in: Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS 2012), Vilamoura, Portugal, pp. 4392-4398, IEEE 2012

# I'll Keep You in Sight: Finding a Good Position to Observe a Person

Jens Kessler[1] and Daniel Iser[1] and Horst-Michael Gross[1]

***Abstract*— Usually, in mobile robotics the robot has to deal with tasks like interacting with a person or performing a driving task. But what happens, if the robot just has to wait and thereby still has to react on user commands? In this case, 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. Such a position has to fulfill different criteria: first, to guarantee the observability of the person with the robots on-board sensors and second, the robot should be able to observe the person when the person changes its position at its resting place. In this paper, a new approach is presented how to find a position, providing all these aspects, by solving an optimization problem using a particle swarm optimizer. We also present first results for that problem in the 2D and 3D case.**

## I. 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 [1], remote controllable robots [2], guide customers in supermarkets and home improvement stores [3], [4], and being robotic butlers to serve drinks or food [5]. One of the projects that provide a robot to remind and entertain elderly users, living in their home environments, is the ALIAS (Adaptable Ambient LIving ASsistant) project. ALIAS has the goal of developing a mobile robot system to "interact with elderly users, monitor and provide cognitive assistance in daily life, and promote social inclusion by creating connections to people and events in the wider world" [6]. The used robot is shown in Fig. 1.

In the domain of navigation, there are a lot of problems to be solved when dealing with interaction, map building, path planning, or localization. But, 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 has to recognize commands (e.g. gestures) from the user while it needs to 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 not disturb 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, especially when the person changes its position only slightly.

*This work was financed by the project AAL-2009-2-049 "Adaptable Ambient Living Assistant" (ALIAS) co-funded by the European Commission, the Federal Ministry of Education and Research (BMBF), and the Thuringian Ministry of Science.

[1]Neuroinformatics and Cognitive Robotics Lab, Ilmenau University of Technology, Ilmenau, Germany, jens.kessler@tu-ilmenau.de

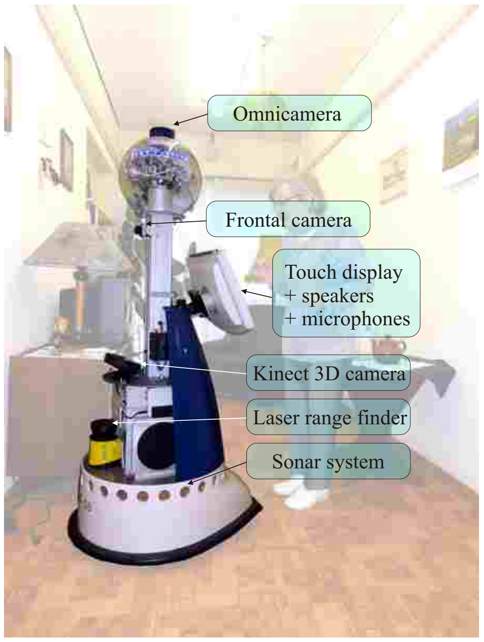

Fig. 1. The ALIAS robot, a SCITOS G5 platform of MetraLabs GmbH, equipped with cameras, a Kinect 3D sensor, and a laser range finder. It interacts with the user by touch-display and speech output.

Similar problems are rarely described in the literature so far. There exist approaches of intelligent photograph robots which attempt to take good pictures in party situations [7], or realize a robotic camera man to distinguish good shooting positions for video conferences [8]. 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 [9] or the structure of the environment (2D and 3D map building) [10], [11]. Within these approaches, the information about an object or a scene has to be gathered in an incremental fashion by using a minimal number of observations. It differs from our approach in a way, that these approaches try to increase the structural information about the observed objects or scenes 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 [12], [13]. 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, e.g. on the couch.

The remainder of the paper is structured as follows: in section II, at first the optimization problem with its different criteria is shown in a principal fashion. In section III the problem is formulated for the 3D case, while in section IV the simplified 2D case is shown. Afterwards, experiments for both cases are presented in section V, and a conclusion is drawn in section VI.

