



# TrimBot2020 Deliverable D3.2

# Implementation and evaluation of SLAM, 3D from binocular and motion stereo


| | |
|---|---|
| Principal Author: | ETH Zürich (ETHZ) |
| Contributors: | University of Edinburgh (UEDIN), Albert-Ludwigs-Universität Freiburg (ALUF) |
| Dissemination: | CO |


**Abstract:** This report describes and evaluates the Simultaneous Localization and Mapping (SLAM) and 3D data generation pipelines developed in the Trimbot2020 project. In particular, the report provides details on the SLAM system used in the Trimbot2020 project to build 3D maps of the scene and determine the position of the robot within these 3D maps.

In addition, the report also describes and evaluates the vehicle localization system developed for the TrimBot2020 project. The description and evaluation was originally planned for Deliverable D3.5 (*Arm and vehicle localization*). However, it will be presented as part of this deliverable as the vehicle localization system is tightly integrated in the SLAM pipeline.

Deliverable due: Month 30

# Contents



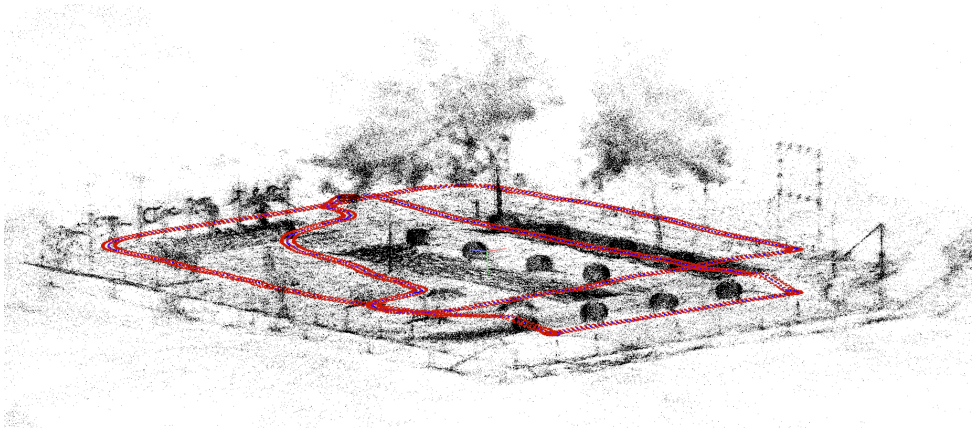


Figure 1: The developed Simultaneous Localization and Mapping (SLAM) system builds a 3D scene map, in this case a sparse point cloud shown as black dots, and uses this map to track the movement of the robot (red and blue points) through the scene. This figure shows a 3D map built with the SLAM system developed for the TrimBot2020 project.

# 1 Overview

This report consists of four parts: The first part (Sec. 2) details the SLAM pipeline developed for the Trimbot2020 project. This includes a description of the vehicle localization module that is part of the SLAM system and allows the robot to re-localize against pre-built 3D maps.

The second part (Sec. 3) describes the algorithms developed in the Trimbot2020 project that compute depth maps from binocular and / or motion stereo. More precisely, Sec. 3.1 reports how the FPGA is used to synchronize the robot's cameras and also computes coarse depth maps. Sec. 3.2 outlines 'DispNet', a convolutional neural network (CNN), designed to compute dense depth maps. Sec. 3.3 describes a depth perception approach that uses a stereo pair and an additional camera and thus performs trinocular stereo.

The third part (Sec. 4) then describes how the different sensor modalities, namely the sparse point clouds generated by SLAM and the dense depth maps generated by the stereo algorithms, are fused into a single scene representation.

Finally, the last part (Sec. 5) provides an experimental evaluation of the SLAM system, the 3D scene perception algorithms, and the data fusion pipeline.

When appropriate, the reports refer to publications for details.

# 2 Simultaneous Localization and Mapping (SLAM) (ETHZ)

Determining the position and orientation of the robot, i.e., its pose, is a key step that allows the TrimBot to autonomously navigate in the garden. We solve this problem in the TrimBot2020 project through *structure-based visual localization* [1–5]: We first build a 3D map of the garden. Once this map is built, we localize the robot with respect to this map by aligning the camera images captured by the robot to the 3D structure represented in the map.

In order to build a 3D map of a garden, we use a Simultaneous Localization and Mapping (SLAM) approach [6–9]. SLAM methods simultaneously construct a 3D map (a sparse 3D point cloud in our case) and use this map to localize the robot while the robot drives through

the scene[1]. Fig. 1 illustrates this concept. Once a 3D map is built via SLAM, we use a Visual Localization algorithm that establishes correspondences between features extracted from camera images and 3D points in the map. The position and orientation of the robot can then be estimated from the resulting 2D-3D correspondences. We integrate localization into the SLAM system, allowing us to continue tracking the pose of the robot even if localization should fail in a frame [10,11]. In the following, we describe the SLAM and localization algorithms developed for the TrimBot2020 project.

The suitability and usefulness of cameras to tackle the SLAM problem has been shown in numerous projects (e.g. [6–9]). At the same time, cameras have advantages over active sensors such as LIDAR or Laser since they typically have a higher resolution and have a significantly lower power consumption. In the TrimBot2020 project, we use a camera rig with 10 cameras arranged into 5 stereo pairs (cf. Fig. 2) to enable dense 3D perception all around the robot. Naturally, we use the images of this multi-camera system for SLAM. This camera setup, which covers the full $360^\circ$ horizontal field of view (FOV) around the robot, is also beneficial for localization [12].

While stereo camera setups are commonly used in SLAM, e.g., [9, 13], multi-camera systems are less often used. An early approach [14] uses a multi-camera setup in combination with wheel encoder measurements to estimate the driven trajectory and a sparse map, but does not perform loop closure detection. [15] models multiple cameras as a single spherical camera and can therefore not be used with arbitrary camera configurations. [16–18] use multi-camera systems with little to no visual overlap between the individual cameras, while [19] uses two stereo pairs without any visual overlap between the two pairs. [20, 21] present multi-camera SLAM systems based on the existing monocular solutions [7, 9] and show the benefits of an increased FOV. [22] presents a VO system using multiple stereo camera pairs and shows a significant increase in robustness under challenging conditions. In this context, the multi-camera system used in the TrimBot2020 project is rather unique. Due to the lack of existing suitable SLAM systems, we thus developed a multi-camera SLAM algorithm that can handle any type of multi-camera system[2]. Our approach [23] models the multi-camera system as a generalized camera [24], i.e., a camera with multiple centers of projection.

The rest of this section describes our SLAM system and is structured as follows: An overview over the system is provided in Sec. 2.1. The rest of the chapter is divided into the different steps of the algorithm, *feature extraction and matching* (Sec. 2.2), *pose estimation* (Sec. 2.2), *mapping* (Sec. 2.4), and *iterative refinement* (Sec. 2.5). We collect all information concerning non-linear optimization of the map in Sec. 2.6. Sec. 2.7 outlines how maps build by the SLAM system can later be used to localize the robot in known environments. Finally, Sec. 2.8 describes how semantic information can be used as part of SLAM.

**Notation** For simplicity and consistency, we define a few frequently used terms and expressions: We denote any point in the 3D scene or in the map simply as a *point*. A *feature* is a projection of an arbitrary 3D point onto an image that can be detected by a feature detector, e.g., SIFT [25], expressed in pixel coordinates. The corresponding projection onto the normalized image plane (in camera space) is a *normalized feature*. If a feature corresponds to a scene point, we call it an *observation* and we *associate* the feature and the point in the map. We use the term

[1] In the TrimBot project, the robot is currently remotely controlled by a human during this initial traversal.

[2] Our approach requires that the intrinsic and extrinsic parameters of the individual cameras in the multi-camera rig are known. For details on the calibration process used in the TrimBot2020 project, we refer the interested reader to Deliverable D3.1.

Figure 2: The camera rig configuration used on the TrimBot. 10 monocular cameras are arranged in 5 stereo pairs. The combined field of views (FOVs) provide a horizontal $360°$ view of the scene.

*optimization* for the non-linear optimization of the map that jointly optimizes the positions of the 3D points and the poses of the robot. The term *refinement* refers to a combination of such an optimization with re-triangulation of the points and filtering of the map.

We refer to our input data with different levels of abstraction: A set of rigidly mounted *monocular cameras* with known intrinsic and extrinsic parameters is referred to as a *generalized camera*. Each monocular camera takes *monocular images*. Accordingly, the set of monocular images captured at the same time[3] by the cameras in the multi-camera rig is then called a *generalized image*. To simplify the explanations in this section[4], and since the concepts are in most cases parallel to algorithms using monocular images, we use the terms *image* and *generalized image* interchangeably for the combination of all monocular images of the same rig at the same time. The data of only a single monocular camera is explicitly denoted as *monocular image*.

