AAL-2009-2-049, ALIAS
D6.4
Update navigation software module

Due Date of Deliverable	2012-06-30
Actual Submission Date	2012-06-30
Workpackage:	6.4
Dissemination Level:	Public
Nature:	Report
Approval Status:	Final
Version:	v0.2
Total Number of Pages:	43
Filename:	D6.4-IUT-NAV-v0.1.pdf
Keyword list:	navigation module, person observation, spatio temporal planning

Abstract
This report describes the updates of the navigation system, developed within the ALIAS project within the year 2011. Also, a brief overview of all navigation modules is given.

History

Version	Date	Reason	RevisedBy
0.1	2012-05-21	created [IUT]	Jens Kessler
0.1	2012-06-18	first review [TUM]	Tobias Rehrl
0.2	2012-06-20	included remarks of first review	Jens Kessler
0.2	2012-06-26	second review [MLAB]	Christian Martin
1.0	2012-07-03	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.

While the preceding deliverables mainly describe methods, how to actively approach a person to support the dialog, this deliverable describes more the passive behavior of the robot, when the robot and the person do not interact with each other. Here, the background behavior of the robot is changed, so that no other modules of other partners have to deal with the improvements described in this deliverable. In detail, a passive observation behavior will be described, and also an approach, how to drive around a person, when no interaction is wanted, is presented. Additionally, the system to remotely control the robot is described, and an overview of all navigation system components is given.

This deliverable is structured as follows: in chapter 2 a short overview of the current navigation system is given, how all modules interact together, and where the newly presented modules are put in place. In chapter 3 our approach on finding a good observation position is shown, while chapter 4 describes the behavior of actively avoiding a person. The chapter 5 describes briefly the mechanism of remote controlling the robot and in chapter 6, the whole navigation system is summarized and an outlook of our future work is given.

2 System Overview

This chapter presents a brief 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" [11], which was extended by MetraLabs to allow a more modularized architecture (see **??**). Here, a set of possible next motion commands is evaluated by voting for each command by a set of so called objectives. These objectives could be switched on or off, depending on the currently given task. 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.

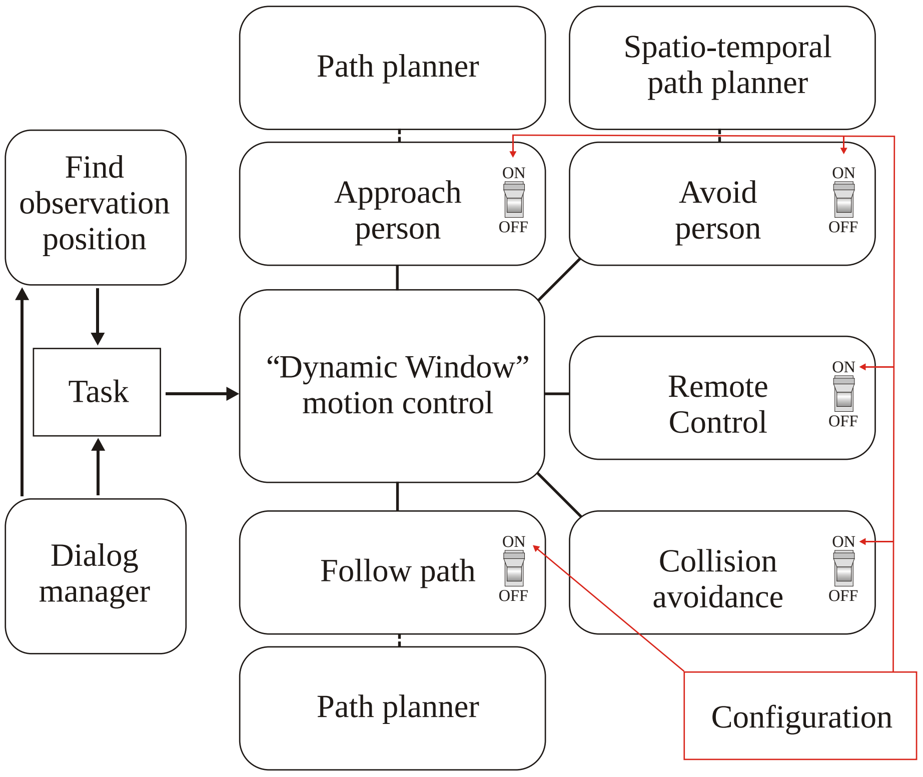

Figure 2.1: the modular structure of the navigator as described in **??**. The dynamic window controller is the core decision unit and is surrounded by a set of objectives, which could be switched on or off. The active objectives vote for the next driving command. The set of active objectives determine the behavior of the robot and has to correspond to the given task.

The configuration of the navigation system is changed by the application designer, or in our case, by the dialog manager. Also, the task is given by the application and it has to be assured, that the navigator configuration and the given task correspond to each other.

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.

In this deliverable, we present also new functionalities like remote control, drive to a good observation position or actively avoid a person. For each functionality, a whole chapter is used to describe the insides. Here, it should only be mentioned, that the remote control behavior as well as the avoiding behavior are realized to be an extra objective, while the observation behavior is a simple "drive to a position" task, where the position is not set manually, but is the result of an optimization process. All new parts, which are described in detail within this deliverable are shown in figure 2.2.

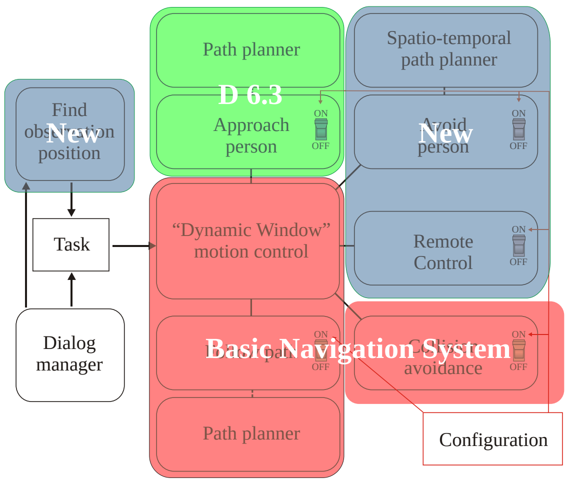

Figure 2.2: the basic modules of the navigation system are shown in red and are provided by MetraLabs. The green part was described in detail in deliverable D6.3 [19] and the blue parts are described within this deliverable.

3 Finding a good position to observe a person in an unobtrusive way

Usually, in mobile robotics the robot has to deal with tasks like interacting with a person or performing a driving task, localizing itself, or building a map of the 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 problem in the 2D and 3D case. This method can be used to "park" the robot at a feasible position and is interfaced by the dialog manager.

3.1 Introduction

In recent years, mobile robotics more and more attain the homes and places of nonexpert users to actually fulfill task like reminders, video call services, emergency aids[13], remote controllable robots[20], guide customers in supermarkets and home improvement stores[12][18], and being robotic butlers to serve drinks or food[24].

In the domain of navigation, there are lot of navigation problems to be solved when dealing only with interaction, map building, path planning, or localization. If one thinks about end user applications in home scenarios, where a robot takes over the role of a service assistant or a butler, there is also a lot of time when the robot is idle and has no actual task to do. Here, the robot only has to recognize commands (e.g. gestures) from the user, while still be able to perceive the user. Due to these reasons, it is also a task for a robot within a long term home scenario to observe a person in a non-intrusive way.

There are several criteria a good observation position should fulfill: (i) the user should of course be visible from that position, (ii) the observation position should be unobtrusive towards the user, (iii) the distance should be sufficient to be able to detect the user with the robot's on board sensors, and (iv) from the chosen observation position the robot should be able to see many possible resting positions, where the user could stay. This knowledge should enable the robot to chose an observation position, where it can place itself most of the time.

Similar problems are rarely described in the literature. There exist approaches of intelligent photograph robots which attempt to take good pictures in party situations [4] or realize a robotic camera man to distinguish good shooting positions for video conferences [31]. In these approaches, the quality of the taken pictures is the central criterion to optimize the observation position. A larger group of publications refers to the so called next-best-view problem. Here, a sequence of observation points should be extracted to gain maximal structural information for example from 3D objects [7] or the structure of the environment (2D and 3D map building) [22], [34]. Within these approaches, the in-

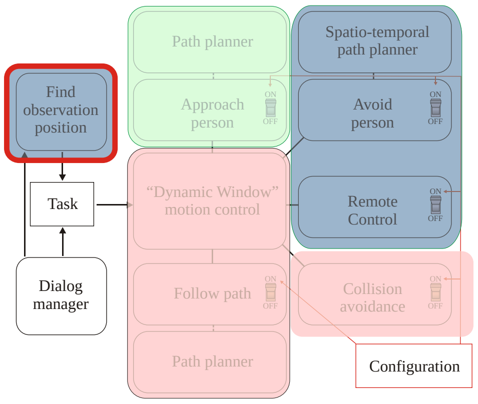

Figure 3.1: the observation software module within the navigation architecture. Note, that this module is separated from the navigation core, since it only emits a sequence of "drive to position" tasks. The module is controlled by tasks, emitted from the dialog manager.

formation of an object structure should be gathered in an incremental fashion by using a minimal number of observations. It differs from our approach, because these approaches try to increase the structural information about the observed objects in an optimal fashion. We do not want to maximize information gain, and it is also not suitable for our problem to observe the person from a position with maximal information retrieval since such a position would eventually not fulfill all our criteria and only guarantees, that the robot's sensors will perceive the person correctly. Other approaches simply try to keep the person in a certain position (and distance) within the camera image or laser scan by using controller schemes [27], [23]. This is called visual servoing and is also not suitable for our problem, since the robot should stay at its position, even when the person moves slightly.

