AAL-2009-2-049, ALIAS
D6.5
Final navigation software module

Due Date of Deliverable	2013-02-28
Actual Submission Date	2013-05-13
Workpackage:	6.5
Dissemination Level:	Public
Nature:	Report
Approval Status:	Final
Version:	v1.0
Total Number of Pages:	47
Filename:	D6.5-IUT-NAV-v0.1.pdf
Keyword list:	navigation summary, appraoch person, person observation, spatio temporal planning, remote control, exploration

Abstract
This report describes the final navigation system, developed within the ALIAS project within the last 3 years. It focuses on the final description of all previous navigation modules, like approach a user, avoid a user, observe a user, and remote control the robot. We also highlight the updates towards the last deliverables. Additionally, the current state of the person tracker is described and a new module to build a map is presented.

History

Version	Date	Reason	RevisedBy
0.1	2013-02-15	created [IUT]	Jens Kessler
0.1	2013-02-28	first review [TUM]	Juergen Geiger
0.2	2013-03-08	first revision with remarks from review	Jens Kessler
0.3	2013-05-08	second review [FhG]	Sven Fischer
0.4	2013-05-08	second review [FhG]	Stephan Wabnik
1.0	2013-05-13	included remarks of second review	Jens Kessler

Authors

Partner	Name	Phone / Fax / Email
IUT	Jens Kessler	Tel: ++49 3677 69-4170 Fax: Email: jens.kessler@tu-ilmenau.de

Table of Contents

1 Introduction

The primary objective of the ALIAS project is to develop a mobile robot platform that is designed to assist elderly users and people in need of care to continue independent living with minimal support from carers. The main functionalities of the robot platform will include the ability to interact with users, monitor their well being and provide cognitive assistance to them while using social networks and other communication platforms in daily life. In these situations, the functionalities are supported by the ability of the robot to move autonomously and also focus during the navigation on persons within its field of operation.

Since the project is now about to end in nearly four months, the purpose of this deliverable is to sum up all developed navigation modules with the current status of development, as well as the state of the person tracker. In former deliverables we described how to politely approach a person [11], how to politely avoid a person, and how to observe a person in an unobtrusive manner [12]. Also, we focused on the ability of the robot to be steered remotely in a safe fashion. All these abilities will be summarized in this deliverable with all updates developed since the previous deliverables. Additionally, we began to investigate the task of autonomous map building, since this is a very useful feature for the robot to get installed into new environments.

This deliverable is structured as follows: in chapter 2 a short overview of the current navigation system with all its modules is given. In chapter 3 our method to approach a person is given with a set of small updates. Afterwards, in chapter 4 the approach to avoid a moving person is described, while in 5 our approach on finding a good observation position is shown. Here, most improvements are done. The chapter 6 describes briefly the mechanism of remote controlling the robot and in chapter 7 the state of autonomous exploration and the current state of the person tracker are shown. The deliverable ends with chapter 9, where the deliverable is summarized.

2 System Overview

This chapter presents an overview of the navigator structure as shown in figure 2.1. The navigation core is defined by the motion controller, which implements the so called "Dynamic Window Approach" [6], which was extended by MetraLabs to allow a more modularized architecture (see [5]) by switching "objectives" on or off. In the dynamic window, a set of possible next motion commands is evaluated by voting for each command by the set of objectives. By switching objectives on or off, different tasks could be realized. So, different driving behaviors could be realized by the same controlling mechanism by just using a different set of objectives. The navigation behaviors could easily switch during runtime operation. But to navigate safely a set of requirements has to be fulfilled, which is namely the ability to localize the robot in a given map and to build such a map. Both aspects are described in the next section.

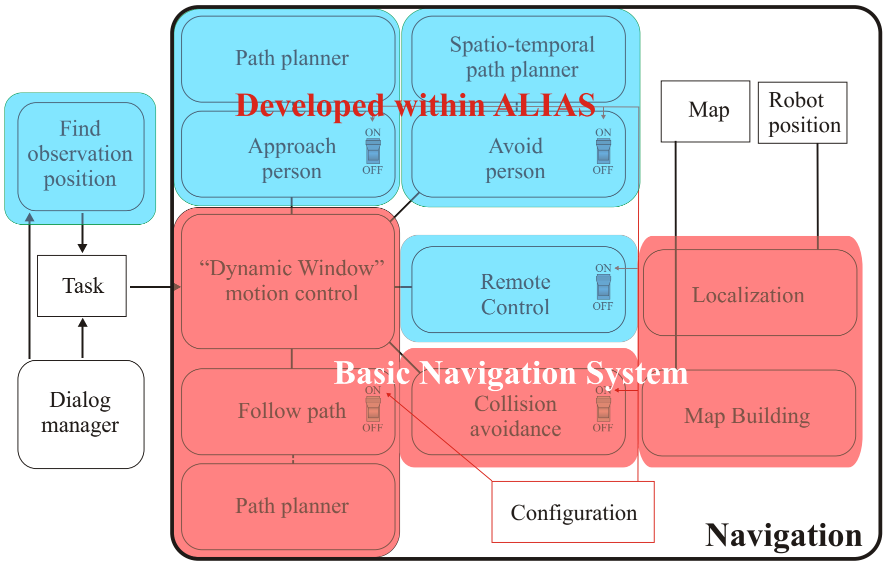

Figure 2.1: The basic modules of the navigation system are shown in red and are provided by MetraLabs. The blue parts are created by IUT. Note, that modules with a switch could be turned on and off during runtime. The full control over the navigation resides within the dialog manager. This deliverable will give an overview over all parts of the navigation system.

2.1 Navigation requirements: mapping and localization

To safely navigate a robot in its environment, the robot has to know its position within the operation environment, which requires the knowledge, how the environment looks like. This is stored in a map, which the robot can use during navigation. It is in fact very problematical to build such a map, since the robot could only know its position with the help of a map and a map could only be estimated with a sequence of known positions. This is a chicken and egg problem. What happens, if the estimation of the sequence of known positions is done in an naive way, is shown in Fig. 2.2. So, the problem of building a map is in fact an estimation problem which estimates the current robot position *and* builds with the sequence of estimated positions and the gathered sensor information a consistent map *at the same time*. This problem is widely known as the simultaneous localization and mapping (SLAM) problem. There are numerous solutions for this problem available. We use two options to build a map. Option one is a simple manual building of the map, where the user defines a set of reference points, where the robot is driven to pre-known absolute positions. So, the problem of finding a set of correct robot locations is simple avoided. The results of this mapping approach simply depends on the correct measurement of the reference points and the correct placement of the robot at these points.

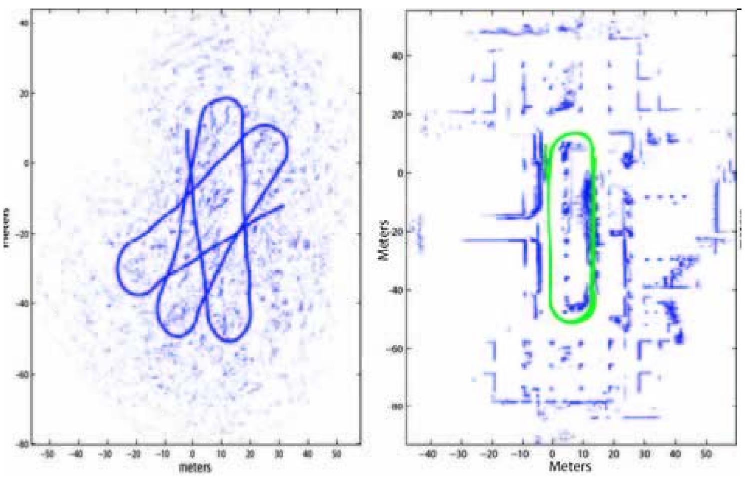

Figure 2.2: The problem of map building: by measuring the robot's position with only wheel spinning or gyroscopes, the resulting track is clearly wrong (left side). A SLAM problem has to be solved to get the correct map (on the right).

The second option is know as GMapping, as proposed by Grisetti [7]. It is open source code and could be used to truly solve the problem of synchronous localization and mapping. For most of our experiments and user trials it was sufficient to use the first simple handmade approach, since we do not have very complex environments. Anyhow, to get the navigation system set-up in an automatic fashion, the second approach is required and has to be extended by an autonomous exploration behavior. During the last period of this project we focus on the aspect of an exploration behavior of the robot (see chapter 7).

The localization is simple. Monte carlo state estimation is used to match the current sensor information from the laser with the map. With its sequential estimation with respect to the previous state distribution this method is relatively stable. Here, again open source software is used, known as AMCL [1].

[1]www.ros.org/wiki/amcl

This should conclude the basic requirements of map building and localization. Both aspects where not part of the development work in ALIAS and are mentioned only to give a complete overview of the system. In ALIAS, we use state-of-the-art methods to solve on the one hand the map building problem and on the other hand to localize the robot within that map. The current robot pose as well as the environment map allows the robot to move. Without both aspects, a useful and goal driven mobile behavior is not possible.

2.2 Navigation core: different behaviors

As stated above (and shown in Fig. 2.1), the core of the behavior generating aspect of navigation is controlled by the dynamic-window motion controller and the set of *active* objectives. The configuration of the navigator is changed by the application controller, or in our case, by the Dialog Manager. Also, the navigation task is given by the application (e.g. drive to a place, observe someone, be remote-controlled) and it has to be assured, that the navigator configuration and the given task corresponds to each other.

The module of the motion controller as well as the tuple of the path-following and the obstacle avoidance objectives have been developed by MetraLabs. We integrated our software into the provided system. During the development of ALIAS the middle ware for the navigation system also switched from the former blackboard based approach towards a point-to-point data transfer approach. All software modules where migrated towards the new middle ware called MIRA [2]. This middle ware was developed by IUT and MetraLabs outside the ALIAS project and is available via an open source license. The interfaces towards the dialog manager where left unchanged, so the migration is transparent towards other partners.

The idea of the dynamic window is very simple. The state of the robot is defined by its current position and the current driving speed of the robot. Now, a set of feasible velocities for the next motion step is evaluated. For each speed configuration a set of "objectives" should give their vote, if this configuration supports the given task or not. And so, the set of objectives can control the robot behavior. For example, during classical navigation only the objectives for following a path and avoiding obstacles are active. The remaining objectives have to be deactivated. The objective for following a path uses a path planning algorithm in the background to be able to vote for feasible driving commands. The collision avoidance objective only reacts on the local perceived obstacle situation and does not consider which driving command leads towards a goal. Only the combination of both objectives enables the robot to drive towards the goal.

One main navigation aspect of ALIAS is, to enhance the set of objectives to enable different robot behaviors like approaching a person or remote control the robot. Further details will follow in chapters 3 and 4.

[2] www.mira-project.org

2.3 Additional navigation modules: observe a person, remote control and exploration

The only module that does not belong directly to the navigator core is the observation module, the GUI for remote controlling the robot at the non-robot side of the use case, and an application to explore the home environment in order to autonomously build a map. The observation module only has one task: to find a good observation position. It does not cause the robot to drive to this position directly, but leaves the position to the dialog manager to use it if needed.

The GUI for the remote control process simply shows the robot's position within the map and adds two camera images to the user interface. It acts as a front end to the remote controller and sends back drive commands, which the robot could (but not have to) execute. This is the only direct connection to the navigation module.

The exploration module is a stand-alone application, which does not interface with any other module from the ALIAS architecture. Currently, this application is in an early prototype state.

Further details will follow in chapters 6,5 and 7.

3 Approaching a person in a social acceptable way

In the previous chapter the idea of the Dynamic Window approach has been presented. Since social acceptable navigation is not supported by the original approach we had to extended it (see figure 3.1). This chapter describes the extension for the first task, to approach a user, previously detected by the person detection module. For a more detailed description of this approach the reader should refer to deliverable D6.3[11]. Here, the main ideas are summed up and shown in short.

First of all we rely on the findings of psychologists and their personal space model. This defines a kind of protection area around each person, where the robot could only intrude from the front. Afterwards the used planning method is describes shortly.

