# Sparse and Precise Reconstruction of Static Obstacles for Real-Time Path Planning in Human-Robot Workspaces

Tobias Werner, Maximilian Sand, and Dominik Henrich
robotics.uni-bayreuth.de, Universität Bayreuth, Bayreuth, Germany

## Abstract

Recent advances in robotics aim at applying the benefits of industrial manipulators to use cases in smaller enterprises and the service sector. In these use cases, no a priori geometry is available for obstacles in the workspace. State-of-art approaches thus employ sensors for a real-time reconstruction of obstacles in the workspace and subsequently avoid collisions through real-time path planning. Usually, reconstruction exposes obstacles to path planning over a copious data structure (e.g. an occupancy grid). This data structure, however, is not a good fit for common static and piece-wise planar obstacles (e.g. tables, shelves). We therefore contribute a novel, three-part approach: At first, we create an a priori, sparse and precise reconstruction of static, piece-wise planar obstacles in form of a boundary representation through a hand-held depth sensor. Subsequently, we combine our sparse reconstruction with an online reconstruction of dynamic obstacles to finally enable online path planning. Our experiments show two main benefits of our contribution: Improved reconstruction precision and reduced execution times for collision queries in the path planner. Both of these allow us to increase the motion speed of the manipulator, for empty and for occupied workspaces.

## 1 Introduction

Over past decades, robot manipulators have already proven their worth in industrial applications. Future use cases in small enterprises and the service sector can also benefit from the strength, endurance, and precision of a robot. Yet, the transition from traditional use cases to future ones poses several challenges, from intuitive robot programming to safety mechanisms. One notable challenge concerns the environment of the robot: Use cases in small enterprises and the service sector envision flexible and shared human-robot workspaces. Shared workspaces imply a priori unknown environments, where there are no CAD models for static workspace obstacles (e.g. tables), and where certain obstacles even are non-deterministic (e.g. humans).

To solve the challenge of unknown environments, state-of-art research proposes a two-part and real-time process: A perception part employs sensors (e.g. with a multi-camera setup [11]) to derive a reconstruction of static and dynamic obstacles in the environment of the robot. A concurrent path planning part evaluates the reconstruction (e.g. with Rapidly Exploring Random Trees (RRTs) [15]) to find a collision-free robot trajectory.

Data exchange between both parts commonly uses naive representations (e.g. voxel grids [1]). Those are intuitively built from pixelwise sensor input and are intuitively queried in the path planning part. However, naive representations require a large number of primitives (e.g. many voxels) to represent obstacles with adequate detail. This enforces a trade-off between reconstruction quality and execution time. Obstacles therefore are often reconstructed as coarse shapes in order to satisfy real-time demands. Coarse shapes in turn limit available robot paths and tolerable robot speed.

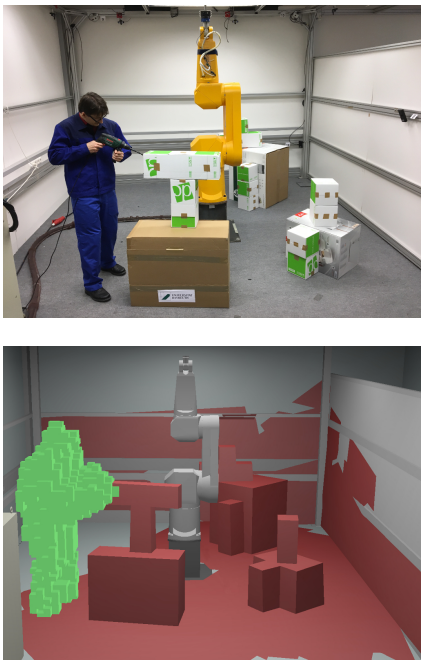

**Figure 1** Top: Robot workspace with mock-up static obstacles and human worker. Bottom: Sparse and precise reconstruction of static, a priori unknown obstacles (red). Coarse and copious octree reconstruction of dynamic obstacles (green). Separate and a priori CAD models for robot and work cell (grey).

## 2 Our approach

Opposed to state-of-art, we contribute a novel three-part process: In a prior offline reconstruction part, we employ a hand-held mobile device (e.g. a smartphone) with a depth camera to gather a sparse and precise boundary represen-