3.2 Integration into the navigation software system

As stated in the introduction, the module for finding a good observation position is separated from the navigation core. It could be triggered by the dialog manager and produces as a result a simple navigation task, that the robot should drive to the current optimal point. New targets are released continuously until the module is deactivated by another dialog manager command. This structure is shown in figure 3.1.

3.3 Formulation of the optimization problem

As stated above, the observation position has to consider a variety of criteria. To find an optimal position to observe a person, the set of criteria has to be evaluated in possible samples 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 next two sections. Additionally, the optimization algorithm is briefly described afterwards.

3.3.1 Boundary conditions

The solution of the optimization process depends on the boundary conditions that exist when the process is started, and which may also 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 3.2 shows all boundary conditions summarized. The person occupancy density function is approximated by building an 3D histogram with bin size w over an infinite time interval to collect user observations, and is no function reflecting time specific intervals:

$$p(o = \vec{x}_i) = \frac{\int_{t=-\infty}^{\infty} o_t(\vec{x} = \vec{x}_i)}{\int_t \int_{\vec{x}} o_t(\vec{x} = \vec{x}_i) \cdot w^3} \tag{3.1}$$

In each cell $\vec{x}$ the number of person observations is counted and normalized over all observations perceived in all positions over the whole observation time interval. Since it is not possible to observe the true function, this function has to be build incrementally, and the estimation of this function could be improved over time. In this work, we chose an efficient grid based space representation, namely an occupancy map representation and a voxel space representation. We collect point cloud data from a 3D Kinect camera, using the OpenNI framework to separate user points from non-user points, and cluster them into voxels to build a person occupancy histogram to represent $p(o)$.

But, is this optimization problem a dynamic or static problem? Looking at the different boundary conditions, the map m is considered to be static. Observed over a huge time interval, even the person occupancy density function $p(o)$ is a static function. But since this function has to be estimated over time, its first representation may be wrong, and the problem begins to show dynamic properties. The person pose o_t is the most fluctuating and dynamic boundary condition, which forced us to handle the problem as a dynamic optimization problem. That is why we have chosen an optimizer suitable for dynamic optimization problems, namely the particle swarm optimization (PSO) technique [9].

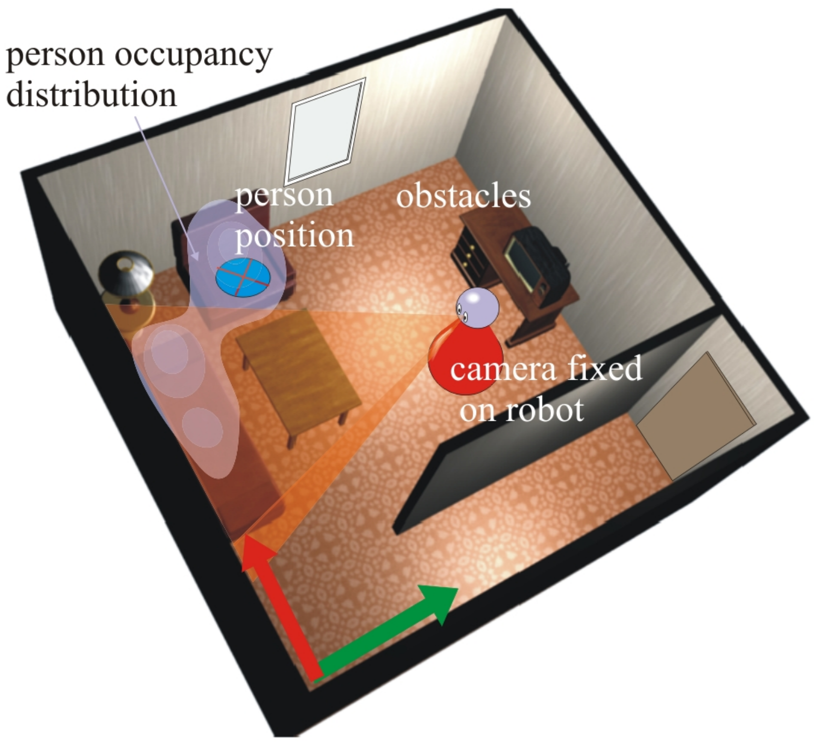

Figure 3.2: the boundary constraints of the optimization problem: the obstacles within the environment, the current person position o_t, the person occupancy distribution $p(o|\vec{x})$ and the position of the camera on the robot.

3.3.2 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 from the front (c_{front}), and (iv) how much of the person's occupancy distribution is observed (c_{podf}). An example of all functions is shown in figure 3.3. 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:

$$\begin{aligned} f(\vec{x}, \phi) &= d(\vec{x}) \cdot v(\vec{x}) \cdot [\alpha_1 \cdot c_{det}(\vec{x}, \phi) + \alpha_2 \cdot c_{dist}(\vec{x}) \\ &\quad + \alpha_3 \cdot c_{front}(\vec{x}) + \alpha_4 \cdot c_{podf}(\vec{x}, \phi)] \end{aligned} \tag{3.2}$$

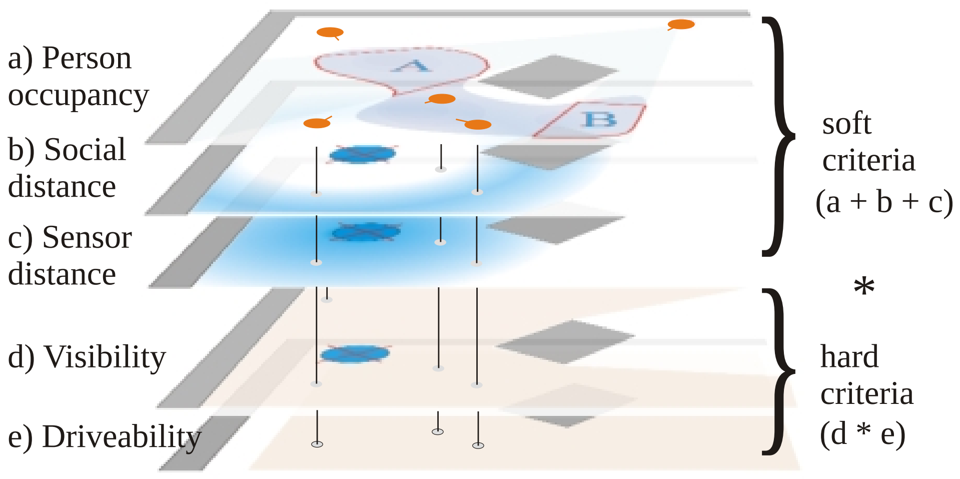

Figure 3.3: hard and soft criteria. The hard criteria mask out the possible search space, and particles (shown as orange dots) are only placed in the remaining region. The soft criteria determine the optimum and are summed up to form the optimization function. Note, that the soft criterion of view direction is not shown here.

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 much of the places where the person *usually* is, are observed by each possible observation pose $\vec{x}_i$? Here, it should be noticed, that the person is observed using the Kinect 3D camera, mounted in a fixed position on the robot and having a certain opening angle. So, a view cone X_f is cast by the camera into the home environment, depending only from the rotation angle of the robot and the position inside X_v. The idea of c_{podf} is now to integrate all observable points $\vec{x}$ from $p(o)$, where $\vec{x} \in X_f$ and where $p(o) > 0$:

$$c_{podf} = \int_{\vec{x}} p(o = \vec{x}) \, , where \; \vec{x} \in X_f \tag{3.3}$$

The key idea is shown in figure 3.4. This function should guarantee that the robot places itself at a position where most of the places the person usually rests at, are observed. Also, the robot should not change position, if the person only moves slightly.

3.3.3 Particle swarm optimization

The optimization problem is simply to find the best 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 [9], [8]) 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. Unlike a particle filter, the particle swarm does not represent a probability distribution. Constriction factor particle swarm optimization (see [8]) is used to solve the problem iteratively. The found solution is further refined by applying a kernel density estimator [29]. Here we will briefly describe what each particle contains, how one iteration of the particle swarm optimization is defined, and how the kernel density estimation is used to get the optimal position from the current iteration step.

