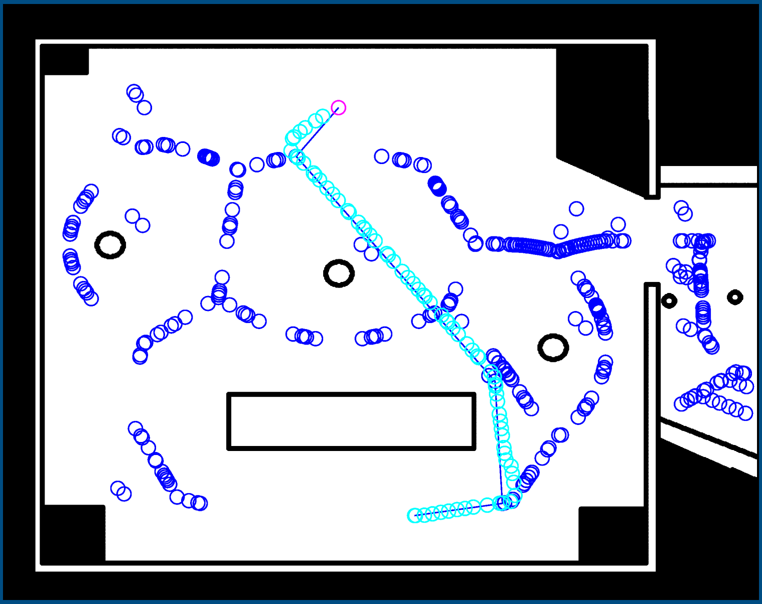
# Autonomous Wheelchair Navigation

Master Thesis

Jeppe Møller Holm
Søren Lynge Pedersen
Group 1031b



June 2008
Section for Automation and Control
Aalborg University, Denmark

AALBORG UNIVERSITY

**Section for Automation and Control**
Fredrik Bajers Vej 7C
Telephone 96 35 87 00
Fax 98 15 17 39
http://www.control.aau.dk

**Title:**
Autonomous Wheelchair Navigation

**Project period:**
Fall semester 2007
- Spring semester 2008

**Project group:**
1031b

**Members:**
Søren Lynge Pedersen
Jeppe Møller Holm

**Supervisor:**
Anders la Cour-Harbo

**Copies:** 4

**Pages in report:** 120

**Printed June 2, 2008**

**Abstract:**

This Project describes how to make an autonomous wheelchair for people with Amyotrophic Lateral Sclerosis (ALS). Because people with ALS cannot use their muscles to propel or even control a normal wheelchair, it is necessary that the wheelchair can be controlled by the only part of the body that they still have full control over; their eyes. It is investigated how to sense the house, how to localise the wheelchair in the house, how to plan a path through the house and how to drive a wheelchair. Mounted on the wheelchair to sense the house is a laser range scanner. The output from this along with the odometry is used in a particle filter for localisation, incorporating an Augmented Adaptive Monte Carlo Localisation. To plan the path through the house, the Configuration Space of the house is computed and a Generalised Voronoi Diagram calculates waypoints with the maximum distance to the walls. A Dijkstras Algorithm finds the shortest path from start to goal. To avoid unmapped obstacles, repulsive velocity vector fields are implemented. The controller to control the wheelchair is an optimal model based LQR. This project has been a proof of concept, where it has been shown that it is possible to make an autonomous wheelchair, capable of transporting a user with Amyotrophic Lateral Sclerosis, or any other disabled person, with no user input except where to go. The wheelchair does not need to be told where it is, or how to get to the goal. It does not even need any artificial landmarks for navigation. Using only onboard sensors and a map of the house, which was available anyway, it performs the required tasks.

# Preface

This project is the master thesis made by group 1031b during the 9th and 10th semester of the Master program in Intelligent Autonomous Systems at the section for Automation and Control at Aalborg University (AAU). The project was carried out in the period from the 1st of September 2007 the to 4th of June 2008

The reader of this report is presumed to have a similar technical background as the project group, including knowledge of the courses given during the 9th semester, which are on the subject of Intelligent Autonomous Systems. This particular project is based on the project proposal "Autonomous Wheelchair Controlled By an Eye Tracker" and discusses the design and implementation of an autonomous wheelchair for disabled people with Amyotrophic Lateral Sclerosis. The project explores the subjects: Perception, Localisation, Cognition and Motion control. The Matlab version used is R2007B with simulink version 6.5.

Citations are made in square brackets and contain the author of the work and the year of publication. Cited works and papers except books, are to be found on the enclosed CD. The enclosed CD also contains the model, m-files and the controller implemented in Matlab/Simulink. Furthermore it contains a PDF and postscript version of the report.

Søren Lynge Pedersen

Jeppe Møller Holm

# Nomenclature

## Symbols

| | | | | |
|---|---|---|---|---|
| $X$ | : | The state vector | : | [$\cdot$] |
| | | $X = [x \;\; y \;\; \theta]^T$ | | |
| $x$ | : | x position of AWC | : | [cm] |
| $y$ | : | y position of AWC | : | [cm] |
| $\theta$ | : | Angle of AWC | : | [degrees] |
| $u_t$ | : | The odometry data | : | [$\cdot$] |
| | | $u_t = [\Delta x \; \Delta y \; \Delta\theta]^T$ | | |
| $z_t$ | : | The combined LRS measurement | : | [$\cdot$] |
| | | $z_t = \begin{bmatrix} \theta_1 & z_1 \\ \vdots & \vdots \\ \theta_k & z_k \end{bmatrix}$ | | |
| $k$ | : | Number of distance measurements in LRS scan | : | [$\cdot$] |
| $z_t^k$ | : | One LRS measurement | : | [cm] |
| $z_{max}$ | : | LRS maximum detection range | : | [cm] |
| $m$ | : | The map | : | [$\cdot$] |

## Abbreviations / Concepts

| | | |
|---|---|---|
| AWC | : | Autonomous Wheelchair |
| FOV | : | Field Of View |
| LRF | : | Laser Range Finder |
| LRS | : | Laser Range Scanner |
| Pose | : | The AWCs position and angle, see state $X$ |
| MaL | : | Markov Localisation |
| MCL | : | Monte Carlo Localisation |
| AMCL | : | Adaptive MCL |
| AAMCL | : | Augmented AMCL |

## Constants

| | | | | | |
|---|---|---|---|---|---|
| $g$ | : | Gravity | : | 9.82 | [m/s] |

# Contents

**I Introduction 9**

**1 Introduction 11**
1.1 Project Formulation . . . . . 11
1.2 Project Limitations . . . . . 11
1.3 Project Requirements . . . . . 13
1.4 Reading Directions . . . . . 14

**2 Analysis 15**
2.1 System Functionality . . . . . 15
2.2 Map of the House . . . . . 17
2.3 System Description . . . . . 17
2.4 Software Platform . . . . . 19
2.5 Software Sequence . . . . . 20

**II Autonomous Navigation 21**

**3 Navigation Introduction 23**

**4 Perception 25**
4.1 The Odometry . . . . . 25
4.2 The Laser Range Scanner . . . . . 26

**5 Localisation 35**
5.1 The Monte Carlo Localisation Method . . . . . 37
5.2 Bayesian Theory . . . . . 44
5.3 MCL, AMCL and AAMCL Algorithms . . . . . 46
5.4 Results . . . . . 50

**6 Cognition 53**
6.1 Path Planning . . . . . 53
6.2 Offline Computational Cost . . . . . 57
6.3 Calculating Path . . . . . 58

6.4 Path Guidance . . . . . . . . . . 60
6.5 Cognition Results . . . . . . . . . . 63

**7 Motion Control 67**
7.1 State Estimation . . . . . . . . . . 68
7.2 Controller Design . . . . . . . . . . 74
7.3 Motion Control Results . . . . . . . . . . 77

**III Results and Conclusion 81**

**8 Acceptance Test 83**
8.1 Localization Results . . . . . . . . . . 83
8.2 Cognition Results . . . . . . . . . . 89
8.3 Motion Control Results . . . . . . . . . . 89

**9 Conclusion 91**
9.1 Results Discussion . . . . . . . . . . 91
9.2 Conclusion . . . . . . . . . . 93

**10 Future Work 95**
10.1 Already Done . . . . . . . . . . 95
10.2 Adjustments . . . . . . . . . . 95
10.3 Limited Out Of Project . . . . . . . . . . 96

**IV Appendix 97**

**A Wheelchair Modelling 99**
A.1 Wheelchair Kinematics . . . . . . . . . . 99
A.2 Wheelchair Dynamics . . . . . . . . . . 100
A.3 Combining Kinematics and Dynamics . . . . . . . . . . 104

**B Parameter Estimation for Wheelchair Model 107**
B.1 Motor Parameter Estimation . . . . . . . . . . 107
B.2 Wheelchair Parameter Estimation . . . . . . . . . . 108

**C PCB For the Wheelchair 111**

**D Sampling From A Discrete Distribution 117**

**Bibliography 119**

# Part I

# Introduction

# Chapter 1

# Introduction

## 1.1 Project Formulation

ALS or Amyotrophic Lateral Sclerosis is a neurodegenerative disease which causes muscles weakness and atrophy. As the disease evolves, the patient gradually losses control of the muscles in his body. One of the last muscles to be affected by the disease is the muscles for eye control, which first occur after an extremely long duration of the disease, often after more than 20 years. And since most patients with ALS will die in 3-5 years from the onset of ALS, it is considered that most of the patients have control over their eye movements [The ALS Association, 2007]. This ability is used in this project to control a wheelchair in order for the ALS patient to move around in his own house. This project will use the movement of the eyes, not to control the wheelchair, but to give a destination for the wheelchair. This is done by having a computer screen in front of the wheelchair user and an eye tracker which tracks where on the screen the user is looking. On the screen a model of the current house can be seen, see figure 1.1. By simply looking at the position that the user wants to go, the controller connected to the wheelchair guides it to the destination. This requires that the wheelchair can locate its current position, can calculate how to get from the current position to the desired position by the most optimal path, how to avoid obstacles that is not on the map, such that people and chairs and control the movement in an acceptable way.

This project is part of the Brain Computer Interface project at the Center for Sensory-Motor Interaction [Cabrera & Dremstrup, 2008], in the department of Health Science Technology. It was proposed by Alvaro Fuentes Cabrera, Omar Feix do Nascimento and Kim Dremstrup.

## 1.2 Project Limitations

Based on the project formulation, this project can be divided into a number of smaller objectives.

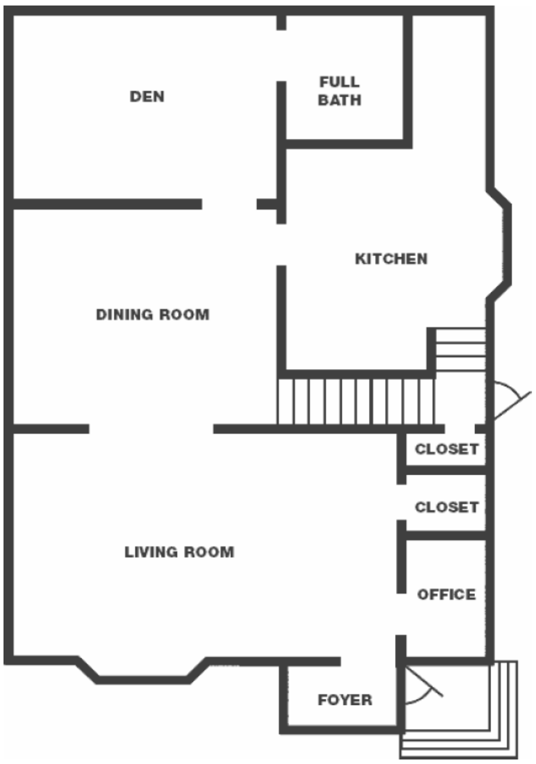


Figure 1.1: Floor plan of an apartment displayed on the screen for the subject to choose the room that the wheelchair should take him to.

### 1.2.1 Parts Covered In This Project

- Implement a map of the house (model). This should be used both for displaying to the user, and internally in the system for path planning.
- Select the number and type of sensors to be mounted on the wheelchair.
- Construct a locating system which can locate the wheelchair in the house, using the sensors and the map.
- Design a path planner.
- Make a dynamic state space model of the wheelchair, for use with the controller and the state estimator.
- Make a controller for a wheelchair which can comfortably move the wheelchair to a desired location within some limits defined on the house in which the wheelchair has to operate.
- Create a test setup, using Lego, a National Instruments 6071E IO card, Mathworks Matlab Simulink and XPC and a scaled model of the apartment.

A picture of the LegoBot can be seen in figure 1.2.

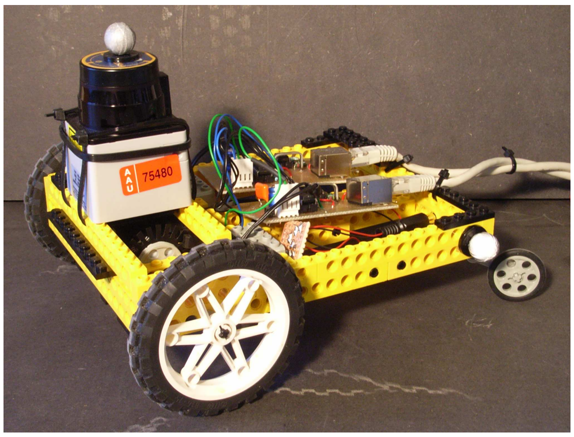


Figure 1.2: Picture of the LegoBot used in this project.

### 1.2.2 Parts Not Covered In This Project

- The wheelchair will not be able to take inputs from the eye tracker. Instead the eye tracker will be simulated by using the computer mouse to give coordinates.
- The wheelchair will not be a real full size wheelchair but a small lego model.
- The software for the wheelchair will be written in Matlab code and can therefore not directly be used in a standard pc.
- The wheelchair will not be able to modify the map of the house as it moves along, e.g. add a new furniture to the map such that the path next time will go around the new furniture.

## 1.3 Project Requirements

Based on the projects limitations in section 1.2, the requirements for the project will be presented.

### 1.3.1 Localization Requirements

The localization requirements are based on two properties. The first property is that the localization can not become better than the precision of the map and the distance measure used in the project. The second property is what the user will find acceptable. It is believed that the user will not be able to distinct a rather larger difference in position, but a higher distinction in angle of the wheelchair. Therefore the requirements for the localization is set to:

$$\begin{aligned} |X_{est} - X_{true}| &< 10 \text{ cm} \\ |Y_{est} - Y_{true}| &< 10 \text{ cm} \\ |\theta_{est} - \theta_{true}| &< 2^\circ \end{aligned}$$

There ought to be requirements for the speed of the localization, because it is out of the scope of this project.

### 1.3.2 Cognition Requirements

The requirements for the cognition are as follows.

- It must be possible to drive the wheelchair from the start position to any position in the house where the wheelchair could go if it was driven manually.
- The wheelchair must not collide with walls or obstacles.

### 1.3.3 Motion Control Requirements

The requirements for the motion control are

- The motion control must drive the wheelchair with the desired speed and orientation.
- The motion control must keep track of its own position and orientation or describe its difference in position and orientation with a stochastic model.

### 1.3.4 Acceptance Test

All the requirements will be tested in chapter 8 on page 83.

## 1.4 Reading Directions

This thesis is partitioned into parts.

Part I is this introduction, and an analysis of the system to be developed, including HW and SW overviews, and a use case formulation of the topics to be covered.

Part II - Autonomous Navigation covers the main work of this project. In this part the various algorithms for making the wheelchair autonomous are developed, implemented and tested. This includes four chapters: Perception, Localisation, Cognition and Motion Control. The part starts out with an introductory chapter, which introduces the concept of robot navigation.

Part III contains the results of the project, a discussion of these, and the final conclusion. The future work in this area is also discussed.

Part IV contains the appendix.

# Chapter 2

# Analysis

This chapter describes an analysis of how a wheelchair can become autonomous. For a wheelchair to become autonomous it is necessary that it knows it current position in order to calculate how it moves to the place where the user wants to go. It has to keep track of its current position to determine when it has reached its goal. The wheelchair must be able to plan a path through a house from its current position to its goal. The wheelchair must be able to control the position and velocity in order to give a smooth ride for the user. In order to fulfill these requirements, it is necessary to have sensors to sense position and velocity of the wheelchair and computational power to calculate paths and keep control of position and velocity.

The first goal of this chapter is to specify the requested functionality of the system. This is done using the 'use case' method.

The second goal of this chapter is to give an overview of the system to be developed. First, the map of the house is described. Then a hardware overview is given, including the interfaces between the different hardware components. After this a an overview of the main software loop is given.

## 2.1 System Functionality

In this section the functionality of the system is specified, in order to determine what functionality the developed system must have. This is specified using use cases, as defined in the Unified Modelling Language (see for example [Douglass, 1999]). The system use case is shown in Figure 2.1 on the next page.

USE CASE
UML

### 2.1.1 Actor Description

**User**
The user of the wheelchair. In this context the user acts directly on the use cases. In practice, the user would interact with the system using the eye tracker and a computer screen, as described in section 2.3 on page 17.
**Map**

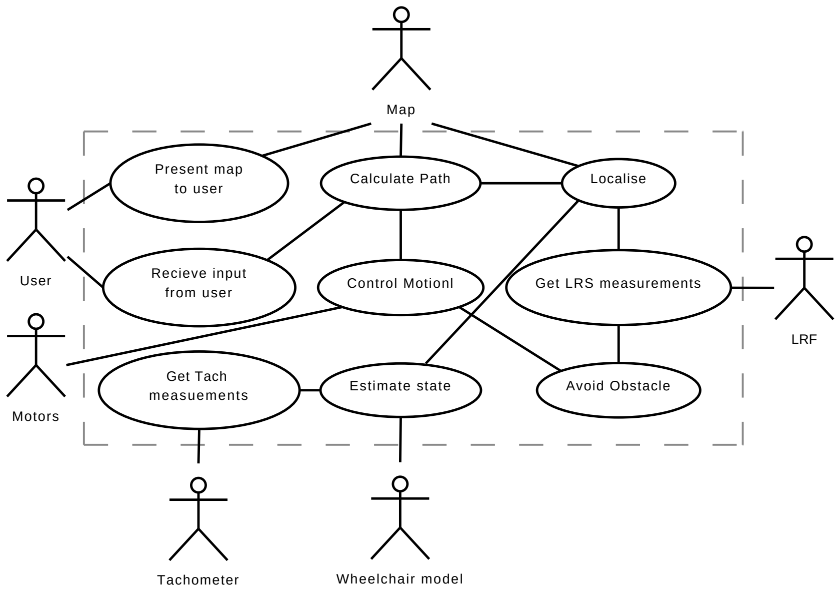


Figure 2.1: The use cases in the system.

The floor plan of the building the wheelchair is operating in. This actor is described in more detail in section 2.2 on the next page.
**LRF**
The Laser Range Finder. This actor detects the range to objects in the vicinity of the AWC, in a 240°arc.
**Tachometer**
This actor measures the current motor rpm, for each of the two motors.
**Motors**
The DC motors on the wheelchair, used to moving the wheelchair.
**Wheelchair model**
The model of the wheelchair, used in state estimation and control.

### 2.1.2 Use Case Description

**Present map to user**
The user is presented with the floor plan of the building on a computer monitor, enabling the user to select a destination by looking at the point on the monitor.
**Receive input from user**
The coordinates for the destination is received from the eye tracker and interpreted. These coordinates are forwarded to the *Calculate Path* use case.
**Get Tach measurements**
The measurements from the actor *Tachometer* must be acquired.
**Calculate path**

Using the map, the current position and the target position, a route to the destination is calculated.
**Control Motion**
On movement input from the *Calculate Path* or *Avoid Obstacle* use cases, the wheelchair is controlled in the requested direction, in a way the is acceptable for the user.
**Estimate State**
Using the use case *Get Tach measurements* and the actor *Wheelchair Model*, the current state of the wheelchair must be estimated.
**Localise**
Using the actor *map*, and the use cases *Estimate State* and *Get LRS Measurements* the current position and orientation of the wheelchair is determined. This includes both global localisation, without prior knowledge, and position tracking.
**Get LRS Measurements**
Get the measurements from the actor *LRS*.
**Avoid Obstacle**
Using the use cases *Get LRF measurements* and *Control Motion*, any obstacles must be avoided.

These use cases define what is to be done in the project, and to some extent the way these different topics are connected. In the next part the different topics are developed.

## 2.2 Map of the House

The map of the house is known in advance because it is necessary in order to make a visual identification of the goal pose. The map is a bitmap where each pixel is 1 cm on each side. In robot navigation, this is known as an - occupancy grid based map. By using a bitmap, it is possible to use different colors in the map to represent a difference in walls and obstacles and areas where the wheelchair can not be, such as under a table. When the laser range scanner scans the room, a table will consists of four legs. So in order to tell the path planning algorithm that it can not move between the legs of the table, a different color than the walls is used to mark were the table top is. Another advantage of using this type of map, is that it poses no restrictions on the shape or size of the obstacles, as for example a feature based map might. The bitmap used for testing in laboratory can be seen in figure 2.2.

OCCUPANCY GRID

## 2.3 System Description

The system chosen for this project consist of a monitor and an eye tracker for communicating with the user, a computer equipped with a National Instruments 6071E io card and running Mathworks Matlab XPC that controls velocity and orientation of the wheelchair, a second computer that can estimate position and plan paths and a wheelchair equipped with dc motors and sensors

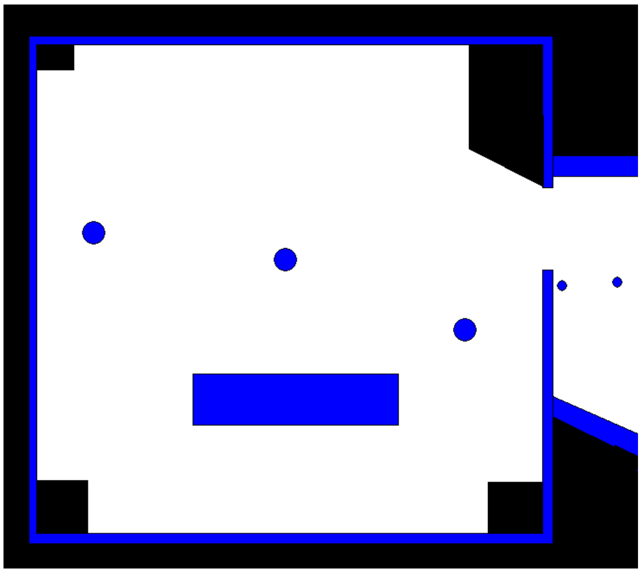


Figure 2.2: Bitmap or occupancy grid base map for the laboratory used in this project. The black pixels are walls and obstacles and the blue are areas where the wheelchair can not be.

(for sensor selection see section 4 on page 25). The system and connections can be seen in figure 2.3.

The xPC configuration is chosen for several reasons. It is easy to set up, it runs simulink functions directly, it is capable of executing software hard real time and it has a predefined simulink block set for the available sampling card. The xPC solution is not suitable when moving to a real wheelchair because it requires two standard PC mounted to the wheelchair, one for xPC hard real time computations and the other for user interface and soft real time computations. There is not executed any computational hard tasks on the xPC, so it is mainly used for real time purposes. Therefore the interface to the xPC computer from the standard PC is chosen such that it is possible to exchange the xPC with a microcontroller. The remaining software is still intended to run on a standard PC. This is due to the interface with the user.

### 2.3.1 Interfaces

The interfaces between the different pieces of hardware can also be seen in figure 2.3. These interfaces are chosen because they are readily available and easy to set up.

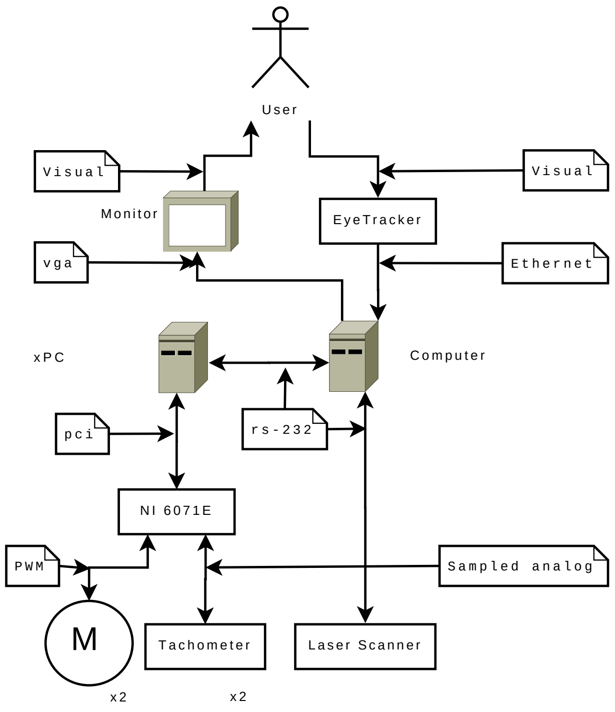


Figure 2.3: An overview of the system hardware.

| **Interface:** | **Connection:** | **Details:** |
|---|---|---|
| Computer - Monitor | VGA / DVI | Std. |
| Computer - Eye Tracker | Ethernet | Custom protocol |
| Computer - xPC | RS-232 | Simulink instrumentation protocol |
| Computer - Laser scanner | RS-232 | Hokuyo protocol |
| xPC - DC Motors | NI 6071E analog out 1+2 | -10 - 10V |
| xPC - Tachometer | NI 6071E analog in 1+2 | -10 - 10V |

## 2.4 Software Platform

The software platform for this project will be Matlab. The advantage of Matlab is that it has many built in algorithms and it is easy to manipulate numbers. A disadvantage is that on a final product, the code has to be rewritten to work in a different environment such as C, because it has to be integrated with the software from the Health Science Project. Matlab also has the disadvantage that it can not as standard make preemptive scheduling of functions, so the functions has to run sequential and thus potentially only in soft real time.

## 2.5 Software Sequence

The sequence that the software will be following can be seen in figure 2.5. The

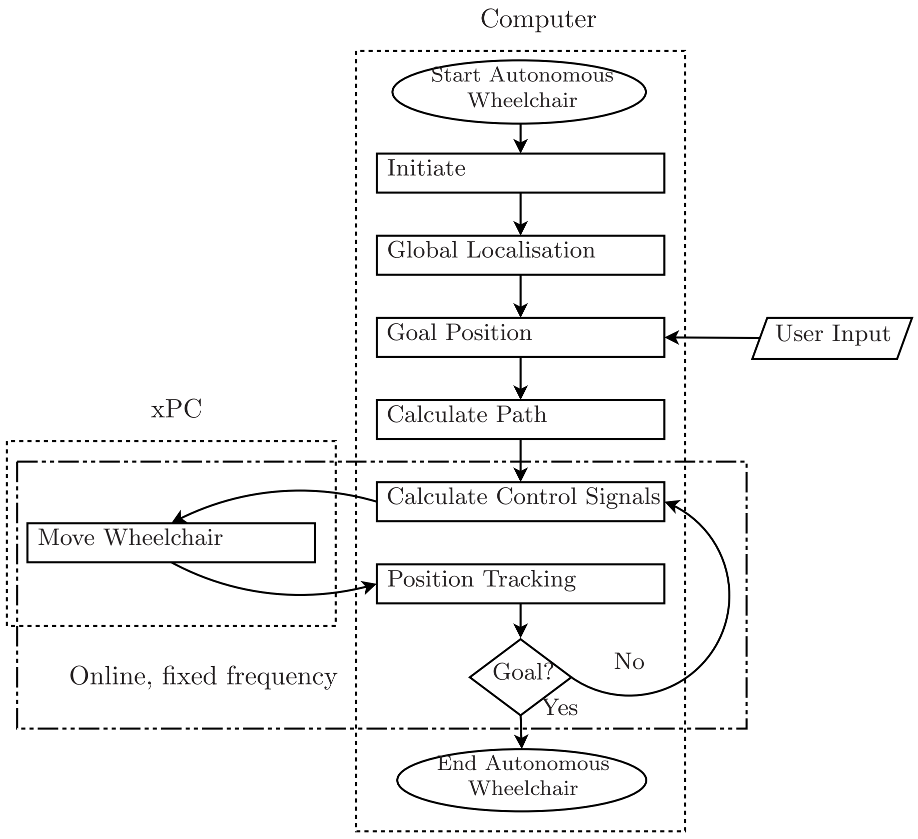


Figure 2.4: Flowchart for the autonomous wheelchair software.

initial block sets up all necessary signals and port and determines if a new map is present and loads this if it is the case. The next task is to localise the wheelchair in the map. When the wheelchair knows it pose, it is possible for the user to give the goal position where he wants to go. This information will with later projects come from the eye tracker. When the wheelchair knows where it is and where it want to go, it calculates a path through the house. When the path is calculated, it is time to move the wheelchair along the path. This is where the wheelchair calculates which control inputs are necessary to move the wheelchair, these are sent to the xPC computer which handles the motion of the wheelchair. Based on the distance the wheelchair has moved and a new laser scanning, the wheelchair calculates its new pose and if it is not at the goal, it will calculate new control inputs.

