# Design of an Autonomous Office Guide

*Master's Thesis in Systems, Control and Mechatronics*

GUSTAF AGRENIUS

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2012
Master's Thesis EX081/2012

## Abstract

In this master thesis a guiding software for an autonomous robot was developed. The purpose was to enable a preexisting robot to guide visitors to the right office space inside of Aros Electronics. The guiding software have four major parts; *path planning*, *path following*, *collision avoidance* and *localization.* The A* algorithm is used on a map of the office to get the path to the destination. The path following is done by driving the robot towards a waypoint located on the path ahead of the robot. The collision avoidance checks that the path is clear and if not the robot is not allowed to continue. Two ways of localization is tested in combination with odometry. First an Xbox kinect camera is used to spot landmarks and the distance and angle to this landmark is used in a probabilistic method called particle filter to estimate the pose. Secondly a wall following method is developed where ultrasonic proximity sensors are used to sense the walls of the office. Both localization methods greatly improves the pose estimation compared to using only odometry but the wall following method shows better performance compared to particle filter method. The main reason for this is that the update frequency is much higher.

***Keywords:*** Autonomous robot, autonomous guiding, path planning, path following, collision avoidance, localization, landmark, kinect camera, particle filter, A* algorithm, wall following.

## Acknowledgment

This master thesis was made at the Department of Signal and Systems in cooperation with Aros Electronics. I would like to thank every one at Aros Electronics for all their support. I especially want to thank my two advisors Henrik Gulvén and Andreas Petterson for helping me with ideas and motivation when needed. Also a special thanks to Daniel Chädström for helping me understand the inner workings of Glenn and to Henrik Neugebauer for helping me save money at lunch time. At Chalmers I would like to thank my examiner Jonas Fredriksson for ideas and input. At last I want to thank Lena for proofreading this text and for always being there.

Gustaf Agrenius, Gothenburg 12/11/28

# Contents

# Variables

A list of variables and abrevations used.

| | |
|---|---|
| PFL | particle filter localization |
| CAN | Controller Area Network |
| $d_{CAN}$ | distance traveled by Glenn |
| $\alpha_{CAN}$ | angle of Glenn |
| $RW_{pos}$ | angular position of right wheel |
| $LW_{pos}$ | angular position of left wheel |
| $\mathbf{p}$ | edge of the map polygon |
| $\mathbf{P}_a$ and $\mathbf{P}_b$ | vertices of $\mathbf{p}$ |
| $\mathbf{c}$ | vector defining the movement of the map polygon edges |
| $\mu$ | the distance to move the map polygon edges |
| $\Omega$ | the direction of $\mathbf{c}$ |
| $\mathbf{d}$ | vector orthogonal to $\mathbf{p}$ |
| $\mathbf{Q}_a$ and $\mathbf{Q}_b$ | points on the contracted map |
| $f(n)$ | the cost of the path through $n$ |
| $g(n)$ | the cost to the current node $n$ |
| $h(n)$ | the heuristic cost from the current node $n$ to the goal |
| $(x_p,y_p)$ | the position of the current waypoint $p$ |
| $d_p$ | distance to the current waypoint $p$ |
| $\alpha_p$ | angle to the current waypoint $p$ |
| $\mathbf{x_k} = [x_k,y_k,\theta_k]^T$ | the pose of Glenn at time $k$ |
| $\hat{\mathbf{x}}_\mathbf{k} = [\hat{x}_k,\hat{y}_k,\hat{\theta}_k]^T$ | the pose estimate of Glenn at time $k$ |
| $\delta\theta_k$ | the change in heading since the last time step |
| $\delta d_k$ | the change in distance since the last time step |
| $\delta r_k$ | the distance traveled by the right wheel since the last time step |
| $\delta l_k$ | the distance traveled by the left wheel since the last time step |
| $W$ | wheelbase of Glenn |
| $D_w$ | wheel diameter |
| $e_{drift}$, $e_{trans}$ and $e_{rot}$ | odometric error correction |
| $\alpha_k$ | azimuth angle to the current landmark |
| $L_k$ | the distance to the current landmark |
| $resX$ | horizontal resolution of the kinect RGB camera |
| $FOV$ | field of view of the kinect RGB camera |
| $Z_{kinect}$ | kinect depth measurement |
| $X_L$ | the x position of center pixel of the current landmark |

| | |
|---|---|
| $W_i$ | weight of particle $i$ |
| $\delta L_i$ | the difference of the actual kinect depth readings and the expected kinect depth readings for particle $i$ |
| $\delta\alpha_i$ | the difference of the actual kinect angle readings and the expected kinect angle readings for particle $i$ |
| $(x_M,y_M)$ | the position of the current landmark $M$ |
| $\mathbf{x_i} = (x_i,y_i,\theta_i)$ | the pose of particle $i$ |
| $\sigma_L$ and $\sigma_\alpha$ | tuning parameters for the particle filter |
| $S$ | the difference between measured distance and expected distance to the wall |
| $\varphi$ | the angle error of Glenn |
| $D_i$ | mean deviation in Y direction of Glenn at position $i$ |
| $T_i$ | mean deviation in X direction of Glenn at position $i$ |
| $G_i$ | distance traveled at position $i$ |
| $R_i$ | mean angle deviation at angle $i$ |
| $A_i$ | actual angle $i$ |

# Chapter 1

# Introduction

This thesis describe the development of an autonomous guiding functionality in a preexisting robot, called Glenn. The desired behavior of Glenn is to get user input in the form of a desired position, where to guide someone, and then drive to that position. After the task is complete Glenn should return back to the starting position. This thesis is made together with Aros Electronics and it is in their office Glenn should be able to guide people.

## 1.1 Autonomous Guiding

Autonomous robots are an increasingly common part of our every day life. They exist in both the industry, as unmanned trucks, and in our homes, as autonomous vacuum cleaners or lawnmowers. Autonomous guiding robots are not very common today but in a near future they will probably be a well known sight at many museums.

To achieve a guiding functionality in an autonomous robot four tasks have been identified that need to be performed. First the path must to be planed (task 1: *path planning*) between the current position and the desired goal and for this the robot need to have a map of the environment. When the path is obtained the robot must be able to follow it (task 2: *path following*) without any collisions (task 3: *collision avoidance*). For any of this to work the robot must know where it is located in the environment (task 4: *localization*).

## 1.2 Related Work

A lot of work exists in the area of autonomous robots. One example is RHINO, a guiding robot that was tested for six days in the Deutsches Museum Bonn [1]. RHINO was equipped with many different sensors, infrared, sonar, lasers and tactile. As a map they used an probabilistic occupancy grid and planned the robots path with value iteration, this is a good way in a dynamically changing environment like a museum with a lot of visitors. They used markov localization to estimate the pose which is a probabilistic approach. They achieved a success rate of 99,75% during the time RHINO was deployed.

Another good example is [2] where the authors develop a differential steered robot that have a map available of the environment and plan its path using the dijkstra's algorithm. They use a laser range finder for localization and to avoid obstacles. They show good results at long term navigation but suggests improvements that would be able to handle failures of the robot.

In [3] and [4] two similar ways of path following and collision avoidance are described. The robot drives towards a point located on the path some distance ahead of the robot. When an obstacle is detected a reactive control makes the robot follow the edge of the obstacle until it reaches the path on the other side and can continue.

Landmark localization is a common way to estimate a robots pose. [5] and [6] both describes ways to estimate a pose by spotting landmarks in the environment. For the calculations to work at least three known landmarks are needed. The idea to use a particle filter localization (PFL) is from [7] where a kinect camera is used to spot landmarks. They test two ways to fuse the kinect and odometric meassurements, extended Kalman filter and particle filter. They reach the conclusion that the particle filter gives the best localization. This approach have the advantage that only one landmark needs to be recognized to estimate the pose.

Particle filter localization is also known as Monte Carlo localization and is a way of probabilistically estimate the pose of a robot. Some different but similar algorithms for the PFL are described in [8], [9] and [10].

## 1.3 Glenn

The robot used in this project is called Glenn (see figure 1.1). It was developed at AROS Electronics and is used as a demo at fairs. Glenn is an autonomous two wheeled self-balancing robot, much like a Segway, but without a driver.

The robot Glenn mainly consists of parts that are designed and produced by Aros Electronics. The balance control is performed in the control board and uses a Gyro (ADXRS610) and an Accelerometer (ADXL203EB) to sense the angle. The motors are controlled by two motor drives and each wheel axis is equipped with an encoder to be able to sense the wheel position and speed. There are 6 ultrasonic proximity sensors (Devantech SRF01) fitted at the front of Glenn that are used for collision avoidance and wall following, these are controlled by the sensor board. To be able to spot landmarks there is an Xbox 360 kinect camera. The guiding software is run on a PC (Zotac) which is equipped with a touchscreen, speakers and a keyboard. Glenn is powered by two Li-Po battery packs and the power board and PC power board are used to distribute the power. The communication between the different parts is done over a CAN bus with a USB/CAN converter to connect the PC. In figure 1.2 a diagram over the different parts of Glenn is shown and in appendix A there is a detailed list of the different parts.

