2.
Theoretical Basis

2.1.
Probabilistic Robotics

"The key idea of Probabilistic Robotics is to represent uncertainty explicitly, using the calculus of probability theory" [1]. In other words, instead of relying on a single "best guess" probabilistic algorithms represent information by probabilistic distributions. By doing so, probabilistic robotics can mathematically represent ambiguity and degree of belief, enabling them to accommodate all sources of uncertainty.

The advantage of probabilistically programming robots, compared to other approaches that do not explicitly represent uncertainty, is simply because:

> *"A robot that carries a notion of its own uncertainty and that acts accordingly is superior to one that does not."* [1].

Probabilistic approaches are typically more robust under sensor limitation, sensor noise, environment dynamics, and so on. They are well suited to complex and unstructured environments, where the ability to deal with uncertainty is quite important. "Probabilistic algorithms are broadly applicable to virtually every problem involving perception and action in the real world" [1].

All these advantages, however, come at a price. The two most cited limitations of probabilistic algorithms are: a need to approximate and computational inefficiency. Because probabilistic algorithms consider entire probability densities, they are less efficient than non-probabilistic ones. Computing exact posterior distributions is typically infeasible, since distributions over the continuum possess infinitely many dimensions (most robot worlds are continuous). Sometimes, uncertainty can be approximated with a compact parametric model (e.g. discrete distributions or Gaussians); in other cases, a more complicated representations most be employed.

"At the core of probabilistic robotics is the idea of estimating state from sensor data" [1]. This sensor data are not directly observable, but that can be inferred. A robot has to rely on its sensors to gather information, while this information is only partial, and corrupted by noise. Thus, state estimation seeks to recover state variables from data.

2.1.1. Bayes Filter and SLAM

In Probabilistic Robotics, all quantities related in estimation such as sensor measurements, controls, state of the robot and its environment might be modeled as random variables. Random variables can take on multiple values, and they behave according to probabilistic laws. Probabilistic inference is the process of calculating these laws.

"Bayes rule is the archetype of probabilistic inference" [5]. It plays a predominant role in probabilistic robotics. Therefore, it is the basic principle underlying virtually every single successful SLAM algorithm. The Bayes rule is stated as [1]:

$$p(x \mid d) = \eta \, p(d \mid x) p(x) \tag{2.1}$$

If the quantity to learn is x (e.g. a map), using measurement data d (e.g. odometry, range scans), then Bayes rule tells that the estimation problem can be solved by multiplying two terms: $p(x|d)$ and $p(x)$. The term $p(x|d)$ is a generative model, it describes the process of generating sensor measurements under different worlds x. The term $p(x)$ is called the prior. It specifies the willingness before the arrival of any data. Finally, η is a normalizer that is necessary to ensure that the left- hand side of Bayes rule is indeed a valid probability distribution [5].

In robotic mapping there are two different types of data: sensor measurements and controls. Let´s denote sensor measurement (e.g. camera

images, LRF scans) by the variable z, and the control (e.g. motion command, odometry) by u. Let us assume that the data is collected in alternation:

$$z_1, u_1, z_2, u_2, \ldots \tag{2.2}$$

where subscripts are used as time index.

"In the field of robot mapping, the single dominating scheme for integrating such temporal data is known as Bayes Filter" [5].

The Bayes Filter is the extension of Bayes rule to temporal estimation problems [5]. It is a recursive estimator to compute posterior probability distributions over quantities that cannot be observed directly – such as a map or robot position. Let's call this unknown quantity the state x_t, where t is the time index. The generic Bayes filter calculates a posterior probability over the state x_t using the recursive equation [1]:

$$p(x_t \mid z^t, u^t) = \eta \; p(z_t \mid x_t) \int p(x_t \mid u_t, x_{t-1}) \, p(x_{t-1} \mid z^{t-1}, u^{t-1}) dx_{t-1} \tag{2.3}$$

where the superscript ^{t} refers to all data leading up to time t, that is:

$$z_t = \{z_1, z_2, \ldots, z_t\} \tag{2.4}$$

$$u_t = \{u_1, u_2, \ldots, u_t\} \tag{2.5}$$

Note that Bayes filter is recursive, that is, the posterior probability $p(x_t | z^t, u^t)$ is calculated from the same probability one time step earlier. The initial probability at time $t = 0$ is $p(x_0 | z^0, u^0) = p(x_0)$.

In the context of robotic mapping the state x_t contains all unknown quantities that are typically two: the map and the robot's pose in the environment. When using probabilistic techniques, the mapping problem is one where both the map and the robot pose have to be estimated in the same time altogether. Using m to denote the map and R for the robot's pose, the following Bayes Filter is obtained [1]:

$$p(R_t, m_t \mid z^t, u^t)$$
$$= \eta\, p(z_t \mid R_t, m_t) \iint p(R_t, m_t \mid u_t, R_{t-1}, m_{t-1})\, p(R_{t-1}, m_{t-1} \mid z^{t-1}, u^{t-1})\, dR_{t-1}\, dm_{t-1} \qquad (2.6)$$

If assumed a static world, the time index can be omitted when referring to the map m. Also, most approaches assume that the robot motion is independent of the map. And finally, using the Markov assumption, which postulates that past and future data are independent if one knows the current state x_t, the state x_t can be estimated using only the state x_{t-1} one step earlier. This results in a convenient form of the Bayes Filter for the robot mapping problem [5]:

$$p(R_t, m \mid z^t, u^t)$$
$$= \eta\; p(z_t \mid R_t, m) \int p(R_t \mid u_t, R_{t-1})\, p(R_{t-1}, m \mid z^{t-1}, u^{t-1})\, dR_{t-1} \qquad (2.7)$$

This estimator does not require integration over maps m, as it was the case for the previous one from eq. (2.6). The static world assumption is quite important, because such integration is difficult due to the high dimensionality of the space of all maps.

In eq. (2.7) two distributions probabilities have to be specified: $p(R_t|u_t, R_{t-1})$ and $p(z_t|R_t,m)$. Both are generative models of the robot and its environment.

The probability distribution $p(R_t|u_t, R_{t-1})$, often called to as *motion model*, specifies the effect of the control u on the state. It describes the probability that the control u, if executed at the world state R_{t-1}, leads to the state R_t.