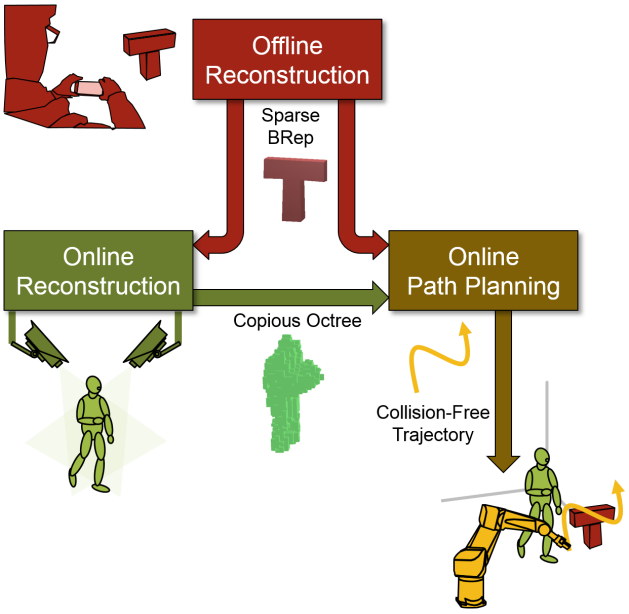


**Figure 2** Overview over our three-part process.

tation (B-Rep) of static, piece-wise planar obstacles in the workspace. A subsequent real-time reconstruction part considers these B-Reps to restrict a coarse and copious octree reconstruction to dynamic obstacles. Finally, a blackboard-based path planner part evaluates the sparse B-Rep and the copious octree representation by efficient hierarchical distance queries for collision-free, speed-controlled robot trajectories. **Figure 1** shows both of our reconstructions, while **Figure 2** gives an overview over our process.

## 2.1 Offline Reconstruction

The goal of the offline reconstruction part is to enable the intuitive acquisition of static obstacles for subsequent use during later online reconstruction and path planning parts. To this end, offline reconstruction utilizes a smartphone with integrated depth camera (Lenovo Phab 2 Pro). Since this smartphone unifies depth camera, display, and computer in one device, no additional devices are required for reconstruction. At the same time, it is possible to preview the results of obstacle acquisition on the integrated screen. This enables even non-experts to efficiently and intuitively perform the reconstruction. See **Figure 3** for an example.

Existing systems for the reconstruction of 3D models with hand-held cameras fall into one of two categories: frame-to-frame methods or frame-to-model methods. The former create a pose graph that represents the spatial relation of single measurements (e.g. [4, 8]), refine that graph through loop closing and graph optimization, and fuse measurements into a global model but in a concluding step. The latter perpetually integrate individual measurements into a global model (e.g. a surfel model [7, 25] or a volumetric model [16, 24, 6]).

When comparing both categories, frame-to-frame methods suffer from maintaining all individual measurements (e.g. point clouds) over the entire acquisition process. Due to memory requirements, this may especially be problematic on memory-limited mobile devices. In contrast, frame-to-model methods avoid this issue by fusing measurements into the global model instantly. Measurements therefore become expendable intermediately and are discarded soon after. However, two notable disadvantages incur for frame-to-model methods: On the one hand, the global model is only an approximation of measurements, which requires balancing precision and memory consumption (e.g. for the subdivision into voxels or surfels). On the other hand, a Graphics Processing Unit (GPU) capable of accelerating general computations becomes a necessity for a real-time reconstruction. Recent smartphones (e.g. the Lenovo Phab 2 Pro), opportunately, are outfitted with a suitably fast GPU for offloading the workload of frame-to-model methods.

**Figure 3** Acquisition of static obstacles through a smartphone.

Following the above assessment, we implement a frame-to-model approach. In our frame-to-model approach, we employ planar B-Reps to represent the global model. The B-Reps apply topological relations between mathematical entities of suitable boundaries to represent model contents. Notably, B-Reps define each individual face of a model by an analytical description of the face surface and by intersections between this surface and neighbouring surfaces — a memory-efficient and coarse, yet precise representation of obstacles. Subsequent sections discuss how we derive a global B-Rep model from individual point clouds.

### 2.1.1 Preprocessing

Offline reconstruction starts with a preprocessing step. We perpetually convert incoming point clouds from the depth sensor on the smartphone into an organized format, namely a 2D indexed grid. This allows for a more streamlined and efficient later reconstruction, since neighbours of a 3D point are intuitively accessed through the 2D grid.

To organize a point cloud into a 2D grid, we apply the a-priori known, intrinsic camera matrix to project original, unorganized points onto 2D indices. Distortions (e.g. due to lens effects) may result in 2D indices without a match in the unorganized point cloud. This causes holes in the 2D grid, which we close by linearly interpolating neighbouring points around each hole.

In addition to organizing the point cloud, preprocessing also determines the current pose of the smartphone through its on-board positioning sensors. Preprocessing annotates the organized point cloud with the determined smartphone pose, resulting in a point cloud that is absolutely located in 3D space. See **Figure 4** for example point clouds.