## 2.1 Overview

We developed a SLAM system that uses SIFT [25] for feature detection and description. Our pipeline is built on top of COLMAP [26], an incremental Structure-from-Motion (SfM)[5] pipeline partially developed at ETH Zurich. This allowed us to build on top of an existing infrastructure that already provides functionality for feature extraction and matching, 3D point triangulation, etc., rather than starting from scratch.

Fig. 3 provides an overview over our pipeline. The input to our SLAM algorithms are the generalized images $G_t$, consisting of monocular images $I_i$, ..., $I_j$, that are captured by the TrimBot's camera system and we process them as they become available. While our approach

[3]We assume that all our cameras are synchronized, which is done by an FPGA in the TrimBot2020 project.

[4]In other sections, the term *image* will, as usual, refer to a single image taken by a single camera.

[5]As SLAM, incremental SfM iteratively builds a map by adding images sequentially. In contrast to SLAM, where the images become available one after another as they are captured, incremental SfM systems typically know all images before starting the reconstruction process.

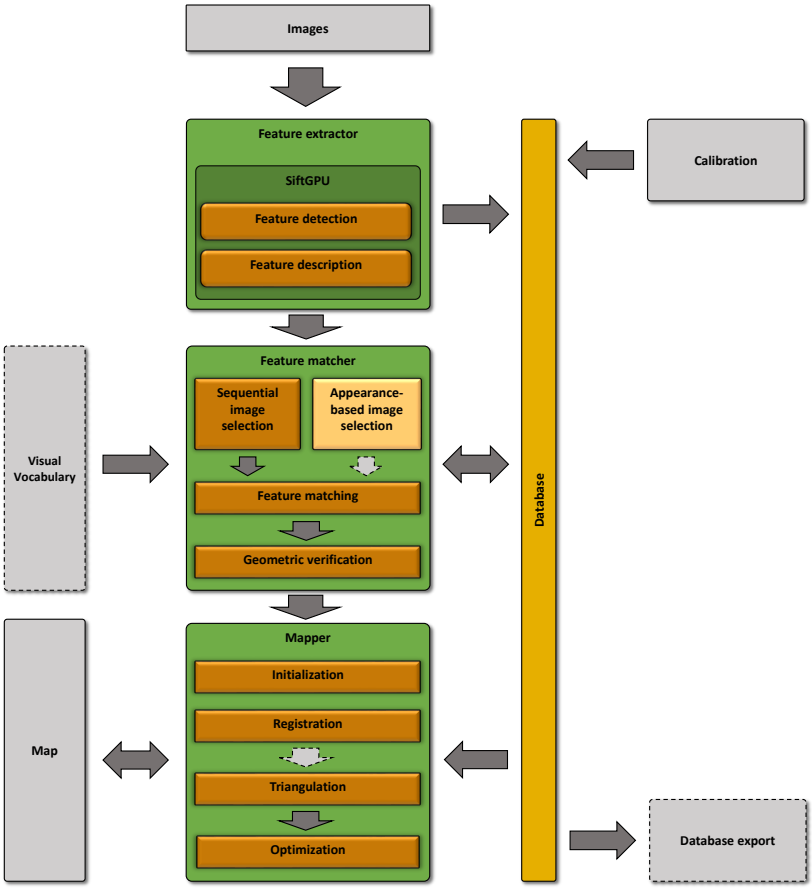



Figure 3: The Generalized Camera SLAM pipeline.

can calibrate the camera extrinsics from scratch[6], in the following we assume that the calibration of the camera rig is known. We use this to represent a generalized image as a collection of monocular images with known relative poses during the whole process. We extract local features from the monocular images and use them to generate a sparse point cloud, which in turn is used to estimate the pose of new generalized images. Afterwards, the known poses are used to extend the map by triangulation and bundle adjustment [27] is used to refine the map and camera poses. We use an image retrieval-based approach [28] for both loop closure detection and re-localization. For this component, our approach requires that a suitable visual vocabulary [29] is trained before operation. In order to handle loop closures, all generalized images are stored in a database.

As described above, we assume the camera rig calibration to be known. To handle potentially inaccurate initial calibrations, we are able to optimize the calibration parameters during bundle adjustment. In this case we still assume that all cameras are rigidly connected and do not experience relative motion during runtime. As a result, the optimized parameters will also be applied to earlier generalized images. Parameter optimization is, however, currently not enabled in order to reduce the runtime of the algorithm and thus adhere to the computational budget of the TrimBot.

[6]See Deliverable D3.1.

## 2.2 Feature Extraction and Matching

As a first step, we extract local features from the monocular images before we are able to use them as observations for the mapping. Since we ignore all appearance information in the later processing and only rely on the estimated correspondences, extracting enough information robustly and accurately is crucial for all subsequent steps.

**Feature extraction** To extract useful information from the input images, we rely on the SIFT [25] feature detector and descriptor. In order to minimize delay and computational overhead, we make use of an efficient implementation running on the GPU, provided by SiftGPU [30]. We set a threshold of 200 features per monocular image and prefer larger scale features over smaller scale ones [31]. With the 10 camera setup, this leads to more than 2000 relatively equally distributed extracted features per generalized image. Feature extraction takes around 0.2 s if all 10 cameras are used, and scales linearly with the number of cameras.

**Sequential image selection** Given the sequential nature of the incoming generalized images, we can limit feature matching to images in a small temporal window. With the sequential image selection, a generalized image $G_t$ taken at time $t$ is only matched to generalized images $G_{t-1}$, $\ldots$, $G_{t-w}$, within a window of size $w \leq t$. Knowing which camera pairs have an overlapping FOV, we can limit the feature matching further by only matching monocular images of these camera pairs. This is done to accelerate frame-to-frame tracking of the features. Formally, given cameras $C_1$, $C_2$, $C_3$ of a generalized camera, where $C_1$, $C_2$ have overlapping FOVs, and denoting the monocular image of camera $C_n$ in $G_m$ as $I_m^n$, we select the set of monocular images to match to a monocular image $I_t^1$ as $\left\{I_t^2, I_{t-1}^1, I_{t-1}^2, \ldots, I_{t-w}^1, I_{t-w}^2\right\}$. When performing the sequential matching with a window size of $w = 4$, the matching takes less than 0.1s when 10 cameras are used, and less than 0.05s when the number of cameras is reduced to 5, e.g., when using only one monocular image per stereo pair.

**Appearance-based image selection for loop closure detection** The sequential image matching strategy described above only handles frame-to-frame camera tracking. In addition, it has problems in the case where the robot rotates on the spot as the selected camera pairs that are considered for matching might not observe the same part of the scene anymore. We handle both problems through loop closure detection.

Our loop closure detection approach is based on image retrieval via the Bag-of-Visual-Words model [29], where we use a predefined vocabulary to quantize the feature descriptors from the images. We use the FLANN library [32] to assign each feature to multiple words in the vocabulary. Afterwards, we use an inverted index [29] to find similar images. We rely on an implementation by Sattler *et al.* [28] for efficient image retrieval using Hamming embedding [33, 34]. To simplify the implementation of the retrieval retrieval step, we handle each monocular image in a generalized image individually. For each monocular image, we retrieved the 10 most similar monocular images in the inverted file and perform feature matching for each of them.

Due to the slow traveling speed of the TrimBot, subsequent generalized images are typically very redundant. To accelerate the loop closure detection stage, we thus subsample the input data by only using every $n$-th generalized image. This $n$ should be selected large enough to avoid redundant information, but small enough to still capture the whole scene with the sampled images. We set $n = 25$, which results with the current robot velocity and camera frame rate in a loop closure detection roughly every 1.5m.

**Feature matching** As with the feature extraction, we use SiftGPU for efficient feature matching on the GPU. This is done for all monocular images pairs selected in both the sequential image selection and the loop closure detection. After matching, we perform geometric verification by computing an estimate of the essential matrix between each monocular image pair using RANSAC [35] and only accept pairs with a minimum number of inliers for this estimate. All outlier correspondences are discarded and not used in the subsequent processing. To efficiently find matching features later on, we save the information in form of a *scene graph*.

## 2.3 Pose Estimation

The robot's estimated pose within the map is the single most important output of the algorithm. Pose estimation can be done either from an existing map, or as part of the mapping process. The algorithm is in both cases the same, as also during mapping, pose estimation is done only on the already existing part of the map.

We obtain 2D-3D matches between the monocular image features and points in the map by finding transitive 2D-2D matches with features in the other monocular images first, and then using the points associated with these features. Since we match each monocular image against multiple other ones, it is possible that we find multiple 3D point candidates for a single image feature. We do not handle this explicitly and still collect all possible correspondences for the following pose estimation and outlier rejection step with RANSAC. We use the minimal solver of Lee et al. [16], which was built specifically for pose estimation of multi-camera systems, to estimate the absolute pose of a generalized image with respect to the 3D map.