The probability $p(z_t|R_t,m)$, often called to as *perception model*, describes in probabilistic terms how sensor measurements z are generated for different poses R_t and maps m.

However, eq. (2.7) cannot be implemented on a digital computer in its general stated form. This is because the posterior over the space of all maps and robot poses is a probability distribution over a continuous space, hence possesses infinitely many dimensions. Therefore, any working mapping algorithm has to take additional assumptions. These assumptions and their implications on the

resulting algorithms and maps constitute the main differences between the different solutions to the SLAM.

Figure 2.1 shows a generative probabilistic model (dynamic Bayes network) that underlies the essential of SLAM.

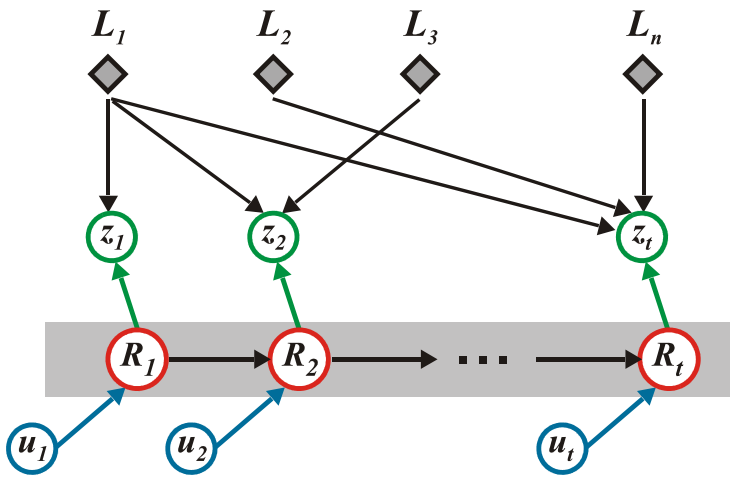

Figure 2.1: SLAM like a Dynamic Bayes Network

In particular, the robot poses, denoted by R_1, R_2, …, R_t, evolve over time as a function of the controls, denoted by u_1, u_2, …, u_t. The map is composed (as it will be seen later) by landmarks and each measurement of them, denoted by z_1, z_2, …, z_t, which are a function of its position L_1, L_2, …, L_n and of the robot pose at the time the measurement was taken.

Note that in this SLAM equation analysis the odometery u_t is assumed to be known, and this assumption will be kept till the Section 4.5.1 where odometry is replaced by Scan Matching.

2.1.2. Motion Model

"Robot motion models play an important role in modern robotics algorithms" [6]. The main purpose of a motion model is to model the relationship between a control input to the robot and a change in the robot´s configuration,

pose and map. Good models will capture not only systematic errors, but it will also capture the stochastic nature of the motion. The same control input will almost never produce the same result. “Thus, the effects of a robot’s action are, therefore, best described as distributions” [6].

This thesis focuses entirely on kinematics for mobile robots operating in planar environments. Kinematics describes the effect of control actions on the configuration of a robot. A rigid mobile robot is commonly described by six variables, its three-dimensional Cartesian coordinates and its three Euler angles (roll, pitch, yaw) referred to an external coordinate frame. But in a planar environment, the position of a mobile robot is summarized by three variables: its two-dimensional planar coordinates referred to an external coordinate frame, along with its angular orientation.

$$R = \begin{pmatrix} Rx \\ Ry \\ R\theta \end{pmatrix} \tag{2.8}$$

Figure 2.2 illustrates a robot pose in a plane.

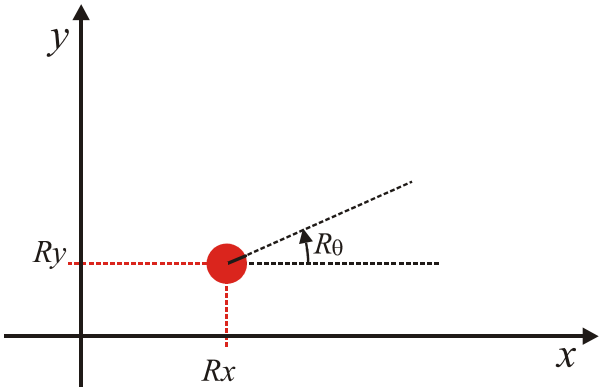

Figure 2.2: Robot pose

The orientation of a robot is often called *bearing*, or *heading direction.*

The probabilistic kinematic model, or *motion model*, plays the role of the state transition model in Mobile Robotics. As described in the previous section, this model is the probability distribution:

$$p(R_t \mid u_t, R_{t-1}) \tag{2.9}$$

Figure 2.3 shows two examples that illustrate the motion model for a rigid mobile robot in a planar environment. The robot's initial pose, in both cases, is R_{t-1}. The distribution $p(R_t|u_t, R_{t-1})$ is visualized in the form of a gray shaded area: darker areas mean more probability in robot position. In both figures, the robot moves forward some distance, during which it may accumulate translational and rotational errors. The right figure shows a larger spread of uncertainty due to a more complicated motion.

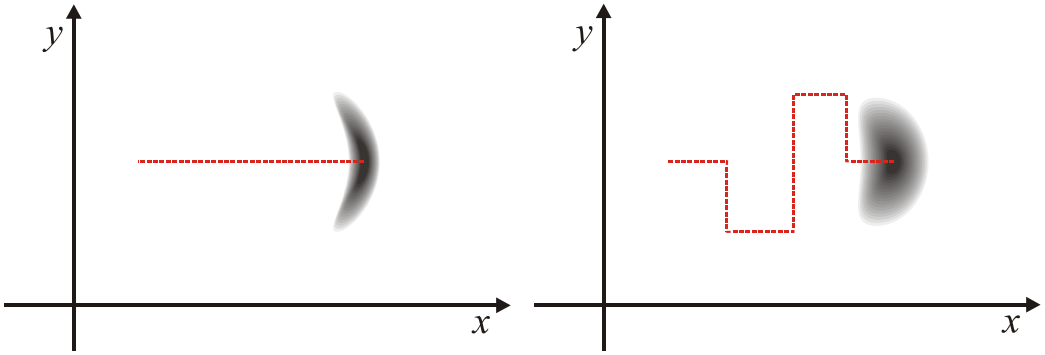

Figure 2.3: The motion model, showing posterior distributions of the robot's pose after executing the motion command u_t (red striped line). The darker a location, the more likely it is.