Each particle contains a state, which is part of the current optimization space, and a speed vector also placed within that space. So, 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 inside X_v, and the optimization function $f(\vec{x}, \phi)$ is calculated for each particle. Here, two special particles have to be remembered: one is the *currently* best (with the highest value of $f_{p^{[i]}}$), called $p^{[loc\ best]}$, and one is the particle with the best value of $f_{p^{[i]}}$ *ever* measured in all iterations, called $p^{[glob\ best]}$. The key idea behind the particle swarm optimization is, that particles tend to search near both positions for better values of the optimization function. The speed component hereby enables the particles to overcome local minima and circle around both positions. The update of the position and speed component is simple, but needs the constriction factor K to guarantee convergence:

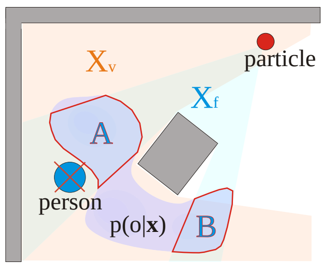

Figure 3.4: in red: the set X_v of all positions where the person is visible. From all other positions the person is covered by an obstacle. Particles only exist in X_v. In blue: the view cone X_f which a particle can observe. Note that X_f defines the area where $p(o|\vec{x})$ is integrated: in this case A+B.

$$\begin{aligned}
\vec{x}_{t+1}^{[i]} &= \vec{x}_t^{[i]} + \Delta t \cdot \vec{v}_t^{[i]} \\
\vec{v}_{t+1}^{[i]} &= K \Big[\vec{v}_t^{[i]} + c_1 \cdot r_1 \cdot (p^{[loc\ best]} - x_{t+1}^{[i]}) \\
&\quad + c_2 \cdot r_2 \cdot (p^{[glob\ best]} - x_{t+1}^{[i]}) \Big] \\
K &= \frac{2}{\left|2 - \theta - \sqrt{\theta^2 - 4\theta}\right|}\ , where\ \theta = c_1 + c_2, \theta > 4
\end{aligned} \tag{3.4}$$

The variables r_1 and r_2 are random numbers from an interval $[0..1]$. The parameters c_1 and c_2 are chosen to prefer either the local best particle or the global best parameter. Since our optimization is defined as a dynamic problem, we select c_1 to have the higher value, because it is not guaranteed that the best position observed contains still the highest value. So, a local optimum is preferred by the particle swarm.

Finally, we use a kernel density estimation to refine our solution of the best position since the particles almost never hit the exact optimum. Here, each particle defines the center of a Gaussian kernel. All particles are summed up to represent a probability distribution $p(x, y)$:

$$p(x,y) = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{2\pi h^2} \cdot e^{-\frac{(x-x_n)^2 + (y-y_n)^2}{2h^2}} \tag{3.5}$$

Here, h represents the kernel size and is estimated by using the highest variance from the x- or y-dimension: $h = max(\sigma_x, \sigma_y)$. Note, that the maximum value could not be extracted from the kernel density in a closed form. Thats why we calculate the density in every point within X_v and simply select the maximum point as the current best observation position. By using the kernel density estimation, we could improve the value of $f(\vec{x}, \phi)$ by 3-4% and we could also supress stochastic outtakes from the random optimization process.

3.4 The 3D case

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.

3.4.1 Data structures

All information is given to the optimizer by using a grid based voxel representation. The obstacle map as well as the person occupancy probability distribution are defined within a voxel grid of the same cell size. The typical size is 10 cm. An example configuration is shown in figure 3.5. Also the view cone consists of a set of voxels, and the robot position is the voxel containing the camera at the defined height over the robot base.

The voxel map is created from an 3D model of the environment in an external process. We do not investigate this problem here, since it refers to the domain of 3D SLAM approaches. The person occupancy probability distribution is a simple histogram, where each voxel volume counts the number of points belonging to a person, which are normalized by all observed person points. These person points are collected from observations with the Kinect 3D camera, the OpenNI framework [1] is used to separate background points from person points. So $p(o = \vec{x})$ for a volume element v is defined as:

$$p_v(o = \vec{x}) = \frac{\sum_{j \in v} o_j}{w^3 \cdot \sum_{i \in m} o_i} \tag{3.6}$$

The numerator counts the 3D points o_j within the volume v, while the denominator counts all observed points o_i ever perceived from the person in the whole map m.

3.4.2 Realization of the single criteria

Driveability

The first criterion we discuss is $d(\vec{x})$. Here, voxels are selected which could be reached by the robots camera. This function is either zero, when the voxel is not reachable, or one when this voxel is reachable. A horizontal cut through the voxel space is created by considering only voxel cells at the same height as the robot base. The resulting layer is dilated by the robot radius, to consider only cells which are reachable by the robot base. Then, a Dijkstra planning algorithm [6] is computed within the layer by starting to propagate a planning function from the current center position of the robot base until no cells of the layer could be added to this function. These voxels are all reachable from the current robot position. They are shifted by the camera height to define the first set of X_v which in fact is the function $d(\vec{x})$.

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 voxel of X_v, if the person could be seen from that voxel. This is done by ray-casting from the current voxel towards the projected person position at the plane defined by the camera height. Note, that also all obstacles have to be projected towards that plane. Here, the set of voxels inside X_v is reduced. An example of both functions is shown in figure 3.5 a). 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 observed voxel $\vec{x}_i$ towards the center of the person position, using the parameter s_{max}, which refers to the maximal distance the sensor could observe:

$$c_{det}(\vec{x}) = \begin{cases} 1 & , if\ d_s < s_{max} - 1 \\ \frac{1}{1+exp(d_s - s_{max} - 0.5)} & , else \end{cases} \tag{3.7}$$

Social distance

The social component is defined in a very similar way. As Hall [15] 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 $\sigma_d = 0.5m$:

$$c_{dist}(\vec{x}) = e^{-\frac{(d_s - 2.5)^2}{2\sigma_d^2}} \tag{3.8}$$

Frontal view

For gesture recognition, face identification and emotion recognition, it is almost every time 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. Since it is a hard detection task to find the gaze direction of the person, we rely on the upper body pose to roughly estimate the view direction of the person. Again, this is provided by the OpenNI framework. 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:

$$c_{front}(\vec{x}) = \begin{cases} 0 & , if\ |\beta| > \pi/2 \\ \frac{1}{1+exp(|\beta| - \pi/6)} & , else \end{cases} \tag{3.9}$$

Person occupancy distribution

As described in section 3.3, the function c_{podf} describes, which part of the person occupancy density function can be seen from the given voxel into the given direction. This is a time consuming operation, since the visibility of the voxels of the person occupancy density function has to be calculated. The key idea is, to cast rays in a regular grid from the hypothetical camera center into the viewing frustrum and follow these rays until an obstacle is hit. Here, the maximal distance from the camera, and the voxel size determine the density of the rays, since the sampling theorem has to be considered. We have to guarantee that at least two rays cross the most distant voxel. All voxels, crossed by a ray, are collected to a set of visible voxels of the viewing frustrum X_{fv}. An example of the frustrum is shown in fig. 3.5 c). Now, all density values which are covered by X_{fv} are summed up from $p(o|\vec{x})$. Figure 3.5 b) shows an example of the function $p(o|\vec{x_i})$.

$$c_{podf} = \sum_i p(o|\vec{x_i}) \text{ , } where \; \vec{x_i} \in \; X_{fv} \tag{3.10}$$

3.4.3 Problems

Although it seems to be the most natural way to calculate the optimization solution within a 3D voxel model, there are some practical problems within the modeling approach. First, we have to build a complete 3D model of the environment before it could be converted into a voxel representation. Incomplete maps tend to find points behind walls as best observation positions, since these walls are incomplete, and a huge effort has to be brought to the robot to construct a feasible map. Second, and most important, the calculation of the visible voxels inside the viewing frustrum X_{fv} of the camera to calculate c_{podf} is very time consuming, which leads to unfeasible calculation times for a real world system (see section 4.5 for details). Regarding these facts, we realized also an approach working only in a 2D world, which drastically reduces the work load per iteration cycle.

3.5 The 2D case

The 2D approach is very similar to the 3D approach. Therefore, we only describe the differences towards the 2D case here.

3.5.1 Data structures

Instead of using voxel representations, we use an occupancy map representation to model our environment. So, all person poses and camera positions are 2D points within that map. The given person occupancy density function is also a 2D function, which is created by integrating $p(o|\vec{x} = (x, y, z))$ over z, leading to $\acute{p}(o|\vec{x} = (x, y))$.

3.5.2 Realization of the single criteria

Driveability

This criterion is also constructed using a Dijkstra planner starting from the given robot position and executed until no further cells could be added to the planning function.

Visibility

Here, from all chosen pixels (which are reachable by the robot) the visibility of the person is checked. If an obstacle is between person and pixel, the corresponding pixel is removed from the search space.

Sensor distance

This function is identical to the 3D case. The Euclidean distance is now calculated in a 2D coordinate system.

Social distance

This function is identical to the 3D case. The Euclidean distance is also calculated in a 2D coordinate system.

Frontal view

This function is identical to the 3D case.

Person occupancy distribution

As mentioned before, the 3D voxel representation of $p(o|\vec{x})$ is projected to one layer resulting in $\acute{p}(o|\vec{x})$. Again, parts of the distribution can be occluded by obstacles, but this time we have to calculate the visibility X_{fv} only for a 2D layer and not for a volume, which significantly speeds up the calculation time. The only problem is, that obstacles in the map may be obstacles in a sense of navigation, but could be overlooked by the robot in the real world. Examples are tables or u-shaped couches, where the robot is not allowed to drive, but the line of view towards the person is free. These cases can be modeled correctly in the voxel grid representation, but appear problematic when dealing only with 2D maps, since the height of obstacles is initially unknown. So, our search space X_v is smaller than necessary. At the moment, this is a recognized problem that will be solved in the future work.

3.6 Experiments

In this section, we show experiments done for the 3D and for the 2D case. The experiments for the 3D environment where done by using two different artificially created (simulated) 3D models (one is shown in figure 3.2), while the 2D experiments where executed by using the 2D occupancy map of our lab. Our experiments where focused on the stability and the speed of our approach.

