2019 IEEE Intelligent Transportation Systems Conference (ITSC)
Auckland, NZ, October 27-30, 2019

Fusing Shape-from-Silhouette and the Sparsity Driven Detector for Camera-Based 3D Multi-Object Localization with Occlusions

Matthias Michael* Daniela Horn* Sebastian Houben*

Abstract— **Position estimation of multiple objects in a 3D environment poses a challenging task, even more so in the presence of occlusions due to infrastructure. In this paper we present a method to accurately localize up to 10 moving pedestrians by fusing the output of a Sparsity Driven Detector with volumes generated by a Shape-from-Silhouette approach. We also show how occlusion information from a 3D map of the environment can be integrated into our algorithm to further improve performance. We investigate the influence of different camera heights and image sizes on the optimization problem and demonstrate real-time capability for certain configurations. Additionally, our code is made publicly available under an open source license.**[1]

I. INTRODUCTION

The tracking and position estimation of multiple moving objects, *e.g.*, for surveillance systems, was, and still continues to be, a major challenge in computer vision. Reliable results in this area can improve safety in a number of applications such as robotics and autonomous driving. Recent progress in the latter requires vehicles to gather more and more information on their environment to maneuver safely on their way. While driving autonomously on highways is already a manageable task, moving vehicles in an inner-city context needs much more precise and detailed information on present objects and obstacles as less assumptions can be made – a circumstance which applies to flowing traffic and off-street scenarios, *e.g.*, in car parks, alike. Often the required kind of information can not be provided by the sensors already installed in the car since they are generally limited in range. Thus, they cannot deal properly with occlusions evoked by immobile objects, such as lampposts or shrubbery, or other traffic participants. Especially so-called vulnerable road users (VRUs), *i.e.*, pedestrians, cyclists etc. are highly endangered in these situations. Using additional static sensors (*e.g.*, CCTV cameras) can overcome a majority of the discussed problems.

Laser-based sensors can provide accurate 3D information of objects in an observed scene. However, they have drawbacks in form of rather large investment and installation costs. Image-based sensors, on the other hand, are affordable and can be more flexibly applied to a variety of applications. In this paper, we present a robust method to detect multiple moving objects and accurately reconstruct their 3D position based on camera images.

*Ruhr University Bochum, Insitute for Neural Computation {firstname.lastname}@ini.rub.de

[1]https://github.com/MatthiasMichael/SparsityDrivenDetector

Our method fuses the results of an adapted version of the *Sparsity Driven Detector* (SDD), as developed by Wu et al. [1], [2], [3], together with the volumes generated by a *Shape-from-Silhouette* (SfS) approach [4] to estimate accurate object positions. While the SDD itself is independent of the number of cameras used to capture input images, SfS requires at least two cameras looking at the scene to infer voxel occupancy. Computation is done independently for each frame and no explicit object tracking or interpolation is employed.

This paper is organized as follows: Section II gives a short overview on already proposed systems and methods dealing with the topic of multiple object tracking. Our method is described in Section III while experimental setup and results are discussed in Section IV. Section V concludes the paper.

II. RELATED WORK

The reconstruction of 3D object positions from multiple cameras generally follows a 2-step process, in which relevant objects are first detected in each camera image individually before being merged to a 3D representation. The crucial aspect is to solve the correspondence problem, *i.e.*, to identify the same object from different camera perspectives. Once the identification process has been successful, objects can optionally be tracked over time for further stabilization.

Bredereck et. al. [5] propose an image-based 3D tracking algorithm which utilizes a greedy matching method considering the initial two-dimensional tracking per camera. 3D positions are then obtained by triangulating the positions of cameras currently seeing the observed object. The authors claim that the approach neither requires special knowledge about the current scene nor any marking of enter-and-exit zones and can be extended to any object class which should be tracked. This system is evaluated on tracking single persons with an overall accuracy of 0.749.

In our own previous approach [6], we implement an efficient change detection algorithm to detect pedestrians and vehicles in a parking garage. This algorithm is tailored to the difficulties arising in indoor surveillance scenarios, using an adapted outlier rejection heuristic to enhance the robustness of the foreground detection. The overall mean position error is 0.78 m or less.

Shape-from-Silhouette approaches require the filled 2D silhouettes of relevant objects to be extracted from the camera images. When working with image sequences, these silhouettes can be generated by applying change detection [7]. In the case of static images, semantic segmentation [8] can be employed. Based on the silhouettes of multiple views, the 3D object can be reconstructed using either a surface- or a voxel (*volume element)*-based approach. Surface-based methods generate a polygon or view cone around the area that can potentially be occupied by the object belonging to a certain silhouette. All polygons from different cameras are then intersected to compute the actual 3D volume [9]. Voxel-based algorithms divide the space into discrete elements and decide for each element whether or not an object occupies it by examining if its projection in all camera images lies inside a silhouette.

Typically SfS is applied with a high resolution and a multitude of cameras to reconstruct the shape of a single object as accurately as possible [10], which in most cases does not reach real-time performance. However, it can also be used with lower resolution on larger areas containing several objects, where the 3D reconstruction primarily gives information about the object's position instead of being a detailed model.

A complete pipeline for the estimation of the 3D position of pedestrians is presented by Cheung et. al. [11]. Each used camera is connected to a PC which calculates the silhouette of the moving person and sends the result to a host computer. A voxel-based 3D reconstruction is estimated employing the sparse pixel occupancy test (SPOT) algorithm. Ellipsoids are then fitted to the reconstructed data to visualize the movement of the person. Although the real-time performance is given (15 fps for a combination of five cameras), the approach neither takes the presence of multiple people into account nor the observed area of $2\,\mathrm{m}\times2\,\mathrm{m}\times2\,\mathrm{m}$, which is undersized for the given purpose.

