THESIS FOR THE DEGREE OF MASTER OF SCIENCE

AUTONOMOUS ROBOT NAVIGATION USING THE UTILITY FUNCTION METHOD AND MICROSOFT KINECT

TOBIAS ERIKSSON AND ERIK RAGNERIUS

Department of Applied Mechanics
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden, 2012

Autonomous robot navigation using the utility function method and Microsoft Kinect

TOBIAS ERIKSSON AND ERIK RAGNERIUS

© TOBIAS ERIKSSON AND ERIK RAGNERIUS, 2012

Master thesis 2012:45
ISSN 1652-8557

Department of Applied Mechanics
Adaptive Systems
Chalmers University of Technology
SE–412 96 Göteborg
Sweden
Telephone: +46 (0)31–772 1000

Chalmers Reproservice
Göteborg, Sweden, 2012

Autonomous robot navigation using the utility function method and Microsoft Kinect

TOBIAS ERIKSSON AND ERIK RAGNERIUS
Department of Applied Mechanics
Chalmers University of Technology

Abstract

In this thesis, a system for autonomous navigation using the Microsoft Kinect sensor and the utility function (UF) method for decision-making, has been developed. In the UF method an artificial brain decides what control system procedures to activate or deactivate, based on their utility. The system uses the Kinect sensor for obstacle avoidance and localization. The Kinect sensor readings are matched to a predefined map, using a scan matching algorithm based on the Hough transform, in order to correct pose estimates acquired through odometry. The A*-algorithm is used for path planning and the algorithm is applied on a grid representing the operating area accessible to the robot.

The results show that the UF method can be applied in systems for autonomous navigation, using the Kinect sensor for localization and obstacle avoidance. The Kinect sensor has proven to be a useful alternative to more expensive range sensors, despite its limitations in terms of field of view, range, and accuracy. On the other hand, the results indicate that improvements are needed, in order to improve the robustness of the system. During testing, the robot completed on average 87.6 % of a 45.1 m long, predefined course and on average 121 m when choosing random targets. Tuning the parameters of the decision-making system and improving the maneuverability at low speeds are examples of future work that could increase the performance of the navigation system.

Key words: Localization, navigation, path planning, Kinect, Hough scan matching, utility function method.

Table of Contents

Chapter 1

Introduction

1.1 Background

In many lines of business high efficiency and quality demands have forced the production to become partly or fully automated. This thesis is focused on the development of a navigation control system for a small autonomous robot working in a typical indoor office environment. The control system will include methods for localization, path planning, and decision-making. The robot will operate in an office and it constitutes a step towards the development of fully autonomous construction machinery. In addition, the robot will be used as a platform for evaluation and development of future functions related to autonomous vehicles.

Even though the conditions vary greatly between an indoor environment and a construction site, many of the fundamental features of a control system should be similar. The main differences are the hardware available, the safety and robustness requirements and the properties of the operating environment available for e.g. localization.

1.2 Autonomous navigation

For any mobile, autonomous agent, navigation is of great importance since virtually every task requires the agent to travel between different positions. In order to navigate safely and efficiently, the agent must be able to localize itself and to plan its future movements. Humans and animals [37], as well as robots [33, 14], commonly perform the localization by estimating the path traveled and by comparing the surroundings to known landmarks.

Since mobile robots are usually built with wheels, the core of most localization algorithms is odometry, in which wheel encoder measurements are used to calculate an estimate of the pose and velocity. Measurement noise, mechanical imperfections and model simplifications will, however, result in erroneous position estimates and

unless the estimates are corrected by an odometry-independent source, the error will tend to grow.

As the operating area of the agent becomes larger, the area that needs to be stored in memory in order to navigate grows and the need for some kind of map arises. If a mobile robot is to navigate autonomously in an unknown environment it must therefore be able to carry out some type of mapping in addition to localization and path planning. **Simultaneous localization and mapping** (SLAM) is, as Leonard and Durrant-Whyte [23] stated, "*a question of which came first, the chicken or the egg*?" since localization requires an accurate map and mapping requires an accurate estimate of the position. The problem has been the focus of many papers, such as [16, 25, 30], and many techniques for solving it have been proposed.

The robot considered in this thesis faces the somewhat easier task of navigating in a known environment, i.e. given a map a priori. In many cases, the assumption of knowing the surroundings is valid since maps or blueprints are often easy to obtain for indoor environments (excluding, of course, movable objects, such as furniture). The major simplification of this problem, as compared to the SLAM problem, is that as long as the robot is able to reliably match a scan of the surroundings to the map, an accurate estimate of the position can be found. In addition, no exploration phase (a by no means trivial task) is needed, the robot can instead directly focus on its primary tasks.

1.2.1 Localization through scan matching

Once a map of the environment has been acquired, it can be used for localization by comparing it to the surroundings. **Scan matching** is a commonly used technique for localization. The idea of scan matching is to find a transformation, i.e. a rotation and a translation, that will minimize the difference between a current scan of the surroundings and a reference scan (e.g. a part of the map). If a suitable transformation is found, it can be used to correct the estimated position.

The most common scan matching methods can roughly be divided into **point-based** methods [3, 12, 24, 29], **feature-based** methods [27, 41], and **correlation-based** methods [10, 38]. The point-based methods compare the raw data points of a scan to the reference and by minimizing some cost function, the transformation is found. Feature-based methods instead rely on extracting features such as lines, corners, or planes from the scans and then comparing these features. The feature extraction phase of course requires additional computations. If the extraction is done efficiently, however, the total computational time might be reduced, since the number of features is usually a lot smaller than the number of points. Correlation-based methods try to utilize the angular information of the points in polar coordinates by correlating spectra or histograms in order to find the poses with maximum correlation.

In [12], Cox presents a scan matching technique based on minimizing the distance

between scan data points and the line segments used to represent obstacles in the map. Besl and McKay use a somewhat similar method in [3] but their algorithm is based on the **iterative closest point** (ICP) algorithm. The method iteratively finds pairs of closest points from the scan and reference and then tries to find a transformation that minimizes the error between the sets of pairs. Lu and Milios [24] created a faster version of the algorithm by using the fact that if the translation is small, a pair of corresponding points can be found by comparing only the distances to the origin of points with polar angles within a certain range.

Sandberg *et al.* [29] use a technique which compares the current scan of a **laser range finder** (LRF) to a **virtual LRF scan** (vLRF). The method continuously places the vLRF in different positions close to the best position estimate and simulates an LRF scan. If the match is better than for previous vLRF scans the position is stored as a new best position estimate.

Many of the papers describing feature-based localization algorithms are probabilistic and rely on Kalman filtering but there are also feature-based scan matching algorithms. In [27], Pradalier and Sekhavat compare triangles of identified landmarks in order to accurately match scan and reference. Zezhong *et al.* [41] instead focus on matching line segments from the map to line segments extracted from the scan. By comparing the lengths of the line segments, different position candidates are found and evaluated.

In [38], Weiss *et al.* try to find the transformation by maximizing the cross correlation between scan and reference. The method relies on finding the angles of vectors connecting consecutive points and summing them up in **angle histograms**, since these are roughly invariant to translation [38]. The angle histograms for scan and reference are cross-correlated in order to find a rotation estimate. The translation is then found in the same manner, by correlation of x and y histograms respectively.

In 2005, Censi *et al.* [10] presented a method which is similar to the one described by Weiss *et al.* but uses the **Hough transform** for scan matching. They introduced the **Hough spectrum** which resembles the histograms in [38] and these spectra are correlated to find the transformation. Since the spectra are not based on consecutive points, they are not as sensitive to noise as the angle histograms.

1.2.2 Path planning

The fundamental problem of path planning is to produce a collision free path from a given starting point A, to a desired end point B. Along with localization and mapping, this is a central part in the design of a system for autonomous navigation.

Several methods for navigating in a known environment have been presented and depending on how the operating area is defined, there are different ways to solve the path planning problem. The most common approaches to navigation are using **grid-based navigation methods** [33, 34, 39, 40] and **potential field methods** [22, 35].

In the grid-based methods, the operating area is described as some sort of grid, with nodes and edges connecting the nodes. The grid may have arbitrary shape but, for the sake of convenience, it is represented as a set of rectangular cells in this thesis. At each grid cell (where a node is the center point), the autonomous agent is allowed to move to successive cells, as long as the line connecting them is free from obstacles. Numerous graph and tree search algorithms solving the problem of path planning in a grid-based representation have been presented, e.g. **best-first search** (BFS), **Dijkstra's algorithm**, and the **A*-algorithm**.

In the BFS algorithm, presented by Pearl [26], the robot finds its way through the grid by estimating the cost of going from the neighboring nodes to the target using a heuristic cost function, typically the euclidean distance. The node associated with lowest cost is chosen as the next node and the previous step is repeated until the goal is reached.

Dijkstra's algorithm, presented in [15], also requires a grid and, to determine which path to choose, it repeatedly evaluates the euclidean distance between the current node and the neighboring nodes. The algorithm then expands outwards from the the starting node until the target is found. The path can then be found by going backwards through the parenting nodes, following the path associated with the lowest accumulated cost.