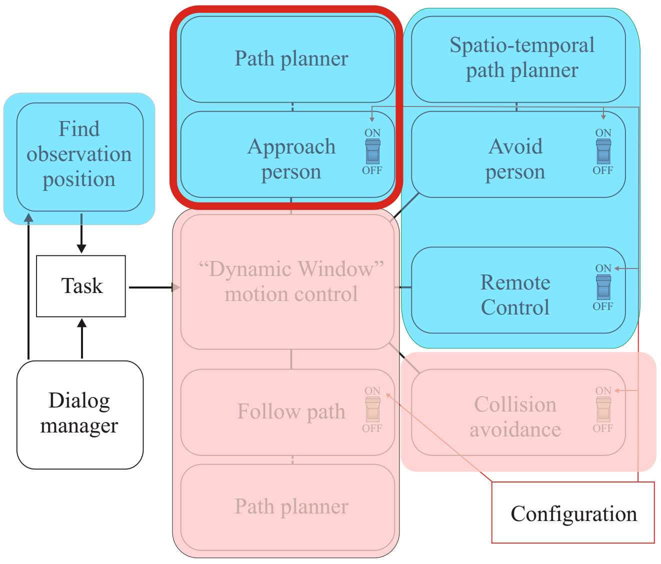

Figure 3.1: the "approach person" software module within the navigation architecture. This module is directly linked to the navigation core and extends the capabilities of the robot to approach a person.

3.1 The model of the personal space

Psychologists investigated the human-to-human interaction in public areas very carefully since the 70s of the last century. One of the foundations and most important findings is the work of Hall [8],[9], who first introduced the concept of different spaces around a human being to support different modes of interaction. There is a space for non-interaction,

zone	**interval**	**example situation**
close intimate	0.0 m - 0.15m	lover or close friend touching
intimate zone	0.15m - 0.45m	lover or close friend talking
personal zone	0.45m - 1.2m	conversion between friends
social zone	1.2m - 3.6m	conversion to non-friend
public zone	from 3.6m	no private interaction

Table 3.1: Psychological definition of the personal space. This space consists of 5 zones, each supporting different activities and different communication intentions.

public interaction, interactions with friends and also an intimate space for interaction with very close relatives.

By formulating the theory that interaction is also coupled to spatial configurations between interaction partners, many investigations on this matter have taken place, and it could be shown that the configuration depends on many aspects like cultural background, age, sex, social status and person's character.

The model of the personal space is the key component to approach a person. We want the robot to approach a person from the front, but with a slight aberration from the direct front, since most user perceive such a behavior more comfortable. For this purpose, obviously we need the position and viewing direction of the person to calculate the configuration of the personal space model. The space configuration should enable the robot to drive around the person in a comfortable distance and turn towards the person when a "front position" is reached. Like in [15], we model the personal space with a sum of Gaussians. The space relative to the persons upper body direction is separated into two regions: a front-region, which is considered to be within $\pm 45°$ around the persons upper body direction, and a back-region, which is the rest (see Fig. 3.2).

In both areas we define a distance function to keep the robot out of the user's personal zone but within his/her social zone while approaching the person.

$$a(x,y) = \frac{\alpha}{2\pi\sigma_1} \cdot e^{-\frac{x^2+y^2}{\sigma_1^2}} - \frac{\beta}{2\pi\sigma_2} \cdot e^{-\frac{x^2+y^2}{\sigma_2^2}} \tag{3.1}$$

The variables $\alpha, \beta, \sigma_1, \sigma_2$ describe a classical Difference of Gaussians function and are set in our case (see Fig. 3.2) to $\alpha = 0.6, \beta = 0.3, \sigma_1 = 2m, \sigma_2 = \sqrt{7}m$ to form a minimum cost region in a distance of 3.5 meters around the person. The front region is treated additionally with an "intrusion function" $i(x,y)$. This is also a Gaussian function and is simply added to $a(x,y)$.

$$\begin{aligned} i(x,y) &= \frac{\gamma}{2\pi\sqrt{|\Sigma|}} \cdot e^{-\vec{x}^T\Sigma^{-1}\vec{x}} \qquad (3.2) \\ \Sigma &= \begin{bmatrix} \sigma_x^2 & 0.0 \\ 0.0 & \sigma_y^2 \end{bmatrix} \cdot \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix} \end{aligned}$$

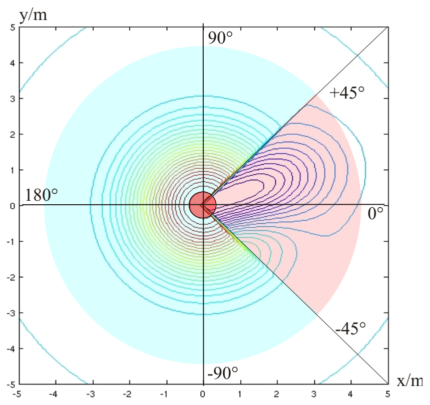

Figure 3.2: Two regions of our personal space model. The front region is within an $\pm 45°$ interval (in red). The back region is the rest (in blue). Note, that the regions are not limited in radial extension, like it is done in the illustration.

Here the variables σ_x and σ_y define an elliptical region, that is rotated towards the needed approaching direction ϕ, as seen from the persons perspective. The vector $\vec{x}$ is simply a column vector $(x, y)^T$. The variables are set to $\gamma = -0.5$, $\sigma_x^2 = 2.9$ and $\sigma_y^2 = 1.1$. Only ϕ and σ_x need to be set at runtime to regulate the approaching distance and direction. All other parameters are constant and are chosen to reflect the properties of the personal space definition in [8]. So, the final definition of the personal space $p(x, y)$ relatively to the person coordinates $x = 0, y = 0$ and upper body pose towards the x-axis is defined as follows:

$$p(x,y) = \begin{cases} a(x,y) \text{ , if } \langle x,y \rangle \text{ in back-region} \\ a(x,y) + i(x,y) \text{ , if } \langle x,y \rangle \text{ in front-region} \end{cases} \tag{3.3}$$

Version update

In the first version of the approaching objective the minimum point of the personal space function was chosen to be also the goal, where the robot has to drive to. In practice this idea is not optimal, since often this place is not reachable by the robot, since a table or a chair could block this position. For that reason we use a bimodal density function, which has the exact similar structure than $i(x, y)$ with the angle ϕ and $-\phi$. So we draw samples from the function $i_\phi(x, y) + i_{-\phi}(x, y)$ until a cell is found, where the robot is able to drive to. This cell is set to be the driving goal for the robot.

3.2 Fast Marching and the cost function

To enable the robot to follow a cost optimal path along the defined cost function, path planning methods are used to create a continuous decreasing function to get to the goal

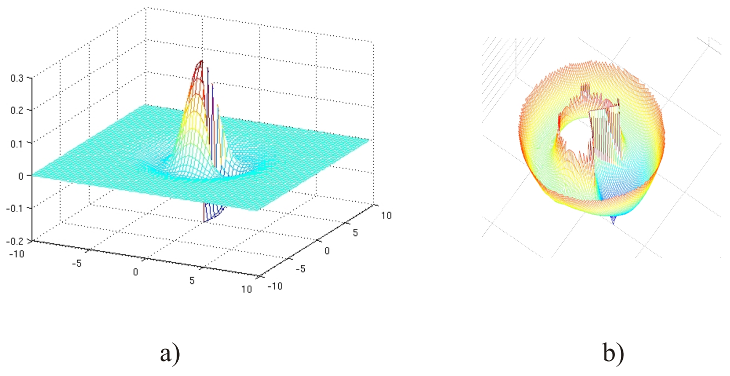

Figure 3.3: From personal space to the planning function. The personal space function in a) is transformed to create the continuously decreasing planning function b).

position by gradient descent (see Fig. 3.3 for cost function on the left and the resulting planning function on the right). An excellent planning technique is the Fast Marching method [14], which origins from the level set methods of single wave fronts and is applied to path planning. The core idea is to code space as a physical medium, where waves can travel with different speeds. For example in obstacles the speed is nearly zero, while in free space the speed can be any positive non-zero value. By propagating a wave front from the target to the robot, a function of the traveling time of the wave for every point in space is constructed. The benefit is, that also fuzzy values, that are neither obstacles nor free space, can be considered in this simulation and deform the initial circular waveform. So all we have to do, is to transform the personal space into a physical "speed-space". High values of the personal space symbolize bad places to drive to, while low values should be preferred. So we define the speed function $v(x, y)$ as follows, where p_{min} is the minimum of the personal space function:

$$v(x, y) = 1/\left(p(x, y) - p_{min} + \epsilon\right) \tag{3.4}$$

The variable ϵ is used to prevent an infinite speed at the minimum point. The function $v(x, y)$ is than used by a standard fast marching planner (see [14]) to extract a path, which the robot can follow.

3.3 Conclusion

In this chapter we summarized a method, working within the Dynamic Window Approach, to approach a person by considering his/her personal space. We could demonstrate, by using a planning strategy, that a stable and reliable solution could be achieved. This approach appears to be stable and is finished within the ALIAS project.

4 Avoiding a moving person during driving

In this chapter, an approach is presented, which on the one hand predicts the movements of persons in a very simple way, and on the other hand uses the predicted movements to plan a motion path, which avoids the moving person. The goal of our development was an early avoiding behavior of the robot, when the robot passes a person. This should increase the acceptance of the robot, when operating in private homes, and signal a "busy"-behavior towards the users. Within the ALIAS project, the proposed method is applied as an alternative planner to the standard planner and therefore transparent to all other modules (see Fig. 4.1).

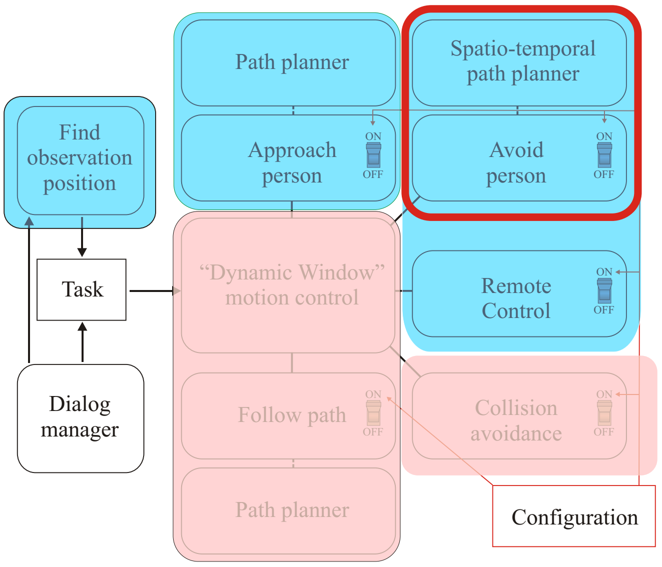

Figure 4.1: the "avoid person" software module within the navigation architecture. This module is directly linked to the navigation core and completely replaces the classical path planning module. So, only one of both modules could be active during operation.

4.1 Introduction

In our work we want to emphasize the case of human-robot interaction, when the robot does *not* want to interact with a person. In semi-public environments, like nursing-homes or hospitals, this is very often the case. For example, when the robot has to drive to its charging station an interaction with a passing person is not wanted. In such cases, the robot has to signal its busy state. In our work, the spatial distance from Hall is used (see chapter 3), which corresponds to "non interaction", and which therefore represents a meaningful distance for a human being. All persons (or robots), that keep a distance

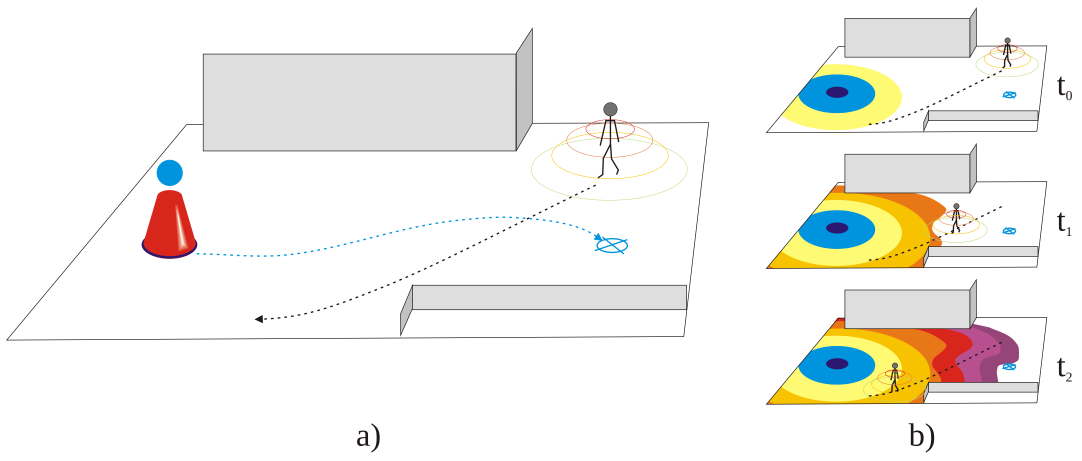

Figure 4.2: the idea of the approach: the robot should be able to politely pass a moving person. To do so, the person path is predicted (see a)) and the personal space of the person is used in a spatio-temporal planning process to compute a feasible path. In b), a planning wave is propagated from the robot origin towards the goal (blue cross). This wavefront could be deformed from the obstacles as well as from the moving personal space from the predicted trajectory.

above this threshold, are interpreted as potential non-interaction partners. In this software module, we use a simple mathematical model of the personal space, to allow the robot during the path planning phase to take into account the predicted motion of an observed person. A non-intrusive path towards a predefined goal, which does not touch the personal space of a person, is planned if enough space is available.