Inlier correspondences for an estimated pose are selected based on their reprojection error. Additionally, we mark all points that do not pass the cheirality check as outliers. We allow rather large errors to still be counted as inliers to maximize the number of point associations. This way, we get a possibly slightly less accurate, but more robust pose estimate. To improve the initial estimate from RANSAC, we use nonlinear optimization of the pose with all inlier correspondences. The optimization is described in detail in Sec. 2.6.

## 2.4 Mapping

When the robot leaves the previously mapped area, we need to extend the map to still be able to localize. The assumption here is that the currently available map is always sufficient to estimate a new generalized image's pose. With all known generalized image poses, the map can then be extended by triangulating new points. This procedure is also used to generate a map offline, which can later be used to localize the robot against a given map.

As usually for SfM, the map does not only consist of the points, but the images are another important part. In fact, it is not really necessary to store the points' positions in the map as they are also implied through the constraints by their observing images, but computing them every time from scratch would not be feasible.

**Keyframe selection** Consecutive generalized images often contain a lot of redundant information. Including all this information in the map does not significantly improve the accuracy, but increases both the required memory as well as processing power when optimizing the map. As a consequence, we try not to include every single generalized image into the map. First, we do not add a keyframe if the mapper is still busy processing the last one. If the mapper is idle, we perform a keyframe selection to determine which generalized images add a significant amount

of new information to the map. We perform localization on all other generalized images, but do not use them for mapping. As a measure for new information in a generalized image, we look at the number of common point observations between the current frame and the last keyframe. If less than 50 observations in total or less than 70% of the last keyframe's observations are shared, we select the current frame to be added to the map. Additionally, even if the observed information stays similar, we insert at least every 100th frame. Another special case arises if the loop closure detection is turned on. Since we can only use feature matching information if the corresponding frame is contained in the map, we select all frames that are used for loop closure detection as keyframes.

**Image registration** To extend the map, the first step in mapping is to add the known correspondences between image features and already existing points from pose estimation to the map. This incorporates the constraints that were used to localize the generalized image into the map and puts additional constraints on the points' positions for later optimization. If there are multiple correspondences for a feature left after RANSAC, we select one of them arbitrarily as we cannot decide which one is the best.

**Point completion and triangulation** We can extend the map by adding additional constraints to existing points as well as triangulating new points, again using transitive feature matches. New constraints do, however, not necessarily need to stem from the new images. It is possible that the new feature just reveals an older feature's observation of an existing point. Similarly, the new data can also just reveal transitive matches between already known features that lead to a new point triangulation. This is again done using RANSAC to construct stable points in the presence of outlier matches. We use a recursive triangulation strategy, where we do not stop after a point could be triangulated, but try to triangulate additional points with the remaining, unassociated matches.

## 2.5 Iterative Refinement

Incrementally adding the generalized images to the map via SLAM typically results in drift as (small) errors in the estimated poses accumulate over time. Therefore, frequent refinement of the camera poses of the generalized images and 3D point positions is necessary. This refinement is done via bundle adjustment. As is common in Structure-from-Motion pipelines, e.g., [26], we use a two-step approach that first applies bundle adjustment and then merges similar points and discards inaccurately triangulated points. Both steps are repeated a fixed number of times.

We use a form of windowed optimization [36] for the refinement: We frequently perform refinement on a small part of the map around the most recently added generalized image. In addition, we perform a more global refinement that optimizes a larger part of the map and camera poses at a lower frequency. This enables us to better handle the drift introduced by the accumulation of small local errors. Notice that we do not optimize the full map (at least once a certain number of generalized images have been added).

We found that it is still possible that we are not able to compensate for the accumulated drift after detecting a loop closure due to only adapting a part of the map. Pose graph optimization [37] of the full map could help to solve this problem. However, we observed that this rarely happens as long as some stable landmarks are observable from most parts of the garden. This is typically the case in the TrimBot gardens, especially in the Wageningen test garden.

For the frequent refinement, we select the 15 last keyframes and their associated points, and

perform multiple iterations of the following steps: We minimize the sum of all reprojection errors through bundle adjustment (cf. Sec. 2.6). After optimization, we try to merge points that are connected by transitive matches but were previously split during RANSAC-based triangulation. We also try to associate new features with existing points as well as triangulate new points. Finally, we try to reduce noise in the map by removing feature-point associations with high reprojection errors and points with a small triangulation angle, which implies a high depth uncertainty. Additionally we remove points with less than 4 observations to reduce the size of the map.

For the less frequent refinement, we select a larger window of the last 40 keyframes and again their associated points. While the general approach for both the small and enlarged window refinement is the same, we adapt some parameters for the larger window to account for the slightly different focus of the problem. We increase the minimum triangulation angle, which leads to fewer, but more stable points. We also try again to find points for previously unassociated image keypoints as well as triangulate new points considering the increased minimum triangulation angle. This helps especially to create constraints from loop closure matches. We can optionally use the larger window to optimize the extrinsic and intrinsic camera parameters. Notice that this optimization cannot be done in a small window: Considering only a small part of the map can easily lead to biased parameter estimates due to local disturbances and as a result corrupt the whole map.

## 2.6 Optimization via Bundle Adjustment

When optimizing the map and the camera poses, we enforce that all cameras that form a generalized camera move rigidly. At the same time, we enforce that this extrinsic calibration of the individual cameras is constant over time. We integrate these camera rig constraints into the bundle adjustment process by expressing the pose $(\mathbf{R}_I, \mathbf{t}_I)$ of a monocular image as a combination of the pose $(\mathbf{R}_G, \mathbf{t}_G)$ of the generalized image and the pose $(\mathbf{R}_C, \mathbf{t}_C)$ of the camera $C$ in the multi-camera rig that took the image, i.e.,

$$\mathbf{R}_I = \mathbf{R}_C \mathbf{R}_G \tag{1a}$$

$$\mathbf{t}_I = \mathbf{R}_C \mathbf{t}_G + \mathbf{t}_C \ . \tag{1b}$$

Here, $\mathbf{R}$ and $\mathbf{t}$ denote a rotation matrix expressing the camera orientation and the translation of that camera, respectively.

Let $\mathcal{G} = \{G_1, \cdots, G_n\}$ be a set of $n$ generalized images taken with the same generalized camera. The $k^{\text{th}}$ generalized image $G_k$ is defined as a set of monocular images $G_k = \{I_k^i\}$, where $I_k^i$ denotes the monocular image taken with the $i^{\text{th}}$ monocular camera in the multi-camera rig. For a monocular image $I_k^i$, let $V(I_k^i) = \{(\mathbf{x}_j^{I_k^i}, \mathbf{X}_j)\}$ be the set of 3D point positions $\mathbf{X}_j$ and their corresponding observations $\mathbf{x}_j^{I_k^i}$ in the image, i.e., the set of 2D-3D matches between 2D pixels in the image and 3D points in the map. For a given set $\mathcal{G}$ and the 3D points visible in all the monocular images contained in it, we minimize the sum of reprojection errors

$$\sum_{G_k \in \mathcal{G}} \sum_{I_k^i \in G_k} \sum_{(\mathbf{x}_j^{I_k^i}, \mathbf{X}_j) \in V(I_k^i)} \left\| \mathbf{x}_j^{I_k^i} - \pi_{I_k^i} \left( \mathbf{X}_j, \mathbf{R}_{I_k^i}, \mathbf{t}_{I_k^i} \right) \right\|^2 \tag{2}$$

when refining the map. That is, we optimize the poses of all generalized images and the positions of all 3D points visible in them. Here, $\pi_{I_k^i}$ is a function that first transforms a map

point into the local coordinate system of $I_k^i$ and then projects this point into the image. Bundle adjustment is implemented via the Ceres solver [38].

## 2.7 Localization against Existing SLAM Maps

Instead of creating a new map from scratch in each run, we are also able to load a previously generated map and localize against it. This has multiple benefits. The map can be created in an offline preprocessing step and therefore be thoroughly optimized. It can then also be aligned with a global coordinate system, so that the robot is able to estimate absolute poses instead of poses relative to the starting point of the trajectory. Finally, using an existing map reduces the required processing power when moving in a known area.

Even when using an existing map, we still perform the mapping step to be able to update the map and incorporate new information in order to facilitate the localization process. The localization pipeline therefore closely resembles the standard SLAM pipeline with only minor details changed. First of all, to be able to localize against the map we need to read in the map file, the map database containing the map's image features and matches, and a visual vocabulary for efficient matching against the map descriptors (cf. Fig. 3). We then generate a visual index from all monocular map images, which can be used for localization in the appearance-based image selection for matching. In contrast to the standard SLAM pipeline, we will not add any more images later on. This means that the loop closure detection is replaced with a localization against the loaded map. As for the standard SLAM, we still only match every 25th frame against the map, and all others only against previous frames of the same run. The matched generalized images are handed to the standard localization and mapping module. The keyframe selection then decides which of the generalized images are only localized and which are added to the map. When the robot moves in a densely mapped area that can also be matched easily to the current generalized images, this usually leads to fewer keyframes compared to standard SLAM. It is important to note that, even though we do not match all frames against the map, the transitive correspondences used for pose estimation and mapping are sufficient to obtain 2D-3D correspondences with the loaded map, as they are propagated from the respective frames. During standard SLAM, it is beneficial to use at least one stereo pair to avoid scale drift, but robustness is increased when using multiple or all available stereo pairs. Using an existing map eliminates the problem as the scale is already fixed. We can therefore reduce the used cameras to one on each side without sacrificing the total FOV. Even with only 3 cameras we can still get satisfying results, although robustness decreases with the decreasing FOV.