# Part II

# Autonomous Navigation

Chapter 3

# Navigation Introduction

The purpose of this project is to design an autonomous wheelchair. Autonomous in this context covers navigation, that is, the wheelchair must be able to autonomously navigate a house, apartment, workplace or similar. In part I, the requirements for the project has been established, and the functionalities to be developed has been specified. In this part, the navigation for the wheelchair is developed.

4 building blocks of (robot) navigation

According to [Siegwart & Nourbakhsh, 2004] there are four distinctive building blocks of (robot) navigation. For the AWC to navigate from its current location to the target location, these building blocks must be constructed:

- *Perception*, the robot must sense the environment, and interpret the data.
- *Localisation*, the robot must be aware of its position in the environment.
- *Cognition*, the robot must decide how to archive its goals.
- *Motion control*, the robot must control its movement in a acceptable manner.

These four building blocks enable the robot to navigate, when they are all running continuously (either in parallel or sequentially). This is shown on Figure 3.1.

These four blocks will be developed in this part. By the end of the part, the AWC will be able to acquire data from its sensors, determine its location in the environment, plan a path to its target location, and control its motion so it drives to this location. Each block is treated independently, as a software component, in the sense that any of the blocks could be used separately in another project, and one or more blocks could easily be replaced by another implementation [1]. To archive this, the signals between the blocks must be defined. This is seen on Figure 3.2.

The signals between the navigation blocks are seen table 3.1:

[1] This follows the concepts of software reuseability and decoupling, as used in Object Orientated Analysis and Design. See for example Deitel & Deitel [2000]

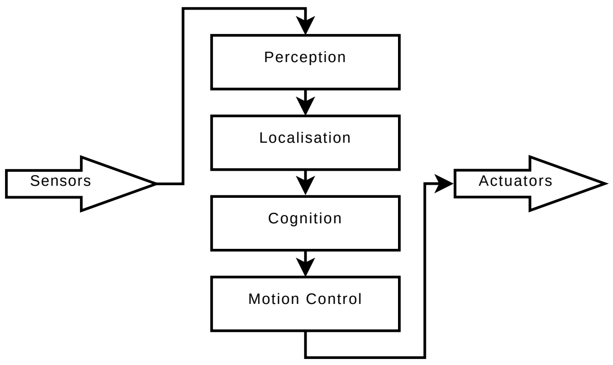


Figure 3.1: The four building blocks of (robot) navigation.

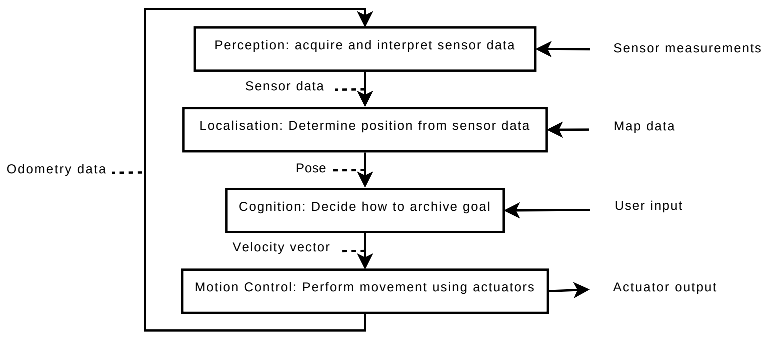

Perception: acquire and interpret sensor data
Sensor measurements
Sensor data
Localisation: Determine position from sensor data
Map data
Odometry data
Pose
Cognition: Decide how to archive goal
User input
Velocity vector
Motion Control: Perform movement using actuators
Actuator output

Figure 3.2: The data between the four building blocks.

Table 3.1: The signals between the navigation blocks.

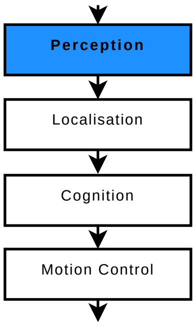
| | |
|---|---|
| Pose: | $X = [x\ y\ \theta]^T$ |
| Sensor data: | Odometry: $\Delta X = u_t = [\Delta x\, \Delta y\, \Delta\theta]^T$<br>LRS: $z_t = \begin{bmatrix} \theta_1 & z_1 \\ \vdots & \vdots \\ \theta_k & z_k \end{bmatrix}$ |
| Velocity Vector: | $\begin{bmatrix} x \\ y \end{bmatrix}$ |
| Odometry data: | Encoder pulses |

The next chapter deals with the Perception block, which is selection and communication with the sensors, and the interpretation of the acquired data. Then the localisation block is treated. After this, in the cognition phase, the robot will decide where to go, and how to get there. This block outputs a velocity vector. How this vector is used to archive movement is treated in the Motion Control section.

# Chapter 4

# Perception

*This chapter is about the perception phase of robot navigation. This is the first of the four building blocks. For the AWC to be able to move in and interact with its environment, it must first sense this environment, and interpret these data. In this phase, the sensors for this are selected, and models for them are developed. These will be used in the later phases.*



## 4.1 The Odometry

The AWC must control its motion in a manner that is comfortable for the user, with no abrupt changes in speed and direction. Therefore a feedback controller is used (see section 7 on page 67). This controller needs a measurement of the current velocity of each of the AWCs driving wheels. To acquire these, it is decided to mount a pulse encoder on each drive shaft. These are active, proprioceptive sensors, meaning that they actively measure the internal state of the robot. The output from these sensors are used as inputs for a Kalman filter, which estimates the true velocity of the AWC. See section 7.1.1 on page 68. The Kalman filter outputs, among others, the change in pose $u_t = [\Delta x \Delta y \Delta \theta]^T$. This is the change since last iteration of the x and y coordinates and angle. These are needed in the localisation phase, described in section 5 on page 35, but in order to be used there they must include noise. The chosen localisation algorithm is probabilistic, and as such needs a probabilistic motion model. This model must take a pose as input, and, using the above mentioned output from the Kalman filter, return a new pose. This returned pose will be an estimate of the robots new pose, after the movement, taking sensor noise into account. This is seen on figure 4.1.

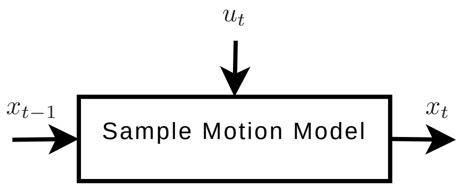


Figure 4.1: The sample motion model with inputs and outputs.

The model is designed and implemented as described in [Thrun et al., 2005, page 136]. In this model, the rotational and translational noise of the odometry can be modelled. This makes it possible to adjust the model to the current wheelchair implementation, as it can take factors such as wheel slip and imprecise wheel diameters into account. In figures 4.2 and 4.3 two different outputs of the motion model are seen. These are made by inputting a pose situated in the origin, and a odometry reading of 10 cm movement in the x axis and 15 cm movement in the y axis. The model is then asked to produce 500 new poses based on this. On figure 4.2 the rotational uncertainty is large, and the translational uncertainty is small, and on figure 4.3 the translational uncertainty is large, and the rotational uncertainty is small.

Generating New Poses

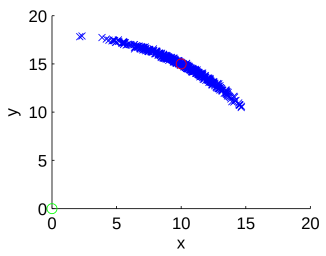


Figure 4.2: Rotational uncertainty.

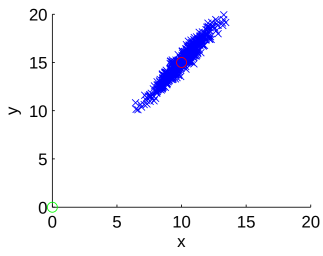


Figure 4.3: Translational uncertainty.

The noise parameters used in this project can be seen on figure 4.4. This has some uncertainty in both rotational and translational movement. This amount noise is present in the model to represent the wheel slip in the Lego model. By modelling the noise in the probabilistic motion model, the localisation algorithm will function.

## 4.2 The Laser Range Scanner

The odometry data can be used for position tracking (see section 5 on page 35), but not for global localisation, and it cannot be used for obstacle avoidance (see section 6 on page 53). For this, the AWC needs a type of sensor that

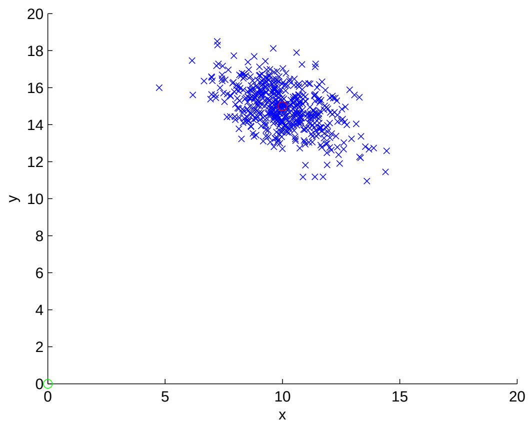


Figure 4.4: The used motion model.

can detect features in the apartment (walls, furniture, people). This will be active, exteroceptive sensors, that is, sensors that actively acquires information about the robots environment. There are several options, two commonly used sensors are vision systems and range finders. Vision systems are not used in this project, due to the complexity of the implementation. There are also different options in range finders: sound emitting and receiving sensors (ultrasonic range finders, also known as sonars) and light emitting and receiving sensors, which can use either normal (typically infrared) or laser light. The sound sensors have a wider spread, which gives them a larger detection area for a single sensor, but also limits the angular resolution severely. For this reason they are discarded in this project. The light sensors can be used as a fixed, single point sensor (such as the infrared Sharp GP series or the laser based Sick range finders), or a ”scanner” type sensor (such as the Hokuyo PBS-03JN Infrared Range Scanner or the Hokuyo URG-04LX Laser Range Scanner), that rotates the sensor to scan a wide area for obstacles. The fixed sensors are by far the cheapest, and can be more accurate, but does have problems in this project, as illustrated in Figure 4.5.

Distance Sensor Selection

In Figure 4.5, the AWC moves from an initial position in the lower right side, along the dotted line, through the door, and along the wall in the top room. Here, it hits the blue object, because the fixed sensor could not detect it. It is seen that a single fixed point light sensor is not enough to perform obstacle avoidance. This could be solved with several fixed point sensors, but this would drive the combined price up to the vicinity of the scanner type sensor, which gives a much higher angular resolution. A single fixed point sensor with also make localisation much more difficult. Therefore, it is chosen to use a scanner type sensor in this project. The Hokuyo PBS-03JN Infrared Range Scanner was tested first due to low cost, but was found to have a too low distance resolution, as well as a undocumented programming interface. Instead, the Hokuyo URG-04LX Laser Range Scanner (LRS) will be used, as it has a very high angular

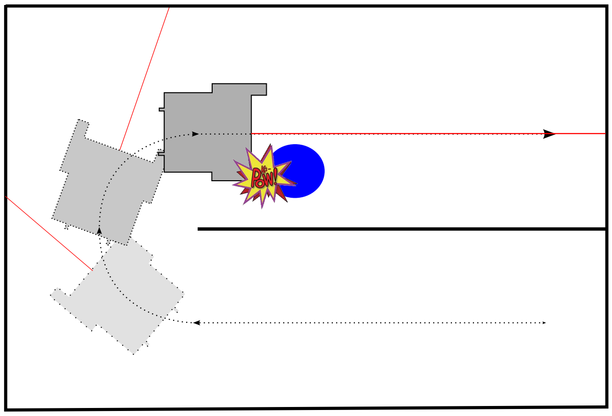


Figure 4.5: The problem with using a fixed point laser range finder.

and distance resolution, and a well documented programming interface. The specifications for the LRS are seen in table 4.1.

Table 4.1: Hokuyo URG-04LX specifications

| | |
|---|---|
| Detectable range: | 20mm to 4000mm |
| Distance resolution: | 1mm |
| Distance accuracy 20-1000mm: | +/- 10mm |
| Distance accuracy 1000-4000mm: | 1% |
| Scanning area: | 240° |
| Angular resolution: | 0.36° |
| Scanning rate: | 10Hz |
| Interface: | USB and RS-232 |
| Power consumption: | 2.5W (5V, 500mA) |
| Peak power consumption: | 4W (5V, 800mA) |
| Size and weight: | 50 x 50 x 70mm, 160g |

The Hokuyo URG-04LX is connected to the system with RS-232, as seen on Figure 2.3 on page 19. The connection is setup according to [Hokuyo, 2004], and the LRS is polled for distance data when needed. It returns (up to) 768 individual distance measurements, which are placed in a matrix as:

$$z_t = \begin{bmatrix} \theta_1 & z_1 \\ \vdots & \vdots \\ \theta_k & z_k \end{bmatrix} \tag{4.1}$$

where

$z_t$ is the sensor data with angles.

$\theta_k$ is the angle of range measurement k, in degrees, with 0 degrees being straight ahead, and positive angles going counterclockwise.

$z_k$ is the distance of measurement k in $[cm]$.

$k$ is the number of distance measurements.

### 4.2.1 Modelling the LRS

The localisation algorithm, described in section 5 on page 35, needs a model of the LRS. More specifically, the Monte Carlo Localisation (MCL) algorithm used, needs to calculate the probability of a sensor reading being correct, given a proposal state and the map. This LRS model is designed and implemented as in [Thrun et al., 2005, page 153]. The needed probability is denoted as:

$$p(z_t|x_t, m) \tag{4.2}$$

where

$z_t$ is the sensor reading.

$x_t$ is the proposal state.

$m$ is the map.

This probability is calculated using a model of the sensor, which includes noise and other uncertainties, and the map. The MCL algorithm needs to be able to sample from the conditional probability distribution $p(z_t|x_t, m)$. This will be explained further in section 5 on page 35, and the map is described in section 2.2 on page 17. The map is used to raytrace a distance from the proposal state to the walls. These are then compared with the measured distance from the LRS. The sensor models inputs and output are seen on figure 4.6.

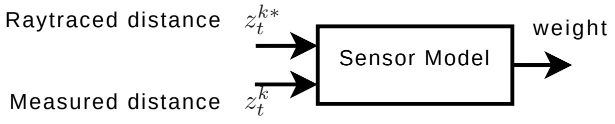


Figure 4.6: The sensor model with inputs and output.

The probability of a sensor reading matching a proposal state is modelled in the following. It is modelled as a probability density function, with four parts, each representing a different type of sensor reading. The most obvious type of reading is that the reading is correct, but corrupted by noise in the sensor itself. Another type of reading comes from objects that are not in the map, such as people and movable furniture. Third, if the room is large, the LRS may reach its maximum distance. And fourth, sometimes a LRS might suffer a random failure and report a wrong distance. Other factors, such as the material of the walls, or the angle of the walls, are not modelled explicitly, but are implicit in the four above mentioned factors. Each of these will be described.

**Correct range and measurement noise:** Ideally, the LRS would return the precise distance to any object in its path. However, factors such as the wall material, and the sensor resolution, means that an amount of noise must be modelled. The probability of the current range measurement being correct is modelled as a narrow Gaussian distribution, and is denoted $p_{hit}(z_t^k|x_t, m)$. The mean of this Gaussian is the raytraced distance to the object denoted $z_t^{k*}$. This is calculated using raytracing, from the proposal state (particle) in the direction of the current laser beam, until an object is encountered in the map. The probability for this measurement is thus:

$$p_{hit}(z_t^k|x_t, m) = \begin{cases} \eta\mathcal{N}(z_t^k; z_t^{k*}; \sigma_{hit}^2) & \text{if } 0 \leq z_t^k \leq z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

where

$p_{hit}(z_t^k|x_t, m)$ is the probability of the measurement being correct.

$\eta$ is a normaliser, which is:

$$\eta = \left( \int_0^{z_{max}} \mathcal{N}(z_t^k; z_t^{k*}; \sigma_{hit}^2) dz_t^k \right)^{-1} \tag{4.4}$$

$z_{max}$ is the maximum detection distance of the LRS.

Gaussian Function

$\mathcal{N}(z_t^k; z_t^{k*}; \sigma_{hit}^2)$ is a Gaussian function:

$$\mathcal{N}(z_t^k; z_t^{k*}; \sigma_{hit}^2) = \frac{1}{\sqrt{2\pi\sigma_{hit}^2}} e^{-\frac{1}{2}\frac{(z_t^k - z_t^{k*})^2}{\sigma_{hit}^2}} \tag{4.5}$$

where

$z_t^k$ current distance measurement.

$z_t^{k*}$ is the raytraced range, calculated using raytracing.

$\sigma_{hit}$ is the standard deviation.

This probability distribution is seen on Figure 4.7 on the next page. The standard deviation $\sigma_{hit}$ is an parameter of the model, which must be adjusted to fit the current environment, the properties of the AMCL algorithm and the specific LRS. Also, when used with a particle filter, this standard deviation must be modelled larger than the actual LRS uncertainty. This is necessary, because the number of particles in such a filter is not infinite, this uncertainty modelled here must take into account that no particle will be situated in the exact correct pose.

**Unmapped objects:** The map, see section 5 on page 35, maps the objects that are known to be stationary, that is, walls, big furniture, and the like. However, movable objects (people, chairs, etc) are not included in the map. In this project they are modelled as sensor noise, and included in the sensor model as unmapped objects. These objects will cause the LRS to report a short distance, shorter than the raytraced distance to the nearest mapped object. The probability of the measurement hitting a unmapped object decreases with

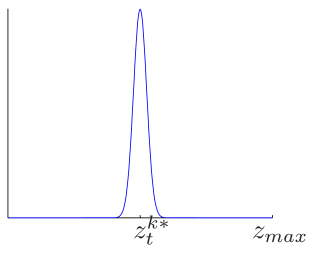


Figure 4.7: $p_{hit}(z_t^k|x_t, m)$

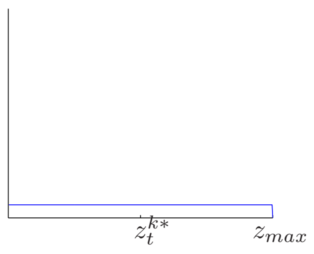


Figure 4.9: $p_{rand}(z_t^k|x_t, m)$

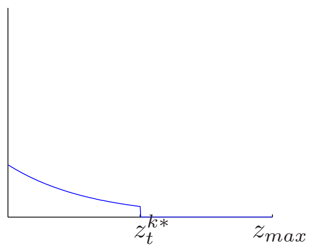


Figure 4.8: $p_{short}(z_t^k|x_t, m)$

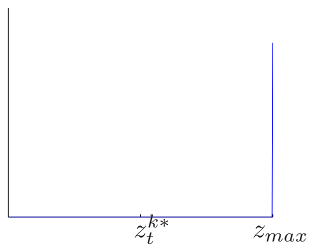


Figure 4.10: $p_{max}(z_t^k|x_t, m)$

distance. This is because, if two unmapped objects are in line, it is the closest one that is detected. This is mapped as an exponential function: EXPONENTIAL FUNCTION

$$p_{short}(z_t^k|x_t, m) = \begin{cases} \eta\lambda_{short}e^{-\lambda_{short}z_t^k} & \text{if } 0 \leq z_t^k \leq z_t^{k*} \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

where

$p_{short}(z_t^k|x_t, m)$ is the probability of a unmapped object.

$\eta$ is a normaliser, which is:

$$\eta = \frac{1}{1 - e^{-\lambda_{short}z_t^{k*}}} \tag{4.7}$$

$\lambda_{short}$ is a model parameter.

This probability distribution is seen on Figure 4.8. The model parameter $\lambda_{short}$ must be tuned to the environment the AWC will operate in, dependent on the amount of unmapped objects that are present.

**Random errors:** Sometimes, the LRS will fail, and simply return random measurements. This can be caused by cross talk between sensors, or just failures, provoked by factors such as electro magnetic noise. It is modelled as a uniform distribution: UNIFORM DISTRIBUTION

$$p_{rand}(z_t^k|x_t,m) = \begin{cases} \frac{1}{z_{max}} & \text{if } 0 \le z_t^k < z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

where

$p_{rand}(z_t^k|x_t,m)$ is the probability of random distance measurement.

This probability distribution is seen on Figure 4.9 on the preceding page.

**Maximum distance:** If the room the AWC is operating in is big, the LRS might not be able to reach the walls or objects. When this happens, the LRS returns a maximum distance reading, in case of the Hokuyo URG-04LX this is 4000mm. Also, the LRS might sometimes report the maximum distance wrongly. This might happen if the object hit by the laser beam is black. These two situations are modelled as a point mass distribution, centered at the maximum distance for the LRS:

Maximum distance measurement

$$p_{max}(z_t^k|x_t,m) = \begin{cases} 1 & \text{if } z = z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{4.9}$$

where

$p_{max}(z_t^k|x_t,m)$ is the probability of a maximum distance measurement.

This probability distribution is seen on Figure 4.10 on the previous page.
The final model is the four probabilities combined, using a weighted average:

$$p(z_t^k|x_t,m) = \begin{pmatrix} z_{hit} \\ z_{short} \\ z_{max} \\ z_{rand} \end{pmatrix}^T \cdot \begin{pmatrix} p_{hit}(z_t^k|x_t,m) \\ p_{short}(z_t^k|x_t,m) \\ p_{max}(z_t^k|x_t,m) \\ p_{rand}(z_t^k|x_t,m) \end{pmatrix} \tag{4.10}$$

where

$z_{hit}$, $z_{short}$, $z_{max}$ and $z_{rand}$ are weighting parameters.

$z_{hit} + z_{short} + z_{max} + z_{rand} = 1$.

This results in a probability distribution seen on figure 4.11.
The parameters in the model are $\sigma_{hit}$, $\lambda_{short}$, $z_{hit}$, $z_{short}$, $z_{max}$ and $z_{rand}$. They are currently set according to literature and best estimates. However, as this project is a proof of concept, the tests are run in a static environment, without any large unmapped obstacles. Therefore, the exponential part of the probability distribution is left out. Also, in the function that fetches the distance measurements from the LRS, any measurements that are at the maximum distance are filtered out. Therefore the maximum distance part is also left out. These two are omitted by setting $z_{short} = 0$ and $z_{max} = 0$.

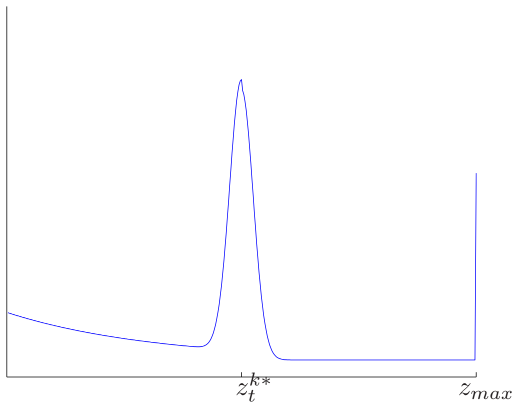
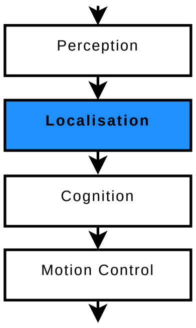


Figure 4.11: The combined distribution $p(z_t|x_t, m)$.

Chapter 5

# Localisation

*This chapter is about the localisation phase of robot navigation. First, the basic concepts of localisation are described. Then the chosen method, Monte Carlo Localisation, and its augmented forms are described. A short introduction to bayesian theory follows, and the a more thorough description of the MCL implementation. Finally, the developed algorithm is tested, and shown to work.*



Localisation is the second of the four elements of navigation (see Figure 3.1 on page 24). In the perception phase, data is acquired and interpreted, in the localisation phase they are used. Localisation is of course the process of localising the robot in an environment (answering the "where am I?" question; "the most fundamental problem to providing a mobile robot with autonomous capabilities" [Cox, 1991]). But what is the environment, and how accurate does the localisation need to be? Does the robot need to know its GPS coordinates, or does it need to know its position relative to some local landmarks? These questions needs answering, in order to select the appropriate localisation technique.

THE MOST FUNDAMENTAL PROBLEM

The intuitive way to localise a robot is to use a GPS receiver. However, these devices are only accurate to within a few meters, they have some difficult noise characteristics, they have trouble operating indoor, and more importantly, they only provide absolute coordinates. Often, a robot would need to know its position relative to the immediate environment, such as walls, furniture, trees, houses, etc. To get this information with only a GPS receiver, the absolute coordinates for all fixed objects on the map would have to be acquired, for every house or apartment the AWC will operate in. This is considered too time consuming and difficult for end users. Therefore, localisation will in this project refer to the problem of localising the robot with reference to the local environment. This problem can be divided into two basic types of localisation: *position tracking* and *global localisation*. In position tracking the initial pose [1] of the robot is known, and the problem reduces to correction of the odometry errors using the sensor data. In global localisation the robots initial pose is unknown, and the localisation algorithm must be able to determine it from scratch, as well as perform position tracking. This relates to the *kidnapped robot problem*, where the robot, after having successfully determined its pose, is "kidnapped" to another position and angle, and has to accommodate this. This is the hardest of the localisation problems [Thrun et al., 2005, page 232].

POSITION TRACKING
GLOBAL LOCALISATION
KIDNAPPED ROBOT PROBLEM

In some mobile robotic applications localisation may not be necessary. It is entirely possible to construct a mobile robot that follow walls, wires in the ground, etc, and does not know its pose (absolute or relative). For example, an autonomous lawn mower could perform very well with only the knowledge that it must never cross the wire and an algorithm for getting around the field. This is called behavior-based navigation. In this project, the AWC must be able to perform global localisation relative to the features (walls and stationary furniture) in the apartment. It must also be able to perform position tracking during AWC movement. If the AWC could solve the kidnapped robot problem, it would be even better, as this goes a long way toward robustness against localisation failures, which is the concept of the algorithm having determined a incorrect pose. This is because, from the localisation algorithms point of view, a localisation failure and the kidnapped robot problem is the same. In both cases the algorithm believes it is at some pose, which is wrong, and it has to be able to find the correct pose. Therefore, solving the kidnapped robot problem also provides a level of robustness against these localisation failures.

The localisation result, an estimation of the AWCs pose, is needed in the next phase of navigation, cognition, where the AWC has to plan a route from where it is, to the location requested by the user.

Localisation of a mobile robot is a difficult problem, and any assistance possible should be exploited. One way of assisting in localisation is to use artificial landmarks; this could be colour markings, radio beacons, wires, etc. In this project however, this would mean that these landmarks must be placed in every apartment where the AWCs should operate, a (too) expensive and time consuming task. The AWC must, therefore, be able to perform localisation using only onboard sensors and the natural environment around it. The nature of this project does provide one assistance to the localisation algorithm: The map. The blueprint of the apartment must be presented to the user for target selection, so it is available up front for localisation. This is called map-based navigation, as the localisation algorithm compares the sensor data with the map to estimate the robots pose. The main advantage of this is that the map can be replaced easily for each new apartment or house the AWC needs to operate in. As the map must be displayed to the user, the map must be stored in a human readable format. The simplest way, which is also suited for machine interpretation, is an two dimensional array of numerical values, with zeros for free space, and non zero values for occupied space. That is, a bitmap file. This

THE MAP

[1]The pose is the position and angle of a mobile robot

is also known as a occupancy grid-based map; the finite resolution creates a grid, and each grid cell is either occupied or not. A map example is shown on Figure 5.1.