In [34], Wang *et al.* use Djikstra's method to find the shortest path for their maze robot in a two dimensional grid map with only static obstacles. Wahde *et al.* [33] use a convex navigation grid and generate a path using Dijkstra's algorithm. In order to solve the problem of moving obstacles, the path planning procedure is updated frequently and a penalty is added for the paths where an obstacle is detected, making such paths unusable.

The A^*-algorithm, first introduced by Hart *et al.* [21], is a combination of the BFS and Dijkstra's algorithms. The algorithm is described in more detail in Section 3.2.2, and has been widely used in robotic path planning [39, 40].

In potential field methods for navigation, introduced by Khatib [22], there is no need to describe the operating area as a grid, unlike in the grid-based navigation methods. In methods based on potential fields, the robot can be seen as a ball moving on a surface under the influence of gravity. The goal is to create a field with a gradient which is attractive toward the desired position (global minimum) and repellent from the obstacles (local maxima) in the arena.

1.2.3 Obstacle detection and avoidance

Once a mobile agent can autonomously navigate, a known or unknown environment, by successfully localizing itself and planning paths, the matter of doing so *safely* remains. When navigating in a dynamic environment, this means detecting and avoiding objects not included in the map, such as people or movable furniture. The first problem is to detect an obstacle and some early methods [5, 13] try to detect edges of obstacles,

while others [6, 7, 32], are based on an evenly spaced grid where the probability of occupancy is estimated for each cell. Another kind of method for avoiding obstacles is based on the idea of **tentacles** sensing the surroundings for obstacles and it has been used in recent papers [9, 11, 32]. The approach is similar to the **dynamic window** approach, introduced by Fox *et al.* [19].

In [6], Borenstein and Koren combine the concepts of occupancy grids [17] and potential fields [22]. Their method, called **virtual force field** (VFF), use the idea that obstacles exert repulsive forces on the robot, in order to avoid obstacles. Because of shortcomings of the VFF method, they developed a new method, called the **vector field histogram** (VFH), which maps the grid based histogram onto a polar coordinate histogram. Directions free from obstacles are then found from the polar histogram.

Wahde *et al.* [33] use a method where a scan of the environment is compared to the expected result of the scan. If a set of measurement points deviate too much from the expected result, they are added to a list of potential obstacles. Depending on where the obstacle is found, different measures are taken to avoid it.

Buschmann *et al.* [9], Cherubini *et al.* [11], and von Hundelshausen *et al.* [32] all use the tentacles approach but in slightly different ways. They all use circular arcs as tentacles examining the surroundings but in [11] and [32] the tentacles are evaluated by comparison to a local map. In [9] the tentacles are evaluated by direct comparison to camera images instead. The evaluation criteria also differ and a notable difference is that von Hundelshausen *et al.* include the desired heading in their evaluation, while the other methods are purely reactive, choosing the path associated with the least direct risk.

1.3 Decision-making

In order for an autonomous agent to be able to complete a set of predefined tasks, e.g. navigate in an indoor office environment while avoiding collisions with obstacles, the robot needs a system for decision-making. In the literature of autonomous robots, there are two main approaches for designing the control architecture, the classical AI approach [31] and the behavior-based decomposition [2, 8, 33, 36].

In classical AI, the decision-making is designed in a **sense-model-plan-act** (SMPA) framework. From the sensor inputs a (typically rather complex) world model is created and based on deliberations carried out within this model, a decision is reached and executed. A disadvantage is that if the model is complex, this method will be very time consuming.

Brooks [8] introduced the idea of a behavior-based decomposition, in which the artificial brain of the autonomous agent relies on a set of basic behavior functions. These functions are organized into layers of increasing complexity, e.g. the first layer can be *avoid obstacles* followed by *explore environment* and *build map* etc. All layers run in

parallel, accessing sensor data and generating actions, and the higher levels subsume the actions of the lower ones. In [2], Arkin presented a motor-scheme-based architecture. Two different types of schemes are created, perceptual and motor schemes, and the interaction between these behaviors (schemes) is the base for the decision-making.

Sakagami *et al.* [28] used a behavior-based architecture relying on agents (symbolizing different behavior modes), in a deliberative and a reactive layer, for an autonomous robot. Each agent in these layers runs simultaneously, based on input from the sensors, and once a condition is fulfilled, the agent will try to perform its corresponding task.

Watanabe *et al.* [36] presented a similar approach, but in addition to Sakagami *et al.*'s [28] method, their agents are clustered into three groups (reflexive, purposive and adaptive agents) handling different tasks, e.g. agents in the purposive group handles detection- and avoidance of obstacles. The decisions are taken by the motion executor, a process getting information from all three groups. This method was introduced in order to increase the robustness in the decision-making procedure.

Wahde *et al.* [33] proposed a system for decision-making based on **utility**, i.e. usefulness to the system. Control system processes, here called **brain processes**, for the different activities are generated and divided into cognitive and motor behaviors. Each brain process is then related to a utility function and the brain processes having a positive utility are the ones chosen to be active, with the exception that only one motor behavior can be active at a time, the one with highest utility.

Chapter 2

Hardware

2.1 The Microsoft Kinect sensor

Commonly used sensors for distance measuring within the field of robotics are IR sensors, ultrasonic sensors, and laser range finders. Since both IR and ultrasonic sensors are limited in terms of range and accuracy, LRFs are used for localization and mapping in many applications [29, 41]. LRFs are usually very accurate and have a wide field of view but they are much more expensive (typically at least 1000 USD) than IR or ultrasonic sensors. A range sensing technology, which has become popular in robot navigation applications, is the Microsoft Kinect[1] sensor [1]. The sensor was released in 2010 and was originally produced as a device for hands-free video gaming. The characteristics and the low price (around 100 USD) of the sensor have since then attracted the attention of researchers of various fields, e.g. in the fields of mapping, mobile robot navigation, and 3D modelling.

In this thesis the Kinect sensor is used as depth measuring device, providing the eyes to the robot, during localization and obstacle avoidance. The Kinect sensor consists of an IR light source, an IR camera, an RGB camera, a multi-array microphone, and an electrical motor, providing the tilt function to the sensor. In Table 2.1, the product specification of the Kinect sensor is presented.

[1] Kinect is trademark of the Microsoft Corporation.

Table 2.1: *The Kinect product specification*

Connectivity	USB 2.0
Resolution RGB camera	640 $\times$ 480 pixels
Resolution IR depth camera	640 $\times$ 480 pixels
Frame rate	30 Hz
Range:	0.7-6 m
Tilt range	54°
Field of view	Horizontal 57°, Vertical 43°

2.1.1 Functionality

The depth sensor technology used in the Kinect sensor is handled by the PS1080 **System-on-a-Chip** (SoC) and was developed in 1997 by *PrimeSense*. The depth data from the Kinect sensor comes in the form of a **depth frame**, i.e. an i by j matrix filled with depth values. It is acquired through a technique called **light coding**. The common way of finding a range is using the **time-of-flight** (TOF) technology, where the distance is found by measuring the time for a beam to travel from the source to reflection and back. Instead of using the TOF approach, the light coding technology used in the Kinect sensor works in an entirely different way.

The Kinect sensor is equipped with an IR light source, constantly emitting light. This light is scattered in a pattern of small dots, projected on the environment ahead of the sensor. The pattern is detected by the IR camera and since the lens distortion and the distance between emitter and receiver is known, the distance to each dot can be calculated by a triangulation process inside the PS1080. This triangulation process is described in more detail in [20]. After the PS1080 has been processing the IR data, a depth frame describing the environment can be obtained.

2.2 Robot

The robot used in this thesis is two-wheeled, differentially steered, and was developed as a bachelor thesis at Chalmers University of Technology [4]. The base of the robot is almost rectangular, with length 0.51 m and width 0.45 m. The height is 0.85 m and the weight approximately 23 kg. The height (from the ground) to the kinect sensor is 0.85 m. The robot is equipped with two batteries, each 30 Ah 12 V, supplying the robot and the Kinect sensor with power. Besides the Kinect sensor, the robot is also equipped with range sensors. Four analogous SHARP IR sensors, GP2Y0A21YK, two placed in the robot's direction of heading and the other two pointing in the opposite direction. In addition, the robot is equipped with two digital ultrasonic Maxsonar-

EZ1 from MaxBotix pointing forward. The IR and ultrasonic sensors are used for emergency situations only, bypassing the navigation system by braking the motors in case the risk of collision is imminent.

The robot is equipped with four microcontrollers. Two of the microcontrollers read wheel encoder values and control the rotational speed of the two motors. The third microcontroller controls the power supply and monitors the charging of the batteries. The fourth microcontroller handles communication between the other controllers as well as processing of sensor input from the IR and ultrasonic sensors. It also handles the communication between the robot and the laptop running the navigation system. The robot and the Kinect sensor are connected via USB to the laptop placed on top of the robot.

Chapter 3

Method

This chapter aims at describing the methods used to solve the tasks outlined in the introduction, namely, *localization*, *path planning*, *obstacle detection and avoidance*, and *decision-making*.