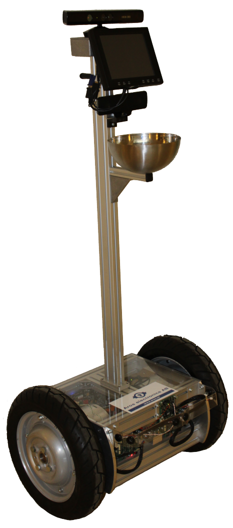

**Figure 1.1:** The robot Glenn

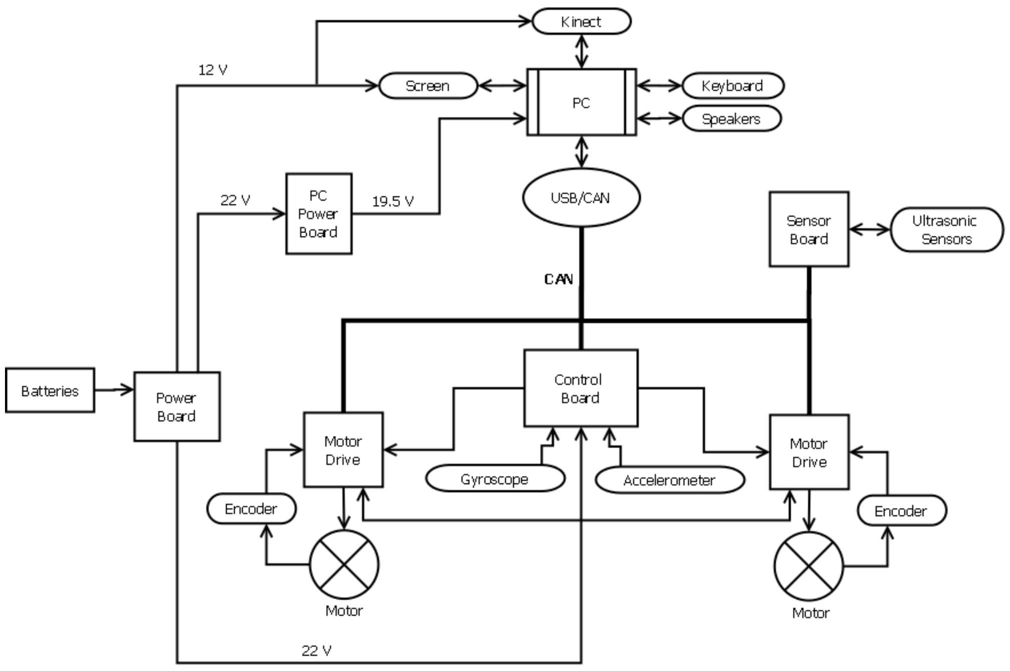

**Figure 1.2:** Diagram of the different parts for the robot Glenn

### 1.3.1 CAN Communication

CAN stands for Controller Area Network and was developed for the automotive industry in the mid-1980s. It is a serial communication protocol that enables for the different nodes in a system to communicate. For further reading see [11].

### 1.3.2 Sensor Board

The sensor board continuously fires the ultrasonic sensors and update the measured distance. To get the sensor readings a request is sent to the sensor board over the CAN bus and a response is returned with the latest reading for each sensor in cm.

### 1.3.3 Control Board

To move Glenn, a desired distance, $d_{CAN}$, and angle, $\alpha_{CAN}$, are sent over the CAN bus and a response with the current distance and angle is returned. This position and angle is measured from the original wheel position when Glenn is powered up according to

$$\begin{aligned} d_{CAN} &= \frac{RW_{pos} + LW_{pos}}{2} \\ \alpha_{CAN} &= \frac{RW_{pos} - LW_{pos}}{2} \end{aligned} \tag{1.1}$$

Where $RW_{pos}$ and $LW_{pos}$ is the angular position of the right and left wheel respectively.

## 1.4 Software Structure

The structure of the guiding software in Glenn is presented in a flow chart in figure 1.3. When the software is started it initializes Glenn with a known starting pose and start the localization process. In this project two different ways of localization is developed and used together with odometry. The first based on landmark detection using an Xbox kinect camera to spot the landmarks and a particle filter to estimate the pose. The other is a wall following approach that uses ultrasonic sensor to sense the walls and from that estimate the pose.

After the software is initialized it waits for a user input in the form of a desired goal. When a desired goal is given the path planning first creates the path. Then the path following process is started which drives Glenn to the desired goal. The path following process also performs the collision avoidance. When the goal is reached Glenn checks if it should return, which is possible to decide beforehand. If yes then the starting point is set as the desired goal and sent to the path planning and the process is repeated. The different tasks are described in more detail in the following chapters.

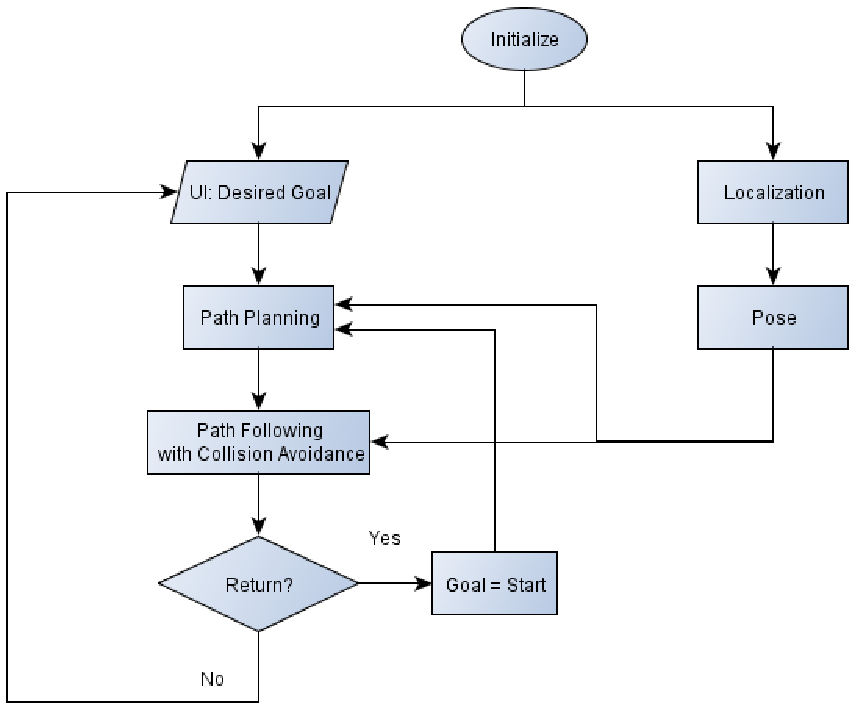

**Figure 1.3:** The flowchart of the software structure

## 1.5 Outline of the Report

In this chapter the main goal and the different tasks where described together with related work studies. Also the robot Glenn and the software structure was introduced. The following chapters describes the different tasks needed for autonomous guiding. Chapter 2 describes the path planning algorithm. In chapter 3 the path following and collision avoidance are described. The localization of Glenn is covered in chapter 4 where two different alternatives are described. First particle filter localization (PFL) and then wall following. Chapter 5 shows the results of the guiding and localization and in chapter 6 the results are discussed and future work and improvements are suggested.

# Chapter 2
# Path Planning

For path planning the A* Algorithm is used which is a heuristic best-first search algorithm [12]. The path planning also requires a map of the environment to define where Glenn is allowed to travel.

## 2.1 Contracted Map

The map is represented by the corner points in the office which defines a polygon where the inside is the allowed space. This area is in reality to big because the position of Glenn is a single point located in the middle of the wheel axis and therefore can not be closer then half of Glenns width to a wall i.e the polygons sides. To handle this a *contracted map* is created where all the sides is moved inwards. This is done by using a method from [2] where for each side of the polygon $\mathbf{p} = (p_x,p_y) = \mathbf{P}_b - \mathbf{P}_a = (x_b - x_a,y_b - y_a)$, where $\mathbf{P}_a$ and $\mathbf{P}_b$ are the vertices's of that side, a vector $\mathbf{c}$ is defined as

$$\mathbf{c} = \mu\Omega\frac{(-p_y,p_x)}{\sqrt{p_x^2 + p_y^2}} \tag{2.1}$$

where $\mu$ is the desired length with which to move the sides and $\Omega = \text{sgn}((\mathbf{p} \times \mathbf{d}) \cdot \hat{z})$ defines the inwards direction by either -1 or 1 given the unit vector $\mathbf{d}$ orthogonal to $\mathbf{p}$ and pointing towards the allowed space. The sign of $\Omega$ depends on which direction the polygon sides are defined, either clockwise or counterclockwise. Now two points on the contracted map side can be defined as

$$\begin{aligned} \mathbf{Q}_a &= \mathbf{P}_a + \mathbf{c} \\ \mathbf{Q}_b &= \mathbf{P}_b + \mathbf{c} \end{aligned} \tag{2.2}$$

These points are not necessarily corner points of the contracted map. Instead the intersections of each neighboring line defined by these new points are taken as the corner points for the contracted map. In figure 2.1 an example of how a contracted map is created can be seen. The solid line represents the original polygon and the dashed lines are the new lines after equation 2.2 is applied. The black dots shows the corner points of the new contracted polygon. The map of the office can be seen in figure 2.2 where the solid line is the original map and the dotted line is the contracted map.