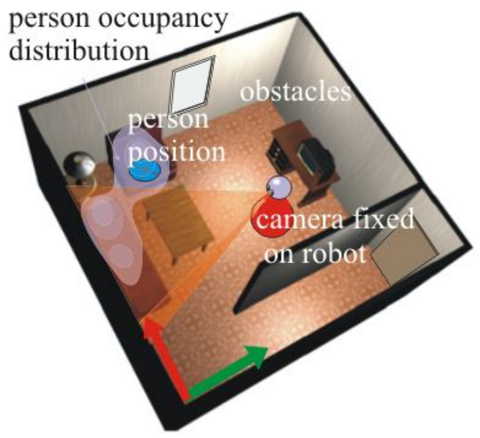

Fig. 2. The boundary constraints of the optimization problem: the obstacles within the environment, the current person position $o_t$, the person occupancy distribution $p(o)$, and the position of the camera on the robot.

## II. 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 is described by the position $\mathbf{x} = (x, y, z)$ of the robot and its horizontal view direction $\phi$. 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 $\phi$ has to be considered. Within this three dimensional search space, the optimal point has to be chosen as the best observation position. Since the problem is formulated as an optimization problem, 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.

### A. 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 $m(\mathbf{x})$ 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 $p(o = \mathbf{x})$ to give the probability that a person can be observed at a certain point $\mathbf{x}$ in the home environment. Figure 2 shows all boundary conditions summarized. The person occupancy density function is approximated by building an 3D histogram.

Here, in each histogram bin the number of person observations is counted and normalized over all observations perceived in all bins over the whole observation time. 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 one? Looking at the different boundary conditions, the map $m$ is considered to be static. Observed over a long 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) [14].

### B. The optimization function

The optimization function $f$, defined in 1, reflects the different criteria to be considered and fuses these criteria into a single function. It is a function over the optimization space $S = \{\mathbf{x}, \phi\}$, where $\mathbf{x} \in \Re^2, \phi \in \Re$. There are two hard criteria to reflect physical properties to constrain the search space and mask out impossible search positions. These are the driveability $d(\mathbf{x})$, and the visibility of the person $v(\mathbf{x}, \phi)$. Both functions $d$ and $v$ are binary functions defined within the map space $m(\mathbf{x})$.

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 in that observation pose ($c_{det}$), (iii) to perceive the person from the front ($c_{front}$), and (iv) how much of the person's occupancy distribution can be observed from that observation pose ($c_{podf}$). An example of all functions is shown in Fig. 3. Since these criteria are no hard criteria, an optimal compromise between the linear superimposed criteria has to be found. So, the resulting optimization function can be defined as follows:

$$\begin{aligned} f(\mathbf{x},\phi) &= d(\mathbf{x})\cdot v(\mathbf{x})\cdot[\alpha_1\cdot c_{det}(\mathbf{x},\phi)+\alpha_2\cdot c_{dist}(\mathbf{x}) \\ &\quad +\alpha_3\cdot c_{front}(\mathbf{x})+\alpha_4\cdot c_{podf}(\mathbf{x},\phi)] \end{aligned} \tag{1}$$

Note that $f(\mathbf{x},\phi)$ is also a function of all boundary conditions, but due to simplicity they are absent in the function formulation and will be described in section III and IV. Most criteria are simple, and at this point we will take a closer look only to $c_{podf}$, the criterion considering the person's occupancy distribution.

Since positions are already masked out, where the person could physically not be seen (using $v(\mathbf{x})\cdot d(\mathbf{x})$), it is possible to observe the person from the subset of all remaining positions $X_v = \mathbf{x}_1 ... \mathbf{x}_n$. Now, the question is: how much of the places where the person *usually* rests at, are observed by each possible observation pose $\mathbf{x}_i$? 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 model of the home environment, depending only on the orientation of the robot and the position inside $X_v$. Note that $X_f$ is not a subset of $X_v$! The idea of $c_{podf}$ is now to integrate all observable points $\mathbf{x}$ from $p(o=\mathbf{x})$, where $\mathbf{x}\in X_f$ and where $p(o)>0$ to indicate, how much of the resting places are observable from an observation position:

$$c_{podf} = \int_x p(o=\mathbf{x}) \;, if \;\mathbf{x}\in X_f \tag{2}$$

The key idea is shown in Fig. 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 its position, if the person only moves slightly.

