_1 \sim \text{Bernoulli}(0.6)) = \text{true}$ **then**
14:             occ$_{l-\text{res},\theta}$ += LikelyBlocked[0.8]
15:             occ$_{l-2\text{res},\theta}$ += LikelyBlocked[0.7]
16:             **if** $(X_2 \sim \text{Bernoulli}(0.8)) = \text{true}$ **then**
17:                 occ$_{l-3\text{res},\theta}$ += LikelyBlocked[0.7]
18:                 occ$_{l-4\text{res},\theta}$ += LikelyBlocked[0.7]
19:     **while** $l > 0$ **do**
20:         $l := l - \text{res}$
21:         **if** occ$_{l,\theta} \neq$ LikelyBlocked[$X$] **then**
22:             occ$_{l,\theta}$ += LikelyFree[1]
23: **for** all cells $c \in$ occ **do**            ▷ likelihoods to probability
24:     **if** $c =$ Untouched **then** $c := 0.5$
25:     **if** $c =$ LikelyBlocked[$L$] **then** $c := e^{-kL}$
26:     **if** $c =$ LikelyFree[$F$] **then** $c := 1 - e^{-kF}$
27:     clamp $c$ to the range: [0.05, 0.9]
```

0.00–1.35°, with a mean of 0.426° and a standard deviation of 0.585°. The maximum recorded error was within the allowed margin (2.49°). Specifically, the accuracy achieved was within $5.75 \tan 0.426° = 0.0427$m at a distance of 5.75m on average.

b) Neural Network: Initially, the network began training using early-stopping as temporal regularisation, with k-fold cross validation ($k = 10$) used to estimate the generalisation error. This approach was abandoned when no overall trend appeared between the number of training epochs and generalisation error, as evidenced by Figure 2.

Due to time constraints, an alternative approach was taken. The network was trained with as many epochs as possible, with partially trained weights being stored every 100 epochs. A tool was written to repeatedly load weights into the network while running SLAM, which could be used to evaluate performance manually.

2) Local Map Comparison: 'Ground truth' local maps for analysing the local map generation methods were gathered from the ROS navigation stack using `rosbag` and processed by a human. Local maps were generated using every method, as well as the 'ground truth' method at 10 random poses. A threshold was applied to each cell of a local map, classifying it into three classes: class 0 are the cells with a high certainty of being blocked ($0.667 \leq x \leq 1.00$); class 1 are the cells

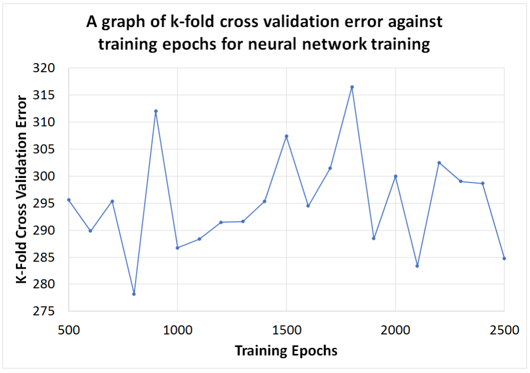

Fig. 2: k-fold validation error against number of training epochs.

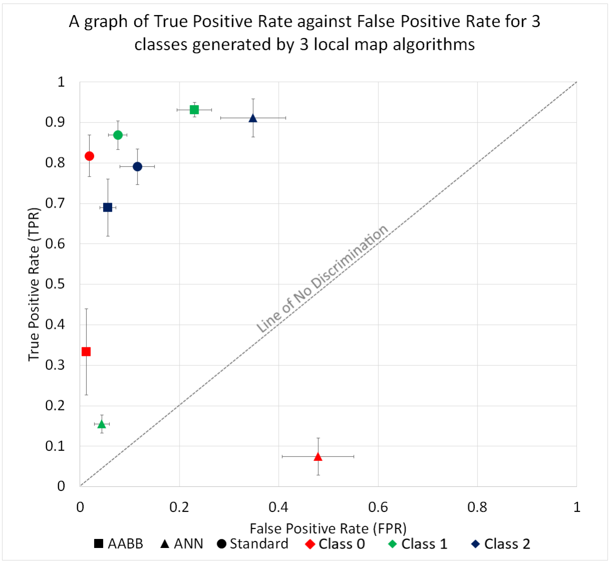

Fig. 3: ROC analysis of each class for each local map generation algorithm. Standard error is plotted.

which are unknown ($0.333 \leq x \leq 0.667$); class 2 are the cells with a high certainty of being free ($0.00 \leq x \leq 0.333$). A confusion matrix was used to calculate sensitivity and specificity values for each class, using a 'one-against-all' method.