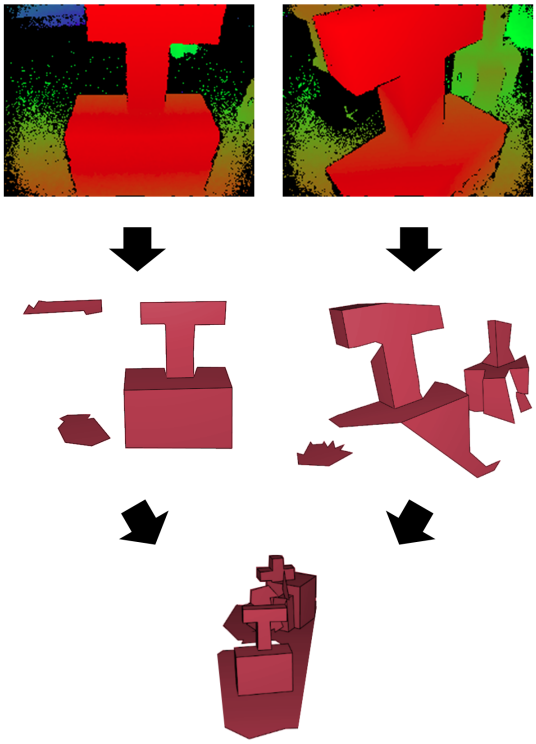

**Figure 4** Top: Two organized point clouds (coloured by $z$ value). Mid: Partial B-Rep models built from individual measurements. Bottom: Fusion of both partial models into the global model.

#### 2.1.2 Incremental Reconstruction of B-Rep Models

After preprocessing, offline reconstruction continues by instantly creating a partial B-Rep from each point cloud. To this end, we use a state-of-art conversion approach (see [18]): First, we tessellate the organized point cloud into intermediate triangles by exploiting point neighbourhoods. The edges of those intermediate triangles define a depth-correct neighbourhood relation over points, which in turn forms the foundation for segmentation of the point cloud by region growing. Region growing yields planar segments. We convert these segments to 2D polygons by finding and fitting contours between neighbouring segments. Finally, we create the desired partial B-Rep by filling a half-edge data structure with the contours of all 2D polygons. See **Figure 4** for an example partial B-Rep.

Directly after building a partial B-Rep, we fuse this partial B-Rep into a global B-Rep model. To do so, we consider pose annotations from preprocessing to find corresponding surfaces in the partial and the global B-Rep. A concluding 2D plane sweep merges the partial B-Rep into the global one. Certainty weights ensure that outliers and other errors do not corrupt the existing global model. **Figure 4** shows the results of merging two partial B-Rep models. Details on B-Rep merging are found in [18].

With an increasing number of partial B-Reps merged into the global model, the global model becomes both more complete and more precise. A real-time model preview on the smartphone enables users to decide whether the global model already meets their requirements. Thus, users may either continue reconstructing or stop and store their final model. A finished model is illustrated in **Figure 5**.

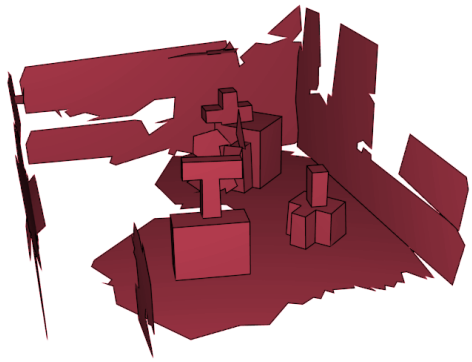

**Figure 5** Global B-Rep model after offline reconstruction.

#### 2.1.3 Model to Robot Calibration

After offline reconstruction, users must perform a quick post-processing step: The final B-Rep must be calibrated to the robot work cell, which involves transforming the model to the robot coordinate system. We suggest performing the calibration by hand in a 3D graphics and modelling suite. This process is both convenient and intuitive, as floors and walls in the offline reconstruction easily align to X, Y, and Z axes. Finally, the user must manually delete any surfaces of the robot that are contained in the offline reconstruction. This is facilitated by common-place tools to select linked faces, which require but a single click to select the entire robot in the reconstruction.

### 2.2 Online Reconstruction

In contrast to the offline reconstruction part, our online reconstruction part uses an intrinsically and extrinsically calibrated network of colour cameras to monitor a shared robot workspace in real-time. Our reconstruction efficiently derives a precise 3D representation of workspace obstacles, which in turn forms the foundation for path planning.

Existing solutions to same task come in several variants: Single-view approaches (e.g. [17]) are readily available for purchase, but suffer from occlusions and thus limit robot paths. Multi-view variants differ by the form of result (e.g. as implicit [2] or explicit [12] geometry), by domain (e.g. as binary [12] or probabilistic [26] obstacles), by algorithm optimizations (e.g. hierarchical [19] or incremental [1]), and, finally, by traits (e.g. a guarantee of conservativeness [9] or knowledge-based refinement [10]). All multi-view variants are robust against occlusions, but require thorough optimizations in order to be both efficient and precise.