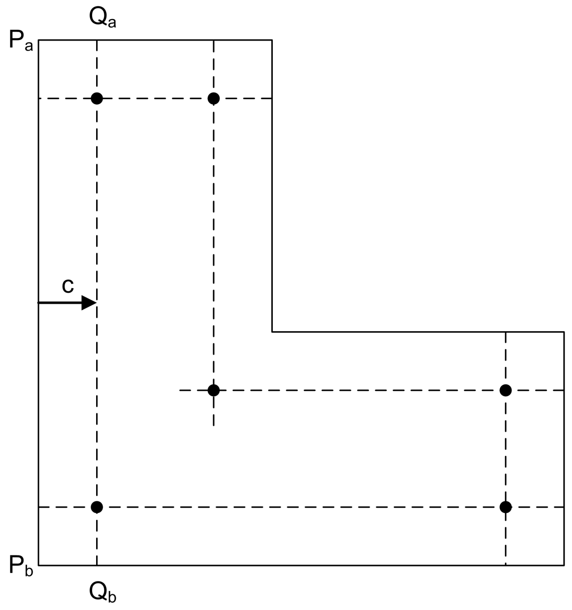

**Figure 2.1:** An example of the creation of a contracted map. The solid line is the original map and the dashed lines are the lines after they are moved. The black dots is the new corner points of the contracted map.

## 2.2 A* Algorithm

A* is a widely used best first search algorithm which uses the function $f(n) = g(n)+h(n)$ where $g(n)$ is the cost, in this case the distance, to reach the current node $n$ and $h(n)$ is the estimated cost of reaching the goal from the current node. $f(n)$ then becomes the estimated cost of the cheapest solution through $n$ [12]. The A* algorithm operates by taking the neighboring nodes of the node with the lowest $f(n)$ and after checking that they are inside the constricted map, calculate their $f(n)$ values, add them to the set of nodes to be evaluated and remove the current one. Each node has eight neighboring nodes that are created in each iteration by adding or subtracting 100 mm to the X and Y position. This is repeated until the goal point is reached.

Then a recursive function recreates the path taken by the algorithm and returns it. $h(n)$ is called a heuristic and can be estimated in different ways. Here it is taken as the shortest straight line distance to the goal. The pseudo code for the A* algorithm is displayed in algorithm 1 in appendix B.

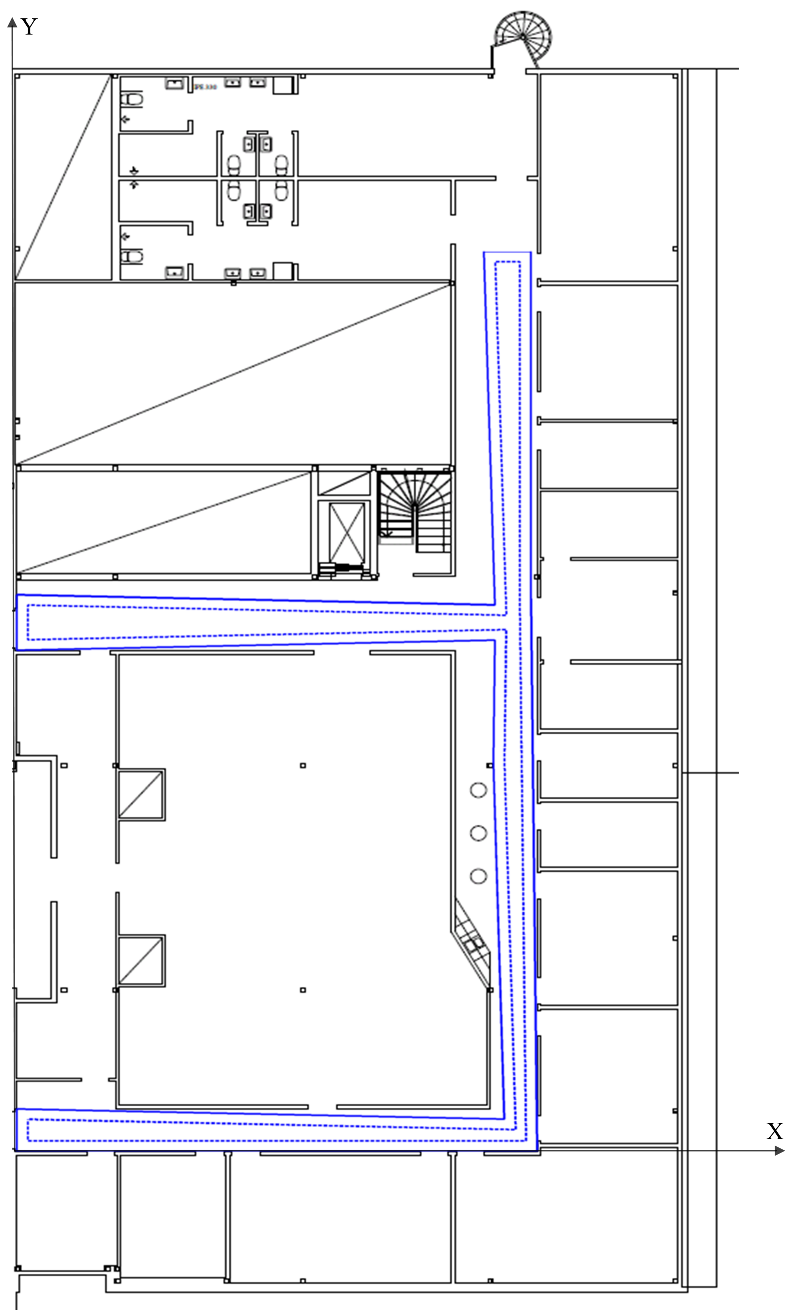

**Figure 2.2:** The map of the office with the original map (solid line) and the contracted map (dotted line)

# Chapter 3

# Path Following

The A* algorithm returns a path consisting of discrete points. Glenn follows this path by driving towards a waypoint ($x_p$,$y_p$) located a certain distance in front of the current position ($x_k$,$y_k$) as displayed in figure 3.1. The drive commands controlling Glenn is an angle and a distance according to equation 3.2. The distance, $d_p$, and angle, $\alpha_p$, between ($x_k$,$y_k$) and ($x_p$,$y_p$) is added to the current distance and angle of Glenn and sent as the new desired distance and angle

$$\begin{aligned} d_{CAN} &= d_{CAN}^{-} + d_p \\ \alpha_{CAN} &= \alpha_{CAN}^{-} + \alpha_p \end{aligned} \tag{3.1}$$

where

$$\begin{aligned} d_p &= \sqrt{(x_k - x_p)^2 + (y_k - y_p)^2} \\ \alpha_p &= \theta_k - \text{atan2}((y_p - y_k),(x_p - x_k)) \end{aligned} \tag{3.2}$$

Here $\theta_k$ is the heading of Glenn with respect to the X-axis as seen in figure 4.1 and atan2 is a function that returns the angle from the X-axis to a vector in this case the vector between ($x_k$,$y_k$) and ($x_p$,$y_p$) [13].

The reason to look at a point ahead of Glenn is that this gives smoother corners and avoids an overshoot when turning. This way on the other hand makes Glenn cut corners. To avoid colliding when turning a corner the map is adjusted by moving all the convex corners further in.