Since detected silhouettes are normally inconsistent due to calibration errors or occlusions, Landabaso et. al. [12] propose a new method to reconstruct 3D objects and their pose from these inconsistent silhouettes. They extend the basic idea of SfS by providing a fast technique to classify parts of the volume which tend to be inconsistent taking into account the 2D detection error probability. The approach reaches a recall of 0.63, which is 20 % higher than the classic SfS approach, and a comparable precision. Unfortunately, the authors did not report on processing time for a single frame.

Using SfS and combining multiple views from different cameras to gain 3D object positions often creates ghost objects, especially when there are multiple real objects in the scene. An illustration of such a situation for two dimensions is given in Fig. 1. These detections primarily occur when silhouettes generated by different objects are used to generate a 3D shape. Michoud et al. [13] introduce a method to remove ghost objects resulting in a more robust tracking. First, they use a statistical approach to enhance the volume reconstruction by calculating a fusion of all captured images, which leads to a compensation of inconsistent silhouettes. In a second step, they identify ghost objects by estimating an SfS probability map, marking segmented pixels which belong to an object.

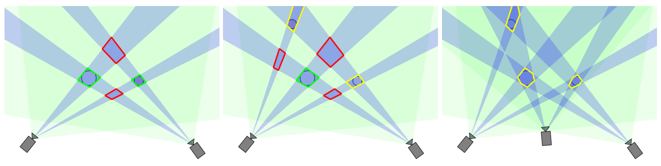
Fig. 1. Illustration of the ghost problem with two cameras and two objects *(left)*, two cameras and three objects *(middle)*, three cameras and three objects *(right)*. *Light blue cones*: Volumes marked as active. *Green outline*: Correct reconstruction. *Red outline*: Ghost object. *Yellow outline*: Reconstruction at the correct position but with distorted shape.

III. ALGORITHM

This section provides an in-depth description of our algorithm starting with the necessary input data in Sec. III-A. Subsequently, an explanation of our SfS and SDD implementations is given with more details on how to integrate occlusion reasoning into those methods (*cf.* Sec. III-B – Sec. III-C). Finally, we explain how the respective results can be fused to obtain the final reconstruction in Sec. III-D.

A. *Input*

Our algorithm requires input in the form of I segmentation images S_i with $I \geq 2$ as well as extrinsic and intrinsic calibration data of cameras C_i. While the SDD can also provide results for areas covered by a single camera (as a special case, SSD can be deployed on a monocular setup), shapes can only be computed from the intersection of at least two view cones.

Additional information about the environment can be provided by two 3D meshes consisting of individual polygons. The first mesh $\mathfrak{M}_S$ describes the static elements of the scene that can lead to occlusions while the second mesh $\mathfrak{M}_N$ marks parts of the ground on which objects can move. Both meshes are optional inputs for the method itself but significantly improve performance when dealing with occlusions. Different steps of the approach are shown in Fig. 2.

B. *Shape-from-Silhouette*

The initial step of the SfS algorithm is the creation of a 3D voxel space. The space should be a cuboid to allow easy indexing of each voxel based on its position. The extent of the cuboid is determined by a minimal bounding box – either spanned around the cameras with the lower z-coordinate being equal to a user-defined offset for the ground plane or around the static scene elements $\mathfrak{M}_S$. The reserved space is then filled with individual voxels. Their side lengths are parameterizable but are usually set to the same value for each dimension, however, different side lengths are possible in our implementation. To determine if a voxel v is segmented based on a segmentation image S_i, the region R_i^v in S_i for each $i = 1,\ldots,I$ that corresponds to v needs to be known. This information is precomputed for each voxel and each camera. First, it is determined if v is visible in S_i by computing the pixel positions $(x_{c,i}^v, y_{c,i}^v)$ for each corner e_c^v with $c = 1,\ldots,8$ of v.

Granted that the static elements $\mathfrak{M}_S$ are available, occlusions can also be taken into account for voxel visibility. A

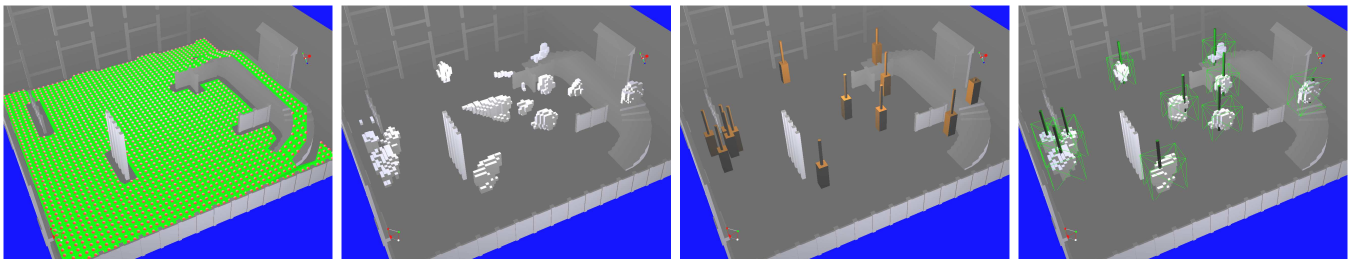

Fig. 2. *From left to right:* Walkable area marked in green with red grid points. Active voxels from the SfS reconstruction. SDD detections. Detections fused with volumes and bounding boxes.

view ray $r^v_{c,i}$ is created between the position of the camera C_i and the voxel corner e^v_c. For each relevant scene polygon $\rho \in \mathfrak{M}_s$ said view ray $r^v_{c,i}$ is tested for an intersection with ρ before reaching the voxel corner e^v_c. In this case e^v_c is occluded by ρ in the image taken by camera C_i.

If any corner e^v_c of v is either outside of the camera image S_i or occluded, the voxel v is considered *not visible* by C_i, which is marked by setting a corresponding variable o^v_i to *true*. Otherwise o^v_i is set to *false* and the convex hull around all reprojected positions $(x^v_{c,i}, y^v_{c,i})$ is constructed to determine which pixels are inside of the hull. To minimize memory consumption, only the first and the last pixel of each pixel row are stored to identify the image region R^v_i corresponding to the voxel v. This setup procedure is listed as pseudocode in Algorithm 1.