### C. Particle swarm optimization

The optimization problem is simply, to find the best values of $(\mathbf{x},\phi)$ that maximizes the output of $f(\mathbf{x},\phi)$. Our solution

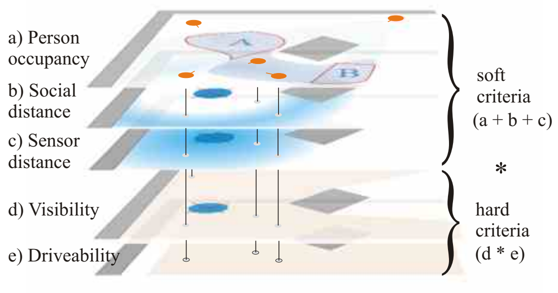

Fig. 3. Hard and soft criteria. The hard criteria mask out the possible search space, and particles are only placed in the remaining region. The soft criteria determine the optimum and are summed up to form the optimization function. Note, that the soft criterion of view direction is not shown here.

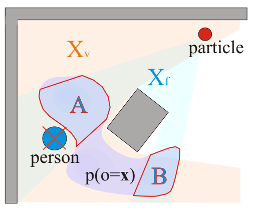

Fig. 4. In red: the set $X_v$ of all positions where the person is visible. From all other positions the person is covered by an obstacle. Particles only exist in $X_v$. In blue: the view cone $X_f$ which one selected particle can observe. Note that $X_f$ defines the area where $p(o)$ is integrated: in this case the region A and B.

to the defined optimization problem uses the particle swarm optimization (PSO) approach. It is a well known technique (see [14], [15]) 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 [15]) is used to solve the problem iteratively. The found solution is further refined by applying a kernel density estimator [16]. Subsequently, we are briefly describing 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 consists of a state, which is part of the current optimization space, and a speed vector also residing within that space. So, in our case one particle is defined by a position $(x,y)\in X_v$ and a view direction $\phi$. Note that we only optimize over $(x,y)$ and chose $\phi$ to view directly towards the current person position $o_t$. That's why, we only need speed components of the particles in $x$ and $y$ direction, namely $v_x$ and $v_y$. This way, each particle is defined by $p^{[i]} = \{\mathbf{x}^{[i]} = (x,y), \phi^{[i]}, \mathbf{v}^{[i]} = (v_x,v_y)\}$. In the first step, particles are randomly initialized inside $X_v$, and the optimization function $f(\mathbf{x},\phi)$ is calculated for each particle. Here, two special particle positions have to be remembered: one is the *currently* best, with the highest value the particle $i$ has experienced so far, called $p^{[loc\ best]}_{[i]}$, and one is the particle with the best value of $f_{p^{[i]}}$ *ever* measured over all particles in all iterations, called $p^{[glob\ best]}$. The 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 move around both positions. 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.

$$\begin{aligned} \mathbf{x}^{[i]}_{t+1} &= \mathbf{x}^{[i]}_t + \Delta t\cdot \mathbf{v}^{[i]}_t \\ \mathbf{v}^{[i]}_{t+1} &= K\Big[\mathbf{v}^{[i]}_t + c_1\cdot r_1\cdot (p^{[loc\ best]}_{[i]} - x^{[i]}_{t+1}) \\ &\quad + c_2\cdot r_2\cdot (p^{[glob\ best]} - x^{[i]}_{t+1})\Big] \end{aligned} \tag{3}$$

The update of the position and speed component is simple, but needs the constriction factor $K$ to guarantee convergence.

$$K = \frac{2}{\left|2-\theta-\sqrt{\theta^2-4\theta}\right|} \;, with\ \theta = c_1 + c_2, \theta > 4 \quad (4)$$

Since our optimization is defined as a dynamic problem, we select $c_1$ to be the higher value to allow each particle to be dominated by its individual best position and not the global best position. This way, the particle swarm should be more explorative and also be able to adapt to changing boundary conditions.