## 2.8 Semantic SLAM

Besides the design and implementation of the multi-camera SLAM system described above, we also worked on integrating semantic information into (general) visual SLAM systems [39]. The goal of this work was to both build semantically annotated 3D maps and at the same time use semantic information to improve camera pose accuracy.

One significant problem in SLAM, illustrated in Fig. 4, is that points typically cannot be tracked over a longer period of time due to the limited invariance of local features or photometric tracking. Thus, SLAM systems are unable to establish longer-term correspondences, leading to drift in the estimated camera trajectory if loop closures cannot be detected (e.g., because the robot never revisits a place that it has visited before). Rather than establishing correspondences

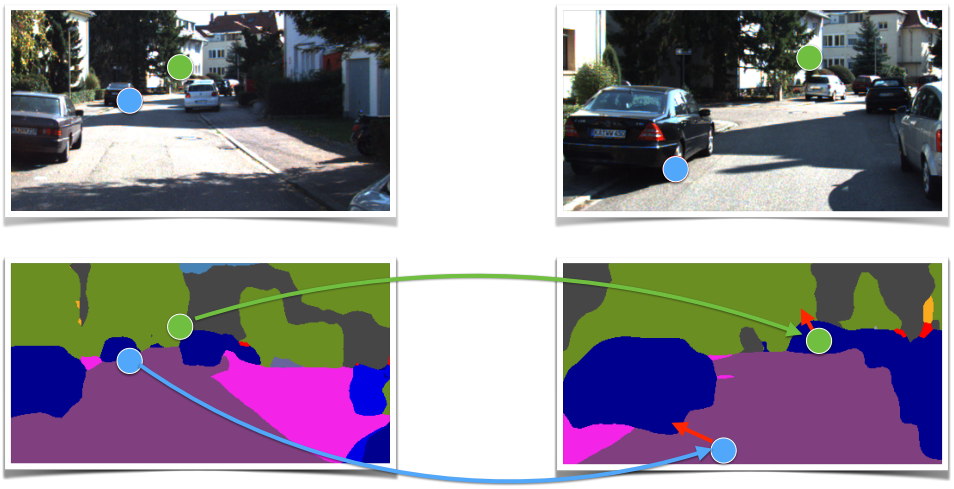


Figure 4: Illustration of the main concept behind Semantic SLAM. (Top row) Due to severe changes in scale, it is typically not possible to associate the blue and green points observed in the left image with their corresponding position in the right image. As a result, only short-term tracks can be established, causing drift in the estimated camera pose trajectory. (Bottom row) Computing semantic segmentations for the input images and assigning labels to the 3D points in the SLAM map, we are able to detect drift: We project the labeled 3D points into the latest frame (right image) and measure whether their labels are consistent with the labels at the corresponding pixel positions. Inconsistent labels yield error terms (red arrows) that can be used inside bundle adjustment to reduce drift.

by matching descriptors or patches, which is inherently limited by the missing invariance of such representations, we instead propose to use semantics to establish longer tracks.

Our semantic SLAM approach performs semantic segmentation on the input images. Using these 2D segmentations, we developed an expectation maximization (EM) algorithm for map refinement. In the expectation step, we project the 3D points into the images using our current belief of the point positions and camera poses. Based on the images' semantic segmentations, we assign each point a (distribution of) semantic label(s). In the maximization step, we fix the labels of the 3D points and optimize the camera poses and point positions to maximize semantic consistency: 3D points projecting into image regions that are assigned to a different label compared to the point are semantically inconsistent. Each such point defines a semantic constraint that can be used during bundle adjustment (see bottom right of Fig. 4). As shown in our publication [39], these constraints can be easily combined with classical visual SLAM constraints such as reprojection and photometric errors. In our work, we show that adding our semantic constraints into existing SLAM systems helps to reduce their positional drift. For details, please see the original publication attached to this report.

Unfortunately, we have not yet been able to evaluate the semantic SLAM approach on TrimBot2020 data. We currently do not have access to the semantic segmentation algorithms developed for the project. Yet, these algorithms are necessary for our semantic SLAM approach as the semantic classes that occur in garden scenes are not modeled in the standard datasets used to train semantic classifiers.

| Disparity [pixel] | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Depth [meter] | 0.48 | 0.5 | 0.52 | 0.54 | 0.55 | 0.57 | 0.6 | 0.63 | 0.65 | 0.68 | 0.71 |
| Disparity [pixel] | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |
| Depth [meter] | 0.75 | 0.79 | 0.83 | 0.88 | 0.94 | 1 | 1.07 | 1.15 | 1.25 | 1.36 | 1.5 |
| Disparity [pixel] | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| Depth [meter] | 1.66 | 1.88 | 2.14 | 2.5 | 3 | 3.75 | 5 | 7.5 | 15 | inf | |

Table 1: Possible disparity values in the range of 0-31 pixel and their corresponding depth values in meter. The focal length is set to $f_c = 500$ pixel and the baseline to $b = 0.03$ meter (the values are identical with the TrimBot camera setup).

# 3 3D Scene Perception

This section describes the 3D perception algorithms developed as part of the TrimBot2020 project. As the SLAM system, they process the images taken by the multi-camera system mounted on the robot. As shown in Fig. 2, the camera system consists of ten cameras arranged as five stereo pairs. Each stereo camera pair consists of one color camera (left) and one greyscale camera (right). The color image is used to aid semantic segmentation. The greyscale camera is typically more sensitive to changes in illumination, which is useful for SLAM and localization. All cameras are synchronized through an FPGA.

## 3.1 FPGA SGM Stereo (ETHZ)

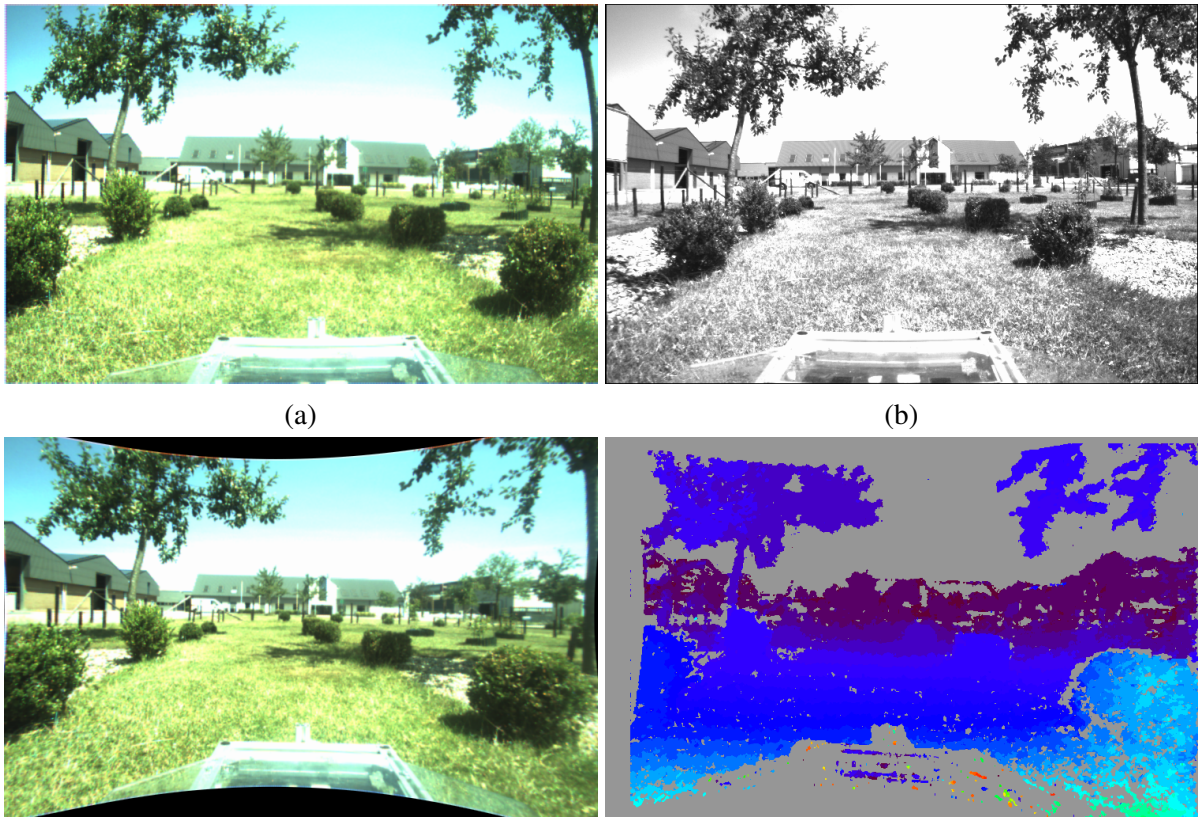
In addition to the data handling and synchronization of all cameras and their corresponding IMUs, the FPGA also performs stereo matching on the image data of the five stereo camera pairs. The stereo matching is based on a Semi-Global Matching (SGM) algorithm as presented in [40] and is leveraging the parallel architecture of the FPGA to perform the matching in real-time and at frame rate, with only little additional latency. Details about the real-time matching implementation are presented in the original publications [41,42]. A visualization of the left and right raw image, the rectified left image and the corresponding disparity map with 32 different disparity values are shown in Fig. 5. Notice that the FPGA stereo algorithm does not incur any computational overhead on the CPU or GPU of the TrimBot platform and thus comes nearly free of any overhead.