Occupancy Grid

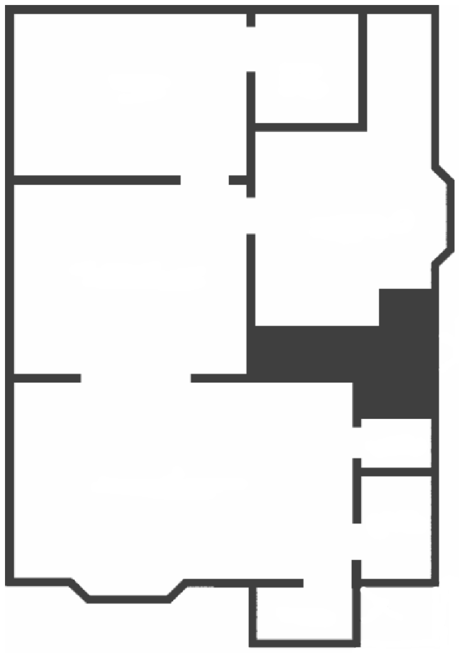


Figure 5.1: An example map that can be used for localisation.

Two of the main challenges in localisation is sensor noise and effector noise. The sensor noise, in case of the laser range scanner, can be limited resolution, min/max distance problems, the reflectivity of the object, etc. Effector noise is gear slip, wheel slip, unequal wheel diameter, pulse encoder resolution, unequal floor contact, etc. This noise must be dealt with by the localisation algorithm. This is accomplished by modelling it, and was done in the Perception phase.

## 5.1 The Monte Carlo Localisation Method

In the previous section several requirements for the localisation algorithm were set up. Most importantly, the robot must be able to perform global localisation, artificial landmarks are not allowed, and a map of the fixed features of the apartment is available. There exist two common probabilistic map-based approaches to robot localisation: Kalman filters and particle filters [Fox et al., 1999],[Dellaert et al., 1999]. The Kalman filter has been proved to perform well in position tracking, however it is not applicable to global localisation [Thrun et al., 2005, page 230][Kwok et al., 2002], as the probability distributions has to be Gaussian. Kalman filtering, including the linearised versions Extended Kalman Filtering and Unscented Kalman Filtering only works well

Kalman Filter

if the uncertainty of the robots pose is small (and can be approximated by a Gaussian). This is not the case when performing global localisation, where the uncertainty is infinite. Because of this, the Kalman filter is rejected for localisation in this project.

Particle Filter

The other promising option is particle filters, also known as survival-of-the-fittest filters, as they share some properties with genetic algorithms. The basic concept is to represent the probability distributions by a set of samples, or particles. In particle filters there are no restrictions on the probability distributions, so the initial distribution can be uniform over the entire state space, the solution to the global localisation problem. In particle filters it is also possible to inject a number of particles spread out randomly over the state space, even after successful localisation. This will enable the robot to re-localise in the event of a localisation failure, the solution to the kidnapped robot problem. Because of these properties particle filters are selected for this project. The remainder of this chapter will describe the properties, theory and implementation of a particle filter for localisation of the AWC.

References

The contents of this section is based on a number of different references . For a general introduction to robotics, see [Siegwart & Nourbakhsh, 2004]. A tutorial on particle filters for mobile robot localisation is found in [Rekleitis, 2004], and a thorough treatment of monte carlo methods in general is found in [Doucet et al., 2001]. [Kwok et al., 2002] deals with the computational burden of particle filters. However, most of the material used is from four authors: Wolfram Burgard, Frank Dellaert, Dieter Fox and Sebastian Thrun. The papers [Dellaert et al., 1999], [Fox et al., 1999] and [Thrun et al., 2000] are from these authors. [Fox, 2003] deals with the topic of making the particle filter adaptive. The book "Probabilistic Robotics" [Thrun et al., 2005] by Thrun, Burgaard and Fox are used extensively throughout this chapter, as it contains a thorough treatment of robot localisation using probabilistic methods, and most of the contents of the four papers of the same authors.

### 5.1.1 The Concept

The basic concept of using a particle filter for mobile robot localisation is fairly simple. For global localisation, all that is needed is the map and the distance measurements from the LRS. Then the job is to find a pose in the map, which covers the entire state space, where the LRS measurements match the map. This is done by simply "guessing" at a large number of different poses, spread uniformly over the state space. For each of these guesses, which are the particles of the particle filter, the LRS measurements are compared with the map, to determine how well they fit. A sketch of this is shown on figure 5.2 on the next page.

On this figure, there are three pose guesses/particles, marked 1, 2 and 3. For each of these, a number of raytraces are performed in the directions of the LRS measurements, starting from the guess coordinates. In this example, there are five measurements. The raytraces return the distances from the pose guess to the obstacles in the map. These are then compared with the LRS measurements. If they match, weight is added to that guess. In the end,

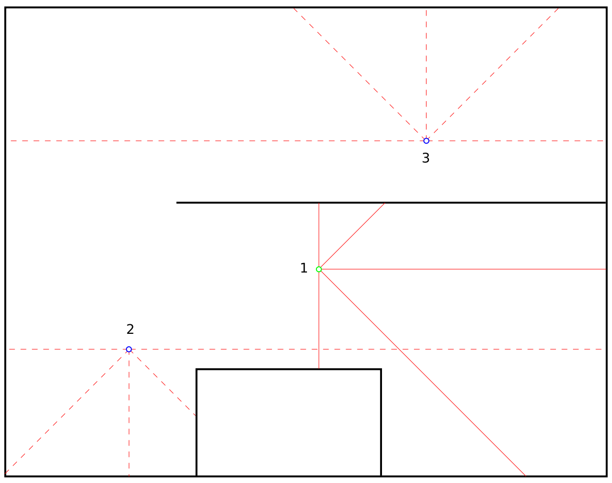


Figure 5.2: A sketch of the raytraced laser scan from three particles.

the particle with the highest weight is the one that most closely matches the true position. If the AWC is moving, each particle is simply moved the same amount. Here, the odometry data is used. There are more, important, steps in Monte Carlo Localisation, but this example illustrates the concept.

### 5.1.2 MCL Explained

The goal of the localisation algorithm is to estimate the pose of the AWC as well as possible, using all available data. There are typically, and also in this project, two types of data available: Odometry data and a form of environment data, in this project laser range measurements. Both data sets are used for pose estimate. The odometry data is used to predict the pose, and the LRS measurements are used to correct this prediction. This is the well known *predictor-corrector* structure, also used in the Kalman filter. When only the odometry readings are used, the pose uncertainty grows, as the errors accumulate. Then a LRS measurement is taken, and used to correct the prediction. When this correction is applied the uncertainty shrinks. This is illustrated on Figure 5.3. Note that in the particle filter the uncertainty is not propagated directly, and this is only a sketch for illustration purposes.

PREDICTOR-CORRECTOR

The basic particle filter thus consist of these two steps, and a selection or calculation of the best unique estimate to forward to the cognition phase. This is the Markov Localisation algorithm, and is shown on Figure 5.4.

The box labelled "MaL" illustrates the basic, grid based, Markov Localisation and the prediction and correction phases are highlighted. Initially, the particles are distributed uniformly over the state space. This is done once, as shown in the top box in the figure. The particle set represent the probability function of the state estimate. Therefore, this is uniform in the beginning, which means that the robot can be anywhere in the possible state space. The state of the robot is:

MAL

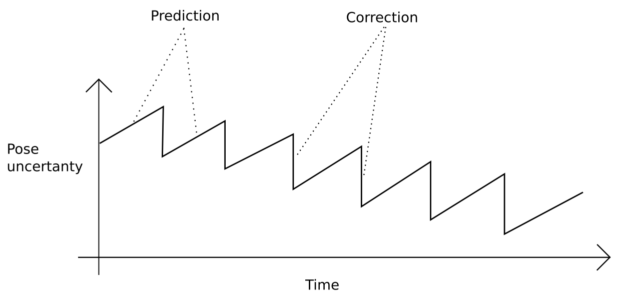


Figure 5.3: A sketch of the pose uncertainty when prediction and correction is applied.

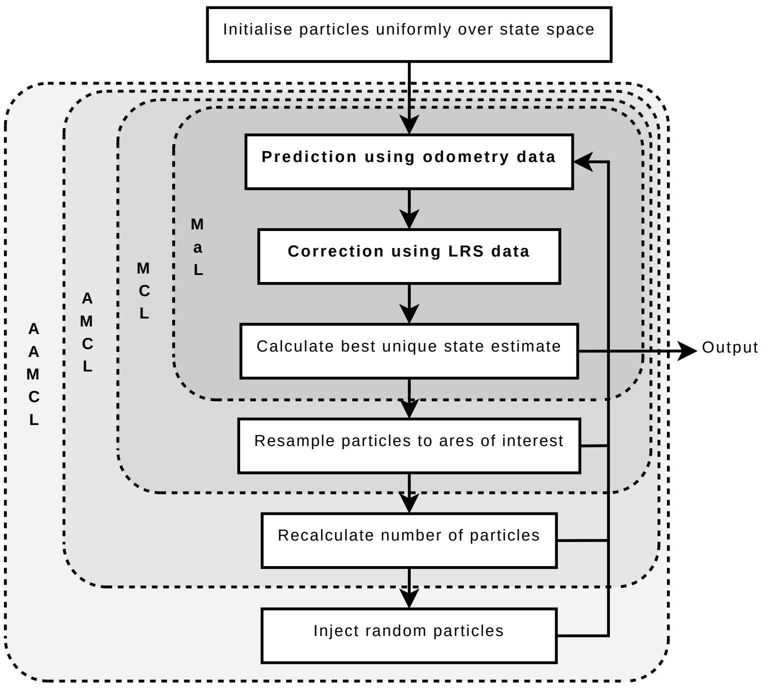

Initialise particles uniformly over state space
Prediction using odometry data
Correction using LRS data
Calculate best unique state estimate
Output
Resample particles to ares of interest
Recalculate number of particles
Inject random particles
MaL
MCL
AMCL
AAMCL

Figure 5.4: The structure of Markov Localisation, Monte Carlo Localisation, Adaptive Monte Carlo Localisation and Augmented Adaptive Monte Carlo Localisation.

$$X = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{5.1}$$

where

$X$ is the state of the robot, or its pose: $\mathbb{R}^2 \times [0; 2\pi]$

$x$ is the x coordinate of the robots position in [cm]

$y$ is the y coordinate of the robots position in [cm]

$\theta$ is the angle of the robot in [°]

The particle set is M particles. Each particle consist of a state estimate and an assigned weight:

$$\begin{aligned} s &= \{s_j | s_j \in X \times \mathbb{R}^+\} \\ s_j &= \begin{bmatrix} X_j \\ w_j \end{bmatrix} \text{ for } j = 1 \ldots M \end{aligned} \tag{5.2}$$

where

$s$ is the particle set, representing the probability function of the robots pose

$s_j$ is an individual particle

$X_j$ is the state guess for this particle

$w_j$ is the weight, that is, how correct this state guess is.

$M$ is the total number of particles. This depends on the number of dimensions in the state space, three in this project, the limits of the state space (the map size in the x and y coordinates and $2\pi$ in the angle) and the desired resolution.

INITIAL PARTICLE SET

In the initial particle set the state estimates are distributed uniformly over the state space, and the weights are equal. After the initial particle set is created, the recursive filter algorithm runs. The prediction step predicts the position and angle of all particles to reflect the estimated movement of the AWC, based on the odometry measurements and the kinematic model. This is described in section 4.1 on page 25. The kinematic model is used to predict the AWC movement, and the pulse encoder measurements are used to correct the estimate. This results in the odometry data for the prediction step in the MCL algorithm. This data is of the form $\Delta x$, $\Delta y$ and $\Delta\theta$. These values are added to the corresponding $x$, $y$ and $\theta$ values for each of the particles in the set $s_{t-1}$. To get the new particle set $s'_t$, the uncertainty must also be modelled. Here, the probabilistic motion model is used. This is performed by the abstract function *Action*:

PREDICTION

ACTION

$$s'_t = \text{Act}(a_t, s_{t-1}) \tag{5.3}$$

where

$s'_t$ is the particle set after prediction

$a_t$ is the action (odometry) data

$s_{t-1}$ is the prior particle set

The state estimate part of the particles ($X_j$) are predicted as a function of the odometry data, using the probabilistic motion model developed in section 4.1 on page 25, resulting in a new particle set. The prediction is the application of the motion model on the odometry data, giving some translatory amount, which is added to each particles $X_j$ values. This new particle set is then corrected using the LRS data, by the abstract function *See*:

SEE

$$s_t = \text{See}(o_t, s'_t) \tag{5.4}$$

where

$s_t$ is the current particle set after correction

$o_t$ is the observation (laser range scanner) data

CORRECTION

In the correction step the LRS data is used to assign a weight, $w_j$, to each particle, indicating the quality of the state estimate. This is done using the probabilistic sensor model developed in section 4.2.1 on page 29. Not all distance measurements from the LRS are used, as this would be to computationally demanding. In this project 18 are used. This is a compromise that delivers acceptable accuracy while keeping the time-to-execute as low as possible. Using the sensor model means that the particle set with weights represents the discrete probability function of the robots pose, given the odometry and LRS data.

UNIQUE ESTIMATE

The final step in the basic Markov Localisation algorithm is to select or calculate an unique state estimate to return. This can be done in several ways. The obvious way is to simply select the particle with the highest weight. Another possibility is to calculate the weighted mean of all particles, and finally the robust mean, which is the weighted mean in a window around the best particle, can be calculated. This gives the best state estimate [Rekleitis, 2004], but it also the most computationally demanding solution. In this project, the particle with the highest weight is used, as this the least computationally demanding solution, and it gives good results.

The basic, grid based, Markov filter can perform global localisation, however it is computationally very intensive. This is because this algorithm maintains a finely spaced grid over the state space, and for each grid it has to continually predict and update the probability of the robot being at this position [Fox, 1998]. To reduce this load, the Markov Localisation algorithm is replaced by the Monte Carlo Localisation (MCL), as seen on Figure 5.4 on page 40 in the box labelled "MCL". In Markov Localisation, all the particles (which are grids in that algorithm) stay where they are first initialised, with only the action model to adjust their position in the state space. This means that the particles stays spread out over the entire state space, even if the localisation has succeeded. The particles weight indicate how good each particles state estimate is, but the prediction and correction step must still be performed on particles with a low or near zero weight, even though these particles does not add any value to the

MCL

quality of the final state estimate. To resolve this problem, a resampling step is added, after the best unique state estimate calculation. The purpose of the resampling step is to "move" the particles in the direction of the best estimate. This is done by eliminating the particles with low weight, and duplicate the particles with high weight. This is a very important step in MCL, as it focuses the computational resources where they are needed. This has been called "the real trick" of particle filters [Thrun et al., 2005, page 99], and is also known as survival of the fittest.

RESAMPLING

Monte Carlo Localisation is more efficient than Markov Localisation, but it is still possible to improve its performance. Consider the case of a very good state estimate. Because of the resampling step introduced before, all the particles converge on the true state. Again, the prediction and correction steps are performed on all the particles, even if only a few particles are necessary to track the state. Instead of wasting resources, the number of particles can be adjusted on-the-fly. If the state estimate is good, M is decreased, and if the state estimate is bad, M is increased. This is the Adaptive Monte Carlo Localisation algorithm, shown in the box labelled "AMCL". This is archived by using the Kullback-Leibler Divergence, which is a measure of the difference between two probability distributions. First, the state space is divided into a histogram, where each bin has the dimensions 60cm x 60cm x 60°. In each iteration of the algorithm, it is determined how many bins in the histogram are non-empty. This shows how spread out the particles are, and as such how much the particles has converged to one pose estimate. Using this measure of convergence, the number of particles needed is adjusted. During global localisation, the particles are spread out over all histogram bins, and the number of particles needed is large. As the particles converge, the number of non-empty bins increase, and the number of particles needed decrease. Thus, the number of particles used during global localisation can be large, and then decrease when performing position tracking.

AMCL
KULLBACK-LEIBLER DIVERGENCE

The AMCL algorithm is capable of performing both global localisation and position tracking, while adjusting the number of particles to the minimum needed. It is not, however, capable of solving the kidnapped robot problem. To do this, it must be augmented, into the Augmented AMCL algorithm, shown in Figure 5.4 in the box labelled "AAMCL". This new augmentation solves the problem by injecting a number of random particles each iteration. These particles are spread out over the entire state space, using a uniform distribution. If the AWC is kidnapped, or looses track of its pose, all the particles from the earlier algorithms will be in the wrong pose, and get a near zero weight. This might, in the earlier algorithms, be unrecoverable. But with the newly injected random particles, some of these will be nearer the new pose, and get a higher weight than the rest of the particles. During the resampling, these new particles will have a higher chance of being duplicated, and the rest of the particles will have a higher chance of being eliminated. This will allow the AWC to relocalise itself. It might not happen in one iteration, as the number of injected particles might not be sufficient to place a particle close to the new pose, but in each iteration new random particles are injected, and at some time one will be placed close to the new pose.

AAMCL

The final AMCL algorithm is seen on Figure 5.4 in the box labelled "AAMCL". The particles are intialised uniformly over the state space before the recursive part of the algorithm. The recursive part consist of *prediction*, *correction*, *best estimate calculation*, *resampling*, *recalculation of the number of particles* and *injection of random particles*.

## 5.2 Bayesian Theory

The Monte Carlo filter is based on probability theory and Bayes rule. In this section a brief overview of these concepts is given, following the reasoning in [Thrun et al., 2000].

In the following, the notation $p(A)$ is used to represent the probability of A. $p(A|B)$ is the conditional probability of A given B.

In the general case, Bayes filters estimate the state X of a dynamical system from sensor measurements. In this specific case, the state is the pose $X$ (5.1), the dynamical system is the AWC and the apartment, and the sensor measurements are the distances and the angles they are measured at from the LRS, and the change in x,y and $\theta$ from the odometry. Monte Carlo localisation works by maintaining a belief of the robots pose, described by a probability density function. This belief, $Bel(x)$, is the posterior distribution, conditioned on the sensor data. The belief is the probability of the state x given the odometry and LRS data:

BELIEF

$$Bel(x_t) = p(x_t|o_t, a_{t-1}, o_{t-1}, a_{t-2}..., o_0) \tag{5.5}$$

where

$Bel(x_t)$ is the probability distribution of the AWCs current pose.

$o$ is the "see", or observation data, from the Laser Range Scanner

$a$ is the "action", or odometry data, from the tachometer

That is, the belief is the probability of the state x being true, given all past LRS and odometry data. The initial belief $Bel(x_0)$ will be uniformly distributed if the AWC must be globally localised, and could be Gaussian if the AWC position is known.

BAYES RULE

Bayes rule states that:

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)}$$

By Bayes rule, (5.5) is transformed to:

$$\begin{aligned} Bel(x_t) &= \frac{p(o_t|x_t, a_{t-1}, ..., o_0)p(x_t|a_{t-1}, ..., o_0)}{p(o_t|a_{t-1}, ..., o_0)} \\ Bel(x_t) &= \alpha p(o_t|x_t, a_{t-1}, ..., o_0)p(x_t|a_{t-1}, ..., o_0) \qquad (5.6) \end{aligned}$$

where

$\alpha$ is a normalisation constant

At this point, the belief depends on the current state, the current and first LRS observation, and all odometry data. But this can be simplified using the Markov assumption, which states that the next state of the system is dependent only on the current state and the last action. This is a acceptable assumption, if not completely correct. For example, one of the wheels on the AWC might have worn more that the other, giving an dependence of previous data. Formally, the Markov assumption for this problem is:

Markov Assumption

$$p(o_t|x_t, a_{t-1}, ..., o_0) = p(o_t|x_t) \tag{5.7}$$

Which means that the probability of the current observation being true given the current state and the previous actions are equal to the probability of the current observation being true given just the current state. Using this, equation (5.6) can be simplified to:

$$Bel(x_t) = \alpha p(o_t|x_t)p(x_t|a_{t-1}, ..., o_0) \tag{5.8}$$

The rightmost term in equation (5.8) is integrated. This is done because in order to compute the probability of the state x, one must integrate over all possible ways this state can be reached given the former state $x_{t-1}$ and the odometry data. By this integration, equation (5.8) becomes equation (5.9):

$$Bel(x_t) = \alpha p(o_t|x_t) \int p(x_t|x_{t-1}, a_{t-1}, ..., o_0)p(x_{t-1}|a_{t-1}, ..., o_0)dx_{t-1} \tag{5.9}$$

Simplifying this using the Markov assumption that the probability of the current state estimate being true, given all previous measurement data, is equal to the probability of the current state estimate being true, given just the one previous odometry data:

$$p(x_t|x_{t-1}, a_{t-1}, ..., o_0) = p(x_t|x_{t-1}, a_{t-1}) \tag{5.10}$$

we get:

$$Bel(x_t) = \alpha p(o_t|x_t) \int p(x_t|x_{t-1}, a_{t-1})p(x_{t-1}|a_{t-1}, ..., o_0) \tag{5.11}$$

By substitution with the original expression for the Belief (5.5) this is transformed into the recursive form used in the MCL algorithm:

$$Bel(x_t) = \alpha p(o_t|x_t) \int p(x_t|x_{t-1}, a_{t-1})Bel(X_{t-1})dx_{t-1} \tag{5.12}$$

where

$Bel(x_t)$ is the probability distribution of the AWCs current pose.

$p(o_t|x_t)$ is the probability of our observation being true given our state

$p(x_t|x_{t-1}, a_{t-1})$ is the probability of our state being true given our previous state and our action (odometry data)

$Bel(x_{t-1})$ is our previous belief

As these functions are typically (and in this project) stationary, or time-invariant, the notation in (5.12) can be relaxed to:

$$Bel(x) = \alpha p(o|x) \int p(x|x', a) Bel(x') dx' \tag{5.13}$$

In this formula, it is seen that the current state, or pose, of the robot is the probability of the current observation being true given the current state, multiplied with the integral of the probability of the current state being true given the previous state and the odometry data multiplied with the previous belief. The integral is necessary, because all the possible ways the new state $x$ can be reached from the old state $x'$ and the action a must be included.

DENSITIES NEEDED

As can be seen, only two conditional densities are needed to implement equation (5.13). This is $p(o|x)$, also known as the sensor model, and $p(x|x', a)$, which is the motion model of the AWC. The nature of these models are dependent on the specific problem, and they influence the solution to the abstract $act()$ (equation (5.3)) and $see()$ (equation (5.4)) functions. The motion model is described in section 4.1 on page 25 and the sensor model is described in section 4.2.1 on page 29.

## 5.3 MCL, AMCL and AAMCL Algorithms

The concept and theory behind the Monte Carlo Localisation algorithm have been explained. Here, the implementation is explained. In this project, a regular MCL algorithm was developed first. This was functional, but, as inherent in this algorithm, it was computationally demanding, as the large amount of particles needed for global localisation were used even when the algorithm only performed position tracking. Therefore, the algorithm was further developed into the Adaptive MCL algorithm. This adjusted the number of particles on-the-fly, so the time for each iteration were significantly lower when performing position tracking. To make the algorithm robust against localisation errors, the algorithm was then further developed into the Augmented AMCL algorithm. These three algorithms will be explained in succession.

### 5.3.1 The Regular MCL Algorithm

The regular MCL algorithm is the basis for the localisation used in this project. The algorithm is shown in algorithm 5.1 on the facing page. As input it takes the previous particle set, $X_{t-1}$, which in the first run is uniformly distributed in the state space, and later is centered around the current location belief. Also, the odometry data, $U_t$, and the latest LRS measurements, $Z_t$, as well as the map, are inputs to this algorithm. The algorithm outputs the new particle set, $X_t$, representing the current pose belief. The particles in this set has been moved, weighted and resampled. Finally, the algorithm also returns the best unique state estimate, $x_{est}$.

MCL INPUTS

MCL OUTPUTS

**Algorithm 5.1** The regular MCL algorithm

```
Input: $X_{t-1}$, $U_t$, $Z_t$, $map$
Output: $X_t$, $x_{est}$
1:  $X_t = X_{temp} = \emptyset$
2:  for $m = 1$ to $M$ do
3:      $x_t^{[m]} = \textbf{action}(u_t, x_{t-1}^{[m]})$
4:      $w_t^{[m]} = \textbf{assignWeight}(z_t, x_t^{[m]}, map)$
5:      $X_{temp} = X_{temp} + \langle x_t^{[m]}, w_t^{[m]} \rangle$
6:  $W_{total} = \sum_{i=1}^{M} w_t^i$
7:  for $i = 1$ to $M$ do
8:      $w_t^{[i]} = \frac{w_t^{[i]}}{W_{total}}$
9:  $X_t = \textbf{resample}(X_{temp})$
10: $index = max(W_t)$
11: $x_{est} = X_t(index)$
12: return $X_t, x_{est}$
```

In lines 2 to 5, all particles in the set are passed through prediction and correction steps.

Moving the particles

In line 3, the current particle is passed through the motion model, implementing the abstract *action* function. This motion model is described in section 4.1 on page 25. This is a probabilistic motion model. It takes a proposal state, or particle $x_{t-1}^{[m]}$, and returns it, moved the distances and angle determined by the odometry data, but with noise included.

Weighting the particles

In line 4 the resulting particle is given a weight, based on the LRS measurements and the map. The assignWeight function takes the current particle $x_t^{[m]}$, the LRS measurements $z_t$ and the map as inputs. The function returns a weight for the particle. Internally, the function first raytraces the distances from the particle to the walls, in the orientations of the LRS measurements. These distances are compared with the LRS measured distances, using the sensor model, described in section 4.2.1 on page 29. The LRS returns many distance readings, each having a different angle from the LRS center. This comparison, using the sensor model, is done for each LRS measurement, and each return a weight. The weight of the particle is the combined probabilities:

$$w_t^{[m]} = p(z_t|x_t, m) = \prod_{k=1}^{k} p(z_t^k|x_t, m) \tag{5.14}$$

where

$w_t^{[m]} = p(z_t|x_t, m)$ is the probability of the LRS measurement being true, given the proposal state and the map.

$p(z_t^k|x_t, m)$ is the probability of the k'th LRS measurement being true, given the proposal state and the map.

$z_t$ is the sensor reading, $z_t =$

$$z_t = \begin{bmatrix} \theta_1 & z_1 \\ \vdots & \vdots \\ \theta_k & z_k \end{bmatrix} \tag{5.15}$$

$z_t^k$ is one individual sensor reading.

$k$ is the number of individual sensor readings.

$x_t$ is the proposal state.

$m$ is the map.

RESAMPLING

In line 5 this new particle, that has been moved and weighted, is added to $X_t$. In lines 6 to 8 the weights are normalised, to ensure that the particles set represents a probability distribution. In line 9, the resampling is performed. Here, the best particles are allowed to continue existing, or even be duplicated, whereas the "bad" particles are eliminated, as in survival of the fittest. This is done probabilistic, meaning that particles with a high weight are likely to be duplicated, and particles with a low weight are likely to be eliminated, but this is only a likelihood. Some of the good particles might sometimes be eliminated, and some bad particles might be duplicated, though this is much less likely than the above scenario [2]. For more information on the resampling step, see appendix D on page 117. In lines 10 and 11 the particle with the highest weight is selected as the best unique pose estimate, and this and the current particle set $X_t$ are returned.

UNIQUE ESTIMATE

## 5.3.2 The Adaptive MCL Algorithm

The regular MCL algorithm works, but does not adjust the number of particles used, even if it has a good pose estimate. Therefore, it is more computationally demanding than necessary. This is changed in the Adaptive MCL algorithm. Here, the number of particles are adjusted on-the-fly. The algorithm is shown on algorithm 5.2 on the facing page.