In terms of the above assessment, our multi-view online reconstruction [21] produces explicit geometry with binary or, optionally, probabilistic obstacles, uses incremental and hierarchical optimizations, and supports conservativeness and knowledge-based refinement. This section provides a short overview over our online reconstruction approach.

From a high-level point of view, the online reconstruction consists of two separate steps: A computer network runs distributed foreground-background segmentation on new camera images. Concurrently, a server computer gathers resulting silhouette images and merges these to derive the explicit 3D reconstruction. **Figure 6** depicts this process.

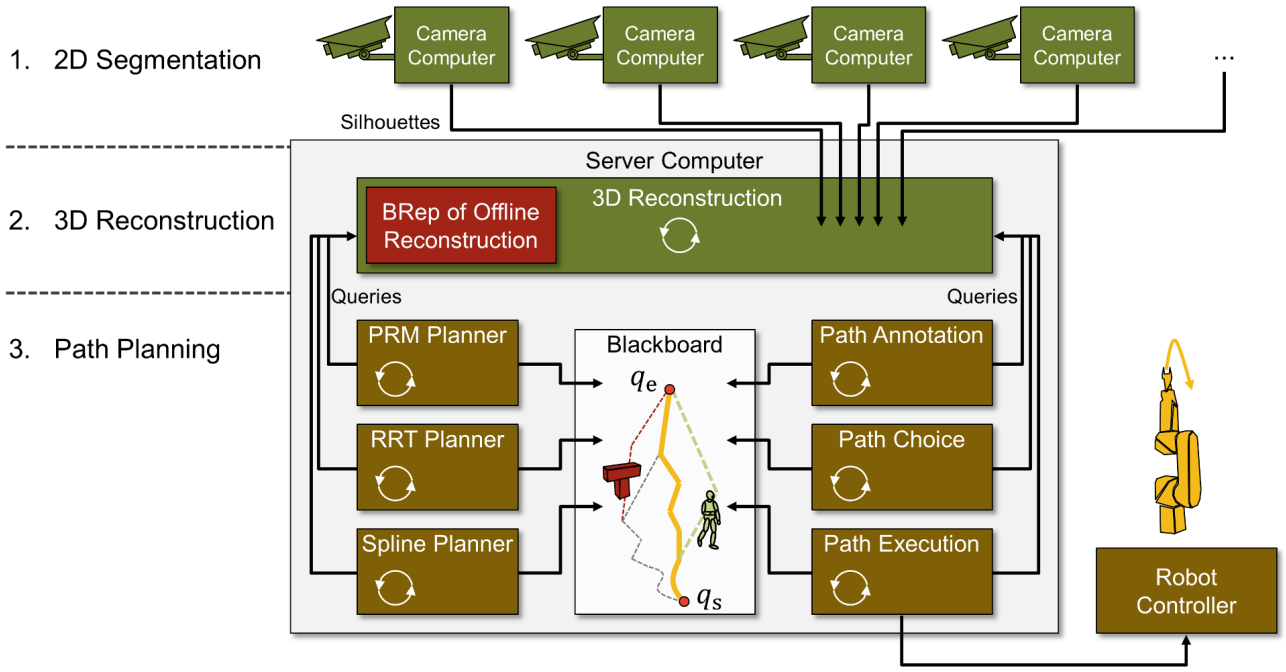


**Figure 6** System architecture of the online reconstruction (green) and path planner (brown) parts: Camera PCs perform distributed foreground segmentation. Resulting silhouettes perpetually update the current 3D reconstruction on the server PC. Individual path planners (e.g. PRM, RRT, Spline) concurrently query the 3D reconstruction to exchange path suggestions from start $q_s$ to goal $q_g$ over the blackboard. Path annotation synchronizes existing suggestions to the 3D reconstruction. Finally, path choice marks paths to be sent to the robot controller through the path execution component. The static B-Rep (red) of the offline reconstruction part is considered both by the 3D reconstruction and by queries from planner components. In the end, colliding paths on the blackboard may be discarded either due to static (left, red path) or dynamic (right, green path) obstacles.

### 2.2.1 Online Preprocessing on Camera Computers

Each camera of the multi-camera system is connected to a dedicated computer on the computer network. This setup is required to cope with the demands in both computation time and peripheral bandwidth for large-resolution (e.g. $1920 \times 1080$ pixels Full HD) camera images.