Due to the limited memory and logic resources in the FPGA, only 32 disparity candidates are estimated with integer precision. Thus, the FPGA stereo algorithm is only able to measure discrete distances due to the limited amount of disparity values with integer precision. A disparity value $d$ is related to the depth $z$ by

$$d = \frac{b f_c}{z} \ , \tag{3}$$

where $b$ is the baseline of the two cameras and $f_c$ is the focal length. The possible distance values for the TrimBot system with a focal length of $f_c = 500$ pixel and a baseline of $b = 0.03$ meter are summarized in Table 1. Notice that the distance measurement accuracy is decreasing with lower disparity values.



(a) (b)

(c) (d)

Figure 5: The raw left and right images of one stereo camera head is shown in 5a and 5b. The rectified image of the left camera is presented in 5c, the corresponding disparity image with 32 different disparity levels is shown in 5d.

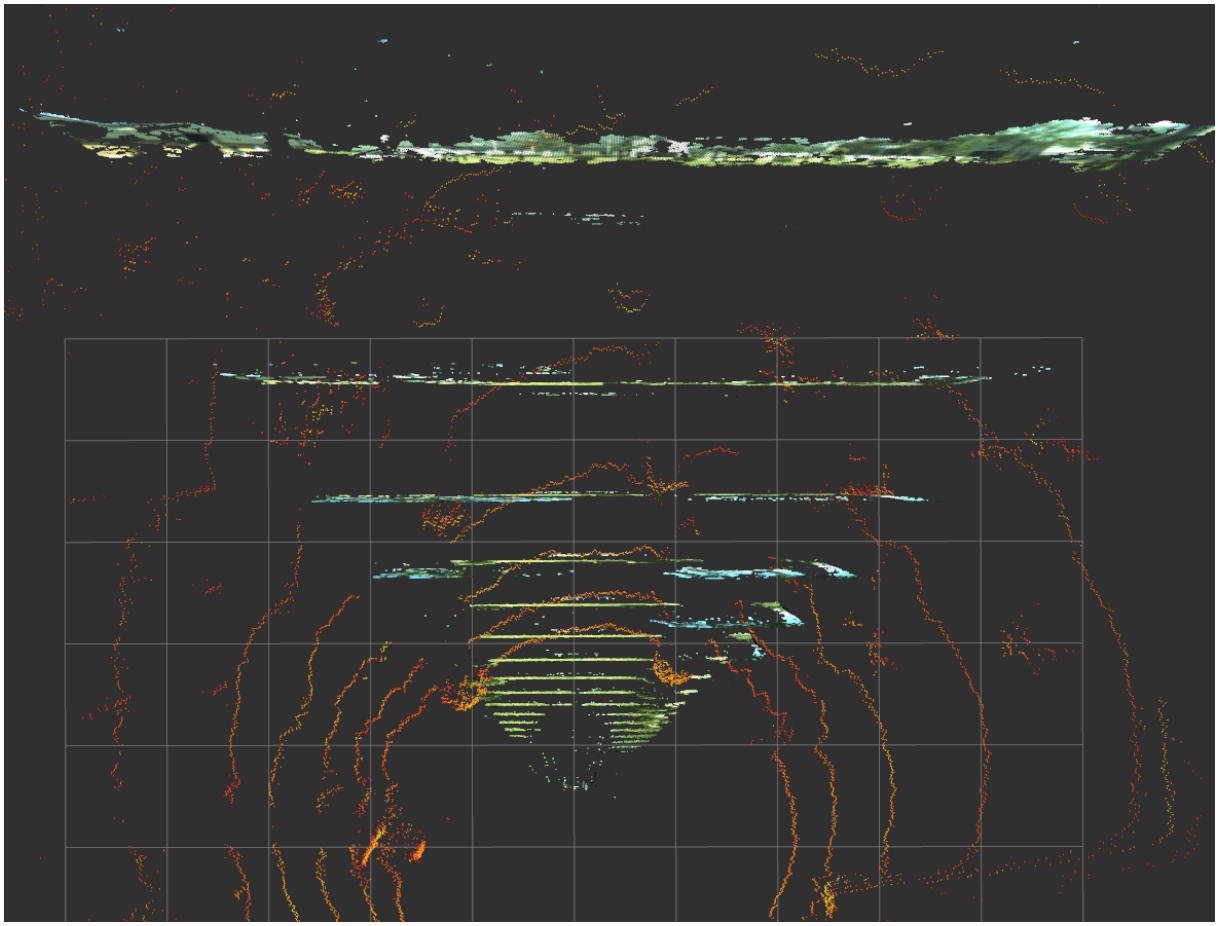


Figure 6: Top-down view of the garden scene as shown in Fig. 5. Image points are projected based on their disparity value and color information. The integer based disparity matching with 32 values results in discrete distance measurements with increasing step size. The figure visualizes a depth map computed from a single stereo pair.

Fig. 6 and Fig. 7 visualize the discrete depth values that are estimated by the FPGA stereo algorithm. The figure shows the garden scene presented in Fig. 5 from a top-down and side view, respectively. The color image is projected in 3D space based on the corresponding disparity values. As comparison, we also show the depth measurements made by a Velodyne lidar scanner as orange dots.

## 3.2 DispNet (ALUF)

*DispNet* is a convolutional neural network (CNN) for end-to-end disparity estimation. Its inputs are full-frame left and right images from a stereo camera, taken at the same time, and undistorted and stereo-rectified. Its output is a dense disparity map for the left input image.

The CNN's encoder-decoder architecture and the possible configuration parameters are described in the deliverable document D5.2. On a high level, the network first computes dense (learned) feature descriptors for the left and right input images. Each descriptor position in the left image is then explicitly matched against a fixed range of descriptor positions in the right image, along the same horizontal line. This formulation effectively uses knowledge