SAMPLING UP FRONT

A big part of the Adaptive MCL algorithm is similar to the regular MCL algorithm. In line 6, however, the sampling is performed. In contrast to the regular MCL algorithm, where the resampling is performed last, it is performed first here. This line selects particles in a probabilistic way, based on the weights of the particles. For more information, see appendix D on page 117. The outcome of this line is one particle. This particle is moved, weighted and added to the set of particles in the following three lines, as in regular MCL.

ADAPTING THE NUMBER OF PARTICLES

In line 10 it is determined whether this particle belongs to a histogram bin that is empty. If this is the case, the algorithm calculates how many particles are needed, based on the number of non-empty bins. This is done in line 14. See [Thrun et al., 2005, page 263].

[2] This is the origin of the "Monte Carlo" name, as it is a game of chance, like in the famous casino in Monte Carlo.

**Algorithm 5.2** The Adaptive MCL algorithm

**Input:** $X_{t-1}$, $U_t$, $Z_t$, $map$, $M_{min}$
**Output:** $X_t$, $x_{est}$

```
1:  $X_t = \emptyset$
2:  $M = 0, M_X = \infty, k = 0$
3:  for all $b$ in Histogram do
4:      $b = \emptyset$
5:  while true do
6:      draw particle i with probability $w_{t-1}^{[i]}$
7:      $x_t^{[M]}$ = action($u_t, x_{t-1}^{[i]}$)
8:      $w_t^{[M]}$ = assignWeight($z_t, x_t^{[M]}, map$)
9:      $X_t = X_t + \langle x_t^{[M]}, w_t^{[M]} \rangle$
10:     if $x_t^{[M]}$ falls into empty bin $b$ then
11:         $k = k + 1$
12:         $b = !\emptyset$
13:         if $k > 1$ then
14:             $M_X = \frac{k-1}{2\epsilon}\left\{1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}}z_{1-\delta}\right\}^3$
15:         else
16:             $M_X = M_{min}$
17:     $M = M + 1$
18:     if $M > M_X$ and $M > M_{min}$ then
19:         break while
20: $W_{total} = \sum_{i=1}^{M} w_t^i$
21: for $i = 1$ to $M$ do
22:     $w_t^{[i]} = \frac{w_t^{[i]}}{W_{total}}$
23: $index = max(W_t)$
24: $x_{est} = X_t(index)$
25: return $X_t, x_{est}$
```

The algorithms main while loop terminates when the created number of particles match the requested number of particles, $M_X$, but not unless the minimum number $M_{min}$ has been reached.

In lines 20 to 22 the weights are normalised, to ensure that the particles set represents a probability distribution. In lines 23 and 24 the particle with the highest weight is selected as the best unique pose estimate, and this and the current particle set $X_t$ are returned. These last lines are identical to the ones in regular MCL.

### 5.3.3 The Augmented AMCL Algorithm

The AMCL algorithm is capable of performing global localisation using many particles, and position tracking using fewer particles. It is not, however, capable of solving the kidnapped robot problem. To do this, a number of random particles must be added each iteration. This algorithm is shown on algorithm 5.3 on the following page.

**Algorithm 5.3** The Augmented Adaptive MCL algorithm

**Input:** $X_{t-1}$, $U_t$, $Z_t$, $map$, $M_{min}$, $P_{rand}$
**Output:** $X_t$, $x_{est}$

```
1:  $X_t = \emptyset$
2:  $M = 0, M_X = \infty, k = 0$
3:  for all $b$ in Histogram do
4:      $b = \emptyset$
5:  while true do
6:      draw particle i with probability $w_{t-1}^{[i]}$
7:      $x_t^{[M]} = \textbf{action}(u_t, x_{t-1}^{[i]})$
8:      $w_t^{[M]} = \textbf{assignWeight}(z_t, x_t^{[M]}, map)$
9:      $X_t = X_t + \langle x_t^{[M]}, w_t^{[M]} \rangle$
10:     if $x_t^{[M]}$ falls into empty bin $b$ then
11:         $k = k + 1$
12:         $b = !\emptyset$
13:         if $k > 1$ then
14:             $M_X = \frac{k-1}{2\epsilon} \left\{ 1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}} z_{1-\delta} \right\}^3$
15:         else
16:             $M_X = M_{min}$
17:     $M = M + 1$
18:     if $M > M_X$ and $M > M_{min}$ then
19:         break while
20: $index = max(W_t)$
21: $x_{est} = X_t(index)$
22: for $p = 1$ to $P_{rand}$ do
23:     $X_t^{[M]}$ = random particle
24:     $M = M + 1$
25: $W_{total} = \sum_{i=1}^{M} w_t^i$
26: for $i = 1$ to $M$ do
27:     $w_t^{[i]} = \frac{w_t^{[i]}}{W_{total}}$
28: return $X_t, x_{est}$
```

Injecting Random Particles

Most of the Augmented Adaptive MCL algorithm is identical to the Adaptive MCL algorithm. The only difference is in lines 22 to 24, where a number of random particles are added to the set. These particles are uniformly distributed over the entire state space, and as such allows the algorithm to relocalise in the event of localisation failure or if the AWC should get kidnapped (to another pose that is still on the map).

## 5.4 Results

The developed algorithm is tested in chapter 8 on page 83. The tests in that chapter are real world tests, performed in a test environment, using real LRS and odometry data. The final result of a global localisation run with three iterations of the algorithm is seen on figure 5.5. It can be seen that all particles have converged to the true position, to within 4cm in the x and y directions, and 1°. This is an acceptable result.

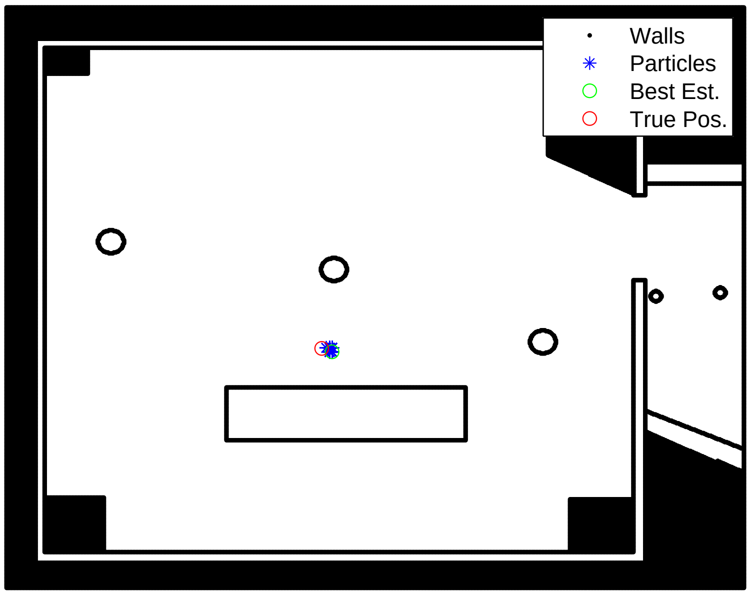


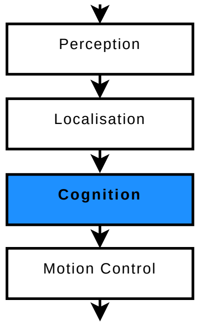
Figure 5.5: Global Localisation result.

The position tracking also performs well, and the algorithm is also capable of solving the kidnapped robot problem, when using injection of random particles. These tests are also shown in chapter 8 on page 83, along with timing and particle counts. It is concluded that the developed algorithm performs better than required.

# Chapter 6

# Cognition

*The third building block in wheelchair navigation is cognition. It is in this building block that the wheelchair makes all its decisions. The most basic decision the wheelchair has to make, is how to get from its initial point to its final point. Another thing that is done in this block is obstacle avoidance. In the next two sections, the path planning and the obstacle avoidance will be described.*



## 6.1 Path Planning

The goal of the path planning is to find a path for the wheelchair from its initial point to its final point. There are several requirements for this path.

- If there exist a path from initial point to final point, the wheelchair has to take this path.
- The wheelchair has to make sure that the chosen path is valid, e.g. that it can pass through the doors.
- If more than one path exist, the wheelchair has to find the shortest path. If the shortest path has a narrow sub path that a person in a wheelchair normally would not take, the wheelchair must take a longer path in order to get more distance to walls and obstacles.

This project has an advantage when it comes to fulfill these requirements. The map of the house is known a priori. Because of this, the wheelchair can make some of the time consuming calculations a priori in order to make faster decisions when the wheelchair is running. Therefore the path planning is split into three parts.

Road Map

1. First time a new map is loaded into the wheelchair it calculates a road maps in the entire house. The road map will be used as a standard path between the rooms of the house. This can be seen in figure 6.1.
2. When the user wants to move the wheelchair into another location, the path that the wheelchair will be driving is calculated before the wheelchair begins to move.
3. When the wheelchair is moving, the guidance of the wheelchair is based on waypoints on the road map and the distance to walls and obstacles.

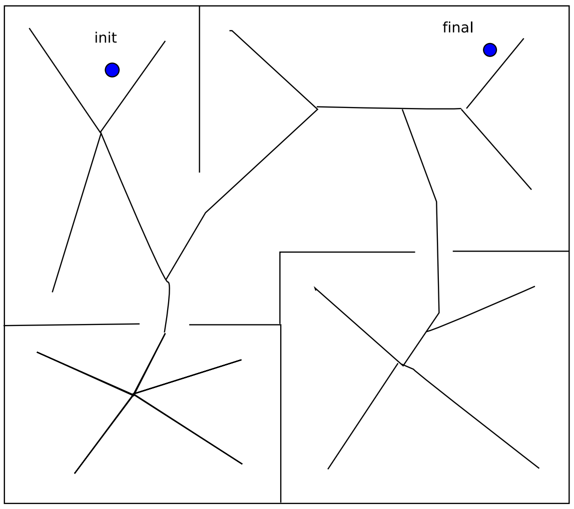


Figure 6.1: House where road maps are calculated a priori. The online calculations is then reduced to finding a path from the initial point to the highway and from the highway to the final point.

Configuration space

In order to calculate both road map and guidance it is useful to calculate the configuration space of the wheelchair [Latombe, 1991]. The configuration space, $\mathcal{C}$, is a space where the wheelchair maps to a single point. A wheelchair $\mathcal{A}$ with reference frame $\mathcal{F}_{\mathcal{A}}$, in a workspace $\mathcal{W} = \mathbb{R}^2$ with reference frame $\mathcal{F}_{\mathcal{W}}$, is a rigid object if every point on the wheelchair has a fixed position relative to $\mathcal{F}_{\mathcal{A}}$. Then every point on the wheelchair can be described in $\mathcal{F}_{\mathcal{W}}$ as a function of $\mathcal{F}_{\mathcal{A}}$ relative to $\mathcal{F}_{\mathcal{W}}$ + the position of the point on the wheelchair relative to $\mathcal{F}_{\mathcal{A}}$. The configuration of $\mathcal{A}$ is a specification of position and orientation of $\mathcal{F}_{\mathcal{A}}$ relative to $\mathcal{F}_{\mathcal{W}}$. But instead of defining the position and orientation relative to $\mathcal{F}_{\mathcal{A}}$, position and orientation can be described relative to $\mathcal{F}_{\mathcal{W}}$ thus mapping $\mathcal{A}$ to the single reference point of $\mathcal{F}_{\mathcal{A}}$. The configuration space is calculated as a Minkowski sum, calculated by adding every element of $\mathcal{A}$ to every element of $\mathcal{W}$. In figure 6.2 a sketch of $\mathcal{A}$, $\mathcal{W}$, $\mathcal{F}_{\mathcal{A}}$ and $\mathcal{C}$ is drawn. The

Minkowski Sum

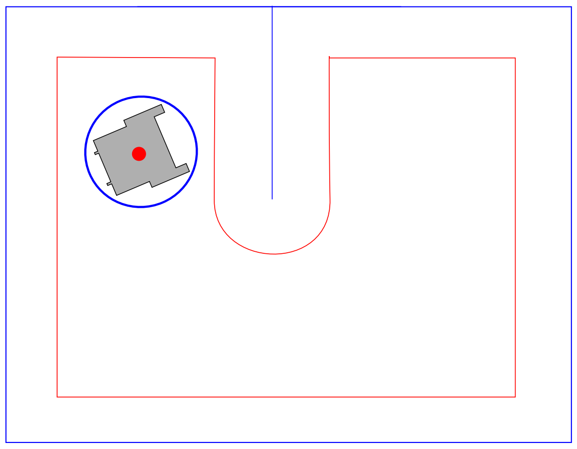


Figure 6.2: Configuration space. The outer blue spaces are wheelchair $\mathcal{A}$ and workspace $\mathcal{W}$. The inner red spaces are configuration space $\mathcal{C}$ and reference point $\mathcal{F}_{\mathcal{A}}$.

configuration of the wheelchair, $\mathcal{A}$, is set to be a circle which circumspheres the entire wheelchair. By doing so, the configuration space is limited to $\mathbb{R}^2$ since the orientation of the wheelchair does not affect the distance from the center of the circle to the outermost point of wheelchair circumsphere. If the configuration of the wheelchair is set to another shape such as a rectangle, the configuration space must also include the orientation of the wheelchair and the complexity of path planning is increased. Path planning in the configuration space is now a matter of finding a continuous path from start to goal.

In order to find a path from start to goal, to construct road maps, it is convenient to generate another topological equivalent set that is a reduction of the configuration space from $\mathbb{R}^2$ to $\mathbb{R}$. There are several methods for this reduction such as cell decomposition and retraction. The cell decomposition divides the configuration space into smaller, bounded regions were a road map will be calculated. This could be useful in this project since a house often consists of individual rooms. The Cell decomposition divides the house into as many cells as vertexes. The cell decomposition can solve the path planning problem, but the solution for this project will be a retraction solution. Another way of calculating road maps could be by using visibility graph. In visibility graphs a line is drawn from all edges to all visible edges. These visibility graphs can be implemented if the map consists of edges and single points as used in this project. A retraction method could be a generalized Voronoi diagram. A generalized Voronoi diagram has the property that it maximizes the distance to the closest walls. The generalized Voronoi diagram is the extension from maximal distance between single points (Voronoi diagram) to maximal distance between lines and curves (generalized Voronoi diagram). In figure 6.3 a generalized Voronoi diagram is shown for the sketch in figure 6.2. A generalized Voronoi diagram consists of straight lines and parabolic lines in a map with polygonal obstacles. Straight lines is between (edge, edge) and (vertex, vertex)

CELL DECOMPOSITION
RETRACTION

GENERALIZED VORONOI DIAGRAM

and parabolic lines are between (edge,vertex).

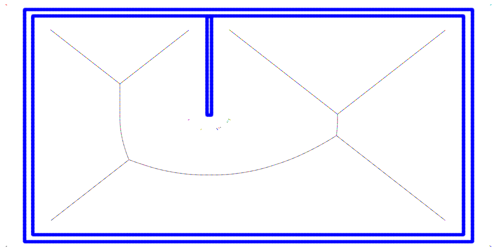


Figure 6.3: Generalized Voronoi diagram for the sketch in figure 6.2.

Because a house not always consists of square furniture and walls and especially because the configuration of the wheelchair is set to a circle, an analytical solution does not always exist. A solution to this is to divide all lines and arches into points [Blaer & Allen, 2003]. Then the generalized Voronoi diagram will consists of only straight lines between every point as in an analytical Voronoi diagram with polygonal obstacles. A sketch of the generalized Voronoi diagram of a house when the map is divided into points, can be seen in figure 6.4. By

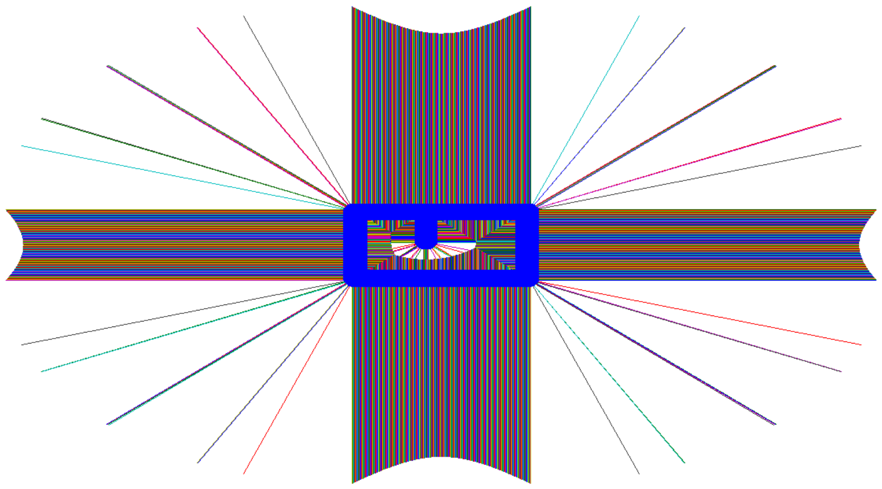


Figure 6.4: Generalized Voronoi diagram in a house where all walls and furniture are mapped as many single points. The Voronoi algorithm constructs the generalized Voronoi diagram around all points and not just the points and edges that the map of the house originally consisted of. The job is now to remove all curves that would not have been a part of the generalized Voronoi diagram in the original map.

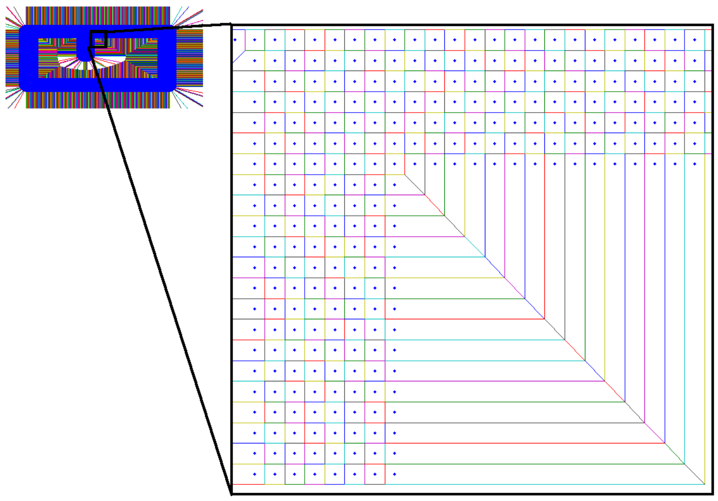


Figure 6.5: Sketch of part of the generalized Voronoi diagram with curves that are constructed when the original map is divided into points. The curves that are in the original generalized Voronoi diagram is the one that does not touch walls or obstacles. The remaining curve in the segment is the diagonal curve.

dividing the walls and obstacles into many single points, the Voronoi algorithm will make a straight Voronoi curve between all points. Many of these curves are not part the Voronoi diagram that would have been constructed if the obstacles and walls were only edges and points as on the original map. A sketch of all the Voronoi curves that are constructed when the map consists of only point can be seen in figure 6.5. By removing any curve that passes through walls and obstacles and curves that are just a single point, the remaining curves forms a good approach to the generalized Voronoi diagram that would have been constructed using the original map of the house. The resulting approximation to the generalized Voronoi diagram can be seen in figure 6.6.

CALCULATING GENERALIZED VORONOI DIAGRAM

The generalized Voronoi diagram is calculated by using Matlabs Voronoi function. It is capable of calculating a Voronoi diagram from points in space. Matlabs Voronoi function is depending on its Delaunay function which again is based on Qhull. The Delaunay function connects three point such that no point are contained in any triangle's circumscribed circle. The Voronoi lines are the perpendicular bisectors to the Delaunay triangles. A sketch of this can be seen in figure 6.7.

HTTP: WWW.QHULL.ORG

## 6.2 Offline Computational Cost

The Voronoi function embedded in Matlab has the computational cost $\mathcal{O}(n)$. Depending on the resolution of the map of the house, several thousand points are needed in order to build a generalized Voronoi diagram. But the generalized Voronoi diagram is only computed once, so the computation of the diagram

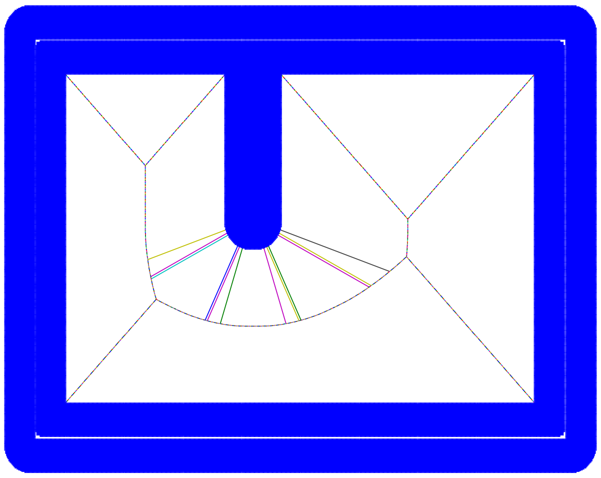


Figure 6.6: Approximation of the generalized Voronoi diagram in configuration space. The nine extra lines from the wall in the middle of the house is a result of the resolution of the map. The end of the wall is resembling an arc, but is in fact consisting of nine corners from which a curve is originating. It is these lines that makes it possible to plan a path using the points where three curves meet.

does not affect the online computation on the wheelchair.

## 6.3 Calculating Path

The generalized Voronoi diagram is the curve that maximizes the distance to the two closest walls or obstacles. Therefore it is the most safe path, but not necessarily the shortest path. A trade off is to use selected point from the generalized Voronoi diagram as waypoints. This way the wheelchair will not always be at the maximum distance to the walls, but it will instead take a shorter path. It is observed that using the divide lines into points and then calculate the generalized Voronoi diagram, that there always exist a point where three curves meet that can be seen from another point where three curves meet and that it is possible to get from any point in the house to a meet point in a straight line and from that meet point to another meet point to any other point in the house by only using straight lines. As it can be seen i figure 6.8 the sketch used in this chapter contains 16 waypoints were three curves meet. The next job is to calculate the shortest path from start to goal. One way to compute the shortest path is to use Dijkstras algorithm. Dijkstras algorithm calculates the shortest path based on the distance between waypoints, visible to each other. This is combined to a lower triangular matrix. The lower triangular

DIJKSTRAS ALGORITHM

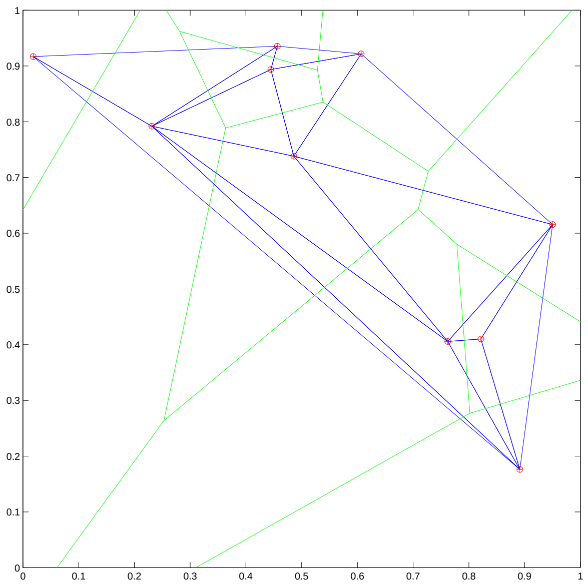


Figure 6.7: Sketch of how Matlab computes Voronoi lines from Delaunay triangles.

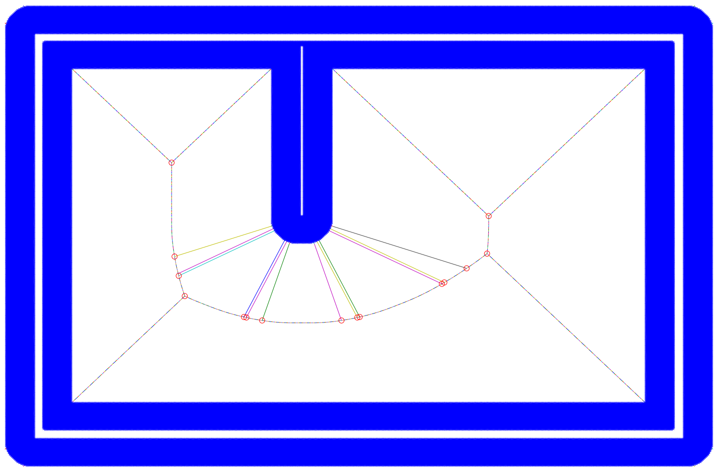


Figure 6.8: The points where three curves meet are marked with a red circle.

matrix for the sketch in the figures above, can be seen in table 6.1.

Dijkstras Algorithm searches the lower triangular sparse matrix containing information on the distance to neighbouring waypoint and returns a vector with the waypoints that the wheelchair must pass in order to move from start to goal without colliding with walls or obstacles. A sketch of this path can be seen in figure 6.9.

| | | | | | |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | ... |
| 0 | 0 | 0 | 0 | 0 | ... |
| 182.9311 | 0 | 0 | 0 | 0 | ... |
| 220.8538 | 580.7327 | 38.3803 | 0 | 0 | ... |
| 328.5111 | 490.3386 | 173.3379 | 143.9766 | 0 | ... |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋱ |

Table 6.1: Part of the triangular matrix used for path planning. The numbers in the table represent the distance between waypoint a and b and zero is no connection.

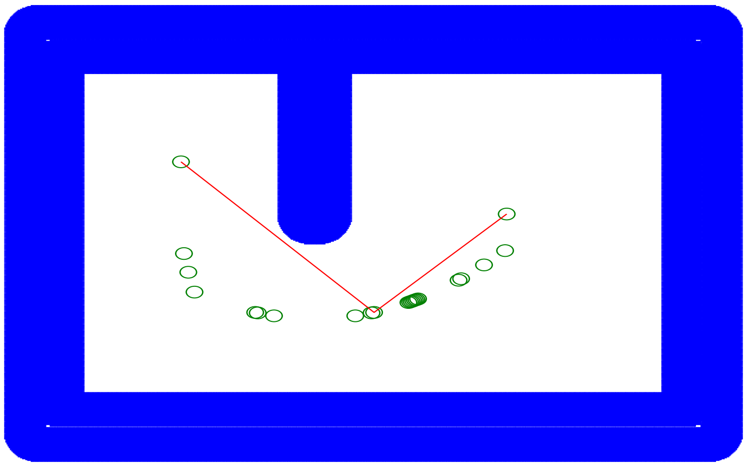


Figure 6.9: A route from start to goal where the path must pass a waypoint on the generalized Voronoi diagram in order to avoid colliding with the wall. it is the shortest path with these waypoints.

With the generalized Voronoi diagram and Dijkstra algorithm it is possible to calculate a path from start to goal. The path will consists of waypoints that the wheelchair has to navigate through. What is not designed yet is how the wheelchair gets from one waypoint to another and what it do if it leaves the path. This is investigated in the following section.

## 6.4 Path Guidance

Path guidance when moving and obstacle avoidance describes the control inputs to the wheelchair when the path has been calculated and the wheelchair is driving. The control signals to the wheelchair specifies the speed and orientation of the wheelchair based on waypoints and distances to walls and obstacles. There are several methods for avoiding obstacles and walls. The method used in this project is based on vector field navigation. Vector field navigation regards the wheelchair as a single particle in $\mathbb{R}^2$ under repulsive forces from walls and obstacles and attractive forces from waypoints and goals. A sketch of such vector fields can be seen in figure 6.10.