3.6.1 Finding positions in 3D

Since we simulate the 3D environment, we also define a person position artificially within this environment. The resolution of our map was 10 cm per voxel. The person occupancy distribution is recorded within a real setup of a living room, which equals the simulated sitting area, and uses a Kinect device and the OpenNI library to track the person around the sitting area. Since our PSO never terminates the optimization process, we measure

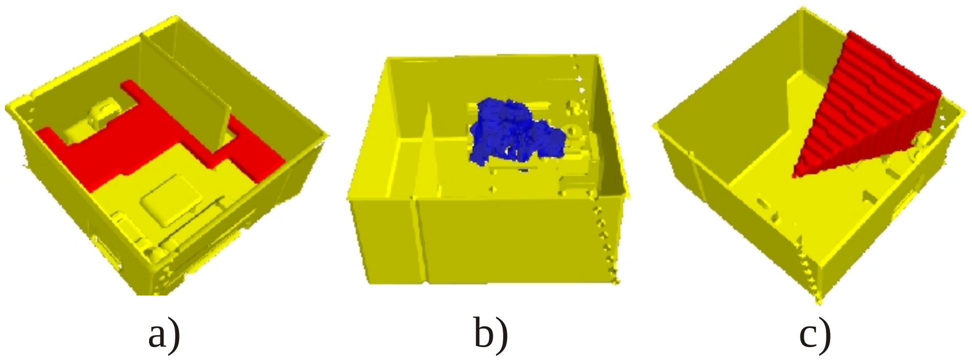

Figure 3.5: an example of a rasterized 3D home environment with a) the two criteria of driveability and visibility $(d(\vec{x}) \cdot v(\vec{x}))$, b) the person occupancy density, and c) a view cone of one particle.

the found best position after 100 iterations, the calculation time for all iterations, and the average time per iteration. The results are shown in table 3.1.

Resulting position				Calculation time	
	mean	variance			t_{avg}
x	4.14 m	7.6 cm		update	54.09 s - 15.34 s
y	2.94 m	4.8 cm		KDE	0.6 s
ϕ	20°	0.0°		total	90 min - 25.5 min

Table 3.1: On the left: resulting optimal position of one map. We perform 10 runs to get the mean position value and variance. We executed 100 iterations per run. On the right: average time consumption for one iteration, the kernel density estimation (KDE), and the total processing time for all 100 iterations. Note, that the highest time value represents one processor core while the lowest value represents six cores. We used 100 particles for the optimization process. Due to these unusable calculation times, we tried to simplify the approach to a more efficient 2D version.

The accuracy of the found position is 7.6 cm at average which is more than sufficient for the task. The calculations are executed using a 6 core 3.5 GHz AMD Phenom II processor at 3.2 GHz. We used up to all 6 cores, since the calculation of $f(\vec{x}, \phi)$ at multiple particle positions could be parallelized easily. It can be seen, that the processing power needed for the optimization process exceeds the limits of the best hardware of todays mainstream computers. Not to speak that such a hardware is usually not used within mobile robots. The user has to wait at least half an hour until the robot reaches a good position. Most of the processing power is used by the soft criterion c_{podf} to calculate the view cone X_{fv}.

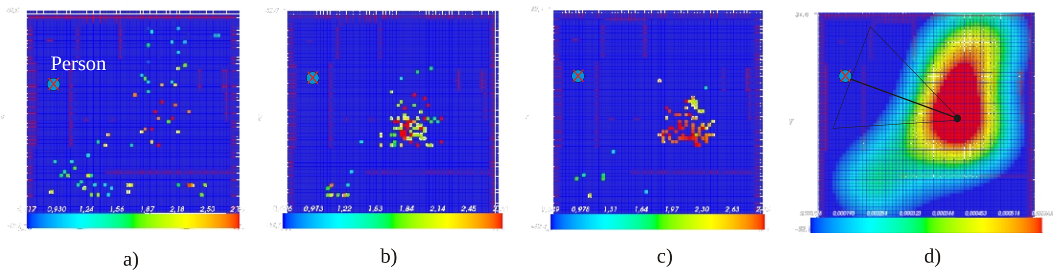

Figure 3.6: Top view of the 3D model with an example timeline of the particle swarm. The person position is shown by the blue crossed circle. Obstacle edges are shown as red grid structures. Solid dots show particles. The colors of the particles code the result of the optimization function $f(\vec{x}, \phi)$. Image a) shows the initial configuration of the swarm, while b) shows the swarm after 7 iterations and c) after 100 iterations. It can be seen, that the particles converge towards one position and increase their values of $f(\vec{x}, \phi)$. The last image d) shows the calculated kernel density estimation of the particle set from c) with the extracted optimal observation position.

3.6.2 Finding positions in 2D

Modeling the optimization problem in 3D seems to be the most detailed and natural solution, but incorporates practical problems like the processing power, which makes this approach impossible on current mobile robots. For this reason, we reduce our approach to work with a 2D world representation. Here, we can use a real map of our lab (see fig. 3.7) and the same static Kinect camera that records $p(o|x, y, z)$ to detect the person position. Note, that the implemented approach is not able to use the same amount of detail, especially on predicting the visibility of the person for c_{podf}. Nevertheless, the found solution also produces reliable results by providing a much faster calculation time. This is shown in table 3.2.

Resulting position				Calculation time	
	mean	variance			t_{avg}
x	-3.1 m	13.1 cm		update	32 ms - 64 ms
y	1.6 m	14.5 cm		KDE	1.7 s
ϕ	$-95°$	$0.5°$		total	8.1 s - 4.9 s

Table 3.2: The comparing results of the 2D case. Here, the calculation of c_{podf} is done in 2D and speeds up the process significantly. Up to two cores of the robots hardware where used. Note, that the resulting coordinates differ from the 3D case since a different map was used.

With a variance of 14.5 cm, the found results are not as reliable as the 3D results, but also provide also stable positions. Here, a 2.66 GHz Intel dual core processor is used, running directly on the mobile robot. The results show calculation times of 5 - 10 seconds, depending on the number of used cores, which are much faster than the 3D results and lead to a usable system on todays robot hardware.

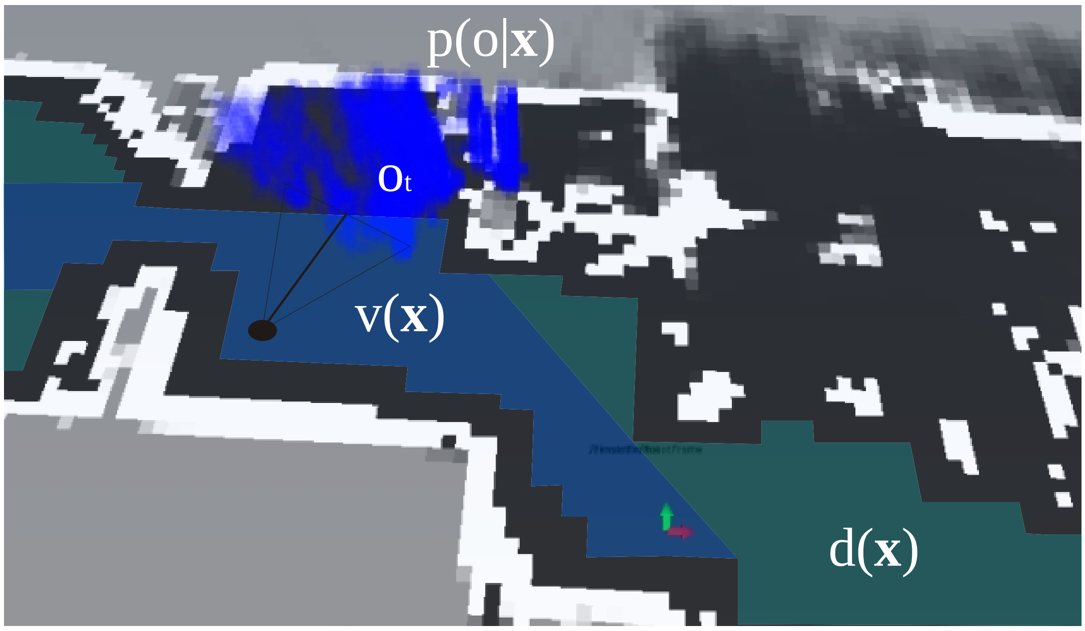

Figure 3.7: 2D map of our lab environment with the person occupancy density function $p(o|\vec{x})$ as the original voxel representation, the drivable space from $d(\vec{x})$, and the visible space $v(\vec{x})$ where the person o_t could be observed. Note that o_t is not visible here, since it is covered by the blue voxel elements from $p(o|\vec{x})$. Also, the final observation position is shown.

3.7 Conclusion

In this chapter, we have shown an approach, how to observe a person by also considering positions, where the person usually sits, by providing a person occupancy distribution, and using a variety of other criteria. We have also shown a 3D version of our approach, which solves this problem in a 3D voxel grid. This solution turns out to be to slow to be computed on actual robot hardware. For this reason, we show a simpler 2D version, which also generates stable results and has the only drawback, that the visibility of the person cannot be reliably predicted, giving only the occupancy map. This is a task we want to solve in the future by learning the visibility using also the Kinect data and adapting a visibility map during operation of the system. It is also planned to deploy a set of experiments to show the influence of the different criteria towards the resulting position and its stability. 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. 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 extend $p(o|\vec{x})$ and use this function for that purpose.