Posterior to the setup, every time new segmentation images are available the algorithm has to determine for each voxel v whether or not it belongs to a moving object in the environment. The status can therefore be *active* or *inactive*, which needs to be inferred from the segmentation images. The precomputations during the setup phase allow to directly access the pixels in each segmentation image S_i that corresponds to the current v. Thus, the number of active pixels n^v_i in the image region R^v_i is counted and compared with $|R^v_i| \cdot t_{\text{seg}}$, where $|R^v_i|$ denotes the total number of pixels in that region. t_{seg} is expressed as a relative threshold since $|R^v_i|$ greatly varies depending on the position of v relative to C_i. If n^v_i exceeds $|R^v_i| \cdot t_{\text{seg}}$, v is marked as segmented in C_i, which is denoted by setting s^v_i to *true*. Otherwise it is set to *false*.

The occupancy a_v of v can now be expressed as a function of the occlusion and segmentation flags o^v_i and s^v_i, respectively. In the case of v being occluded in every camera image, no inference can be made about its status and it will always be considered inactive. Otherwise we found

$$a_v = \begin{cases} 1 & \text{if } s^v_i = \neg o^v_i \; \forall i \wedge \exists i : o^v_i = true \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

to be most reasonable for the given scenario. It states that in order to be considered active, a voxel v has to be segmented in exactly those cameras in which v is visible. This function allows to detect superfluous segmentations in single camera images. Missing segmentations, however, might fail to detect a certain object.

If no information on occlusions is available, the occupancy

```
Input:
v ← voxel
i ← camera index
C_i ← camera
S_i ← segmentation image of C_i
𝔐_S ← static mesh (optional)
Result:
R^v_i ← region in S_i
o^v_i ← flag indicating if v is occluded outside of C_i

o^v_i ← false;
R^v_i ← empty;
for c ← 1 to 8 do
    e^v_c ← corner of v;
    (x^v_{c,i}, y^v_{c,i}) ← backprojected pixel position of e^v_c in C_i;
    if (x^v_{c,i}, y^v_{c,i}) not visible in C_i then
        o^v_i ← true;
        break;
    end
    if 𝔐_S is present then
        r^v_{c,i} ← view ray between C_i and e^v_c;
        if r^v_{c,i} intersects any polygon ρ ∈ 𝔐_S then
            o^v_i ← true;
            break;
        end
    end
end
if o^v_i is false then
    R^v_i ← convex hull around all (x^v_{c,i}, y^v_{c,i}), c = 1,...,8;
end
```

Algorithm 1: SfS initial setup as 3D voxel space. The procedure is repeated for each voxel and each camera.

a_v needs to be computed as

$$a_v = \begin{cases} 1 & \text{if } \sum_i (s^v_i) \geq 2 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Since it is uncertain whether the segmentation in a third camera is missing due to occlusion or due to an error, objects are detected as soon as two cameras agree on their existence. Based on the number of cameras recording the scene and the scene's structure, this number can be increased.

After the previous step each voxel occupancy a_v is known and hypotheses for the position and shape of actual objects need to be inferred from the given information. This is done by assigning an object hypothesis to a group of active voxels that are spatially connected, or at least so close together that it is reasonable to assign them to the same object. While this is usually done by clustering active voxels together based on their distance, it is not feasible in our application due to the large number of potentially active voxels.

To overcome these problems, we use a sequential fill procedure which only requires two passes over the entire space. An object hypothesis index h_v is assigned to all voxels v. It is initially set to 0 and the voxels are traversed sequentially. h_v will later denote the affinity of v to a certain object. During voxel traversal, and when a_v is 1, the indices $h_{v'}$ of neighboring occupied voxels v' that have already been visited and are within a certain distance threshold t_{obj} are checked. If no such v' exists, h_v is set to the next higher unused value starting with 1. In case of only a single value for $h_{v'}$ (either because there is only one v' or because all $h_{v'}$ are equal), h_v is set to that value. Otherwise, if multiple values for $h_{v'}$ are found, a clash exists. This means that voxels which should belong to the same object have been assigned different indices – something that needs to be corrected afterwards. To do this, h_v is set to the minimal index $h_{\min} = \min(h_{v'}) > 0$. All clashing indices $h_{v'} \neq h_{\min}$ are inserted into a map Q as $Q(h_{v'}) = h_{\min}$.

After the first traversal the remaining clashes have to be removed. This is done by traversing the space a second time and setting h_v to $Q(h_v)$ for each v of which the corresponding index h_v can be found in Q. Thus, each connected group of voxels ends up with the lowest index that has been assigned to a voxel inside of it. All voxels with the same index are then used to create a new object hypothesis ζ.

C. Sparsity Driven Detector

The SDD implicitly solves the correspondence problem, which leads to ghost detections of the SfS algorithm, by placing template objects on a discrete grid and thereby recreating the segmentation images. The first step of this approach is the selection of appropriate grid points. In the simplest case, the grid points are laid out uniformly over the entire voxel space at its lowest z-coordinate. However, if additional data is provided by the previously mentioned ground mesh $\mathfrak{M}_N$, these grid points can only be created in areas covered by $\mathfrak{M}_N$ at an appropriate height. This procedure eliminates the generally required assumption of a flat ground plane, which many other methods employ.

Based on the grid, the dictionaries can be created. This requires one or more binary templates τ for each detectable object. Besides a binary image showing the template, τ has to be equipped with additional information w_τ and h_τ about the width and height of the displayed object in world coordinates. A dictionary D^τ contains one entry d_j^τ for each grid point $p_j, j = 1,...,J$. A single entry d_j^τ in turn comprises information about how the segmentation images $S_1,\ldots,S_I$ would look like if the template τ would be placed at p_j. d_j^τ therefore consists of I binary images of size $W \times H$ that are stored in a linear fashion in a single column vector. This allows for D^τ to be interpreted as an $M \times J$ matrix with $M = W \cdot H \cdot I$.