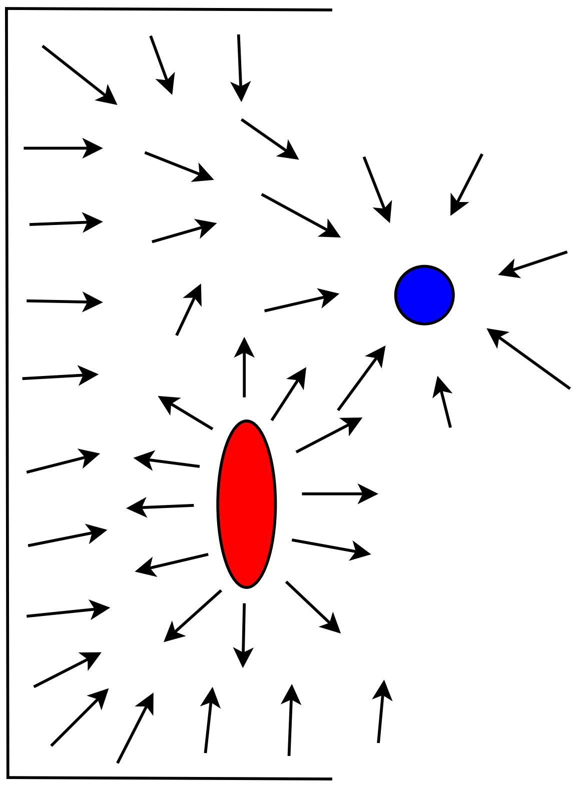


Figure 6.10: Repulsive and attractive vector fields. The walls and the red obstacle have repulsive vector fields and the blue waypoint/goal has an attractive vector field.

Velocity Vector Field

This project however does not use force vector fields but velocity vector fields. By using velocity vector field rather than acceleration vector field, the dynamics of the wheelchair is not used to design the vector field. The vector field from walls and obstacles is a velocity vector pointing away from the wall/obstacle and the magnitude of the velocity is depending on how close the wheelchair is to the wall/obstacle. This method is chosen because the input to the wheelchair is motor voltage which maps to motor velocity and thus it is easier to apply the vector field to to wheelchair. The repulsive velocities from walls and obstacles are limited in range. That is that the repulsive velocity component is zero when the wheelchair is far from walls and obstacles. The attractive velocity on the other hand, always points to the next waypoint, independent on the distance to the waypoint. Because the wheelchair has to transport people, a safe space is defined. The safe space is a space in configuration space which gives the wheelchair some distance to walls and obstacles. It is called a safe space, $\mathcal{C}_{safe}$, because it makes the passenger of the wheelchair more safe when driving round. A sketch of the safe space can be seen in figure 6.11.

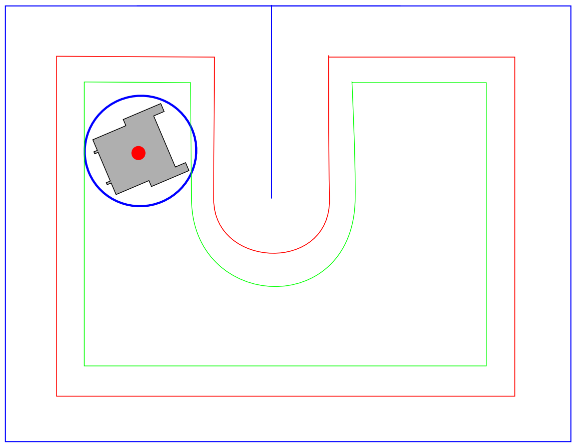


Figure 6.11: Safe space. the green line represents the safe space while the red line is configuration space and the blue is walls and obstacles.

### 6.4.1 Repulsive Velocity Vectors

The repulsive velocity vectors main purpose is to keep the wheelchair away from walls and obstacles that the attractive velocity vector field is guiding it near. When the wheelchairs distance to walls and obstacles are greater than the configuration of the wheelchair, it is in the safe space. A sketch of repulsive velocity vectors pointing away from a wall and an obstacle can be seen in figure 6.12.

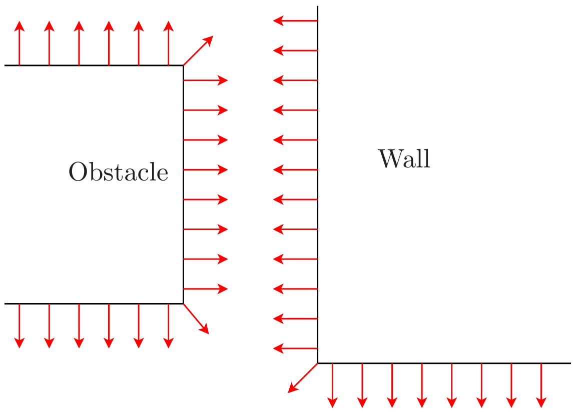


Figure 6.12: A sketch of repulsive velocity vector.

Repulsive velocity vector function

The repulsive velocity vector function is a function where the velocity vector depends on the distance to walls or obstacles. The closer the wheelchair is to the wall or obstacle, the more magnitude the repulsive velocity vector must have in order to guide the wheelchair away from the wall or obstacle. The wheelchair also has to keep a minimal distance to walls and obstacles in order to be in the safe space. Therefore the repulsive velocity vector must have a magnitude which is larger than the maximal velocity of the wheelchair. The distance between the minimal distance and the safe distance is made to make a smooth ride. The magnitude depends on the distance from the wheelchair to the wall or obstacle and ranges from 0 to the magnitude of the minimal distance. And last, when the wheelchair is in the safe distance, the repulsive velocity vector is 0. A sketch of the magnitude of the repulsive velocity vector as a function of the distance can be seen in figure 6.13.

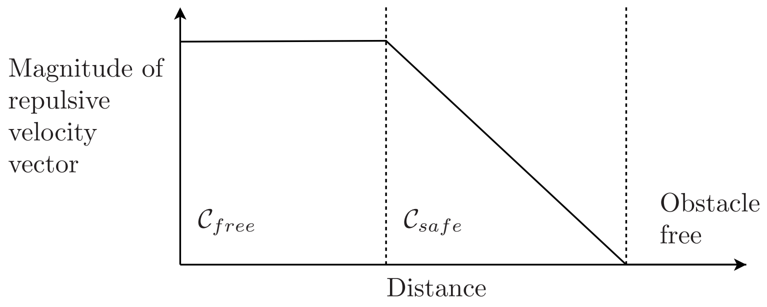


Figure 6.13: Sketch of magnitude of repulsive velocity vectors.

The repulsive velocity vector is calculated based on the measurements from the laser range finder. It has the advantage that objects either not in the map or modelled wrong in the map also gets avoided. It measures the distance and angle from the wheelchair to several points in its environment. Since the laser range finder is not capable of scanning 360°, a repulsive sum of velocity vectors from the front of the wheelchair could become more powerful than the repulsive velocity vector from the rear of the wheelchair. One way to overcome this, it to use the single shortest measurement from the laser range scanner. By pointing the repulsive velocity vector field in the opposite direction of the shortest measurement, the wheelchair will get a resulting velocity that directs in along the obstacle. A sketch of this can be seen in figure 6.14. By tuning the size and magnitude of $\mathcal{C}_{safe}$ and the transition to $\mathcal{C}_{safe}$, it is possible to guide the wheelchair smoothly along the wall or obstacle. As it can be seen in figure 6.15, the wheelchair is interacting with its environment and adapts its route accordingly. If the repulsive velocity vector field is not present the wheelchair would drive much closer to the obstacle.

## 6.5 Cognition Results

The path planning is implemented in the computer for the wheelchair and it gets a map that matches the room that the wheelchair is navigating in.

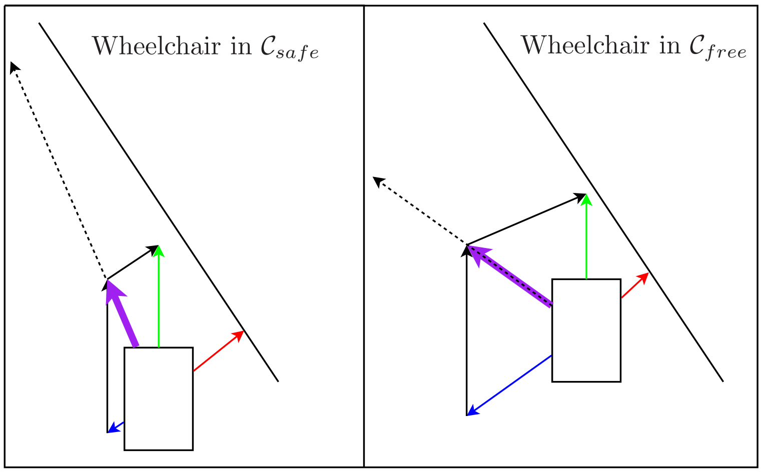

Wheelchair in $\mathcal{C}_{safe}$
Wheelchair in $\mathcal{C}_{free}$

Figure 6.14: Resulting velocity vector field in two cases. The red arrow is the shortest laser measurement, the green arrow is the attractive velocity vector field, the blue arrow is the repulsive velocity vector field and the purple arrow is the resulting velocity vector field. The dotted arrow is just to illustrate the direction of the resulting field. The resulting field is pointing into $\mathcal{C}_{free}$ when the wheelchair is in $\mathcal{C}_{safe}$ and the resulting field is pointing into $\mathcal{C}_{safe}$ when the wheelchair is in $\mathcal{C}_{free}$.

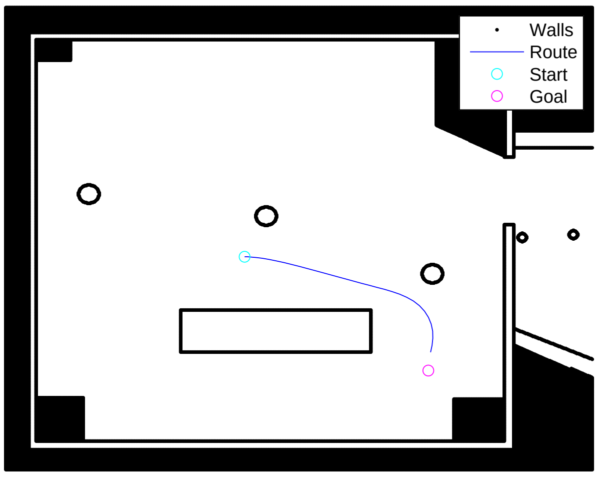


Figure 6.15: Simulated route with wheelchair model when it is sensing it environment.

The waypoints in the map are generated using the Voronoi function. The Voronoi function returns several thousand curves and those curves that passes through a wall or obstacle or continue toward infinity are removed. In the remaining curves all the points where three curves meet are extracted and declared waypoints. When the wheelchair knows it current position and its goal position, it is possible for it to generate a path through the waypoints. The route and all the waypoints in the room can be seen in figure 6.16. In the figure, all the waypoints that the wheelchair may use are shown along with the waypoints that it uses for this test. As it also can be seen, the path does not follow the Voronoi curves but takes the shortest path between start and goal. In the figure it is not possible to see the configuration space, but the path is very close to it at the corner of the box, but it does not touch.

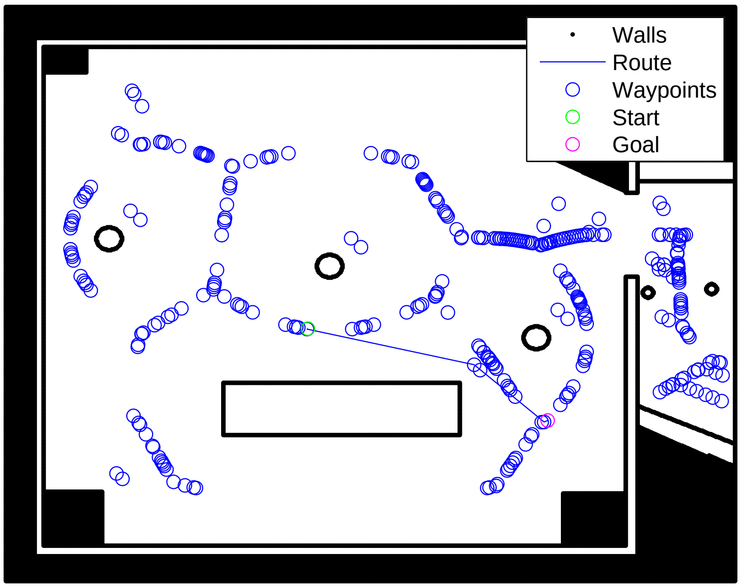


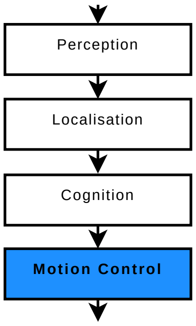
Figure 6.16: Path generated in room.

It was not possible in the given time to tune the repulsive velocity vector field to make a smooth ride for the wheelchair. It was observed that while the resulting velocity vector directs the wheelchair along a wall, the path was neither direct or stable. The wheelchair began to oscillate toward and away from the wall as it drove along. This is because of a mismatch in the magnitude of repulsive velocity vector and the sample time of the same. The computer where the algorithm is implemented currently can not get the measurements fast enough from the laser range scanner due to limitations in the transfer speed and the implementation method in Matlab. This results in a wheelchair that gets too far into $\mathcal{C}_{free}$, gets the measurement that results in a resulting vector which magnitude is large and the wheelchair drives back into $\mathcal{C}_{safe}$ where it again drives deep into $\mathcal{C}_{free}$. A solution to this is to increase the baud rate of the laser range scanner in order to get the measurements faster and then retune the repulsive velocity vector field.

# Chapter 7

# Motion Control

*The purpose of this chapter is to describe how the wheelchair motion control is designed. The section is divided into two parts, a state filtering, estimating and calculating part and a controller part. The controller receives reference signals from the Cognition block and converts them into wheel speed based on the current angle and velocity of the wheelchair.*



The motion control consists of basically two blocks. The control block and the state estimation block. along with these a third block which converts the states to the states needed in localisation. A connected diagram of the control can be seen in figure 7.1. As it can be seen from figure 7.1, there are 7 state

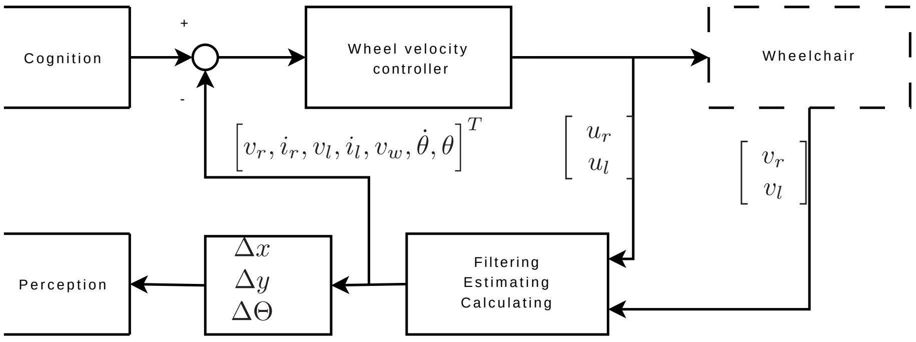

Cognition
+
-
Wheel velocity controller
Wheelchair
$\left[v_r, i_r, v_l, i_l, v_w, \dot{\theta}, \theta\right]^T$
$\begin{bmatrix} u_r \\ u_l \end{bmatrix}$
$\begin{bmatrix} v_r \\ v_l \end{bmatrix}$
Perception
$\Delta x$
$\Delta y$
$\Delta\Theta$
Filtering Estimating Calculating

Figure 7.1: Control Schematic for the wheelchair.

variables.

- $v_r$ is the velocity of the right wheel
- $i_r$ is the current to the right motor
- $v_l$ is the velocity of the left wheel
- $i_l$ is the current to the left motor
- $v_w$ is the velocity of the wheelchair
- $\dot{\theta}$ is the angular velocity of the wheelchair
- $\theta$ is the angle of the wheelchair

The cognition block supplies the wanted states to the controller. The controller, based on the current states, calculates the voltages $\begin{bmatrix} u_r \\ u_l \end{bmatrix}$ to the motors on the wheelchair. From the wheelchair is measured the velocities of the wheels $\begin{bmatrix} v_r \\ v_l \end{bmatrix}$. Based on the input voltages and the output velocities, the $\begin{bmatrix} \text{Filtering} \\ \text{Estimating} \\ \text{Calculating} \end{bmatrix}$ block filter, estimate and calculates its way to the $\left[v_r, i_r, v_l, i_l, v_w, \dot{\theta}, \theta\right]^T$ states. Because the Perception block needs the changes in current position, $(dx, dy)$, these are also calculated.

## 7.1 State Estimation

For the controller to control the wheelchair, it is necessary that the states of the wheelchair is found. In this section, some tools for finding states is designed.

### 7.1.1 Velocity Estimation

In order to measure the velocity of the wheels, each wheel has been fitted with as tach sensor providing 50 pulses per rotation. A picture of one of the discs can be seen in figure 7.2 The output of a tach sensor is a pulse signal where the amplitude of the signal determinates if it is a black or a white field at the sensor. In order to calculate the velocity from such pulse signals, it is used that the wheel has rotated a fixed distance each time the amplitude of the signal changes. It is then a matter of knowing this distance and divide it with the time between two pulses. The conversion can be seen in equation 7.1.

$$v = \frac{d_{wheel}}{dt} \tag{7.1}$$

where

$v$ is the velocity of the wheel [m/s].

$d_{wheel}$ is the distance the wheelchair has moved at the wheel between two pulses.

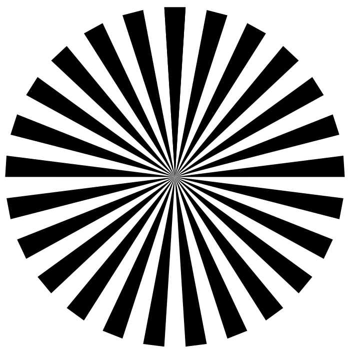


Figure 7.2: Picture of a disc used for velocity measurements.

$dt$ is the time between two pulses.

The output from the tach sensor can be seen in figure 7.3

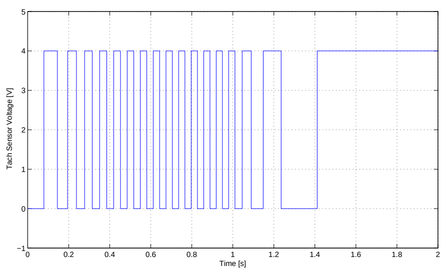


Figure 7.3: Output voltage from the tach sensor.

The implementation of equation 7.1 can be seen in equation 7.2.

$$\begin{aligned} \text{if } z_k &\neq z_{k-1} \text{ ( A new pulse from the tach wheel)} \\ v_k &= \frac{d_{wheel}}{t_{\text{current}} - t_{\text{last pulse}}} \\ \text{else } v_k &= v_{k-1} \end{aligned} \tag{7.2}$$

where

$z_i$ is the sampled tach wheel

$t_{\text{current}}$ is the current sample time at the pulse

$t_{\text{last pulse}}$ is the sample time at the last pulse

$v_i$ is the measured velocity

The conversion performed in equation 7.2 has the limitation that it samples the pulses, thus the resolution depends on the sample frequency. A conversion

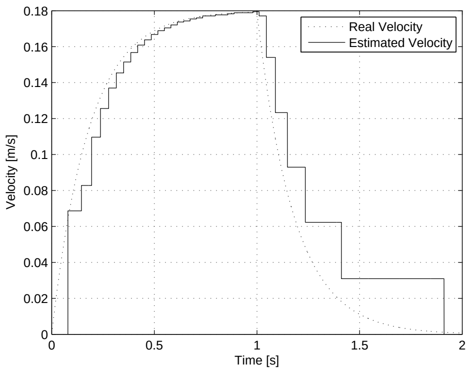


Figure 7.4: Velocity measurement using simple integration over tach measurements.

of the pulses in figure 7.3 along with the original velocity function can be seen in figure 7.4. Here it can be seen that the pulse function does not always fit the original curve because of the sampling time resolution. As it also can be seen, the resolution is especially poor at low speeds. This means that the velocity control gets equally poor and makes an unpleasant travel. If fitting a wheel with better resolution, the sampling time must be smaller to sample all the pulses with a reasonable resolution, but still the controller will see a velocity that either has the same velocity or has sudden changes in velocity. So instead of changing the tach wheel, another method of estimating the velocity is investigated. A method for estimating system states is Kalman filtering. A Kalman filter uses system information to estimate system states corrupted with white noise. Furthermore, the Kalman filter can be extended to predict/estimate states when there are no reliable data [Grewal & Andrews, 2001, Page 128].

KALMAN FILTER

A Kalman filter uses the covariance of the estimated state to calculate the update gain of the filter. When there are no reliable data to estimate from, the covariance of the estimate must grow. The rate of the grow is determined by the covariance of the system noise. The noise covariance is determined in appendix B on page 107.

Basically the Kalman filter is a standard Kalman filter as seen in figure 7.5 [Grewal & Andrews, 2001]:

When there is no reliable data it is not possible to estimate $\hat{x}_k(+)$ from $z_k$. In stead it is possible to build a Luenberger observer to estimate the states $\hat{x}_k(+)$. But because there is no feedback, no gain update, the covariance $P_k$ has to grow. This is simply done by not running the covariance update function. Then the state estimation becomes as in figure 7.6.

By using the Luenberger observer, the left side of figure 7.6 is the same as in figure 7.5 if there is reliable data. But when there is no reliable data, the new covariance is the same as the old one as seen in the right side of figure 7.6. The estimate states in figure 7.6 is pure feed forward since there is no feedback before there is reliable data. The feed forward estimation or Luenberger observer is

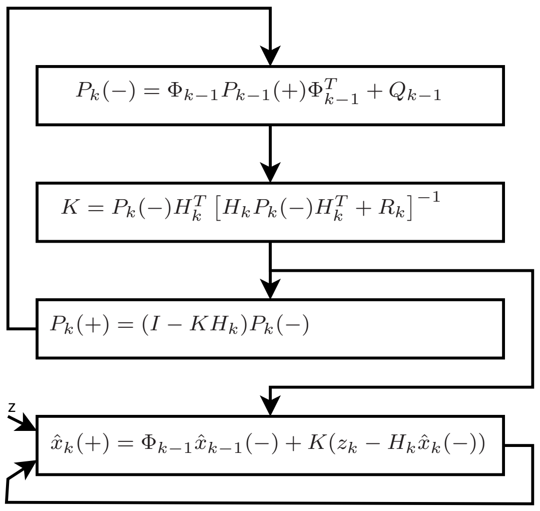
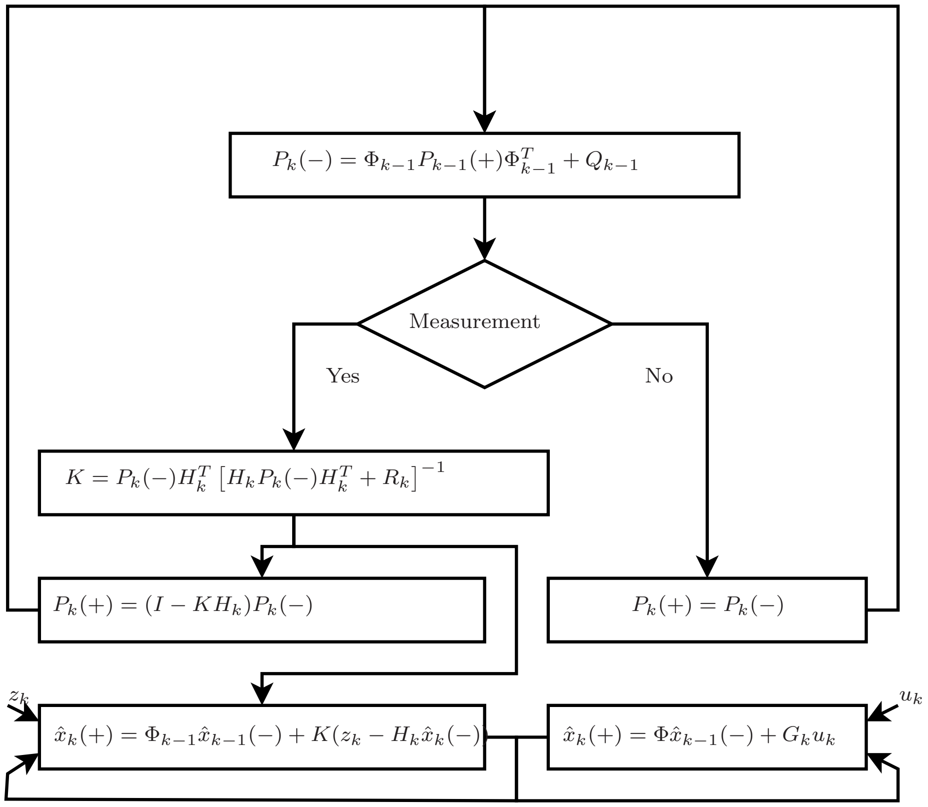

$P_k(-) = \Phi_{k-1}P_{k-1}(+)\Phi_{k-1}^T + Q_{k-1}$
$K = P_k(-)H_k^T\left[H_kP_k(-)H_k^T + R_k\right]^{-1}$
$P_k(+) = (I - KH_k)P_k(-)$
z
$\hat{x}_k(+) = \Phi_{k-1}\hat{x}_{k-1}(-) + K(z_k - H_k\hat{x}_k(-))$

Figure 7.5: Standard Kalman filter implementation.

not described in [Grewal & Andrews, 2001], but only a state update with no external input, e.g. $u_k = 0$. But since there is longer periods when there is no reliable data, it is necessary to include the input in order to have a useful estimation. The input to the Kalman filter along with the control signal is the output from equation 7.2. This signal is chosen because it is the signal with the most information about the velocity of the wheels. To determine if there is a new pulse, the differentiation of the output from equation 7.2 results in a signal that is different from zero when there is a new measurement. The estimated velocity after the Kalman filter can be seen in figure 7.7. It can be seen that the estimated signal is much closer to the real signal than that of figure 7.4. The covariance is varying because there is times when there is no reliable data. This can be seen in figure 7.8.

From the model of the wheelchair in appendix A, the model for a single wheel is extracted. The reason for choosing only a part of the model for estimation is that it is simpler to to implement two identical Kalman filters and then calculate the remaining states on the filtered Kalman states. A reason that it is simpler to implement only a part of the model in the Kalman filter is that the sensors on the wheelchair does not give any information on the direction of the wheelchair, only the speed. Therefore the Kalman filter also has to estimate the direction of the wheel. Because the Kalman filter predicts most of the time at low speed when no pulses passes through the sensor, the Kalman filter predicts only on past states and input signals. It is therefore assumed that the Kalman filter predicts the correct direction the first time the wheelchair begins to move. When this initial direction estimate is correct, the direction of of the unfiltered input signal is multiplied with the sign of the output. This can be done because the input and one of the outputs of the Kalman filter is the same state. This system can be seen in figure 7.9.

Direction estimate



$P_k(-) = \Phi_{k-1} P_{k-1}(+) \Phi_{k-1}^T + Q_{k-1}$
Measurement
Yes
No
$K = P_k(-) H_k^T \left[ H_k P_k(-) H_k^T + R_k \right]^{-1}$
$P_k(+) = (I - K H_k) P_k(-)$
$P_k(+) = P_k(-)$
$z_k$
$u_k$
$\hat{x}_k(+) = \Phi_{k-1} \hat{x}_{k-1}(-) + K(z_k - H_k \hat{x}_k(-))$
$\hat{x}_k(+) = \Phi \hat{x}_{k-1}(-) + G_k u_k$

Figure 7.6: Flow diagram of the Kalman filter when sometimes no reliable data are available.

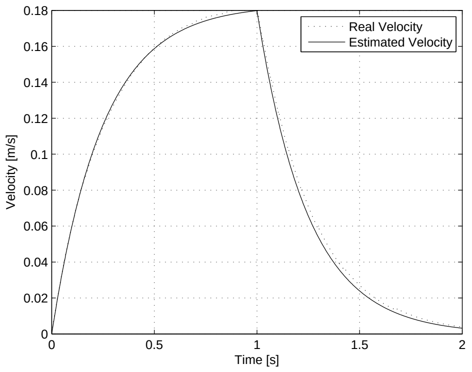


Figure 7.7: Velocity estimation using Kalman filtering.

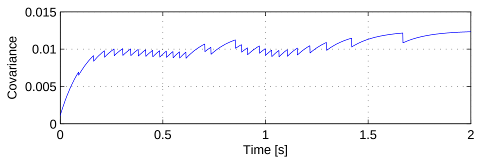


Figure 7.8: Varying covariance because of times with no reliable data.