Presented approach

Our approach uses a modified version of the Fast Marching Method (see [14]), to propagate a wavefront into the environment. The passing times of the wavefront could be afterwards used to extract an optimal path. The passing time of the wavefront is determined by physically correct simulation of the wave, and is directly related to the physical abilities of the robot, like maximum traveling speed, and the restrictions of traveling speed coming from the static and dynamic environment. The static restrictions are the obstacles. The dynamic aspects of the environment are considered to be the predicted motion trajectories of persons (and their personal space). As stated before, we use a potential field method to predict the trajectory of the moving person. A brief overview of the key idea of the presented approach is shown in figure 4.2.

4.2 Integration into the navigation software system

This module completely replaces the classical (static) path planning approach. For this reason it is also designed as an objective with path planner included. The only modification is the usage of time during the planning process and the additional short term prediction of person motion. The mechanism to follow the planned path is exactly the same as in the standard module. The idea is simply to choose the particular action (rotational and translational speed), which best follows the gradient of the resulting navigation function.

4.3 Prediction of the person's trajectory

In this section, the prediction method of the person trajectory is presented. We propose a very simple, physically inspired model, also known as potential field. This model is very often used in robot navigation to avoid obstacles or approach a target [10, 13]. However, here it is used to predict near future person movements. The key idea is to model the environment as a set of point like electrical charges, which create an electrical field. This field could affect other charges by applying a force towards them. Two forces are modeled to predict the motion trajectory. On the one hand, the pushing forces of the obstacles are used, so the person does not collide, and on the other hand, the pulling forces of a virtual target line in front of the person are modeled.

4.3.1 The Potential Field

To compute the vector field of forces, a grid based world representation is used. If a cell contains an obstacle, a negative charge is defined there. A free cell does not contain any charge. The person itself represents also a negative charge and is attracted by a virtual line of positive charges in the current motion direction of the person. An example setting is shown in figure 4.3. To extract the impact of all charges, the definition of the electrical field is applied to compute the resulting force. For a given set of charges in positions $\vec{x_i}$, the field at a position $\vec{x}$ is defined as:

$$\vec{E}(\vec{x}) = \sum_{i=0}^{n} Q_i^- \cdot \frac{\vec{x} - \vec{x_i}}{|\vec{x} - \vec{x_i}|^3} \tag{4.1}$$

Note, that the resulting force on a negative charge is proportional to the vector $\vec{E}(\vec{x})$. For the static part of the environment, in each free cell the vector of the field could be preprocessed. Also, the moving person is attracted by a line of positive charges, which is always placed in front of the person. So, the resulting force is the vector sum of a force towards the current motion direction and a disturbing force, sourced by the obstacle configuration:

$$\vec{F}(\vec{x}) = Q^-(\vec{E}_{obs}(\vec{x}) + \vec{E}_{target}(\vec{x})) \tag{4.2}$$

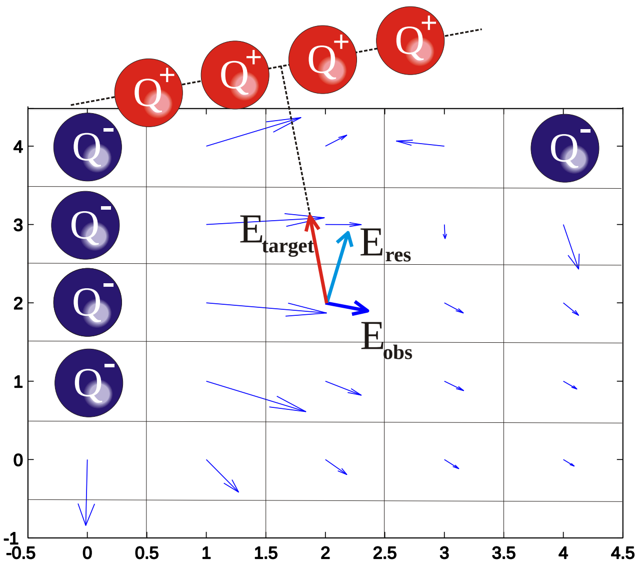

Figure 4.3: this image shows the resulting vector field $\vec{E}(\vec{x})$, which is sourced by the negative charges of the obstacle cells. The resulting force on the moving person is defined by two components. The pushing field E_{obs} of the obstacles (blue) and the pulling force E_{target} of the virtual target line (red). This results in a field vector E_{res} (light blue), which is proportional to the applied force.

The idea of predicting the trajectory is, to simulate the movement of a person by considering the force $\vec{F}(\vec{x_j})$ in the currently predicted position $\vec{x_j}$ and create the next motion vector and position. It is easy to understand, that such an approach only needs a map of the current environment, a valid person position and a valid walking direction to provide a sufficient prediction of the person's trajectory. Note, that this method does not give good results on long term trajectories! In this case statistical methods should be used, which need a rather large training set of recognized trajectories.

4.3.2 Motion Prediction

We "predict" the motion of a person as if the person would be a charged particle inside the previously calculated electrical field. If the motion of a charged particle within the resulting force field should be processed, the well known momentum equation could be used for that: $m \cdot \vec{v}_{t+1} = m \cdot \vec{v}_t + \vec{F} \cdot \Delta t$. Here, m denotes the mass of the charged particle, $\vec{v}_i$ denotes the speed at time i, and Δt is the time interval for one simulation step. Reformulated to $\vec{v}_{t+1} = \vec{v}_t + \vec{F}/m \cdot \Delta t$, it could be seen, that the mass influences the update of the speed. With a huge mass, the speed update is fairly slow and could lead to collisions. This changes, when the mass tends to small values. Since a collision free path of the person should be constructed, the mass is set to zero and only an approximation of the momentum equation is used to update the current person speed:

$$\vec{v}_{t+1} = |\vec{v}_t| \cdot \frac{\vec{F}}{|\vec{F}|} \cdot \Delta t \tag{4.3}$$

By re-defining the momentum equation, only the direction of the person prediction is influenced by the potential field and the absolute value of the person speed is left constant. The trajectory of the moving person is calculated by sequentially applying equation 4.3.

The predicted person's path is used for the robot's motion planning.

4.4 The Adapted Fast Marching Planner

As stated before, the Fast Marching Method approach from Setian [14] is used for robot path planning. It is executed on a regular grid, where each grid cell contains a cost value that physically reflects a speed, at which a virtual wavefront is able to travel through this cell. Near zero values are assigned to obstacle cells, whereas high values are assigned to free space. In our planner, the main idea is to evaluate the speed, the waveform can travel through a cell element *at the time*, the cell is reached by the wavefront. The wavefront thereby represents all positions, the robot is able to reach at that given time.

4.4.1 The Fast Marching Method

Before the actual planning process starts, the given map is divided into obstacle and non-obstacle cells by a simple threshold operation. Afterwards, the map is dilated by the radius of the robot, so that obstacles now appear larger and a point-like robot can be assumed. This setting defines a static velocity value for each cell, which is v_{max} in free space and linear interpolated towards zero, when the cell is near an obstacle.

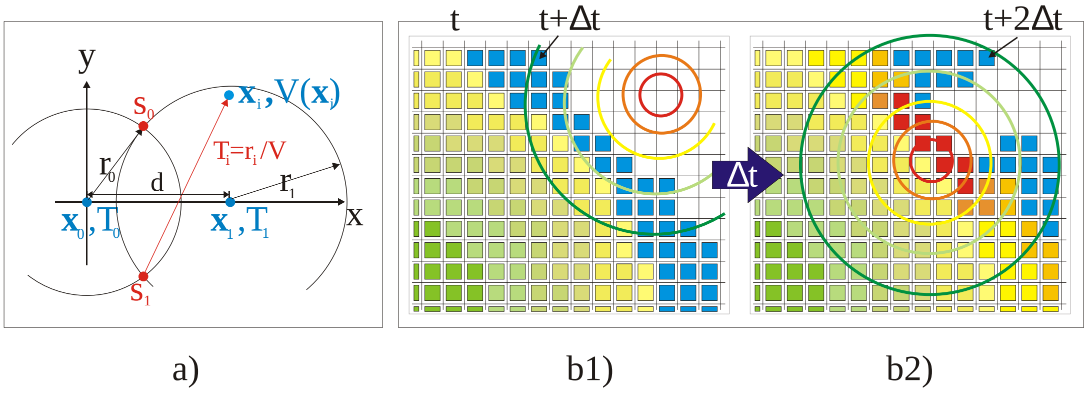

Figure 4.4: in image a), the details of the interpolation of one cell element of the wavefront are shown. Blue values are the given ones, while black values are computed. The red values describe the final step of interpolation, where from the virtual wave sources s_0 or s_1 the passing time of the wavefront is calculated. On the right side b) a full simulation step is shown, where the personal space intersects the wavefront. Note, that only the blue elements of the wavefront investigating the current speed configuration, while the computed values remain unchanged. The wavefront is only updated with the current configuration until the elements reach the simulation time $t + \Delta t$, shown in b1). Afterwards, the speed configuration is updated to $t + \Delta t$ and the propagation of the wave runs until $t + 2\Delta t$ is reached (see b2)).

Fast Marching proposes a very simple numerical solution to the so called Eikonal equation problem, where an ever expanding closed curve is propagated over time. The wave starts from a single point (e.g. S_0 from figure 4.4 a)) and expands to neighboring points by using grid cell centers, which are currently part of the wavefront. All expanded points of the waveform are added to an open list, sorted by the interpolated travel times. Sequentially, the elements of the open list with the smallest traveling time values are expanded and removed from the list afterwards, until no expandable cells remain. In this way, the wavefront expands slowly over all points. Each expansion step is done by interpolating the wavefront for the currently observed cell element $\vec{x}_i$. For the interpolation of the cell element, the traveling times T_0, T_1 and positions $\vec{x}_0, \vec{x}_1$ of the two neighboring elements with the shortest traveling times are needed. Also the current valid speed of that cell $v(\vec{x}_i)$ has to be known. In the first step, the positions $\vec{s_0}, \vec{s_1}$ of possible sources of the wavefront are calculated:

$$\begin{aligned} r_0 &= v(\vec{x}_i) * T_0 \\ r_1 &= v(\vec{x}_i) * T_1 \\ s_x &= (d^2 + r_0^2 - r_1^2)/2d \\ s_y &= \pm\sqrt{r_0^2 - s_x^2} \\ \vec{s_0} &= \langle s_x \; ; \; +s_y \rangle \\ \vec{s_1} &= \langle s_x \; ; \; -s_y \rangle \end{aligned}$$

Here, d is the distance between $\vec{x}_0$ and $\vec{x}_1$ and defines the X-axis of the solution. As seen in figure 4.4a), there exist two possible sources $\vec{s_0}, \vec{s_1}$ of the wave origin to reach $\vec{x}_0$ in T_0 and $\vec{x}_1$ in T_1. The most distance source to our point $\vec{x}_i$ is chosen, since the point $\vec{x}_i$ would already have been interpolated if the nearest source is correct. With the correct source $\vec{s_j}$, the interpolation of the wave crossing time at position $\vec{x}_i$ is trivial:

$$T_i = \frac{|\vec{x}_i - \vec{s_j}|}{v(\vec{x}_i)} \tag{4.4}$$

Note, that for very small values of the traveling speed, the passing time T_i will become very large and such elements of the open list are expanded later. This is the case when the wave hits an obstacle cell or the personal space in our case.

4.4.2 Adaptation for Predicted Motions

To adapt the described interpolation method to time variant traveling speeds of $v(\vec{x}_i, t)$, a number of changes are necessary. First, the planning direction is reversed. Normally, a path from the target position to the current robot's position is planned. Since the traveling times of the wave have in our case a physical meaning, and to fuse the motion prediction with the planning process, the path is planned from the robot towards the goal. This means, the current robot's position is the source of the wavefront. Second, the fusion of time slices and space is the fundamental change in wavefront propagation. Hereby, the system starts from a time t_0 and updates the prediction of the person movement *as well as* the propagation of the wavefront in time intervals Δt. In Fig. 4.5 an example force field for person motion prediction is shown. The obstacle and world configuration are assumed to be static in this time interval, and the wave traveling is continued only in this short interval. This means for the n-th planning step, that only those elements from the open list are expanded, whose travel times are smaller than $t_o + n \cdot \Delta t$ and for the expanded elements, the dynamic speed function $v(\vec{x_i}, t_0 + n \cdot \Delta t)$ is evaluated. A sketch of the idea is shown in figure 4.4.

