# Design Specification
# AB Mail Robot

Version 1.0

Author: Daniel Bagarn Barac
Date: December 24, 2010

| | | | |
|---|---|---|---|
| Course name: | Control Project | E-mail: | danba185@student.liu.se |
| Project group: | AB Mail Men | Document responsible: | Daniel Bagarn Barac |
| Course code: | TSRT10 | Author's E-mail: | marme287@student.liu.se |
| Project: | AB Mail Robot | Document name: | DesignSpecification_v10.pdf |

## Status

| | | |
|---|---|---|
| Reviewed | Karl Granström | 2010-10-08 |
| Approved | Karl Granström | 2010-10-11 |

## Project Identity

| | |
|---|---|
| **Group E-mail:** | danba185@student.liu.se |
| **Homepage:** | http://www.isy.liu.se/edu/projekt/tsrt10/2010/postrobot-2010/ |
| **Orderer:** | Karl Granström, Linköping University<br>**Phone:** +46 13 281333, **E-mail:** karl@isy.liu.se |
| **Customer:** | Marcus Pettersson, ABB Västerås<br>**Phone:** +46 21 345194, **E-mail:** marcus.pettersson@se.abb.com |
| **Course Responsible:** | David Törnqvist, Linköping University<br>**Phone:** +46 13 281882, **E-mail:** tornqvist@isy.liu.se |
| **Project Manager:** | Daniel Barac |
| **Advisors:** | André Carvalho Bittencourt, Linköping University<br>**Phone:** +46 13 282622 , **E-mail:** andrecb@isy.liu.se |

## Group Members

| Name | Responsibility | Phone | E-mail (@student.liu.se) |
|---|---|---|---|
| Daniel Barac (DB) | Project Leader | +46 702641857 | danba185@student.liu.se |
| Martin Melin (MM) | Document responsible | +46 702887793 | marme287@student.liu.se |
| Erik Erkstam (EE) | Software responsible | +46 737266812 | erier982@student.liu.se |
| Hugo Kinner (HK) | Test responsible | +46 733688378 | hugki634@student.liu.se |
| Manfred Hallström (MH) | | +46 703679569 | manha932@student.liu.se |
| Simon Hugosson (SH) | Control responsible | +46 708778471 | simhu610@student.liu.se |
| Niklas Carlsson (NC) | Information responsible | +46 709601672 | nikca291@student.liu.se |
| Nicklas Forslöw (NF) | SLAM responsible | +46 739516430 | nicfo307@student.liu.se |

# Document History

| Version | Date | Changes made | Sign | Reviewer |
|---|---|---|---|---|
| 0.1 | 2010-10-04 | First draft | MH SH HK NC NF MM EE | MM, DB |
| 0.2 | 2010-10-07 | Second draft | SH MM EE MH NC NF DB HK | EE MM MH NF |
| 0.3 | 2010-10-08 | Third draft | SH MM EE MH NC NF DB HK | DB |
| 0.4 | 2010-10-10 | Fourth draft | SH MM EE DB | DB |
| 1.0 | 2010-10-11 | First Version | DB | DB |

# Contents

# 1 Introduction

Fido-Dido (Figure 1) is a wheeled mobile robot manufactured by MobileRobots Inc. Its most important features are a SICK laser range finder, 8 forward-facing ultrasonic array sonars, odometers and a Micro-Controller. The robot shall be able to autonomously navigate in an office-landscape using a pre-defined map which it also shall be able to update when changes are detected. It shall also be able to find its way from point A to point B.

Figure 1: Fido-Dido

This document provides a detailed description of how the software and hardware will interact in order to achieve our goals, specified in section 1.2.

## 1.1 Parties

Client of the project is Karl Granström, PhD-student in Automatic Control, Department of Electrical Engineering at Linköping University. Supervisor of the project is André Carvalho Bittencourt, PhD-student in Automatic Control, Department of Electrical Engineering at Linköping University. Customer of the project is Marcus Pettersson at ABB Cooperate Research Center (ABB CRC). The development is performed by the project group ABB MailMen.

## 1.2 Purpose and goal

The purpose of the project is to develop software and then implement it on a robot so that the robot can navigate in an office environment. The project group will evaluate future possibilities to make this robot able to deliver objects, e.g. mail. The goal of the project is to develop a system for an autonomous robot which shall be able to navigate from point A to point B in an office space, specifically at ABB CRC. The long term goal of the product is to extend the robot with a robotic arm to enable the delivery of objects, e.g. mail.

## 1.3 Usage

The product can be used as a platform for further research at ABB CRC or at Linköping University.

## 1.4 Background information

This project is a part of the CDIO-course TSRT10 at Linköping University and is performed in collaboration with ABB CRC. The group consists of eight students with expertise within automatic control and sensor fusion.

## 1.5 Environmental limitations

The robot is limited to operate in an office environment. The robot cannot be expected to be able to open doors or move objects. The floor on which the robot will operate must be free from obstacles which the robot cannot detect.

# 2 Overview of the System

The system consists of a MobileRobots P2-DX robot with a HP ProBook 6450b laptop mounted on it. The laptop runs the application that controls the robot and could also be used as an interface if a user wants to manually control the robot or view the robot's map. The robot and the laptop communicate via a RS232 serial port. The different parts of the system and how they communicate can be viewed in figure 2. The laser and sonars send range measurements and the odometers send wheel position and speed estimates. The micro controller (MC) decodes the data from the sonars and odometers and passes it on the program. The MC also translates the motion commands from the program to control signals for the motors. When collecting data for Offline-SLAM, see section 5.2.2, the user drives the robot with the laptop's arrow keys. When mapping is done the user can tell the program where the robot should go. The output from the program to the user is a map and the robot's current position in the map.

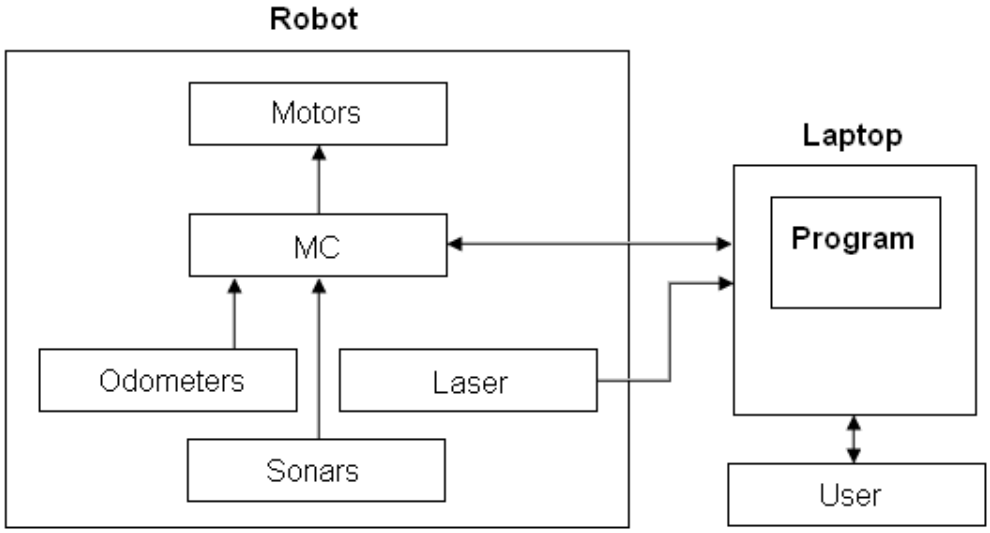


Figure 2: Overview of the system.

## 2.1 Subsytems

This section includes a short description of the subsystems.

### 2.1.1 Robot

The robot is a wheeled MobileRobots P2-DX robot equipped with odometers, a laser, sonars and a micro controller (MC). The micro controller receives data from the sonars and odometers. It also sends control signals, calculated by the software on the laptop, to the servos.

### 2.1.2 Laptop

The laptop receives sensor data from the laser and the robot's micro controller. It is also responsible for the software which handles SLAM, trajectory planning and calculation of

the control signals.

The application is written in C++ and uses the ActivMedia Robotics Interface for Application (ARIA) library for all communication with the robot. ARIA has utilities for both receiving data from the sensors and controlling the motors. The program can be divided into three subsystems with specific tasks according to figure 3. These subsystems will be thoroughly explained in sections 5 to 7. The SLAM system receives range measurements from the laser and sonars. The control system receives a pose estimate from the odometers and pass it on to the SLAM system together with a calculated covariance matrix. The SLAM system sends a grid map and an improved pose estimation to the trajectory planner which plans a trajectory to the goal and sends the next position to go to, to the control system. It, in turn, sends the desired wheel velocities to the motors.