For actual foreground-background segmentation on each dedicated computer, we apply two neural networks: One neural network performs state-of-art per-pixel foreground detection through an optimized variant of [3]. The other neural network exploits a-priori knowledge of contiguous obstacles by accenting spatial coherence of neighbouring pixels. An adaptive online learning strategy for both networks reduces false-negative detection for static obstacles in favour of false-positives close to slow-moving obstacles. Initial background learning for both networks requires few obstacle-free startup frames. To avoid recording the robot manipulator as background, the robot must be excluded from learning through its CAD model, and the robot must change poses during initialization to allow for capturing the background behind the robot. Later online learning keeps the neural networks in synch with gradual changes in background (e.g. due to changes in lighting conditions or colour drift in the cameras). Finally, we exploit the graphics hardware in each camera computer through an optimized GPU implementation of both networks to achieve real-time segmentation for named high-resolution images. We conclude by optionally binarizing probabilistic segmentation results to silhouettes for a binary reconstruction output. See [22] for more details on our preprocessing.

Apart from foreground-background segmentation, camera computers concurrently create occlusion masks for the current robot pose from a CAD model of the robot. Occlusion masks are silhouette images that mark all camera pixels as occupied by the robot in a given pose. Occupied pixels in an occlusion mask have to be treated differently from background or foreground pixels: The background model of the neural networks can not reliably capture the robot manipulator as background without an exhaustive learning phase and excessive false-negatives on any robot-coloured obstacles. An incomplete background model, in turn, may lead to false-positive detection of the manipulator as an obstacle, which prohibits further robot movement. To enable correct behaviour, we use a simple hue colour model (plus threshold) over a uniform colour of the robot manipulator to determine foreground state for pixels occupied by the robot. Mind that later 3D reconstruction must also check occlusion masks for correct behaviour on any workspace regions that are occluded by the manipulator in regards to the view of the respective camera.

Preprocessing maintains both the current binary silhouette and a matching occlusion mask for concurrent queries by the actual 3D reconstruction. Pull semantics (as opposed to an obligatory push from camera computers) cut down on network traffic, as 3D reconstruction typically runs at a slightly slower rate than preprocessing. Compressing both silhouettes before transfer further saves network bandwidth at a negligible calculation overhead.

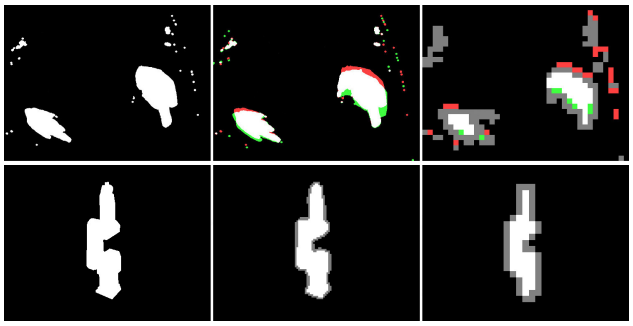

**Figure 7** Quadtrees as an input for the online reconstruction part. Top, left to right: Silhouette image with foreground (white) and background (black). Silhouette with marks for pixels that are now foreground (green) or now background (red). Coarse quadtree level with mixed content (grey). Bottom, left to right: Original occlusion mask for current robot pose. Mid-range quadtree level with mixed content. Coarse quadtree level.

### 2.2.2 Online Reconstruction on Server Computer

Actual 3D reconstruction from silhouettes and occlusion masks executes on the server computer. Reconstruction runs at a variable rate (e.g. depending on the amount of occupied regions in the work cell). This allows for higher-fidelity path planning in low-load situations. At the start of each reconstruction frame, the server concurrently pulls a series of camera images from the camera computers for the next reconstruction frame. Once a reconstruction frame has finished, current results are exposed to path planning instead of preceding results.

Building a 3D reconstruction for each frame involves an incremental and hierarchical process: We first derive quadtrees from all incoming images, then use these quadtrees to efficiently and incrementally derive a copious 3D octree as a representation of workspace occupation.

Quadtrees iteratively merge all silhouette pixels to larger regions that either have foreground, background, or mixed content, down to a single region as the quadtree root. For occlusion masks, quadtrees apply a similar notion, where regions are either robot free, robot-occupied, or of mixed content. Additionally, we determine which nodes in the quadtrees have changed since the preceding reconstruction frame to enable later, incremental processing. We do not distribute this process, as transfers of quadtrees are more expensive than node merging. See **Figure 7** for example quadtrees of silhouettes and occlusion masks.

Unlike bottom-up quadtrees, we create the 3D octree in a top-down, incremental manner: We start with the octree of the preceding frame, or, initially, a single node that spans the entirety of the monitored work cell. We traverse the octree from root to leaves and check for nodes the projection of which has been modified in any of the quadtrees over silhouettes or occlusion masks. As quadtrees allow for an efficient early-out on coherent regions in the silhouettes, we can perform efficient lookups per octree node.