4.4.3 Following the Calculated Path

The planning is complete, if the wavefront has reached the predefined target cell. Note, that our approach also calculates *when* the target is reached. At this point each cell, passed by the wavefront, contains the passing time. The needed driving path is calculated by performing a gradient descent from the target cell towards the robot's original position. The robot has to follow this path as good as possible with the defined speeds, also calculated during the planning process. Note, if the person deviates to much from the predicted path in space and time, a replanning has to be performed. This is triggered, if the three dimensional Euclidean distance $|(x_p^{pred} - x_p^{obs}), (y_p^{pred} - y_p^{obs}), (t^{pred} - t^{obs})|$ is above a certain threshold (in our case 2.6 meters).

4.5 Experiments and Results

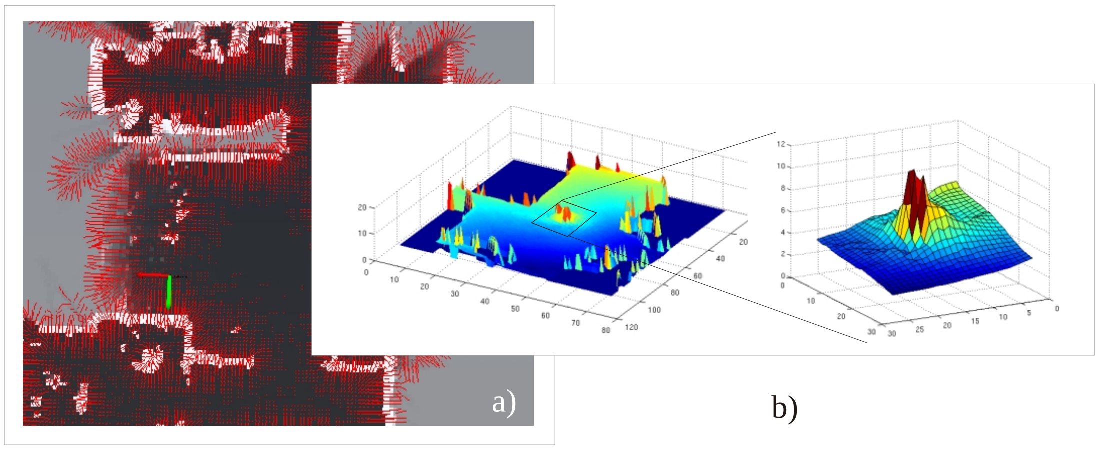

Figure 4.5: in a), an example of the force field is shown, which is used for motion prediction. In b) the function of the passing times of the wave is shown. From this function the resulting path is created by gradient descent from the target towards the robot's position. It can be seen, that the traveling time raises, when the wavefront hits the personal space of the person. A detailed view of that part of the function is shown on the right.

During the experiments, two scenarios with different characteristics where evaluated. In the first scenario, a person moves on a straight line in the narrow space of the test area (see Fig. 4.6) and the robot has to plan a path which crosses this line. In the second scenario, the person meets the robot in a wide corridor (see Fig. 4.7). The person moves also in a straight line and the robot should approach a goal by driving in the opposite direction and also has to plan a path to avoid the person. Both scenarios are based on real world map data of our institute. In both scenarios the personal space is propagated correctly forward in time, while the wavefront passes these regions. So, in both cases the personal space of the moving person slows down the wavefront and guides the wavefront around a region of space *in front of the person*, which is the main difference towards a static path planning . When the goal is reached by the wavefront, gradient descent is used to extract the optimal path.

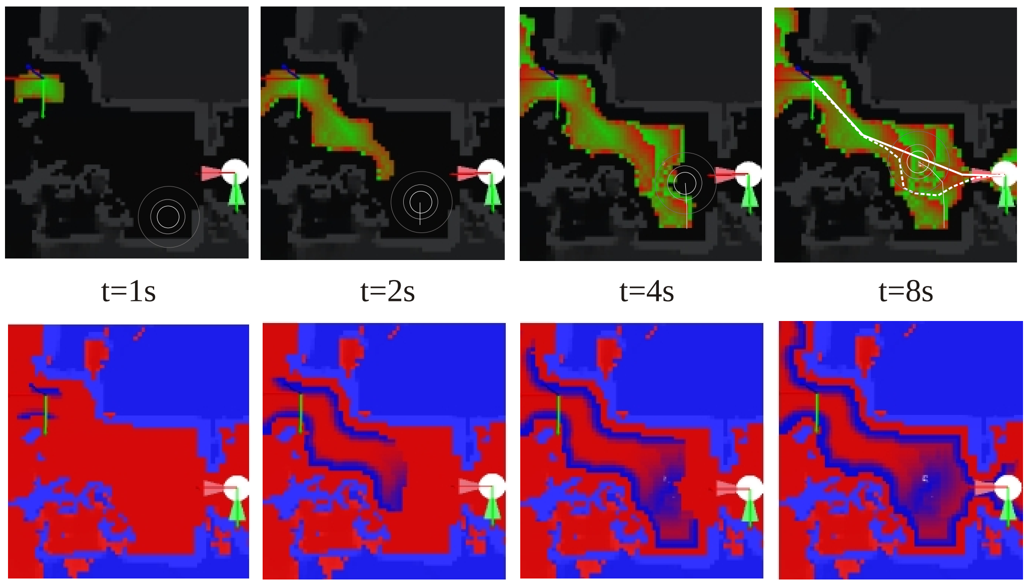

Figure 4.6: propagation of the planning wave in a narrow space. The robot starts on the left side and has to reach the goal on the lower right. The person is located at the bottom (multiple bright circles) and walks through the room. The wavefront travels through the room until the target is reached and avoids the personal space. The final path is shown as a dashed line, whereas the planned path without a person is shown as a solid line.

To provide a practical system, the robot should be able to plan this path much faster than real time. In fact, it must be possible to plan the path in a fraction of a second for multiple seconds beforehand. The planning of five seconds of motion can be done in 350 milliseconds.
The calculation of the force field $\vec{E}_{obs}$ is constant for the given map and is done once before the algorithm starts. It took 10.3 seconds for the given map of the lab building to build the vector field. For the experiments a standard dual core mobile processor with

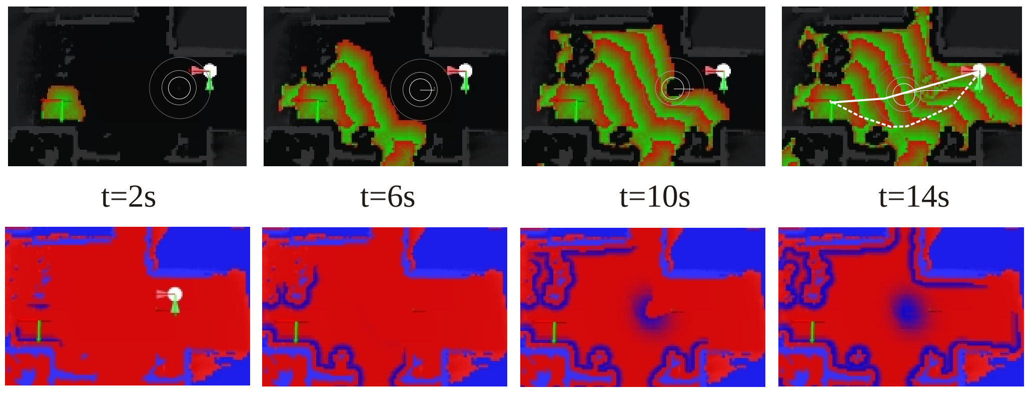

Figure 4.7: propagation of the planning wave on a floor. The robot starts on the left side and has to reach the goal on the upper right. The person is located at the right side (multiple bright circles) and walks through the corridor towards the left. The wavefront travels through the room until the target is reached and avoids the personal space.

2.66 GHz was used.

4.6 Conclusion

At this stage of development, the proposed module is used as the default planner within the ALIAS system. Even if the prediction of the person motion trajectory is not working correctly, this system creates at least the same results as a static planner and improves the motion plans, if the prediction works correctly. The presented planning approach is stable within the ALIAS project, and could be considered as finished. Only the person prediction part needs more fine tuning on the construction of the artificial electrical field to minimize the phenomenon that the simulated person get stuck into a local minimum.

5 Observing a person in an unobtrusive way

Usually, in mobile robotics the robot has to deal with a lot of tasks like interacting with a person, driving to its charging station, or building a map of its environment. But what happens, if the robot just has to wait and thereby still has to react on user commands? During all-day-operation of the ALIAS robot, the robot has to find a good position where the user can still be observed, and the robot does not disturb the user's activities. In this chapter, we present an approach, how to find such a position by solving an optimization problem using a particle swarm optimizer, and we show results for that. This method can be used to "park" the robot at a feasible position by using the position provided by this software module.

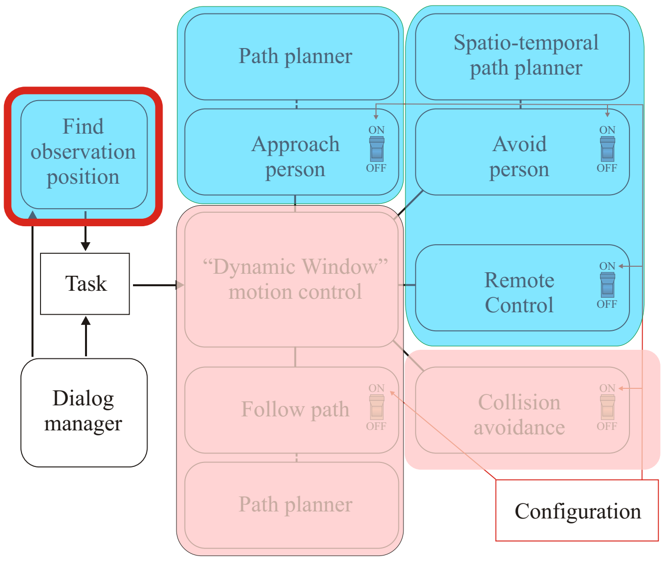

Figure 5.1: the observation software module within the navigation architecture. Note, that this module is separated from the navigation core, since it only emits a sequence of "drive to position" tasks. The module is controlled by tasks, emitted from the dialog manager.

5.1 Integration into the navigation software system

The module for finding a good observation position is separated from the navigation core. It can be triggered by the dialog manager and produces as a result a position, where the robot should drive to. The proposed position is updated continuously until the module is deactivated by another dialog manager command. This structure is shown in figure 5.1.

5.2 Formulation of the optimization problem

The observation position has to consider a variety of criteria. To find an optimal position to observe a person, a set of criteria has to be evaluated at each valid sample of the search space. The search space contains the position $\vec{x} = (x, y, z)$ of the robot and the view direction ϕ. We assume the robot can only move at the ground plane, so the z component is fixed by the robots height. Also, the pitch and roll angle of the camera are fixed, and only the yaw angle ϕ has to be considered. From this three dimensional search space the optimal point is chosen as the best observation position. Since the problem is formulated as an optimization process, we have to consider at the one hand the bounding conditions, and on the other hand the optimization function. Both aspects are described in the following two sections. Additionally, the optimization algorithm is briefly described afterwards.

5.2.1 Boundary conditions

The solution of the optimization process depends on the boundary conditions that exist when the process is started, and may even change during the process. In fact, these conditions reflect the knowledge we have about the respective environment. On the one hand, this is the map $m(\vec{x})$ of the environment, which gives information about known obstacles, and on the other hand it is the position o_t of the person to be observed at a given time t. Additionally, we also include knowledge where the person *usually* sits, stands or lies.

This is done by providing a density function $p(o = \vec{x}_i)$ to give a probability that a person can be observed at a certain point $\vec{x}$ in the home environment. Figure 5.2 shows all boundary conditions summarized. The person occupancy density function is approximated by building a histogram with bin size w over an infinite time interval to collect user observations.

As an update to the first version of the observation module the histogram of the person density is now built only in the 2D space, where the data from the person tracker are collected. The histogram is now estimated online and in real time over a long time period.

Since we operate on a time variant optimization problem, we choose an optimizer suitable for dynamic optimization problems, namely the particle swarm optimization (PSO) technique [4].

5.2.2 The optimization function

The optimization function f reflects the different criteria to be considered and fuses these criteria into a single function. It is a function over the optimization space $S = \{\vec{x}, \phi\}$, where $\vec{x} \in \Re^2, \phi \in \Re$. There are two hard criteria to reflect physical properties to constrain the search space and mask out impossible search positions. These are the driveability $d(\vec{x})$, and visibility of the person $v(\vec{x}, \phi)$. Both functions d and v are binary functions.