There are two motion models usually used. "The first model assumes that the motion data u_t specifies the velocity commands given to the robot's motors" [1]. Many commercial mobile robots (e.g. differential drive, synchro drive) are actuated by independent translational and rotational velocities. The second model, which is used in this work, assumes that u_t contains odometry information (distance traveled, angle turned).

However, odometry is only available as the robot moves. Hence it cannot be used for motion planning, such as collision avoidance [1]. “Technically, odometry are sensor measurements, not controls” [1]. But it is common to simply consider odometry as if it was a control input signal.

2.1.3. Perception Model

“The perception model comprises the second domain-specific model in probabilistic robotics, next to the motion model” [1]. In probabilistic terms, the Perception Model describes how sensor measurements z are generated for different poses R and maps m. As described before, this is modeled by:

$$p(z_t \mid R_t, m) \tag{2.10}$$

The Perception Model account for the uncertainty in the robot’s sensors. Thus, it explicitly models noise in sensor measurement. It could say that better results are acquired by a more accurate Perception Model. However, it is almost impossible to accurately model a sensor; reference [1] gives two primarily reasons[1]: “First, developing an accurate perception model can be extremely time-consuming; and second, an accurate model may require state variables that are not known, such as the surface material” [1]. In this way Probabilistic Robotics, instead of modeling the Perception Model by a deterministic function $z=f(x)$, accommodates the inaccuracies of Perception Model by a conditional probability density, $p(z|x)$. “Herein lies a key advantage of probabilistic techniques over classical robotics” [1].

Figure 2.4 shows a robot in an environment getting measurements from its Laser Range Finder (LRF). Given a position and the map of the environment, it is possible to use ray-tracing to get expected measurements for each rangefinder angle.

The result of modeling the sensor is shown in Figure 2.5. For a particular expected distance, the sensor will give a value near that distance with some

probability. So, given an actual measurement and an expected distance, it is possible to find the probability of getting that measurement using the graph from Figure 2.5.

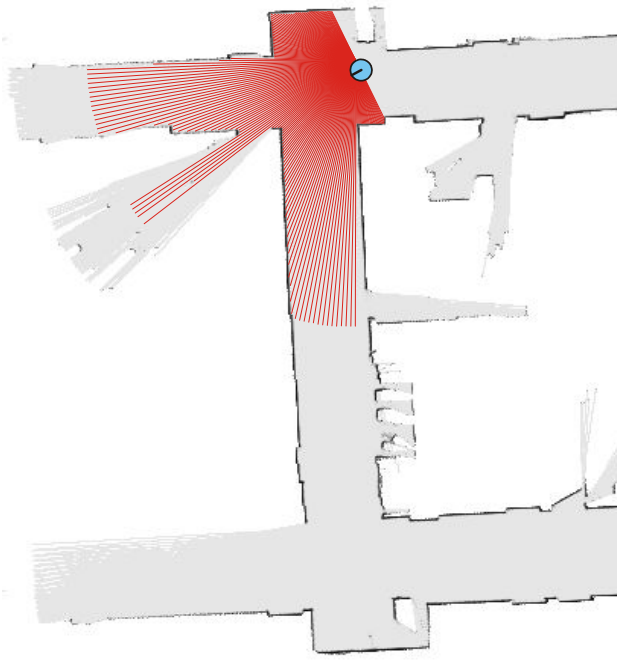

Figure 2.4: Robot in a map getting measurements from its LRF.

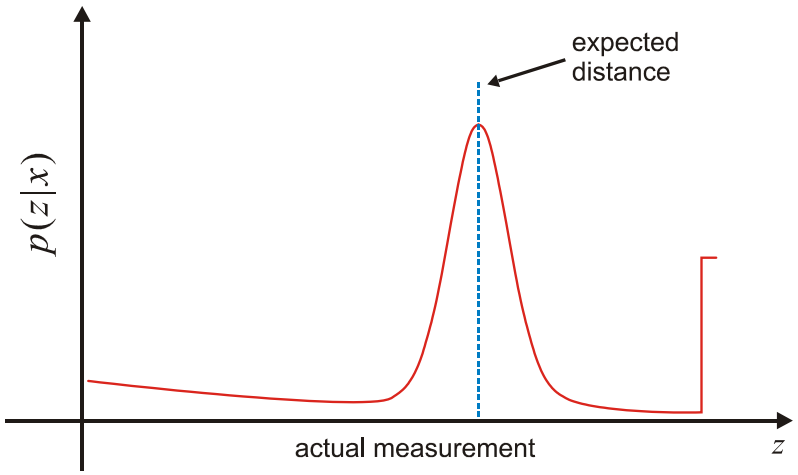

Figure 2.5: Given an actual measurement and an expected distance, the probability of getting that measurement is given by the red line in the graph.

Today's robots use a variety of sensor types, such as tactile sensors, range finder sensors, sonar sensors or cameras. The model specifications depend on the sensor type.

2.2. Map Representation

There are many reasons to have a representation of the robot's environment. Some of the purposes of having a map are listed in the following:

- **Localization.** The robot localization is possible making a correspondence between a given map and the observation of the robot's environment.
- **Motion planning.** Once the robot is located and given a target position, all necessary movements in the map can be compute to successfully move to the target.
- **Collision avoidance.** Using a map and robot's localization the navigation is possible without collisions.
- **Human use.** The map constructed by the robot can be used for exploration tasks in potentially hazardous environments.

In general, the map representation can be grouped into three main types: Geometric, Topological and Hybrid. However the general tendency in SLAM is to use geometric representation. Thus, this representation has also a subdivision: Landmark (or feature maps) and Grid maps.

2.2.1. Landmark Maps

The landmark-based maps consist of a set of distinctive point localizations that are referred to a global reference system. In structured domains such as indoor environments, landmarks are usually modeled as geometric primitives such as points, lines or surfaces.

The main advantage of landmark-base maps is their representation compactness. By contrast, this kind of map requires the existence of structures or objects that are distinguishable enough from each other. Thus, an extra algorithm for recognizable and repeatedly detectable landmark extraction is needed.