A ROC graph (Figure 3) was plotted with a point for each method and class of pixel averaged over all 10 poses. From a visual inspection of the combined classes, the standard method performs best, followed by the AABB method and finally the neural network. The data follows the same trend for the error size and analysing each class individually. This suggests that the standard method performs more consistently than the other algorithms with regards to whether a pixel is occupied or not. The False Positive Rate (FPR) and True Positive Rate (TPR) were defined to be weighted equally[†].

3) Discussion: The experimental evidence suggests the use of the standard method, as it performed consistently better than the other two methods. The neural network's poor performance was surprising, achieving an average TPR of 0.38 and FPR of 0.29. In some cases, inverting the output would increase performance. As previous research shows this technique working well in similar scenarios, such as the varying versions compared by Cruz *et al.* (2002), it is worth analysing why the neural network performed so poorly.

A subtle difference between the Sense-and-Drive algorithm implemented and the one described by Van-Dam, Kröse and Groen (1996) may have lead to inaccurate training data. Since a laser sensor was used for collision detection instead of a touch sensor, distance measurements may fall slightly short of the true values, causing issues when run through the neural network training.

The neural network training displayed strange oscillating behaviour (Figure 2), suggesting the neural network weights never reached their global minimum with respect to the cost function. A possible cause is that the learning rate, η, needed to decay so that the weight changes do not 'jump over' the global minimum. Improvements to Sense-and-Drive and changing the learning rate were not possible due to time constraints so these would be suggestions for the future.

V. Stitching

A. Background

Stitching is the process of correcting odometry errors, solving the closed loop problem. This can be thought of as finding the 'best fit' for the local map onto the global map prior to updating occupancy values. Even the smallest odometry errors accumulate over time, resulting in incorrect maps. This leads to curved walls in place of straight ones on the global map. The values provided by the odometry are used as a first estimate of the robot's position; then the stitching method searches locally for an improved match.

1) Literature Review: Feng and Borensein (1996) produced a paper observing the effects of systematic errors in odometry. Systematic errors, also referred to as drift, are consistent and predictable, e.g. a camber in the tyre balance of the robot. This can lead to significant odometry errors after just 10 units of travel (Feng and Borensein, 1996). However, their relative consistency allows the errors to be predicted and nullified. Kümmerle, Grisetti and Burgard (2011) acknowledged the importance of identifying the source of errors, but argued that a strong method of general error correction can negate both systematic and non-systematic errors and could be a more efficient use of time. Given the time limitations of this project, learning the so-called 'motion model' of the robot was infeasible. This should not distract from the benefit of estimating the systematic translation and rotation errors. A particularly intriguing study used an automated process to consistently refine the motion model, eliminating the need for human input and reducing the effects of dynamic obstacles (Roy and Thrun, 1999).

Many methods have been proposed to reduce odometry errors by using laser readings to improve localisation. Roy and Thrun (1999) acknowledge this method's usefulness when a model of odometry data is unavailable and non-systematic errors high. However, they state this approach is understandably ineffective in areas will little input (large open spaces) or when the sensor feedback is unreliable (in small, crowded places), arguing their approach can produce more desirable results. They neglect that their method does not account for sudden large odometry changes, such as small collisions, which could cause the robot to become lost. This shows a balance between learning systematic errors and an appreciation for non-systematic errors is needed for effective odometry error neutralisation.

Wyatt (2015) presented a method to minimise a cost function of the local map applied to the global map, implemented as 'Brute' below. Kleeman and Diosi (2005) noted brute's $\mathcal{O}(n^2)$ time complexity and tendency to become stuck on local minima, so produced a more efficient algorithm that works directly on the laser data's raw polar form. Burgard, Fox and Thrun (2000) states that as pose errors become arbitrarily large, this algorithm has a higher chance of failing to find the optimal solution. In terms of the closed loop problem, they state that past poses *might* have to be revisited to generate a consistent map. They produced an algorithm that purely does this, even without odometry data in certain situations. Despite these points, it is still possible to void odometry error using extensions on the brute algorithm. Lingemann *et al.* (2004) proposed a method to extract 'interesting' aspects of the local map (such as line segments and corners), extract the same data from the global map and attempt the match using a similar concept to brute.

B. Design

If sufficient useful information is not present in the local map, then all of the methods perform badly. To roughly determine the information in a local map, one can count the cells with values above a threshold (i.e. walls). Stitching is not performed if enough of these cells are not present. Other error reducing mechanisms were also employed, such as enabling stitching only every few local map updates and assuming the odometry to be wholly accurate otherwise.