Octree nodes that have been changed with respect to any quadtree must be updated. To this end, we efficiently check the occupation state of the node projection in the quadtrees: A node that only maps to background pixels in one image is free of obstacles, while a node that maps to foreground pixels in all images is fully occupied by an obstacle. Any such node has any children discarded and ends up with a uniform state in the final reconstruction. Other nodes are of mixed content. Those must be split (i.e. if children-free in the previous frame). Processing then continues on children nodes, up to single-pixel projections in each camera.

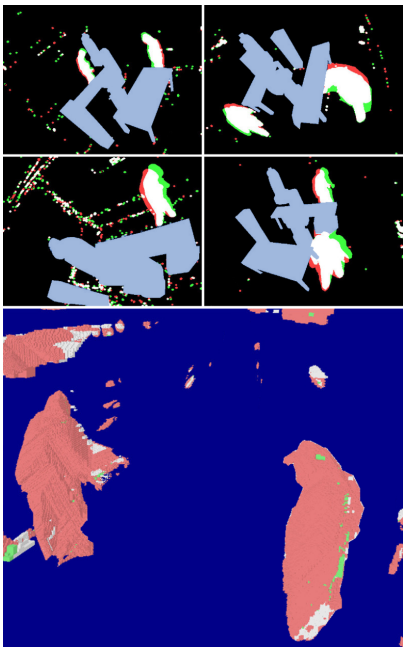

**Figure 8** A 3D reconstruction (bottom) from silhouettes (top) with potential occluders (i.e. the robot or B-Reps of the offline reconstruction, blue). Obstacles may false-negatively register as background if behind occluders that are part of the background model (e.g. B-Reps of the offline reconstruction part), such as the legs of the person in the top left silhouette image. To solve this problem, background pixels that map to occlusions must be excluded from reconstruction.

Apart from silhouettes, we must also test for occlusions by the robot (i.e. occlusion masks) or static obstacles (i.e. B-Reps of the offline reconstruction). For the former, we perform a likewise check against the occlusion mask quadtrees: If a node is entirely occluded in a camera image, that image is ignored when determining node occupation (as above). For the latter, we first require a lookup-friendly variant of the B-Rep model. There are two variants to this: The first variant implies sampling the occluded spaces as a separate, static octree, then traversing both octrees in parallel while looking up per-camera occlusions in the static octree. Alternatively, a second variant involves rendering separate silhouette images of B-Reps, building quadtrees over these, and finally performing lookups for occlusions, possibly splitting on mixed occlusion. The first variant has a bias for performance over memory footprint, while the second variant favours a low memory footprint over speed. Since memory is plenty on modern (non-mobile) hardware, we decided to realize the first variant.

This concludes the summary of our online reconstruction part. **Figure 8** gives examples for silhouette images and a respective 3D reconstruction. See [21, 23] for more details on our online reconstruction, including omissions such as knowledge-based refinement and probabilistic variants.

### 2.3 Online Path planning

Concurrent to the online reconstruction, our path planning part drives the robot manipulator by transmitting speeds and path segments. We derive speeds and path segments from results of both reconstruction parts by an efficient, hierarchical distance query (see [5]),

$$d_H : \mathscr{H}_1 \times \mathscr{H}_2 \to \mathbb{R},$$

for any conservative bounding volume hierarchy (BVH) types $\mathscr{H}_1$ and $\mathscr{H}_2$, where $d_H(H_1, H_2)$ denotes the minimum distance between leaf objects of $H_1 \in \mathscr{H}_1$ and $H_2 \in \mathscr{H}_2$.

To use the hierarchical distance query, we need three conservative bounding volume hierarchies: One hierarchy for the octree of the online reconstruction, another hierarchy for the B-Rep model of the offline reconstruction, and a final hierarchy for the CAD model of the robot. Octrees already are a conservative bounding volume hierarchy, so we only discuss the other two cases.

State-of-art considers bottom-up approaches (e.g. space-filling curves [14]) and top-down approaches (e.g. splitting heuristics [20]) to derive conservative BVHs. In general, the former take less time to build, while the latter generate better-optimized hierarchies.

B-Reps for static obstacles do not change at run-time. In this case, the run-time spent for hierarchy building is mostly irrelevant. We therefore use a splitting-based approach. We start with a bounding primitive around each B-Rep, then recursively split the set of all boundings along the best of several splitter candidates to generate a bounding volume hierarchy. Our experiments indicate that box boundings (opposed to spheres) alongside splitter candidates between neighbouring primitives sorted along each axis (as opposed to random splitters) and a splitting heuristic of minimized intersection volume (as opposed to intersection diameter) give well-performing — but, not necessarily, balanced — bounding volume hierarchies.