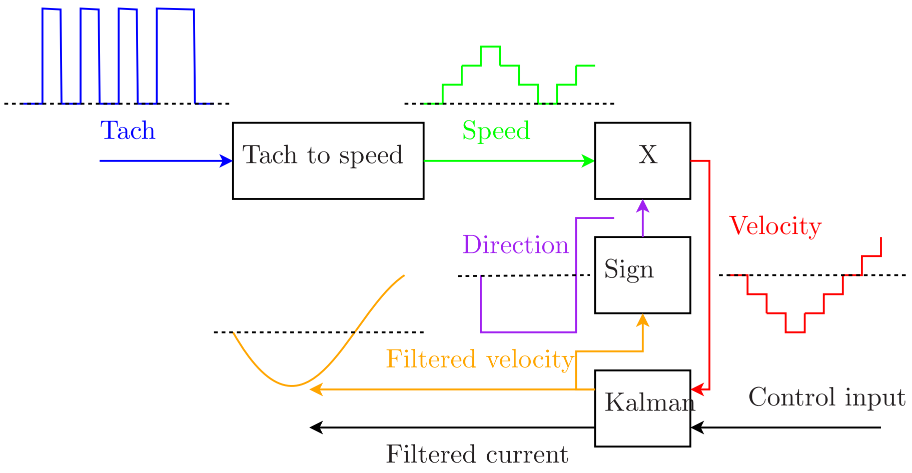


Figure 7.9: Sketch on how the tach signal (blue) is converted into speed (green), multiplied with the direction of the estimated velocity (purple), the resulting velocity (red) and finally the filtered velocity (orange) and current (black). The dotted line with every signal represents zero.

## 7.2 Controller Design

LQR Controller

The controller design chosen for this project is a LQR controller. This design method is chosen because the system is a multiple input, multiple output (MIMO) system and that it gives a phase margin of 60° [Levine, 1996]. This relatively large phase margin is useful when there could be some mismodelling in the model. When designing a LQR controller, it minimizes the cost function $J = \int_0^{\inf} (x^t Q x + u^t R u) dt$ where $Q$ and $R$ are design matrices. The design matrix $Q$ determinates the importance of the states. There are seven states in the state matrix $\left[ v_r, i_r, v_l, i_l, v_w, \dot{\theta}, \theta \right]^T$. Since it is control of wheelchair and not control of wheels, the $Q$ matrix becomes

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1000 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1000 \end{bmatrix} \tag{7.3}$$

The control on $v_w$ and $\theta$ is chosen because they are closely bounded with the other states in the wheelchair. They are also the inputs to the wheelchair, so its is natural to control these two states. There is a low weight on the rest of the states such that the controller can put these states to zero when there only are steady state inputs to the two main control states. To bound the dynamic of the controlled wheelchair such that it is usable by humans, the design matrix $R$ becomes

$$\begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix} \tag{7.4}$$

which joins to the input states $\begin{bmatrix} u_r \\ u_l \end{bmatrix}$. The weights in the $R$ matrix has to be the same because they each control the voltage to a single wheel. If they were different the wheelchair would behave different if it is turning left or right. By using Matlabs lqr function, a control matrix K is found. The control matrix K maps the seven states into two control signals. The values in the control matrix can be seen in equation 7.5. As stated in the weight matrix $Q$, the most weight is on the velocity and angle state of the wheelchair.

$$K = \begin{bmatrix} 0.7331 & 0.0014 & -0.0084 & -0.0000 & 12.7500 & 1.4244 & 12.5925 \\ -0.0084 & -0.0000 & 0.7331 & 0.0014 & 12.7500 & -1.4244 & -12.5925 \end{bmatrix} \tag{7.5}$$

The reference input to the controller is added as stated in [Franklin et al., 2002, page 525]. A matrix $\bar{N}$ is constructed based on the model on the control matrix. The $\bar{N}$ matrix is a reference gain matrix introduced to remove steady

Reference Gain

state offset on the output from the controller. The reference gain matrix can be seen in equation 7.6.

$$\bar{N} = \begin{bmatrix} 49.1712 & 56.0130 \\ 49.1712 & -56.0130 \end{bmatrix} \tag{7.6}$$

### 7.2.1 Controller Verification

The controller and the reference gain matrices are added as feedback and feed forward to the model respectively. To verify that the controller along with the Kalman filters and reference gain can control the wheelchair, five different scenarios of input signal are tested. These scenarios are:

- Driving forward with constant velocity and constant angle. Step input.
- Turning the wheelchair a quarter of a round without moving the wheelchair. Step input.
- First turn a quarter round one way and then half a round the other way. Step input.
- Turn a full round. Ramp input.
- Turn and drive forward. Step velocity input, ramp angle input.

The controller and reference gain matrices are verified using motion tracking equipment. The systems return the position and angle of the wheelchair. The output from the motion tracking equipment is compared with the integrated output from the wheelchair. If noting else is stated, the two measurements are equal.

The first test is driving forward with a constant velocity. In figure 7.10 the reference to the wheelchair and the differentiated position from the motion tracking equipment are compared. The measured velocity is noisy because the

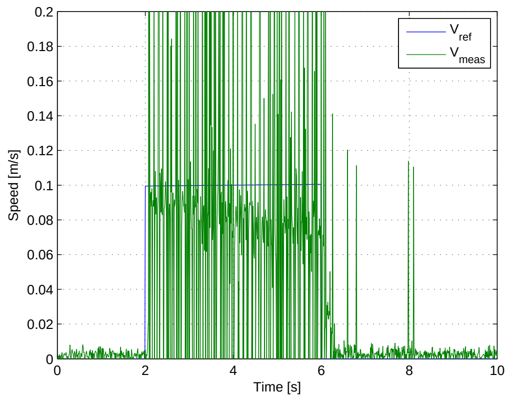


Figure 7.10: Comparison of reference input and measured output.

motion tracking equipment only returns the position of the wheelchair. In order to get the velocity of the wheelchair, it is necessary to differentiate the position, and small position errors at sampling speed results in a noisy velocity. At approximately 0.1 [m/s] a sort of baseline can be seen. This baseline fits the reference signal very well.

The second test is to test the steady state error on the angle. The wheelchair is orientated with angle 0, and the reference changes to $\pi/2$ at t=4 seconds. This can be seen in figure 7.11 As it can be seen, the wheelchair tracks the

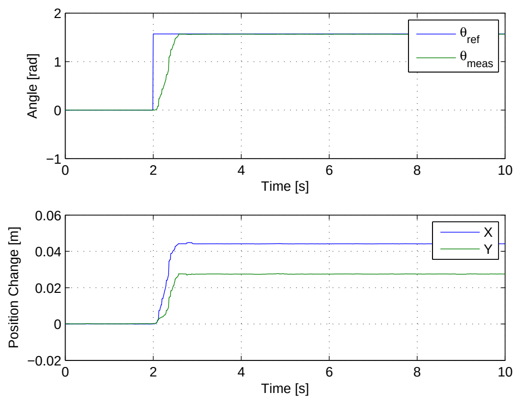


Figure 7.11: Steady state test of angle of the wheelchair.

angle well.

The third test is also a steady state angle test. The wheelchair is placed with angle = 0 and the reference input is $pi/2$. at t=4 seconds the reference is changed to $-pi/2$. The result of this can be seen in figure 7.12.

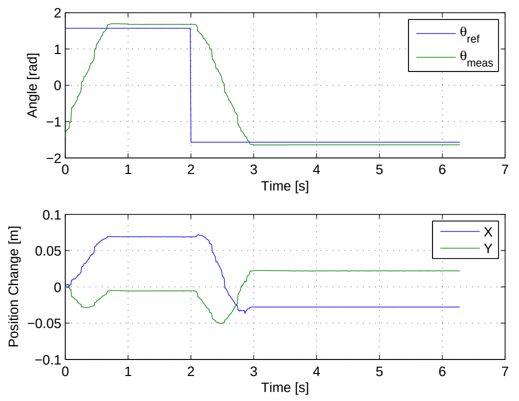


Figure 7.12: Another steady state angle test for the wheelchair. The steady state error is a bit larger here than in the other test.

The fourth test is to test how well the wheelchair tracks an increasing angle. The reference input is a ramp with an incline of 1 rad/s. This can be seen ion figure 7.13. At first the angle of the wheelchair is smaller than the reference

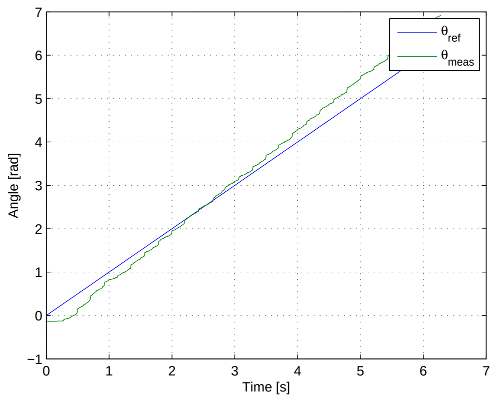


Figure 7.13: Test of how the wheelchair tracks and increasing angle.

angle, but at a point the angle of the wheelchair is greater than the reference angle. This is most likely to be caused by the reference gain matrix. The three other test already performed is step input with steady state check. The wheelchair has a great deal of slip of various reasons, and this slip is most pronounced at high accelerations as a step input is. Because the reference gain matrix is tuned to steady state performance, the reference input is too large with smooth input. The reference gain matrix should have been tuned to smooth input as this is what the wheelchair most likely will get, but any error in position or angle will be corrected by the measurements from the laser range scanner.

The last test is a combined test where both the velocity and the angle of the wheelchair is tested. The wheelchair gets a step input on velocity and a ramp input on angle. As it can be seen in figure 7.14, the wheelchair can keep a constant velocity and track an increasing angle at the same time.

## 7.3 Motion Control Results

This section describes how the motion control is tested and presents the results of the tests. There are some prerequisite that needs to be fulfilled before the motion control can be tested. The Wheelchair has located itself in the room, and the cognition has found a path from start to goal. The test for the motion control is then to move the wheelchair along the path.

The initial test drives indicates that the wheelchair has a great amount of wheel slip when turning, low grip tyres and heavy wires to the control computer. As it can be seen in the section above, the wheelchair is capable of calculating the right angle of itself. But whenever the wheelchair has made a turn, the angle

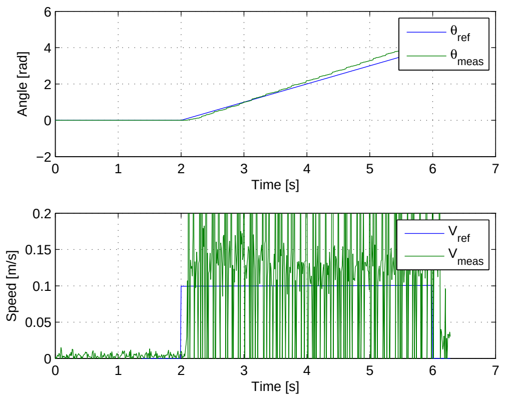


Figure 7.14: Combination of driving forward and turning.

of the wheelchair based on the motion tracking equipment were compared with the angle calculated by the wheelchair. If a difference in angles were present, the wheelchair was paused and turned to the right angle. There are three reasons for validating this. On a real wheelchair the slip between wheels and floor is assumed to be neglectable. That is that the angle that is calculated by the wheelchair is assumed to be correct. Second is that the localisation were disabled during the run. The localisation is computationally heavy and because the computer where it is implemented is the same as the path guidance is located, the wheelchair gets old reference signals when the localization is running. And third, in the current implementation of the motion control on the wheelchair, it is not possible to reset the states to anything except zero. That is, when the localisation has calculated a new angle for the wheelchair, it can not be loaded into the angle state on the wheelchair.

With the pause and reset of the angle of the wheelchair manually, figure 7.15 were made. The figure shows the position for the wheelchair when it is driving along a path. The reason that it not directly on the path is that the wheelchair is guided toward the next waypoint and not the path between the waypoints. this means that, with reservation for the wheel slips, the control is able to move the wheelchair along a predefined path. A longer path is followed in chapter 8, Acceptance test.

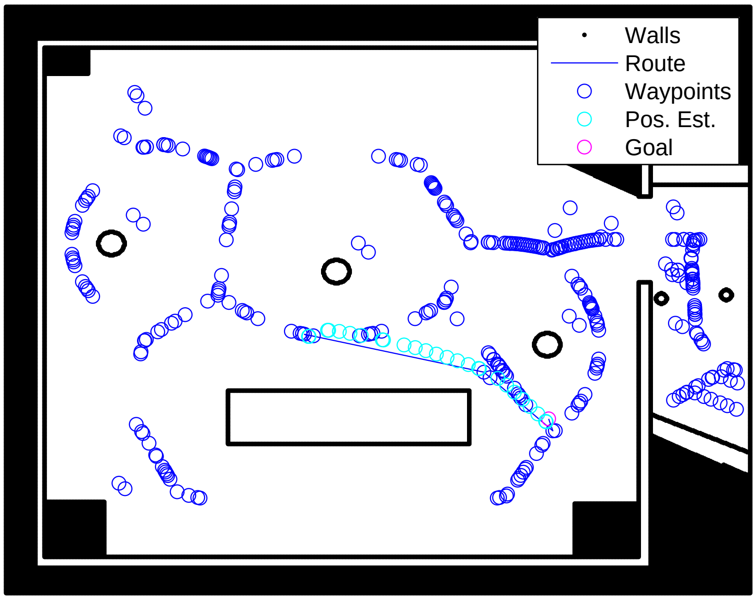


Figure 7.15: Path followed by the wheelchair with no relocalisation is done.

# Part III

# Results and Conclusion

# Chapter 8

# Acceptance Test

This chapter will present the result from the test of the autonomous wheelchair and compare them with the requirements from section 1.3 on page 13.

## 8.1 Localization Results

The developed localisation algorithm must be able to perform global localisation. This is tested first. Then, position tracking is tested, to ensure that this is also working. Finally, the algorithms ability to solve the hardest problem in localisation, the kidnapped robot problem, is tested.

The tests are performed by placing the AWC in the standard test environment, which is a room about 5 by 5 meters, with boxes in three corners and one, longer box in the center. Three round obstacles are present in the room, and one door is left open. This test environment can be seen on all figures in this section. The AWC then performs the different localisation related tasks, using the LRS data, odometry and the room. These are real tests, to ensure that the algorithm works with true measurements.

On the figures in this section, four things are plotted: the walls are plotted with black, all the particles are plotted with blue stars, the best unique estimate from the MCL algorithm is plotted with a green circle, and the true position is plotted with a red circle. Only coordinates are plotted, to keep the figures manageable.

### 8.1.1 Global localisation

As stated earlier, the AWC must be able to perform global localisation. The AMCL algorithm that is used in this project was specifically chosen because of its ability to do this. This test is performed by placing the AWC in the test environment, and determine its true position using a Motion Tracking system. The actual global localisation is performed by running the AMCL algorithm three times. The AWC is not moved between each iteration. On figure 8.1 the state of the localisation is shown after the first iteration. At this time, the particles are still spread out over the entire state space, but the best estimate (the particle with the highest weight) is already close to the true pose. On figure 8.2 the state of the localisation is shown after the second iteration. At this time, all particles are very close to the true pose, due to the sampling process. The best estimate is very close to the true pose as well.

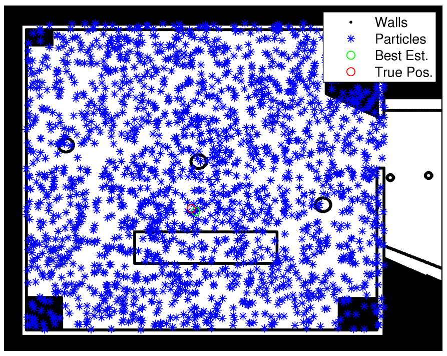


Figure 8.1: 1st iteration.

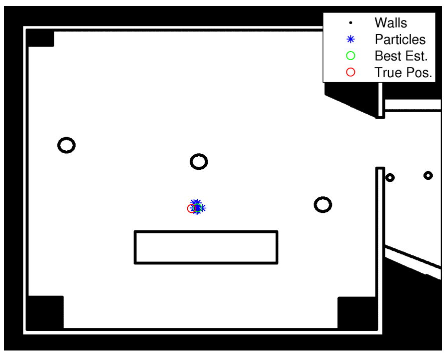


Figure 8.2: 2nd iteration.

On figure 8.3 the state of the localisation is shown after the third iteration. At this time, the particles are even closer to the true pose, again due to the sampling process. The best estimate is very close to the true pose as well.

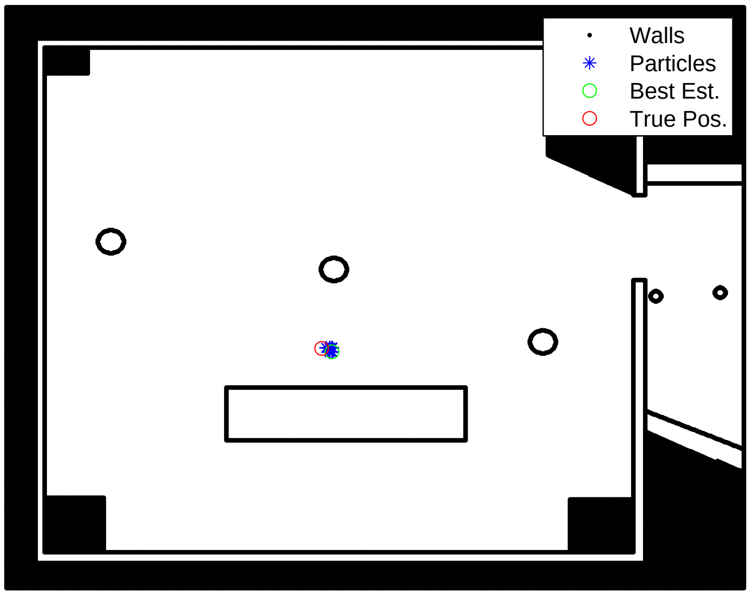


Figure 8.3: 3rd iteration.

The final best unique estimate is within 10 cm of the true position in both x and y directions, and within 2°of the true angle, as required. The true and estimated poses can be seen in table 8.1. Several tests of the global localisation where performed, all resulting in a successful localisation with similar difference in true and estimated pose.l localisation with similar difference in true and estimated pose.

Table 8.1: Real and estimated pose of global localisation.

| | x | y | $\theta$ |
|---|---|---|---|
| Real Pose | 316 | 285 | 0 |
| Est. pose | 320 | 289 | -1 |

Global Localisation Performed

It can be concluded that the AMCL algorithm works , and is capable of performing global localisation of the robots pose, even with a non-moving robot (at least when there are no symmetry problems).

In this test, the algorithm adapted the number of particles from 2100 in the first iteration to 200 (the specified minimum) in the third iteration. It took 32.8s to run the first iteration, and 2.5s to run the third.

Run Times

This test is an example. Several tests of the global localisation algorithm were performed, placing the AWC model in different poses in the test environment. The results of these tests were similar to the ones described here, and also within the required accuracy.

### 8.1.2 Position tracking

The AMCL algorithm has now been proved able to perform global localisation. However, the AWC will move around the house or apartment, and therefore it is also necessary to perform position tracking. This could be done by running a full global localisation each time a pose estimate is needed, but this is too time consuming. As described in section 5.1 on page 37, the method used in the AMCL algorithm is to pass all particles through a probabilistic motion model, and continue without resetting the algorithm.

This is tested by first letting the AWC perform a global localisation, and then moving it 10 cm each time step. This is done manually, and the 10 cm are also entered into the code manually, so any potential influence from the odometry is eliminated. On figure 8.4 on the next page the particles, best unique estimate and true position is seen before movement. On figure 8.6 the robot has performed two more time steps, each with a 10 cm movement, for a total 20 cm movement along the x axis. It is seen that the particles and pose estimate follows the true position. On figure 8.5 on the following page the robot has performed two more time steps, each with a 10 cm movement, for a total 40 cm movement along the x axis. It is seen that the particles and pose estimate follows the true position. On figure 8.5 on the next page the robot has performed three more time steps, each with a 10 cm movement, for a total 70 cm movement along the x axis. It is seen that the particles and pose estimate follows the true position.

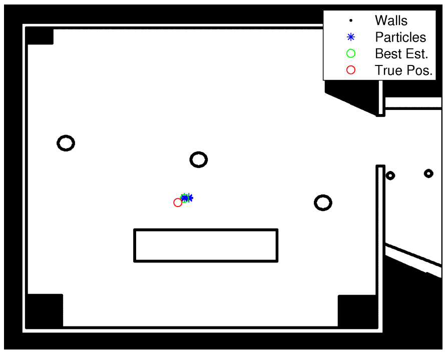


Figure 8.4: No movement.

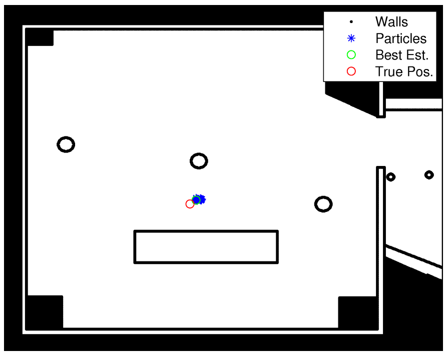


Figure 8.6: 20 cm movement.

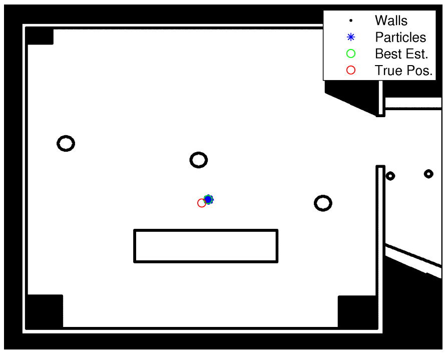


Figure 8.5: 40 cm movement.

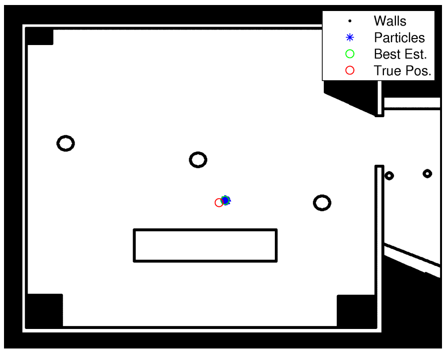


Figure 8.7: 70 cm movement.

POSITION TRACKING PERFORMED

From these figures it can be seen that the AMCL algorithm is capable of performing position tracking. During this test, the number of particles remained constant, at the minimum 200 particles.

### 8.1.3 Kidnapped robot

At this point, the AMCL algorithm has proved capable of performing both global localisation and position tracking, and has as such performed to the specifications and requirements of this project. A harder problem is to solve the kidnapped robot problem, where a robot is "kidnapped", after having successfully localised itself. After this kidnapping, the robot must relocalise itself.

In this project, the kidnapped robot problem is solved by augmenting the AMCL algorithm, and creating the AAMCL algorithm. The ability of the AAMCL algorithm to solve this problem is tested by first performing global localisation, and then moving the AWC to a new pose. None of these poses are of course known to the AWC.

On figure 8.8 on the facing page the AAMCL algorithm has performed global localisation, and has not been moved yet. There number of particles still spread out over the state space are the random particles added in the AAMCL algorithm. On figure 8.9 the AWC has been kidnapped to a new position, which is still indicated by the red circle. However, the best unique estimate is wrong. The AWCs angle is 0 degrees, and by carefully inspection of the figure, it can be seen that the (wrong) estimated actually "looks" a lot like the true position, from the AWCs point of view (that is, the LRS measured distances to the objects are quite similar). On figure 8.10 it is the same, but the particles are being resampled to the wrong pose. On figure 8.11 the AAMCL algorithm has managed to identify the correct pose. A total of four iterations has been performed since figure 8.8.

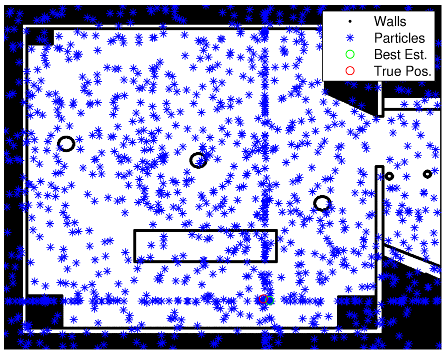


Figure 8.8: Not kidnapped.

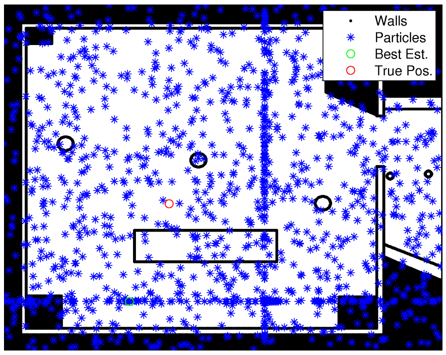


Figure 8.9: Kidnapped.

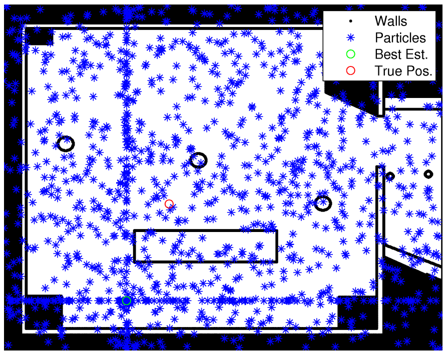


Figure 8.10: Wrong Estimate.

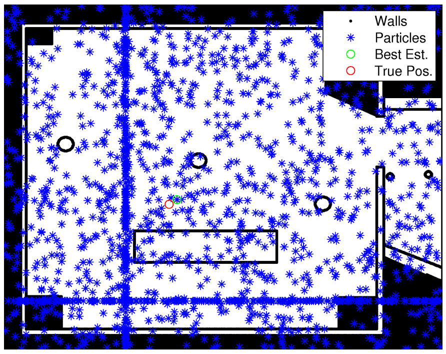


Figure 8.11: New pose acquired.

On figure 8.12 on the next page it is the same, but the particles are being resampled to the new, correct pose.

The reason that the figures showing the kidnapped robot are different from the figures in the other localization problems, is that it requires additional random particles spread in the entire state space in order to solve the kidnapped robot problem. When the MCL algorithm is set to also solve the kidnapped robot problem, the computation time raises to approximately 6 s. pr iteration. This is a result of injecting more particles to compute, in this case 1000 random particles are injected. Also, on the figures 8.8 to 8.12 on the next page a

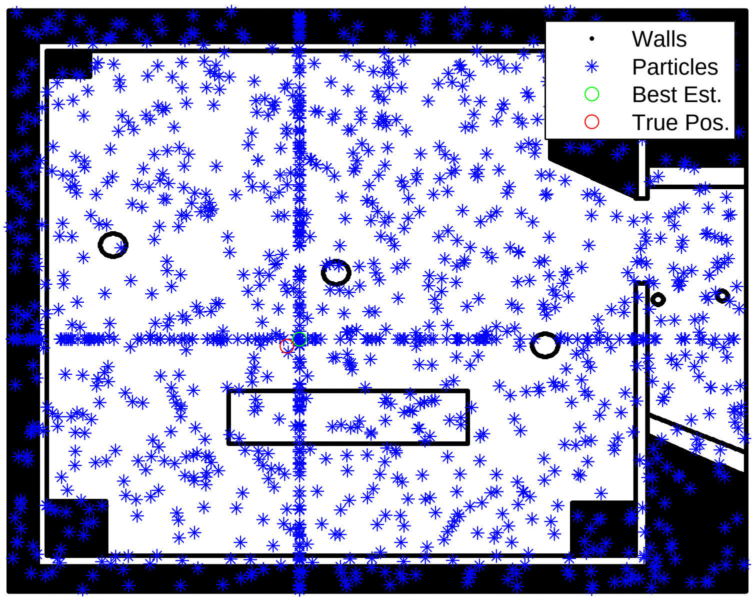


Figure 8.12: Particles resampled to new pose.