Moreover, a set of soft criteria c_i has to ensure: (i) an appropriate distance to the user (c_{dist}), (ii) the ability of the sensor to detect a person (c_{det}), (iii) to perceive the person

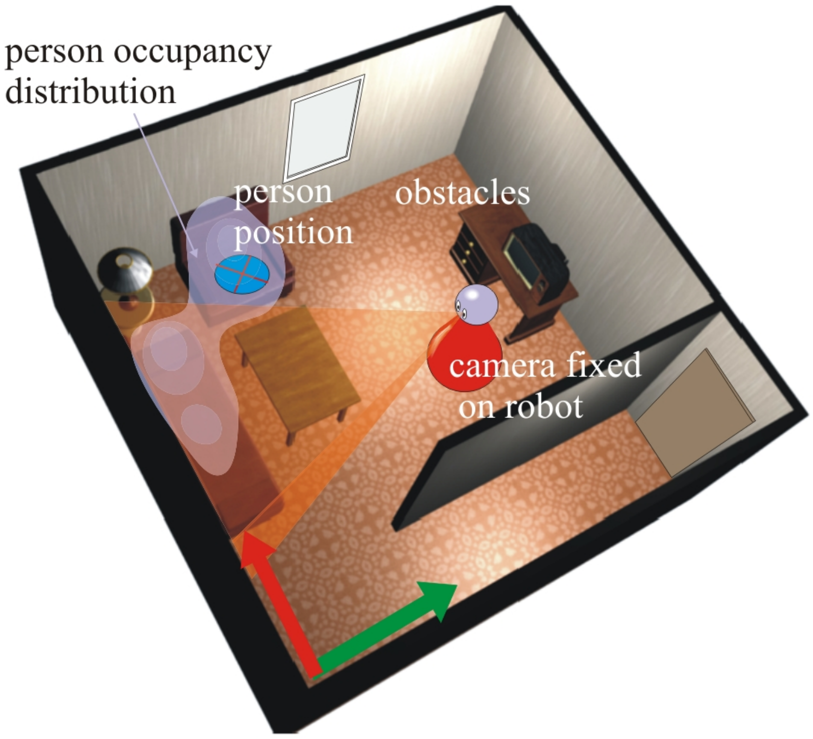

Figure 5.2: the boundary constraints of the optimization problem: the obstacles within the environment, the current person position o_t, the person occupancy distribution $p(o = \vec{x})$ and the position of the camera on the robot.

from the front (c_{front}), and (iv) how many places of the person's occupancy distribution are observable (c_{podf}). An example of all functions is shown in figure 5.3. Since these criteria are no hard criteria, an optimal compromise between them has to be found. These criteria are fused by the superposition principle. So the resulting optimization function is defined as follows:

$$\begin{aligned} f(\vec{x}, \phi) &= d(\vec{x}) \cdot v(\vec{x}) \cdot [\alpha_1 \cdot c_{det}(\vec{x}, \phi) + \alpha_2 \cdot c_{dist}(\vec{x}) \\ &\quad + \alpha_3 \cdot c_{front}(\vec{x}) + \alpha_4 \cdot c_{podf}(\vec{x}, \phi)] \end{aligned} \tag{5.1}$$

Most criteria are simple, and at this point we will take a closer look only to c_{podf}, the criterion of the person's occupancy distribution. Since positions are already masked out, where the person could physically not be seen (using $v(\vec{x}) \cdot d(\vec{x})$), it is possible to observe the person from the subset of all remaining positions $X_v = \vec{x}_1 ... \vec{x}_n$. Now the question is: how many places where the person *usually* is, are observable by each possible observation pose $\vec{x}_i$? A view cone X_f is cast by the camera into the home environment, depending only on the rotation angle of the robot and the position inside X_v. The idea of c_{podf} is now

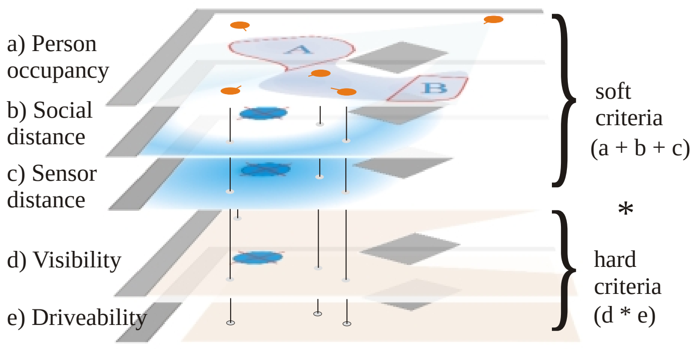

Figure 5.3: hard and soft criteria. The hard criteria mask out the possible search space, and particles (shown as orange dots) are only placed in the remaining region. The soft criteria determine the optimum and are summed up to form the optimization function. Note, that the soft criterion of view direction is not shown here.

to integrate all observable points $\vec{x}$ from $p(o)$, where $\vec{x} \in X_f$ and where $p(o) > 0$:

$$c_{podf} = \int_{\vec{x}} p(o = \vec{x}) \;, where\; \vec{x} \in X_f \tag{5.2}$$

This function should guarantee that the robot places itself at a position where most of the places the person usually rests at, are observed.

5.2.3 Particle swarm optimization

The optimization problem is simply to find the values of $(\vec{x}, \phi)$ that maximizes the output of $f(\vec{x}, \phi)$. Our solution to the defined optimization problem uses the particle swarm optimization (PSO) approach. It is a well known technique (see [4], [3]) to find a global optimum by sampling from a defined optimization function, and uses a mixture of directed and random search within the search space to iterate towards the optimum.

Each particle contains a state, which is part of the current optimization space, and a speed vector also placed within that space. In our case one particle contains a position $(x, y) \in X_v$ and a view direction ϕ. Note, that we only optimize over (x, y) and chose ϕ to view directly towards the current person position o_t. Thats why we only need speed components of the particles in x and y direction, namely v_x and v_y. Each particle is defined by $p^{[i]} = \{\vec{x}^{[i]} = (x, y), \phi^{[i]}, \vec{v}^{[i]} = (v_x, v_y)\}$. In the first step, particles are randomly initialized only in those cells, where all hard criteria are fulfilled. The key idea behind the particle swarm optimization is, that particles tend to search near positions where good results for the optimization function are already measured. The speed component hereby enables the particles to overcome local minima and circle around good positions.

5.3 Optimization Criteria

In this section, we will describe in detail all functions which are part of the optimization, and we also show the representation of the environment.

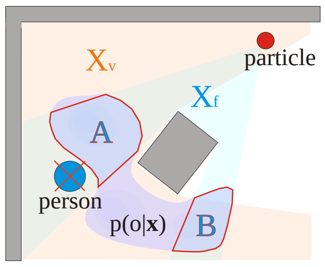

Figure 5.4: in red: the set X_v of all positions where the person is visible. From all other positions the person is covered by an obstacle. Particles only exist in X_v. In blue: the view cone X_f which a particle can observe. Note that X_f defines the area where $p(o = \vec{x})$ is integrated: in this case A+B.

5.3.1 Data structures

Since our early experiments show that the 3D case is very slow to compute we focus on modeling our environment in 2D. The used occupancy map is created by a separate mapping process, sketched in chapter 2. The person occupancy probability distribution is a simple histogram, where each cell counts the number of points belonging to a person, which are normalized by all observed person points. These person points are collected from the output of the person tracker.

5.3.2 Realization of the single criteria

Driveability

The first criterion we discuss is $d(\vec{x})$. Here, cells are selected which could be reached by the robots camera. This function is either zero, when the cell is not reachable, or one when this cell is reachable. The map is dilated by the robot radius, to assume a point-like robot. Then we use simple flood filling to get the set of reachable cells.

Visibility

Our next function is the visibility criterion $v(\vec{x})$. Although this function is independent from $d(\vec{x})$, it makes sense to only consider points which are inside X_v, since $d(\vec{x})$ and $v(\vec{x})$ are multiplied. So the task is to check every cell from the previous step, if the person could be seen from that position. This is done by ray-casting from the current cell towards the person position. Here, the set of cells inside X_v is reduced. With both functions known for a given map and a given person position, the particle swarm could be initialized with regard to X_v.

Sensor distance

The next function is the sensor distance $c_{det}(\vec{x})$. Since we use the Kinect sensor, the recognition distance is limited to 3 meters. So, we use the sensor distance $d_s = |\vec{x}_i - o_t|$, which is the distance from the current cell $\vec{x}_i$ towards the center of the person position, using the parameter s_{max}, which refers to the maximal distance the sensor could observe:

$$c_{det}(\vec{x}) = \begin{cases} 1 & , if\ d_s < s_{max} - 1 \\ \frac{1}{1+exp(d_s - s_{max} - 0.5)} & , else \end{cases} \tag{5.3}$$

Social distance

The social component is defined in a very similar way. As Hall [9] explains, the social distance, where persons do not consider to interact with each other, is around 2.5 meters and above. This is our social distance to make an observed person feel comfortable. The function to consider this fact is defined as follows, using the parameter $\sigma_d = 0.5m$:

$$c_{dist}(\vec{x}) = e^{-\frac{(d_s - 2.5)^2}{2\sigma_d^2}} \tag{5.4}$$

Frontal view

For gesture recognition, face identification and emotion recognition, it is necessary to observe the user from the front. Thats why we define an angle interval where a good viewing angle from the front could be guaranteed. The deviation from the person's view direction towards the robot's pose is defined as angle β. With that angle, c_{front} could be defined as follows:

$$c_{front}(\vec{x}) = \begin{cases} 0 & , if\ |\beta| > \pi/2 \\ \frac{1}{1+exp(|\beta| - \pi/6)} & , else \end{cases} \tag{5.5}$$

Person occupancy distribution

As described in section 5.2, the function c_{podf} describes, which part of the person occupancy density function can be seen from the given hypothetical robot position into the given direction (see Fig. 5.4, blue area). This is a time consuming operation, since a visibility check of the person occupancy density function has to be calculated.

Updated: When we proposed the optimization technique for the first time, we kept the person occupancy density function static and measured this function before the actual optimization started. In the updated version, we estimate the density function online. Also, we first assumed, that each occupied cell has an infinite height, where the robot is not able to view over obstacles. This is off course not true, since tables, chairs and couches could be easily overlooked. To improve that particular point, we now estimate an elevation map with the help of the 3D Kinect camera (see an example in Fig. 5.5). With the elevation map, we could improve the results of the 2D visibility check very close to the results of the full 3D model, but with the difference, that the 3D model is computational far too expensive and the 2D model could be calculated in only a few seconds.

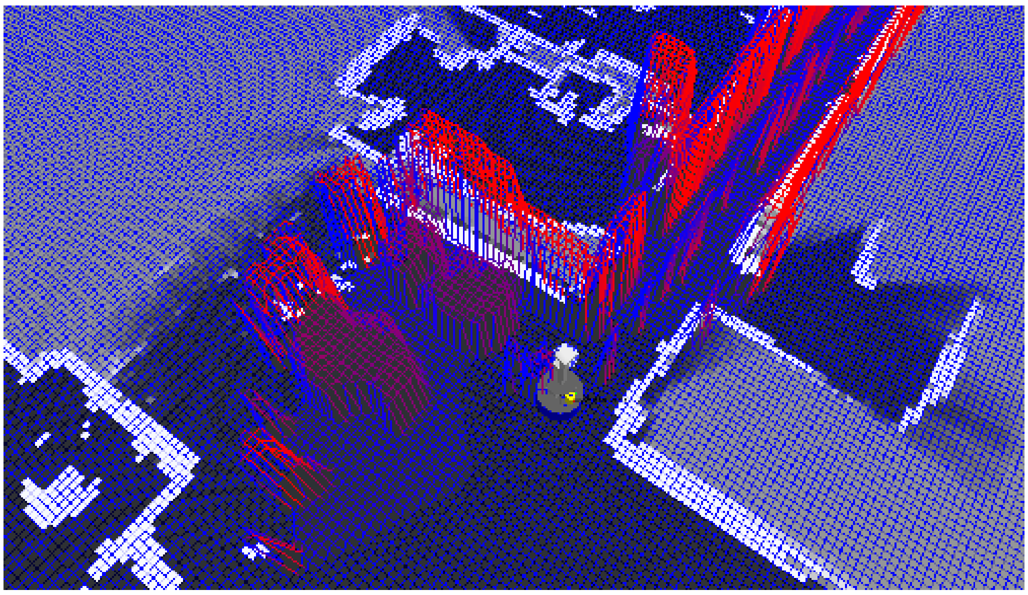

Figure 5.5: Part of the elevation map of our lab. Here, some tables with chairs are integrated into the map. Red means higher elevation above ground while blue means lower elevation values. We use this knowledge to check, if a person could be observed from a certain point.