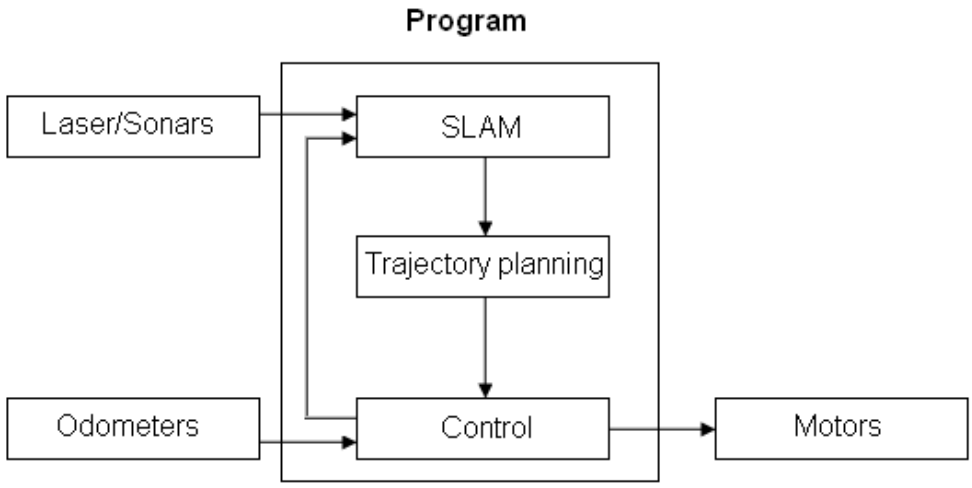


Figure 3: Program structure.

# 3 The Robot

The robot has two drive wheels with encoders for measuring their angular position and angular velocity. It also has two range sensors, a sonar array and a Sick LMS200 laser range finder, both facing forward and with a field of view of 180°. The laptop is connected to a micro controller in the robot that is responsible for reading sonar and odometry data as well as controlling the motors. The range data from the laser is sent to the laptop through a RS232 connection.

## 3.1 Odometers

The robot wheels have 800 tick quadrature shaft encoders that are used to provide wheel position and speed estimates.

## 3.2 Laser Sensor

The SICK LMS200 has a field of view of 180° and an angular resolution that can be set to 0.25°, 0.5° or 1°. The resolution in the range measurements is 10 mm and the accuracy typically ±15 mm. The range is 10 m for 10 reflectivity, which corresponds to cardboard, and can be up to 32 m for more reflective materials. The sampling rate is around 4 Hz.

## 3.3 Sonar Sensors

The robot has an array of eight sonars in directions ±10°, ±30°, ±50° and ±90° (where 0° is straight forward). Both the acquisition rate and the sonar gain are adjustable. The acquisition rate is 25 Hz by default and the sensitivity ranges from 10 cm to over 4 m, depending on the gain. One important feature of the sonar is that it can detect transparent objects, which the laser cannot.

# 4 Motion Model

The robot pose is given by the following vector, where x and y are the position coordinates and $\theta$ is the heading direction, $X_i = (x_i, y_i, \theta_i)^T$. To aid the estimation of the robot's pose, a motion model is used. The motion model gives the probability of the pose $X_i$ given the previous pose $X_{i-1}$ and the motion command $u_t = (\Delta_x^{odo}, \Delta_y^{odo}, \Delta_\theta^{odo})$, $(p(X_i|u_t, X_{i-1}))$. $u_t$ is defined as the difference between the current pose and the next pose, view figure 4.

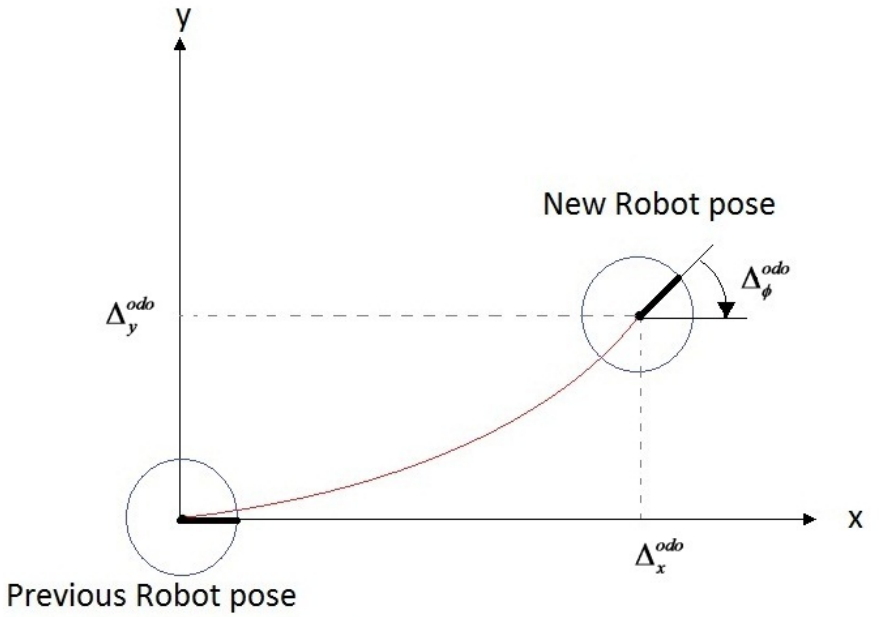


Figure 4: Model of $u_t$

According to propositions in [9] the new position can be calculated in the following way,

$$\begin{pmatrix} x_i \\ y_i \\ \theta_i \end{pmatrix} = \begin{pmatrix} x_{i-1} \\ y_{i-1} \\ \theta_{i-1} \end{pmatrix} + \begin{pmatrix} cos(\theta + \frac{\Delta_\theta^{odo}}{2}) & -sin(\theta + \frac{\Delta_\theta^{odo}}{2}) & 0 \\ sin(\theta + \frac{\Delta_\theta^{odo}}{2}) & cos(\theta + \frac{\Delta_\theta^{odo}}{2}) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \Delta_x^{odo} \\ \Delta_y^{odo} \\ \Delta_\theta^{odo} \end{pmatrix} =$$

$$= \begin{pmatrix} x_{i-1} \\ y_{i-1} \\ \theta_{i-1} \end{pmatrix} + H \begin{pmatrix} \Delta_x^{odo} \\ \Delta_y^{odo} \\ \Delta_\theta^{odo} \end{pmatrix}. \quad (1)$$

The distribution of the new pose is considered to be Gaussian with mean equal to the odometer movement added to the previous pose. To obtain the covariance of the pose, the variance of each odometry variable must be estimated. The matrix $\Sigma$ is defined as a diagonal matrix,

$$\Sigma = \begin{pmatrix} \sigma_{\Delta_x^{odo}} & 0 & 0 \\ 0 & \sigma\Delta_y^{odo} & 0 \\ 0 & 0 & \sigma\Delta_\theta^{odo} \end{pmatrix}. \quad (2)$$

The variances of the odometers are defined as

$$\sigma_{\Delta_x^{odo}} = \sigma_{\Delta_y^{odo}} = \sigma_{xy}^{min} + \alpha_1\sqrt{(\Delta_x^{odo})^2 + (\Delta_y^{odo})^2} + \alpha_2|\Delta_\theta^{odo}|, \tag{3}$$

$$\sigma_{\Delta_\theta^{odo}} = \sigma_\theta^{min} + \alpha_3\sqrt{(\Delta_x^{odo})^2 + (\Delta_y^{odo})^2} + \alpha_4|\Delta_\theta^{odo}|. \tag{4}$$

The following parameters will be set after analyzing robot data

$$\begin{bmatrix} \alpha_1 & [meters/meter] \\ \alpha_2 & [meters/degree] \\ \alpha_3 & [degree/meter] \\ \alpha_4 & [degrees/degree] \\ \sigma_{xy}^{min} & [meters] \\ \sigma_\theta^{min} & [degrees] \end{bmatrix}. \tag{5}$$

Finally the covariance matrix can be calculated using the Jacobian $J$ of $H$ and the matrix $\Sigma$. The Jacobian matrix can be computed by following the propositions in [1]

$$J = \begin{pmatrix} 0 & 0 & \frac{-sin(\theta+\frac{\Delta_\theta^{odo}}{2})}{2} - \frac{cos(\theta+\frac{\Delta_\theta^{odo}}{2})}{2} \\ 0 & 0 & \frac{cos(\theta+\frac{\Delta_\theta^{odo}}{2})}{2} - \frac{sin(\theta+\frac{\Delta_\theta^{odo}}{2})}{2} \\ 0 & 0 & 0 \end{pmatrix}. \tag{6}$$

Finally the covariance matrix is calculated as follows,

$$Cov = J\Sigma J^T. \tag{7}$$

# 5 SLAM

This subsystem is responsible for the simultaneous localization and mapping (SLAM). Sensor fusion is used to estimate the robot position and heading in a map and to update the map as changes are observed. Here, sensor models and a motion model of the robot together with data from all the sensors are used.

The system will contain two different SLAM-algorithms which are called Offline-SLAM and Online-SLAM. The purpose of Offline-SLAM is to create an accurate map of the environment in which the robot will operate. This algorithm is too computationally demanding to be used in real time and therefore Online-SLAM is needed for real time applications. Online-SLAM runs in real time and it uses a map built with the help of Offline-SLAM in which it localizes the robot.

For simplicity, the sensors used by Offline-SLAM do not include the sonars, only the odometers and the laser are used. Therefore, the resulting map only contains "what the laser sees". When the map is built it needs to be converted for computational reasons. The converted map is a line based map which Online-SLAM uses for localization. The idea is that the line based map is a sort of filtered version of the map arising from Offline-SLAM in the way that it manly contains larger objects such as walls. Because these kind of objects are not likely to be moved, removed or added, Online-SLAM can use the line based map to achieve good localization without manipulating it.

However, the line based map cannot be used for navigation. The reason for this is that the robot has to know about any obstacles that it could collide with and it is not likely that the line based map contains this information. Therefore, Online-SLAM has to build and update a map for navigation. This map is a occupancy grid map which is built in real time by measurements from both the laser and the sonars.