In practice, good landmarks may have similar traits, which often make them difficult to distinguish from each other. When this happens the problem of data association, also known as the *correspondence problem*, has to be addressed. "The correspondence problem is the problem of determining if sensor measurements taken at different points in time correspond to the same physical object in the world" [5]. It is a difficult problem, because the number of possible hypotheses can grow exponentially.

Figure 2.6 shows a simulated landmark-based map, where the blue asterisks represent the landmarks and the small triangle the robot position.

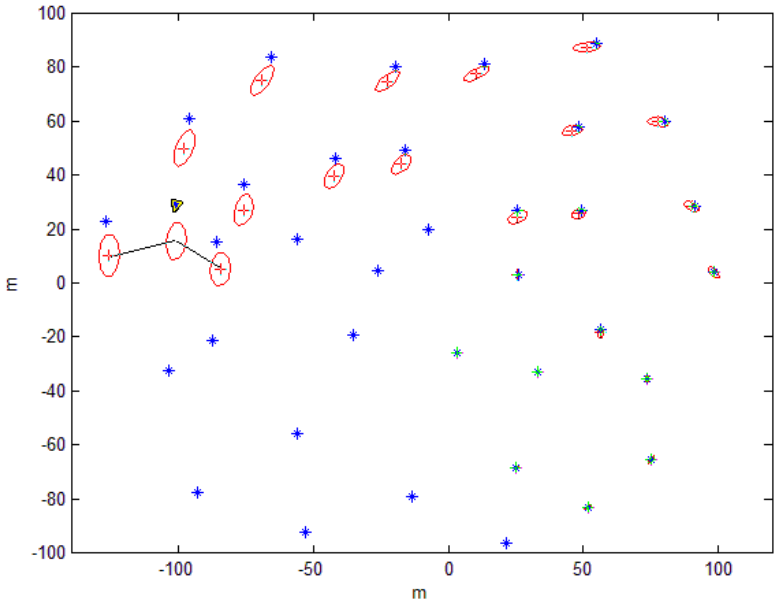

Figure 2.6: Simulated Landmark Map

2.2.2. Grid Maps

A grid map, or occupancy grid, is a popular and intuitive method to describe an environment. Occupancy grids were originally developed at Carnegie Mellon University in 1983 for sonar navigation within a lab [7].

Occupancy grids divide the environment up into a regular grid square, all of equal size. “Each of these grid squares correspond to a physical area in the environment and, as such, each square contains a different set of objects or portions of an object” [6]. An occupancy grid is an ideal representation of the environment, containing information on whether a square in the real environment is occupied or not.

The occupancy grid representation can be generalized into two types: deterministic and stochastic [6], described as follows.

- Deterministic map grids are the simplest representation, having two values for each grid square. Typically squares are considered as either Empty or Occupied, also sometimes is include a value for Unknown (or Unobserved). However, this representation is an exaggerated simplification for the sensors, since almost never a sensor will see a square of the environment which is both completely occupied and accurately observed.

- Stochastic maps, besides of Occupied and Empty, have a gradual scale of various degrees of occupancy. What percentage of the square is believed to be occupied, or how transparent the object is to the sensor are some factors that affect the occupancy value. The stochastic representation and the corresponding observation model need to be properly tuned for the sensor used.

Figure 2.7 shows a stochastic grid map, where the occupancy of each square is given in gray scale color, and darkest squares mean high probability of occupancy.

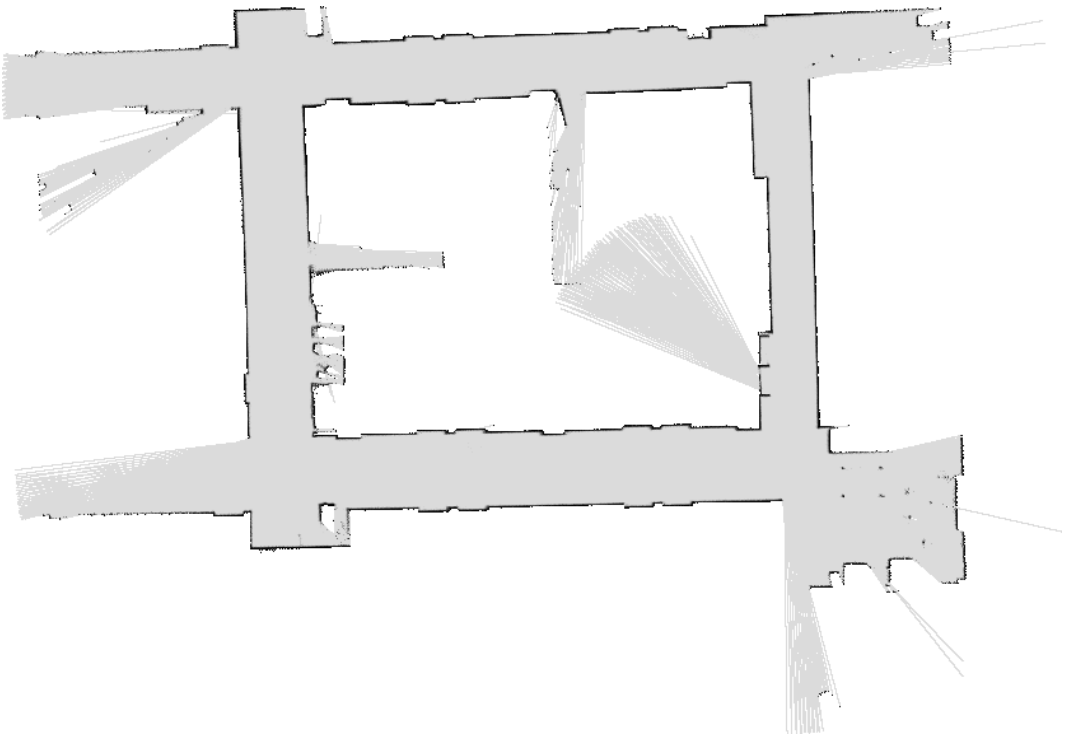

Figure 2.7: Grid Map: White regions mean unknown areas, light gray unoccupied areas, and darker gray to black represent increasingly occupied areas [8].

This work uses occupancy grid maps for environment representation, specifically stochastic occupancy grid maps.