3.1 Localization

3.1.1 Odometry

The core of the localization task is odometry, in which the distance each wheel travels is calculated from wheel encoder measurements. The simple motion model used in this thesis is described in [14]. At each time t the distance traveled since t - $\mathrm{d}t$ and the change in heading is approximated as

$$\delta d = \frac{d_\mathrm{r} + d_\mathrm{l}}{2} \tag{3.1}$$

$$\delta\theta = \frac{d_\mathrm{r} - d_\mathrm{l}}{b} \tag{3.2}$$

where d_r and d_l are the distances travelled by the right and left wheel respectively, since the last sample, and b is the wheel base. Using these approximations, the estimate of the robot's pose can be updated according to

$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} x_{t-\mathrm{d}t} \\ y_{t-\mathrm{d}t} \\ \theta_{t-\mathrm{d}t} \end{bmatrix} + \begin{bmatrix} \delta d \cos(\theta_{t-\mathrm{d}t} + \frac{\delta\theta}{2}) \\ \delta d \sin(\theta_{t-\mathrm{d}t} + \frac{\delta\theta}{2}) \\ \delta\theta \end{bmatrix} \tag{3.3}$$

The error between the actual pose and the estimated pose will, however, generally tend to grow over time due to mechanical imperfections and modelling errors. This phenomenon is known as **odometric drift** and the pose estimate will eventually need

to be corrected by some function independent of odometry, such as a scan matching algorithm.

3.1.2 Processing of Kinect data

In order to correct the pose estimates through scan matching, a suitable representation of the surroundings is needed. Many algorithms therefore require some preprocessing of the raw data. The Kinect data comes in the form of a depth frame and the first preprocessing step consists of creating a **point cloud**. In the point cloud, the depth values have been transformed into points in a 3D-coordinate system originating at the center of the Kinect. The z-coordinate is extracted directly from the depth frame,

$$z_{ij} = d_{ij} \tag{3.4}$$

where d_{ij} is the depth value at the i^{th} row and j^{th} column of the depth frame. The x and y-coordinates are calculated as linear functions of the z-coordinate and the frame indices,

$$x_{ij} = \left(i - \frac{R_{\text{h}}}{2}\right) z_{ij} f \tag{3.5}$$

$$y_{ij} = \left(\frac{R_{\text{w}}}{2} - j\right) z_{ij} f \tag{3.6}$$

where $R_{\text{w}} \times R_{\text{h}}$ is the size of the depth frame in pixels and f is a scaling factor found by calibration to be 0.0017. An example of a point cloud created by these equations is shown in Figure 3.1.

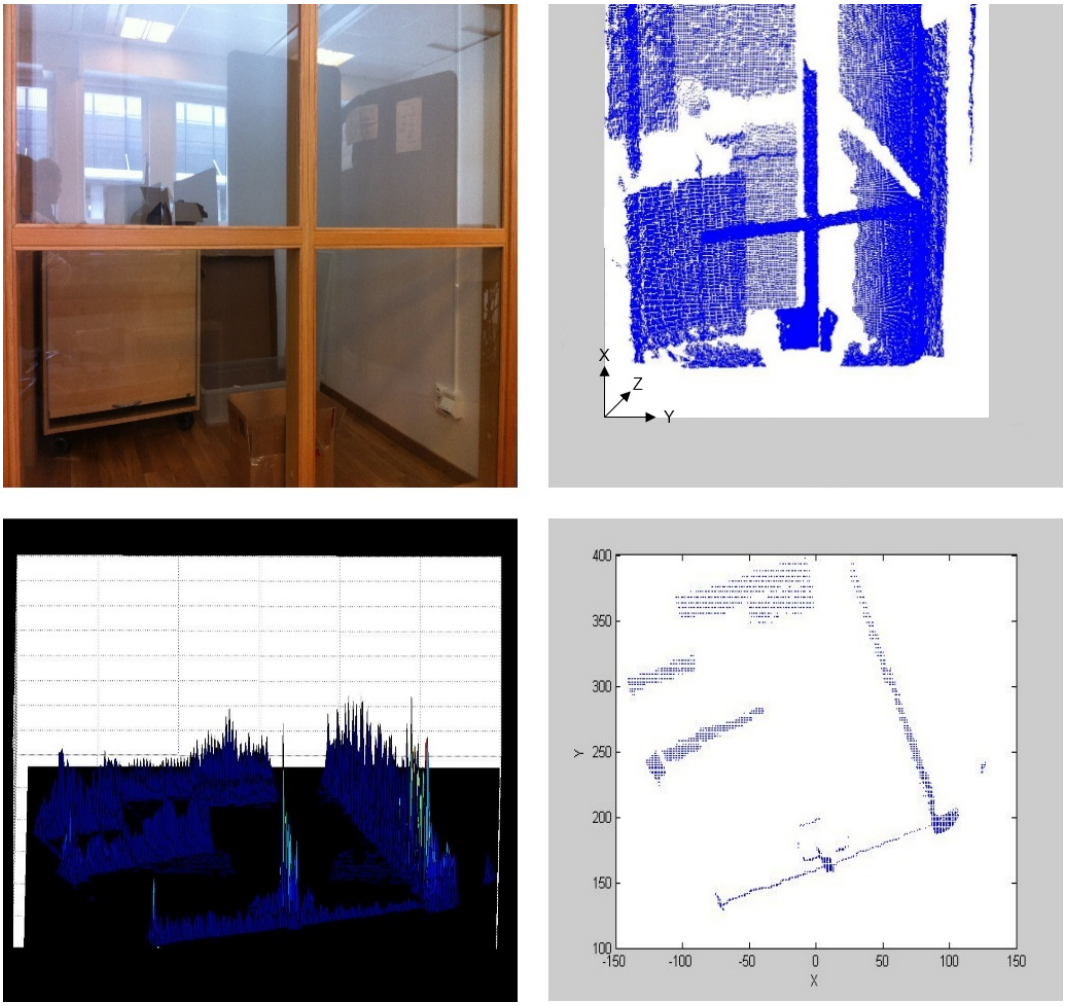

Figure 3.1: *Upper left panel: Picture of a room at the CPAC office. Upper right panel: A point cloud of the same room, created from a depth frame from the Kinect sensor. Lower left panel: The histogram created from the point cloud. The walls and larger pieces of furniture result in high peaks. Lower right panel: The Kinect scan created from the histogram. This kind of 2D-representation of the environment is used for the scan matching.*

Once the point cloud has been created, some scan matching algorithms (such as the ICP algorithm) could be applied directly on the 3D-points. Matching up to 307200 (640×480) points every frame is, however, computationally heavy. The approach in this thesis therefore requires additional preprocessing, before the scan matching is applied.

The second preprocessing step transforms the 3D-point cloud into a set of 2D-points at the edges of obstacles (referred to as a **Kinect scan**). To create the Kinect scan from the point cloud, a 2D-**histogram grid** is used. In the histogram all points in 3D-

space lying within a certain square cell of the YZ-plane are summed up. This means that objects with clear edges along the X-axis (e.g. walls or furniture) will result in high peaks in the histogram. If only cells with a sum above some threshold are considered occupied, a set consisting of 2D-points with (y, z)-coordinates can be extracted, see Figure 3.1. The histogram grid is set to span a rectangular area of size 5 m $\times$ 4.15 m in front of the robot because it roughly corresponds to the range of the Kinect. The size of each cell is one square centimeter.

The Kinect scan, acquired by the method described above, resembles an LRF scan but with the great advantage of being a 2D-representation of *all* objects within the field of view, not just those present at a certain height. The disadvantage of the Kinect scan, as compared to an LRF scan, is the limited field of view and the poor accuracy. The fact that the Kinect scan is able to include walls, despite obstacles being in the way, makes it very useful in such a cluttered and dynamic environment as an office.

3.1.3 Reference scan

When the Kinect scan has been created, it can be compared to a map, stored in the memory, in order to correct the pose estimate. To make the comparison effective, a local map needs to be extracted from the global map. The method used to extract a local map is the one used in [29], where an LRF scan is simulated. The scan (referred to as the **reference scan**) is, however, simulated to resemble a Kinect scan instead.

3.1.4 Hough Scan Matching

The aim of the scan matching algorithm is to find a correction ($\mathrm{x_c}$, $\mathrm{y_c}$, θ_c) of the pose estimate by matching the Kinect scan to a reference scan. The scan matching technique used is the one introduced in [10], in which the Hough transform is utilized.

In general, the Hough transform is used to map the input space to a parameter space, specifically the 2D Cartesian space to the parameter space representing lines in polar coordinates. This means that each point P_i of the scan is mapped to a set S_i of lines passing through that point. A line can be described in polar coordinates as,

$$x \cos\theta + y \sin\theta = \rho \tag{3.7}$$

where ρ is the perpendicular distance from the origin to the line and θ is the direction of the line's normal, see Figure 3.2. The set S_i is therefore a set of (θ, ρ)-combinations defining lines passing through P_i. In theory, the number of lines passing through a single point is infinite but in practice the **discrete Hough transform** (DHT) is used and the the set S_i is limited by boundaries and by the choice of resolution of the transform. By summing up the sets S_1, S_2,..., S_i corresponding to points P_1, P_2, ..., P_i, the DHT for the entire scan is found. The DHT is calculated in the following way:

1. Choose boundaries and resolution of the transform i.e. $\rho_{\min}$, $\rho_{\max}$, $\Delta\rho$ and $\theta_{\min}$, $\theta_{\max}$, $\Delta\theta$.

2. Create an empty matrix of size M $\times$ N where M is equal to $\frac{\rho_{\max}}{\Delta\rho}$ and N is equal to $\frac{\theta_{\max}}{\Delta\theta}$, i.e. every row corresponds to a certain ρ value and each column to a certain θ value. Repeat 3 and 4 for each point P_i.