How these tasks are solved and what sensor models are used is described below.

## 5.1 Sensor Models

This section describes what data is delivered by each type of sensor and how this data can be related to a global coordinate system. To derive the sensors locations in the global coordinate system there is a need to define the robot's local coordinate system. If a position is given in the robot's local coordinate system as $(p_x^r, p_y^r)^T$ the corresponding global position $(p_x, p_y)^T$ is given by

$$\begin{pmatrix} p_x \\ p_y \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} p_x^r \\ p_y^r \end{pmatrix} \tag{8}$$

where $X = (x, y, \theta)^T$ is the pose of the robot. This is operation is denoted by

$$(p_x, p_y)^T = X \oplus (p_x^r, p_y^r)^T. \tag{9}$$

### 5.1.1 Laser Scans

The laser sensor delivers scans of the enviroment at a rate of about 4 Hz. One laser scan is a sequence $E = (e_0, e_1, ..., e_{360})$, where $e_i$ is a distance measurement taken from the robot's position with an angle $\psi_k$ relative to the robot's heading. The angle $\psi_k$ is given by $\psi_k = -90 + 0.5k$. If the robot's pose is $X = (x, y, \theta)^T$ and the position of the laser sensor is $(0, 0)$ in the robot's local coordinate system, then each measurement $e_i$

corresponds to a physical position $(u_x, u_y)_i$ by

$$(u_x, u_y)_i = (x, y) + e_i(\cos(\psi_i + \theta), \sin(\psi_i + \theta)) \tag{10}$$

and a scan $E$ corresponds to a set of positions $U = \bigcup_{k=0}^{360}(u_x, u_y)_i^k$.

### 5.1.2 Sonar Measurement

The sonars have the same function as the laser in the way that they give distance measurements. However, there is a big difference between the information one can derive from a sonar measurement. While a laser measurement can be described as a line segment, a sonar measurement is more of a cone and it is more difficult to derive the position of the object which the sonar measurement gives the distance to. This is illustrated in figure 5. On the other hand one can derive a circle segment on which there is a physical object and the amount of negative information is larger than what is given by a laser measurement: the laser gives only a line on which there is no objects while the sonar gives a cone. Negative information means information about where there are no objects. The opening angles of the sonars are about 30°.

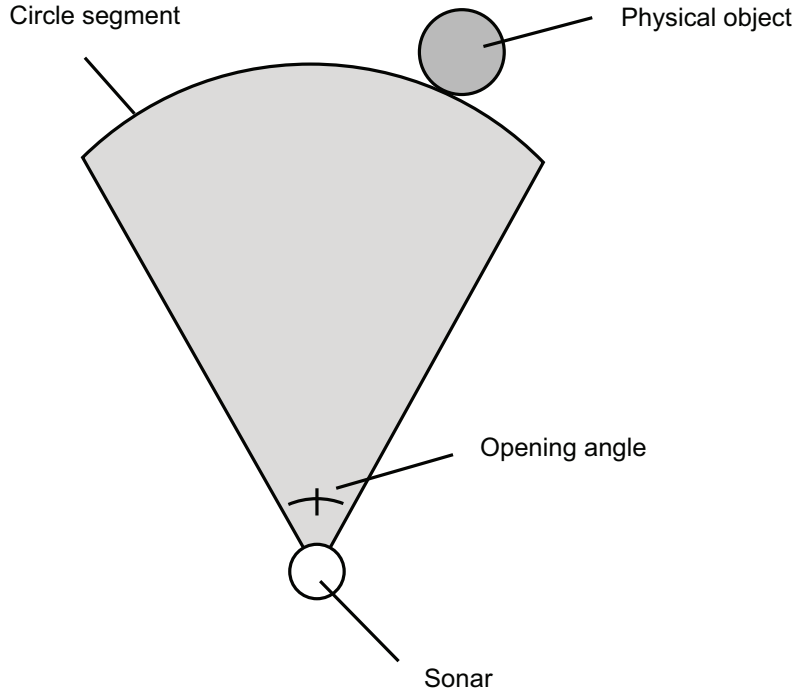


Figure 5: The problem with the sonar

If we let $(s_x^r, s_y^r)^T$ denote the position of a sonar in the robot's local coordinate system and let $v$ be the direction of the sonar relative to the robot's heading, then the circle segment $B$ is given by

$$B = \{X \oplus ((s_x^r, s_y^r)^T + \rho(\cos(v + \alpha), \sin(v + \alpha))^T) : |\alpha| \leq 15\} \tag{11}$$

where $\rho$ is the distance measured by the sonar.

## 5.2 Building the Map

This section describes how the map of the environment, in which the robot will operate, is built. It is divided into three subsections: Data Collection, Offline-SLAM and Map Conversion.

### 5.2.1 Data Collection

The robot is manually maneuvered in some area and at certain moments data are saved. The saved data contains an estimate of the relative motion since the previous save point as well as laser measurements. Instead of saving data at certain points in time, data will only be saved when the robot's pose has changed significantly. New data will be saved if new laser measurements are collected and if the odometers estimate that the robot has traveled a distance greater than some distance $d$, or that the robot has changed its heading with an angle greater than some angle $\beta$. The values of $d$ and $\beta$ will be set experimentally. The reason why data is saved in this manner is because there is no use of saving a lot of data if the robot does not change its pose.

### 5.2.2 Offline-SLAM

Since the laser scans are very accurate, the problem of building an accurate map from laser scans is the problem of estimating the poses from which the scans where taken as accurately as possible. Therefore, one can consider the following estimation problem.

Assume that the robot has taken $n+1$ laser scans. Let $X_i = (x_i, y_i, \theta_i)^T$ describe the robot position and heading when scan $i$ was aquired and simply let $X_0 = (0,0,0)^T$. Let

$$D_{ij} = X_i - X_j \tag{12}$$

be the measurement equation. An observation $\overline{D}_{ij}$ of $D_{ij}$ is defined as

$$\overline{D}_{ij} = D_{ij} + \Delta D_{ij} \tag{13}$$

where $\Delta D_{ij}$ is a random Gaussian noise with zero mean and known covariance matrix $C_{ij}$. Assume that there is a set $S$ consisting of pairs $(i,j)$, where $0 \leq i < j \leq n$ such that there is an observation of $\overline{D}_{ij}$ with a corresponding covarince matrix $C_{ij}$ for each pair $(i,j) \in S$.

Then a criterion of optimal estimation can be formed based on the *maximum likelihood* concept. By assuming that the observation errors are mutually independent, the criterion is to minimize the following Mahalanobis distance

$$W = \sum_{(i,j)\in S} (D_{ij} - \overline{D}_{ij})^T C_{ij}^{-1} (D_{ij} - \overline{D}_{ij}) \tag{14}$$

The solution, which can be found in [8], of this problem yields an approximation

$$\hat{\mathbf{X}} = (\hat{X_0}, \hat{X_1}, .., \hat{X_n})^T \tag{15}$$

of

$$\mathbf{X} = (X_0, X_1, .., X_n)^T \tag{16}$$

as well as a covariance matrix

$$\mathbf{C}_{\hat{\mathbf{X}}} = Cov(\hat{\mathbf{X}}) \tag{17}$$

The quality of the solution depends on the amount of observations as well as the quality of them. In practice, an observation is an estimation of a pose change that is obtained from measurements together with mathematical models.

The algorithm contains two different kinds of observations, called weak and strong links. A weak link is simply an observation taken from the odometry of the pose difference between two adjacent poses. Thus a weak link, $\overline{D}_{ij}$, and the corresponding covariance matrix, $C_{ij}$, is obtained from the motion model. While weak links are simply taken from the motion model, strong links are more sophisticated. The idea is to find pairs of poses with scans that are strongly correlated. In other words this means that pairs of poses from which the laser detects the same physical objects has to be found. By matching the scans with a tching algorithm, a new observation $\overline{D}_{ij}$ and a corresponding covariance matrix $C_{ij}$ can then be obtained. Scan matching can be described as a process of fitting two scans by performing a rotation followed by a translation of one of the scans.

The amount of strong links depends on the initial estimates of the covariance matrix between two poses. This will be further described in the next section. Since the solution yields new estimates as well as a covariance matrix, this algorithm can be run again to get an even better estimate of the poses. This time the weak links will be taken from the new estimates. This procedure can be repeated until the estimations converge. An illustration of the algorithm can be seen in figure 6.