## 3.1 Collision Avoidance

Collision avoidance is very important for all kinds of autonomous robots to avoid those obstacles that are not known before hand such as people or things placed temporarily in the robots way. So far only a simple solution is implemented which makes Glenn stop

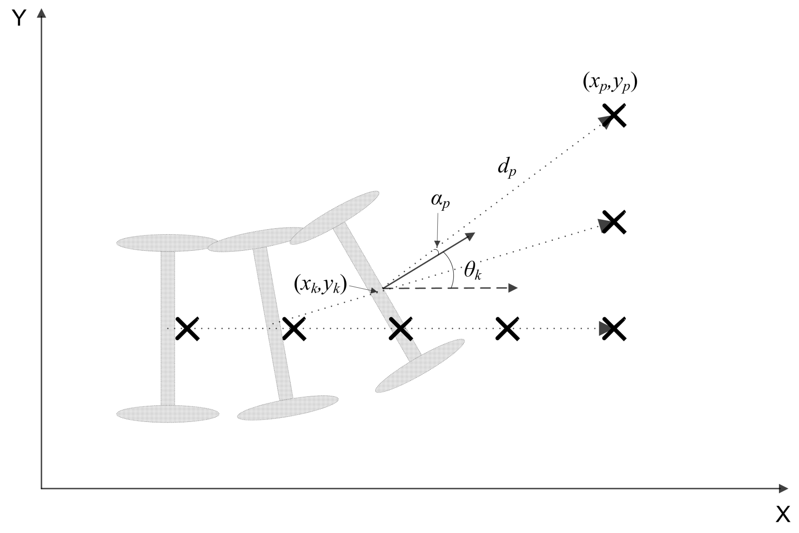

**Figure 3.1:** The path following algorithm where the crosses are the way points and the dotted arrows indicates which one Glenn drives towards at the given positions

if the ultrasonic sensors senses something getting to close. This solution works to keep Glenn from colliding but it is possible for Glenn to get stuck in front of an obstacle unable to continue. Therefore future improvements is needed in this area to enable Glenn to drive around obstacles. There are many ways to achieve this. For example in [4] six proximity sensors (IR) is used to adjust the speed and angle to avoid obstacles and in [3] a laser is used to spot the closest point on an obstacle and from this recalculate the next way point.

# Chapter 4
# Localization

A prerequisite for the autonomous guiding to work is that Glenn can locate in a global coordinate system (figure 4.1). To achieve this odometric calculations is used on the information about wheel rotation received from Glenn. Odometry is subjected to a localization error that grows with the distance traveled. To compensate for these errors two different ways are implemented. First particle filter localization (PFL) with measurements from the kinect camera. Then a wall following algorithm that uses the ultrasonic sensors.

## 4.1 Odometry

Odometry also known as dead reckoning is the use of odometers, sensors that measures distance traveled (in this case encoders), to estimate the pose, $\mathbf{x}_k = [x_k y_k \theta_k]^T$, of a mobile robot at time step $k$. Where $x_k$ and $y_k$ is the position of the robot in the global coordinate system in figure 4.1 and $\theta_k$ is the angle relative the global X axis. For small values of $\delta\theta_k$ the pose can be estimated as

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_{k-1} + \begin{bmatrix} \delta d_k \cos(\hat{\theta}_{k-1} + \frac{\delta\theta_k}{2}) \\ \delta d_k \sin(\hat{\theta}_{k-1} + \frac{\delta\theta_k}{2}) \\ \delta\theta_k \end{bmatrix} \tag{4.1}$$

$$\delta d_k = \frac{\delta r_k + \delta l_k}{2} \quad \delta\theta_k = \frac{\delta r_k - \delta l_k}{W}$$

where $W$ is the wheelbase, $\hat{\mathbf{x}}_{k-1} = [\hat{x}_{k-1}\hat{y}_{k-1}\hat{\theta}_{k-1}]^T$ is the previous pose estimation and $\delta r_k$ and $\delta l_k$ is the distance traveled by the right and left wheel respectively since the last time step [7]. The guiding program does not have access to the individual wheel distances so $\delta r_k$ and $\delta l_k$ instead becomes

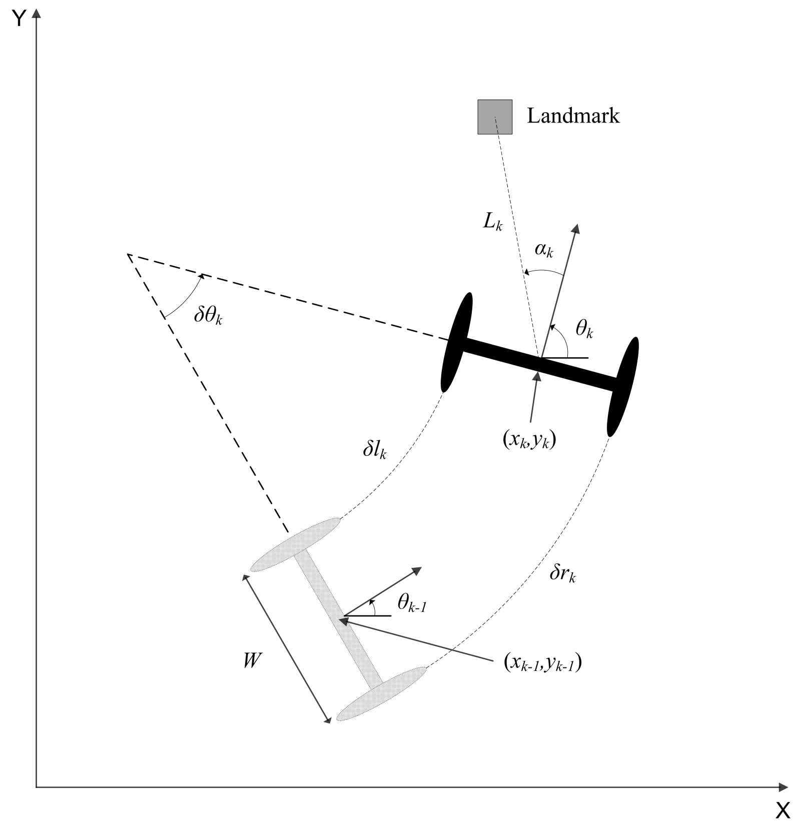

**Figure 4.1:** The coordinate system of Glenn

$$\begin{aligned} \delta d_k &= D_w \pi (d_{CAN} - d_{CAN}^{-}) \\ \delta \theta_k &= 2 D_w \pi \frac{(\alpha_{CAN} - \alpha_{CAN}^{-})}{W} \end{aligned} \tag{4.2}$$

where $D_w$ is the wheel diameter and $d_{CAN}$ and $\alpha_{CAN}$ is from equation 3.2.

### 4.1.1 Odometric Errors

Because odometry uses the previous pose to estimate the current one errors will accumulate over time. There are two types of odometric errors, *systematic errors* and *non-systematic errors* [14]. Examples of these error sources are:

- Systematic errors

  1. unequal wheel diameter, for example caused by unequal air pressure in the wheels.
  2. uncertain wheelbase, because the wheels contact the floor over an area not in a single point. Therefore the wheelbase is difficult to measure correct.

3. misalignment of wheels

- Non-systematic errors

  1. uneven floors
  2. objects in the way
  3. wheel slippage

Non-systematic errors are different each run and can not be predicted beforehand but do not contribute as much as systematic errors when traveling in a smooth indoor environment. Systematic errors are vehicle specific and easier to predict beforehand and can thus be compensated for [14]. One widely used procedure to correct systematic odometric errors are the UMBmark procedure [15]. The UMBmark procedure corrects the errors in the wheelbase and the wheel diameter for each wheel respectively. In this case this is not useful because the guiding program does not have access to the individual wheels. Instead an error correction is added to the pose prediction which becomes

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_{k-1} + \begin{bmatrix} \delta d_k \cos(\hat{\theta}_{k-1} + \frac{\delta\theta_k}{2}) + e_{drift}\delta d_k \sin(\hat{\theta}_k) + e_{trans}\delta d_k \cos(\hat{\theta}_k) \\ \delta d_k \sin(\hat{\theta}_{k-1} + \frac{\delta\theta_k}{2}) + e_{drift}\delta d_k \cos(\hat{\theta}_k) + e_{trans}\delta d_k \sin(\hat{\theta}_k) \\ \delta\theta_k + e_{rot}\delta\theta_k \end{bmatrix} \tag{4.3}$$

where $e_{drift}$, $e_{trans}$ and $e_{rot}$ are derived experimentally (see chapter 5).

## 4.2 Kinect

The Microsoft Kinect sensor is a low cost RGB camera and depth sensor that was released in 2010 [16]. The Kinect camera is used to recognize the landmarks and measure the distance $L_k$ and azimuth angle $\alpha_k$ as shown in figure 4.1. The landmarks are detected by first creating a threshold image of the kinect RGB image where one color (in this case red) is white and everything else is black. Then contours over a certain size is analyzed as potential landmarks. The different landmarks are identified by letters. This procedure is shown in figure 4.2.