"cross" of particles can be seen. This is a result of the sampling used. The particles are being sampled individually in the x, y and $\theta$ directions, as discussed in appendix D on page 117. Therefore, the particles that match in one direction are more likely to be propagated than those that does not match in any direction. Only x and y are plotted in these figures, but the particles shown are likely to have an angle that matches the true angle.

Kidnapped Robot Problem Solved

It can be concluded by this test, that the developed AAMCL algorithm is capable of solving the kidnapped robot problem.

### 8.1.4 Comments

From the first two tests in this section, it is concluded that the developed localisation algorithm lives up to the requirements. It is capable of performing global localisation to within the specified limits, and of performing position tracking.

The developed AAMCL algorithm is also capable of solving the kidnapped robot problem, as shown by the third test in this section. This will hopefully not be necessary on the AWC, but it does show that the AAMCL algorithm is robust against otherwise catastrophic localisation failures, as this is essentially the same problem as kidnapped robots.

## 8.2 Cognition Results

The requirement for the cognition was that it should be able to make a path from its initial pose to any pose that the manual wheelchair can drive. This is hard to prove, but what can be proved is that the cognition can make a path from its initial position to the goal position. A test of this can be seen in figure 8.13.

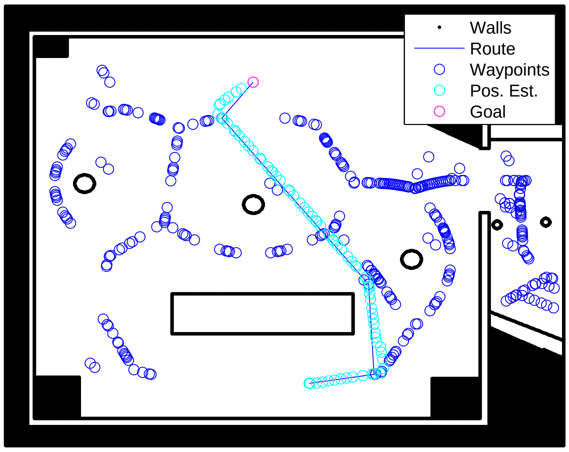


Figure 8.13: Route calculated by cognition and path followed by wheelchair.

As it can be seen in the figure, the route that the cognition calculated is round corners and obstacles such that the wheelchair will not collide with either of them. Also shown in figure 8.13 are the path that the wheelchair followed afterward which leads to the next result section.

## 8.3 Motion Control Results

The wheelchair is capable of following the route that the cognition had calculated as it can be seen in figure 8.13. There is however a catch with this figure. Because there is slip at the wheels of the wheelchair, any time the wheelchair is rotating, the slip is significant and the odometry data does not fit the real world. Because it is likely that a real wheelchair does not have any wheel slip, the model wheelchair is stopped some times and manually moving it to the pose that the odometry has measured. The manual correction is typically about 15°that the wheelchair has not rotated. What is shown in the figure are the position that the odometry of the wheelchair provided.

# Chapter 9

# Conclusion

This chapter will discuss the results from the previous chapter and conclude on the project.

## 9.1 Results Discussion

### 9.1.1 Perception

For the perception phase, two sensors were selected. For odometry data, a pulse encoder is fitted to each wheel. This provides 50 pulses per wheel revolution. This is sufficient for integrating into a position change, which can be used in the localisation phase, but not to give a sufficient good approximation to the velocity. Therefore, a Kalman filter is implemented. This gives a smooth and accurate velocity, which can be used in the controller. The final odometry data is accepted for use in the later phases of navigation.

For acquiring information about the environment, a laser range scanner is selected. This provides very accurate distances to objects, with high angular resolution. Another type of distance sensor, the infrared range scanner, was investigated, but was found to have too low resolution, both angular and in distance. It was considered to use a fixed pointer laser range finder, but this would not allow the AWC to see all obstacles, and thus it would not perform to requirements. The chosen LRS solution is very acceptable. It does have problems with black wooden walls, which could be a problem in some cases.

### 9.1.2 Localisation

As it can be seen from the acceptance test in chapter 8, the wheelchair is capable of localizing itself in its environment. The precision of the localization depends on the accuracy of the map and the calibration of the equipment to verify the pose of the wheelchair. Any offset between the true pose and the estimated pose is less than what is possible to detect using visual inspection. The localization is at all times better than $\pm$ 10 cm in x and y direction and less than $\pm$ 2°in the orientation. This is an acceptable result and will not affect the performance of the wheelchair because the area that the wheelchair must be within at waypoints and goals are set to $\pm$ 20 cm in x and y direction. If the odometry of the wheelchair looses track of where it is, that is, the kidnapped robot problem, the wheelchair can within a few iterations relocalise itself and thus solve the kidnapped robot problem. This also provides a level of robustness, because the wheelchair then can relocalise itself if it localises itself to a wrong pose.

The chosen AAMCL algorithm thus performs as required, and even better, as the kidnapped robot problem is also solved. Other algorithms can be used in localisations of mobile robots, such as the basic grid based Markov filter, and the various forms of the Kalman filter. The Kalman filter and its variants can perform position tracking, but not global localisation, and cannot solve the kidnapped robot problem, and were as such not applicable to this project. The basic Markov filter can solve the same problems as the MCL algorithm, but is more computationally demanding. This leads to the one big problem with the MCL algorithm: it is very computationally demanding. Even with the extension to AMCL, it takes 2.5 s to run each iteration during position tracking. This is usable if the AMCL algorithm could have its own CPU core, and just give the wheelchair an updated position when it was done calculating.

The augmentation of algorithm into the AAMCL algorithm provides extra robustness, at the expense of time for each iteration. Whether this compromise is necessary will depend on the actual use of the wheelchair.

In this project, it was decided that no artificial landmarks were allowed. This makes it much cheaper and easier for the wheelchair supplier to setup a wheelchair for the end user. The only necessary task is to load a new bitmap file into the software. It is necessary that this map is accurate, as the accuracy of the localisation cannot be better than the map. Note that this map would have to be constructed anyway, for this particular application of the AWC, as the end user must be presented with it to select a target pose.

### 9.1.3 Cognition

Based on the map, the wheelchair is capable of calculating a configuration space when the configuration of the wheelchair is set to a circle. Based on the configuration space, the wheelchair is capable of calculating a generalized Voronoi diagram and extract waypoints from the generalized Voronoi diagram in order to make a road map. Based on the waypoints in the road map, the wheelchair is capable of calculating a path from its current position to the goal that is defined by the user of the wheelchair. Obstacle avoidance is currently not working in the implemented application, mostly caused by low sampling time. It works in the simulation.

The configuration space ensures that the wheelchair does not collide with the walls or furniture of the house. This was a requirement. Using the generalised Voronoi diagram for waypoints keeps these waypoints as far away as possible from the walls and furniture of the house. This makes the trip much for comfortable for the user, as close encounters with walls or furniture would probably make the user feel unsafe. However, if the wheelchair follow the paths directly from the Voronoi diagram, the trip would be much longer than necessary. A compromise was made, where the path planner considers any line-of-sight between the waypoints as a valid route. This still maintains a minimum safe distance to the walls, due to the configuration space, and maintains a large distance to the walls most of the time, while still not driving an excessively long path. This is considered a good result. Other algorithms could have been used, such as a visibility graph for path planning, but the chosen solution provides a good compromise between a comfortable distance to objects and a short path.

By constructing a roadmap, it is ensured that there is a path to drive. This is a requirement in this project. Other techniques could have been used, such as Vector Field Navigation (VFN) as a stand alone path planner, but this technique might guide the wheelchair into blind allays, known as local minima. Therefore this was not applicable in this project. However, in this project VFN has been used as a local path planner, guiding the AWC around both mapped and unmapped obstacles, while driving toward the next waypoint. This way local minimas are avoided.

The VFN algorithm does not work in the implemented application, however the concept has been proved in simulation, and with faster sampling time it should be able to perform local path planning of the AWC.

### 9.1.4 Motion Control

The controller to move the wheelchair is an optimal model based controller, where the model of the wheelchair is a linear model. The controller is able to receive reference signals from the wheelchair cognition and make the wheelchair move accordingly.

The developed optimal controller is easy to move to a real wheelchair. Some parameters would have to be reestimated, and a new LQR controller calculated. This is easy in contrast to many other controllers, such as PID and fuzzy logic controllers, which would have to be retuned when the model was exchanged for a real wheelchair.

## 9.2 Conclusion

This project has been a proof of concept, where it has been shown that it is possible to make an autonomous wheelchair, capable of transporting a user with Amyotrophic Lateral Sclerosis, or any other disabled person, with no user input except where to go. The wheelchair does not need to be told where it is, or how to get to the goal. It does not even need any artificial landmarks for navigation. Using only onboard sensors and a map of the house, which was available anyway, it performs the required tasks.

After implementing perception, localisation, cognition and motion control on the wheelchair, the wheelchair is an autonomous wheelchair capable of navigating around a house or apartment. It is capable of sensing the environment, localising itself in this environment, make a road map, make a path from its current position to the road map, along the road and of the road map and make the wheelchair move along the chosen path. This has been accomplished using several state of the art techniques and algorithms, such as the Kalman filter for velocity estimation, the augmented adaptive Monte Carlo localisation algorithm for localisation, the generalised Voronoi diagram in the configuration space for roadmaps and waypoints, Dijkstras algorithm for path planning between these, vector field navigation for local path planning and an optimal controller for motion control.

The different algorithms all sum up to a complete autonomous wheelchair, that is capable of performing the required tasks (with the slight exception of the VFN, which has problems due to the sampling frequency). As discussed in section 9.1 on page 91, the different choices of algorithms have been good.

The goal of this project was to create a system for Autonomous Wheelchair Navigation. This has been successful, as the system is nearly ready for implementation on a real wheelchair. This means that this project is ready to become a part of the larger project with the laboratory for Brain-Computer Interfaces at the Center for Sensory-Motor Interaction. The final goal of this large project is to create an entire system for communication, navigation, etc, for people with Amyotrophic Lateral Sclerosis, who can only control their eyes. The communication between the user and the system will at first be an eye tracker, and later hopefully a Brain Computer Interface, [Cabrera & Dremstrup, 2008]. The potential of this project to help these people is huge. Suddenly they would be able to both communicate with friends and family, and they would be able to get around. This will make a immense difference in their life. But even without the extra functionalities being developed there, the goal of this Autonomous Wheelchair Navigation project has the potential to be a tremendous help in these disabled peoples daily life. The fact that they would be able to get around in their home in a fast, safe and easy way, without assistance, must be a great boost in life quality.

# Chapter 10

# Future Work

This chapter describes what work is needed on the project before it can be used with the project from the laboratory for Brain Computer Interface project at the Center for Sensory-Motor Interaction [Cabrera & Dremstrup, 2008]. The chapter is divided into three sections: *Already done*, A small summary of what is done in the project, *Adjustment*, what needs to be adjusted on the project, and *limited out*, what was out of the scope of this particular project but is needed for the entire system to work.

## 10.1 Already Done

As stated in the conclusion, the wheelchair is capable of localising it self in its environment, get user input, plan a path and generate control signals to move the wheelchair along the chosen path.

## 10.2 Adjustments

Although the wheelchair is capable of autonomous movement, it still needs a bit of help when moving around. There are two issues in this section. The first issue is that during obstacle avoidance, the repulsive vector fields are making the wheelchair unstable or limited stable. This will be solved when the baud rate of the laser range scanner is increased since most of the sampling time for the repulsive vector field is used to receive data from the laser range scanner. If this is combined with a preemptive scheduler or multicore processor such that the computation of localisation and attractive vector fields can be done while the computer is receiving the data from the laser range scanner, the sample time will decrease and thus calculate new repulsive vectors fast enough for a stable system. The second issue here is that the wheels of the wheelchair slips when it is accelerating. The wheel slip is due to low weight of the wheelchair and low grip tyres. Furthermore, all the control signals and sensor measurements from the wheelchair is done using two relatively heavy and stiff cables that is also increasing the slip of the wheels. Therefore the odometry data is not equal to the real pose of the wheelchair after an acceleration. The solution to this is out of the limits of this project and can therefore be seen in the next section.

## 10.3 Limited Out Of Project

As stated in the previous section, the solution to some of the problems occurring in this project is limited out of the project. These issues and the remaining work on the project will be described in this section.

The issue of wheel slip will be solved by migrating to a real wheelchair with onboard computers and batteries. Since the wheel slip of a real wheelchair is neglectable the odometry data will fit the pose of the wheelchair much better. When the odometry data is closer to the real pose of the wheelchair, the particle filter used to localisate the wheelchair will need many fewer particles and will therefore compute the localisation of the wheelchair faster. At the current moment the particle filter is executing at the same sample rate as the vector fields. By moving the particle filter to a separate processor or even computer, the vector fields does not have to wait until the particle filter has computed the localisation of the wheelchair. By doing so, the sample time of the vector fields will become lower and more stable and thus is the vector field guiding the wheelchair around obstacles.

All the software has to be migrated into the same system as the rest of the wheelchair project from Center for Sensory-Motor Interaction is using. When this is done, the user input will come from the eye tracker such that the wheelchair can be used by people with ALS.

# Part IV

# Appendix

# Appendix A

# Wheelchair Modelling

## A.1 Wheelchair Kinematics

The kinematic of the wheelchair describes how the different frames in the wheelchair control is defined and connected. The wheelchair consists of two interconnected wheels at the front and two caster wheels at the back which introduces no constraints on the mobility of the wheelchair [Angeles, 2002, page 417]. The wheelchair has thus 2 degrees of freedom and is operating in $\mathbb{R}^2 \times [0; 2\pi]$. A sketch of this can be seen in figure A.4 on page 102. What is wanted from the kinematics is a relationship between the velocity of the wheelchair at the wheels, $v_r$ and $v_l$ and the velocity in $x$ and $y$ coordinates. In this section, the velocity of the wheelchair at e.g. the right wheel is written as the velocity of the right wheel. If e.g. the right wheel is not rotating, and the left wheel has a constant velocity $v_l$, then the point located halfway between the wheels must have velocity $v_l/2$. If the left wheel has velocity $v_l$ and the right wheel has velocity $v_r = -v_l$, the point in the middle must have velocity 0. Then the equation for the velocity of the wheelchair must be

$$v_w = \frac{v_l + v_r}{2} \tag{A.1}$$

The angular velocity of the $POI$ in figure A.4 on page 102 also depend on the wheelchairs velocity at the wheels. The normal conversion between angular velocity and tangential velocity is

$$\dot{\theta} = \frac{v}{r}$$

where $\dot{\theta}$ is the angular velocity and $r$ is the distance between the rotation point and the point where the tangential velocity is measured. This can be seen in figure A.1.

If the left wheel is not rotating, and the right wheel is rotating with velocity $v_r$, the angular velocity is

$$\dot{\theta} = \frac{v_r}{r}$$

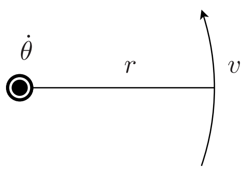


Figure A.1: Relation between $\dot{\theta}$,$r$ and $v$.

If the left wheel is rotating with velocity $v_l$, and the right wheel does not rotate, the resulting angular velocity is

$$\dot{\theta} = -\frac{v_r}{r}$$

If the right wheel is rotating with velocity $v_r$ and the left wheel is rotating with velocity $v_l = -v_r$, the resulting angular velocity is

$$\dot{\theta} = \frac{v_r - (-v_l)}{r}$$

This is also true if both wheels has velocity $v_r = v_l$, i.e. the wheelchair is driving in a straight line and is not rotating, since the angular velocity becomes

$$\begin{aligned}\dot{\theta} &= \frac{v_r - v_l}{r}\\ \dot{\theta} &= 0\end{aligned}$$

Because both the velocity and the angular velocity of the wheelchair is linear combinations of the velocity of the wheels, they can be fitted into a single transformation matrix from wheel velocity to resulting velocity and angular velocity as in equation A.2

$$\begin{bmatrix} v_w \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{r} & -\frac{1}{r} \end{bmatrix} \begin{bmatrix} v_r \\ v_l \end{bmatrix} \tag{A.2}$$

There is however a problem using equation A.2 to calculate the current angle. The velocity used as input comes from an estimating Kalman filter designed in section 7.1.1 on page 68. Because the filter is estimating the velocity, the angle will also be an estimation. E.g. if the wheelchair is standing still, but there is a small input for one of the wheels. Then the Kalman filter will estimate that that wheel is rotating slowly for a while. This results in the angle of the wheelchair changing, though it is not. The transformation is suited for estimating the rate of rotation, but it is not suitable for integration. However, the measurements directly from the tact sensor that is not suited for velocity measurements, can be used as position measurement. The integration of velocity can be seen in figure A.2. Therefore, if the current orientation or position is needed, then the data must come directly from the sensor without using the Kalman filter.

## A.2 Wheelchair Dynamics

The wheelchair chosen for this project is a wheelchair with powered front wheels and casters at the back. This is chosen for its manoeuvring abilities

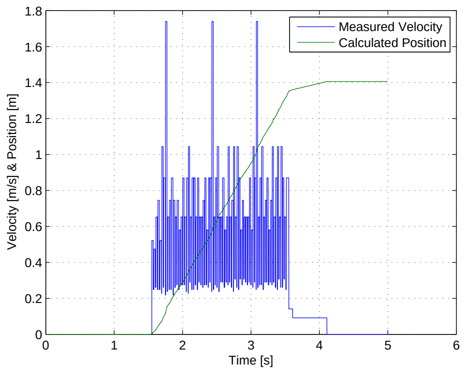


Figure A.2: Integration of the velocity gives a smooth position.

although it is unstable at high velocities [de Vries et al., 1999]. A sketch of such a wheelchair can be seen in figure A.3

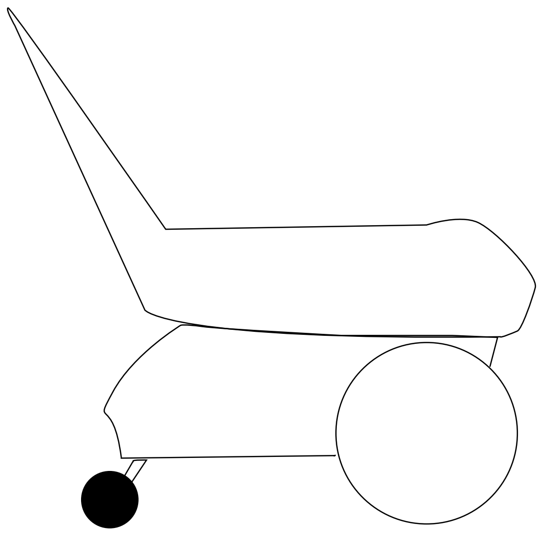


Figure A.3: Sketch of a wheelchair with powered front wheels.

### A.2.1 Wheelchair Model

The wheelchair consists of two driving wheels at the front and two caster wheels at the back. Propulsion and manoeuvring is done solely by the front wheels. The center of gravity is assumed directly on the middle between the front wheels and the caster wheels. There are two frames in the wheelchair as seen in figure A.4. The inertial frame is fixed and refers to the house/room in which

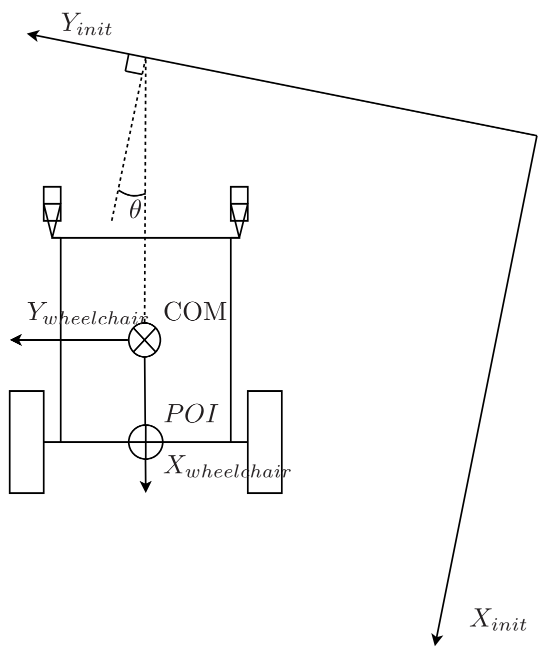


Figure A.4: Free body diagram of a wheelchair.

the wheelchair is maneuvering. The wheelchair frame has its origin in the wheelchairs center of mass, $COM$, and the x axis is on the perpendicular to the axle between the front wheels. The angle $\theta$ describes the angle between the two frames.

There exist several papers on modelling the dynamics of the wheelchair, including [de Vries et al., 1999], [Ding et al., 2004] and [Johnson & Aylor, 1985]. Common for all papers is that they come up with a complex, nonlinear model where the input is a force rather than a voltage. In this project a more simple model is investigated where the wheelchair is voltage controlled. The dynamics of the model does not cope with wheel slip, and changes in e.g. moment of inertia due to rotation of the wheelchair. Instead it treats the wheelchair as two independent systems, each consisting of a mass and an electric motor, where the input to the system is the voltage to the motor, and the output the velocity of the wheels. A schematic of the electric circuit of the motor and a force diagram of the wheelchair can be seen in figure A.5 on the facing page.

The constant $K$ is the motor constant and describes the relation between the electrical and the mechanical part of the motor. There are missing some symbols on the diagram such as moment of inertia of the wheel, but since both the mass of the wheelchair and the moment of inertia is affected by the torque/force of the motor, it affects the dynamics in the same way. The frictional force represented by $b_1$ and $b_2$ can be summed to a single constant $b$. Stiction forces and other nonlinear forces is not modeled. Most of these forces only affects the wheelchair when it is starting and stopping, and is assumed to be smaller than the requirement for position of the wheelchair. The last assumption is that the

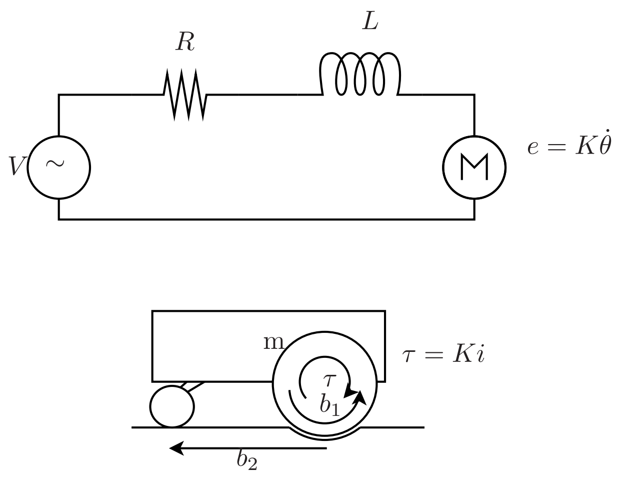


Figure A.5: Electric motor circuit and wheelchair forces.

center of mass is placed close to the front wheels or that the rolling friction of the rear casters is relatively large. Else the wheelchair would become unstable at high speeds if the controller has a large gain [de Vries et al., 1999].

The forces and voltages in figure A.5 can be combined to two equations.

$$L\dot{i} + Ri + K\dot{\theta} - V = 0 \tag{A.3}$$
$$J\ddot{\theta} + b\dot{\theta} - Ki = 0 \tag{A.4}$$

where

$L$ is the inductance of the motor $[H]$.

$R$ is the resistance of the motor $[\Omega]$.

$K$ is the motor constant $[Nm/A]$ or $[rad/s/v]$.

$J$ is the moment of inertia of half the wheelchair $[kgm^2/s^2]$.

$b$ is the damping ratio of the wheelchair $[Nms]$.

$v$ is the voltage supplied to the motor.

$i$ is the current drawn by the motor.

$\dot{\theta}$ is the angular velocity of the motor, converted 1:1 to the velocity of the wheelchair at the wheel $[rad/s]$ or $[m/s]$.

By simply rearranging the terms in equation A.3 and A.4, it is possible to get a state space representation of a model for the wheelchair.

$$\begin{bmatrix} \ddot{\theta} \\ \dot{i} \end{bmatrix} = \begin{bmatrix} -\frac{b}{J} & \frac{K}{J} \\ -\frac{K}{L} & -\frac{R}{L} \end{bmatrix} \begin{bmatrix} \dot{\theta} \\ i \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{L} \end{bmatrix} V \tag{A.5}$$

This model is however only valid when the wheelchair is driving in a straight line. If the wheelchair is rotating, the inertia from the $COM$ is not just acting in the $X_w$ axis. A sketch of a rotating wheelchair can be seen in figure A.6. This means that the moment of inertia is not only depending on the mass of

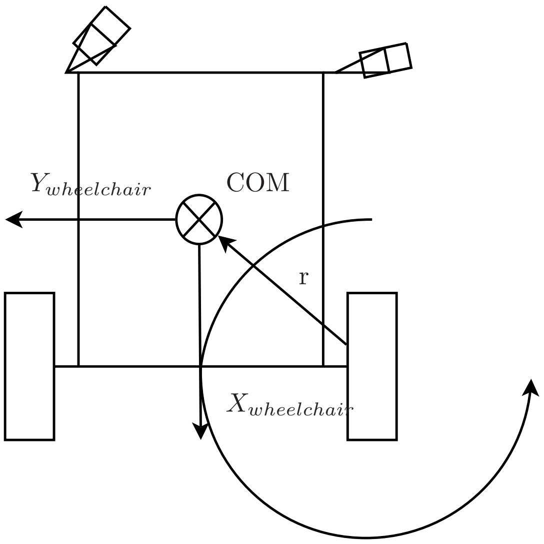


Figure A.6: Wheelchair rotating round the left front wheel.

the wheelchair, but also on the distance from the rotation point to the $COM$. If all mass of the wheelchair is located in $COM$, the moment of inertia is then

$$J = mr^2$$

where $r$ is the distance from the rotation point to $COM$ as seen in figure A.6. This means that a maximum and a minimum value for the moment of inertia and probably also for the friction $b$. Therefore a controller which is capable of including variations in the model in the design, is well suited for this sort of control.

The wheelchair consists of two such systems, connected as described in section A.1 on page 99. The next step is to estimate the values for the parameters. In appendix B the test procedure and results can be seen. The conclusion on this model is that is not perfect under curtain conditions such as fast rotations. Because this is a state that is unwanted for the user of the wheelchair, the controller for the wheelchair must be designed to avoid such states. Therefore the model is valid for this application.

## A.3 Combining Kinematics and Dynamics

In this section is described how the dynamics and kinematics of the wheelchair is combined in order to construct a controller for the entire wheelchair. The states chosen for the model, is the states for the motors and the states necessary to convert the states from the motor to the states of the wheelchair. The states are thus $\left[v_r, i_r, v_l, i_l, v_w, \dot{\theta}, \theta\right]^T$ which are velocity right wheel, current right motor, velocity left wheel, current left wheel, velocity of the wheelchair, change in angle of the wheelchair and angle of the wheelchair. The state matrix for the wheelchair can be seen in table A.6

$$\begin{aligned}
\dot{x} &= Ax + Bu \\
&= \begin{bmatrix} -b_1/J_1 & K_1/J_1 & 0 & 0 & 0 & 0 & 0 \\ -K_1/L_1 & -R_1/L_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -b_2/J_2 & K_2/J_2 & 0 & 0 & 0 \\ 0 & 0 & -K_2/L_2 & -R_2/L_2 & 0 & 0 & 0 \\ r/2 & 0 & r/2 & 0 & 0 & 0 & 0 \\ r/2l & 0 & -r/2l & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} x + \begin{bmatrix} 0 & 0 \\ 1/L_1 & 0 \\ 0 & 0 \\ 0 & 1/L_2 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} u \\
y &= Cx \\
&= \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} x
\end{aligned} \tag{A.6}$$

The different colors used in the matrices represent the different dynamics and kinematics from which the model is constructed. Blue and green are the model for right and left wheel represently, red is kinematics for forward velocity, purple are kinematics for angular velocity of the wheelchair and orange is the integrated angular velocity of the wheelchair, the angle of the wheelchair. The model is verified in appendix B. The model is used for design of controller for the wheelchair. The wheelchair is controllable because its controllability matrix $\mathcal{C}$ has rank = 7 as it can be seen in equation A.7.

$$\begin{aligned}
\mathcal{C} &= \left[B\ AB\ A^2B\ A^3B\ A^4B\ A^5B\ A^6B\right] \\
rank\,(\mathcal{C}) &= 7
\end{aligned} \tag{A.7}$$

The poles of the system can be seen in equation A.8.

$$[0, 0, 0, -2500, -4, -4, -2500]^T \tag{A.8}$$

The system is limited stable because of the poles in zero. If the system is mismodelled it is possible that the wheelchair could become unstable if the controller for the wheelchair is not designed properly.