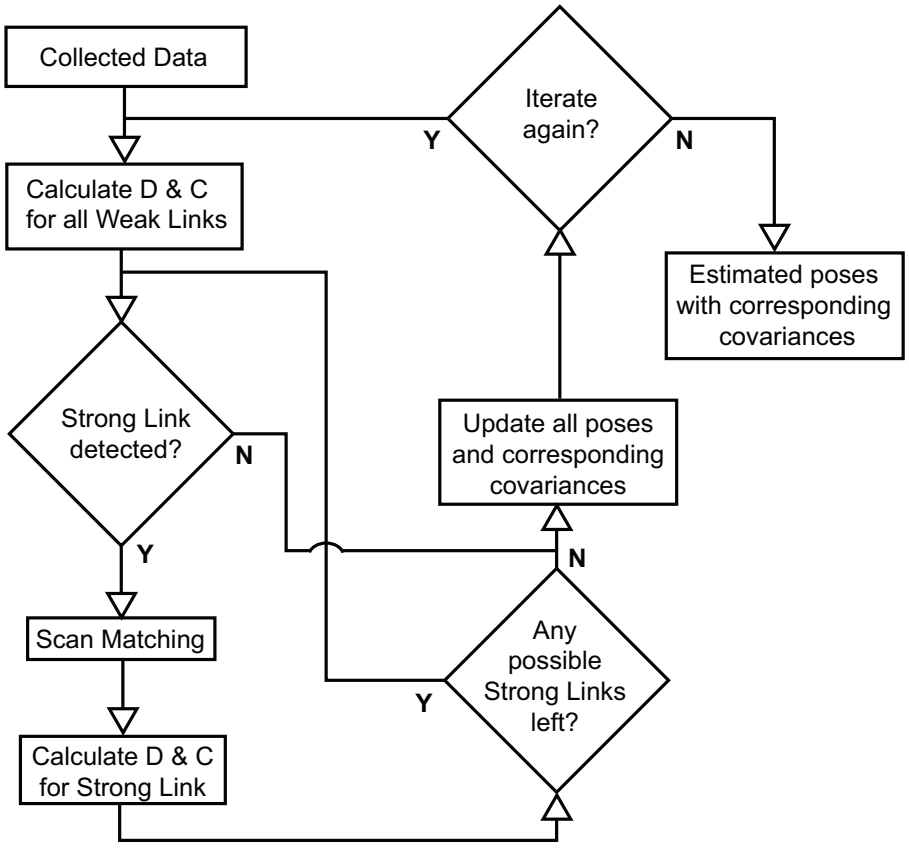


Figure 6: Offline-SLAM algorithm

### 5.2.3 Detection of Strong Links

The detection of the strong links is much of a key to estimate the poses well. The question is how to detect that there is a a strong link between two poses. Hence, one must find conditions such that if they are fulfilled, it is likely that the laser scans overlap. The fact that the laser scans only cover a field of view of 180 degrees limits the ways of detecting strong links. However, the following idea will handle the problem of the limited field of view.

Let $U_i = \bigcup_{k=0}^{N} (u_x, u_y)_i^k$ denote the set of points in the scan taken from $X_i = (x_i, y_i, \theta_i)^T$. To decide if there is a strong link $i \leftrightarrow j$ the following will be done. Define a set $W_i$ as

$$W_i = \{(x, y) = (x_i, y_i) + r(\cos\alpha, \sin\alpha) : r_{min} \leq r \leq r_{max}, |\theta_i - \alpha| \leq \varphi\} \tag{18}$$

where

$$r_{min} = (1 - r_0) \cdot \min_{0 \leq k \leq N} ||(x_i, y_i) - (u_x, u_y)_i^k||_2 \tag{19}$$

$$r_{max} = (1 + r_1) \cdot \max_{0 \leq k \leq N} ||(x_i, y_i) - (u_x, u_y)_i^k||_2 \tag{20}$$

$$\varphi = \pi/2 + \varphi_0 \tag{21}$$

The values of $r_0$, $r_1$ and $\varphi_0$ will be set experimentally, but it must hold that $r_0 \in [0, 1]$, $r_1 \geq 0$ and $\varphi_0 \in [0, \pi/2]$.

Let $M_{ij} = \#(u_x, u_y) \in U_j \bigcap W_i$. Define $M_{ji}$ in the same way and regard the link between $i$ and $j$ as a strong link if and only if $\min(M_{ij}, M_{ji}) > M$. If the link is strong, $U_j \bigcap W_i$ and $U_i \bigcap W_j$ is used for scan matching.

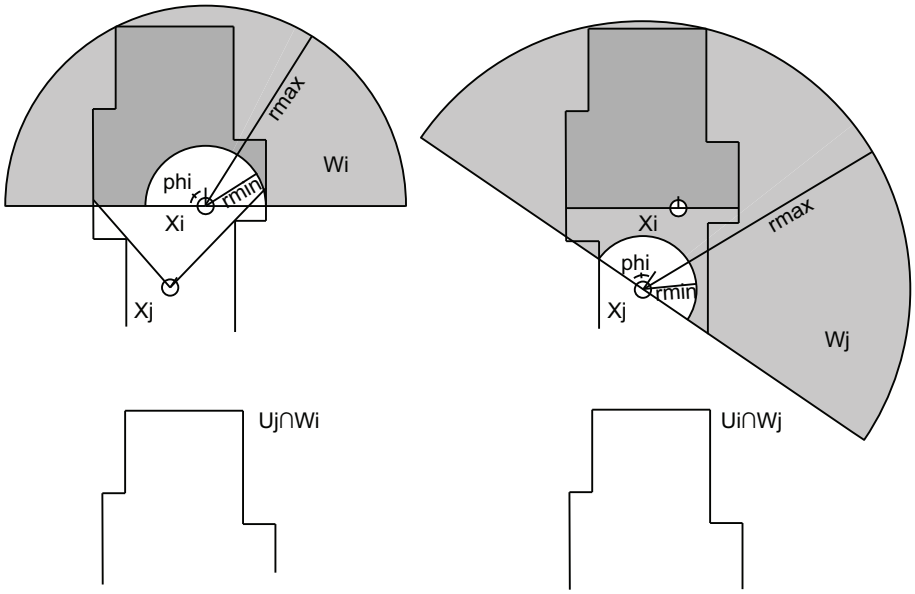


Figure 7: Detection of strong link

Illustrations of the idea behind the algorithm can be seen in figure 7 and figure 8. In these illustrations the values of $r_0$, $r_1$ and $\varphi_0$ are all zero. The purpose of these design parameters is because the pose $X_i$ can only be estimated. With help of the illustrations

one might realize that if the estimations are bad a strong link might be missed if the parameters are set to zero. By increasing the values of these parameters, the sets $W_i$ and $W_j$ will grow and the risk of missing a strong link decreases. However, increasing them too much could result in a false detection.

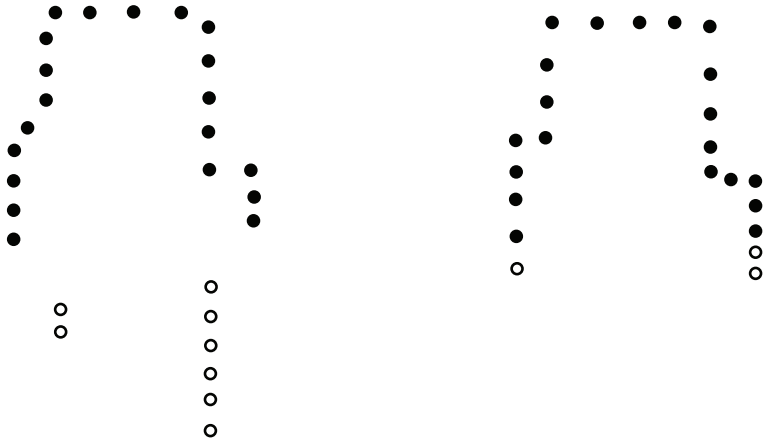

Figure 8: Illustration of $U$ and $U \bigcap W$. The circles represent the measured points $U$. The filled circles represent the points that are counted and eventually used for scan matching $U \bigcap W$. To the left is $U_j$ and $U_j \bigcap W_i$ and to the right is $U_i$ and $U_i \bigcap W_j$.

Another way of avoiding false detection is to require that the estimated covariance from the motion model, $C_{ij}$, between two poses is small enough. This means that the above described procedure does not have to run for all possible pairs of poses since $C_{ij}$ is growing with the difference between $i$ and $j$. The weak links will also be kept and therefore a requirement is that $|i - j| > 1$. An example of how the network could look like can be seen in Figure 9.

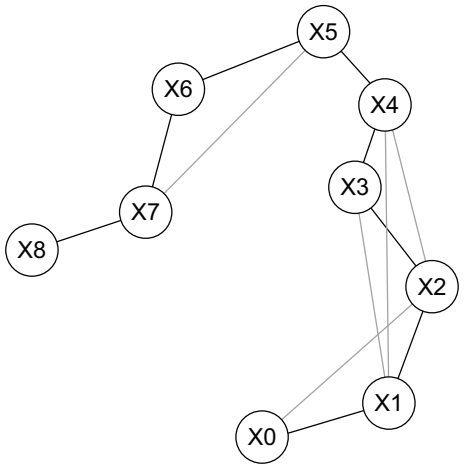


Figure 9: Network of weak and strong links. The black lines represent weak links and the gray lines represent strong links.

### 5.2.4 Scan Matching

Scan matching is the process of aligning an observed set of measurement positions $U_o$ with a reference set of measurement positions $U_r$. Here $U_o$ and $U_r$ refers to the sets of measurement positions generated by a laser range scans at two different robot poses $X_o$ and $X_r$. A set of measurement positions can be described as $U = \bigcup_{k=0}^{360}(u_x, u_y)^k$ in the local cartesian coordinate system of the specific pose according to Eq. (10).