4 Actively avoiding the motion path of a moving person

When mobile robots operate in environments, like home environments or care centers, a robot should take into account the inhabitants while moving around. In this chapter, an approach is presented, which at 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. We deploy a potential field approach to predict the person's movement trajectory, and use a modified Fast Marching planner to access a time-variable cost function for the planning process. The goal of our development is 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. Within the ALIAS project, the proposed method is applied in the background and therefore transparent to all other modules.

4.1 Introduction

If mobile robots are used in everyday life, the acceptance of these robots should be good, especially, when the users are non-expert users. As experiments show [28], humans tend to observe a technical device, like a robot, as a social entity and project emotions on such a machine. This causes the users to expect human-like behaviors from mobile robots. Normally, the scenario of human-robot interaction is investigated, when the robot and human want to interact in a dialog with each other. 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 is on a tour to collect food orders, or 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. Even if humans do not want to interact with each other, there is some kind of communication between them, known also as body language. These human-human behaviors are quite complex and investigated deeply by psychologists. One aspect of the body language is the theory of the personal space, founded by Hall [14]. In our work, the spatial distance from Hall is used, which corresponds to "non interaction", and which therefore represents a meaningful distance for a human being. All persons (or robots), which keep a distance above this threshold, are interpreted as potential non-interaction partners. In our work, we want to 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. An non-intrusive path towards a predefined goal, which does not touch the personal space of a person, is planned.

Related work: In the COGNIRON project a lot of work was done to investigate, if the model of the personal space is also valid for human-robot interaction, and if the findings are comparable to human-human interaction [5, 36]. These investigations found out, that

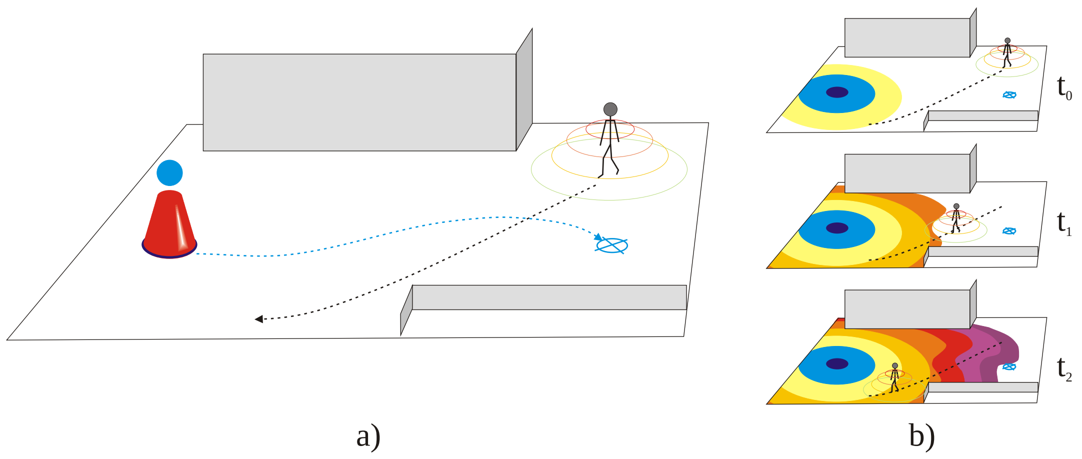

Figure 4.1: the idea of the presented approach: the robot should be able to politely pass a moving person. To do so, the person path is predicted (see a)) and the personal space of the person is used in a spatio-temporal planning process to compute a feasible path. In b), a planning wave is propagated 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.

spatial configurations indeed carry information about the intention of a dialog act, and that they are similar to the findings of Hall. In robotics, the personal space is used regularly in tasks such as approaching a person [20, 35] and also path planning [33]. While the person is approached, the method of [35] can deal with changing person positions in a reactive way. The method of Sisbot[33] is only defined in an static environment, and cannot deal with changing environment situations during the planning phase, but uses the same simple personal space model than our approach. In [25], a rule based behavior was constructed to pass a person in a feasible distance in straight floors. This behavior has the same motivation than our method, but only works in floor-like environments and fails in complex situations or environments like narrow door passages. Today, there are no further known publications on the topic of politely passing a moving person with respect to the person's distances. However, there are many approaches which concern spatio-temporal path planning, which is also the core technique of our approach. The most advanced methods operate on planning trees. For example in [30, 21], lattice graphs are used to create a tree with spatial and temporal information as long as the motion prediction of the moving objects are certain. When the predictions become uncertain the algorithm only uses the spatial knowledge to plan further. Anyhow, this algorithm is very time consuming and is not processable in real time on a robot system. Another approach is presented by [17] and [20], where expanding space trees are used to create a collision free path in space and time to steer a robot. These approaches are very powerful in terms of describing the spatial-temporal information and are fast to calculate, but are weak when the robot deviates from the planned path. In this work, we use a modified standard planning approach, where the robot is able to find an optimal path from every position in space, even when deviations from the optimal path occur. A fundamental precondition for spatio-temporal planning is the prediction of the motion trajectories of the observed person. Here, a large set of prediction algorithms exist, mostly using probability densities, which are build upon a large set of trajectory observations [18, 3]. The disadvantage of these approaches is the need of an exhaustive data collection of trajectories over a long time. We prefer an out-of-the-box approach, where the trajectory of a person is predicted using the current motion direction and a potential field, presented in [17], to predict the person movement for the next few seconds.

Presented approach: Our approach uses a modified version of the Fast Marching Method (see [32]), 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 the maximal 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 4.1.

4.2 Integration into the navigation software system

This module should completely replace the classical (static) path planning approach. Thats why it is also designed as an objective with connected path planner. 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.

Note that, because of the ambiguity of both approaches, only one objective could be active at a time!

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, but it is used here to predict near future person movement. 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

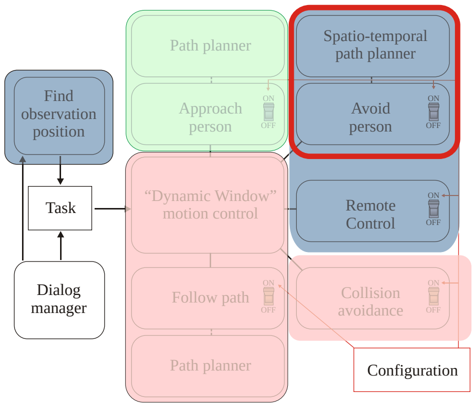

Figure 4.2: the "avoid person" software module within the navigation architecture. This module is directly linked to the navigation core and completely replaces the classical path planning module. So, only one of both modules could be active during operation.

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. But, the resulting force is not only determined by the obstacle configuration, it is also influenced by the virtual target of the person, resulting in a pulling vector field $\vec{E}_{target}$. This target assumes a tangential line with positive charges towards the current direction of motion at a defined distance (see figure 4.3), and is in any case constant relative to the person's coordinate system. Since the resulting force could be also calculated as shown in equation 4.1, for an infinite line the resulting force is a force towards the current motion direction of the person. So, the resulting force is the vector sum of a force towards

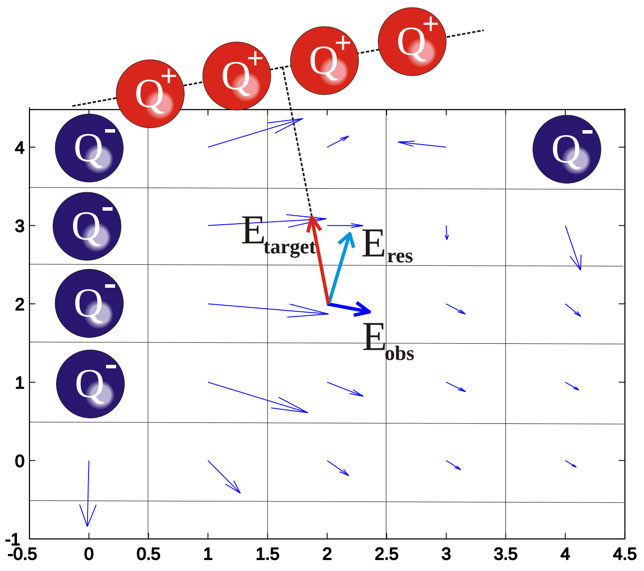

Figure 4.3: this image shows the resulting vector field $\vec{E}(\vec{x})$, which is sourced by the negative charges of the obstacle cells. The resulting force on the moving person is defined by two components. The pushing field E_{obs} of the obstacles (blue) and the pulling force E_{target} of the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

the current motion direction and a disturbing force, sourced by the obstacle configuration:

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

The idea of predicting the trajectory is, to simulate the movement 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 and a valid person position and a valid walking direction to provide a sufficient prediction of the person's trajectory.

4.3.2 Motion Prediction

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 [32] 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. In static cases, near zero values are assigned to obstacle cells, whereas high values are assigned to free space. The advantage of this planner is, that all positive real values can be applied to the map cells, while in most common planning approaches [6, 16] only binary values could be used. This section describes briefly the original function of the Fast Marching Method, and afterwards the changes applied to the classical approach. Here, 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. This is the main difference to other planning approaches (e.g. E* [26]) that use the Fast Marching Method, where the traveling times of the wave in each cell are only used to define the gradient a robot should follow, but do not carry any real world information. The main benefit of the standard Fast Marching Method is the ability, to construct monotonical raising functions with *any* configuration of positive speed values, which is essential for a path planning algorithm.

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. Now, a distance transform is performed to slow down the robot (and in this case the traveling wave) when traveling near an obstacle. This setting defines a static velocity value for each cell, which is v_{max} in free space and linear interpolated towards zero by using the distance transform.