To create the entry d_j^τ, first for each camera C_i the ray $r_i^j = p_j - C_i$ is computed and its upward-pointing z-coordinate is set to 0 since we are merely interested in the horizontal direction. The normal to this modified direction vector is denoted as $\hat{r}_i^j$. Next, four points are computed as

$$q_{i,s}^j = \begin{cases} p_j - \frac{w_\tau}{2} \cdot \frac{\hat{r}_i^j}{||\hat{r}_i^j||} + (0,0,h_t)^T & \text{if } s = 1 \\ p_j + \frac{w_\tau}{2} \cdot \frac{\hat{r}_i^j}{||\hat{r}_i^j||} + (0,0,h_t)^T & \text{if } s = 2 \\ p_j + \frac{w_\tau}{2} \cdot \frac{\hat{r}_i^j}{||\hat{r}_i^j||} & \text{if } s = 3 \\ p_j - \frac{w_\tau}{2} \cdot \frac{\hat{r}_i^j}{||\hat{r}_i^j||} & \text{if } s = 4 \end{cases} \tag{3}$$

This ensures that the template is standing upright and facing the camera. A projective transform from the template edge pointing to the projections of the template corners $q_{i,s}^j$ is calculated and used to transform the template image itself. Occlusion reasoning can be included by checking, if the initial ray r_i^j intersects any static scene polygon $\rho \in \mathfrak{M}_s$ and – if it does – by filling the image of ρ in C_i with black, thus removing a part of the previously projected template.

The benefit of describing D^τ as a matrix is the fact that multiplying D^τ with a vector $\mathbf{x} \in \{0,1\}^J$ creates a linear combination of the entries chosen by the non-zero indices of $\mathbf{x}$. The task of solving the correspondence problem can then be formulated as choosing $\mathbf{x}$ such that the segmentation images S_i are recreated as closely as possible:

$$\min_{\mathbf{x}} ||\bar{S} - D^\tau \cdot \mathbf{x}||_0 \tag{4}$$

where $\bar{S}$ denotes the concatenation of all segmentation images S_i. To keep the optimization linear, the L_0-norm is replaced by the L_1-norm and corresponding auxiliary variables u are introduced. The problem can then be formulated as:

$$\min_{\mathbf{x},\mathbf{u}} \quad \begin{bmatrix} \mathbf{x}^T & \mathbf{u}^T \end{bmatrix} \cdot \begin{bmatrix} \mathbf{0} \\ \mathbf{1} \end{bmatrix} \tag{5}$$

$$\text{s.t.} \quad \begin{bmatrix} -D^\tau & -\mathbf{I} \\ D^\tau & -\mathbf{I} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{x} \\ \mathbf{u} \end{bmatrix} \leq \begin{bmatrix} -\bar{S} \\ \bar{S} \end{bmatrix} \tag{6}$$

$$\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix} \leq \begin{bmatrix} \mathbf{x} \\ \mathbf{u} \end{bmatrix} \leq \begin{bmatrix} \mathbf{1} \\ \infty \end{bmatrix} \tag{7}$$

where $\mathbf{I}$ is the identity matrix of size $M \times M$. This can be used as input to a linear solver to estimate object positions all from the same template τ. To extend this method to K different templates, Eq. (6) has to be changed to incorporate multiple dictionaries while ensuring that only one template can be chosen for each grid point:

$$\begin{bmatrix} -D^{\tau_1} & \ldots & -D^{\tau_K} & -\mathbf{I} \\ D^{\tau_1} & \ldots & D^{\tau_K} & -\mathbf{I} \\ \mathbf{I} & \ldots & \mathbf{I} & \mathbf{0}^T\mathbf{0} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_K \\ \mathbf{u} \end{bmatrix} \leq \begin{bmatrix} -\bar{S} \\ \bar{S} \\ \mathbf{1} \end{bmatrix} \tag{8}$$

The original publication [3] further outlines a method to estimate dynamic occlusions between objects by optimizing

$$\min_{\mathbf{x},\bar{L}} \left[||\bar{S} - D^{\tau} \cdot \mathbf{x}||_1 + \beta \cdot ||\bar{L} - \bar{S}||_1 \right], \tag{9}$$

where $\bar{L}$ estimates the number of layers that explain all camera segmentation images $\bar{S}$. This implies that

$$\forall m = 1, ..., M \begin{cases} \bar{L}_m \geq \bar{S}_m & \text{if} \quad \bar{S}_m > 0 \\ \bar{L}_m = 0 & \text{if} \quad \bar{S}_m = 0 \end{cases} \tag{10}$$

β is a weighting parameter that is set to 0.1, following the recommendations of the original publication. The entire optimization problem therefore becomes:

$$\min_{\mathbf{x},\bar{L},\mathbf{u}} \quad \begin{bmatrix} \mathbf{x}^T & \bar{L}^T & \mathbf{u}^T \end{bmatrix} \cdot \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{1} \end{bmatrix} \tag{11}$$

$$\text{s.t.} \quad \begin{bmatrix} -D^{\tau} & (\ \ \beta+1)\mathbf{I} & -\mathbf{I} \\ -D^{\tau} & (-\beta+1)\mathbf{I} & -\mathbf{I} \\ D^{\tau} & (\ \ \beta-1)\mathbf{I} & -\mathbf{I} \\ D^{\tau} & (-\beta-1)\mathbf{I} & -\mathbf{I} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{x} \\ \bar{L} \\ \mathbf{u} \end{bmatrix} \leq \begin{bmatrix} \beta \cdot \bar{S} \\ -\beta \cdot \bar{S} \\ \beta \cdot \bar{S} \\ -\beta \cdot \bar{S} \end{bmatrix}, \tag{12}$$