2.3. Scan Matching

"Many SLAM algorithms are based on the ability to match two range scans or to match a range scan to a map" [9]. Laser Range Fiders (LRF) are popular sensors to get the input for scan matching, since their high reliability and their low noise in many situations.

The goal of scan matching is to find the relative displacement between the two positions at which the scans were taken. If a robot starts at position P_r (which is a reference pose), takes a scan S_r (reference scan), after that it moves through a static environment to a new pose P_n and takes another scan S_n (new scan), then scan matching seeks the difference of position P_n from posistion P_r (the relative translation and rotation) by aligning the two scans.

"The basis of most successful algorithms is the establishment of correspondences between primitives of the two scans" [9], i.e. point-to-point or feature-to-feature.

Different routines are developed to use point-to-point matching approaches such as the Iterative Closest Point (ICP) and the Iterative Dual Correspondence (IDC), both proposed by Lu and Milios [10]; and another, The Iterative Closest Line (ICL) proposed by Alshawa [11].

In [12] it is proposed a method that searches for features like corners and jump-edges from raw range scans. Another method based on feature extraction is HAYAI proposed in [13]. This method solves the self-localization problem for high speed robots.

One method that does not use correspondences between scans is the Normal Distribution Transform proposed in [9]. This method transforms the discrete set of 2D points reconstructed from a single scan into a piecewise continuous differentiable probability density defined on the 2D plane.

This work uses the NDT for scan matching but without using odometry information. But before getting to it; let's briefly review some of methods used for scan matching including NTD.

2.3.1. Point to Point Correspondence Methods.

- **The Iterative Closest Point (ICP)**

  The most general matching point to point approach was introduced by Lu and Milios in [10]. This is essentially a variant of the ICP (Iterative Closest Point) algorithm applied to laser scan matching.

  A scan is a sequence of points which represent a 2D plane contour of the local environment. "Due to the existence of random sensing noise and self-occlusion, it may be impossible to align two scans perfectly" [10]. Thus this method assumes two types of discrepancies between scans:

- in the first type, there are small deviations of scan points from the true contour due to random sensing noise, and
- the other type of discrepancy is the gross difference between the scans caused by occlusion. These discrepancy types are called outliers.

Adopting these criterions, ICP finds the best alignment of the overlapping part in the sense of minimum least-square errors, while ignoring the outlier parts. That is way ICP also need of correspondence search and outlier detection algorithms.

Lu and Milios [10] present two scan matching methods based on ICP. The first considers the two components (rotational and translational) separately; alternately fixing one, then optimizing the other. Given the rotation, least-square optimization is used to acquire translation.

Their second method called Iterative Dual Correspondence (IDC) combines two ICP-like algorithms with different point-matching heuristics.

- **The Iterative Closest Line (ICL)**

"ICL is similar to ICP, except that instead of matching query points to reference points, the query points are matched to lines extracted from the reference points" [14].

2.3.2. Feature to Feature Correspondence Methods.

- **Feature Based Laser Scan Matching for Accurate and high speed Mobile Robot Localization.**

Proposed by Aghamohammadi *et al.* [12]. This method divides the features into two types: features corresponding to the jump-edges and those corresponding to the corners detected in the scan**.**

In order to detect jump-edges, this method uses the natural consecutive order of points in the scan. Thus it defines a d_{th} which is the

maximum distance between two consecutive scan points. Beyond d_{th} these two consecutive points can be considered as jump-edges.

To obtain the second class of features, the corners, this method uses a line fitting algorithm. Thus the split-and-merge algorithm is used but only for line fitting. In this way, using two points taken of two consecutive lines, it searches for the farthest point to the straight line joining these two points.

Finally, after extracting features for two consecutive scans, a matching algorithm, based on a dissimilarity function is calculated.

This method is fast and it can be used for high speed mobile robot, but it suffers when the environment does not have corners or when it has circular walls, because no corners could be extracted and false jump-edges could be acquired.

- **The Highspeed and Yet Accurate Indoor/outdoor-tracking (HAYAI)**

HAYAI was proposed by Lingemann *et al.* [13]. This uses the inherent order of the scan data, allowing the application of linear filters for fast reliable feature detection.

Thus, this method chooses *extrema* in the polar representation of a scan as natural features. These *extrema* correlate to corners and jump-edges in Cartesian space. The usage of polar coordinates implicates a reduction by one dimension, since all operations deployed for feature extraction are fast linear filters.

For feature detection, HAYAI filters the scan signal using three one dimensional filters $\boldsymbol{\psi} = [\psi_{-1}, \psi_0, \psi_{+1}]$.The first one sharpens the data in order to emphasize the significant parts of the scan. The second one computes the derivation signal using a gradient filter. And, the last one smoothes the gradient signal to simplify the detection of zero crossing using a softening filter.

After generating the sets of features from both scans, the matching between both sets is calculated. But instead of solving the hard

optimization problem of searching for an optimal match, HAYAI uses a heuristic approach, utilizing inherent knowledge about the problem of matching features, e.g., "the fact that the features topology cannot change fundamentally from one scan to the following" [13].

"Although this method is a fast and feature based method for scan matching, it suffers from the lack of satisfying robustness property of feature extraction. It is well-suited for high range sensors" [12].

2.3.3. The Normal Distribution Transform

The assumed correspondences between two scans captured from two different poses of the robot are generally not true. That is why Biber [9] proposed a new method that does not need correspondences. Thus NDT makes an occupancy grid and subdivide the 2D plane into cells. To each cell, it assigns a normal distribution, which models the probability of measuring a point. "The result of the transform is a piecewise continuous and differentiable probability density, that can be used to match another scan using Newton's algorithm" [9].

This work uses the NDT for scan matching, which will be explained in detail next.

The NDT representation of one scan is built as follows: first, it subdivides regularly into cells of constant size the 2D space around the robot. Then, for each cell that contains at least three points:

1. collects all 2D-Points $x_{i=1\ldots n}$ contained in this cell.
2. calculates the mean:

$$\boldsymbol{q} = \frac{1}{n}\sum_i x_i \tag{2.11}$$

3. calculates the covariance matrix

$$\Sigma = \frac{1}{n}\sum_i (x_i - \boldsymbol{q})(x_i - \boldsymbol{q})^T \tag{2.12}$$

The probability of a 2D-point x contained in this cell is now modeled by the normal distribution $N(\boldsymbol{q},\sum)$:

$$p(x) \sim \exp\left(-\frac{(x-\boldsymbol{q})^T \Sigma^{-1}(x-\boldsymbol{q})}{2}\right) \tag{2.13}$$

Unlike to occupancy grid that represents the probability of a cell being occupied, the NDT represents the probability of measuring a point for each position within the cell. NDT proposes a cell size of 1000 mm by 1000 mm and this value will be adopted in this work.