The kinect sensor measures distance to the plane parallel to the sensor where the object is located as shown in figure 4.3 [16]. This means if a landmark is not in the middle of the image the distance returned from the kinect is to small. Therefore the azimuth angle $\alpha_k$ needs to be taken into account when measuring the distance $L_k$. $\alpha_k$ and $L_k$ becomes

$$\begin{aligned} \alpha_k &= \frac{resX/2 - X_L}{FOV/resX} \\ L_k &= \frac{Z_{kinect}}{\cos(\alpha_k)} \end{aligned} \tag{4.4}$$

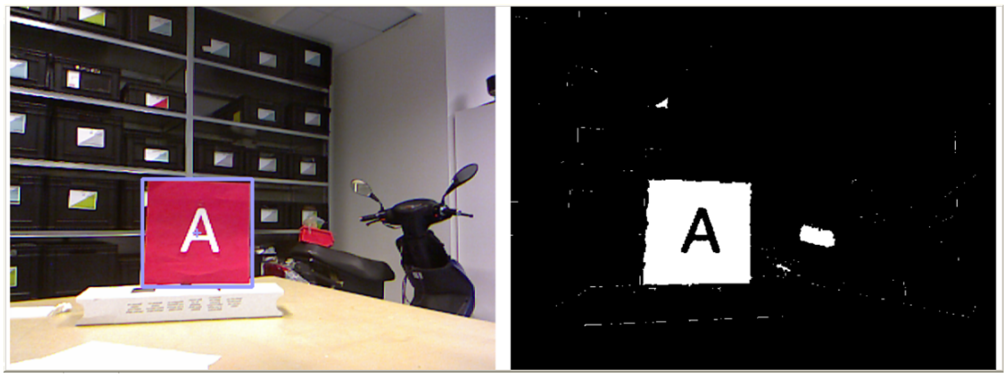

**Figure 4.2:** The RGB image (left) from the kinect and the threshold image (right). The blue rectangle represent the detected landmark. In the threshold we can see that smaller contours are ignored

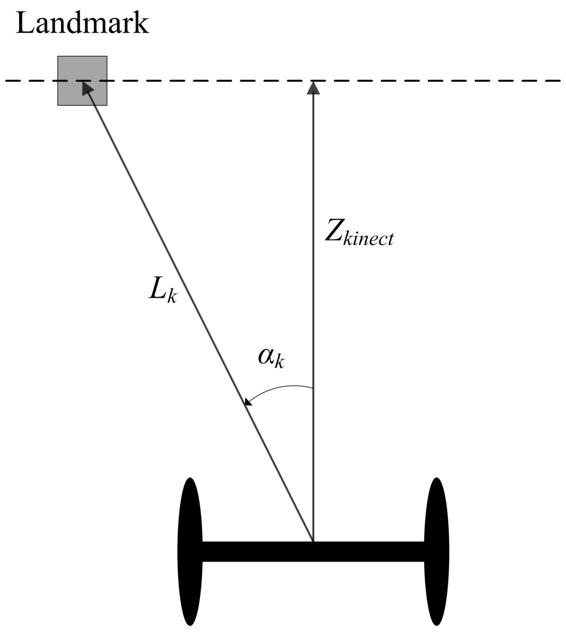

**Figure 4.3:** The kinect camera measures the distance to a plane parallel to the sensor (dashed line)

where $resX$ is the horizontal resolution of the kinect RGB camera, $X_L$ is the x position of the landmarks center pixel, $FOV$ is the Field Of View for the kinect RGB camera and $Z_{kinect}$ is the returned depth measurement from the kinect sensor. According to [17] $resX = 640$ pixels and $FOV = 62.7$ deg. After some experiments the $FOV$ was adjusted to 63 deg which gave better results. Figure 4.4 displays the kinect measurements compared to the real values. This shows a precision for the angle within 0.6 deg and for the distance within 1 cm up to 2 m. After 2 m the distance error grows but it is still within 3-4 cm.

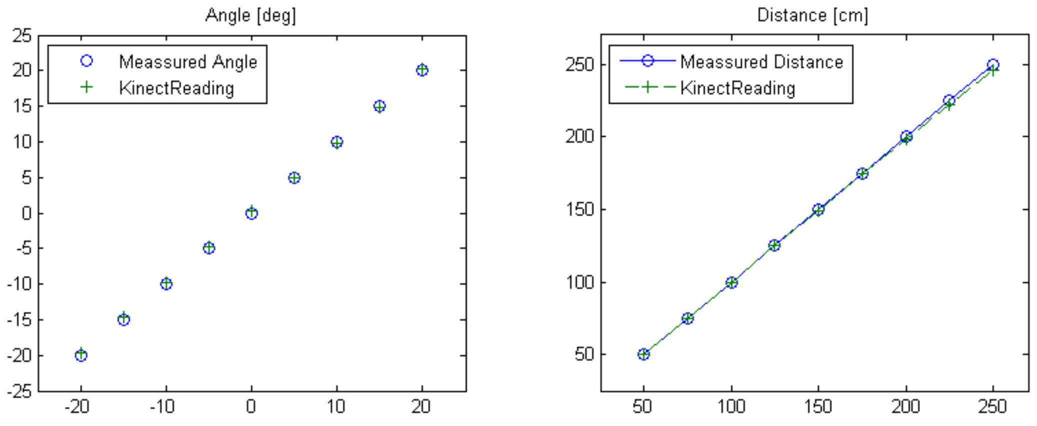

**Figure 4.4:** Test of the kinect cameras depth and angle measurements

## 4.3 Particle Filter

The kinect angle and distance measurements from one landmark is not enough to determine the pose of Glenn. Therefore particle filter localization (PFL, also known as Monte Carlo localization [7]) is used. The PFL estimates the pose by keeping multiple "guesses" of the pose called particles and each particle has a weight that signifies the probability of that particle being the current pose of Glenn. The PFL works in two phases a *prediction* phase and an *update* phase [8]. During the prediction phase each particle is moved according to the odometry model (equation 4.1) with an added random noise to represent the odometric errors. This results in a distribution of particles over an area around the current pose which grows with the distance. When a landmark is detected the particle filter performs the update phase where the kinect readings is used to calculate the weight of each particle as

$$W_i = \frac{1}{\sigma_L} e^{\frac{-\delta L_i^2}{2\sigma_L^2}} \frac{1}{\sigma_\alpha} e^{\frac{-\delta \alpha_i^2}{2\sigma_\alpha^2}} \tag{4.5}$$

which is an adaption from the approach in [8]. $\delta L_i$ and $\delta \alpha_i$ is the difference of the actual kinect readings and the expected kinect readings for particle $i$ according to

$$\begin{aligned}\delta L_i &= L_k - \sqrt{(x_M - x_i)^2 + (y_M - y_i)^2} \\ \delta\alpha_i &= \alpha_k - \tan^{-1}(\frac{y_M - y_i}{x_M - x_i}) - \theta_i\end{aligned} \tag{4.6}$$

where $\mathbf{x}_i = (x_i,y_i,\theta_i)$ is the pose of particle $i$ and $(x_M,y_M)$ is the position of the spotted landmark [7]. $\sigma_L$ and $\sigma_\alpha$ determines which kinect measurement effects the weight most and are derived experimentally.

After the weights are calculated they are normalized. This results in a *pdf* (probability density function) of the random pose, $\hat{\mathbf{X}}$, of Glenn. After that a process called re-sampling is performed which probabilistically multiplies the particles according to their weights. This means particles with small weights have a small chance of multiplying while particles with high weights have a big chance. In practice this removes the particles with small weights and multiplying the ones with higher weights. Then the particle filter is run again from the prediction phase and repeated until a good pose estimation can be obtained. To get the final estimate of the pose three ways are suggested, [8]:

- Best particle
  $max(W)$
- Weighted mean
  $\hat{\mathbf{x}} = \sum W_i \mathbf{x}_i$
- Robust mean

The best particle solution is simply to take the particle with the highest weight. This solution is subjected to discretization errors when the actual pose is not represented by a particle. The weighted mean solution adds all particles multiplied with its respective weight. This fails when the pdf is multi-modal, have multiple local maxima. The best solution is robust mean which is a combination of the above, the weighted mean is taken in an small area around the best particle [8].

To be able to run the PFL, Glenn needs to stand still. This is because the image processing involved in getting the kinect measurements is not fast enough. If Glenn is moving the measurements is from an old position when the particle filter calculates the weights resulting in a faulty pose estimate. Instead Glenn stops each time a landmark is detected. Then two iterations of the filter is performed with re-sampling after which the robust mean is taken as the estimated pose. Glenn standing still results in a problem with the prediction phase. After the first iteration there are no new movements of Glenn to use for predicting new particles so instead each particle are disturbed randomly around its current pose. Figure 4.5 shows the particles after the four steps used. In A the particles from the prediction phase is shown. B shows the particles left after the first iteration. C shows the particles after the third iteration where the particles from B where disturbed. Then D shows the final step where the robust mean has been calculated. The black particle is the actual pose and the red in D is the estimated pose which is difficult to see because it is situated underneath the actual pose.