$$\begin{bmatrix} \mathbf{0} \\ \bar{S} \\ \mathbf{0} \end{bmatrix} \leq \begin{bmatrix} \mathbf{x} \\ \bar{L} \\ \mathbf{u} \end{bmatrix} \leq \begin{bmatrix} \mathbf{1} \\ \infty \cdot \bar{S} \\ \infty \end{bmatrix} \tag{13}$$

where $\infty \cdot \bar{S}$ is meant to be 0 for zero-elements of $\bar{S}$. This can also be extended to multiple templates by adapting Eq. (12) to:

$$\begin{bmatrix} -D^{\tau_1} & \dots & -D^{\tau_K} & (\ \ \beta+1)\mathbf{I} & -\mathbf{I} \\ -D^{\tau_1} & \dots & -D^{\tau_K} & (-\beta+1)\mathbf{I} & -\mathbf{I} \\ D^{\tau_1} & \dots & D^{\tau_K} & (\ \ \beta-1)\mathbf{I} & -\mathbf{I} \\ D^{\tau_1} & \dots & D^{\tau_K} & (-\beta-1)\mathbf{I} & -\mathbf{I} \\ \mathbf{I} & \dots & \mathbf{I} & \mathbf{0}^T\mathbf{0} & \mathbf{0}^T\mathbf{0} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_K \\ \bar{L} \\ \mathbf{u} \end{bmatrix} \leq \begin{bmatrix} \beta \cdot \bar{S} \\ -\beta \cdot \bar{S} \\ \beta \cdot \bar{S} \\ -\beta \cdot \bar{S} \\ \mathbf{1} \end{bmatrix}$$

We refer to those four optimization problems as *single* (Eq. (5)), *multi* (Eq. (8)), *single-layered* (not stated explicitly), and *multi-layered* (Eq. (12)), respectively. After solving, the association between non-zero values of $\mathbf{x}$ or $\mathbf{x}_k, k = 1, ..., K$ to the corresponding grid point and template has to be made to recreate the actual object class and position $\vartheta^n = \{\vartheta^n_c, \vartheta^n_p\}, n = 1, ..., N$. In [3], the optimization problems described here are embedded into a network-flow data association framework to ensure temporal consistency of the detections. The local detection problem as well as the global network-flow problem have to be solved iteratively. This means, however, that the entire image sequence needs to be available to the algorithm at once and online usage is not possible.

D. Fusion

Indeed we found that the solutions to the individual optimization problems sometimes place multiple objects in close proximity to each other when the silhouettes in the images can be explained better this way. Additionally, sometimes false positive detections are produced. But instead of employing a global optimization scheme, we propose local frame-based methods to improve the correctness of the results.

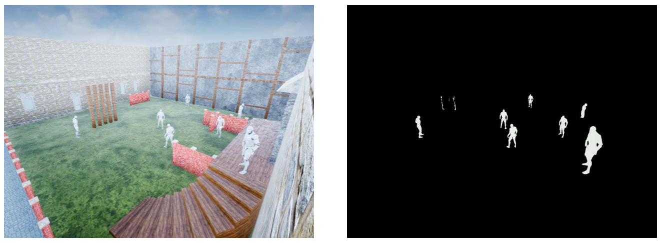

Fig. 3. Example of virtual experiment input from the *occluded* scenario. Both images were generated with Unreal Engine 4 *Left:* Virtual camera image of sample test environment. *Right:* Corresponding binary segmentation image.

First, all distinct object detections ϑ^{n_1} and ϑ^{n_2}, $n_1 \neq n_2$ are merged to a new detection if

$$\vartheta^{n_1}_c = \vartheta^{n_2}_c \quad \text{and} \quad ||\vartheta^{n_1}_p - \vartheta^{n_2}_p|| < t_{\text{merge}},$$

where t_{merge} is a user-defined merging threshold. The resulting detection has the same object class as the individual merged detection and is located at its mean position (and is not longer restricted to the discrete positions). This eliminates all false positives due to multiple detections at the same location.

Now all remaining object detections ϑ^n are assigned to a corresponding SfS object hypothesis ζ^z if ϑ^n_p is inside the bounding box of ζ^z. Multiple ϑ^n can be assigned to the same ζ^z (the inverse is not possible due to the voxel clustering process). In this case ζ^z is split into several smaller object hypotheses by assigning its voxels each to the closest ϑ^n. Afterwards, all SDD detections ϑ^n without a corresponding SfS detection ζ^z are discarded as false positive. Likewise, all voxels – and, thus, all clusters ζ^z – that are outside of the range of all ϑ^n are removed and only the final object hypotheses remain.

IV. EXPERIMENTS

The following experiments have been conducted to evaluate different aspects of the presented approach. In order to receive accurate and very precise evaluation results, we use synthetic image data including automatically generated ground truth for all experiments. The data generation process is set out in Sec. IV-A, while Sec. IV-B elaborates the chosen test scenarios and our reasoning behind them. The respective results are given in detail in Sec. IV-C.

A. Data Generation

When it comes to data generation, the use of artificial image data is advantageous compared to real-world images in many respects. For our experiments we therefore opted for Unreal Engine 4[2], currently one of the major open-source game engines, to create all necessary data. This includes image data, ground-truth annotation, as well as metadata regarding the virtual environment.

The input data required for our approach are binary segmentation images similar to those produced by background subtraction algorithms. While these images would normally

[2]`https://www.unrealengine.com`

result from pre-processing camera images, *e.g.*, by certain object detection algorithms or foreground segmentation, we opt for artificial image data as our sole input. By leaving the image segmentation process out of our scope of work, illumination changes and other challenges that occur in the segmentation task, can be set aside not being part of our approach. On a side note, however, one must take into account that the quality of the segmentation images has a huge impact on the performance of our method, which is why we abandon natural image data for our proof of concept. Instead, we can fully profit from the pixel-perfect and fully automatic annotation of synthetic image data (*cf.* Fig. 3), as only the resulting binary segmentation image is used as input to our algorithm.