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 $p(x, y)$:

$$p(x,y) = \frac{1}{N}\sum_{n=1}^{N} \frac{1}{2\pi h^2} \cdot e^{-\frac{(x-x_n)^2+(y-y_n)^2}{2h^2}} \quad (5)$$

Here, $h$ represents the kernel size and is estimated by using the highest variance of the particles positions in 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. That's 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 $f(\mathbf{x}, \phi)$ by 3-4%, and we could also supress stochastic outtakes from the random optimization process.

## III. 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.

### A. 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 Fig. 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 derived from a 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 [19]. The person occupancy probability distribution is a simple histogram, where each voxel volume counts the number of points belonging to a person observed earlier. These person points are collected from observations with the Kinect 3D camera. The OpenNI framework [17] is used to separate background points from person points. So, $p(o)$ for a cubic voxel element $v$ with the width $w$ is defined as:

$$p_v(o = \mathbf{x}) = \frac{\sum_{j\in v} o_j}{\sum_{i\in m} o_i \cdot w^3} \quad (6)$$

The numerator counts the 3D points $o_j$ within the voxel $v$ and normalize them by the volume $w^3$, while the denominator counts all observed points $o_i$ ever perceived from the person in the whole map $m$.

### B. 3D-Implementation of the single criteria

*1) Driveability:* Here, we discuss the first criterion of equation 1: $d(\mathbf{x})$. This function selects all voxels 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, and to be able to assume a point like robot. Then, flood-filling at the current robot position is applied to efficiently extract all reachable voxels. But, since potential camera viewing positions should be simulated by a set of particles, the voxel positions on the ground layer are shifted by the camera height to define the first set of $X_v$, which defines the function $d(\mathbf{x})$.

*2) Visibility:* The next function is the visibility criterion $v(\mathbf{x})$. Although this function is independent from $d(\mathbf{x})$, it makes sense to only consider points which are inside $X_v$, since $d(\mathbf{x})$ and $v(\mathbf{x})$ 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 inside $X_v$ towards the projected person position at the plane defined by the camera height. Note that all obstacles have to be projected towards that plane. With that, the set of voxels inside $X_v$ is reduced. An example of both functions is shown in Fig. 5 a). With both functions known for a given map and a given person position, the particle swarm could be initialized by randomly drawing voxels from $X_v$, if no other particle already represents that voxel. The selection is finished, when the full particle count is reached.

*3) Sensor distance:* The sensor distance criterion $c_{det}(\mathbf{x})$ has to consider the ability to detect a person with a certain sensor. Since we use the Kinect sensor, the recognition distance is limited to 3 meters. So, the sensor distance $d_s = |\mathbf{x}_i - o_t|$, which is the Euclidean distance from the observed voxel $\mathbf{x}_i$ towards the center of the person position, and the maximal sensor distance $s_{max}$ is considered:

$$c_{det}(\mathbf{x}) = \begin{cases} 1 & , if\ d_s < s_{max} - 1 \\ \frac{1}{1+exp(d_s - s_{max} - 0.5)} & , else \end{cases} \quad (7)$$

*4) Social distance:* The social criterion $c_{dist}$ should keep the robot away from the observed person. As Hall [18] explains, the social distance, where persons do not consider to interact with each other, is around 2.5 meters and above. This value is our social distance to ensure an observed person to feel comfortable. To consider this fact, a circular function is defined around the person, using the parameter $\sigma_d$ to define the thickness of the circle:

$$c_{dist}(\mathbf{x}) = e^{-\frac{(d_s-2.5)^2}{2\sigma_d^2}} \tag{8}$$

*5) Frontal view:* For gesture recognition, face identification, and emotion recognition, it is often necessary to observe the user from the front. That's 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 a 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 upper body orientation towards the robot's pose is defined to be angle $\beta$. With that angle, $c_{front}$ could be described as follows:

$$c_{front}(\mathbf{x}) = \begin{cases} 0 & ,if\ |\beta| > \pi/2 \\ \frac{1}{1+exp(|\beta|-\pi/6)} & ,else \end{cases} \tag{9}$$