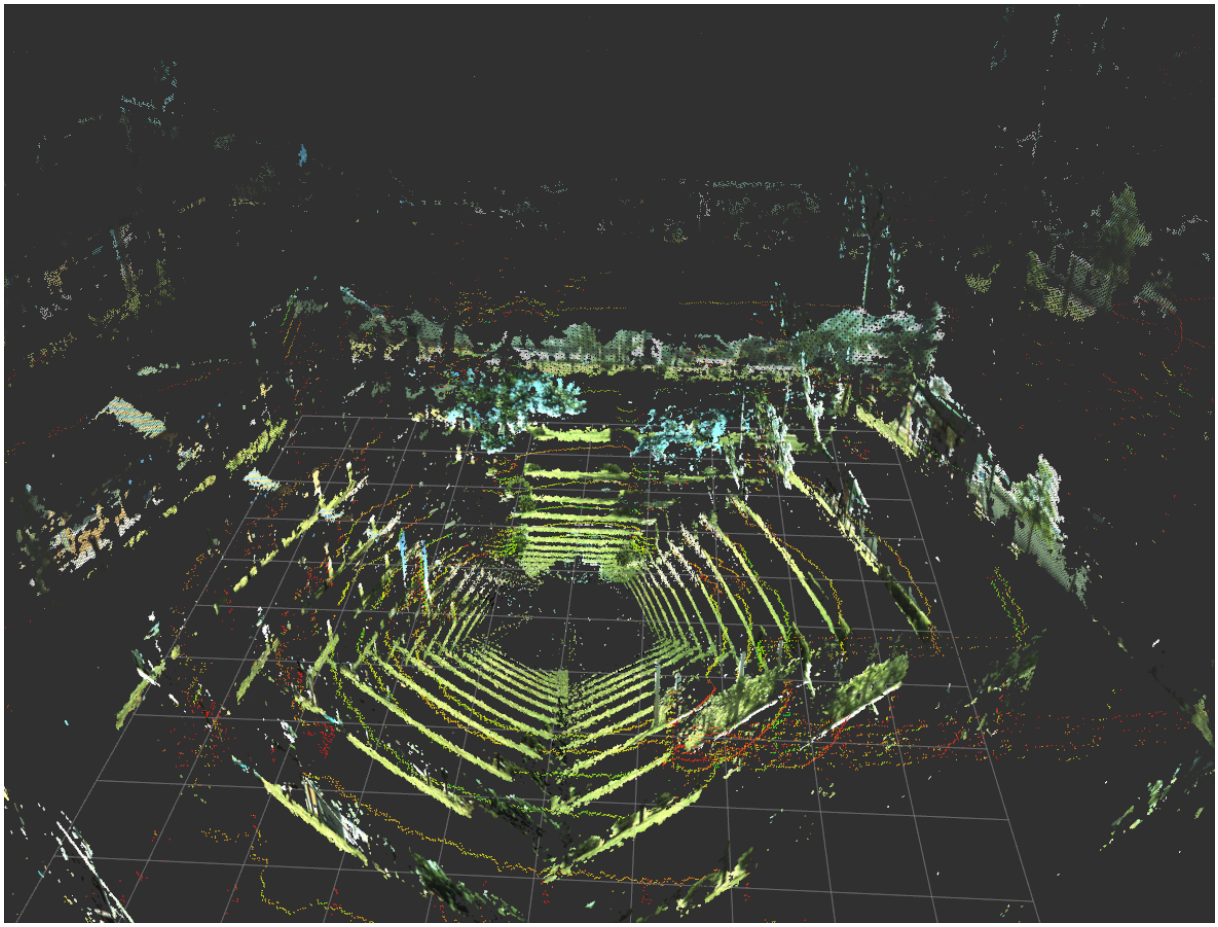


Figure 7: Side view of the garden scene as shown in Fig. 5. All five disparity images are used to project image points in 3D. The integer-based disparity matching with 32 values results in discrete distance measurements with increasing step size.

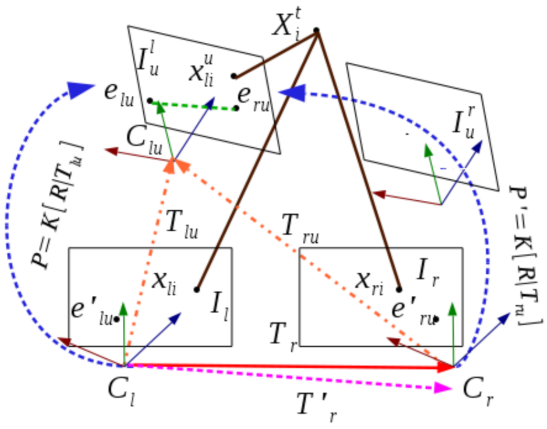



Figure 8: Trinocular stereo scheme shows the geometric relations among left camera $C_l$, right camera $C_r$ and next left frame camera $C_{lu}$. See [45] for details.

about the epipolar geometry of a rectified camera pair to guide the CNN computation (this guided matching is not strictly necessary but leads to better results [43]). The results of the epipolar matching step traverse the rest of the CNN, and the output is directly a disparity map (postprocessing is possible, but not required).

The output of the network is a dense disparity map by default, but depending on the input geometry (invalid areas due to rectification) and program configuration (consistency checking etc), the final result will generally not contain valid information for every input pixel. Invalid pixels are marked as such. The software package is modular: new DispNet variants (even when using different architectures) can easily be added.

The DispNet is suitable for realtime disparity estimation (ca. 4 to 12 frames per second, depending on the specific network and runtime configuration), but occupies significant shared hardware resources to achieve this.

## 3.3 3D from (Trinocular) Stereo (UEDIN)

We have developed two algorithms to accurately estimate stereo disparity, one using the same rectified input pair as in the previous section, the other additionally considers subsequent frame from the left camera.

**3D Plane Labeling Stereo Matching.** The first algorithm [44] exploits both the underlying 3D structure and image entropy to generate an adaptive matching window. The real-valued disparity maps are estimated by smartly exploring a 3D search space using a novel hypothesis generation approach that acts like a propagation scheduler. The employed MRF model makes the assumption that each pixel has associated a set of planes from which a single optimal plane assignment is found using TRW-S propagation.

**Trinocular stereo matching with baseline recovery.** The second algorithm [45] recovers the rigid transformation that describes the displacement of a binocular stereo rig in a scene, and uses this to include a third image to perform dense trinocular stereo matching and reduce some of the ambiguities inherent to binocular stereo. The core assumption shown in Fig. 8 is that the binocular baseline is projected to the third view, and thus can be used to constrain the transformation estimation of the stereo rig. Our approach shows improved performance over binocular stereo [44], and the accuracy of the recovered motion allows to compute optical flow

from a single disparity map.

The CPU implementation for the core MRF inference used by both variants was optimized for accuracy, with typical computation time of 10 min per disparity map. As such it is not suitable for real-time integration, but can be used for construction of dense 3D maps of the garden offline.

# 4 3D Data Fusion (UEDIN)

The previously mentioned stereo sensor rig (Sections 3.1,3.2) outputs images and 3D information at high frame rates, which is useful for dynamic visual feedback, but results in transmission of large and redundant (overlapping) 3D data. Particularly in the case of FPGA the output is quantized and noisy, which limits its ability to estimate shapes of objects directly. Fusion of the produced 3D data stream is required to give better shape estimates by averaging multiple sensor inputs, and also to absorb its redundancy using different 3D representations better suitable to the subsequent 3D understanding tasks.

## 4.1 Representations

There are specific 3D representations needed to accomplish tasks assigned to visual components. For this purpose the diagram in Figure 9 suggests how various input data are transformed (fused) to intermediate 3D representations (maps).

They can be either *local* (relative to the current robot/camera pose) or *global* (in world/map coordinates). Knowledge of the current *global pose* (translation, rotation) described in Sec. 2 is used to update the global map with local data. Generally, local representations miss wider context but can be updated faster, which makes them suitable for tasks where real-time feedback is required. In contrast, global representations accumulate information over a period of time, resulting in lower update frequency, but can cover shape of larger objects, such as hedges.

In particular the following representations were considered to hold 3D information, with their key properties and ROS message type listed:

- Disparity Map (`stereo_msgs/DisparityImage`)
  - intrinsic to input image data
- Point Cloud (`sensor_msgs/PointCloud2`)
  - accurate 3D locations
  - details and thin structures
- Voxel Grid (`trimbot_msgs/Volume`)
  - orthogonally structured (neighbors)
  - limited flexibility (variable size of voxels / octree)
  - unseen space notion
- Surface Mesh (`shape_msgs/Mesh`)

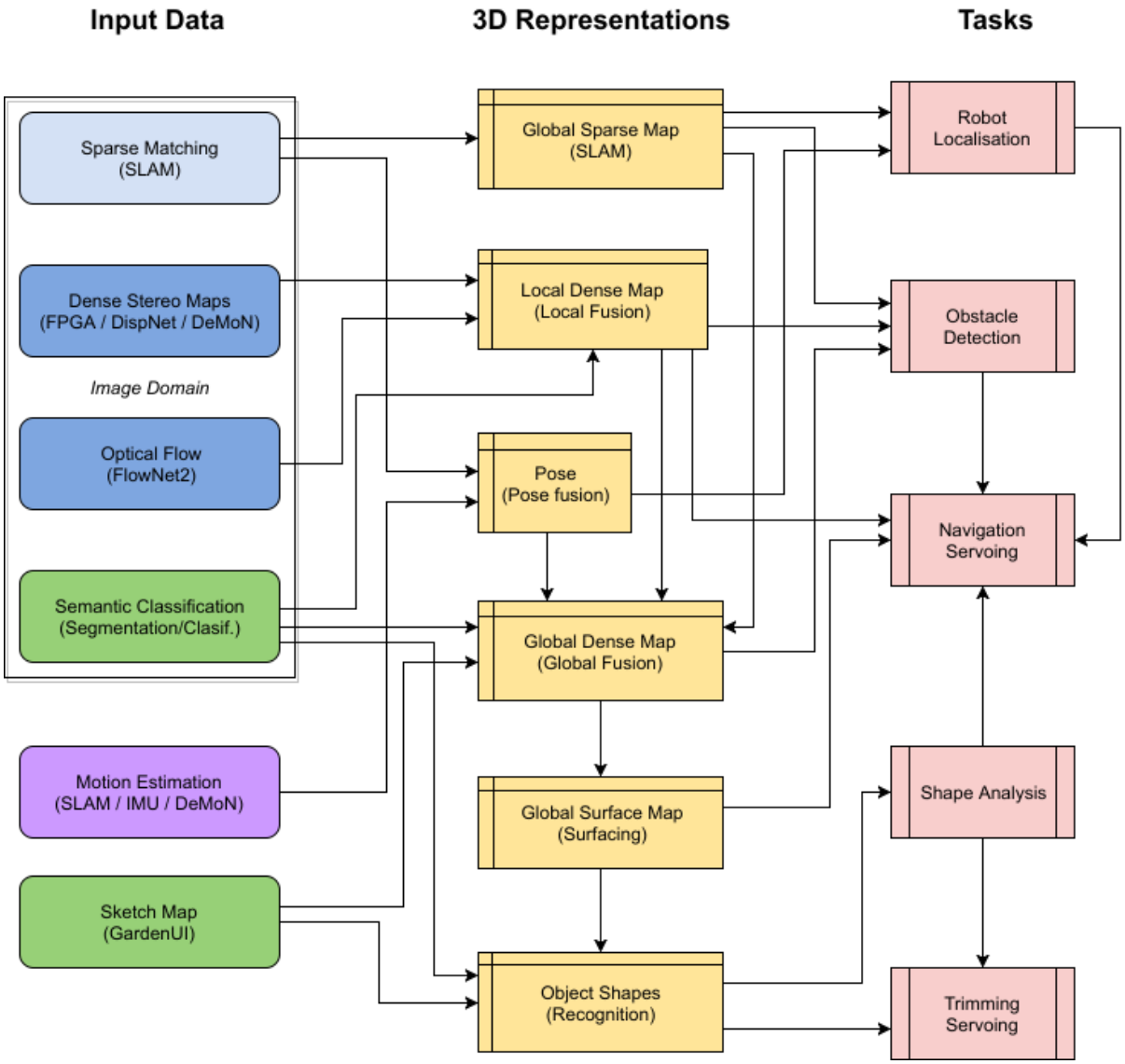
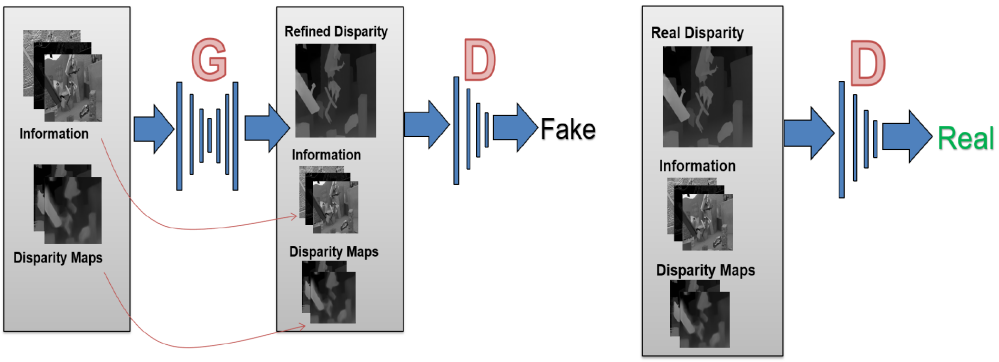