For scan matching an ICP (Iterative Closest Point) algorithm will be used. The ICP algorithm is iterative and consists of two steps. First, corresponding measurement positions from the observed and reference measurement positions sets are paired to form $(o_l, r_m)$ using an initial guess of the relative pose and a nearest neighbor approach where $o_l = (u_x, u_y)_o^l \in U_o$ and $r_m = (u_x, u_y)_r^m \in U_r$. Second, a new relative pose is estimated by calculating the relative pose that minimizes the least-mean-squared error between the associated measurement positions. This is iterated until the pose estimate satisfies some convergence criterion. Denote the set of all associated measurement position pairs as $A$ and $\Delta = (\Delta x, \Delta y, \Delta\theta)^T$ as the relative pose displacement. By using $\Delta$, $o_l$ can be projected onto the reference frame in which $r_m$ is described by

$$o_l^r = \begin{pmatrix} cos(\Delta\theta) & sin(\Delta\theta) \\ -sin(\Delta\theta) & cos(\Delta\theta) \end{pmatrix} o_l + \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = Ro_l + T \tag{22}$$

where $R$ is a clockwize rotation by the angle $\Delta\theta$ about the origin and $T = (\Delta x, \Delta y)^T$ a translation. The aim of ICP is to obtain the rotation and translation between two scans that minimizes the MSE (mean square error) which is defined by

$$\epsilon^2 = \min_{\Delta} \sum_{\forall(o_l, r_m)\in A} \|r_m - Ro_l - T\|^2 \tag{23}$$

Each iteration will yield a new estimation of relative pose displacement

$$\Delta^{new} = (\Delta x^{new}, \Delta y^{new}, \Delta\theta^{new})^T$$

and our initial guess is the odometry estimation of $\overline{D}_{or} = \Delta_{ini}$ For a thorough explanation of ICP and the analytical solution of the optimization problem, refer to [4].The ICP algorithm will give a new estimate of $D_{ij}$ i.e. $\overline{D}_{ij}$ for the strong links. Since it is difficult to get a good estimate of the uncertainty of $\overline{D}_{ij}$ the corresponding $C_{ij}$ will initially be a constant diagonal matrix.

### 5.2.5 Mapping and Map Conversion

The Offline-SLAM algorithm generates a set of estimated robot poses that together with their corresponding laser and sonar measurements form a data set from which maps can be constructed. Using the extracted data set one can plot every measurement in a global coordinate system forming a map consisting of points. This map is not particularly useful and needs to be converted. Two maps will be extracted, one line based map to be used in Online-SLAM and one grid map used for trajectory planning.

### 5.2.6 Line Based Map

To minimize the computation in the Online-SLAM algorithm a set of lines will be extracted from the map consisting of points as described in the previous section. These will form a line based map. Since the Online-SLAM will only use laser measurements for localization

only laser measurements from the data set will be used when constructing this map. The lines will be computed by an iterative procedure which is based on the Hough transform [7]. The algorithm is similar to the one used in [2]. The algorithm consists of the following steps.

I The Accumulator of the Hough transform will be computed for all the points now described in a global coordinate system. The Accumulator describes how many times a line on the form

$$\rho = xcos(\theta) + ysin(\theta) \tag{24}$$

has been generated by the points when inserted into the same equation together with discrete values of $\theta$ where $0 \leq \theta \leq \pi$. Extracting the values $\rho$ and $\theta$ that corresponds to the maximum value of the Accumulator gives the line that intersects the largest amount of points.

II The points that are within a distance $a_1$ from the line will be extracted.

III The extracted points will be split into a number of segments. This will be done by iteratively adding points to a segment if the distance between any of the points in the segment and the current point is less than a certain value $a_2$. If a segment consists of less than $a_3$ points the segment will be disregarded.

IV The remaining segments will be evaluated to see if and which lines should be extracted. This will be done by calculating the maximum distance between all points in a segment and extracting the two points corresponding to this value if the maximum distance is larger than $a_4$. If the distance is less than $a_4$ the segment will be disregarded. By calculating the intersection of the segment line and the lines that are orthogonal to the segment line and goes through the two extracted points, the end points of the segment line are extracted.

V The points corresponding to the remaining segments will be removed from the total set of points, i.e from the map consisting of points, and the extracted end points will be saved. The process is then iterated for the remaining points until the maximum value of the Accumulator falls below the threshold $a_5$.

Since the lines that have been extracted has to be of at least length $a_4$ and be generated by at least $a_3$ points, small objects will be filtered out.

The map that can be constructed from the extracted end points will most likely have undesired line gaps that needs to be connected. This will be done by calculating the distance between all the extracted end points and adding a new line consisting of two end points if this distance is less than $a_6$. The value $a_6$ will be low meaning that only very short lines can be generated in this step.

The value of the distances $a_1, a_2, a_4, a_6$ , the number of points $a_3$ and the value $a_5$ will be set experimentally.

### 5.2.7 Grid Map

The grid map will be constructed from both laser and sonar measurements since not all objects, e.g glass, are detected by the laser. Since this map is used for trajectory planning it will be much more detailed than its line based counterpart. This map will also be updated by the Online-SLAM algorithm.

Occupancy grid mapping is computer algorithms of generating maps from noisy and uncertain sensor measurement data, with the assumption that the robot pose is known. The basic idea of the occupancy grid is to represent a map of the environment as binary random variables. Each random variable is binary and representing the presence of an obstacle at that location in the environment. Occupancy grid algorithms compute approximate posterior estimates for these random variables. The occupancy grid is used after solving the SLAM problem and therefore take the path estimates for granted. Hence the controls and odometry data have no part in the occupancy grid since the path is assumed known.

The occupancy grid is used to estimate the posterior probability over maps given the data:

$$p(m|z_{1:t}, x_{1:t}) \tag{25}$$

where $m$ is the map, $z_{1:t}$ is the set of measurements from time 1 to t, and $x_{1:t}$ is the set of robot poses from time 1 to t. Occupancy grid represent the map as a fine-grained grid over the area in the environment that represent the map $m$. Consequently this posterior is almost impossible to calculate because an occupancy map divides the environment into finitely many grid cells.

Denote $m_i$ as the grid cell with index i, then $p(m_i)$ represents the probability that grid cell i is occupied. The thought is then to break down the problem into smaller problems of estimating:

$$p(m_i|z_{1:t}, x_{1:t}) \tag{26}$$

for all grid cells. Each of these estimation problems is then a binary problem. This breakdown is appropriate but does not enable to represent dependencies between neighboring cells. Instead, the posterior of a map is approximated as the product of each grid cell:

$$p(m|z_{1:t}, x_{1:t}) = \prod p(m_i|z_{1:t}, x_{1:t}) \tag{27}$$

This factorization makes it possible to use a binary Bayes filter to estimate the occupancy probability for each grid cell.

A very useful algorithm in grid mapping is the occupancy grid algorithm which apply the log odds representation of occupancy. The odds of a state is defined as the ratio of the probability for an event divided by the probability of its negate:

$$\frac{p(x)}{p(-x)} = \frac{p(x)}{1 - p(x)} \tag{28}$$

As a consequence, the log odds ratio is the logarithm of the odds:

$$l(x) = log\frac{p(x)}{1 - p(x)} \tag{29}$$

A belief reflects the robot's internal knowledge about the state of the environment and will be denoted $bel(x_t)$, where:

$$bel(x_t) = p(x_t|z_{1:t}, u_{1:t}) = p(x_t|z_{1:t}) \tag{30}$$

The controls and odometry $u_{1:t}$ play no role here either because the belief only depend on the measurements when the state is static. The belief is commonly implemented as a log odds ratio and denoted $l_t(x_t)$, the log odds belief at time t is given by:

$$l_t(x_t) = log\frac{p(x_t|z_t)}{1 - p(x_t|z_t)} - log\frac{p(x_t)}{1 - p(x_t)} + l_{t-1}(x_t) \tag{31}$$

where

$$l_{t-1}(x_t) = log\frac{p(x_t|z_{1:t-1})}{1 - p(x_t|z_{1:t-1})} \tag{32}$$

A slightly modified version of equation 31 will be used as the algorithm for occupancy grid mapping. The general idea of the algorithm is:

In argument: $l_{t-1,i}, x_t, z_t$

for all grid cells $m_i$ do
    if the cell $m_i$ is in the measurement $z_t$ then
        $l_{t,i} = l_{t-1,i} + InverseSensorModel(m_i, z_t, x_t) - l_0$
    else $m_i$ is outside $z_t$ then
        $l_{t,i} = l_{t-1,i}$
end iteration when all grid cells have been examined.
return $l_{t,i}$

The function $InverseSensorModel(m_i, z_t, x_t)$ implements the inverse measurement model $p(m_i|z_{1:t}, x_{1:t})$ in its log odds form:

$$InverseSensorModel(m_i, z_t, x_t) = log\frac{p(x_t|z_t, x_t)}{1 - p(x_t|z_t, x_t)} \tag{33}$$

Inverse models are often used in situations where measurements distribution are more difficult to produce than a binary state. An example of such a situation is the problem of estimating a door being open or closed using camera images. The state, open or closed, is extremely simple while the measurement model $p(z_t|x)$ is much more complex, i.e. it is easier to determine the probability of a door being open from a camera image than calculating the distribution of all camera images showing an open door.

## 5.3 Online-SLAM

This section describes how the localization and mapping is performed when the robot is operating.

### 5.3.1 Monte Carlo Localization

The robot will localize itself using a Monte Carlo Localization (MCL) algorithm. The MCL uses a collection of samples (also known as particles) to estimate the robot's pose. These particles are spread out on the predefined map which is built as described above. All particles describe a possible pose for the robot and the possibility (importance weight) that the robot actually have that pose.

The general idea of the MCL algorithm. Also illustrated by figure 10:

I The first step is to uniformly distribute all particles on the map with equal importance weights. However at first a simpler version will be implemented since the robot knows its initial pose. Therefore the particles are distributed in the neighborhood of the robots initial position. The knowledge of the robots heading will also be used.