With the help of a self-defined render mode in Unreal Engine, these segmentation images can easily be captured from regular levels. Fig. 3 shows one of our experiment setups, in the following referred to as "*occluded*", and the recorded corresponding binary segmentation image. Apart from pixel-perfect segmentation results, the use of a simulated environment provides full control over every detail of the data creation. The definition of walkable areas and a waypoint system direct a walking character's movements such that the same scene is perfectly reproducible, as required for at least one of our experiments (see Sec. IV-B). At the same time, a less controlled manner can be implemented as well, should a randomness of movement be required. Furthermore, the exported metadata can be adapted to export exactly the required information in a custom format.

B. Setup

We test our implementation on three different scenarios all displaying a 20 m × 20 m environment. The first scenario – "*one-person*" – consists of a single person moving around in a flat area without occlusions and provides images captured by three cameras. Scenario "*multi-person*" shows the same space with 5 moving persons and is filmed by 4 cameras. Here, we also vary the camera height between 2 m and 6 m with increments of 1 m to test the influence on the localization quality. The "*occluded*" scenario adds static occlusions and non-flat walkable areas. It contains a total of 10 moving persons, which are filmed by 3 cameras at 6 m height.

For evaluation we consider a ground-truth position detected correctly, if it lies within the bounding box of a detection. All ground-truth positions without a corresponding detection are regarded as false negative and all detections without a ground-truth position as false positive, which allows to compute precision and recall of our method. Additionally we report on e_{mean}, the mean distance between a detection and its ground-truth point. For this calculation a detection is represented by the mean position of all corresponding active voxels. Only the horizontal component of the distance is considered for evaluation since all actors are assumed to be vertically anchored on the floor.

Our implementation exposes several parameters, which can influence accuracy and speed of the overall method.

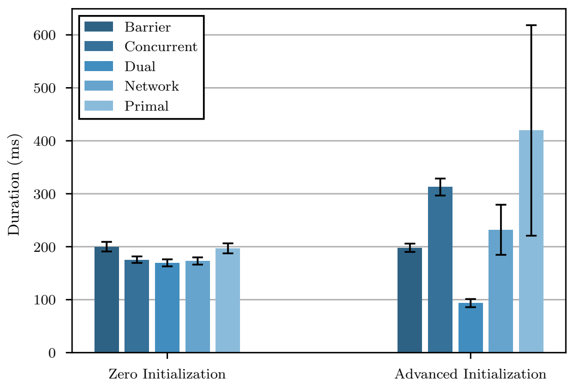

Fig. 4. Timing comparison of the different root algorithms provided by CPLEX on the scenario *one-person* with and without advanced initialization.

These are:

d_g	the horizontal distance of grid points in cm,
s_v	the side length of each voxel in cm,
W,H	the width and height of the images used for optimization in pixels,
t_{merge}	the distance threshold for merging multiple detections in cm, and
t_{segm}	the segmentation threshold as a factor between 0 and 1 to mark a voxel as active.

Furthermore, it is possible to choose between the 4 optimization types described in Section III. We use the same 2 templates for pedestrians as proposed in [3], with the closed-legs template being omitted for optimization types *single* and *single-layered*.

Shape-from-Silhouette object hypotheses are computed on a Quadro M1000M-GPU (993 MHz, 2 GB RAM) with NVIDIA CUDA[3]. The occupancy of a single voxel is independent of its neighboring voxels, which allows for straightforward parallelization. Only clustering is executed on the CPU. Our implementation of the SDD's optimization problem uses IBM's ILOG CPLEX[4] C++ interface. This provides two additional parameters – the *root algorithm*, which can be set to "*Barrier*", "*Concurrent*", "*Dual*", "*Network*", or "*Primal*", and a choice of whether or not the solution of the previous frame should be used as an advanced initialization of the current frame. In our experiments we used an Intel i7-6820HQ CPU, with 2.7 GHz and 16 GB RAM.

C. Results

Unless stated otherwise, we use the image size $W \times H = 160 \times 120$ pixel. In our first experiment we investigate the influence of the *root algorithm* and optional activation of advanced initialization.

We set the parameters of our algorithm to

$$\{d_g = 40, \quad s_v = 30, \quad t_{\text{merge}} = 60, \quad t_{\text{segm}} = 0.5\}$$

and execute the pipeline with optimization type *single* on scenario *one-person* with all ten possible CPLEX configurations. The timings depicted in Fig. 4 show that it is optimal to

[3]`https://developer.nvidia.com/cuda-zone`
[4]`https://www.ibm.com/analytics/optimization-modeling-interfaces`

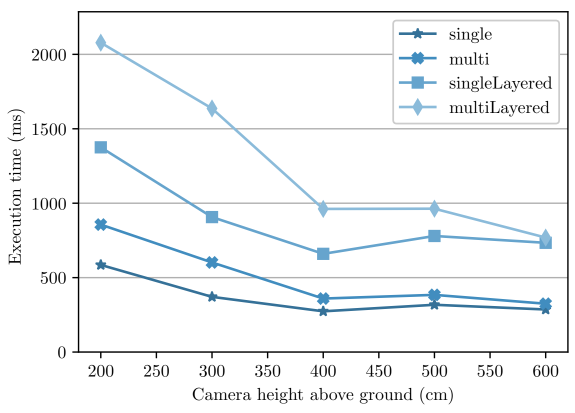

Fig. 5. Mean execution time of all four presented approaches for an image size of 160 × 120 and different camera heights between 2 m and 6 m