Opposed to the B-Reps for static obstacles, the BVH over the CAD model of the robot changes very frequently. In particular, the robot hierarchy must adapt each time that path planning queries any pose for distances to obstacles. We hence use an incremental, bottom-up approach: We supply bounding primitives (again, boxes turn out faster than spheres in our experiments) over model triangles, then classify primitives according to their Morton curve index (see [14]), and recursively merge primitives with close-by curve indices until we reach the hierarchy root. To avoid perpetual collisions with the floor, this hierarchy contains only movable limbs of the robot, but not the robot base.

To derive an overall distance between robot and obstacles, we assume a robot manipulator with $n \in \mathbb{N}$ degrees of freedom and poses $p \in C$ in configuration space $C \subset \mathbb{R}^n$. With conservative BVHs $H_{\text{dyn}}$ for the octree, $H_{\text{stat}}$ for the static obstacles, and $H_{\text{rob}}(p)$ for the robot in pose $p$, we find a minimal distance

$$d_{\text{dyn}}(p) = d_H(H_{\text{rob}}(p),\, H_{\text{dyn}})$$

between the robot in pose $p$ and dynamic obstacles, and another minimal distance

$$d_{\text{stat}}(p) = d_H(H_{\text{rob}}(p),\, H_{\text{stat}})$$

between the robot in pose $p$ and static obstacles. Together, both distances give an overall minimal distance

$$d(p) = \min(d_{\text{dyn}}(p),\, d_{\text{stat}}(p)).$$

from robot to any obstacles. Based on the overall minimal distance, we directly determine allowed speed through a straightforward heuristic

$$v_{\text{normalized}}(p) = \min(1,\, d(p) \,/\, \lambda_{\text{heuristic}})$$

with parameter $\lambda_{\text{heuristic}} \in \mathbb{R}^+$.

While distances directly allow for speed control, we still need to generate path segments. To do so, path planning incorporates a series of concurrent agents. Each of these agents realizes a state-of-art path planning approach (e.g. PRM, RRT, spline). All agents collaborate through a blackboard mechanism, where one agent may replace segment suggestions of another agent with better (e.g. collision-free or faster) path segments. This enables us to combine the benefits of different state-of-art planners without enforcing a fixed higher-level hierarchy or pipeline. Internally, all planners use the above distance queries to find collision-free path segments, and they employ the speed heuristic to evaluate segment speed. Finally, a separate execution agent iteratively removes segments from the fastest path on the blackboard and sends these to the robot for execution. **Figure 6** illustrates the blackboard mechanism. See [23] for an in-depth discussion of our path planner, including omissions such as path annotation and path selection.

## 3 Evaluation

We have evaluated our contribution through path planning on a real-world robot work cell. The work cell contains a Stäubli RX130 industrial manipulator and static mock-up obstacles as shown in **Figure 1**.

At first, we recorded B-Reps of static obstacles through the offline reconstruction part using — as mentioned — a Lenovo Phab 2 Pro mobile device with on-board depth sensor. This process took several minutes and resulted in around fifty B-Rep surfaces in the global model. We then used the open source 3D graphics suite Blender to remove fragments of the robot from the offline reconstruction.

For the subsequent online reconstruction part, a network of eight inexpensive Logitech C930e web cameras monitors the work cell. Each camera connects to a separate computer as described earlier, where a GPU of type NVIDIA GTX 950 performs foreground-background segmentation and renders occlusion masks for the current robot pose. A consumer-grade server computer (Intel Core I7-6700, quad core) receives results of preprocessing and performs both online reconstruction and path planning.

In our test sequence, a worker enters the work cell and starts working close to one of the static obstacles. The precise and sparse B-Rep representation of static obstacles shows four benefits in this experiment: Most importantly, precision and performance of both reconstruction and path planning improve, but B-Reps additionally reduce memory consumption and improve path quality.

Precision of distance queries for static obstacles by mean increases from the size of octree leaf nodes (5 cm error margin at a typical $512^3$ resolution) to the structural size of sparse B-Reps (1 cm error margin). However, there are peaks well above the average (up to 30 cm improvement in distance error). Cause for these peaks is that a visual hull (see [13]) as built by the online reconstruction cannot correctly reconstruct surfaces that face all cameras, such as the tops of our mock-up obstacles. Improved precision in turn allows path planning to pick paths closer to static obstacles and gives slightly better robot speeds (about 10%, depending on paths and the exact speed heuristic). At the same time, memory consumption and octree node count decrease (about 30k octree nodes as opposed to around 50 B-Reps for our static mock-up obstacles). Finally, the increase in performance is a direct consequence of the sparse B-Rep representation, which allows for testing few B-Reps instead of many octree nodes. For the mock-up workspace occupation (e.g. about 50k nodes or about 20k nodes and about 50 B-Reps), the execution time of distance queries improves by approximately 20%, even notwithstanding the additional gain in accuracy.