II The robot will continuously scans the surroundings using the laser range sensor. When the robot receives new measurement data it checks if the robot has moved a distance greater than some distance $d$ or changed its heading with an angle greater than some angle $\beta$. If the condition is reached the robot will proceed to the next step or else wait for new data and redo the procedure.

III All particles poses are updated by sampling new positions from the motion model.

IV Each robot laser scan consists of 361 measurements. The 180 degree field of view is divided into N segments. Each segment is filtered to remove M measurements which has detected the shortest distances. This is to prevent that smaller objects close to the robot affect the next step where one measurement is extracted from each segment. The chosen measurements will originate from the part of each segment where the variance between the adjacent samples are the smallest. This is done to prevent samples which measures smaller objects from being chosen. The N samples' measured distances and their corresponding angles are stored. Then each particle simulates N sensor measurements in the same angles as the previously extracted samples. The simulated laser rays are vectors originating from the particles location. When they intersect with a line segment in the map the measured distance from the particle to the intersection are stored.

V The N measured distances for each particle are subtracted with the M actual measured distances. The error between the measurements is accumulated and are used to determine the importance weight for the particles. The weight for each particle will be the inverse of the error divided by the total error of all particles so that the importance weights sum up to one. If the particles measurement data is similar to the actual data the environment for both the particle and the robot are similar therefore the importance weight increases. In the case when the measurement data are very different the importance weight decreases.

VI A new set of particles is created. By sampling(with replacement) from the current set the new particles will be distributed according to the importance weights of the current particles. More particles will be sampled from the particles with high importance weight since they have a higher probability.

VII This new set will replace the old set of particles and all particles will have an equal importance weight. After this the whole procedure will be repeated from step II.

By using this iterative process the particles will eventually gather where the most probable location of the robot is. Still they will not converge into a single point so in order to make a point estimation the first approach will be to determine which K particles that have the highest importance weights. Around each particle a circle with a radius R originating from the particle is created. Then the weights from all particles within the circle will be accumulated. The circle that have the highest sum will be used and a mean of the particles poses is calculated to determine the estimated pose of the robot.

As already mentioned, the map used for localization will consist of a set of lines. The reason for this choice is to make the MCL as efficient as possible. By using a line based map, the environment can be described by a relative small set of objects without loss of resolution. Also, since the only measurements used for localization arise from the laser sensor and since a laser ray can be described as a line, the simulation of the measurements will be simple. However, as mentioned earlier, the map used for localization will most likely not contain small objects and therefore it is important that measurements arising from small objects are detected and filtered out as described in IV.

### 5.3.2 Online Mapping

The online mapping will differ from the offline mapping previously described. The map needs to be updated when new objects are located and objects that are no longer present should be removed from the map. Another difference is that not only the laser will be

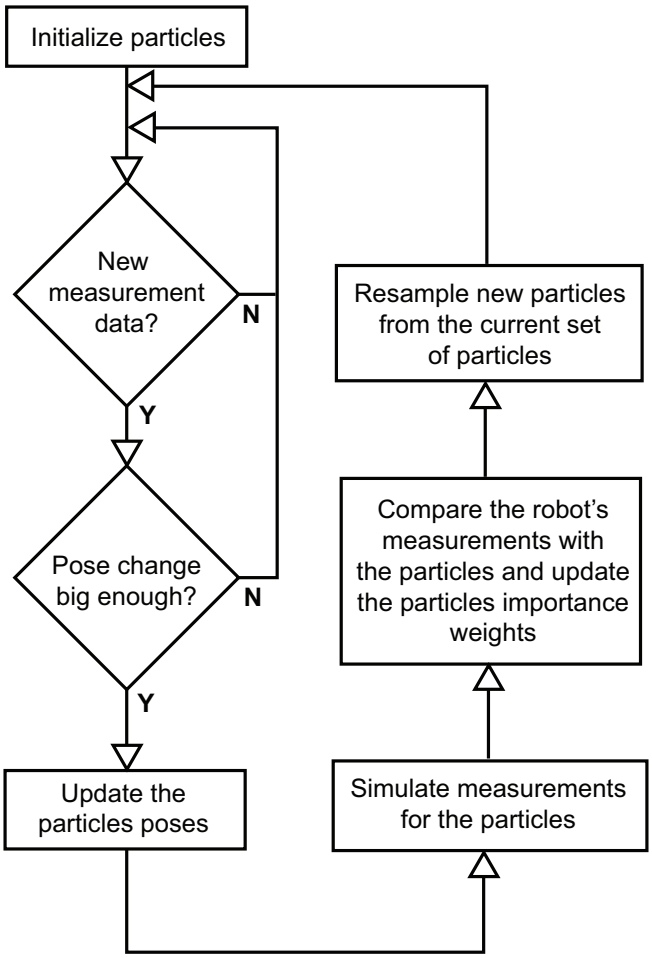


Figure 10: The MCL algorithm

used for online mapping, but also the sonars. The reason for this is that the robot needs to detect as many obstacles as possible in order to avoid collisions. The ambition is that the robot should be able to update the map dynamically as it moves in the office so that it always have access to an updated map.

However the first approach will be a little more restrictive. The idea is that the robot will have two different sets of maps. One described by line segments which the MCL will use and another grid based map that will be used by the control unit to navigate and calculate the shortest path.

The grid map needs to be continuously updated if changes in the map are detected so that the control unit knows about the environment. The line based map used by the MCL will not be updated. The reason for this is that the main objects in the line based map, e.g. walls, will not likely be moved or removed. Therefore, the big difference between the line based map and the reality will be dynamic objects such as people, smaller objects such as tables and chairs and objects that the laser can not detect, e.g. glass.

This is why the algorithm that chooses the angles, which the particles simulate in MCL, has to be used to filter out the possible outliers caused by objects not included in the map. However the algorithm will not be able to detect all dynamic objects which will cause the robot to be more uncertain of its pose at those locations. Since the MCL is used,

the particles will spread out over a larger area when the robot moves in an environment where the map differs from the reality. But when it exits that area the particles will gather around the robot again making the uncertainty of the pose smaller.

# 6 Trajectory planning

The purpose of this subsystem is to find a route from the current position to a given point in the map. If a user has defined forbidden zones in the map the program should consider this when planning the route. The trajectory planner gets a map and robot pose (position and heading) from the SLAM system and sends a route to the control system.

## 6.1 Pathfinding

The robot will employ the Dynamic A* Lite (D* Lite) pathfinding algorithm, an incremental heuristic search algorithm [5]. The A* algorithm is a commonly used search algorithm based on Dijkstra's algorithm but with a heuristic approach to achieve a faster computation time [10]. However, it does not cope well with a large dynamic environment, through which the robot will be needed to navigate, since a changed path always needs to be recalculated from scratch. This could be solved by using the Dynamic A* algorithm (i.e. D*) which adds an incremental enhancement to the A* algorithm to reduce the computation time. By reusing old information when detecting new obstacles, only the affected part of the path needs to be recalculated [12]. The D* Lite algorithm, though technically not based on the D* algorithm, shares its functionality and performance but is far easier to implement than the original D* algorithm [5].

## 6.2 The D* Lite algorithm

The algorithm will be used on a supplied grid map, where each cell corresponds to a node. The cost of moving between two adjacent nodes is always one, as long as the movement is possible. If a cell is blocked by an obstacle, the cost of traveling to the node it represents will be set to infinity. The same applies for a node to which the robot does not yet know a path.

### 6.2.1 Definitions

The following notations are used throughout the description.

1. $s$ refers to an arbitrary node.
2. $s_{i,j}$ refers to a node $s$ at position $i, j$ in the grid.
3. $c(s', s)$ is the cost of moving from node $s'$ to node $s$.
4. $g(s)$ is the cost of reaching node $s$ through node $s'$ and can be defined as $g(s) = g(s') + c(s', s)$.
5. $rhs(s)$ is the one step look ahead cost of reaching a node $s$ and is equal to $g(s)$ as long as the node is consistent. If $g(s) \neq rhs(s)$ the cost of reaching the node has changed (i.e. a new obstacle has been detected or an old one has disappeared) and the path needs to be recalculated [11].
6. $h(s', s)$ is the heuristic cost of reaching node $s$ from node $s'$. Here, this corresponds to the shortest path with no consideration of walls or obstacles.
7. $k_m$ is the key modifier value, used when recalculating the path.
8. $k_1$ and $k_2$ are the two key values each node is given. They determine in which order the nodes will be searched and are combined into a two component vector.

9. $U$ is the priority queue for node keys and is implemented as a stack, where the keys are stored in order. The keys will be added or removed during the computation under given conditions.

### 6.2.2 Initialization

The initial layout of a map is exemplified in figure 11. Each cell is associated to a heuristic value which corresponds to the shortest path from the start node $s_{0,1}$ with no consideration of walls or obstacles [3].