1) Brute Force Search: 'Brute': Transformations of the local map over a small area are evaluated using a cost function and the optimal transformation is returned. Not searching through translations and only evaluating different orientations gave better results in parameter testing. The following cost function was used: $\alpha\sqrt{\Delta x^2 + \Delta y^2} + \beta \cdot \Delta\theta + \chi\,\mathrm{mapdiff}$. $\mathrm{mapdiff}$ is the summed absolute difference for each pixel between the rotated local map and global map. Parameter testing showed the best values were: $\alpha = 1$, $\beta = 100$ and $\chi = 1$ with a search range of $-5°$–$5°$.

2) Gradient Descent Search: The same cost function as brute is evaluated for two different rotations. Then gradient descent is used to search for the rotation, resulting in a minimum of the cost function. An exponentially decreasing learning rate $\eta = 0.01 \times 0.85^n$ is used, initially set relatively high, so the algorithm converges to acceptable results in few iterations. This method has a longer computation time than Brute ($\approx$1.5 times), but does provide sub-step accuracy unlike brute, which is locked to a set step size.

3) Iterative Closest Point: 'edge': This method takes a different approach, used in several existing SLAM implementations (Tomono, 2009). It uses the laser scan data directly rather than the local map cells. This method differs

Stitching Type	Pose	SD	Orientation	SD
Edge	23.79	10.27	122.9	43.17
Gradient	29.97	5.88	92.99	43.14
Brute	41.55	5.79	87.11	72.51
No Stitching	31.60	13.09	101.95	53.14

TABLE I: average Δpose and respective SD in translation and rotation.

T-Test	Edge	Gradient	Brute	None
Edge	-	0.33	0.34	0.5
Gradient	0.27	-	0.85	0.8
Brute	0.0072	0.0106	-	0.7
None	0.31	0.8	0.15	-

TABLE II: t-test p values for all combinations of stitching for orientation (blue) and total translation offset (red). The deeper colours indicate significant values.

from other ICP SLAM implementations as it uses the cells of the global map as the reference point cloud, rather than storing a global map point cloud directly. The laser scan data forms the 'source' point cloud and the global map cells which are above the 'wall threshold' form the 'reference' point cloud. ICP from the library '`libpointmatcher`' is used to find a good rigid transformation (on the $z = 1$ plane) of the source point cloud onto the reference one. The data from the resulting transformation matrix is extracted to a translation and rotation, which is then applied to the local map cells. To improve results, out of range and erroneous laser readings are discarded. 'Point-to-plane' matching is performed, as it allows points to 'slide' along planes (edges) of the reference cloud, leading to closer matches.

C. Empirical Study

1) Aim: To prove the following hypothesis: edge stitching is statistically better at reducing the effect of odometry noise on the global maps produced than either brute, gradient or without stitching enabled.

2) Method of Testing: The robot was started with a deterministic 'wall hugger' algorithm and allowed to explore the map. Due to the nature of 'wall hugger', the robot will end at its starting pose, hence forming a loop. If the true difference in any of x, y or orientation was not 0, it was discarded as an anomalous result. Afterwards, the absolute differences between the estimated starting and ending poses and orientations were calculated. This was repeated six times for all types of stitching and with stitching disabled.

Another separate run was completed using the same starting positions and 'wall hugger'. For each stitching type, the x and y translation and rotational offset for each independent stitch was noted, until 31 stitches were collected.

3) Results: The mean, standard deviation (Table I) and t-tests (Table II) were calculated for every possible combination. The translation offset for all Edge stitches, as well as the calculated average and first SD, were plotted onto Figure 4. All angle offsets and the respective SD, for all three stitching types, were plotted onto a histogram (Figure 5).

The statistical analysis on stitching showed that none of the methods corrected for rotational odometry noise

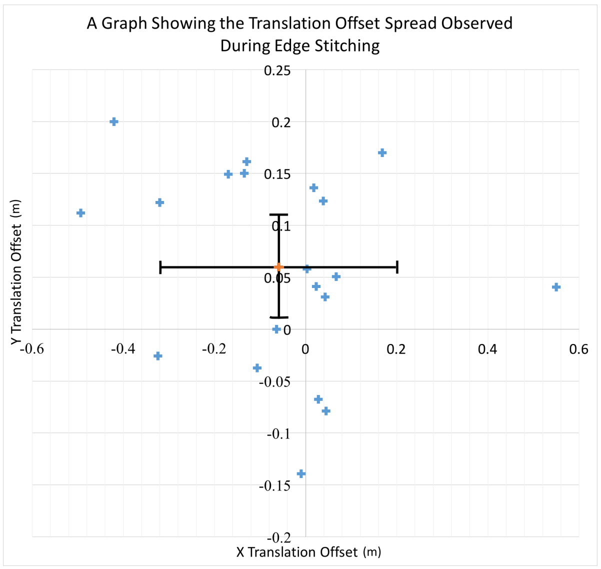

Fig. 4: Edge stitching translation. Graph shows individual stitches in blue and the average (with outliers removed) in orange. First SD is also shown.