5.4 Experiments

In this section, we show experiments done for the 2D case. The experiments for the 2D case where executed by using the 2D occupancy map of our lab (see Fig. 5.6 for the hard criteria masks and the person distribution). Our experiments where focused on the stability and the speed of our approach. The found solution produces reliable results by using 30 particles and 40 iterations in typical home environments and large corridors.

With a variance of 14.5 cm, the found results are sufficient for the real world scenarios. Here, a 2.16 GHz Intel dual core processor is used, running directly on the mobile robot.

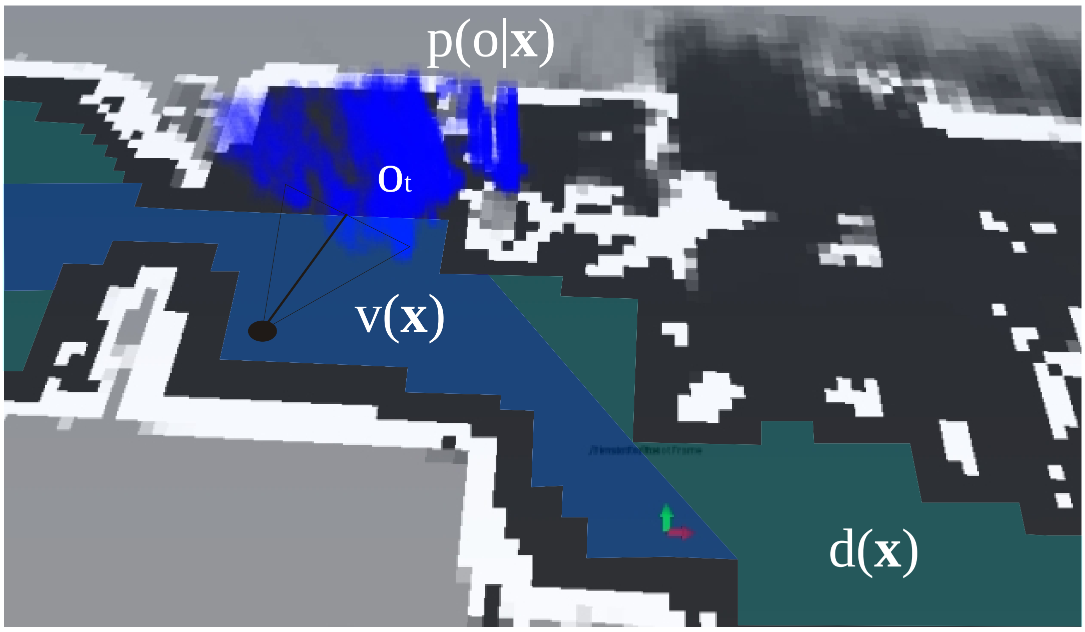

Figure 5.6: 2D map of our lab environment with the person occupancy density function $p(o|\vec{x})$ as the original voxel representation, the drivable space from $d(\vec{x})$, and the visible space $v(\vec{x})$ where the person o_t could be observed. Note that o_t is not visible here, since it is covered by the blue voxel elements from $p(o|\vec{x})$. Also, the final observation position is shown.

The results show calculation times of 5 - 10 seconds, depending on the number of used cores.

5.5 Conclusion

In this chapter, we have shown an approach, how to observe a person by also considering positions, where the person usually sits. Therefore, we provide a person occupancy distribution and use a variety of other criteria. We also plan to include additional hard and soft criteria towards the optimization problem. One planned hard criterion is, to keep the line of view from the observed user towards "objects of interests" (like television, clocks or fish tanks) free from the robot presence. These points of interest should be labeled by hand into the robot knowledge. Another possible soft criterion is, that the robot should not place itself on paths, the person usually walks, which means that we simply have to use the person occupancy density function for that purpose.

6 Robot Remote Control

6.1 Introduction

In this chapter we will describe our approach to make the robot remote controllable. Note, that this is a very simple approach and so we will not discuss, what other approaches exist, since this discussion will not have significant value towards the reader. The need of this module emerged during the user trials and questionnaires, where the users demanded the need of some kind of security feature. An exact description, what this feeling for security means, was not given. So, the consortium decided to enable the robot to be remote controllable and to send video images towards the controller. It is clear, that only certified and authorized persons should be able to see those video images. We decided to use Skype to transfer one video channel and the audio channel of a camera mounted on the touch screen and transfer an additional video channel from the front camera, to enable the remote-user to see where to drive to. These configuration was voted best during the second user trials.

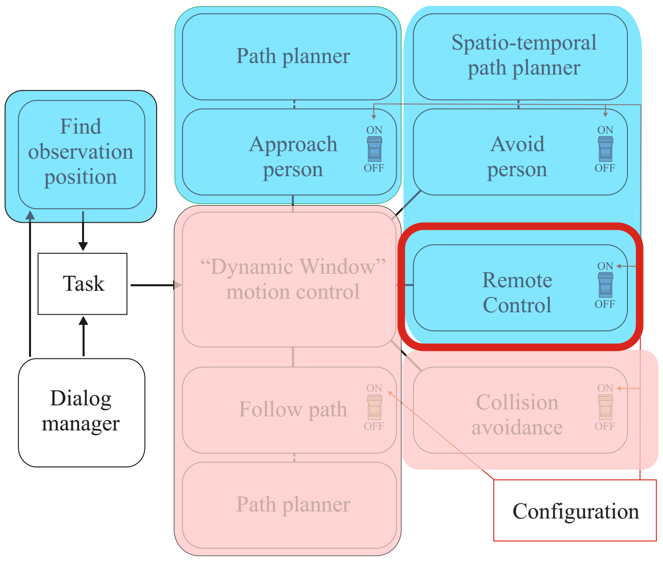

Figure 6.1: the "remote control" software module within the navigation architecture. This module is directly linked to the navigation core.

The main task of this module is, to make the robot steerable by a remote person in case of emergency. *How* a case of emergency is exactly defined, lies within the domain of the user inclusion work group of work package one. Here, only the functionality is defined and implemented. The core ability should be, to control the robot to drive in the defined direction without collision, and give some obstacle feedback to the remote controlling person.

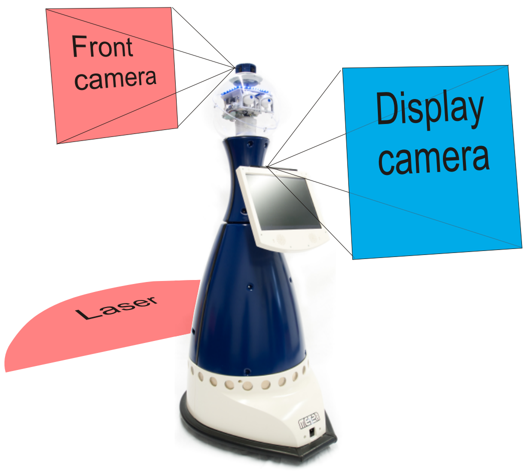

Figure 6.2: The sensor information, transfered towards the remote controlling person. In blue the Skype application is used to transfer camera images and sound. Red are the additional channels for the controlling person to be able to localize itself within the home and also to get an idea of the obstacle situation in the flat. Therefore, the laser data and the front camera image are transfered.

6.2 Remote Architecture

Since the mathematical background of this task is quite simple, the main challenge here is the transfer of control data from the remote control client towards the server (on the robot) and in the other direction the sensor feedback from the obstacle situation from the server to the client. See figure 6.3 for the definition of the client and server.

We use the AngleScript interface from MetraLabs to exchange the command data and the laser data between client and server. The skype API is used to transfer audio and video data from client to server. In the next sections, the details of data flow, and the mathematical realization of the remote control, are shown.

6.2.1 Remote Client

The remote back end is the client side of the remote control and is designed to work on every standard PC with network access. At the moment, only the robot or the robot's user is allowed to establish a connection to a remote controller. The remote controller can call the robot's user but is *not* allowed to take control over the robot without approval. A Wii controller is connected to the client PC via Bluetooth and the steering cross buttons or the nunchuck joystick could be used to control the robot. From the remote controller, pairs of rotation speed and translation speed $\langle V_{rot}, V_{trans} \rangle$ are send frequently. These speeds are evaluated on the server side, if they are feasible or if it is necessary to deviate from the

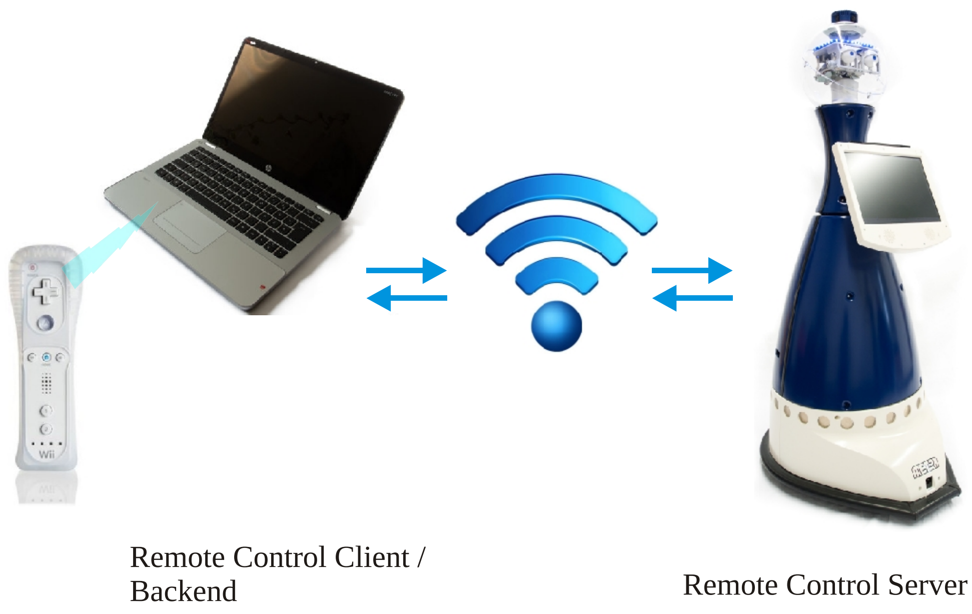

Figure 6.3: the network configuration during the remote control process. The robot is steerable with the Wii remote controller, which is connected via Bluetooth to the remote client. The client uses a standard TCP/IP connection to the remote server (the robot). Over this connection the wished user speed is transferred from the client to the robot and laser distance data plus video and audio data are transferred from the robot to the client.

wished speed configuration to avoid collisions. The deviation is done by the robot in an autonomous fashion. So, even when the connection is lost, the robot is able to generate safe driving commands.

The client side is responsible to open the TCP/IP connection towards the robot, to connect to the Wii controller, and to display the front camera image (without sound), the back camera image as a Skype connection, and the laser scan to visualize the current obstacle situation if the robot or the user gives permission to the remote side (see Fig. 6.2 for the sensor configuration on the robot). It also sends new driving speed commands to the robot. These commands are evaluated on the server side.