The general goal of the Fast Marching Method is the numeric solution of the so called Eikonal equation $\vec{v}(\vec{x}) \cdot |\Delta T(\vec{x})| = 1$. The solution of this equation describes the evolution of a closed curve in time T, reacting on the different speeds $\vec{v}(\vec{x})$ at the positions $\vec{x}$. In most cases the solution could not be found in closed form. Fast Marching proposes a very simple numerical solution to this problem. The wave starts from a single point and spreads to neighboring points by expanding grid cells, which are currently part of the wavefront. In a regular grid, the neighbors are easy to find and 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 deleted from the list, until no expandable cells remain. 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

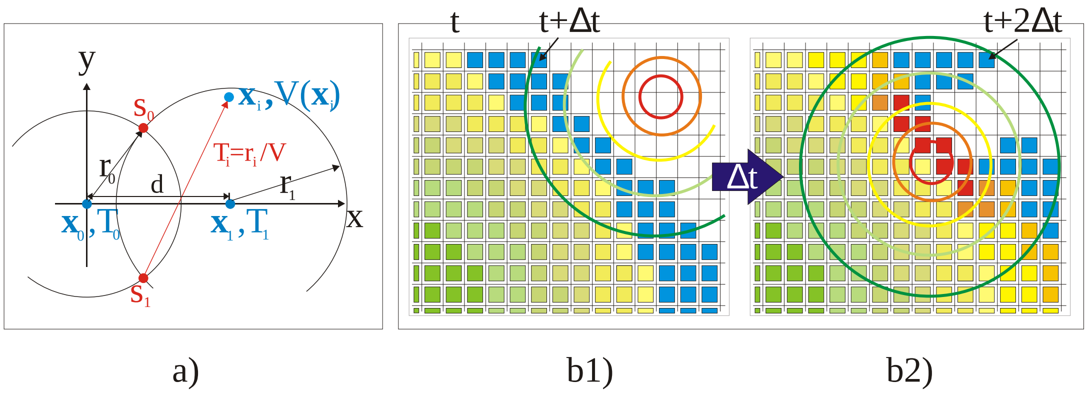

Figure 4.4: in image a), the details of the interpolation of one cell element of the wavefront are shown. Blue values are the given ones, while black values are computed. The red values describe the final step of interpolation, where from the virtual wave sources s_0 or s_1 the passing time of the wavefront 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)).

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 only in the end. 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 robots position is the source of the wavefront. Second, the fusion process 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. 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.
The dynamic speed function consists of two parts: the static part $v_{st}(\vec{x_i})$ from the obstacle configuration, described in section 4.4.1, and a dynamic part $v_{dyn}(\vec{x_i}, t_0 + n \cdot \Delta t)$, coming from the predicted motion trajectory of the person and their corresponding personal space:

$$v_{st}(\vec{x_i}) = \begin{cases} v_{max} \cdot \frac{d(\vec{x_i}) - d_{min}}{d_{max} - d_{min}} \; , if \; d(\vec{x_i}) \leq d_{max} \\ v_{max} \; , else \end{cases} \tag{4.5}$$

$$v_{dyn}(\vec{x_i}, t_0 + n \cdot \Delta t) = 1 - exp\left(-\frac{|\vec{x_i} - \vec{x}_p(t_0 + n \cdot \Delta t)|^2}{2\pi\sigma_p^2}\right) \tag{4.6}$$

Here, $d(\vec{x_i})$ is the distance to the next obstacle cell, described by the distance transform of the map, and $\vec{x}_p(t_0 + n \cdot \Delta t)$ is the predicted position of the person at the current simulation time. The personal space is set to be above 2.6 meters to symbolize non-interaction, and so, the value of σ_p is set to 2.6 meters. The fusion is done by a simple minimum operation:

$$v(\vec{x_i}, t_0 + n \cdot \Delta t) = min(v_{st}(\vec{x_i}), v_{dyn}(\vec{x_i}, t_0 + n \cdot \Delta t)) \tag{4.7}$$

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 also 2.6).

4.5 Experiments and Results

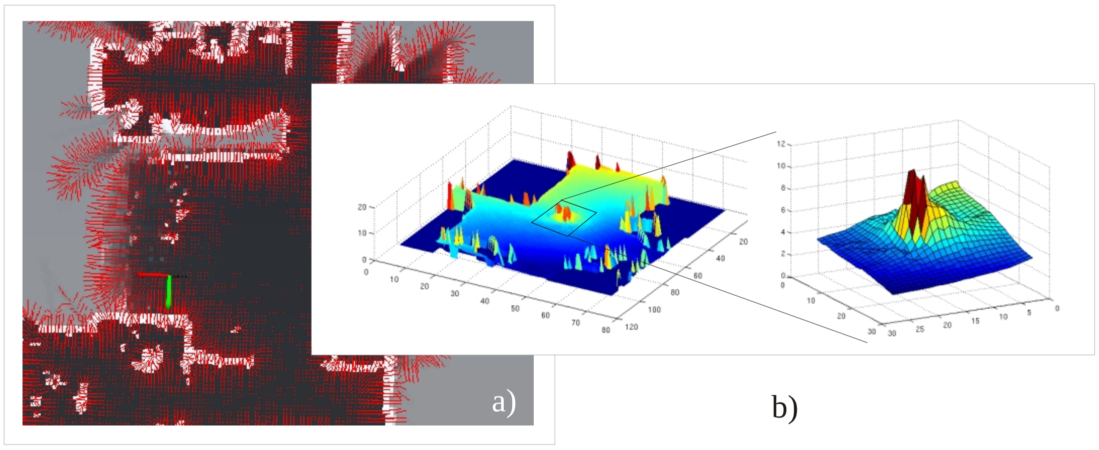

Figure 4.5: in a), an example of the force field is shown, which is used for motion prediction. In b) the function of the passing times of the wave is shown. From this function the resulting path is created by gradient descent from the target towards the robot's position. It can be seen, that the traveling 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 our living lab 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. 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. The map is 15m x 100m and has a map resolution of 10cm per cell. Person detection and tracking is done by using a laser based leg detector, based on the approach of Arras [2]. The resulting planning function and the associated cell speeds, which correspond to the passing time of the wavefront, are shown in figure 4.6 for the narrow space scenario and figure 4.7 for the passing scenario. It can be seen, that in both cases the personal space of the moving person slows down the wavefront and guides the wavefront around the person. When the goal is reached by the wavefront, gradient descent is used to extract the optimal path.

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. We measure the average runtime of the algorithm with different prediction intervals Δt for a total prediction period of 10 seconds. Smaller time intervals Δt mean more accurate motion prediction and wave propagation. Table 4.1 shows the results of the runtime investigation. In average, the method is capable of simulating 13 times faster than real time. The simulation step time of 0.5 seconds is used for the motion prediction of the person and the update of the planning function, since this time provides

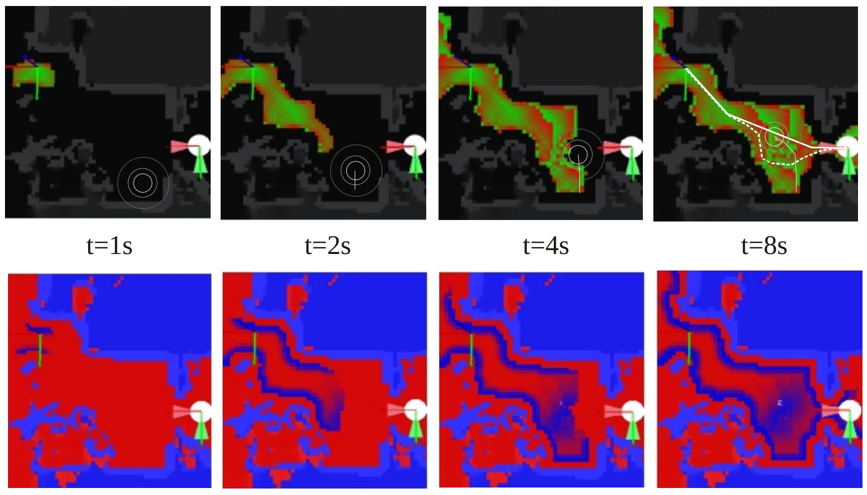

Figure 4.6: propagation of the planning wave in a narrow space. The robot starts on the left side and has to reach the goal on the lower right. The person is located at the bottom (multiple bright circles) and walks through the room. The wavefront travels through the room until the target is reached and avoids the personal space. Finally, with gradient descent a path is extracted from the wave's passing times. The final path is shown as a dashed line, whereas the planned path without a person is shown as a solid line. Note that every two seconds in simulation time the color of the wavefront changes from red to green. Below the traveling time function the used cell speeds are shown, which are calculated when the wavefront passes the cells. Blue correspond to slow traveling speeds, while red corresponds to high traveling speeds.

maximal accuracy by providing still good performance. The simualtion of ten seconds of motion can be done in 770 milliseconds.

The calculation of the force field $\vec{E}_{obs}$ is constant for the given map and is done once before the algorithm starts. Since this is a time consuming operation, 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 2.66 GHz was used. Only one core does the wavefront propagation since this is a highly sequential task and it is hard to parallelize this algorithm.