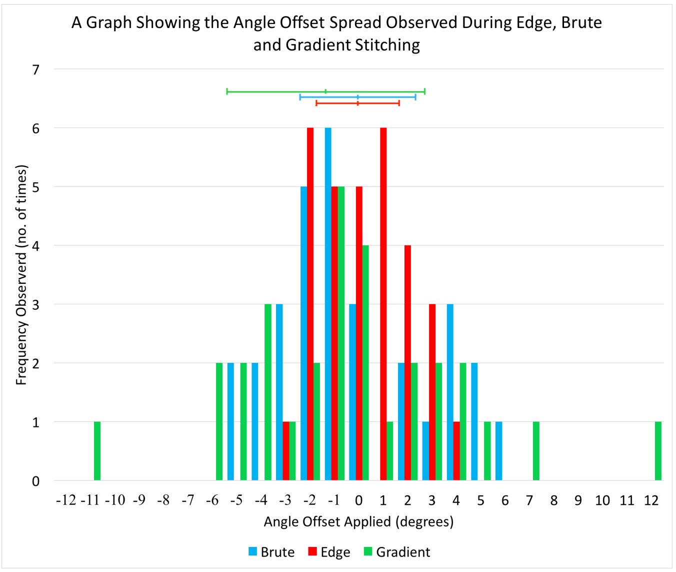

Fig. 5: A histogram in 1° increments, showing every Edge, Brute and Gradient rotational stitch that has been observed. The mean and first SD have been shown.

significantly, with the best result facing 87° in the wrong direction. Pose correction had different results: that Edge and Gradient were significantly better than Brute (being 23.79m and 29.97m from the true destination, rather than 41.55m). However, the results show that neither Edge nor Gradient were statistically significant in comparison to where stitching was disabled (Table II).

Figure 5 shows that Gradient chose a larger range of rotational offsets to apply (-11–$12°$), in comparison to Brute (-5–$6°$). Edge cannot be directly compared as it also applied translation. Figure 4 shows that Edge never chose a translation offset greater than 0.57m and, on average, chose a very small offset of $(-0.05\text{m}, 0.05\text{m})$. Edge also produced a very small range of orientation offsets ($-3°$–$4°$); this could explain why it had the worst orientation accuracy (Table I). It should be noted that it was not statistically significant in comparison to the other algorithms.

D. Discussion

Many reasons account to the failure of stitching. Systematic errors in the robot were not accounted for (subsubsection V-A.1). In doing so, measurable and consistent drift could have been identified and accounted for, reducing the total odometry noise in the system. Looking purely at the pixel-to-pixel map difference to ascertain how closely the local and global maps match, leads to the algorithms becoming stuck on local minima (Kleeman and Diosi, 2005). The cost function should prevent the algorithm choosing very large translation and rotational offset, even if that map difference is low. However, tuning these values to produce consistently valid results is difficult; different situations required alternate weighting of values.

Lingemann *et al.* (2004) went further with feature extraction: rather than extracting just walls (like Brute), they extracted corners and lines then matched these features. They found this to be very effective. This would reduce errors where the pure map difference produced incorrect results.

VI. Occupancy Grid Update

A. Background

Following stitching, the probabilities from the local map are used to update the posterior of the corresponding cells on the global map. In particular, the probability for each cell being occupied *given* the observations recorded is updated. Two methods can be used for this: a probability based method and one using log-odds.

The probability based update is derived from Bayes' rule, assuming that noise in the different sensor readings is independent of the true occupancy value of a cell, i.e. the cell (x, y), $\mathrm{P}(s^t \mid \mathrm{occ}_{xy})$ is independent of $\mathrm{P}(s^{t'} \mid \mathrm{occ}_{xy})$ where $t \neq t'$ (Thrun, 1998, p. 28). An issue with this method is that floating point numbers close to 0 are present and, when multiplied, the outcome remains close to 0. As such, Thrun, Burgard and Fox (2005) suggest a log-odds representation, which produces more stable results and removes the aforementioned problems.

B. Design

Both implementations use a local map containing probabilities and a global map with the same resolution. The local map may contain cells that are out of range of the LiDAR, these are denoted as -1 and ignored.