To minimize discretization effects, NDT uses four overlapping grids as follows: one grid with side length $\boldsymbol{l}$ of a single cell is place first, then a second one, shifted by half cell horizontally, a third one, shifted by half vertically and finally a fourth one, shifted by half horizontally and vertically. In this way, each 2D point falls into four cells. Thus, if the probability density of a point is calculated the densities of all four cells are evaluated and the result is summed up.

Figure 2.8 shows an example laser scan and a visualization of the resulting NDT. This visualization is created by evaluating a fine mesh of points; bright areas indicate high probability of being occupied.

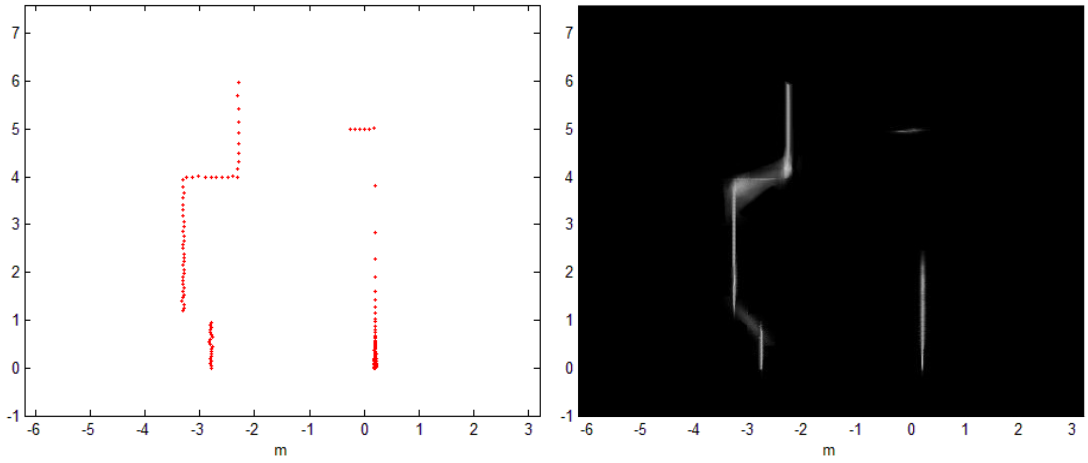

Figure 2.8: An example of NTD: the original laser scan (left) and the resulting probability density (right).

The spatial transformation T between two robot positions is given by:

$$T:\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} \tag{2.14}$$

where t_x and t_y describes the translation and ϕ the rotation between the two positions. As described in Section 2.3 the goal of the scan matching is to recover these values using the laser scans taken at two positions. The outline of NTD, given two scans, is as follows:

1. first, the NDT of the first is built;
2. a estimate for the variables (t_x ,t_y ,ϕ), is initialized (by zero or by using odometry data);
3. for each point of the second scan: a reconstructed 2D point into the coordinate frame of the first scan is mapped, according to the value of variables;
4. the corresponding normal distribution for each mapped point is determined;
5. the score for the variables is determined by evaluating the distribution for each mapped point and summing the result;
6. a new estimate for variables are calculated by trying to optimize the score, this is done by performing one step of Newton's Algorithm, and
7. go to step 3 until a convergence criterion is met.

The steps one to four are straightforward. The remaining is described using the following notation:

- $\boldsymbol{p} = (t_x, t_y, \phi)^T$: the vector of the variables to estimate.
- x_i : the reconstructed 2D point of laser scan point i of the second scan in the coordinate frame of the second scan.

- x_i': the point x_i mapped into the coordinate frame of the first scan according to the vector $\boldsymbol{p}$, that is $x_i' = T(x_i, \boldsymbol{p})$

- Σ_i, $\boldsymbol{q_i}$: the covariance matrix and the mean of the corresponding normal distribution to point x_i' looked up in the NDT of the first scan.

"The mapping according to $\boldsymbol{p}$ could be considered optimal, if the sum evaluating the normal distribution of all points x_i' with parameters Σ_i and $\boldsymbol{q_i}$ is a maximum" [9]. NDT calls this sum the score of $\boldsymbol{p}$, defined as:

$$score(\boldsymbol{p}) = \sum_i \exp\left(-\frac{(x_i' - q_i)^{\mathrm{T}} \Sigma_i^{-1} (x_i' - q_i)}{2}\right) \tag{2.15}$$

NDT normalization problems are described as minimization problems, thus NDT adopts its notation to this convention. Therefore the function to minimize is the negative of *score*.

NDT uses Newton's algorithm iteratively to find the vector $\boldsymbol{p} = (t_x, t_y, \phi)^T$ that minimizes the function $\boldsymbol{f} = -score$. Each iteration solves the equation:

$$\mathbf{H}\Delta\boldsymbol{p} = -\boldsymbol{g} \tag{2.16}$$

where $\boldsymbol{g}$ is the transposed gradient of $\boldsymbol{f}$ with entries

$$g_i = \frac{\partial f}{\partial p_i} \tag{2.17}$$

and $\mathbf{H}$ is the Hessian of $\boldsymbol{f}$ with entries

$$H_{ij} = \frac{\partial f}{\partial p_i \partial p_j} \tag{2.18}$$

The solution of this linear system is an increment $\Delta\boldsymbol{p}$ which is added to the current estimate:

$$\boldsymbol{p} \leftarrow \boldsymbol{p} + \Delta\boldsymbol{p} \tag{2.19}$$

2.4. Genetic Algorithms

The Genetic Algorithm (GA) is a search heuristic that imitates the process of natural evolution; this heuristic is routinely used to generate useful solutions to optimization and search problems. Problem solving using genetic algorithms isn´t new, the pioneering work of J. H. Holland in the 1970's [15] showed significant contribution for engineering applications.