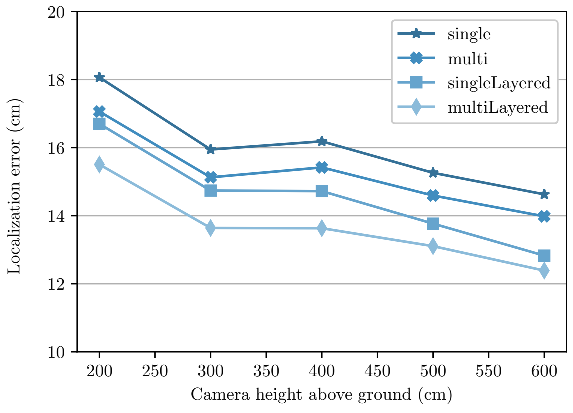

Fig. 6. Mean localization error of all four presented approaches for an image size of 160 × 120 and different camera heights between 2 m and 6 m

use the *Dual* simplex algorithm with advanced initialization, which results in an average of 91 ms per optimization per frame and, thus, is used in all further tests. The detected object positions were almost identical for each variant.

The same scenario is used to estimate appropriate values for d_g, s_v, t_{merge}, and t_{segm} – again with optimization type *single*. Table I shows the selected values and corresponding results. We executed all tests for $s_v = 20$ and $s_v = 30$ but found only negligible differences with setup times being slightly longer for $s_v = 20$. The single actor is found in each frame and in rare cases superfluous detections are made. When changing the grid distance from $d_g = 10$ to $d_g = 20$, the time necessary for computation decreases by roughly 100 ms, however e_{mean} increases by only 1.4 cm. Therefore we use

$$\{d_g = 20, \quad s_v = 30, \quad t_{\text{segm}} = 0.05\}$$

and examine the influence of t_{merge} on more complex scenes.

We explore the *multi-person* scenario and – in addition to camera height – also use all four optimization types T_{optim} with $t_{\text{merge}} = 40$ and $t_{\text{merge}} = 100$. Additionally, we test the effect of using a smaller image size of 80 × 60 pixel for the dictionary, which leads to a total of 80 different configurations. Table II shows the results of all experiments for a camera height of 2 m. The best results for each other camera height and both image sizes, respectively, are presented as well. We learn that the larger merging threshold generally performs better. The runtime is influenced by both camera height and optimization type. Apparently, lower camera heights result in problems that are harder to optimize, and adding layer estimation requires at least double the execution time. This relationship is illustrated in figures 5 and 6. Reducing the image size leads to many false positive detections for low camera heights but also greatly improves the runtime approaching real-time capability for scenarios with cameras at greater heights. All experiments yield a recall of 99 % or higher. So the main problem are false positive detections.

For the *occluded* scenario we set $t_{\text{merge}} = 100$ and test both image sizes for a camera height of 6 m. The results are displayed in Tab. III. Larger image sizes and complex versions of the optimization problem clearly lead to more accurate localization. At the cost of ∼ 4 cm accuracy and one percentage point of precision, real-time capability is achieved. In comparison to Tab. II, the shorter runtime suggests that adding occlusions simplifies the optimization problem since less grid points have to be considered for recreating the silhouettes of the input images.

TABLE I

RESULTS OF OUR SYSTEM FOR THE LOCALIZATION OF A SINGLE PERSON. RESULTS ARE IDENTICAL FOR $s_v = 20$ AND $s_v = 30$.

d_g	t_{merge}	t_{segm}	e_{mean} (cm)	Precision	Time (ms)
10	100	0.05	14.105	0.999	266.80
10	100	0.50	14.105	0.999	256.43
10	40	0.05	14.284	0.982	254.08
10	40	0.50	14.273	0.982	253.99
20	100	0.05	15.474	1.000	155.13
20	100	0.50	15.596	0.981	155.40
20	40	0.05	15.474	1.000	152.57
20	40	0.50	15.596	0.981	165.79
30	100	0.05	16.571	1.000	125.02
30	100	0.50	16.793	0.986	133.11
30	40	0.05	16.571	1.000	130.61
30	40	0.50	16.793	0.986	134.06
40	100	0.05	20.805	0.997	118.25
40	100	0.50	20.805	0.997	119.40
40	40	0.05	20.805	0.997	116.20
40	40	0.50	21.514	0.876	118.61

V. CONCLUSION

In this paper we present a flexible algorithm for 3D multi-object localization which runs online and exhibits real-time capability for certain camera and parameter setups. While cameras mounted at higher locations are generally preferable, even for cameras at 2 m height a good precision and mean localization error is reached at the cost of longer runtime. The reason for the increased runtime most likely is the similar appearance of binary templates placed at neighboring grid points when viewed at a low angle. In this case fewer positions can be easily ruled out and the optimization problem becomes more complex.

For the general application of SfS higher cameras are also preferable. Cameras oriented parallel to the ground plane can lead to significantly more plausible ghost detections

TABLE II
RESULTS FOR THE *Multi-Person* SCENE. FOR CAMERA HEIGHTS ABOVE 2 M ONLY THE BEST CONFIGURATIONS ARE SHOWN.

Height	$W \times H$	T_{optim}	t_{merge}	e_{mean}	Prec	Time (ms)
200	160×120	*m.-l.*	100	15.505	0.861	2078.34
200	160×120	*m.-l.*	40	15.591	0.743	1172.01
200	160×120	*s.-l.*	100	16.700	0.883	1375.37
200	160×120	*s.-l.*	40	16.599	0.804	808.64
200	160×120	*m.*	100	17.060	0.937	856.28
200	160×120	*m.*	40	17.106	0.915	470.43
200	160×120	*s.*	100	18.065	0.971	585.47
200	160×120	*s.*	40	18.124	0.958	329.29
200	80×60	*m.-l.*	100	17.344	0.802	357.20
200	80×60	*m.-l.*	40	19.385	0.552	212.96
200	80×60	*s.-l.*	100	17.617	0.821	253.28
200	80×60	*s.-l.*	40	18.829	0.613	249.57
200	80×60	*m.*	100	19.393	0.840	212.93
200	80×60	*m.*	40	21.064	0.631	115.34
200	80×60	*s.*	100	19.888	0.876	127.28
200	80×60	*s.*	40	20.946	0.754	126.99
300	160×120	*s.*	100	15.945	0.997	370.85
300	80×60	*s.*	100	17.538	0.951	89.92
400	160×120	*s.*	100	16.185	0.999	273.72
400	80×60	*s.-l.*	100	15.953	0.977	159.41
500	160×120	*m.*	100	14.593	0.999	384.19
500	80×60	*s.-l.*	100	14.942	0.991	178.23
600	160×120	*m.*	100	13.981	1.000	324.70
600	80×60	*m.-l.*	100	14.110	0.998	159.95