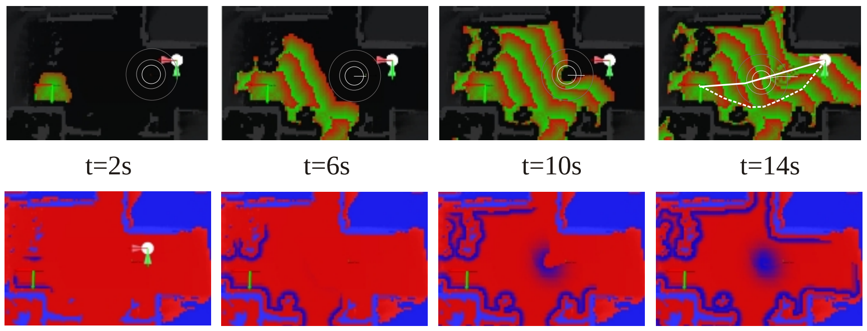

Figure 4.7: propagation of the planning wave on a floor. The robot starts on the left side and has to reach the goal on the upper right. The person is located at the right side (multiple bright circles) and walks through the corridor towards the left. The wavefront travels through the room until the target is reached and avoids the personal space. Finally, with gradient descent a path is extracted from the wave's passing times. The final path is shown as a dashed line, whereas the planned path without a person is shown as a solid line. For a further description of the lower series of pictures, refer to image 4.6.

4.6 Conclusion and future work

In this chapter, an approach for spatio-temporal path planning with regard of one moving person is shown. Up to this stage, the problem of re planning is only addressed when the person deviates from the predicted path. At the one hand, this behavior of the robot has to be investigated in further experiments. At the other hand, an investigation has to be done, what happens if the robot could not keep track of the planned path and planned time and deviates from the given task. The approach could be easily extended towards usage of multiple persons, but may need in this case a more complex prediction model. Also, the planning process could be further improved by optimizing the execution speed and try to parallelize the most time consuming operation of wave propagation.

Simulation Step	**Δt=3s**	**Δt=1.5s**	**Δt=0.5s**	**Δt=0.2s**
t_{avg}	75ms	75ms	75ms	89.2ms
t_σ	72ms	35ms	18ms	13.4ms
Speed factor	13	13	13	11

Table 4.1: overview of the achieved computation times for different time steps Δt. Here, t_{avg} is the average computation time, while t_σ is the variance of the computation time per iteration step. On larger time steps up to 0.5 seconds the system is able to predict and plan 13 times faster than real time. Only on small simulation steps, this factor begins to lower. In test runs a simulation time of 0.5 seconds is chosen.

5 Robot Remote Control

5.1 Introduction

During 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 emerges during the user trials and questionnaires, where the users demand 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.

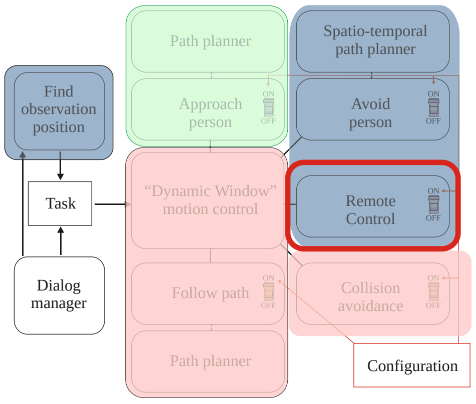

Figure 5.1: the "remote control" software module within the navigation architecture. This module is directly linked to the navigation core.

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, relies 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.

5.2 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 and in the other direction the sensor feedback from the obstacle situation from server (robot) to client. See figure 5.2 for the definition of the client and server.

We use the AngleScript interface from MetraLabs to send 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.

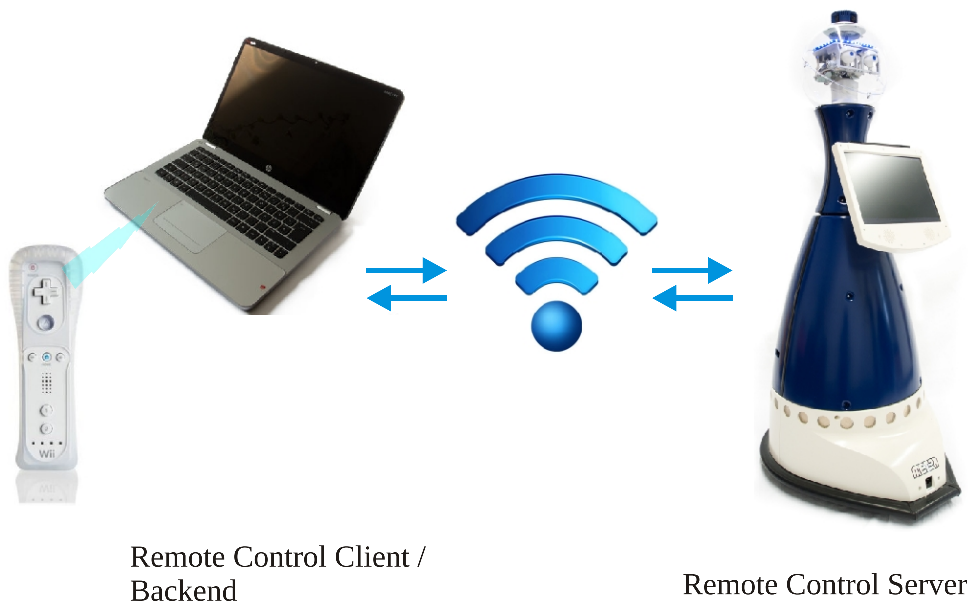

Figure 5.2: the network configuration during the remote control process. The robot is steerable with the Wii remote controller, which is connected via Bluetooth to the remote client. The client uses a standard TCP/IP connection to the remote server (the robot). Over this connection the wished user speed 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.

5.2.1 Remote Back end

The remote back end is the client side of the remote control and is designed to work on every standard PC with network access. 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 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 and if it is necessary to deviate from the wished speed configuration to avoid collisions. The deviation is done by the robot in an autonomous fashion.

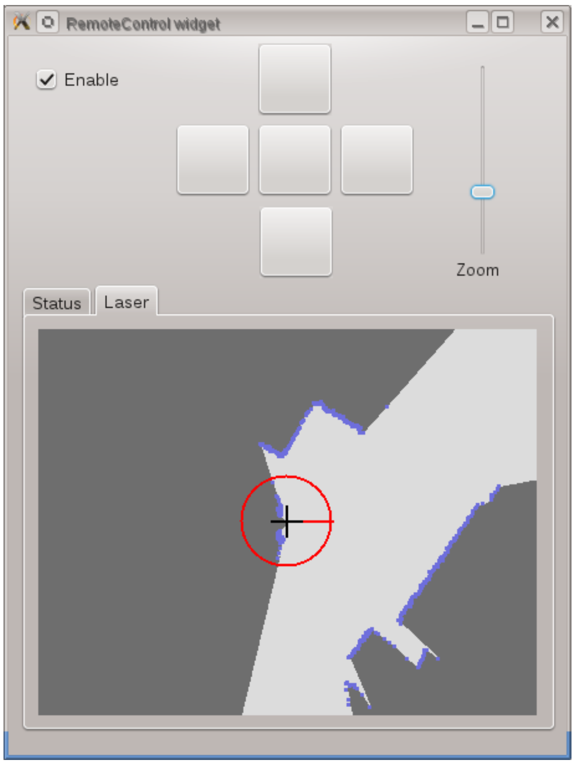

Figure 5.3: the remote back end. The robot is controlled by the Wii controller. The commands are send to the robot and the current laser scan is sent back and visualized on the screen. Since the robot has a diameter of 0.6m, the user can estimate the distance towards the next obstacle. Note, that the robot's front direction is marked by the red line in the circle. Free space is shown in white, while walls are shown in blue color.

The client side is responsible to open the TCP/IP connection towards the robot, to connect to the Wii controller and, most important, to display the last laser scan to visualize the current obstacle situation. It also sends new driving speed commands to the robot. These commands are evaluated on the server side. Note, that also a Skype session could be started, to initiate or accept a video conference with the robot's user or to investigate the emergency situation, but this is handled by the dialog manager.

5.2.2 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 default *not* coupled to a video call, but both functionalities could be combined. 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 suggest, that it is no sharp decision the remote controller sets, when steering the robot with the controller. All speed combinations nearby the given speed pair have to be supported also by the system and should be treated as an alternative.

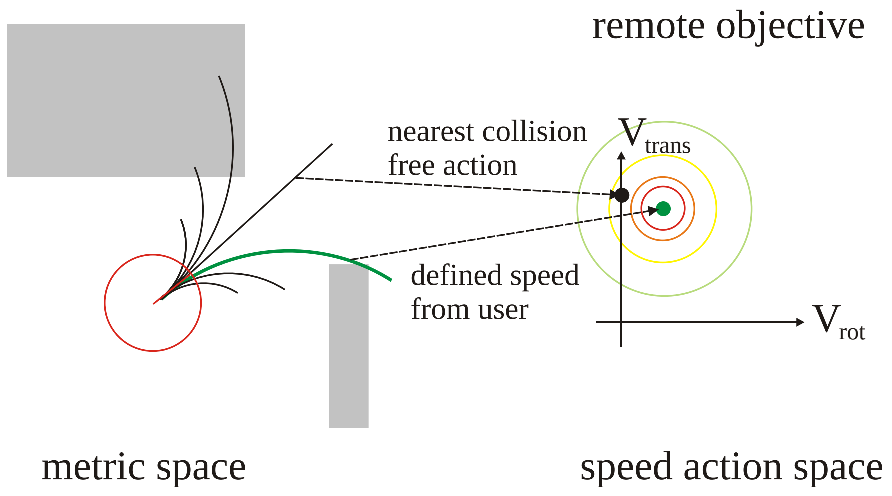

Figure 5.4: the voting process within the dynamic window: in each step, the dynamic window evaluates a set of speed actions which in term lead to a specific trajectory. The user defines one speed pair, which lead to the wanted trajectory (green on the left side). Our objective votes this speed pair best. 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.