*6) Person occupancy distribution:* As described in section II, 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 multiple rays in a regular grid from the hypothetical camera view into the viewing cone 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 cone $X_{fv}$. An example of the cone is shown in Fig. 5 c). Now, all density values which are covered by $X_{fv}$ are summed up from $p(o)$. Figure 5 b) shows an example of the function $p(o)$.

$$c_{podf} = \sum_i p(o = \mathbf{x}_i)\ ,if\ \mathbf{x_i} \in\ X_{fv} \tag{10}$$

## C. Complexity 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 this 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 done to construct a feasible map. Second, and most important, the calculation of the visible voxels inside the viewing cone $X_{fv}$ of the camera to calculate $c_{podf}$ is very time consuming, which leads to calculation times not feasible for a real world system (see section V for details). Because of these facts, we implemented also an approach working only in a 2D world, which drastically reduces the work load per iteration cycle.

## IV. THE 2D CASE

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

### A. 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 $p(o = (x, y, z))$ over $z$, leading to $\acute{p}(o = (x, y))$.

### B. 2D-Implementation of the single criteria

*1) Driveability:* This criterion is also constructed using flood-filling, starting from the current robot position, to mark all reachable cells.

*2) Visibility:* Here, from all chosen cells inside $X_v$(which could all be reached by the robot) the visibility of the person is checked. If an obstacle is between the person and the investigated cell, the corresponding cell is removed from the search space.

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

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

*5) Frontal view:* This function is identical to the 3D case.

*6) Person occupancy distribution:* As mentioned before, the 3D voxel representation of $p(o = \mathbf{x})$ is projected to one layer resulting in $\acute{p}(o = \mathbf{x})$. 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 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 future work by adding a probabilistic 2.5D elevation map estimation which also uses the Kinect measurements.

## V. EXPERIMENTS

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

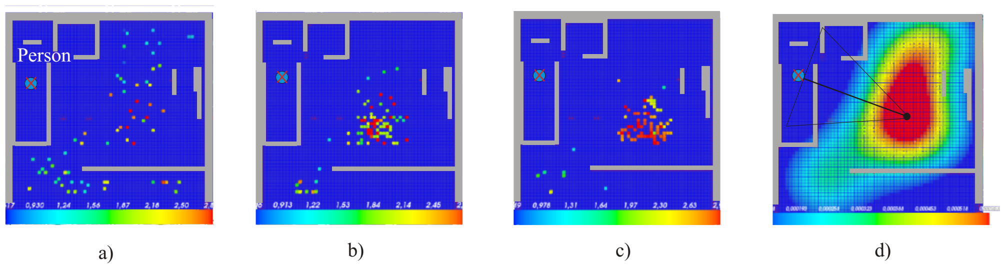

Fig. 6. Top view of the 3D model with an example timeline of the particle swarm. The person position is shown by the blue crossed circle. Driving obstacle edges are shown as gray boxes. Solid dots show particles. The colors of the particles code the result of the optimization function $f(\mathbf{x}, \phi)$. Red color represent high values while blue color represent small values. 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 $f(\mathbf{x}, \phi)$. The last image d) shows the calculated kernel density estimation of the particle set from c) with the extracted optimal observation position.

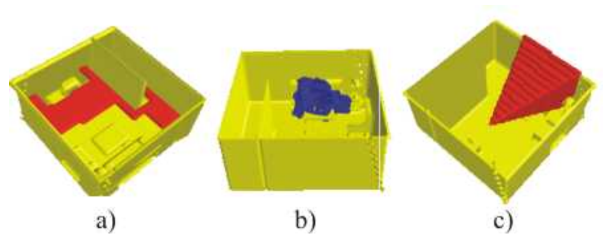

Fig. 5. An example of a rasterized 3D home environment with a) the two criteria of driveability and visibility $(d(\mathbf{x}) \cdot v(\mathbf{x}))$ , b) the person occupancy density, and c) a view cone of one particle.

| Resulting position | | | Calculation time | |
|---|---|---|---|---|
| | mean | variance | | $t_{avg}$ |
| $x$ | 4.14 m | 7.6 cm | update | 54 s - 15 s |
| $y$ | 2.94 m | 4.8 cm | KDE | 0.6 s |
| $\phi$ | 20° | 0.0° | total | 90 min - 25 min |

Fig. 7. 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 within the PSO for the optimization process.