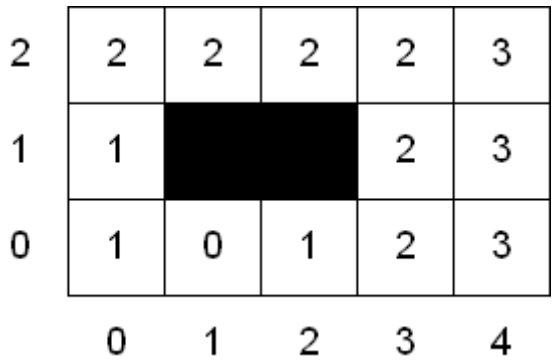


Figure 11: An example map with heuristic values

The initialization will be done by the function $Initialize()$, called by the function $Main()$ [5]. The algorithm performs a backward search; thus the g-value and the rhs-value for each cell is set to infinity, except the rhs-value of the goal node $s_{4,2}$ which is set to zero. The goal node is also mapped with its key. The key for a node $s$ is a two component vector and is used to determine the order in which the nodes will be searched. It will be calculated with the function $CalculateKey(s)$ for a node $s$ [5]

$$\begin{pmatrix} k_1 \\ k_2 \end{pmatrix} = \begin{pmatrix} min(g(s), rhs(s)) + h(s_{start}, s) + k_m \\ min(g(s), rhs(s)) \end{pmatrix}. \tag{34}$$

The $g(s)$ function calculates the cost of reaching node $s$, the $rhs(s)$ function calculates the one step look ahead of reaching node $s$, the $h(s_{start}, s)$ function gives the heuristic cost from the start node to node $s$ (According to figure 11, this value is initially 3 for the goal node). The key modifier variable $k_m$ is used later on when determining keys for subsequent nodes after changes in edge costs and is initially 0 [5]. After calculation, as a part of the initialization process, the key for the goal node $s_{4,2}$ will be saved in the queue $U$. This is represented in the figure by showing the key-value in the cell. Figure 12 shows the example map after initialization [3].

### 6.2.3 First time computation of the shortest path

The path will be calculated by the function $ComputeShortestPath()$ which will be called by the function $Main()$. The function will run as long as the lowest priority key in the queue is lower than the key of the start node, or as long as the start node is locally inconsistent.

It will then pick out the lowest priority key of the queue and the lowest priority node (at the same time removing it). Then one out of three scenarios will unfold:

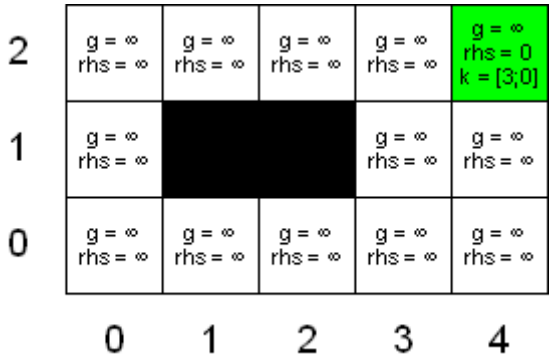


Figure 12: The example map after initialization

1. If the picked key is lower than a new calculated key of the acquired node, the node will be inserted once again in the queue $U$ together with this key.
2. If the acquired node is found out to be over consistent (its g-value larger than its rhs-value) the rhs-value will be set as its g-value and the function $UpdateVertex(s)$ will be run on all nodes $s$ preceding the node.
3. If none of these scenarios apply, the g-value will be set to infinity for the node and the function $UpdateVertex(s)$ will be run on all nodes preceding the node.

The function $UpdateVertex(s)$ updates a node $s$ under three different conditions:

1. If the node is not the goal-node, the rhs-value for the node will be calculated.
2. If the node $s$ is in the queue $U$, it will be removed.
3. If the node's g-value differs from its rhs-value (i.e. it is inconsistent) it will be added to the queue $U$ along with its associated key.

Seen in figure 13 are the five steps of the first time path computation. In step 1, the three nodes neighboring the goal node have all had their rhs-values and keys calculated and have subsequently been added to the queue $U$. Since both node $s_{3,2}$ and $s_{3,1}$ have the same key value the order in which they will be examined is irrelevant. In this example, the node $s_{3,1}$ is chosen and will have its g-value calculated and be removed from the priority queue $U$. This is shown to have happened it step 2, together with the calculation of the key and rhs-value of all nodes neighboring node $s_{3,1}$. The same procedure continues through step 3 and 4. In step 5 the algorithm terminates because the start node (i.e. the "goal" node) has been found.

### 6.2.4 Dynamic re-calculation of the path

As the robot moves from node $s_{1,0}$ to $s_{2,0}$ the function $Main()$ will reset the new node as the start node and update the heuristics, as according to figure 14. The goal node is still node $s_{4,2}$. For the sake of the example, a new wall segment seems to have appeared in the robot's path [3].

In figure 15 the robot has moved to its new node and the scan for changed edge costs yields a positive response. The key modifier value $k_m$ is calculated as $k_m = k_m + h(s_{last}, s_{start})$

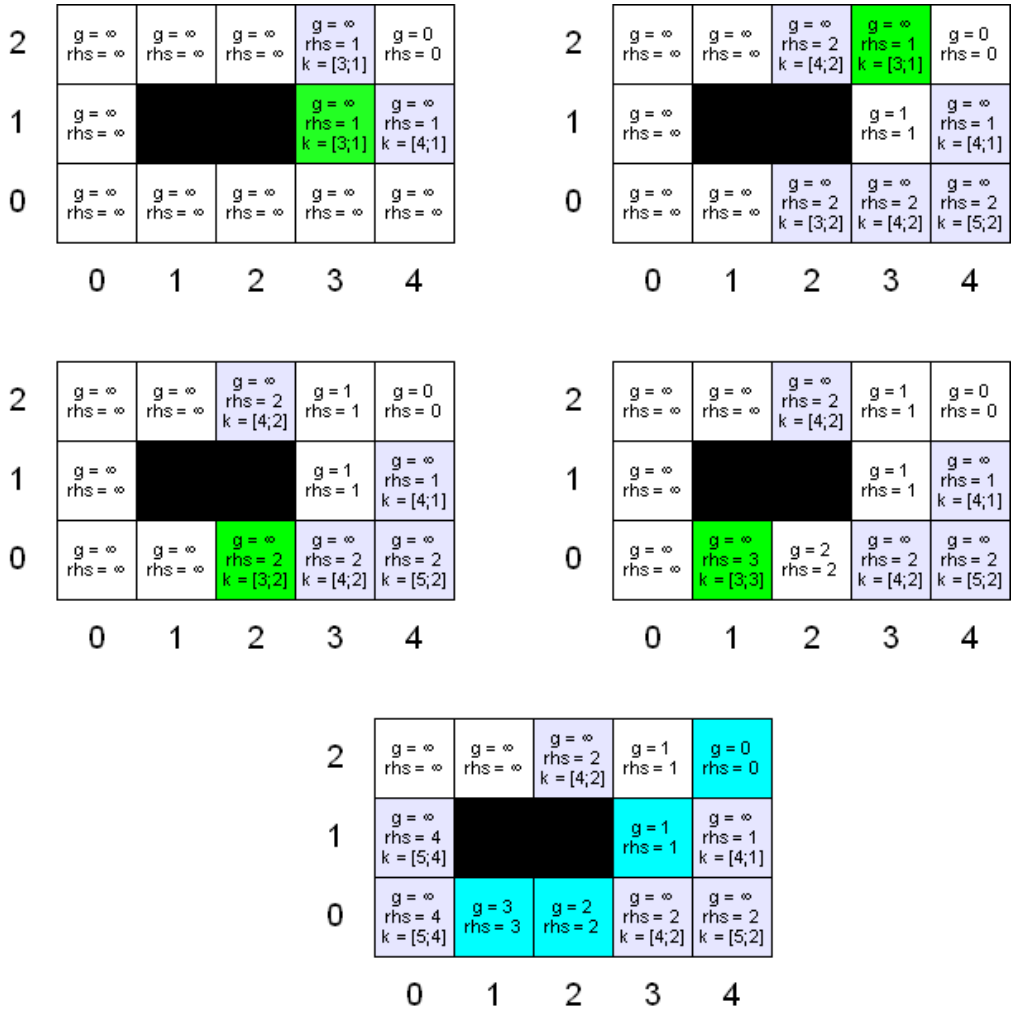


Figure 13: Step 1 to 5 of first time path calculation

where $h(s_{last}, s_{start})$ is the heuristic value between the last node visited and the new node, which is set as the new start node.

In figure 16 values for the nodes with changed edge costs are updated and a new traversable path is calculated accordingly [3].

### 6.2.5 Practical implementation

When implemented, each cell in the grid map will correspond to a 10 by 10 cm square, in order to achieve a high resolution. In a two meter wide and 200 meter long corridor, this will result in approximately 20 000 cells. It is still an open question weather or not this is too computational heavy (both for the algorithm and the system as a whole). Should this be the case, it is simple to just acquire a map update every fifth cell or so and only do a recalculation with a new map update if the safety sensors detects an obstacle.

The actual movement between the cells will be actuated by the control system.