6.2.2 Remote Control Objective

On the server side, the send speed commands $\langle V_{rot}, V_{trans} \rangle$ are received and evaluated. This is done by a remote control objective, which has to be activated from the dialog manager when an emergency situation occurs. The activation of the remote control is by

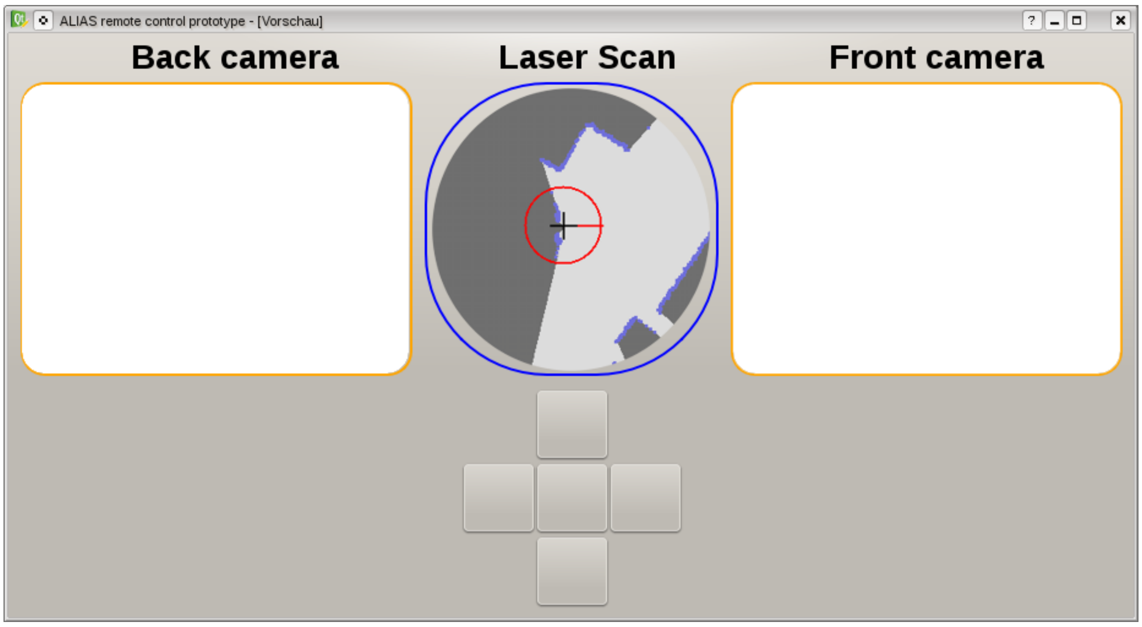

Figure 6.4: The remote back end. The robot is controlled by the Wii controller or the steering cross at the bottom. The commands are send to the robot and the current laser scan, the front camera image, and the back camra image are sent back and visualized on the screen. Since the robot has a diameter of 0.6m, the user can estimate the distance towards the next obstacle. The back camera is actually a Skype client, so the remote controller is able to talk to a person, seen on the back camera. Note, that the shown interface is only a mock-up. Work is in progress to get the Skype component into the interface.

default *not* coupled to a video call, but both functionalities are at the moment combined in the remote control case. Since the remote control objective is just one part of the whole navigation setup, also the obstacle avoidance is activated during the remote control process and those, the aspect of obstacle avoidance has not to be considered within this objective. The only task is, to support the defined speed pair, given by the user. We assume, that it is no strict command the remote controller sets, when steering the robot with the controller. It is interpreted as an approximate wish, where the robot should drive to. All speed combinations nearby the given speed pair have to be supported also by the system and should be treated as an alternative.

Luckily, the dynamic window approach operates in the speed domain. A rasterized set of speed pairs $\left\langle V_{rot}^{(i)}, V_{trans}^{(i)} \right\rangle$ is evaluated in each step and the best action is selected. Hereby, the collision avoidance marks all actions, which lead to a collision in the next step, as *not feasible* (see Fig. 6.5). From the remaining actions, the best is chosen. We vote for each of the speed pairs by using a Gaussian function centered at the given user

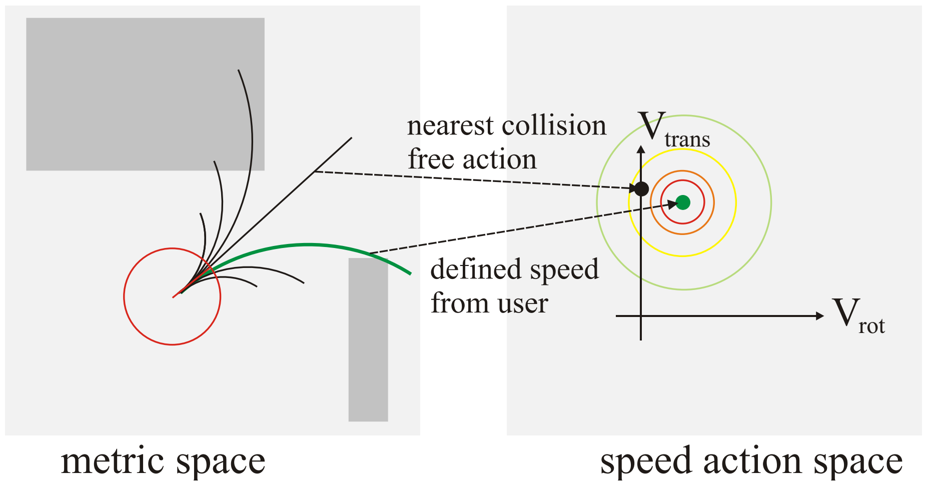

Figure 6.5: the voting process within the dynamic window: in each step, the dynamic window evaluates a set of speed actions which in term lead to a specific trajectory. The user defines one speed pair, which lead to the wanted trajectory (green on the left side). Our objective votes this speed pair best. But other, nearby speed pairs will also given a good vote, since they reflect also the "will" of the remote controller. This is done by a Gaussian function. So, if the wished speed is not possible (due to a collision), the next best, collision free action is chosen automatically by the dynamic window approach.

speed $\langle V^u_{rot}, V^u_{trans} \rangle$:

$$f(V^{(i)}_{trans}, V^{(i)}_{rot}) = e^{\frac{(V^{(i)}_{rot} - V^u_{rot})^2 + (V^{(i)}_{trans} - V^u_{trans})^2}{\sigma^2}} \tag{6.1}$$

Here, only σ is a free parameter to define the range of equally good speed alternatives with a high vote, and so, how much room for alternative decisions is left for the robot. Note again, that the remote control objective does not check for collisions. This is done by another objective, which is re-used for our purpose.

7 Autonomous map building

One of the challenges the team of ALIAS encountered during user trials or on fairs was the problem of enabling the robot to move, when no "navigation expert" is at hand. As stated in the introduction, this is mainly due to the fact that the mapping of the operation environment is done manually. In addition, this process needs detailed knowledge of the mapping tool chain. That raises the question, how we could possibly overcome the drawback of requiring expert knowledge on setting up the robot.

In the second half year of 2012 we started to build a prototype of an automatic mapping process, where the robot should explore its environment autonomously and returns a map when exploration is finished.

Since this process is only required upon setup time, it is not involved in the classical ALIAS application. That is why we decided to build a stand alone application to get some first impressions of the complexity of this approach. In this chapter, we provide an overview of the different stages of the mapping operation and show the details, where the most attention was given to, the process of finding the next position to observe.

Note, that the current work is in an early stage and only has prototype character. Since it was not foreseen in the navigation part we use all the time we could spare into this task to show what could be possible.

7.1 Application Architecture

The application is mainly a very simple state machine (see Fig. 7.1), that includes some basic parts of the navigation system, that allows the robot to drive to defined positions. The state machine synchronizes the states by listening to events, coming from the navigation process. For example, when a position is reached or a position could not be reached. For that synchronization we use the same mechanisms as the dialog manager.

The robot resides within one of two states. Either, it drives to a new exploration position (and turns around itself one time when the position is reached) or the robot calculates the position it should drive to next. Note, that we do not terminate this cycle automatically at the moment, but we trigger finishing manually. What we use from our standard navigation setup is: the dynamic window to control the robot motion, the path planner to get to a goal, and the interface to enable the robot to drive autonomously. We use also the standard localization module when driving towards a new exploration point.

What has to be adapted from existing code, is the SLAM module, which simultaneously

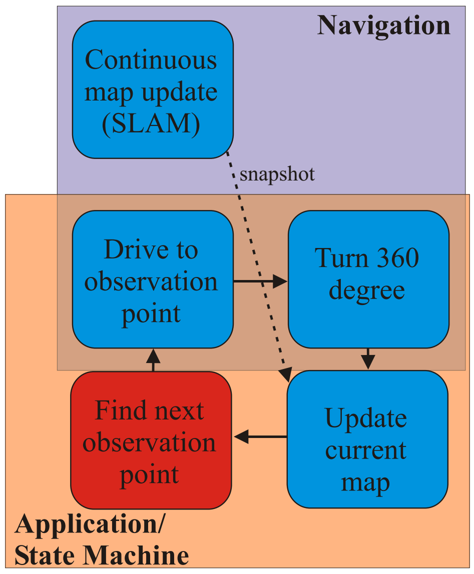
Navigation
Continuous map update (SLAM)
snapshot
Drive to observation point
Turn 360 degree
Find next observation point
Update current map
Application/ State Machine

Figure 7.1: The coarse structure of our application: while most parts are directly re-used from the navigation, the part of finding the next best exploration point is new. All parts are coordinated by a state machine, which actually is the exploration application.

localizes the robot and updates the map. As stated in the introduction, the SLAM approach from Grisetti [7] is used here. The localization and map update is running constantly in the background.

Additionally, the robot needs to find an optimal exploration position by knowing the actual estimated map and the current position within that map. A feasible method is described in the next section.

7.2 Find the next observation pose

The input for this module is the currently estimated map and the robot position. In this occupancy map, obstacle values have high value (appear white in the Fig. 7.2) and known free space has low values (appears black). Grey values are unknown cells, which have never been observed. The goal of a good exploration position is

- the position has to be as near as possible
- from the position, a lot of unknown space should be visible
- the robot should still be able to localize itself at the goal position (means, the robot should see enough distinct structure like corners etc.)

To fulfill the first condition, we simple chose the nearest point from a set of possible candidates. But how to get to that set of candidates? First, we search for edges of free space, which directly adjoin to unknown territory and are reachable by the robot. This reduces the number of candidates greatly (see figure 7.2 below). Afterwards, we look for distinct features (corners) in the image, to enable the robot still to be able to localize itself. This is also important for the SLAM process, since in such places "bad" trajectories could be distinguished more easily from "good" trajectories. But we also want to observe unknown places!

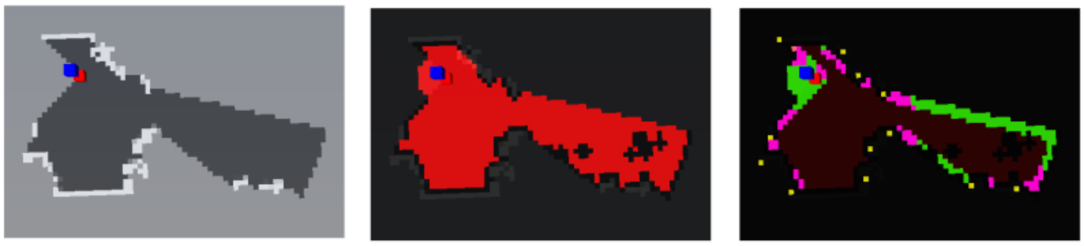
Figure 7.2: The sequence of extracting the next exploration candidate, shown on a small map. On the left the current map is shown (with the robot as a blue and red box). In the middle the reachable cells are shown in red, while on the right side the distinct features are shown in yellow, the edge cells are shown in green, and possible candidates are shown as purple pixels.

In the last step we investigate a circular region around each possible candidate. Within that region we count the distinct features f and also the amount of unknown space u in percent of investigated cells. Than we apply a simple ranking scheme, where cells with a higher f-count dominate another cell, and cells with an equal f-count dominate other cells, when u is greater. So, the set is reduced significantly, since only the best cells of this ranking process are considered. And finally, if two cells have an equal ranking, we choose the nearest one.

This leads to robust results, but sometimes cells are selected, where a lot of corners are present and the goal to reach is in fact one cell only. At this point, we have to improve the filtering process, that only positions are observed, where a lot of equally good points are near. So, single points with no neighbor candidates should be avoided, which makes driving to that point more easy.

7.3 Map Building

Map building is done by a SLAM approach [7]. In this approach a particle filter is used to estimate a variety of possible paths the robot could have driven. For each position of this path, also the sensor information (in this case a laser scan) is stored and forms a hypothetical map. When a path is correct and the robot reaches known territory, the perceived sensor information matches to what the robot expects to read from the sensor. In such a situation wrong trajectories could be sorted out and only the correct trajectories remain. If the particle count is not to large, the matching between expected sensor impression and real sensor impression could be calculated in real time.

7.4 Conclusion

Currently, we could demonstrate that the robot starts to drive to several observation positions. The rule-set, which chooses possible next candidates, has to be improved since very often undesired positions are chosen. These positions could not be approached robustly and the state machine gets stuck. Also, a lot of fine tuning has to be done, regarding the switching of the map during the navigation process. So, re-initializing the localization at each time step is currently a problem, as well as keeping a correct robot localization during navigating with a (very) sub-optimal map.

8 Person tracker

This chapter should shortly give the latest update of the work on the person tracker. This module is a crucial module for the whole project. It enables the robot to perceive persons, which is required for social acceptable navigation and the face recognition modules, as well as for the dialog system. During the project, it turned out very soon, that the existing methods are not sufficient (based only on face detection) and the person tracker has to be improved. Here, we show the latest version of the tracker.

8.1 Detection channels

The tracker has several detection channels, based on the laser scanner, the Kinect camera and the four cameras, mounted on the top of the robot's head. We do not rely on detecting a speaker direction by using microphone arrays.

Laser channel

The laser channel does the leg-pair detection, which splits the laser scan into consecutive segments and for each segment a 10-dimensional feature vector is calculated. These feature include length of the segment, circularity, linearity, mean value, and some more (see [1] for more details). Then, an decision tree is trained to separate leg segments from non-leg segments. This channel gives the most reliable results, but false positive leg segments could still be detected. But even for humans these false positive leg segments are indistinguishable from real leg segments.