Figure 9: 3D representations with components producing them (in brackets) and their relations to tasks.




a) negative samples b) positive samples

Figure 10: Overview of SDF-MAN architecture. The refiner $G$ tries to predict a refined disparity map close to the ground truth. The discriminator $D$ tries to discriminate whether the input (from $G$) is fake or true (given ground truth). The generator and discriminator can see both the supplementary information and disparity inputs simultaneously as the conditional information.

  - structured intrinsically to the object
  - flexible (variable size of faces)
  - inside/outside notion

- Parametric Surfaces (`trimbot_msgs/ProtoObject`)
  - primitive shapes / NURBS with control points
  - compact

## 4.2 Local Fusion

The first level of fusion comprises of two steps: 1) incoming depth estimates from two inputs corresponding to the same time instant are combined and filtered, then 2) refined disparity maps of frames consecutive in time are projected to point clouds and locally registered to estimate their relative pose.

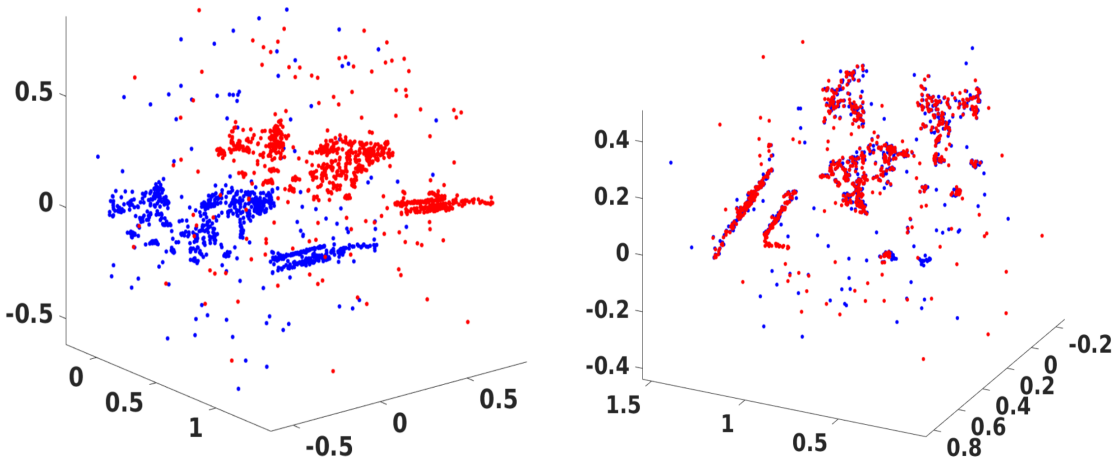
### 4.2.1 Instant Fusion

For a given current synchronized image frame set, there are two sources of disparity information - FPGA (Sec. 3.1) and DispNet (Sec. 3.2). Uncertainties and complex disparity relationships between neighboring pixels limit the accuracy and robustness of individual methods. We fuse the disparity maps from the two different algorithms to exploit their complementary advantages.

For this purpose we have introduced an SDF-MAN method [46] to incorporate supplementary information (intensity, gradient constraints etc.) into a Multi-scale Adversarial Network to better refine each input disparity value (Figure 10). By adopting a multi-scale strategy, the disparity relationship in the fused disparity map is a better estimate of the real distribution. The approach includes a robust object function to avoid blurry edges, impaints invalid disparity values and requires less ground truth data to train.

The TensorFlow implementation of the method runs at interactive frame rates.




a) before registration b) after registration

Figure 11: Example of aligning two noisy point clouds from the garden with outliers and varying densities using DUGMA.

### 4.2.2 Temporal Fusion

We first convert instantly fused disparity maps to point clouds, using the stereo baseline and camera parameters known from the initial calibration. Consecutive point clouds are then registered to get their relative transform using iterative algorithm described below, which could provide more accurate pose estimate for dense reconstruction than SLAM.

We have proposed a simple architecture combining error estimation from sample covariances and dual dynamic global probability alignment using the convolution of uncertainty-based Gaussian Mixture Models (GMM) from point clouds. DUGMA [47] incorporates the 3D uncertainty distribution of each 3D point from a sensor into a dynamic Gaussian mixture alignment system.

Unlike the invariant GMM (representing a fixed point cloud) in traditional Gaussian mixture alignment, we use two uncertainty-based GMMs that change and interact with each other in each iteration. In order to have a wider basin of convergence than other local algorithms, we design a more robust energy function by convolving efficiently the two GMMs over the whole 3D space. The iterative optimization is based on EM scheme, alternating between covariance updates and pose estimation.

The current Matlab/CUDA implementation needs around 1 min to register a pair of point clouds subsampled to 5000 points in a fixed number of 50 iterations. Its real-time optimization is in progress.

## 4.3 Global Fusion

On the next level we incrementally update a global volumetric representation with local (previously fused) data given its localization and derive a surface mesh from it. For this purpose this package subscribes to the locally fused point clouds (Sec. 4.2.1) and poses obtained from localization (Sec. 2) and potentially also other sources as listed below.

### 4.3.1 Global Pose Fusion

We can take into account multiple sources of estimated motion to provide the current global pose:

- Visual odometry (SLAM) - global pose,
- IMU - acceleration,
- Track odometry - velocity,
- Point cloud registration (DUGMA) - relative pose.

For the most of the operational time, we can directly make use of the SLAM-provided global pose. For periods when it becomes unavailable, we need to integrate other sources of motion to extrapolate the global pose. At the same time, they can constrain the SLAM estimate to reject invalid poses that jump too far from the previous location, ie. perform outlier rejection. For this purpose we implemented an Extended Kalman Filter node [48], which handles different sampling rates of inputs, and takes into account their uncertainty. It is able to process delayed, relative and absolute measurements from different sensors while allowing self-calibration of the sensor-suite online. The EKF is reset when the estimated pose uncertainty exceeds a given threshold. Details and evaluation can be found in *D3.5 - Vehicle and arm localization*.

### 4.3.2 Volumetric Fusion

We use a variational approach [49] which takes a set of oriented points with scale information as input. This requires normals to be calculated first on the incoming local point clouds and normals oriented towards the viewpoint. The scale is set proportionally to the distance from the camera (depth). The points are then transformed to the global coordinates using the refined viewpoint pose and such point clouds from a given time period are merged in a single point cloud.