# 4 Conclusion

In the preceding, we have contributed a novel approach to handling both static and dynamic obstacles in the context of path planning for a robot manipulator: Capturing static obstacles separately in coarse, but precise B-Rep form with an intuitive offline process reduces runtime and increases precision for online reconstruction of dynamic obstacles and for subsequent online path planning. In conclusion, our combined offline/online reconstruction approach has notable benefits at just minor costs (i.e. the offline B-Rep reconstruction must be performed manually). That said, our approach is not suitable for use cases where seemingly static obstacles change at times (e.g. when workers may move or adjust tables). Future work includes handling non-planar B-Rep faces, using depth maps of static obstacles, and real-world (as opposed to mock-up) experiments.

# 5 Acknowledgements

This work has partly been supported by the Deutsche Forschungsgemeinschaft (DFG) under grant agreement HE26-96/11 SIMERO.

# 6 Literature

[1] A. Bigdelou, A. Ladikos, N. Navab, "Incremental visual hull reconstruction", Machine Vision Association, 2009

[2] J. R. Casas, J. Salvador, "Image-based multi-view scene analysis using conexels", Use of vision in human-computer interaction, Volume 56, Australian Computer Society, 2006

[3] D. Culibrk, V. Crnojevi: GPU-Based Complex-Background Segmentation Using Neural Networks, The Irish Machine Vision and Image Processing, 2010.

[4] F. Endres et al., "An evaluation of the RGB-D SLAM system", ICRA, 2012

[5] H. Dominik, C. Xiaoqing, "Fast Distance Computation for On-line Collision Detection with Multi-Arm Robots", ICRA, 1992

[6] O. Kähler et al. "Very High Frame Rate Volumetric Integration of Depth Images on Mobile Devices", Visualization and Computer Graphics, vol. 21, no. 11, pp. 12411250, 2015

[7] M. Keller et al., "Real-Time 3D Reconstruction in Dynamic Scenes Using Point-Based Fusion", Conference on 3D Vision, 2013

[8] C. Kerl, J. Sturm, D. Cremers, "Dense visual SLAM for RGB-D cameras", Intelligent Robots and Systems, 2013

[9] S. Kuhn, "Multi-view reconstruction in-between known environments", Technical Report, Universität Bayreuth, 2010

[10] S, Kuhn, "Wissens- und sensorbasierte geometrische Rekonstruktion", PhD thesis, Universität Bayreuth, 2012

[11] A. Ladikos, S. Benhimane, N. Navab, "Real-time 3d reconstruction for collision avoidance in interventional environments", Medical Image Computing and Computer-Assisted Intervention, 2008

[12] A. Ladikos, S. Benhimane, N. Navab, "Efficient Visual Hull Computation for Real-Time 3D Reconstruction using CUDA", Computer Vision and Pattern Recognition Workshops, 2008

[13] A. Laurentini, "The visual hull concept for silhouette-based image understanding", Pattern analysis and machine intelligence, 1994

[14] C. Lauterbach et al., "Fast BVH construction on GPUs" Computer Graphics Forum, 2009

[15] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning", Iowa State University, TR, 1998

[16] R. A. Newcombe et al., "KinectFusion: Real-time dense surface mapping and tracking", Mixed and Augmented Reality, 2011

[17] Pilz Safety Eye, Web: `https://www.pilz.com/en-DE/eshop/00106002207042/SafetyEYE-Safe-camera-system`

[18] M. Sand, D. Henrich, "Incremental reconstruction of planar B-Rep models from multiple point clouds", The Visual Computer, 2016

[19] L. Soares et al., "Work stealing for time-constrained octree exploration: Application to real-time 3d modeling", Eurographics, 2007

[20] I. Wald, "On fast construction of SAH-based bounding volume hierarchies", Interactive Ray Tracing, 2007

[21] T. Werner, D. Henrich, "Efficient and Precise Multi-Camera Reconstruction", Distributed and Smart Cameras, 2014

[22] T. Werner, J. Bloess, D. Henrich, "Neural Networks for Real-Time, Probabilistic Obstacle Detection", RAAD, 2017

[23] T. Werner, D. Henrich, D. Riedelbauch, "Design and Evaluation of a Multi-Agent Software Architecture for Risk-Minimized Path Planning in Human-Robot Workcells", Montage Handhabung Industrieroboter, 2017

[24] T. Whelan et al., "Real-time large-scale dense RGB-D SLAM with volumetric fusion", Robotics Research, vol. 34, no. 45, pp. 598626, 2015

[25] T. Whelan et al., "ElasticFusion: Real-time dense SLAM and light source estimation", Robotics Research, vol. 35, no. 14, pp. 16971716, 2016.

[26] K. M. Wurm et al., "OctoMap: A Probabilistic, Flexible, and Compact 3D Map Representation for Robotic Systems", ICRA, 2010