3. For each value of θ solve Equation 3.7, with (x, y) being the coordinates of P_i, to get a value of ρ.

4. If ρ is within the boundaries, increase the matrix element corresponding to (ρ, θ) by one. Otherwise, continue with the next θ value.

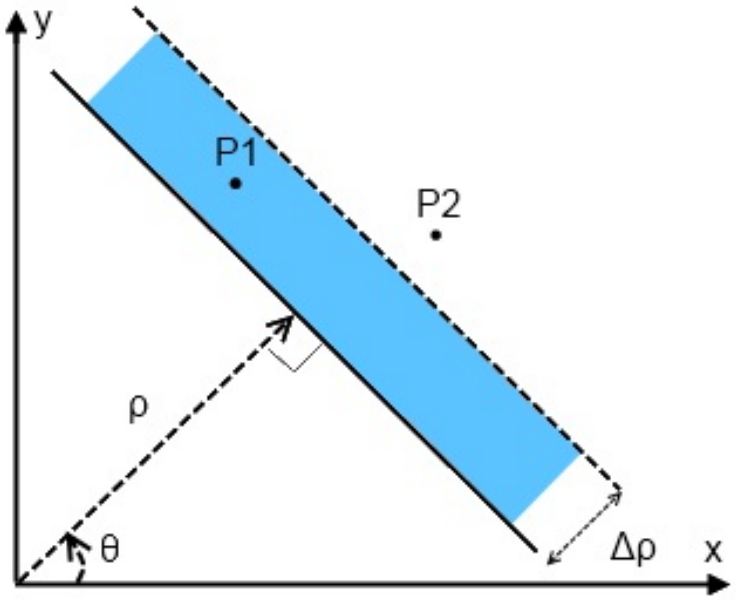

Figure 3.2: *The figure shows how the DHT is calculated. The blue area corresponds to a (θ, ρ) combination and all points (such as P1) inside the area will increase the value of DHT(θ, ρ) by one. The point P2 on the other hand would not increase DHT(θ, ρ) but rather DHT(θ, ρ + $\Delta\rho$).*

The DHT now consists of a matrix filled with values indicating to what extent the scan fits every (θ, ρ)-combination. The scan matching is performed by first finding the rotational displacement between the Kinect scan and the reference scan, ϕ, and then the translation, T = (Δx, Δy), between them[1]. In order to find ϕ, Censi *et al.* [10]

[1]Note that (Δx, Δy) is the translation in the robot's coordinate system and not the global coordinate system.

introduced the Hough spectrum (HS), which is calculated as

$$HS(\theta_i) = \sum_{j=0}^{M} DHT(\rho_j, \theta_i)^2. \tag{3.8}$$

By summing the square of each element along each column of the DHT, HS(θ_i) holds a measure of how many points occur along lines with normal direction θ_i (disregarding at what distance from the origin). An important feature of the HS is that it is invariant to rotations in input space in the sense that the spectrum will only be circularly shifted if the input space is rotated. This fact is exploited in order to find the rotational displacement ϕ. The Hough spectra of the Kinect scan and a reference scan are cross correlated according to

$$HS_{\text{corr}}(\phi) = \sum_{\theta=\theta_{\min}}^{\theta_{\max}} HS_{\text{scan}}(\theta) \cdot HS_{\text{ref}}(\theta - \phi) \tag{3.9}$$

and the circular shift corresponding to the maximum of the correlation is used as a hypothesis for ϕ, see Figure 3.3. In a global search the correlation may have several local maxima and in [10] these maxima are used as ϕ *candidates*. In this application however, ϕ is assumed to be small[2] and the correlation is only calculated for shifts $|\phi| \leq 20°$. Only the ϕ corresponding to the global maximum is used.

[2]The assumption is based on the fact that the pose estimate obtained through odometry and scan matching should be rather accurate (at the very least within $|\phi| \leq 20°$).

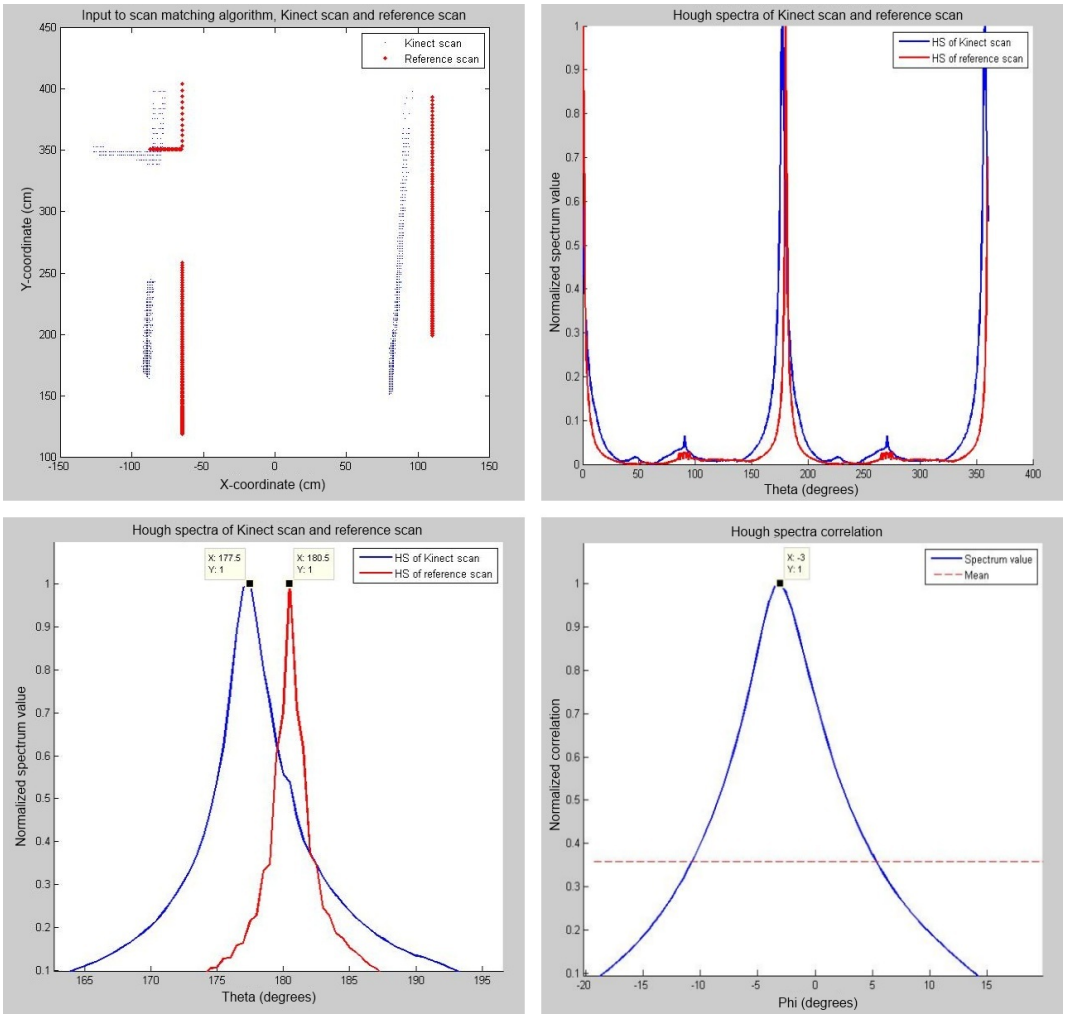

Figure 3.3: *Upper left panel: The input to the scan matching algorithm, a Kinect scan (blue) and a reference scan (red) of a hallway. Upper right panel: The Hough spectra of the Kinect and reference scan. The highest peaks appear at $\approx 0°$ and $\approx 180°$ because most points lie along lines with normals in those directions. Smaller peaks appear at $\approx 90°$ and $\approx 270°$. Lower left panel: An enlarged version of the Hough spectra where the shift between them is clearly visible. Lower right panel: The correlation of the spectra. The maximum value is found at -3° which will be be used as hypothesis for ϕ.*

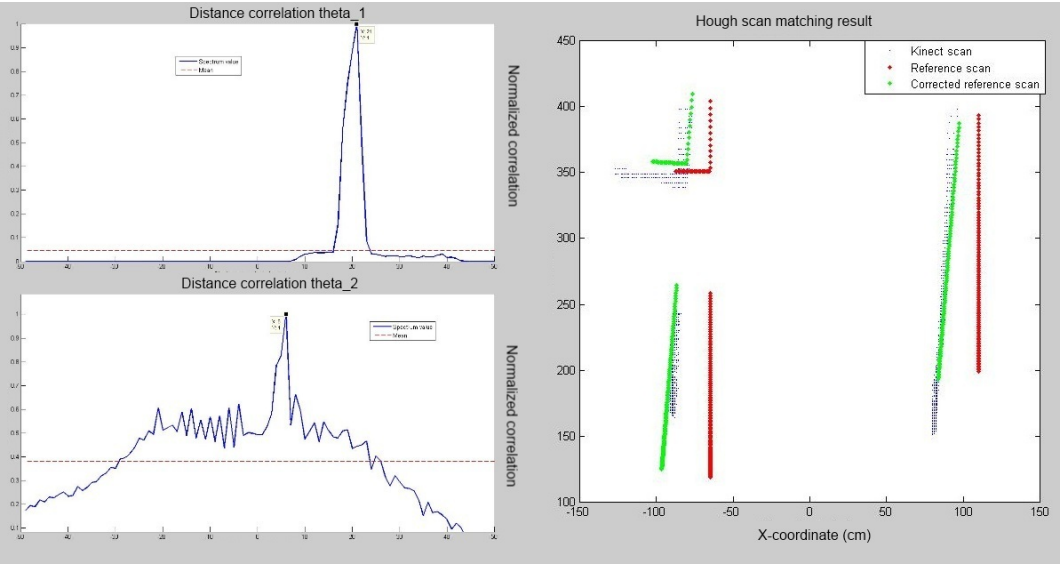

Figure 3.4: *Upper left panel: The correlation of rows corresponding to θ_1. Lower left panel: The correlation of rows corresponding to θ_2. The dashed lines show the mean values in both plots. Right panel: The Kinect scan (blue), the reference scan (red), and the corrected reference scan (green).*