# Appendix B

# Parameter Estimation for Wheelchair Model

The purpose of this chapter is to estimate values for the parameters used in the model for the wheelchair.

## B.1 Motor Parameter Estimation

All of the parameters in the model are physical parameters, but some of them, like the moment of inertia depends on more than just one component. The dynamic model of the wheelchair can be seen in equation B.1.

$$\begin{bmatrix} \ddot{\theta} \\ \dot{i} \end{bmatrix} = \begin{bmatrix} -\frac{b}{J} & \frac{K}{J} \\ -\frac{K}{L} & -\frac{R}{L} \end{bmatrix} \begin{bmatrix} \dot{\theta} \\ i \end{bmatrix} + \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix} V \tag{B.1}$$

Most parameters for the motors come from [Hurbain, 2007]. Here the series resistance is calculated to 25 Ω and the inductance to 10 mH. The motor constant can be approximated by using the equation for the electric part of the motor.

$$L\dot{i} + Ri = V - K\dot{\theta}$$

If $i$ is constant, i.e. steady state, $L\dot{i}$ is 0 and then Hurbain [2007] has measured voltage, current and angular velocity to be used in equation B.2.

$$\frac{V - Ri}{\dot{\theta}} = K \tag{B.2}$$

The mean of the four steady state points results in a $K$ value of 0.24 V/rad.

The last two parameters is moment of inertia, and friction. The moment of inertia consists of the moment of inertia of the wheel, the rotation of the wheelchair and the mass of the wheelchair. This parameter along with friction is due to be estimated based on measurements from the lab.

## B.2 Wheelchair Parameter Estimation

The model of the wheelchair is a simplified model which does not account for forces from casters and rotation. Therefore it is essential that the parameter estimation is also done in regions where these forces are present. There are three different scenarios to be tested.

- Both wheels is driving in the same direction with the same signal. This scenario is the one that is modeled.
- Only a single wheel is driving and the other wheel is braked. This scenario involves rotational moments that is not modeled.
- The wheels are driving opposite of each other, thus the wheelchair is rotating about its own axis. Here are also non modeled rotational moments.

### B.2.1 Setup

The setup for obtaining measurements for parameter estimation is as follow:

- Pc running Matlab xPC with appropriate hardware.
- Simulink program to create PWM signals for the motors
- Motion tracking equipment.

The simulink program can be found on [CDROM, 2008]. In the following sections, a number of figures with measured and simulated responses will be shown. There have prior to the creation of the figures been a parameter estimation by hand to fit the model to the measurements.

### B.2.2 Test 1

Test 1 is driving both wheels in the same direction and at the same velocity. There are two tests in this test. The first is at full speed and the second is at a increasing speed. In figure B.1 the measured and simulated velocity of the wheelchair can be seen. There is almost no difference in final velocity and the dynamics of the simulation is also close to the measured dynamic. The breaking dynamics is not equal to the acceleration dynamics, but since the acceleration dynamic is very close to the measured dynamics, it is considered within the acceptable range. The second test with both wheels in same direction and at same speed can be seen in figure B.2. Again it is possible to see that the breaking dynamics are not the same on the wheelchair as in the model. There is also a difference in amplitude at high speeds. This is because the model does not model the maximal velocity of the motor. It is decided that the parameters are usable for designing kalman filters and controller for the model. The values for the different parameters can be seen in equation B.3.

$$\begin{bmatrix} \ddot{\theta} \\ \dot{i} \end{bmatrix} = \begin{bmatrix} -\frac{120}{0.03} & \frac{K}{0.03} \\ -\frac{0.144}{0.01} & -\frac{25}{0.01} \end{bmatrix} \begin{bmatrix} \dot{\theta} \\ i \end{bmatrix} + \begin{bmatrix} \frac{1}{0.01} \\ 0 \end{bmatrix} V \tag{B.3}$$

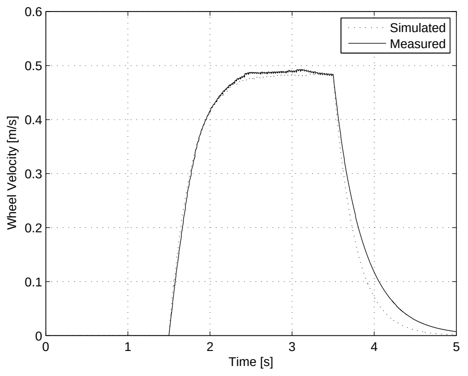


Figure B.1: Measured and simulated velocity of the wheelchair with both wheels at same velocity. The input voltage to the motors is constant.

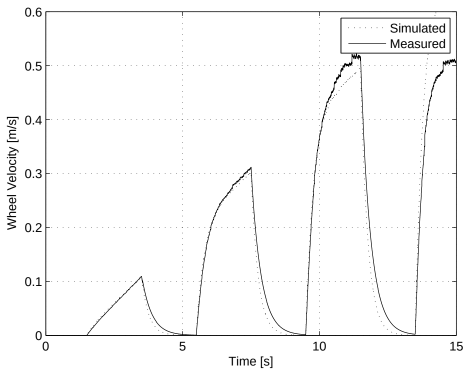


Figure B.2: Measured and simulated velocity of the wheelchair with both wheels at same velocity. The input voltage to the motors is a ramp multiplied with a pulse signal.

Appendix C

# PCB For the Wheelchair

In order to control the wheelchair, it is necessary to send signals to the motors and measure tach signals from the wheels of the wheelchair. To do this, three pieces of printed circuit board is designed and manufactured. The most important pcbs are the two identically pcbs, one for each wheel. They consists of a H-bridge amplifier, a measurement resistor and two differential amplifiers to measure the voltage drop across the measurement resistor. The schematic for such a circuit can be seen in figure C.1 A picture of the circuit on the wheelchair model can be seen in figure C.2.

There are also circuit for the tach sensors. All signals are send through twisted pair cables, so all signals except tach signals are send differentially. In figure C.3 the interface between the wheelchair and the sampling card is shown. It splits PWM signals into differential signals and convert the differential current measurement to single ended signal. A picture of the interface circuit can be seen in figure C.4.

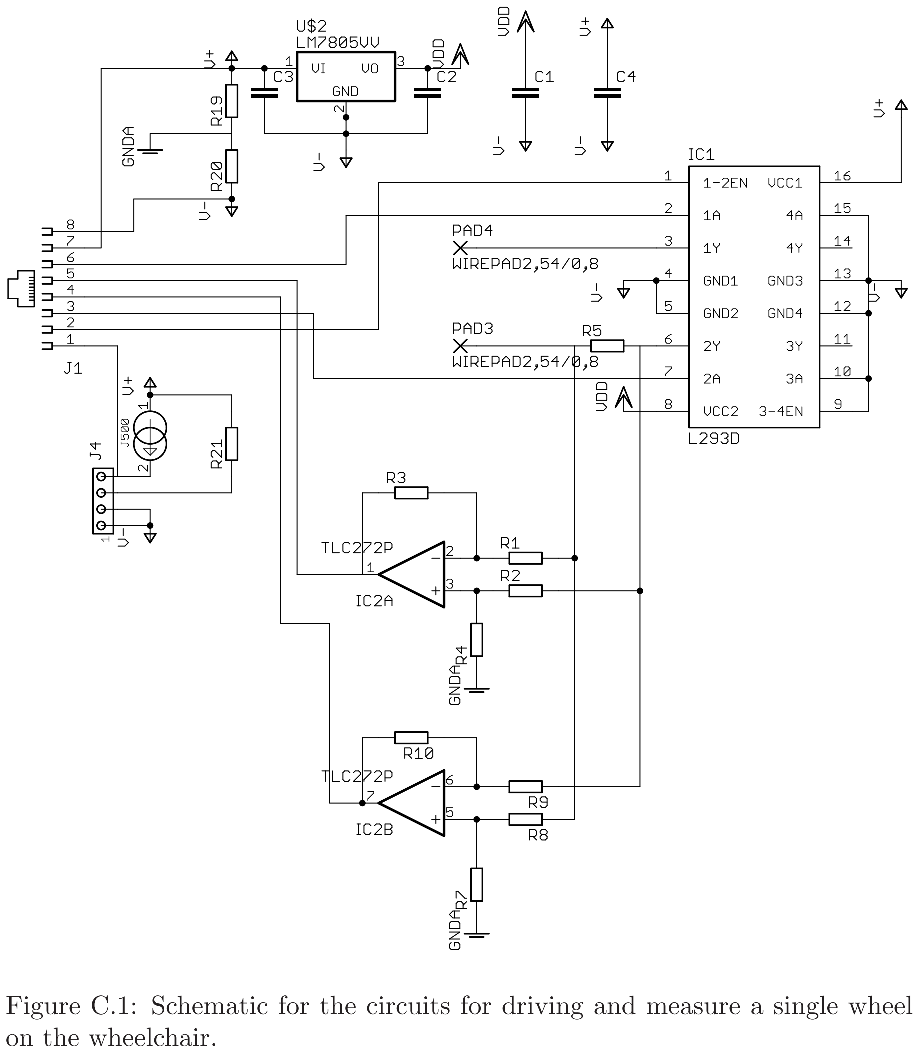


Figure C.1: Schematic for the circuits for driving and measure a single wheel on the wheelchair.

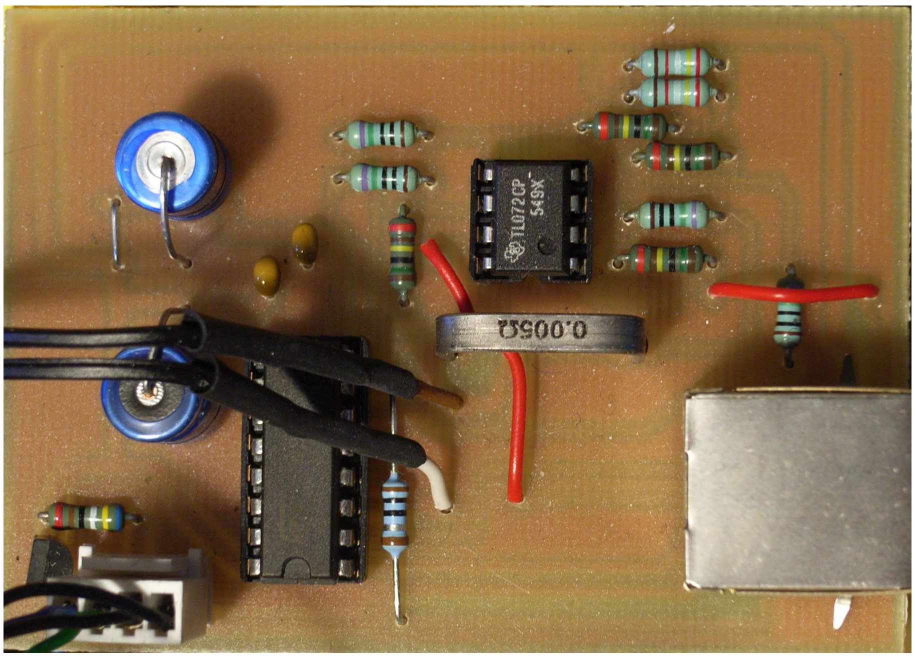


Figure C.2: Picture of the circuit.

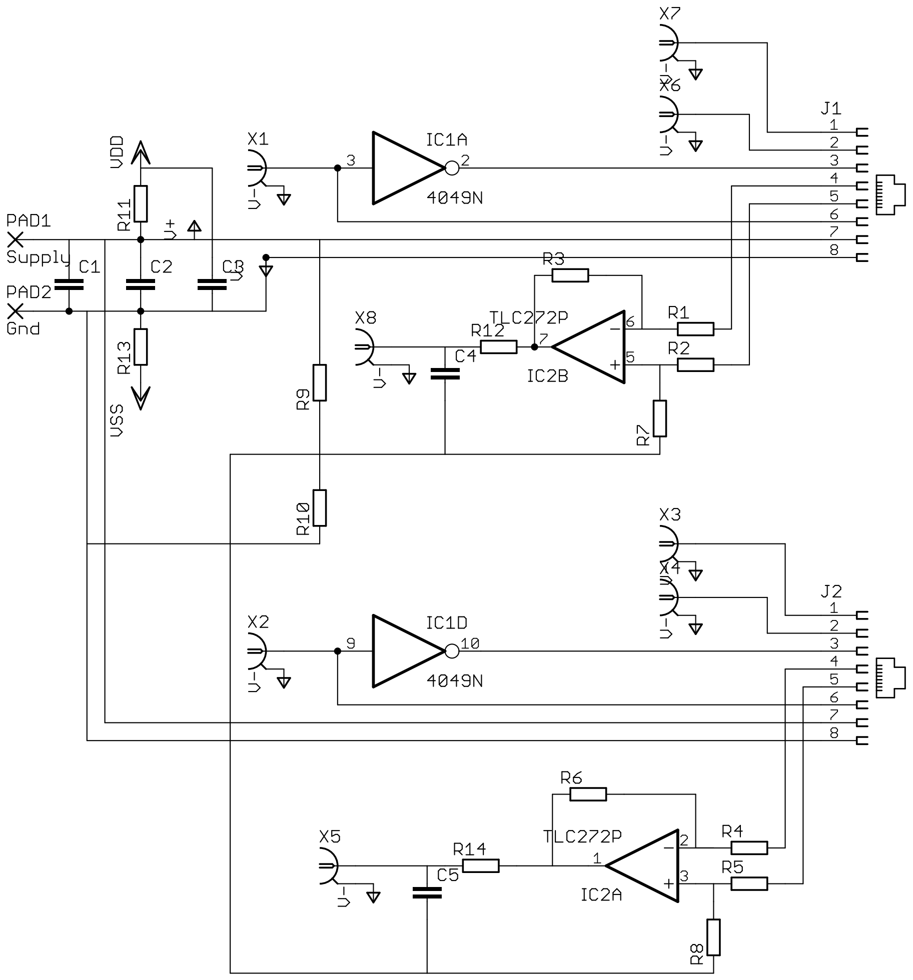


Figure C.3: Schematic for the circuits for interfacing the wheelchair to the National Instruments sampling card.

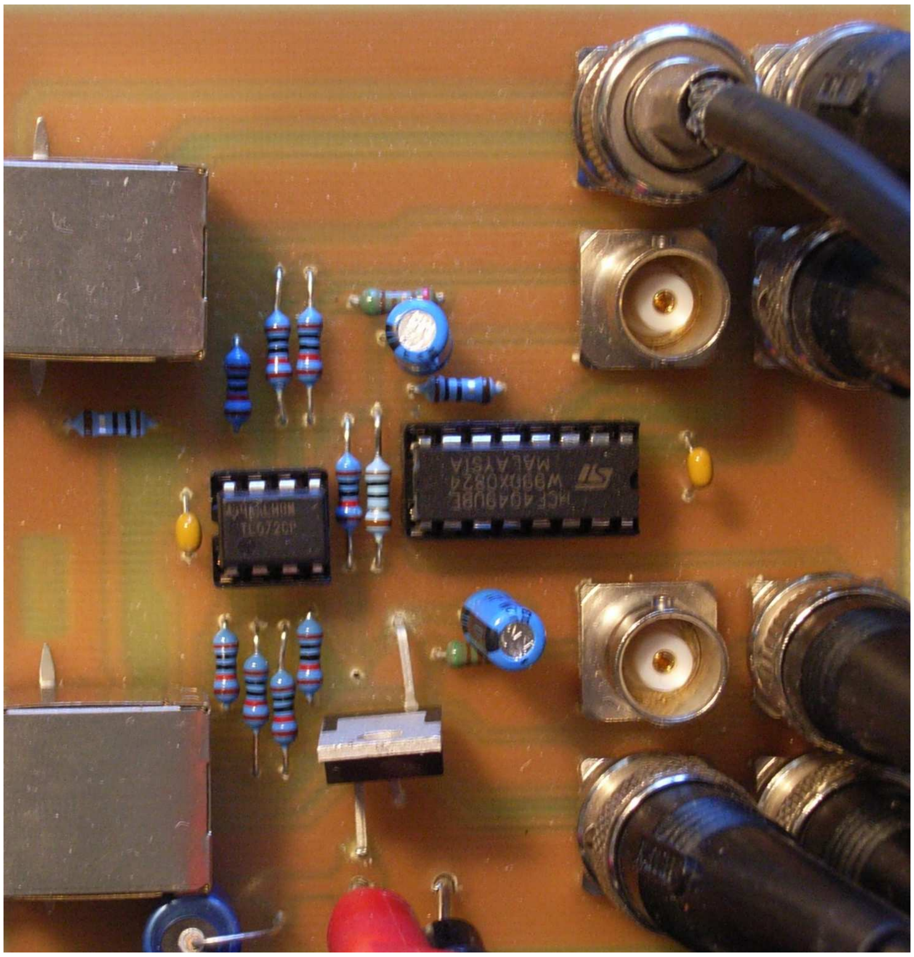


Figure C.4: Picture of the circuit.

Appendix D

# Sampling From A Discrete Distribution

The purpose of this appendix is to describe the process of sampling from a discrete distribution. First, see figures D.1 and D.2 on the following page. Here the Probability Density Functions and Cumulative Distribution Functions for a uniform and a Gaussian function are shown. The aim is to select an index i, based on the weights $w^{[i]}$, so that there is a greater chance of selecting the index which has a high associated weight. This is done using the CDF. The CDF is the integral of the PDF for continuous functions, and for a discrete distribution, the CDF is the cumulative sum of the PDF, implemented in Matlab in the *cumsum()* function.

In order to sample from the distribution, a uniformly distributed random number [0,1] is generated. This is as an entry point on the y axis on the plots, and the matching x value is then the index used as output. On the uniform distribution, it is seen that there is an equal chance of getting any index value using this method. On the Gaussian distribution, it can be seen that there is a greater chance of getting the index matching the center of the Gaussian than the other values, with the chance decreasing further away from the center. This implements the "survival of the fittest" concept, as the indexes with the highest weight are most likely to be selected.

Survival Of The Fittest

The sampling algorithm used is seen in algorithm D.1 on the next page. This algorithm is used thrice for each time a particle must be sampled. One time for each of the dimensions of the state space: $x$, $y$ and $\theta$. This actually implements a form of genetic behavior, as the best particles are combined to form new particles, that are even better.

On algorithm D.2 on the following page the resampling method used in the regular MCL algorithm is seen. This does not implements any genetic mutations, just the basic survival of the fittest concept, where the particles with the highest weight are most likely to be duplicated and those with a low weight are likely to be eliminated.

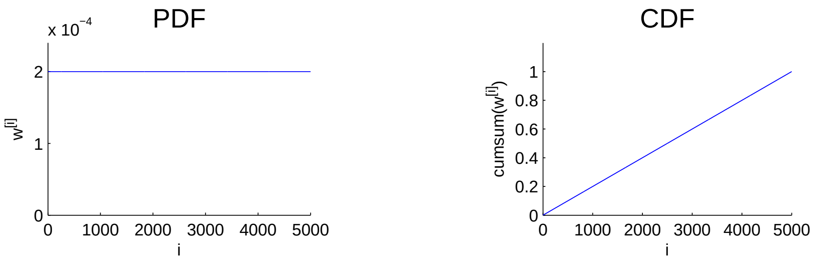


Figure D.1: Uniform PDF and CDF

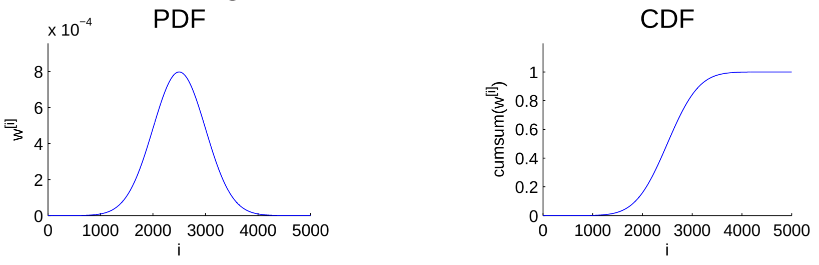


Figure D.2: Gaussian PDF and CDF

**Algorithm D.1** Sampling from a discrete distribution

**Input:** $W_t$, $X_{t-1}$

**Output:** $x_{t-1}^{[i]}$

```
$Q = \text{cumsum}(W)$
$t = \text{rand}()$
**for** $k = 1$ to $\text{size}(Q)$ **do**
    **if** $t < Q[k]$ **then**
        $x_{t-1}^{[i]} = X_{t-1}^{[k]}$
        break
**return** $x_{t-1}^{[i]}$
```

**Algorithm D.2** The Resampling algorithm

**Input:** $W_t$

**Output:** $Index$

```
$Q = \text{cumsum}(W)$
$t = \text{rand}(N+1)$
$T = \text{sort}(t)$
$T(N+1) = 1,\ i = 1,\ j = 1$
**while** $i \leq N$ **do**
    **if** $T[i] < Q[j]$ **then**
        $\text{Index}[i] = j$
        $i = i + 1$
    **else**
        $j = j + 1$
**return** Index
```

# Bibliography

Angeles, Jorge, *Fundamentals of Robotic Mechanical Systems* (Springer, 2002), 2th edition. ISBN 038795368X.

Blaer, Paul & Allen, Peter K. (2003). *Topbot: Automated network topology detection with a mobile robot. Robotics and Automation*, volume 2, no. 14-19:pages 1582.

Cabrera, Alvaro Fuentes & Dremstrup, Kim (2008). *Steady-state visual evoked potentials to drive a brain computer interface.* Technical Report, Center for Sensory-Motor Interaction (SMI), Aalborg University. ISBN 978-87-90562-71-7.

CDROM (2008).
*CDROM*
(2008).

Cox, I. J. (1991). *Blanche - an experiment in guidance and navigation of an autonomous robot vehicle. IEEE Transactions on Robotics and Automation*, volume 7, no. 2:pages 193. ISSN 1042-296X.

de Vries, Theo J.A., van Heteren, Corwin & Huttenhuis, Louis (1999).
*Modeling and control of a fast moving highly maneuverable wheelchair*
(1999).

Deitel, Harvey M. & Deitel, Paul J., *C++ How To Program* (Prentice Hall, 2000), 3rd edition. ISBN 0-13-089571-7.

Dellaert, Frank, Fox, Dieter, Burgard, Wolfram & Thrun, Sebastian, *Monte carlo localization for mobile robots.* In *IEEE International Conference on Robotics and Automation (ICRA99)* (1999).

Ding, Dan, Cooper, Rory A., Guo, Songfeng & Corfman, Thomas A. (2004).
*Analysis of driving backward in an electric-powered wheelchair*
(2004).

Doucet, Arnaud, De Freitas, Nando & Gordon, Neil (editors), *Sequential Monte Carlo Methods In Practice* (Springer, 2001), 1st edition. ISBN 0387951466.

Douglass, Bruce Powel, *Real-Time UML: Developing Efficient Objects For Embedded Systems.* Addison-Wesley Object Technology Series (Addison-Wesley, 1999), 2nd edition. ISBN 0-201-65784-8.

Fox, D., Burgard, W., Dellaert, F. & Thrun, S., *Monte carlo localization: Efficient position estimation for mobile robots.* In *Proc. of the National Conference on Artificial Intelligence* (1999).

Fox, Dieter (1998). *Markov Localization: A Probabilistic Framework for Mobile Robot Localization and Navigation.* Ph.D. thesis, University of Bonn.

Fox, Dieter (2003). *Adapting the sample size in particle filters through kld-sampling. International Journal of Robotic Resarch*, volume 22, no. 12:pages 985.

Franklin, Gene F., Powell, J. David & Emami-Naeini, Abbas, *Feedback Control of Dynamic Systems* (Pearson Education, 2002), 4th edition. ISBN 0130980412.

Grewal, Mohinder S. & Andrews, Angus P., *Kalman filtering Theory and Practice Using MATLAB* (Wiley Interscience, 2001), 2nd edition. ISBN 0471392545.

Hokuyo (2004). *URG Series - Communication Protocol Specification* (Hokuyo Automatic Co., LTD, 2004).

Hurbain, Philippe (2007). *Lego® 9v technic motors compared characteristics*. Http://www.philohome.com/motors/motorcomp.htm.

Johnson, Barry W. & Aylor, James H. (1985). *Dynamic modeling of an electric wheelchair* (1985).

Kwok, Cody C. T., Fox, Dieter & Meila, Marina, *Real-time particle filters*. In Suzanna Becker, Sebastian Thrun & Klaus Obermayer (editors), *Neural Information Processing Systems* (MIT Press, 2002). ISBN 0-262-02550-7, (pages 1057–1064).

Latombe, Jean-Claude, *Robot Motion Planning* (Kluwer Academic Publishers, 1991), 1st edition. ISBN 079239206-x.

Levine, William S., *The Control Handbook* (CRC Press, 1996). ISBN 0849385709.

Rekleitis, Ioannis M. (2004). *A particle filter tutorial for mobile robot localization*. Technical Report TR-CIM-04-02, Centre for Intelligent Machines, McGill University.

Siegwart, Roland & Nourbakhsh, Illah R., *Introduction to Autonomous Mobile Robots* (The MIT Press, 2004). ISBN 026219502X.

The ALS Association (2007). URL `http://www.alsa.org/`.

Thrun, S., Fox, D., Burgard, W. & Dellaert, F. (2000). *Robust monte carlo localization for mobile robots. Artificial Intelligence*, volume 128, no. 1-2:pages 99.

Thrun, Sebastian, Burgard, Wolfram & Fox, Dieter, *Probabilistic Robotics*. Intelligent Robotics and Autonomous Agents (The MIT Press, 2005). ISBN 0262201623.

# Autonomous Wheelchair Navigation

Jeppe Møller Holm and Søren Lynge Pedersen

This Project describes how to make an autonomous wheelchair for people with Amyotrophic Lateral Sclerosis (ALS). Because people with ALS cannot use their muscles to propel or even control a normal wheelchair, it is necessary that the wheelchair can be controlled by the only part of the body that they still have full control over; their eyes. It is investigated how to sense the house, how to localise the wheelchair in the house, how to plan a path through the house and how to drive a wheelchair. Mounted on the wheelchair to sense the house is a laser range scanner. The output from this along with the odometry is used in a particle filter for localisation, incorporating an Augmented Adaptive Monte Carlo Localisation. To plan the path through the house, the Configuration Space of the house is computed and a Generalised Voronoi Diagram calculates waypoints with the maximum distance to the walls. A Dijkstras Algorithm finds the shortest path from start to goal. To avoid unmapped obstacles, repulsive velocity vector fields are implemented. The controller to control the wheelchair is an optimal model based LQR. This project has been a proof of concept, where it has been shown that it is possible to make an autonomous wheelchair, capable of transporting a user with Amyotrophic Lateral Sclerosis, or any other disabled person, with no user input except where to go. The wheelchair does not need to be told where it is, or how to get to the goal. It does not even need any artificial landmarks for navigation. Using only onboard sensors and a map of the house, which was available anyway, it performs the required tasks.

Keywords: *Mobile Robot Navigation, Occupancy Grid Based Map, Laser Range Scanner, Kalman Filter, Augmented Adaptive Monte Carlo Localisation, Configuration Space, Generalised Voronoi Diagram, Dijkstras Algorithm, Vector Field Navigation, LQR.*

June 2008
Section for Automation and Control
Aalborg University, Denmark