GA´s are inspired by a biological process in which best individuals are likely to be the winners in a competing environment. The potential solution of a problem is an individual which can be represented by a set of variables. These variables are considered as the genes of a chromosome and they are usually structured by a sequence of bits. A positive value (known as *fitness value*), obtained by a *Fitness Function*, reflects the degree of "quality" of the chromosome in order to solve the problem, and this value is narrowly related to its *objective value*.

In the process of a genetic evolution, a chromosome with high quality has the tendency to produce good-quality offsprings, which means better solutions to the problem. "In a practical application of GA, a population pool of chromosomes has to be installed, which can be randomly set initially" [16]. In each cycle of genetic process, a subsequent generation is created from the best chromosomes in the current population. This group of chromosomes, generally called "parents", are selected via a specific selection routine. The roulette wheel selection [17] is one of the most commonly used techniques to provide selection mechanism; this selection is based on the fitness value of chromosomes.

The parents are mixed and recombined to produce offsprings for the next generation. From this process of evolution, it is expected that the best chromosomes will create more offsprings, and thus having a higher probability of surviving in the subsequent generation. This emulates the survival-of-the-fittest mechanism in nature. The evolution cycle is repeated until a desired termination criterion is reached. The criterion used could be the number of evolution cycles, the amount of variation of individuals between different generations, or a

predefined fitness value. In order to achieve a GA evolution cycle, two fundamental operators, crossover and mutation, are required.

The procedure described above can be applied in many different ways to solve a wide range of problems.

However, in the design of a GA to solve a specific problem, there are always two major decisions: specifying the mapping between the chromosome structure and candidate solutions (representation problem) and defining a concrete fitness function.

2.4.1. Chromosome Representation

"Bit-string encoding is the most classical approach used by GA researchers because of its simplicity and traceability" [16]. A slight modification is the use of Gray code in the binary coding; "in practice, Gray-coded representation if often more successful for multi-variable function optimization applications" [18].

Real-valued chromosomes were introduced to deal with real variable problems. "Many works indicate that the floating point representation would be faster in computation" [16].

2.4.2. The Fitness Function

"The Fitness Function is at the heart of an evolutionary computing application" [19]. It determines which solutions within a population are better at solving the particular problem [19], being an important link between GA and the system. The Fitness Function takes a chromosome as an input and outputs a number which represents the measure of the chromosome performance.

An ideal fitness function correlates closely with the algorithm goal, and besides may be computed quickly. Speed of execution is very important, thus, a

typical GA must be iterated many, many times, in order to produce a usable result for a non-trivial problem.

Definition of the *Fitness Function* is not straightforward in many cases, and it is often performed iteratively if the solutions produced by GA are not what it is desired.

2.4.3. Fundamental Operators

The crossover operator is shown in Figure 2.9. The portion of the two chromosomes beyond the crossover point to the right is exchanged to form the offspring. An operation rate (p_c) with a typical value between 0.6 - 1.0 is normally used as the probability of crossover.

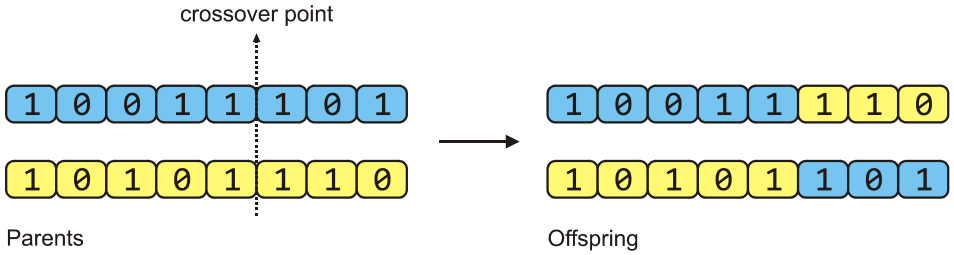

Figure 2.9: The crossover operator

Although one-point crossover was inspired by biological processes, it has one major drawback in the certain combination of *schema* (encoded form of the chromosome): sets of strings that have one or more features in common cannot be combined in some situations. “A multipoint crossover can be introduced to overcome this problem” [16]. As a result, the generating offspring performance is much improved.

The mutation operator, on the other hand, is applied to each offspring individually after the crossover exercise. Figure 2.10 shows the mutation process. It commutes each bit randomly with a probability p_m with a typical value of less than 0.1 [16].

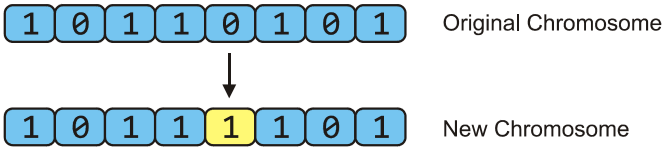

Figure 2.10: The mutation operator

The choice value of p_m and p_c can be a complex, nonlinear operation problem; furthermore, their settings are critically dependent upon the nature of the fitness function [16].

2.4.4. Genetic Algorithms to Solve Problems

Arguably the most obvious application of GA is the multi-variable function optimization. By searching for some optimal value, many problems can be formulated; where the value is a complicated function of its input parameters. In some cases, the interest is on variable settings that lead to the greatest value of the function. In other cases, the exact optimum is not required, just a near optimum, or inclusive a value that represents an improvement over the previously best known value [18].

2.4.5. Differential Evolution

Differential Evolution (DE), like GA, owned to the family of Evolutionary Computation. It is an optimization technique that uses an exceptionally simple evolution strategy, being significantly faster and robust at numerical optimization. It is more likely to find a function's true global optimum.

"DE uses real coding of floating point numbers" [20], and the population is represented by *NP* individuals, where an individual is formed by a vector of *D* real variables, where D is the number of problem's variables.

DE uses both, crossover and mutation operators. However, both operations are redefined in its context. DE creates a vector x_c', a mutated form of any individual x_c (an individual randomly picked from the initial population *NP*), using the vector difference between two other randomly picked individuals x_a and x_b such that: $x_c' = x_c + F(x_a - x_b)$, where *F* is an user-supplied scaling factor. The optimal value of *F* for most functions lies in the range of 0.4 to 1.0 [21]. This operation is known as *mutating with vector differentials*.

After that, the crossover is applied between any individual member of the population x_i and the mutated vector x_c', by swapping the vector elements in the corresponding locations. Like GA, this is also done probabilistically, and the decision of performing (or not performing) crossover is determined by a crossover constant *CR* in the range 0 to 1.