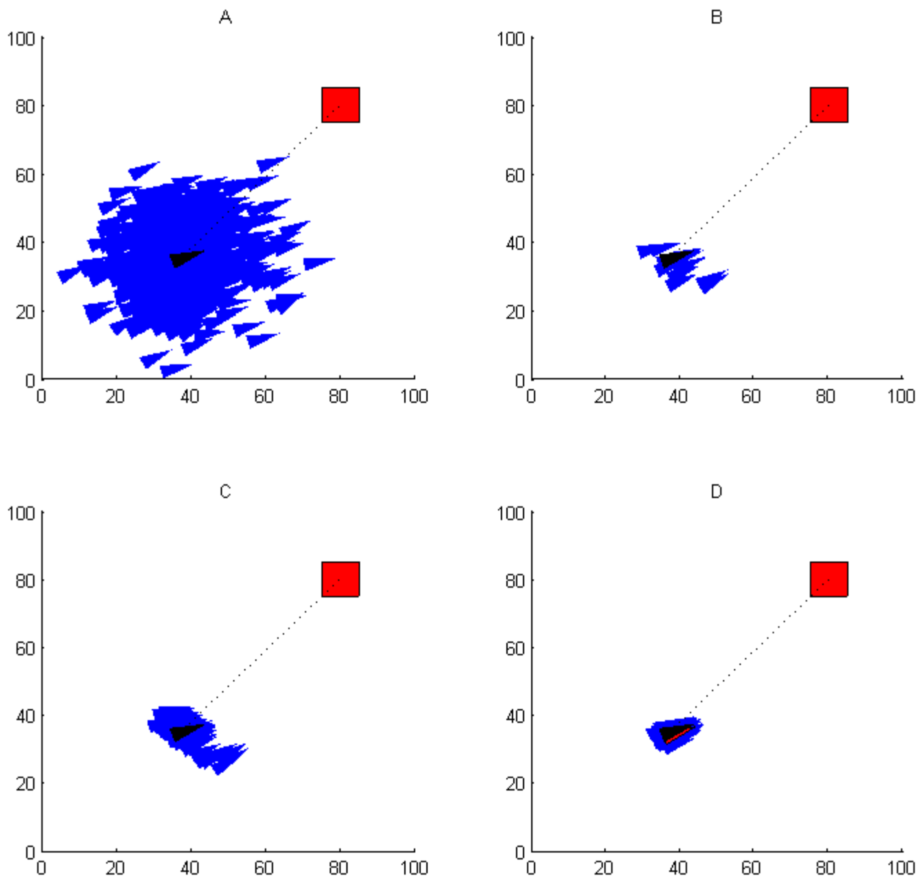

**Figure 4.5:** A: The particles after the prediction phase. B: The particles after the first iteration. C: The particles after the second iteration. D: The particles and estimated pose (red) after the robust mean has been calculated. The black particle represent the actual pose and the red rectangle is the landmark used

## 4.4 Wall Following

As an alternative to the PFL a wall following algorithm is developed. First the map is divided into zones depending on which side of Glenn the best wall to follow is located. A smooth and straight wall is preferred. Then several measurements from the ultrasonic sensor on that side is recorded together with the estimated pose of Glenn on the axis parallel to the wall. The difference $S$ between the recorded distance to the wall and the distance it should be depending on the estimated pose is calculated. Then the least square method is used to approximate a line corresponding to the path Glenn actually travels related to its estimated path. The angle $\varphi$ between this line and its x-axis then becomes the angle error of Glenn. This angle together with the last sensor measurement is then used to update the pose of Glenn, figure 4.6 shows a graphic representation of the algorithm.

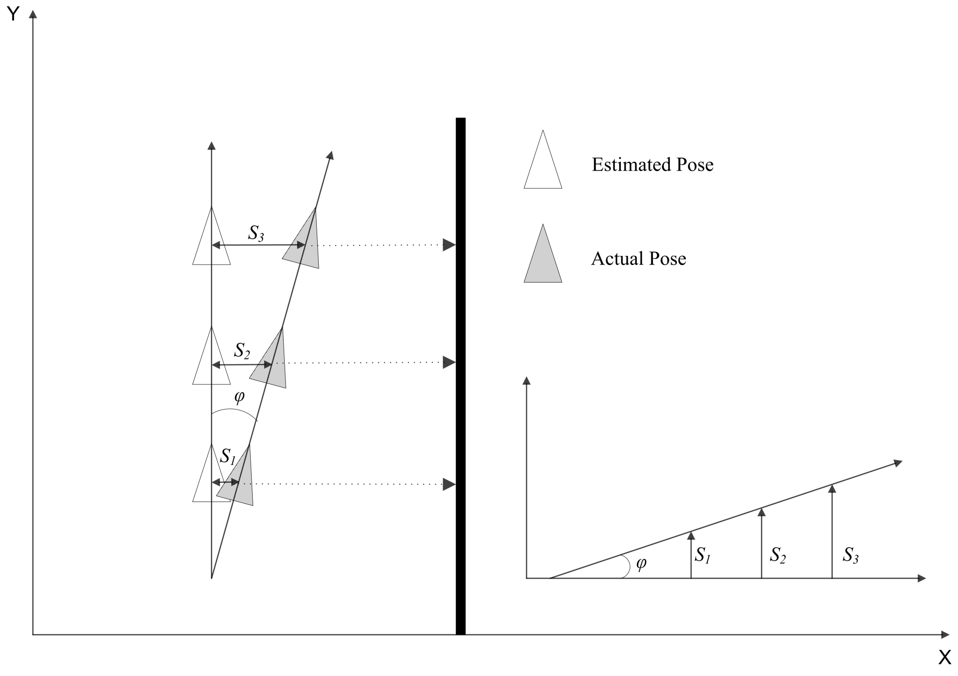

**Figure 4.6:** The wall following algorithm

This approach has some limitations. First, a smooth and straight wall is needed, if the wall is to uneven it does not work. This is the case in some parts of the office, here the pose is just updated with the distance from the wall at regular intervals. This does not correct the heading but keeps Glenn away from the wall. Second, the ultrasonic sensor used are not located at the turning axis of Glenn. This results in to large sensor readings when Glenn is not parallel to the wall. This is solved by limiting the angle $\varphi$ so that large values has a limited effect. Third, only the position to the side is corrected, so when Glenn approached a corner it could turn to soon or to late. This is solved by stopping before a turn and using the kinect camera to update the distance to the corner.

# Chapter 5

# Result

To see how well Glenn performs, first the localization is tested. Then Glenn is set to perform different guiding tests to see how all the parts work together.

## 5.1 Localization Results

Two steps where taken to improve the localization. First the systematic errors where corrected using equation 4.3 then the PFL and wall following was implemented. The localization tests where performed by leading Glenn by hand along a predefined path and logging the pose, +, at certain positions, ∘. 20 runs were performed for each setting. In figure 5.1 the localization of Glenn when using uncorrected odomerty is shown. Here we clearly see the problems with odometry as the uncertainty of the pose grows by the distance traveled. There is also a big systematic error as seen from the mean error that deviates more and more from the path.

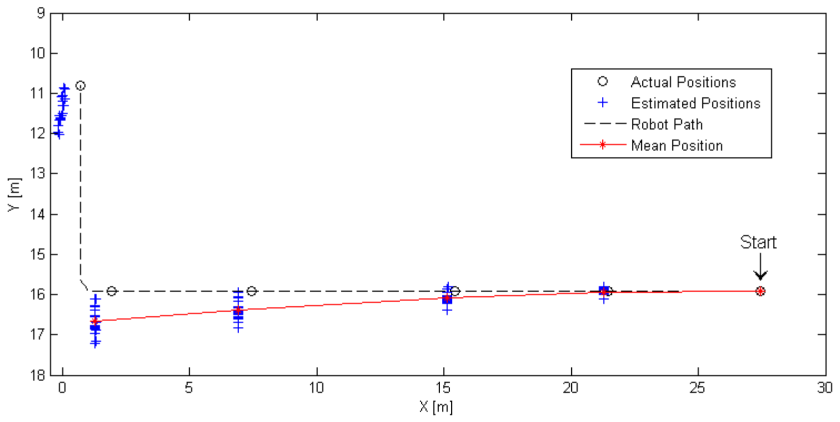

**Figure 5.1:** Localization test using odometry

To correct the systematic errors $e_{drift}$ and $e_{trans}$ where obtained from the meassurements from figure 5.1 according to

$$\begin{aligned} e_{drift} &= \frac{1}{n}\sum_{i=1}^{n}\frac{D_i}{G_i} \\ e_{trans} &= \frac{1}{n}\sum_{i=1}^{n}\frac{T_i}{G_i} \end{aligned} \tag{5.1}$$

Where $n = 4$ is the first four test positions not including the start position, $D_i$ and $T_i$ is the mean deviation from the actual position $i$ in the Y and X direction respectively and $G_i$ is the actual distance traveled. To obtain $e_{rot}$ an additional test where needed where Glenn was standing in a fixed position and turned around its axis and the angle was logged every 90 degrees. The result of the angle measurements of the uncorrected odometry is seen in figure 5.2, where the arrow indicates the measurements after a 360 degree turn.