Once ϕ has been found, the columns of the DHT of the reference scan are circularly shifted by ϕ, in order to align the reference scan rotationally to the Kinect scan[3]. The translation is then found in a similar manner as ϕ but instead of correlating the Hough spectra, columns of the DHTs are correlated. This means that the Kinect scan and the reference scan are correlated by shifting ρ along a certain direction θ,

$$\rho_{\text{corr}}(\theta, d) = \sum_{\rho=\rho_{\max}}^{\rho_{\max}} DHT_{\text{scan}}(\rho, \theta) \cdot DHT_{\text{ref}}(\rho - d, \theta). \tag{3.10}$$

By correlating columns of the DHTs corresponding to θ, according to Equation 3.10, the displacement in the direction of θ is found by taking $\text{d}(\theta) = \arg\max_{d} \rho_{\text{corr}}(\theta, \text{d})$.

In order to find T, at least two directions need to be correlated. The directions used are $\theta_1 = \arg\max_{\theta} \text{HS}(\theta)$ and $\theta_2 = \theta_1 + 90°$. $\text{d}_1(\theta_1)$ and $\text{d}_2(\theta_2)$ are then used to calculate the translation

$$T = (\Delta x, \Delta y) = (d_1 \cos\theta_1 + d_2 \cos\theta_2, d_1 \sin\theta_1 + d_2 \sin\theta_2). \tag{3.11}$$

Since the result from the scan matching algorithm is used as a direct correction of the pose estimate, it is crucial to avoid incorrect matchings. A requirement, to even

[3]Remember that a rotation in input space corresponds to a shift of columns in parameter space.

start the scan matching, is therefore that both the Kinect scan and the reference scan contain enough information, i.e. the number of points should exceed a certain value. If the requirement is met and the algorithm is executed, a measure of its accuracy is needed.

As Figures 3.3 and 3.4 show, in an indoor environment the HS will typically have clear peaks when the scan includes walls and the peaks will most likely appear with a 90-degree difference. The correlation of both spectra and DHT columns are normalized and a low correlation mean value indicates that the highest peak is very narrow (and therefore that there is no ambiguity about the result). In order to increase the probability that the scan matching result is accurate, three mean value thresholds are used. The scan matching quality is controlled in the following way:

- If mean ($\mathrm{HS}_{\mathrm{corr}}$) $< \mathrm{T}_1$ the ϕ hypothesis is accepted and the algorithm tries to find the translation, otherwise the result is rejected and the algorithm exits.
- If mean ($\rho_{\mathrm{corr}}(\theta_1)$) $< \mathrm{T}_2$ the translation in the direction of θ_1 is accepted.
- If mean ($\rho_{\mathrm{corr}}(\theta_2)$) $< \mathrm{T}_3$ the translation in the direction of θ_2 is accepted.

The thresholds $\mathrm{T}_1 = 0.5$ and $\mathrm{T}_2 = \mathrm{T}_3 = 0.1$ were chosen empirically after testing and no formal optimization has been carried out. Figure 3.4 shows an example where the translation along θ_1 is accepted but the translation along θ_2 is rejected.

When the scan matching algorithm is completed, it provides a correction (Δx, Δy, ϕ) that will align the Kinect scan with the reference scan. In order to use the result to correct the pose estimate, the corrections ($\mathrm{x_c}$, $\mathrm{y_c}$, θ_c) in the global coordinate system are calculated according to,

$$\theta_\mathrm{c} = -\phi \tag{3.12}$$

$$x_\mathrm{c} = \Delta y \cos(\theta_\mathrm{p} + \theta_\mathrm{c}) + \Delta x \cos(\theta_\mathrm{p} + \theta_\mathrm{c} - 90^\circ) \tag{3.13}$$

$$y_\mathrm{c} = \Delta y \sin(\theta_\mathrm{p} + \theta_\mathrm{c}) + \Delta x \sin(\theta_\mathrm{p} + \theta_\mathrm{c} - 90^\circ). \tag{3.14}$$

where θ_p is the most recent estimation of the heading. The pose estimate is then corrected by simply adding these corrections to the pose estimate.

3.2 Path planning

In this thesis the heuristic A*-algorithm, described in [21], is used to find the optimal path between the starting and the desired position. The operating area, a predefined map given a priori, is described with the grid-based approach, i.e. covered with rectangular cells, see the left panel of Figure 3.5.

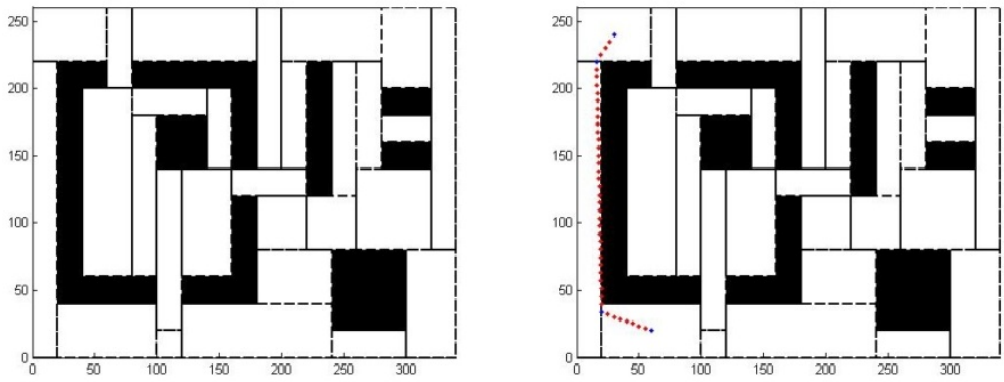

Figure 3.5: *Path planning: The left panel shows an example of an operating area, described with rectangular grid cells. The obstacles are marked as black cells. The right panel shows the final result of the path planning algorithm. In this case, the starting position is in the lower left corner and the desired target position is in the upper left corner. The blue dots in the figure are the waypoints chosen by the A*-algorithm (iterated random points on the edges of the cells) while the red dots are the interpolated waypoints.*

3.2.1 Grid

Two totally different approaches for generating the grid were examined. The first approach is based on decomposing the environment using the **quadtree** algorithm, first introduced in [18]. In the second approach the operating area is simply divided into cells by hand. The quadtree algorithm recursively decomposes the operating area into four identical cells. Each cell can be considered either *free* (no obstacles within the cell), *full* (the cell is totally covered by an obstacle) or *mixed* (neither free nor full). Mixed cells are decomposed again and this procedure continues until a user-defined resolution is reached.

It is desired that the robot should be able to move freely in the the final operating area and the obstacles were therefore increased by the size of the robot during the grid generation phase, i.e. as long as the robot stays within the grid, it can safely navigate.

3.2.2 A*-algorithm

When initializing the algorithm, a rectangular grid consisting of cells, a start position (c_s), and a target position (c_t) is given. In the first iteration of the algorithm, the neighbor cells (c_1, c_2, ..., c_n) of c_s are evaluated. This is done by combining the cost of going to cell c_i from c_s, $h(n)$, and the cost of going from c_i to c_t, $g(n)$. In this implementation, the euclidean distance was used as the cost function for $h(n)$ and $g(n)$. The total cost

$$f(n) = g(n) + h(n), \tag{3.15}$$

is calculated for all successive cells from c_{s} and the cell associated with the lowest cost is chosen as the next cell. The algorithm is iterative and this procedure continues until the target is found. The algorithm still maintains a queue of the alternative path segments along the way.

In this way, if an encountered cell along the way is associated with a higher cost than another encountered path segment, it abandons the higher-cost path segment and traverses the lower-cost path segment instead. This is done by using two lists of paths, one that keeps track of the paths that have been explored while the other stores the ones that are yet to be explored. In the list of paths that have been explored, the parent cells (all cells from c_{s} to the current cell c_{c}) are also stored. Once the target position is reached by the algorithm, the parent cells are traced back to the starting location and the complete path is found. All the cells in the grid are necessarily not explored, since the algorithm explores the cheapest paths first.

The obtained path depends on random points between the cells and the algorithm is iterated n times in order to optimize the path. When this path is found, the final path is created by adding interpolated waypoints between the waypoints chosen by the A*-algorithm. In the right panel of Figure 3.5 an example of a path is shown.