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*. 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 speed $\langle V_{rot}^{u}, V_{trans}^{u} \rangle$:

$$f(V_{trans}^{(i)}, V_{rot}^{(i)}) = e^{\frac{(V_{rot}^{(i)} - V_{rot}^{u})^2 + (V_{trans}^{(i)} - V_{trans}^{u})^2}{\sigma^2}} \tag{5.1}$$

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 could be re-used for our purpose.

6 Conclusions

In this deliverable, we showed the update of the navigation system with some new modules like the person avoidance module, the remote control module (with back end), and the observation module. In figure 6.1 the whole robot system is sketched to give a brief summery, where these modules belong to and how they interact with the overall system.

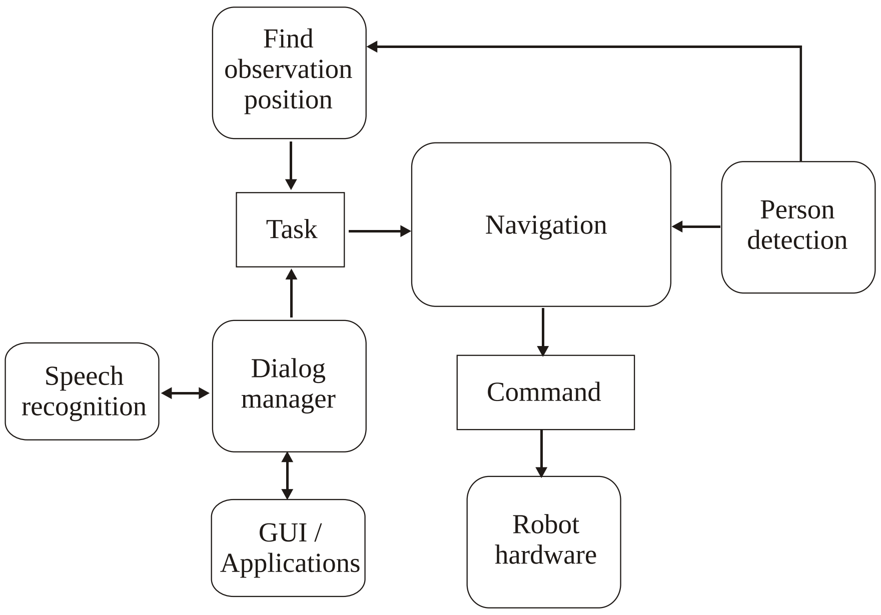

Figure 6.1: system overview: while the observation module is only loosely coupled to the navigation system, the remote control objective and person avoidance module belong directly to the navigator. The observation module and person avoidance module need the input of the person detector.

Note, that the navigation system is interfaced only by giving tasks and configuring the active objectives. This could only be done be the dialog manager (and with the help of AngleScript). The core of the navigation system, consisting of the dynamic window and a set of objectives, needs the help of several subsystems, like person detection, localization and map-building. These techniques are not described in this deliverable, since they are used "as is" and no further improvements are done in these sectors.

From the presented modules, only the observation module is loosely coupled towards the navigation system. The remote control and the person avoidance are very closely coupled towards the navigator, since these modules are objectives of the dynamic window. The person avoidance module should replace the normal "drive to goal" objective and only changes the behavior when approaching a goal, while the remote control objective extends the functionality of navigation system towards an new operation mode.

6.1 Outlook

With the described functionality of this deliverable, the development work within the domain of navigation is nearly done. In the time period to the next deliverable, we will mainly test our approaches and improve details like person detection, runtime optimization and so on. Additionally, we will deal with an autonomous map building and setup process, since it showed in our user tests, that an expert has to build the map and this should be also do-able by a standard technician. This functionality should be implemented more user centered and this problem will be addressed during the remaining reporting time.

Bibliography

[1] Openni framework, 2011. http://openni.org/Documentation/.

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

[3] A. Bruce and G. G. Gordon. Better motion prediction for people-tracking. In *Proc. ICRA*, 2004.

[4] Z. Byers, M. Dixon, K. Goodier, C. Grimm, and W. Smart. An autonomous robot photographer. In *Proc. IROS*, pages 2636–2641. IEEE, 2003.

[5] K. Dautenhahn et al. How may i serve you? a robot companion approaching a seated person in a helping context. In *Proc. HRI*, pages 172–179, 2006.

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

[7] E. Dunn, J. van den Berg, and J.-M. Frahm. Developing visual sensing strategies through next best view planning. In *Proc. IROS*, pages 4001–4008. IEEE, 2009.

[8] 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.

[9] 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.

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

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

[12] H.-M. Gross, H.-J. Boehme, C. Schroeter, S. Mueller, A. Koenig, E. Einhorn, C. Martin, M. Merten, and A. Bley. Toomas: Interactive shopping guide robots in everyday use - final implementation and experiences from long-term field trials. In *Proc. IROS*, pages 2005–2012. IEEE, 2009.

[13] H.-M. Gross, C. Schroeter, S. Mueller, M. Volkhardt, E. Einhorn, A. Bley, C. Martin, T. Langner, and M. Merten. I'll keep an eye on you: Home robot companion for elderly people with cognitive impairment. In *Proc. on IEEE Int. Conf. on Systems, Man, and Cybernetics (IEEE-SMC 2011)*, pages 2481–2488. IEEE, 2011.

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

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

[16] E. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science and Cybernetics*, 4:100–107, 1968.

[17] F. Hoeller, D. Schulz, M. Moors, and F. Schneider. Accompanying persons with a mobile robot using motion prediction and probabilistic roadmaps. In *Proc.IROS*, pages 1260Ű–1265, 2007.

[18] T. Kanda, M. Shiomi, Z. Miyashita, H. Ishiguro, and N. Hagita. A communication robot in a shopping mall. *IEEE Transactions on Robotics*, 26(5):897–913, 2010.

[19] J. Kessler. D6.3: First navigation software module. 2011.

[20] J. Kessler, A. Scheidig, and H.-M. Gross. Approaching a person in a socially acceptable manner using expanding random trees. In *Proc. 5th European Conference on Mobile Robots (ECMR)*, pages 95–100, 2011.

[21] M. Likhachev and D. Ferguson. Planning long dynamically-feasible manuevers for autonomous vehicles. *Int. Journal of Robotics Research*, 28(8):933–945, 2009.

[22] K. Low and A. Lastra. Efficient constraint evaluation algorithms for hierarchical next-best-view planning. In *Third International Symposium on 3D Data Processing, Visualization, and Transmission*, pages 830–837, 2006.

[23] X. Ma, C. Hu, X. Dai, and K. Qian. Sensor integration for person tracking and following with mobile robot. In *Proc. IROS*, pages 3254–3259, 2008.

[24] A. Mertens, U. Reiser, B. Brenken, M. Luedtke, M. Haegele, A. Verl, C. Brandl, and C. Schlick. Assistive robots in eldercare and daily living: Automation of individual services for senior citizens. In *Proc. of 4th International Conference on Intelligent Robotics and Applications*, pages 542–552. Springer LNAI 7101, 2011.

[25] E. Pacchierotti, H. Christensen, and P. Jensfelt. Evaluation of passing distance for social robots. In *Proc. RO-MAN*, 2006.

[26] R. Philippsen. *Motion Planning and Obstacle Avoidance for Mobile Robots in Highly Cluttered Dynamic Environments, PHD Thesis*. Univ. of Toulouse, Ecole Polytechnique Federale de Lausanne, 2004.

[27] M. Piaggio, R. Fornaro, A. Piombo, L. Sanna, and R. Zaccaria. An optical-flow person following behaviour. In *Proceedings of the IEEE ISIC/CIRA/ISAS Joint Conference*, pages 301–306, 1998.

[28] B. Reeves and C. Nass. *The Media Equation: How People Treat Computers, Television, and New Medial Like Real People and Places*. CSLI Press, Stanford, 1996.

[29] M. Rosenblatt. Remarks on some nonparametric estimates of a density function. *Annals of Mathematical Statistics*, 27:832–837, 1968.

[30] M. Rufli and R. Siegwart. On the application of the d* search algorithm to time-based planning on lattice graphs. In *Proc. ECMR*, pages 105–110, 2009.

[31] C. Schroeter, M. Hoechemer, S. Mueller, and H.-M. Gross. An autonomous robot photographer. In *Proc. ICRA*, pages 424–429. IEEE, 2009.

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

[33] E. Sisbot. *Towards Human-Aware Robot Motions, PHD Thesis*. Univ. of Toulouse, Toulouse, 2006.

[34] M. Strand and R. Dillmann. Using an attributed 2d-grid for next-best-view planning on 3d environment data for an autonomous robot. In *Proc. ICRA*, pages 314–319, 2008.

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

[36] L. Takayama and C. Pantofaru. Influences on proxemic behaviours in human-robot interaction. In *Proc. IROS*, pages 5495–5502, 2009.