## A. Finding positions in 3D

Since we simulate the 3D environment, we also define a person position artificially within this environment. The resolution of our simulated map was 10 cm per voxel. The person occupancy distribution was recorded within a real sitting setup, which equals the simulated sitting area, using 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 the found best position after 100 iterations, the calculation time for all iterations, and the average time per iteration. The results are shown in Fig. 7.

The variance of the found observation positions from 10 test runs is 7.6 cm, 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 $f(\mathbf{x}, \phi)$ at multiple particle positions could be parallelized easily. But still, the user has to wait at least half an hour until the robot calculates a good position. Most of the processing power is used by the soft criterion $c_{podf}$ to calculate the view cone $X_{fv}$. This calculation time is not feasible for real-world applications, and even the used hardware setup is much faster than our standard robot hardware. So, calculation times on a real robot will become even worse.

| 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 |
| $\phi$ | −95° | 0.5° | total | 8.1 s - 4.9 s |

Fig. 8. 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 robot's hardware were used. Note, that the resulting coordinates differ from the 3D case since a different map was used for this search.

## B. Finding positions in 2D

Due to the high demands of processing power, we reduced our approach to work with a 2D world representation as described in IV. Here, we can use a real map of our lab (see Fig. 9) 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. Note that the resulting positions are not comparable to the 3D case, since the 2D case reflects real world data, while the 3D case uses a simulated environment. The results for the 2D case are shown in Fig. 8.

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

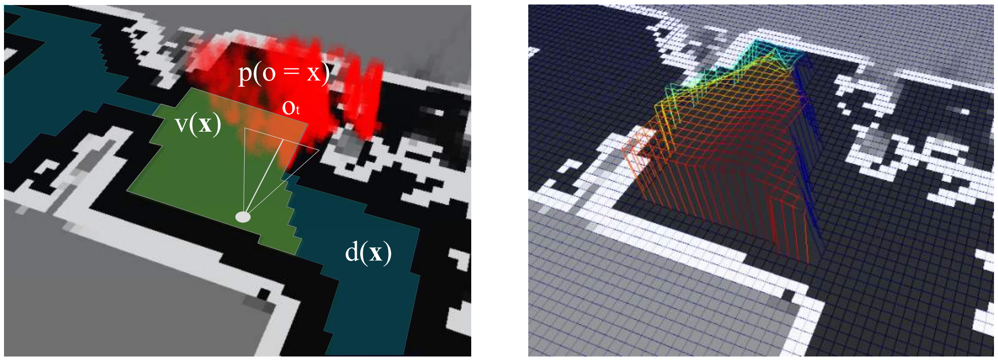

Fig. 9. On the left: 2D map of our lab environment with the person occupancy density function $p(o)$ as the original voxel representation, the drivable space from $d(\mathbf{x})$, and the visible space $v(\mathbf{x})$ where the person $o_t$ could be observed. Note that $o_t$ is not visible here, since it is occluded by the red voxel elements from $p(o = \mathbf{x})$. Also, the final observation position is shown. On the right: the evaluation function $f(x, \phi)$, calculated for each cell of the map. The calculation of the full function in each point is computational expensive and is only shown for convenience. The particle swarm optimization replaces the full calculation to find a more efficient solution.