3.3 Obstacle detection and avoidance

The obstacle detection algorithm is inspired by [7] and it is not based on finding explicit objects. Instead, the field of view is split into 25 **sectors** in front of the robot that are considered either free or occupied. Each sector containing at least one point from the Kinect scan, within a certain range, is considered occupied. The obstacle avoidance behavior is divided into two different strategies, *Stop* and *Turn away*, each being executed in separate control system procedures.

In order to determine which strategy to use, five different **zones**, each containing five of the sectors, are used, see Figure 3.6. The decision-making system (described in Section 3.4) chooses when to activate the procedures but in general, if the obstacle is located in the center zone the *Stop*-behavior should be activated, while if the obstacle is on either side, the *Turn away*-behavior should be activated.

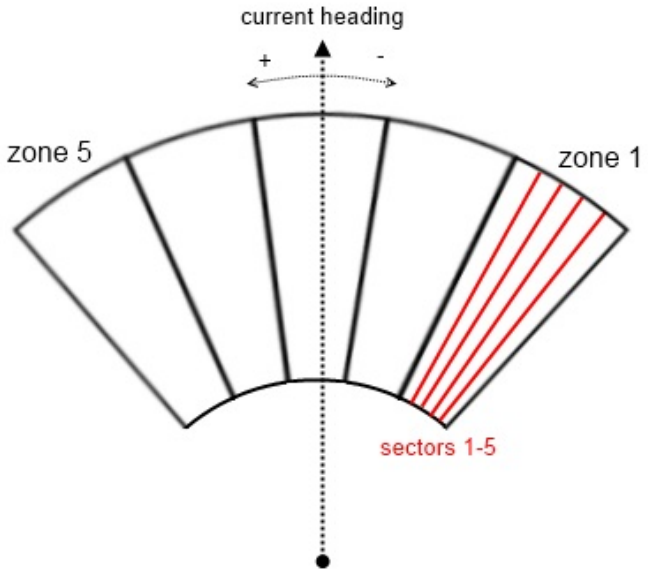

Figure 3.6: *The field of view is divided into five zones (black), each containing five sectors (red).*

If the *Stop*-behavior is activated, the robot simply stops and waits for the obstacle to disappear. The rationale for this strategy is that the most likely causes of this state is either that a person has appeared right in front of the robot or that the localization has failed. If a person has appeared, he will most likely pass the robot or move aside soon and the path will be cleared. If the localization has failed, manual handling of the system failure is most likely needed.

If the *Turn away*-behavior is activated the robot will try to turn away from the obstacle by changing its direction (d) in the following way,

$$d = (N_1 + N_2 - N_4 - N_5)\alpha. \tag{3.16}$$

N_i is the number of occupied sectors within the corresponding zone (see Figure 3.6) and α is a scalar. This means that if many sectors to the right are occupied, the robot will turn to the left and vice versa. If obstacles appear on both sides, e.g. at a narrow passage, the effects will cancel each other out and the robot will continue straight ahead.

3.4 Decision-making

The decision-making system used in this thesis is based on the **utility function method** (UF method), presented in [29, 33]. In the UF method, the **artificial brain** (the control

system of the robot) consists of a decision-making system and a set of predefined brain processes (B_1, B_2, ..., B_n). The tasks of the artificial brain are to decide which brain processes that should be active at what time and to execute their corresponding functions. This is achieved by associating a **utility function** with every brain process, providing a weighted value between each brain process to the artificial brain. The artificial brain then activates and deactivates the brain processes according to these weighted values. Two types of brain processes are used, cognitive and motor processes. In order to avoid conflicting control signals being sent to the motors, only one motor process is allowed to be active at each time.

The utility functions, that indicate whether a brain process should be active or not, depend on a set of **state variables** that describe the state of the system. The state variables (z_1, ..., z_m) can be chosen in different ways, depending on desired system performance and available sensors. In this implementation, the values of the state vector z are acquired during a sensor preprocessing phase. The preprocessing phase takes, as input, depth data from the Kinect sensor, the pose estimate, and the desired heading. Whenever new sensor input is available, the state variables are updated. The utility function u_i,

$$\tau_i \dot{u_i} + u_i = \tanh(c_i \sum_{k=1}^{m} a_{ik} z_k + b_i + \Gamma_i), i = 1, ..., n, \tag{3.17}$$

is calculated using the most recent state variables available. n is the number of processes, m the number of state variables while a_{ik}, b_i, and c_i are parameters that are tuned separately for each brain process. τ_i is a time constant determining how quickly the utility changes. The $\tanh$-function is used to keep the utility in an interval between $\{$-1,1$\}$. The artificial brain will activate all cognitive brain processes B_i with $u_i \geq 0$ and the motor brain process B_j with highest utility, provided that $u_j \geq 0$, and deactivate the other processes not fulfilling these conditions. In Figure 3.7, an example of how the utility changes over time is presented.

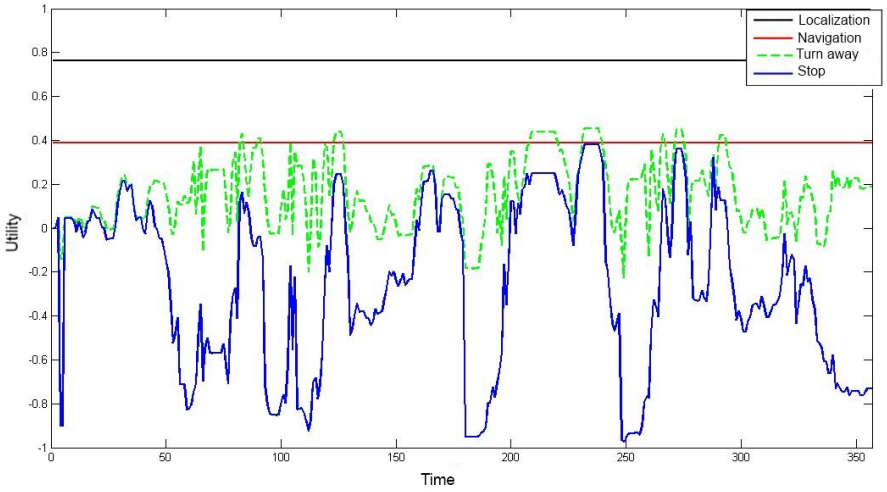

Figure 3.7: *Utility: An example of how four different utility functions change with time at a regular run with the robot. Note that two of the utility functions have static utility, i.e. do not depend on the state variables.*

The gamma parameters (Γ_i), by default set to zero, are used for immediate activation, or in case the of a high negative value, deactivation of a brain process. The gamma function is updated according to

$$\tau_i^{\Gamma}\dot{\Gamma}_i = -\Gamma_i \tag{3.18}$$

in order to allow a delay in deactivation once the gamma parameter has been set to a large value.

3.4.1 Brain processes

The artificial brain of the autonomous robot consists of four brain processes, *Localization* (B_1), *Navigation* (B_2), *Turn away* (B_3), and *Stop* (B_4). B_1 is a cognitive brain process, while the others are motor-related processes. The cognitive process B_1 is handling localization, i.e. it keeps track of the current pose and therefore needs to be active at all time. A brain process can be kept running at all times by setting the corresponding utility function parameter b to an arbitrary positive value and all a-parameters to zero, i.e. ignoring the state variables. In B_1, the pose estimate (x, y, and θ) is updated from odometry readings, see Section 3.1.1. In case a new depth frame from the Kinect sensor is ready and the Hough scan matching conditions are fulfilled, the HSM is carried out, see Section 3.1.4, and the pose estimate is corrected.

In B_2, a path consisting of a sequence of (x, y)-coordinates is planned by the A*-algorithm, described in Section 3.2.2, between the current and the desired position. In

order to reach the targets, B_2 generates control signals that will take the robot to each waypoint on the path. When the target is reached, a new path leading to the next target is planned.

B_3 is used as a complement to the navigation brain process, B_2, with the main task to avoid getting too close to obstacles. This brain process relies on the sector values described in Section 3.3. When the robot approaches an obstacle, the process is activated and the robot will try to turn away.

B_4 is the most important brain process, at least from a safety perspective. The aim of the *Stop*-brain process is to detect obstacles in front of the robot, at a dangerously close distance. When the process is active, the robot brakes and stands still until the danger level decreases.

Chapter 4

Results

In this chapter, the results from several tests runs are presented. As a measure of the system performance, the number of successfully reached targets is used.

The arena, in which the test runs were carried out, is an office environment with a size of approximately 148 square meters; see upper left panel of Figure 4.1. Note that two hallways in the map were too cluttered with objects for the robot to pass through them. No grid was therefore created in those hallways, making them inaccessible to the robot.

4.1 System parameters

The decision-making system used five state variables as input to the utility functions. The state variables used were, $|N_1 + N_2 - N_4 - N_5|$ as z_1 (see Section 3.3), N_3 as z_2, the percentage of entries in the central third of the depth frame registered as *too close* by the Kinect sensor as z_3, the difference between the desired and the current heading ($\Delta\theta = \theta_{\mathrm{d}} - \theta_{\mathrm{c}}$) as z_4, and the percentage of entries in the depth frame registered as *unknown* by the Kinect sensor as z_5.

Localization (B_1), the only cognitive brain process, was set to be active at all time since knowing the pose is one of the most essential features of the system. Therefore, the parameters a_{1k} were all set to 0 and b_1 to 1[1].