TABLE III
RESULTS FOR THE *Occluded* SCENARIO WITH $t_{\text{MERGE}} = 100$ AND FOUR CAMERAS AT 6 M HEIGHT.

$W \times H$	T_{optim}	e_{mean}	Prec	Rec	Time (ms)
160×120	*multi-layered*	13.187	0.997	0.999	288.20
160×120	*single-layered*	13.955	0.996	0.999	238.48
160×120	*multi*	14.529	0.998	0.999	126.08
160×120	*single*	15.313	0.999	0.999	101.38
80×60	*multi-layered*	15.564	0.988	0.998	76.17
80×60	*single-layered*	16.003	0.989	0.998	66.60
80×60	*multi*	17.165	0.985	0.998	47.90
80×60	*single*	17.674	0.988	0.998	39.78

compared to cameras positioned at a steep angle. In our system however, these false detections are ruled out by the SDD. Only when SfS and SDD both produce a false positive object hypothesis at the same position a wrong detection is created.

We investigated the localization performance for scenes containing only pedestrians. Future work will have to consider differently shaped objects. In the context of autonomous driving, cars would be an important extension – especially their variable appearance w.r.t. the camera's viewing angle. These additional objects would only require respective templates to be provided during dictionary creation of the SDD. The optimization problem however might become considerably more complex depending on the number of templates which in turn reduces the achievable framerate.

The method described in this paper is single-frame-based and employs temporal integration only implicitly by reusing the solution of the optimization problem of the previous frame. However, classic tracking methods like the Kalman filter could be used to improve the precision and accuracy [14]. It might also be possible to incorporate other ideas of [3] and only allow specific grid points to be used as entry or exit points where object tracks can be created or destroyed.

Besides camera height our method provides several options for performance tuning. The distance of the grid points can be adjusted on the desired trade-off between localization error and processing time. The walkable area can be adapted so that impossible object positions are ruled out beforehand. A few measures however can be taken to further improve the runtime of our implementation. The discrepancy between $t_{\text{merge}} = 40$ and $t_{\text{merge}} = 100$ suggests optimization potential in the merging step. The volumes computed by SfS can be seen as the convex hull of all objects that can potentially be in the environment. Therefore, they might be used to select plausible grid points for the SDD to consider. It is worth investigating if the time saved by solving a smaller optimization problem outweighs the time needed to adapt the problem in each frame – especially for low camera heights.

REFERENCES

[1] Z. Wu, N. I. Hristov, T. L. Hedrick, T. H. Kunz, and M. Betke, "Tracking a large number of objects from multiple views," in *Proceedings of the IEEE International Conference on Computer Vision*, 2009, pp. 1546–1553.

[2] Z. Wu, A. Thangali, S. Sclaroff, and M. Betke, "Coupling detection and data association for multiple object tracking," in *Prodceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 1948–1955.

[3] Z. Wu and M. Betke, "Global optimization for coupled detection and data association in multiple object tracking," *Computer Vision and Image Understanding*, vol. 143, pp. 25–37, 2016.

[4] L. Guan, J.-S. Franco, and M. Pollefeys, "Multi-object shape estimation and tracking from silhouette cues," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2008, pp. 1–8.

[5] M. Bredereck, X. Jiang, M. Korner, and J. Denzler, "Data association for multi-object tracking-by-detection in multi-camera networks," in *Distributed Smart Cameras*, 2012, pp. 1–6.

[6] M. Michael, C. Feist, F. Schuller, and M. Tschentscher, "Fast change detection for camera-based surveillance systems," in *Proceedings of the IEEE International Conference on Intelligent Transportation Systems*, vol. 19, 2016, pp. 1–8.

[7] N. Goyette, P. Jodoin, F. Porikli, J. Konrad, and P. Ishwar, "Changedetection.net: A new change detection benchmark dataset," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2012, pp. 1–8.

[8] B. De Brabandere, D. Neven, and L. Van Gool, "Semantic instance segmentation with a discriminative loss function," *arXiv: 1708.02551*, 2017.

[9] J.-S. Franco and E. Boyer, "Exact polyhedral visual hulls," in *British Machine Vision Conference*, vol. 1, 2003, pp. 329–338.

[10] K. M. Cheung, S. Baker, and T. Kanade, "Shape-from-silhouette of articulated objects and its use for human body kinematics estimation and motion capture," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2003.

[11] K. M. Cheung, T. Kanade, J.-Y. Bouguet, and M. Holler, "A real time system for robust 3d voxel reconstruction of human motions," in *Computer Vision and Pattern Recognition*, 2000, pp. 714–720.

[12] J.-L. Landabaso, M. Pardàs, and J. R. Casas, "Shape from inconsistent silhouette," *Computer Vision and Image Understanding*, vol. 112, no. 2, pp. 210–224, 2008.

[13] B. Michoud, E. Guillou, H. M. Briceño, and S. Bouakaz, "Silhouettes fusion for 3d shapes modeling with ghost object removal," 2008.

[14] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transaction of the ASME Journal of Basic Engineering*, no. 82, pp. 35–45, 1960.