1) Probability Update: The global map stores the probability that a cell is occupied *given* all recorded observations. When given a new local map and the best fit between the two, the following update function is applied (Wyatt, 2015):

$$\mathrm{P}(\mathrm{occ}_{xy} \mid s^t \wedge s^{t-1} \wedge \cdots \wedge s^1) = 1 - (1 + p)^{-1}$$

Where

$$p = \frac{\mathrm{P}(\mathrm{occ}_{xy} \mid s^t)}{1 - \mathrm{P}(\mathrm{occ}_{xy} \mid s^t)} \frac{\mathrm{P}(\mathrm{occ}_{xy} \mid s^{1:t-1})}{1 - \mathrm{P}(\mathrm{occ}_{xy} \mid s^{1:t-1})} \frac{1 - \mathrm{P}(\mathrm{occ}_{xy})}{\mathrm{P}(\mathrm{occ}_{xy})}$$

Where $\mathrm{P}(\mathrm{occ}_{xy} \mid s^t)$ corresponds to the value held in the local map and $\mathrm{P}(\mathrm{occ}_{xy} \mid s^{1:t-1})$ represents the value held in the global map. Because the global map is initialised to 0.5, the prior: $\frac{1-\mathrm{P}(\mathrm{occ}_{xy})}{\mathrm{P}(\mathrm{occ}_{xy})} = 1$ and therefore is ignored.

2) Log-Odds Update: This method stores two synchronised global maps: one holds the log-odds that a cell is occupied and the other stores the probability. The occupancy grid update is applied to the log-odds map; the probabilities global map is only used for visualisation. The following update function is used (Thrun, Burgard and Fox, 2005):

$$\mathrm{logOcc}_{xy} = \mathrm{logOcc}_{xy} + \ln\left(\frac{\mathrm{P}(\mathrm{local}_{xy})}{1 - \mathrm{local}_{xy}}\right)$$

The value is then clamped to within $\ln\left(\frac{0.99}{1-0.99}\right)$ and $\ln\left(\frac{0.01}{1-0.01}\right)$ to prevent extreme values from occurring. Finally, the probability-based global map is updated using:

$$\mathrm{probOcc}_{xy} = 1 - \frac{1}{1 + e^{\mathrm{logOcc}_{xy}}}$$

C. Empirical Study

Due to the highly theoretical nature of this component, an informed decision can be made to use the log-odds method. This method eliminates issues with floating point values close to 0, improving stability.

VII. Planning Under Uncertainty

In order to increase the area traversed by the robot, an exploration algorithm was developed which, given an incomplete global map, identifies areas of interest and chooses to explore them.

A. Background

Newman *et al.* (2003) presents an exploration algorithm which assigns costs to potential goals and moves towards the goal with optimal cost. Nearby unknown areas are rated highly in an attempt to explore efficiently. Goals are rated by sampling nearby points enabling an accurate view of the uncertainty in the surroundings. In contrast, Sim and Little (2006) argues that shooting rays from the estimated robot's position is faster and still produces good coverage. When Newman *et al.*'s (2003) algorithm fails to find suitable plans, it opts to fall back to scanning for goals within the local map.

B. Design

This exploration algorithm consists of two main stages. Firstly, all possible plans within the algorithm's scope are generated. Then the optimal plan is chosen and executed.

1) Generating Plans: A snapshot of the global map and robot location is recorded immediately prior to a planning session (i.e. a full run of the algorithm). Snapshots are beneficial over live data, as it remains in a consistent state whilst planning. During a planning session, 38 rays are traced concurrently (each at separate angles) from the robot's location until an unknown cell is found or a ray surpasses 1000 cells in length. If an unknown cell is found, a route is computed using greedy first search, balancing optimal routes against the time for the planning to complete. If no unknown cells are found, the algorithm will wait until the session is finished, which usually occurs after 15 seconds.

If a route is formulated by a planner then the session continues to run for a further 4 seconds. Time limits are employed to reduce the discrepancy between the real robot's position and the snapshot location. Additionally, without time-outs, the planner may try to explore an infinite graph such that it would never terminate.

2) Executing the best plan: In a session, multiple potential plans may be generated, but only one can be executed. Therefore, plans are assessed based on the cost function below using live global map data. A new plan will interrupt the currently executing plan if its cost is at least 3 less.

$$\begin{aligned}
0.1 &\times \text{number of cells in path} &+\\
200 &\times |\text{global map weight of the goal cell} - 0.5| &+\\
2 &\times \text{number of turns in path} &+\\
4 &\times \text{number of walls in the path} &+\\
8 &\times \|\text{robot location} - \text{closest node on the path}\|
\end{aligned}$$

To execute a plan, the robot searches for the closest node on the path. If the robot is not in that node's corresponding cell, it rotates and moves towards it; otherwise it will aim for the next turn in the path. This is repeated until the destination is reached or the plan execution is aborted (either because a better plan is found or the robot has executed more steps than $8 + N_{\text{turns}}$ in the plan). The robot stops moving if the laser readings detect any imminent obstacles. Should there be no plans to execute, another exploration algorithm is utilised, such as 'wall hugger' or 'hoover'.