Navigation (B_2), was used as the default motor-related brain process, i.e. the robot was set to follow the calculated waypoints as long as no obstacles were present close to the robot. The utility was desired to be a constant positive value, hence a_{2k} were all set to 0 and b_2 to 0.4.

[1]Observe that an arbitrarily positive constant could have been chosen for b_1.

[2]The brain process *Stop* was activated after reaching seven targets and travelling 55.5 % of the distance. Once the *Stop*-behavior was manually aborted, the robot continued and reached all assigned targets.

[3]The brain process *Stop* was activated after reaching 10 targets and travelling 88.9 % of the distance.

Table 4.1: *The results from the 7 test runs.*

Run	Targets reached	Completed distance [m]	Percent of distance [%]	Distance measured by robot [m]	Speed [ms^{-1}]
Run 1	12 / 12[2]	45.1	100	64.3	0.3
Run 2	12 / 12[2]	45.1	100	61.5	0.3
Run 3	11 / 12[3]	42.1	93.4	59.6	0.2
Run 4	10 / 12	40.1	88.9	57.2	0.2
Run 5	7 / 12	25.1	55.5	44.3	0.2
Run 6	12	129.1	N/A	151.5	0.2
Run 7	9	112.9	N/A	138.3	0.2

The remaining two motor-related brain processes, *Turn away* (B_3) and *Stop* (B_4), were set to be activated only once an obstacle was found on either side or at a critical distance right in front of the robot.

a_{3k} were set to be $\{0.05, 0, 0, -0.1, 0\}$ and b_3 to 0. a_{4k} were set to be $\{0, 0.05, 1, -0.9, 0.5\}$ and b_4 to 0. The gamma parameter Γ_4 was used in order to avoid deactivation of B_4 too quickly after a stop-procedure had been executed. The gamma parameter was set to 1 once B_4 was activated, decaying with a time constant τ_4^{Γ} = 10 s[4]. The parameters c_i and τ_i (corresponds to the reaction time of the robot) were set to be 1.0 and 0.15 s respectively, for all brain processes.

4.2 Test runs

In order to evaluate different system properties, three kinds of tests were carried out. In *Run 1 - Run 3*, the interaction between *Localization* and *Navigation*, the most basic brain processes, was tested. In *Run 4* and *Run 5*, the aim was to determine the robot's ability to detect and move around obstacles, or in the case of coming too close to one, stop. In this way, the functionality of the obstacle avoidance brain processes was tested. In *Run 6* and *Run 7*, the long term navigation capabilities and the path planning algorithm were tested. The results from the different test runs are presented in Table 4.1 and in Figure 4.1 the trajectories of *Run 2*, *Run 4*, and *Run 6* are shown.

In *Run 1 - Run 3*, the robot moved in the arena without any obstacles present (except for the arena itself). A list of 11 predefined targets was given to the robot; see the upper left panel of Figure 4.1. Once a target was reached, the next one in the list was chosen and waypoints were generated. If all targets were reached, the run was considered

Once the *Stop*-behavior was manually aborted, the robot continued and reached one more target.

[4]The value was in many cases somewhat large but it was intentionally chosen that way in order to ensure that people had enough time to move aside.

successful and was terminated. In *Run 1* and *Run 2*, the robot completed seven targets before the *Stop*-behavior was activated, see the trajectory of *Run 2* in the upper right panel of Figure 4.1. The robot did not lose track of its position. Instead, the failure to reach all its targets was due to improper a_{ik} parameter selection. The *Turn away*-behavior was activated in a situation where it was not supposed to be activated and the robot could not avoid the obstacle (a wall), resulting in activation of the *Stop*-behavior. Once the *Stop*-behavior was manually aborted, the robot continued and reached all the targets. In *Run 3*, an overshoot from a 90-degree turn resulted in the robot getting too close to a wall and the *Stop*-behavior was once again activated.

In *Run 4* and *Run 5*, the robot used the same arena as in the previous test runs, but with three obstacles present in the operating area[5]. The same targets were used as in *Run 1 - Run 3*. In *Run 4*, the robot handled the first two obstacles well, i.e. the *Turn Away*-behavior was activated. Just before the last obstacle, an overshoot in the same place as in *Run 3* caused the robot to head straight towards the obstacle and the *Stop*-behavior was activated instead of the *Turn Away*-behavior. This was, however, in accordance with the desired behavior when faced with such a situation, since persons blocking the path will most likely move soon. The trajectory of *Run 4* is shown in the lower left panel of Figure 4.1. In *Run 5*, the first obstacle was detected and avoided but the *Stop*-behavior was activated after reaching the first seven targets, in the same way as in *Run 1* and *Run 2*. The robot could not, in that case, be corrected and the test was terminated.

[5]Note that these obstacles were not included in the map.

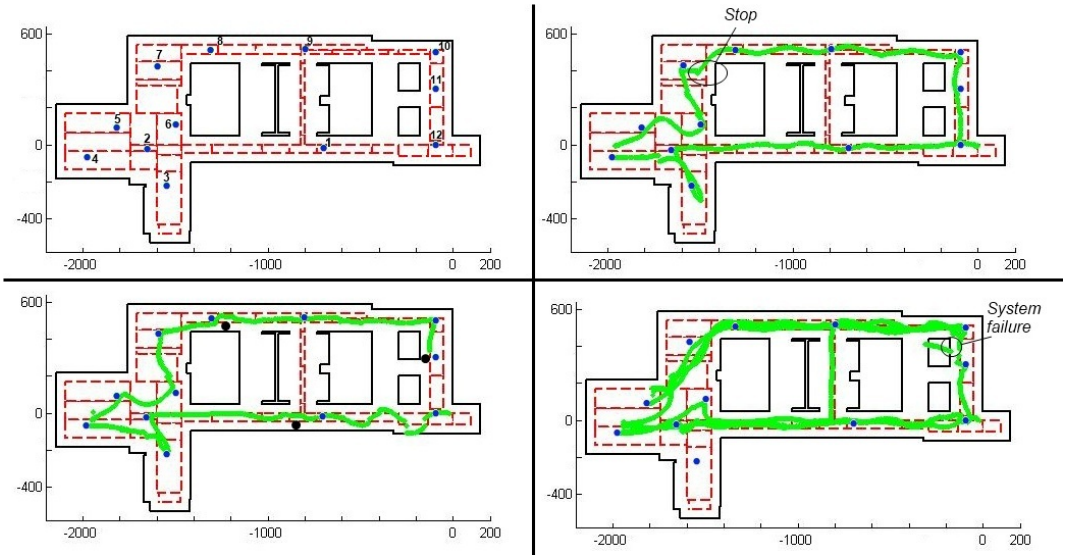

Figure 4.1: *Upper left panel: The operating area and the 11 predefined targets used in the test runs. Obstacles are marked as black lines, the grid cells as dashed red lines, and the targets as blue dots. Upper right panel: The trajectory (in green) of Run 2. Note that the Stop-behavior is activated in the highlighted ellipse. Lower left panel: The trajectory of Run 4 where three obstacles (filled black circles) are present in the arena. Lower right panel: The targets are given randomly to the robot. After a run of approximately 150 m, a system failure occurred, and the robot lost track of the pose estimate.*

In the last two runs, the targets were chosen randomly by the robot and the maximum number of targets was unlimited. In *Run 6*, the robot reached 12 targets and ran approximately 150 m before it suffered the same fate as in *Run 3*. When manually trying to abort the *Stop*-behavior the robot suffered from a system failure where it lost track of the position. This is highlighted by the ellipse in the lower right panel of Figure 4.1. In the seventh and final run, the robot reached nine targets and ran approximately 140 m before the *Stop*-behavior was activated. This was caused by oscillating behavior after the *Turn Away*-behavior had been activated. The robot could not, in that case, be corrected and the test was terminated.

Chapter 5

Conclusions and further work

A system has been developed for autonomous robot navigation in an indoor office environment. The system uses a Kinect sensor and a decision-making system based on the utility function (UF) method [29, 33]. The artificial brain consist of the decision-making system and four brain processes, *Localization*, *Navigation*, *Turn away*, and *Stop*. Results show that the UF method can be used for autonomous navigation, using the Kinect for localization and obstacle avoidance. During testing, the robot completed on average 87.6 % of a 45.1 m long course. When choosing random targets, the robot travelled on average 121 m before ending up in emergency mode.

A common reason for failures during testing was overshoots when carrying out large turns, combined with too slow control of the heading, resulting in activation of the *Stop*-behavior. Possible causes of these problems are the robot's inability to turn sufficiently slowly and too simple heading control (proportional). Another cause of problems were the limitations of the Kinect sensor. In some of the cases where the *Stop*-behavior was activated, the robot could have continued, but it would have had to do so without any sensor input, since the Kinect sensor cannot measure short distances. The risk associated with such behavior was considered too large, resulting in some unnecessary emergency stops.

The localization, based on odometry and scan matching, works well but there is no guarantee that the localization will not fail, causing a complete system failure. Possible causes of localization failures are incorrect scan matchings or large, open areas where the robot has to rely solely on odometry.