## VI. CONCLUSIONS

In this paper, we have presented an iterative approach how to find a feasible observation position by considering multiple criteria and a technique, how to fuse these criteria. We have proposed a 3D version of our approach, which solves this problem by using a 3D voxel map. This solution turns out to be too slow to be computed on up-to-date robot hardware. For this reason, we implemented 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 2D occupancy map. This task is planned to be solved in the future by learning the visibility using the Kinect data and adapting a 2.5D elevation map during operation of the system. We also plan to include additional hard and soft criteria towards the optimization problem. One further hard criterion, which has to be considered, is to not block the line of view from the user towards known ”objects of interest” (like the TV). Another desired soft criterion could be, that the robot should not place itself on paths where the person usually walks in his/her living environment. Overall, we want to provide an observation system with multiple modular, exchangeable, and reconfigurable conditions, that could be adapted to different observation tasks with one common optimization solution and fusion technique.

## REFERENCES

[1] Gross, H.-M., Schroeter, Ch., Mueller, S., Volkhardt, M., Einhorn, E., Bley, A., Martin, Ch., Langner, T., Merten,M , I’ll keep an Eye on You: Home Robot Companion for Elderly People with Cognitive Impairment, in Proc. IEEE-SMC 2011, pp. 2481-2488, 2011

[2] Kessler, J., Schroeter, Ch., Gross, H.-M., Approaching a Person in a Socially Acceptable Manner Using Expanding Random Trees, in: Proc. 5th ECMR, pp. 95-100, 2011

[3] Gross, H.-M., Boehme, H.-J., Schroeter, Ch., Mueller, St., Koenig, A., Einhorn, E., Martin, Ch., Merten, M., Bley, A., TOOMAS: Interactive Shopping Guide Robots in Everyday Use - Final Implementation and Experiences from Long-Term Field Trials, in: Proc. IROS, pp. 2005-2012, 2009

[4] Kanda,T. et al.,N., A Communication Robot in a Shopping Mall, in: IEEE Transactions on Robotics, vol. 26, nr.5, pp. 897-913, 2010

[5] A. Mertens et al., 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, Springer LNAI 7101, pp. 542-552, 2011

[6] F. Walhoff and E. Bourginion, ALIAS Project description, http://www.aal-alias.eu/content/project-overview, 2012

[7] Z. Byers et al., An autonomous robot photographer, in: Proc. IROS, pp. 2636-2641, 2003

[8] Schroeter, Ch., Hoechemer, M., Mueller, St., Gross, H.-M., Autonomous Robot Cameraman - Observation Pose Optimization for a Mobile Service Robot in Indoor Living Space, in: Proc. ICRA, Kobe, Japan, pp. 424-429, 2009

[9] E. Dunn and J. van den Berg and J.-M. Frahm, Developing Visual Sensing Strategies through Next Best View Planning, in: Proc. IROS , pp. 4001-4008 , 2009

[10] K.L. Low and A. Lastra, Efficient Constraint Evaluation Algorithms for Hierarchical Next-Best-View Planning, Third International Symposium on 3D Data Processing, Visualization, and Transmission , pp. 830-837 , 2006

[11] 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 , pp. 314-319 , 2008

[12] M. Piaggio et al., An optical-flow person following behaviour, in Proceedings of the IEEE ISIC/CIRA/ISAS Joint Conference, pp. 301-306, 1998

[13] X. Ma et al., Sensor integration for person tracking and following with mobile robot, in: Proc. IROS, pp. 3254-3259, 2008

[14] R. Eberhart and Y. Shi, Particle swarm optimization: developments, applications and resources, in: Proceedings of the 2001 Congress on Evolutionary Computation, vol. 1, pp.81-86, 2001

[15] R. Eberhart and Y. Shi, Comparing inertia weights and constriction factors in particle swarm optimization, Proc. of the Congress on Evolutionary Computation, vol. 1, pp. 84-88, 2000

[16] M. Rosenblatt, Remarks on some nonparametric estimates of a density function, in: Annals of Mathematical Statistics, vol 27, pp.832-837, 1956

[17] http://www.openni.org, 2010

[18] E.T. Hall, Proxemics, in: Current Anthropology, vol. 9, nr.2, pp. 83+, 1968

[19] N. Engelhard and F. Endres and J. Hess and J. Sturm and W. Burgard, Real-time 3D visual SLAM with a hand-held RGB-D camera, Proc. of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum, Sweden, 2011