C. Empirical Study

Given that multiple exploration algorithms are available, it is worth testing each in order to determine which is most appropriate for use with the rest of the system. The algorithms available to the system are: 'explore with hoover', 'explore with wall hugger', 'hoover' and 'wall hugger'.

D. Method

In a simulator, a so-called 'fake' SLAM was produced. This version performs identically to the 'real' implementation, with the exception that each global map produced is perfect, eliminating both odometry and laser noise. This isolates the test to evaluate only the exploration ability, rather than external factors like the quality of the 'real' SLAM.

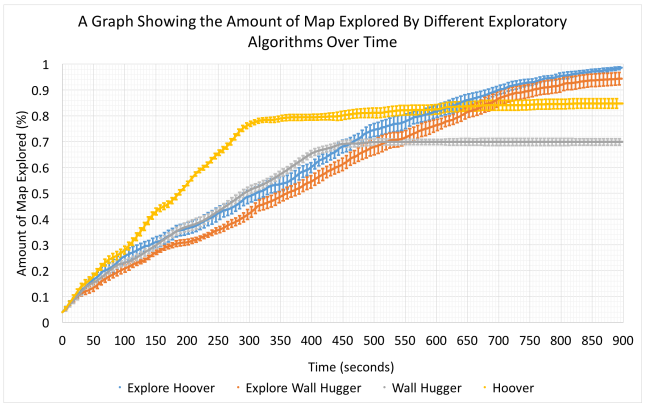

Fig. 6: Explored percentage of the empty square map for each exploration algorithm with respect to time. Standard error bars are displayed for every fifth second.

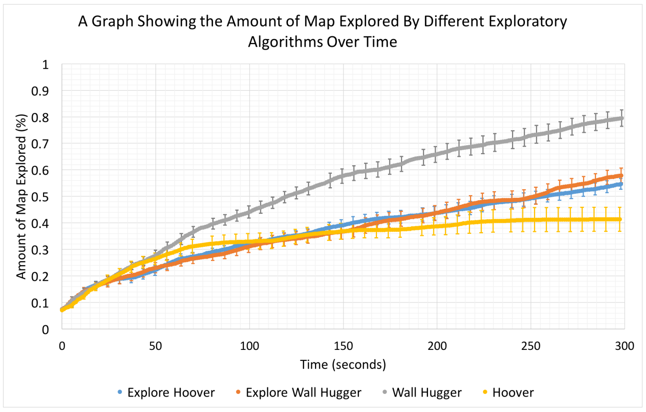

Fig. 7: Explored percentage of the LG floor map for each exploration algorithm with respect to time. Standard error bars are displayed for every fifth second.

Two maps were used for testing, an empty square and the LG floor of the Computer Science building. For the former, each algorithm was given 15 minutes to explore starting from one of 8 random poses. For the latter, each algorithm was allocated 5 minutes, repeated from 16 random starting poses. For every second of the tests, the percentage of the map explored by the algorithm was recorded. This was achieved using two versions of the same map: a fully revealed map and a map updated by the 'fake' SLAM over time.

E. Results

Figure 6 shows the results for all four algorithms on the square shaped map; Figure 7 shows the results for the LG floor map. *t*-tests were performed for each algorithm pair on each map and found that all differences were significant ($p < 0.05$), apart from the differences between both explore algorithms on the LG floor.

In Figure 6, it can be seen that 'hoover' has the fastest exploration rate. However, 'hoover' and 'wall hugger' cease to gain any further significant information past the 300 and 450 second mark, leading to 84% and 70% of the map being explored respectively. The highest exploration performance was 'explore with hoover', exploring 98% of the map in the

15 minutes. Closely followed by ‘explore with wall hugger’, exploring 95% after 15 minutes.

As shown by Figure 7, ‘wall hugger’ performed significantly better than all the algorithms when executed on the LG floor map; having the highest exploration rate and total performance (80% explored after 5 minutes, with an average percentage change of 0.26 per second). The *t*-tests showed no significant difference between the two variations of explore. However, both of these were significantly better than ‘hoover’, which managed to explore only 42% of the map on average, in comparison to 55% and 57% for ‘explore with hoover’ and ‘explore with wall hugger’.

F. Discussion

In an open square map, ‘explore’ eventually explores the entirety of the map, exposing the weaknesses of unintelligent methods such as ‘wall hugger’ and ‘hoover’. Though the slow nature of tracing 38 rays in the plan generation stage means that the unintelligent methods perform better at first. Even though Sim and Little (2006) traced a similar amount of rays (36), parameter testing could be performed to find the best balance between the number of rays (which is proportional to the time taken) and the quality of plans. Additionally, using A* search or using efficient data structures, such as octrees or k-d trees for ray tracing, could improve the overall performance.