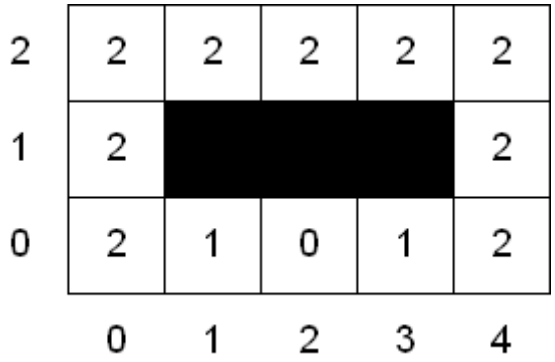


Figure 14: New heuristics after robotic movement

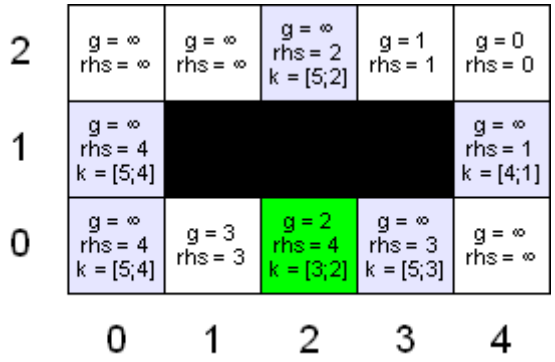


Figure 15: New edge costs after robotic movement

# 7 Control

The path finding algorithm will decide which grid in the grid map to go to. This will then be translated to a position in the robot's coordinate system and the control system will make the robot go there. When the robot has reached this position it will ask the path finder where to go next. While moving, the robot will also send data from the range finders to the SLAM system.

The ActivMedia Robotics Interface for Application (ARIA) library [13] is used for all communication with the robot.

## 7.1 Robot Interface

The following ARIA classes and methods will be used.

- ArRobot is used for communication with odometers, sonars and motors
  - getX(), getY(), getTh() returns an estimation of the robot's pose determined by odometry
  - moveTo() updates the robot's idea of its pose with the values from Online-SLAM
  - setVel() sets the translational velocity

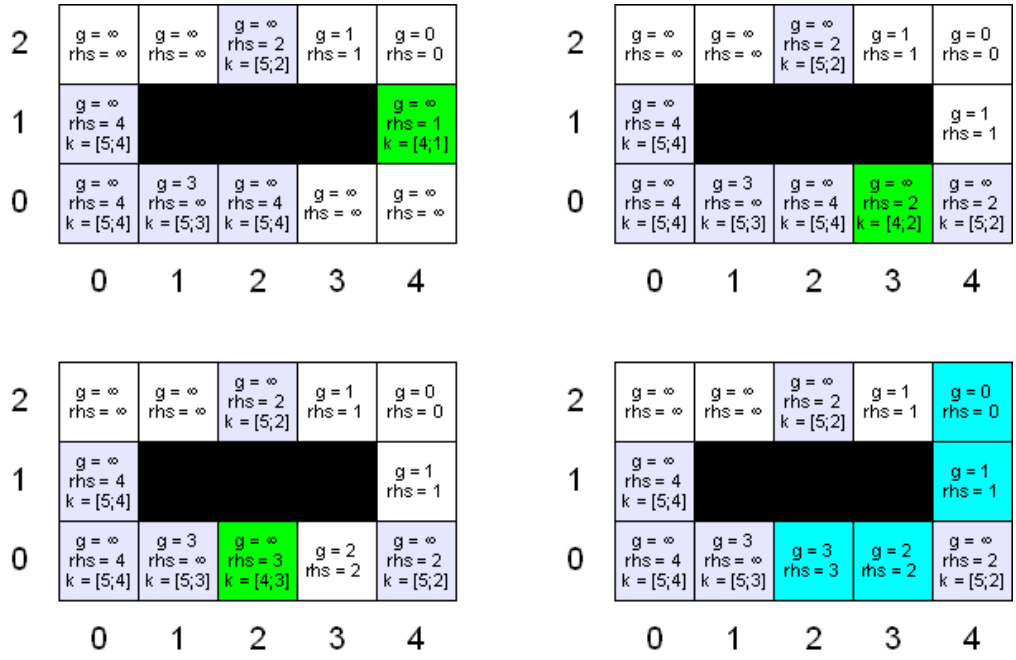


Figure 16: Step 1 to 4 of dynamic path re-calculation

  - setRotVel() sets the rotational velocity
  - addAction() adds an action with a priority
  - getSonarReading() returns the reading from a given sonar
- ArSick is used for communication with the laser range finder
  - getRawReadings() returns a list with 361 laser readings from the latest scan
- ArPose contains an x and y postition as well as a heading
- ArSensorReading contains the range of a laser or sonar reading together with the pose from where the reading was taken
- ArActionGoto takes a position and sets the wheel velocities to make the robot go to the given position
- ArActionAvoidFront makes the robot slow down, turn or stop if any of the range finders detects something in front of the robot

The central class of the interface is ArRobot. It has methods for receiving odometry and sonar data as well as sending commands to the motors. The method getPose() returns an ArPose object which contains the robot's position and heading determined by odometry. ArPose also has helper functions for calculating distance and angle to another pose. Both ArRobot and ArPose have methods getX(), getY() and getTh() for extracting the individual values. These values, together with the covariance matrix calculated according to section 4, will be sent to the Online-SLAM. When localization have been performed in Online-SLAM and a more accurate pose is available, moveTo() is used to update the robot's idea of its pose so that the next pose estimation from the odometers will depend on the pose from localization and the movement since localization.

To get the robot to move, setVel() and setRotVel() are used to set the translational and rotational velocities. Alternatively setVel2() can set the velocities of the left and right wheel, individually. To make the robot go to a given position an ArActionGoto object is created and added to the robot with addAction(). This action takes the desired x and y coordinates as parameters and moves the robot there. Another action, ArActionAvoidFront, is added to avoid obstacles in front of the robot. This action uses readings from both the laser and the sonars to determine if there is an object in front of the robot. If it detects something within 20 cm the robot will stop. In case the object disappears within 3 seconds, the robot will continue moving. Otherwise the grid map will be updated and the path replanned. Each action added to the robot is given a priority. The avoid action will have a higher priority than the goto action.

ArRobot also provides the sonar range data through getSonarReading(). ArSick is the class that handles laser data. It has methods that return the closest current reading in a region, currentReadingBox() and currentReadingPolar(). To get a list of all current readings getRawReadings() is used, which returns a list of ArSensorReading's. The range of these readings will be sent to the SLAM algorithms.

The robot's physical dimensions and some other parameters that are used in the program are stored in a file (p2d8.p) which is loaded on initialization.

All numbers are of type double. The units are millimeters for lengths, millimeters per second for velocities and degrees for angles.

# 8 Documentation and coding conventions

Stylistic rules for documents and code to ensure a cohesive appearance.

- **Documents**
  These rules only apply for documents that make up the project documentation. Other documents, e.g. presentations and posters, are not covered by these rules.

  1. All documentation must be written in Latex and published using the lips_eng.sty Latex style sheet file [6].
  2. All headlines must be followed by a short descriptive text of the section in question.

- **Coding**
  The standard C++ coding convention is used, as described by the Robotics Interface for Application (ARIA) documentation [13].

# References

[1] Kai Oliver Arras. An introduction to error propagation. http://www.nada.kth.se/~kai-a/papers/arrasTR-9801-R3.pdf(2010-10-04). [Online; accessed 02-oct-2010].

[2] W. Burgard D. Sack. A comparison of methods for line extraction from range data. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.118.629&rep=rep1&type=pdf. [Online; accessed 06-Oct-2010].

[3] C. Erogul. D*-lite path finding algorithm and its variations. http://www2.ceng.metu.edu.tr/~erogul/ceng585/13_DEC_2007_Presentation/d-star-hidden-slides-deleted.pdf. [Online; accessed 6-oct-2010].

[4] J. Morales A. Mandow J. L. Martínez, J. González and A. J. García-Cerezo. Genetic and icp laser point matching for 2d mobile robot motion estimation. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.167.9279&rep=rep1&type=pdf. [Online; accessed 06-Oct-2010].

[5] S. Koenig and M. Likhachev. D* lite. http://idm-lab.org/bib/abstracts/papers/aaai02b.pdf. [Online; accessed 26-Sep-2010].

[6] T. Svensson;C. Krysander. *Projektmodellen LIPS (2007)*. Linköping University, 2007.

[7] K. Lee. Application of the hough transform. http://www.cs.uml.edu/~lkyewook/hough/APPLICATION%20OF%20THE%20HOUGH%20TRANSFORM%20V1.9.pdf. [Online; accessed 06-Oct-2010].

[8] F. Lu*; E. Milios. Globally consistent range scan alignment for environment mapping. http://cogvis.nada.kth.se/hic/~SLAM/Papers/lu_milios.pdf. [Online; accessed 01-oct-2010].

[9] Austin I. Eliazar;Ronald Parr. Learning probabilistic motion models for mobile robots. http://delivery.acm.org/10.1145/1020000/1015413/p284-eliazar.pdf?key1=1015413&key2=9668716821&coll=GUIDE&dl=GUIDE&CFID=107205476&CFTOKEN=75751825. [Online; accessed 28-Sep-2010].

[10] A. Patel. Amit's a* pages. http://theory.stanford.edu/~amitp/GameProgramming/. [Online; accessed 5-oct-2010].

[11] A. Sotelo. D* lite and dynamic pathfinding. http://www.adriansotelo.com/Research/Dstar%20Lite.pptx. [Online; accessed 6-oct-2010].

[12] T. Stentz. Real-time replanning in dynamic and unknown environments. http://www.frc.ri.cmu.edu/~axs/dynamic_plan.html. [Online; accessed 5-oct-2010].

[13] Dimitri van Heesch. Aria reference manual. http://www.control.isy.liu.se/~andrecb/fidodido/doc/AriaReference.pdf. [Online; accessed 29-sep-2010].