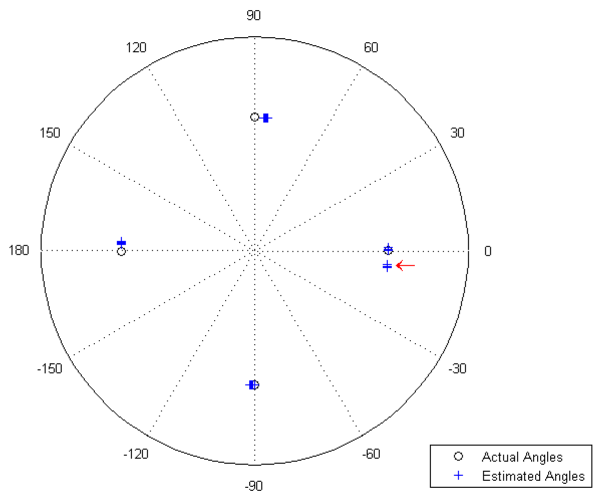

**Figure 5.2:** Angle test using odometry

Here it is also obvious there is a systematic error because the angle measurements is not centered around the actual angles but concentrated a certain angle away. Also this angle grows at every step which suggests a systematic error. $e_{rot}$ is obtained as

$$e_{rot} = \frac{1}{m}\sum_{i=1}^{m}\frac{R_i}{A_i} \tag{5.2}$$

Where $m = 4$ represents the test angles, $R_i$ is the mean deviation from the actual angle at angle $i$ and $A_i$ is the actual angle. In figure 5.3 the localization test for the corrected odometry is shown. The localization is clearly improved. The errors still grows without bounds with the distance but the systematic errors are reduced. This does not remove the systematic errors and over a large enough distance they would increase more and more. This will be enough in this case because the pose of Glenn will be updated at regular intervals.

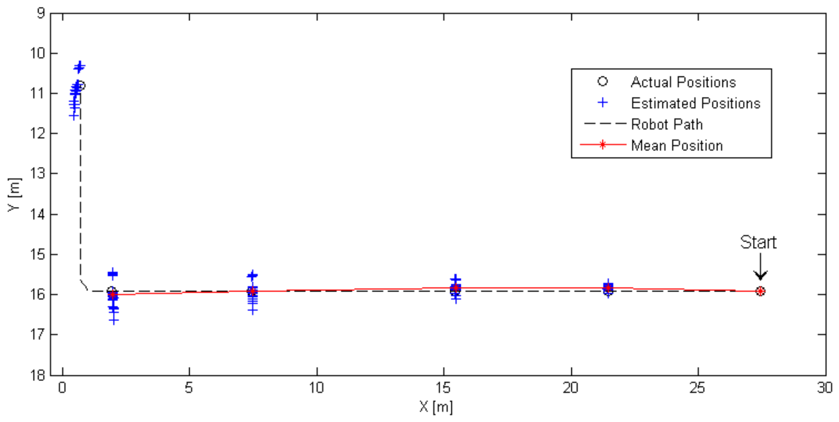

**Figure 5.3:** Localization test using corrected odometry

Figure 5.4 and 5.5 shows the result of the same test performed with PFL and wall following respectively. Both solutions shows a significant improvement. To decide the tuning parameters, $\sigma_L$ and $\sigma_\alpha$, for the PFL first $\delta\alpha_i$ and $\delta L_i$ were set to similar size by taking $\frac{\delta L_i}{100}$ before inserting in to equation 4.5. Then different settings of $\sigma_L$ and $\sigma_\alpha$ where tested. The best localization was found for $\sigma_L = 0.05$ and $\sigma_\alpha = 0.07$.

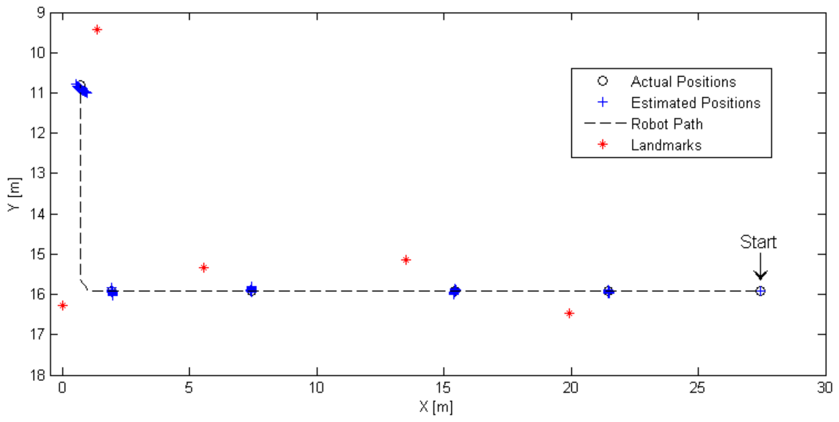

**Figure 5.4:** Localization test using PFL

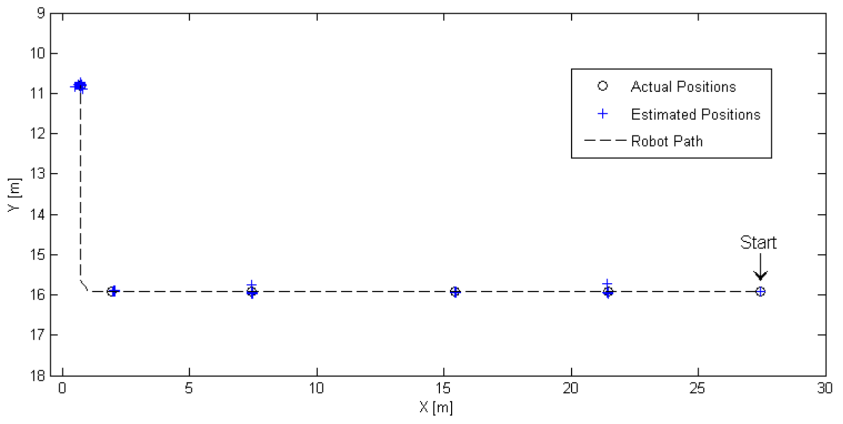

**Figure 5.5:** Localization test using wall following

To compare the different results the *RMSE* (Root Mean Square Error) for each position is plotted in figure 5.6. This clearly shows that PFL and wall following are the most accurate methods for positioning. Wall following even shows better results after the 90 degree turn. It is also evident that great improvements can be made by calibrating the odometric calculations.

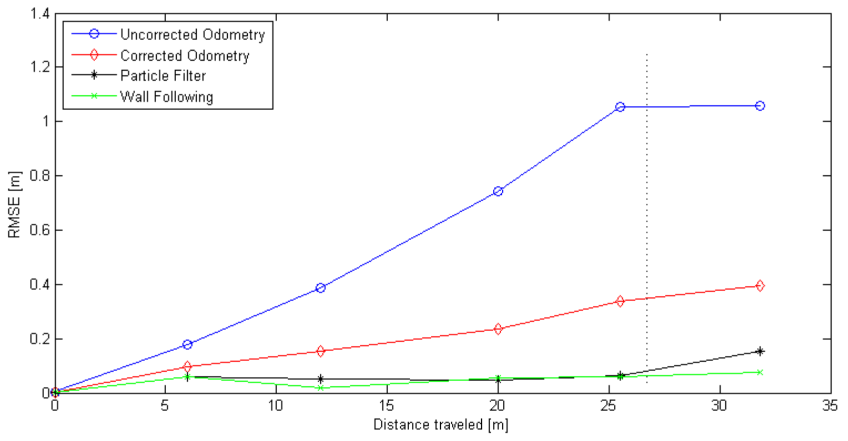

**Figure 5.6:** RMSE for the positioning. The dotted line marks the 90 degree turn

Localization for an autonomous robot does not only include positioning but also heading. In figure 5.7 the RMSE for the heading at each position is displayed. Here the PFL is not that much better than only using odometry, it even becomes worse after the 90 degree turn performed between the last two positions. The angle errors are small, $\theta < 5$ deg, but angle errors effects on the positioning grows with the distance traveled. So small errors can have big effects. The wall following on the other hand does improve the heading. Here only the last two measurements are of interest because the first three positions is in a zone with a uneven wall and therefore no heading update are performed here.