Another consideration is that the current plan is not aborted if the robot maps the unknown area before reaching the goal, wasting additional seconds. Newman *et al.* (2003) produced a very effective implementation that falls-back to searching for goals within the local map, rather than falling-back to a different crude exploratory algorithm. This method does increase the time taken for plan generation, although by a relatively small amount as the local map search space is magnitudes smaller than the global map.

Another limitation is exposed when occupied cells are misidentified by the local map generation, commonly seen near glass. Explore will ‘see’ this as an interesting area and attempt to drive into the glass. Neither Newman *et al.* (2003) or Sim and Little (2006) address this issue; perhaps sampling the surrounding area reduces the effect of this as per the implementation described by Newman *et al.* (2003). However, it would be more effective to reduce the false negative detection near reflective surfaces, using a technique outlined by Wang and Wang (2016). This prompts a consideration to the validity of this empirical study: using a ‘perfect’ SLAM implementation will over-emphasise the effectiveness of ‘explore’ as no cells will be misidentified. This study should be repeated with a standard SLAM implementation in the real world, rather than assuming a noise-free environment, to produce more representative results.

VIII. Full System Testing

A. Aim

To statistically analyse the overall effectiveness of the SLAM implementation, using only global maps generated in the real world.

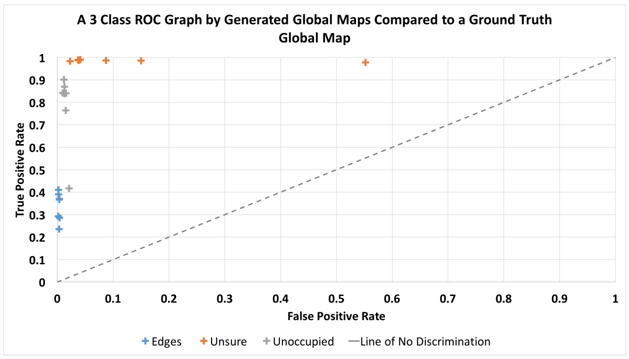

Fig. 8: The results produced from the global maps generated by the best SLAM implementation. The TPR and FPR for each global map were calculated for three classes: edge cells, unoccupied cells and uncertain cells. The two outliers were from the ‘wall hugger’ repeat. Each point represents a different run.

B. Method of Testing

Firstly, a ‘ground truth’ global map of the LG floor was produced using the `rosbag`, manually corrected to maintain the relative thicknesses of each wall. The robot was placed at a random pose in the LG floor and SLAM with edge stitching and standard local maps was launched, with exploration provided by the ‘wall hugger’ algorithm. When the LG floor had been entirely explored (or 15 minutes had passed), the global map was saved. This process was repeated five times at different locations using different exploratory algorithms (‘explore with wall hugger’, ‘explore with hoover’, ‘hoover’ and two human drivers). Every saved global map was compared to the ‘ground truth’ map using ROC analysis, separated into three classes: edges, unoccupied space and unsure occupation (just as in subsubsection IV-C.2).

C. Results

The sensitivities and specificities for all three classes were plotted on a graph of TPR against FPR (Figure 8).

As shown by Figure 8, on average, the FPR for free space placement was very low (0.014). The TPR was slightly less effective (ranging from 0.4 to 0.9, on average 0.78), predicting free space with an accuracy greater than a 0.7 TPR for six of the seven exploratory algorithms. Edge classification was again very effective at not falsely identifying walls (with the FPR ranging from 0.001–0.003, with an average of 0.002). However, the TPR was low (0.23–0.43).

D. Discussion

Many walls were simply not placed when present in reality. This could be explained by the abundance of glass and reflective surfaces in the environment, to which the lasers respond very poorly, returning larger distances than reality. Additionally, due to the uncorrected odometry errors, walls will be placed in the wrong position over time; this was confirmed visually, suggesting a failure in stitching as discussed. If many walls were placed in the wrong position, one would expect a large FPR in edge classification. However, acknowledging that the occupancy grid update will reduce the placement of walls over previously unoccupied space (large odometry errors will frequently cause this) explains the unexpected results obtained.

Two results are significantly worse[†] than the average for both unsure and unoccupied classification. Both of these points are produced from the run using 'wall hugger'. Due to the winding movement of this exploratory algorithm, odometry errors become prominent very quickly, decomposing the entire global map structure. It follows that edge placement should be worse than average; this is not the case however. Perhaps this error is offset by the more accurate laser readings against glass and reflective surfaces, due to the small distance 'wall hugger' maintains to the walls. The run using explore produced the best[†] unsure and unoccupied results. The slower movement and limited, deliberate rotation reduced the effect of odometry errors, allowing a more accurate map to be produced.