3D camera channel (updated)

The Kinect 3D camera creates a point cloud from the current scene. This point cloud is at the first step segmented into different segments (see figure 8.1 top left). In first prototypes we used the OpenNI framework [18] to segment the user points from the background points, but this requires the Kinect to run on 30 frames per second and uses almost all of the robot's processing power. And unfortunately, while integrating the Kinect into the whole system, we found out that this device completely blocks the camera system of our four head cameras. For these reasons we no longer rely on the OpenNI framework. We tweaked the Kinect linux driver to run on a much lower frame rate, which is now 2 frames per second, and also replaced the segmentation by an algorithm from the famous Point Cloud Library [17].
Unfortunately, we do not have any information, which segment is a human, and which not. That is why we do not want to detect humans in the segmented point cloud, but

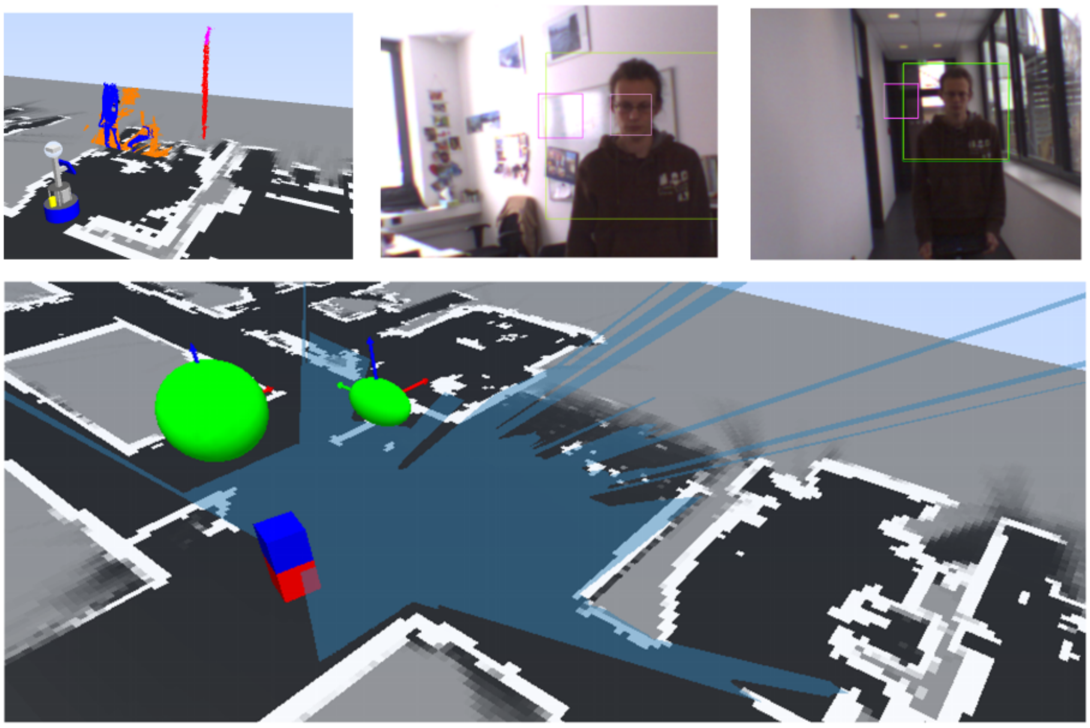

Figure 8.1: The tracker channels. Top left: the segmented point cloud with (blue) person pixels. Top center: two examples of the HOG upper body detection (dark green boxes) and Face detection (pink boxes). Note, that false positives are often detected with face detection. Bottom: one laser based hypothesis and a hypothesis from the HOG channel. Both are fused into one set of hypotheses.

only upper body directions. To do this, we use the other detection channels to give an hypothesis, where a human is. With this hypothesis we can search the nearest segment and extract an characteristic point segment. This segment is than used to calculate a principal component analysis, which gives an optimal 3D coordinate system of the segment. The axis with the lowest variance (or eigenvalue) is our gaze direction. The channel with the highest variance is the z-direction, assuming the person to sit or stand. The channel with the second highest variance is the body width, since the width of a human is greater than the depth.

Omnivision channel (updated)

This channel handles the four images of the head cameras. Here, most work is involved to get a visual feedback, where in the image a person is. We use 2 main detectors, namely the face detection approach from [16] and an upper body detector, which uses histograms of oriented gradients [2] (in short HoG). This detector requires very large images to detect persons, which are up to 4 meters away from the robot. This makes the detector very slow, but also more reliable than the face detector.

The face detector very often finds false positive faces (positions in the image, where actually no face is, but the detector belief so). The HoG is much better here, but requires a lot more processing power. To keep the robot operable, we decided to detect persons in the image channel only every 2 seconds. This allows all other (also time consuming) operations for navigation to get an calculation time slot.

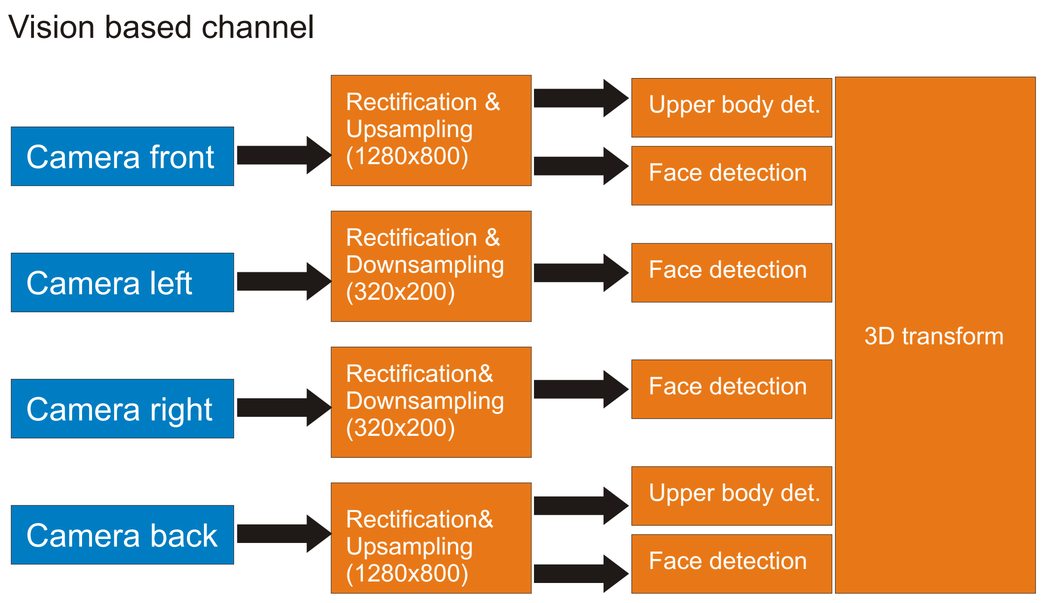

Figure 8.2: The vision detection channel. We use four separate images, pointing to the left, right, front, and back. All channels use face detection to detect a user. Only the front and back channel use also upper body detection (with HoGs) which requires significantly larger images. The upper body detection is also computational complex, why we could only calculate every 2 seconds the upper body detection.

Note, that all detection channels are image based. Afterwards, all detected hypotheses are transformed separately into the 3D domain and are presented to the robot in world coordinates. An overview is given in figure 8.2.

8.2 Hypotheses fusion

The hypotheses from laser channel and image channel needs to be fused together. Here, nearby hypotheses should strengthen each other to build one final, more robust hypothesis. Hypotheses, which are more distant should be left untouched. Also, if in consecutive time frames a hypothesis is found multiple times, this should strengthen the hypothesis.

To track the person over time, and strengthen the belief for that person, we use a Kalman filter. So, consecutive detections can now keep the covariance matrix small (as a measure of "belief"), while the uncertainty will grow, if detections will not appear (due to the influence of an artificial motion model). This is a standard approach. But the hypotheses we present to the Kalman filter are a result of a previously merging step of single hypotheses. The approach is simple: if one hypothesis is near in time and space to another hypothesis (already covered by a filter), than it is merged with the current filter hypothesis by covariance intersection (where the smallest values of the covariance matrix are used). If a hypothesis could not be matched to an existing one, a new Kalman filter is constructed with a low certainty.

8.3 Conclusion

In the current state, the person tracker is usable for user trials. At the one hand, we where able to get more robust results but need on the other hand high computational power. Also, the problem of false positive detections is not solved, as well as missing detections. To build a market ready product, this point has to be investigated very intensely.

9 Conclusions

In this deliverable, we presented the latest versions of our software modules for approaching a person, avoiding a person, observing a person, and remote controlling the robot. Also, the challenges of autonomous map building and person detection are shown.

At this point we summarize the achievements and remaining problems of each of these points:

- **Approaching a person**
  achieved: stable running approach
  remain: improve person tracker robustness
- **Avoiding a person**
  achieved: stable running planner, average person trajectory prediction
  remain: improve person trajectory prediction parameters
- **Observing a person**
  achieved: stable running approach
  remain: improve person tracker robustness, add more criteria (stay at wall and not in walking path, do not block line of sight to television)
- **Remote control**
  achieved: stable running approach
  remain: improve graphical user interface
- **Automatic map building**
  achieved: very early prototype
  remain: extensive testing to find problems, test different SLAM methods, many minor problems ... (a large field of work)
- **Person tracking**
  achieved: able to detect persons with camera and laser
  remain: robust, fast camera-based detectors

During the development of this project we found the aspects of remote control, person tracking, and autonomous map building as additional aspects, which are either necessary or nice-to-have. Not all additional aspects could be solved during the remaining time of the project. We focus mainly on what we have promised in the description of work, which includes the minor issues on avoiding a person, observing a person, and remote controlling the robot. We will also try to implement a demonstrable version of the automatic map building application, but this will still be a prototype with many smaller problems. The person tracker will be left as it is, since we where able to use it for all demonstrations we intended to show during the user trials. But for a final product especially this point has to be improved and the focus should be on it for future projects. A last update of the remaining points will come in the final report of the project.

Bibliography

[1] K. Arras, O. Mozos, and W. Burgard. Using boosted features for the detection of people in 2d range data. In *Proc. ICRA*, pages 3402–3407, 2007.

[2] N. Dalal and B. Triggs. Histogram of oriented gradients for human detection. In *Proc. on Computer Vision and Pattern Recognition*, volume 1, pages 886–893, 2005.

[3] R. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proc. of the Congress on Evolutionary Computation*, volume 1, pages 84–88, 2000.

[4] R. Eberhart and Y. Shi. Particle swarm optimization: developments, applications and resources. In *Proceedings of the 2001 Congress on Evolutionary Computation*, volume 1, pages 81–86, 2001.

[5] E. Einhorn and T. Langner. Pilot-modular robot navigation for real-world-applications. In *Proc. 55th International Scintific Colloquium*, pages 382–393, 2010.

[6] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. In *IEEE Robotics and Automation Magazine*, pages 23–33, 1997.

[7] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, 2006.

[8] E. Hall. *The hidden dimension*. Doubleday, NY, 1966.

[9] E. Hall. Proxemics. *Current Anthropology*, 9(2):83++, 1968.

[10] F. Hoeller, D. Schulz, M. Mark Moors, and F. Schneider. Accompanying persons with a mobile robot using motion prediction and probabilistic roadmaps. In *Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, page 1260Ű1265, 2007.

[11] J. Kessler. D6.3 first navigation software module, 2011.

[12] J. Kessler. D6.4 update navigation software module, 2012.

[13] T. Laue and T. Röfer. A behavior architecture for autonomous mobile robots based on potential fields. In *In 8th International. Workshop on RoboCup 2004 (Robot World Cup Soccer Games and Conferences), Lecture Notes in Artificial Intelligence, Lecture Notes in Computer Science*, pages 122–133. Springer, 2004.

[14] J. Sethian. A fast marching level set method for monotonically advancing fronts. *Proc. Nat. Acad. Sci.*, 93(4):1591–1595, 1996.

[15] M. Svenstrup, S. Tranberg, H. Andersen, and T. Bak. Pose estimation and adaptive robot behaviour for human-robot interaction. In *Proc. ICRA*, pages 3571–3576, 2009.

[16] P. Viola and M. Jones. Robust real-time object detection. In *2nd int. workshop on Statistical and Computational Theories of Vision - Modeling,Learning,Computing,and Sampling*, pages 1–25, 2001.

[17] Point cloud library, 2013. http://pointclouds.org/.

[18] Openni framework, 2013. http://openni.org/Documentation/.