The UF method was successfully used but many of the parameters have been chosen from intuition or test results and no formal optimization has been carried out. The most difficult parameters to set were a_{3k} and a_{4k}, the weights of the state variables when calculating the utilities of B_3 (*Turn away*) and B_4 (*Stop*). The results show that the choices of a_{4k} may have caused B_4 to be activated somewhat too easily and the choices of a_{3k} caused B_3 to be activated in improper situations.

To increase the robustness and the system performance, optimization of parame-

ters and a brain process for recovering from emergency stops is needed. Thorough testing and optimization is, however, very time-consuming. Hence, using some sort of simulation environment is recommended. The fact that the robot failed at the same two locations in *Run 1* - *Run 6* indeed indicate that further testing is needed and that solving a few problems could improve the robustness significantly. A simple way to recover from emergency stops would be to make sure that the area behind the robot is free and then simply move backwards until the area right in front of the robot is clear. In case the robot has lost track of its position, a function for global pose correction would be needed in order for the robot to be able to resume its tasks. Such a function would match the Kinect scans to the global map, instead of the local. The robot would, most likely, need to match scans in several directions to the global map in order to find a reliable result.

An interesting future improvement of the localization could include an alternative to using the scan matching result as a direct correction of the pose. An example of a different approach is using a probability-based method, such as a Kalman filter, in order to reduce the effect of incorrect matchings.

In terms of hardware, the Kinect sensor can be used for localization and obstacle avoidance but it has clear limitations in terms of range, field of view, and accuracy. These limitations, especially its inability to measure small distances, suggest that the Kinect sensor should be complemented with additional sensors. For short range measurements, e.g. sonar sensors could be used, providing accurate measurements at a relatively low price.

To conclude, the system works rather well but it needs tuning and a brain process for recovering from emergency stops. The brain processes for navigation and localization work well but in the brain processes for obstacle avoidance and the decision-making system, there is room for improvements that could increase the system performance.

Bibliography

[1] Microsoft kinect. `http://www.xbox.com/sv-SE/Kinect`, 2012.

[2] Arkin R.C. Motor schema based navigation for a mobile robot: An approach to programming by behavior. In *IEEE International Conference on Robotics and Automation.*, volume 4, pages 264–271, 1987.

[3] Besl P.J. and McKay N.D. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence.*, 14(2):239–256, 1992.

[4] Bolmvik Palmgren T., Hidén Rudander J., Johansson S., Sandberg J., and Stensson V. Marg-e - konstruktion av en autonom guiderobot. *Institutionen för Signaler och System, Chalmers Tekniska Högskola, Kandidatarbete/rapport nr SSYX02-09-09-a*, 2009.

[5] Borenstein J. and Koren Y. Obstacle avoidance with ultrasonic sensors. *IEEE Journal of Robotics and Automation.*, 4(2):213–218, 1988.

[6] Borenstein J. and Koren Y. Real-time obstacle avoidance for fast mobile robots in cluttered environments. In *IEEE International Conference on Robotics and Automation.*, volume 1, pages 572–577, 1990.

[7] Borenstein J. and Koren Y. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation.*, 7(3):278–288, 1991.

[8] Brooks R. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986.

[9] Buschmann T., Lohmeier S., Schweinbacher M., Favot V., Ulbrich H., von Hundelshausen F., Rohe G., and Wuensche H. Walking in unknown environments - a step towards more autonomy. In *10th IEEE-RAS International Conference on Humanoid Robots (Humanoids).*, pages 237–244, 2010.

[10] Censi A., Iocchi L., and Grisetti G. Scan matching in the hough domain. In *Proceedings of the IEEE International Conference on Robotics and Automation.*, pages 2739–2744, 2005.

[11] Cherubini A., Spindel F., and Chaumette F. A new tentacles-based technique for avoiding obstacles during visual navigation. In *IEEE International Conference on Robotics and Automation (ICRA).*, pages 4850–4855, 2012.

[12] Cox I. J. Blanche- an experiment in guidance and navigation of an autonomous robot vehicle. *IEEE Transactions on Robotics and Automation*, 7(2):193–204, 1991.

[13] Crowley J.L. World modeling and position estimation for a mobile robot using ultrasonic ranging. In *IEEE International Conference on Robotics and Automation.*, volume 2, pages 674–680, 1989.

[14] Dall Larsen T., Lenfer Hansen K., Andersen N., and Ravn O. Design of kalman filters for mobile robots; evaluation of the kinematic and odometric approach. In *Proceedings of the IEEE International Conference on Control Applications.*, volume 2, pages 1021–1026, 1999.

[15] Dijkstra E. A note on two problems in connexion with graphs. *Numerische Mathematik 1*, pages 269–271, 1959.

[16] Dissanayake M. W. M. G., Newman P., Clark S., Durrant-Whyte H. F., and Csorba M. A solution to the simultaneous localization and map building (slam) problem. *IEEE Transactions on Robotics and Automation.*, 17(3):229–241, 2001.

[17] Elfes A. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[18] Finkel R.A. and Bentley J.L. Quad-trees: A data structure for retrieval on composite keys. *ACTA informatica*, 4:1–9, 1974.

[19] Fox D., Burgard W., and Thrun S. The dynamic window approach to collision avoidance. *IEEE Robotics Automation Magazine.*, 4(1):23–33, 1997.

[20] Freedman B., Shpunt A., Machline .M, and Arieli Y. Depth mapping using projected patterns. 2010.

[21] Hart P.E, Nilsson N.J, and Raphael B. A formal basis for the heuristic determination of minimum costs paths. *IEEE Transactions on Systems Science and Cybernetics.*, 4(2):100–107, 1968.

[22] Khatib O. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation.*, volume 2, pages 500–505, 1985.

[23] Leonard L. and Durrant-Whyte H. Simultaneous map building and localization for an autonomous mobile robot. In *IEEE/RSJ International Workshop on Intelligent Robots and Systems.*, volume 3, pages 1442–1447, 1991.

[24] Lu F. and Milios E. Robot pose estimation in unknown environments by matching 2d range scans. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition.*, pages 935–938, 1994.

[25] Montmerlo M., Thrun S., Koller D., and Wegreit B. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 593–598, 2002.

[26] Pearl J. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.

[27] Pradalier C. and Sekhavat S. Concurrent matching, localization and map building using invariant features. In *IEEE/RSJ International Conference on Intelligent Robots and Systems.*, volume 1, pages 514–520, 2002.

[28] Sakagami Y., Watanabe R., Aoyama C., Matsunaga S., Higaki N., and Fujimura K. The intelligent asimo: System overview and integration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems.*, volume 3, pages 2478–2483, 2002.

[29] Sandberg M., Wolff K., and Wahde M. A robot localization method based on laser scan matching. In Kim J.-H., Ge S.S., Vadakkepat P., and Jesse N. et al., editors, *Proceedings of FIRA2009, LNCS 5744*, pages 179–186. Springer, 2009.

[30] Smith R., Self M., and Cheeseman P. Estimating uncertain spatial relationships in robotics. In *IEEE International Conference on Robotics and Automation.*, volume 4, page 850, 1987.

[31] Thorpe C., Hebert M.H., Kanade T., and Shafer S.A. Vision and navigation for the carnegie-mellon navlab. *IEEE Transactions on Pattern Analysis and Machine Intelligence.*, 10(3):362–373, 1988.

[32] von Hundelshausen F., Himmelsbach M., Hecker F., Mueller A., and Wuensche H.-J. Driving with tentacles: Integral structures for sensing and motion. *Journal of Field Robotics*, 4:640–673, 2008.

[33] Wahde M., Sandberg M., and Wolff K. Reliable long-term navigation in indoor environments. In Venelinov Topalov A., editor, *Recent Advances in Mobile Robotics*, pages 261–286. InTech, 2011.

[34] Wang H., Yu Y., and Yuan Q. Application of dijkstra algorithm in robot path-planning. *Second International Conference on Mechanic Automation and Control Engineering*, pages 1067–1069, 2011.

[35] Warren C.W. Global path planning using artificial potential fields. *IEEE International Conference on Robotics and Automation*, 1:316–321, 1989.

[36] Watanabe M., Onoguchi K., Kweon I., and Kuno Y. Architecture of behavior-based mobile robot in dynamic environment. *IEEE International Conference on Robotics and Automation*, 2:2711–2718, 1992.

[37] Wehner R. Desert ant navigation: how miniature brains solve complex tasks. *Journal of Comparative Physiology A: Neuroethology, Sensory, Neural, and Behavioral Physiology*, 189:579–588, 2003.

[38] Weiss G., Wetzler C., and von Puttkamer E. Keeping track of position and orientation of moving indoor systems by correlation of range-finder scans. In *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems.*, volume 1, pages 595–601, 1994.

[39] Wenfeng B., Lina H., and Yunfeng S. An improved a*-algortihm in path planning. In *International Conference on Computer, Mechatronics, Control and Electronic Engineering (CMCE).*, volume 3, pages 235–237, 2010.

[40] Zeng C., Zhang Q., and Wei X. Robotic global path-planning based modified genetic algorithm and a* algorithm. In *Third International Conference on Measuring Technology and Mechatronics Automation (ICMTMA).*, volume 3, pages 167–170, 2011.

[41] Zezhong X., Jilin L., and Zhiyu X. Map building and localization using 2d range scanner. In *IEEE International Symposium on Computational Intelligence in Robotics and Automation.*, volume 2, pages 848–853, 2003.