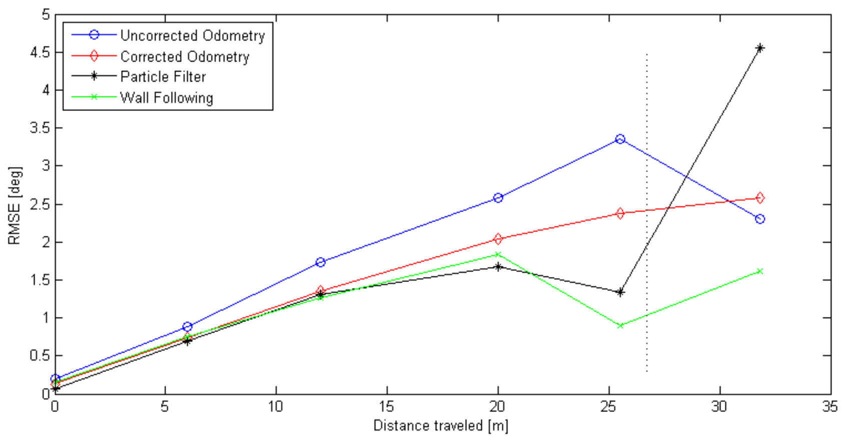

**Figure 5.7:** RMSE for the heading. The dotted line marks the 90 degree turn

## 5.2 Guiding Results

When testing the full guiding functionality of Glenn it becomes obvious that the localization is not robust enough when using PFL. The path planning with the A* algorithm works perfectly. Glenn always finds a path when there is a possible one. The path following control also works well, if we consider the pose Glenn thinks it has. The problem arises when the estimated pose do not coincide with the real pose, with other words the localization. The problem lies with the heading as the results in figure 5.7 indicates. If the heading is wrong after an update phase then Glenn will go in the wrong direction and because it is some meters between the landmarks Glenn will get to close to a wall and stop before being able to update the position again. Another problem is that Glenn sometimes misses to spot a landmark.

When using wall following the movement of Glenn becomes more jerky because of the constant updating of the pose but this approach works much better.

In table 5.1 the results of 10 guiding tests together with how many was successful is displayed. Here it is clear that the PFL is not robust enough and that the wall following works.

| **Mode Used** | **Performed Tests** | **Successful Tests** | **Success Rate [%]** |
|---|---|---|---|
| **Particle Filter** | 10 | 5 | 50 |
| **Wall Following** | 10 | 10 | 100 |

**Table 5.1:** The results of 10 guiding tests for each mode

# Chapter 6
# Conclusion

The result of this project is straight forward. Using a particle filter together with a kinect camera greatly improves the positioning of an autonomous robot which agrees with the results in [7]. The heading on the other hand is not improved as much. The heading is not explored in [7] so there are no research to compare with. It can not be excluded that it is possible of improving the PFL to also give a more accurate heading. On the other hand I do not believe that would solve the problem for the guiding. The PFL developed in this project is not robust enough for an indoor guiding functionality as can be seen in table 5.1, it worked only 50% of the times. The wall following method shows a better localization according to figure 5.6 and 5.7, especially when considering the heading. The guiding test presented in table 5.1 shows that it works 100% of the times. Better localization is not the only reason the guiding works so well. The main reason is it gets readings and update the position at a much higher frequency. If after one update the pose is wrong a new update occurs before Glenn has gotten to much off course to not be able to continue. This also suggests that the PFL also would work better if Glenn could update its pose at a higher frequency through more landmarks, and I believe it would. Still the wall following would be preferred, because Glenn would need to stop at every landmark and would travel very slow which would not be optimal for guiding visitors.

## 6.1 Future Work

To make Glenn work as a guiding robot my suggestions is to skip localization with the kinect camera and instead focus on developing better wall following. Adding more ultrasonic sensors or using a laser for proximity sensing would be an advantage and would probably result in smoother movements. Also the collision avoidance should be improved to make Glenn able to go around obstacles and continue following the path. With some more work I believe Glenn could become an awesome office guide.

# Bibliography

[1] Wolfram Burgard, Armin B. Cremers, Dieter Fox, Dirk Hähnel, Gerhard Lakemeyery, Dirk Schulz, Walter Steiner, and Sebastian Thrunz, "The Interactive Museum Tour-Guide Robot," *AAAI-98 Proceedings*, 1998.

[2] D. S. Mattias Wahde and K. Wolff, "Reliable Long-Term Navigation in Indoor Environments," *Recent Advances in Mobile Robotics*, 2011.

[3] J. L. Martinez, A. Pozo-Ruz, S . Pedraza and R. Fernhdez, "Object Following and Obstacle Avoidance Using a Laser Scanner in the Outdoor Mobile Robot Auriga-$\alpha$," *Proceedings of the 1998 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems*, 1998.

[4] X. Hu, D. Fuentes Alarcón and T. Gustavi, "Sensor-Based Navigation Coordination for Mobile Robots," *42nd IEEE Conference on Decision and Control*, 2006.

[5] Margrit Betke and Leonid Gurvits, "Mobile Robot Localization Using Landmarks," *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, VOL. 13, NO. 2*, 1997.

[6] Kimihiro Okuyama, Tohru Kawasaki, and Valeri Kroumov, "Localization and Position Correction for Mobile Robot Using Artificial Visual Landmarks," *Proceedings of the 2011 International Conference on Advanced Mechatronic Systems, Zhengzhou, China*, 2011.

[7] N. Ganganath and H. Leung, "MOBILE ROBOT LOCALIZATION USING ODOMETRY AND KINECT SENSOR," 2012.

[8] Ioannis M. Rekleitis, "A Particle Filter Tutorial for Mobile Robot Localization," 2003.

[9] P. Hiemstra, and A. Nederveen, "Monte Carlo Localization," 2007.

[10] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust Monte Carlo Localization for Mobile Robots," 2001.

[11] K. Pazul, "Controller Area Network (CAN) Basics," Microchip Technology Inc., Tech. Rep., 1999.

[12] S. Russel, P. Norvig, *Artificial Intelligence - A Modern Approach*, 2nd ed., 2003.

[13] Microsoft, "Math.Atan2 Method," 2012. [Online]. Available: http://msdn.microsoft.com/en-us/library/system.math.atan2.aspx

[14] Johann Borenstein and Liqiang Feng, "Maesurement and Correction of Systematic Odometry Errors in Mobile Robots," *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, VOL. 12, NO. 6*, 1996.

[15] ——, "UMBmark: A Benchmark Test for Measuring Odometry Errors in Mobile Robots ," *SPIE Conference on Mobile Robots, Philadelphia, October 22-26*, 1995.

[16] K. Khoshelham, "ACCURACY ANALYSIS OF KINECT DEPTH DATA," 2011.

[17] ROS.org, 2012. [Online]. Available: http://www.ros.org/wiki/kinect_calibration/technical#Focal_lengths

# Appendix A
# The Parts of Glenn

- 2 Motor drives, from an existing Aros product.
- 2 Permanent magnet synchronous motors, designed and produced by Aros.
- 1 Control board where the balance control is performed, designed and produced by Aros.
- 1 Gyro sensor, 1 axis, ADXRS610.
- 1 Accelerometer, 2 axis, ADXL203EB.
- 2 Encoders one for each motor, Tamagawa 1024 pulses/rev.
- 6 Devantech SRF01 ultrasonic sensors that measures distance used for collision avoidance and wall following.
- 1 Sensor board controlling the ultrasonic sensors over a single pin serial interface, designed and produced by Aros.
- 1 Zotac PC with a 1.6 GHz Intel Atom processor and Nvidia graphics equipped with a touchscreen, speakers and a keyboard.
- 2 Li-Po battery packs, 22.2 V and 6000 mAh.
- 1 Power board for distributing the power.
- 1 PC power board for transforming the voltage for the PC.
- 1 PC with attached keyboard, speakers and touchscreen.
- 1 Xbox 360 Kinect camera to make Glenn able to see.
- 1 USB to CAN converter designed and produced by Aros.
- 1 Candy bowl.

# Appendix B

# A* Algorithm

```
input  : Start and Goal Point
output: List of points along the closest path
openSet = {startPoint }; // The set of points to be tested
closedSet = ∅; // The set of points already tested
g(startPoint) = 0;
f(startPoint) = g(startPoint) + ShortestDistance(startPoint,goalPoint);
while openSet ≠ ∅ do
    currentPoint = Point with lowest f;
    if currentPoint = goalPoint then
        return ReconstructPath(cameFrom,goalPoint)
    end
    Remove currentPoint from openSet;
    Add currentPoint to closedSet;
    foreach neighbourPoint to currentPoint do
        if (neighbourPoint ∈ closedSet) ∨ (neighbourPoint ∉ map) then
            continue;
        end
        tentativeGScore = g(currentPoint) +
        ShortestDistance(currentPoint,neighbourPoint);
        if (neighbourPoint ∉ openSet) ∨ (tentativeGScore <
        g(neighbourPoint)) then
            Add neighbourPoint to openSet;
            cameFrom(neighbourPoint) = currentPoint;
            g(neighbourPoint) = tentativeGScore;
            f(neighbourPoint) = g(neighbourPoint) +
            ShortestDistance(neighbourPoint,goalPoint);
        end
    end
end
function ReconstructPath(cameFrom,point)
    if cameFrom(point) ≠ startPoint then
        return ReconstructPath(cameFrom,cameFrom(point))
    else
        return point
    end
```

**Algorithm 1:** A* Algorithm