Overall, the SLAM implementation was effective at setting cells to unoccupied and unsure, but failed at locating edges and accurately placing them. This was possibly due to failures in local map generation and stitching. Figure 8 showed how sensitive the map generation was to the exploratory algorithm chosen; algorithms with many rotations significantly reduce the quality of the map produced. To improve, the SLAM implementation needs to: generate local maps more accurately, especially near reflective objects; reduce systematic and non-systematic odometry errors more effectively; and intelligently ignore global map discrepancies when exploring under uncertainty.

IX. References

Burgard, W., Fox, D., and Thrun, S. (2000) 'A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping', *Robotics and Automation*, 1, pp. 321-328.

Cruz, J., Nunes, U., Metrolho, J., and Lopes, E. (2002) 'A comparison of three neural networks for building local grid maps', *Sensors*, 1, p. 6.

Ellore, B.K. (2002) *Dynamically Expanding Occupancy Grids*. M.Sc. Thesis. Texas Technical University.

Feng, L. and Borensein, J. (1996) 'Measurement and correction of systematic odometry errors in mobile robots', *Transactions on Robotics and Automation*, 12 (6), pp. 869-880.

Karlsson, N., Di Bernardo, E., Ostrowski, J., Goncalves, L., Pirjanian, P., and Munich, M.E. (2005) 'The vSLAM algorithm for robust localization and mapping', *IEEE International Conference on Robotics and Automation*, pp. 24-29.

Kleeman, L. and Diosi, A. (2005) 'Laser scan matching in polar coordinates with application to SLAM', *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3317-3322.

Kümmerle, R., Grisetti, G., and Burgard, W. (2011) 'Simultaneous calibration, localization, and mapping', *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3716-3721.

LeCun, Y., Bottou, L., Orr, G.B., and Muller, K.R. (1998) 'Efficient BackProp' in *Neural Networks: tricks of the trade*. Berlin, Germany: Springer, p. 9-48.

Lingemann, K., Surmann, H., Nuchter, A., and Hertzberg, J. (2004) 'Indoor and outdoor localization for fast mobile robots', *Intelligent Robots and Systems*, 3, pp. 2185-2190.

Mitchell, T. (1997) 'Maximum Likelihood Hypothesis For Predicting Probabilities' in *Machine Learning*. 1st edn. Heidelberg, Berlin: Springer Science & Business, pp. 167-170.

Moravec, H. and Elfes, A. (1985) 'High resolution maps from wide angle sonar', *IEEE International Conference on Robotics and Automation*, 2, pp. 116-121.

Newman, P., Bosse, M., and Leonard, J. (2003) 'Autonomous feature-based exploration', *IEEE International Conference on Robotics and Automation*, 1, pp. 1234-1240.

Newman, P. and Ho, K. (2005) 'SLAM-loop closing with visually salient features', *IEEE International Conference on Robotics and Automation*, pp. 635-642.

Oliver, A., Kang, S., Wünsche, B.C., and MacDonald, B. (2012) 'Using the Kinect as a navigation sensor for mobile robotics', *Image and Vision Computing*, pp. 509-514.

Roy, N. and Thrun, S. (1999) 'Online self-calibration for mobile robots', *International Conference on Robotics & Automation*, pp. 0-8.

Sim, R. and Little, J.J. (2006) 'Autonomous vision-based exploration and mapping using hybrid maps and Rao-Blackwellised particle filters' *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2082-2089.

Thrun, S. (1998) 'Learning metric-topological maps for indoor mobile robot navigation', *Artificial Intelligence*, 99 (1), pp. 21-71.

Thrun, S., Burgard, W., and Fox, D. (2005) *Probabilistic robotics*. Vancouver: MIT Press.

Tomono, M. (2009) 'Robust 3D SLAM with a stereo camera based on an edge-point ICP algorithm', *Robotics and Automation*, 1, pp. 4306-4311.

Van-Dam, J.W., Kröse, B.J., and Groen, F.C. (1996) *Neural Network Applications in Sensor Fusion For An Autonomous Mobile Robot*. Berlin, Germany: Springer.

Wang, X. and Wang, J. (2016) 'Detecting Glass in Simultaneous Localisation and Mapping', *Robotics and Autonomous Systems*, 88, pp. 97-103.

Wyatt, J. L. (2015), 'Lecture 12: Mapping'. 06-13520: *Intelligent Robotics*. Available at: `https://canvas.bham.ac.uk/courses/21815` (Accessed: 21 December 2016).

[†] Note that the 'best' and 'worst' ROC results are calculated by ranking all the results by euclidean distance from the point (0, 1); a point which represents a result with no false positives nor false negatives.