The new vector x_t produced is known as the *trial vector*. "Thus, the trial vector is the child of two parents, a noisy random vector x_c' and the target vector x_i, against which it must compete" [20]. *CR* represents the probability that the child vector inherits the parameter values from the noisy random vector x_c'. When $CR = 1$, for example, every trial vector parameter comes from x_c'. If $CR = 0$, all but one trial vector parameter comes from the target vector x_t. To ensure that x_t differs from x_i by at least one parameter, the final trial vector parameter always comes from the noisy random vector, even when $CR = 0$, so that it does not become an exact replica of the original parent vector. Thus, the trial vector is allowed to pass on the next generation if and only if, its fitness is higher than that of its parent vector x_i, otherwise the parent vector yields to the next generation. Figure 2.11 shows the process of DE.

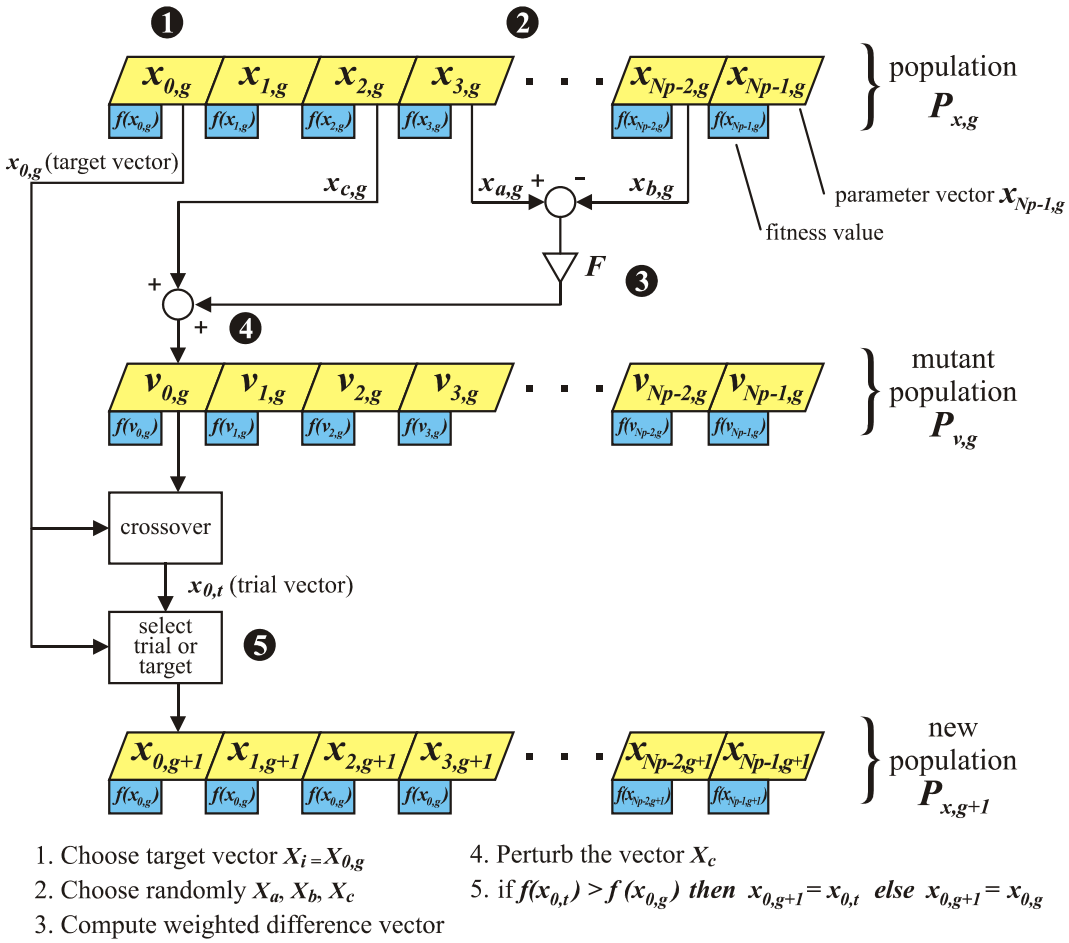

Figure 2.11: Differential Evolution Process [22]

Among all, just three factors control evolution under DE: the population size *NP*, the weight *F* applied to the random differential, and the crossover constant *CR*.

2.4.6. Different Strategies of DE

Depending on the type of problem, different strategies can be adopted in the DE algorithm. "The strategies can vary based on the vector to be perturbed, number of difference vectors considered for perturbation, and finally the type of crossover used" [20]. The following are the 10 different working strategies proposed by Price and Storn [22].

1. DE/best/1/exp

2. DE/rand/1/exp
3. DE/rand-to-best/1/exp
4. DE/best/2/exp
5. DE/rand/2/exp
6. DE/best/1/bin
7. DE/rand/1/bin
8. DE/rand-to-best/1/bin
9. DE/best/2/bin
10. DE/rand/2/bin

The convention used above is DE*/x/y/z*. DE means Differential Evolution, *x* denotes a string representing the vector to be perturbed, *y* is the number of difference vectors used for perturbation of *x*, and *z* denotes the type of crossover being used (exp: exponential, bin: binomial).

For perturbation with a single vector difference, out of the three distinct randomly chosen vectors, the weighted vector differential of any two vectors is added to the third one. In the same way for perturbation with two vector differences, five distinct vectors, other than the target vector, are chosen randomly from the current population. Out of these, the weighted vector difference of each pair of any four vectors is added to the fifth one for perturbation.

In exponential crossover, the crossover is performed on the *D* variables in one loop until it is within the *CR* bound. In binomial crossover, the crossover is performed on each of the *D* variables whenever a randomly picked number between 0 and 1 is within the *CR* value. So, for high values of *CR*, the exponential and binomial crossovers yield similar results. In the binomial case, the last variable always comes from a random noisy vector to ensure that is different from the target vector, and hence the above procedure is applied up to $D - 1$ variables.

"The strategy to be adopted for each problem is to be determined separately by trial and error" [20]. The best strategy for a given problem may not work well when applied to a different problem.

In the next chapter, the presented analytical background is applied to the SLAM problem.