The algorithm integrates the scale information in the objective and globally optimizes the signed distance function of the surface on a balanced octree grid to adapt to the data. Using a finite element discretization on the dual structure of the octree minimizing the number of variables, the algorithm is able to handle seamlessly nonuniform point densities due to images taken from different distances.

The resulting volumetric structure (Figure 12) describes the occupancy of the scene by means of the signed distance value for each octree cell, which allows to segment the space into free, occupied/surface or unseen/inside voxels.

### 4.3.3 Surface Extraction

Using the same framework [49] as above for the volumetric fusion, a tetrahedral mesh is generated efficiently with a lookup table which allows to map octree cells to the nodes of the finite elements. The memory efficiency is optimized by data aggregation, such that robust data terms can be used even on very large scenes. The surface normals are explicitly optimized and used for surface extraction to improve the reconstruction at edges and corners.

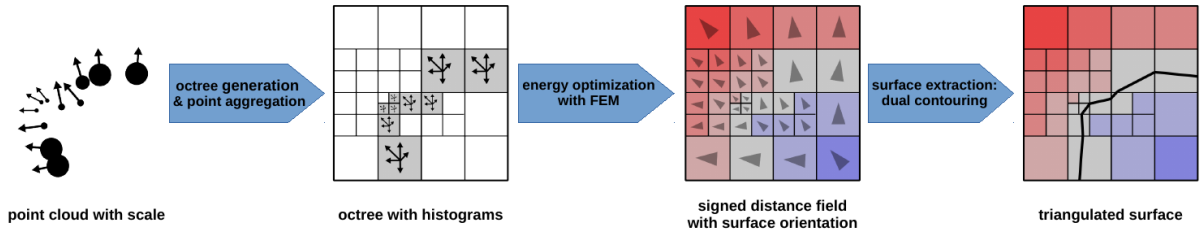



Figure 12: 3D point fusion overview scheme. Volume color legend: *red* = seen/free, *grey* = surface/occupied, *blue* = unseen/occupied

# 5 Experimental Evaluation

In the following, we evaluate the different components, i.e., the multi-camera SLAM, the depth perception, and the data fusion pipelines, on data recorded in the Wageningen test garden. Note that this evaluation often complements the experimental results presented in the original publications.

## 5.1 SLAM and Vehicle Localization

In the evaluation of our SLAM system, we analyze the performance of the offline mapping and the online localization components. The mapping component creates a 3D reconstruction of the garden from scratch from a relatively long recording session of the scene without any prior information. The localization component then uses this pre-built map to localize the robot within the scene.

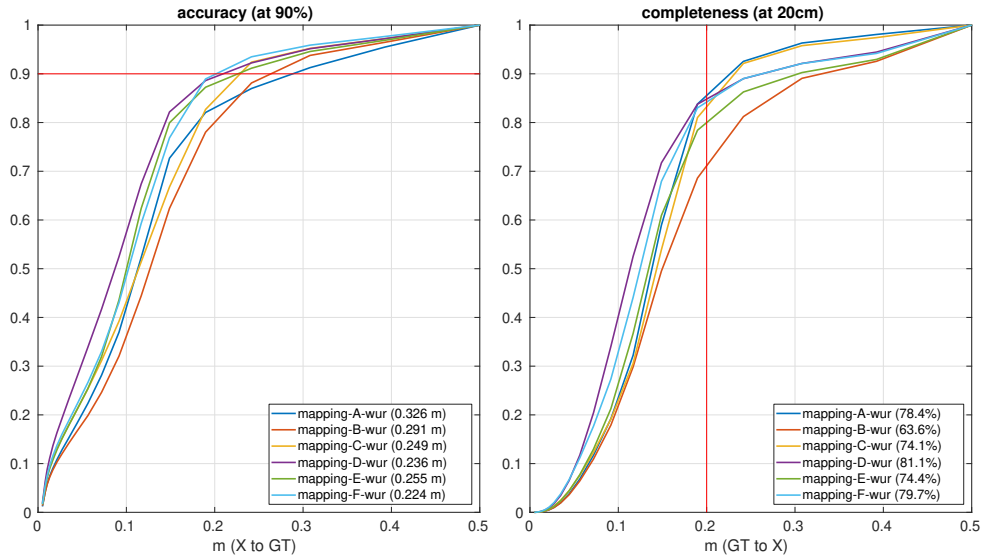
**Experimental setup** To analyze the two components, we recorded 6 mapping sessions and 25 localization sessions in the test garden in Wageningen. The different sessions were captured at different times of the day. The sessions were recorded during different times of the day over the course of 3 consecutive days in June 2018. As described in Deliverable D7.4 (*Ground-truth data definitions and acquisition*), a ground truth 3D map of the garden was obtained using a laser scanner. During the recording sessions, the ground truth location of the robot was tracked using a Topcon TotalStation. The tracked locations and camera streams are synchronized in order to allow for an accurate alignment and comparison in the evaluation of the SLAM system.

### 5.1.1 SLAM Evaluation

We evaluate the performance of the mapping component in terms of the accuracy of the reconstructed 3D map and the reconstructed trajectory.

**Map quality** To evaluate the accuracy of the reconstructed 3D map of the SLAM system, we align the reconstructed 3D map to the ground truth by manually selecting multiple corresponding reference points and by estimating an over-determined 3D similarity transformation. Fig. 14 shows the 6 maps constructed by our method. Note that due to tracking errors in the map construction step, our SLAM system might produce multiple maps per recording session, as our system instantiates a new map whenever tracking fails. In our evaluation, the majority of frames (more than 99%) could be reconstructed into one single map of the environment and we ignore any of the other small maps.



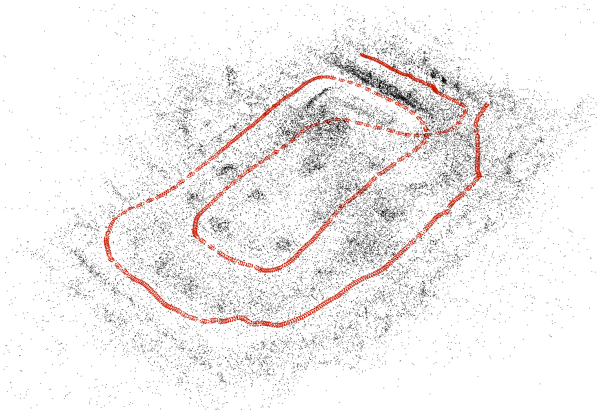

(a) (b)

Figure 13: Evaluation of the SLAM mapping quality: Cumulative histograms over the accuracy (13a) and completeness (13b) of the SLAM maps for different distance thresholds for the 6 different maps recorded in the Wagening garden. For each map, we also report the threshold $t$ for which 90% of the SLAM points have a distance of $t$ or less to the closest laser point (about 25 cm) and the completeness at a threshold of 20 cm (about 70% to 85% of the points).

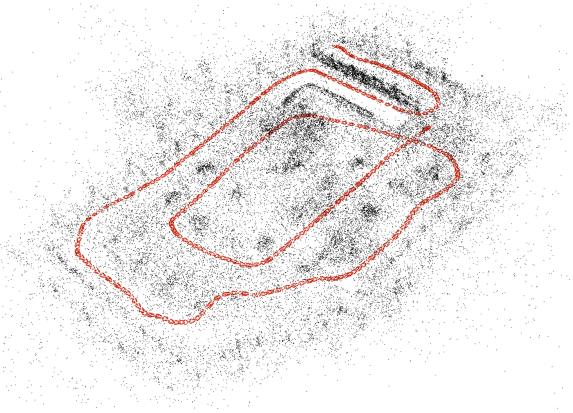
Given the alignment, we measure the accuracy and completeness of the SLAM map. Following [50], the accuracy of the sparse SLAM point cloud for a given distance threshold $t$ is defined as the fraction of SLAM points within distance $t$ to the laser scan point cloud. Similarly, completeness is defined as the fraction of laser scan points within a given distance $t$ to the SLAM point cloud. Fig. 13 shows the accuracy and completeness as a function of $t$ for the 6 recordings. Fig. 15 and 16 visualize the accuracy and completeness for the 6 maps we use for evaluation. As can be seen, obstacles such as bushes are reconstructed rather accurately. Thus, the SLAM point cloud could be used to define potential obstacles by segmenting them from the ground. This was actually done for the first demonstrator. Objects further away from the area where the robot drove are less accurately reconstructed due to a higher triangulation uncertainty.

**Trajectory quality** To evaluate the accuracy of the reconstructed SLAM trajectory, we align the SLAM trajectory to the ground truth. As the tracked locations and the camera streams are not measured at the same point in time, the alignment is done as follows: for each camera frame, we find the two closest ground truth locations in time. If one of the two ground truth locations has a time offset of more than 0.2s (e.g. when the laser tracker lost the target for a short period of time), we ignore this camera location in the evaluation, as we assume there is no accurate ground truth available. Otherwise, we determine the ground truth location of the robot using linear interpolation between the two ground truth locations, which is scaled accordingly by the two corresponding time offsets. Using all corresponding locations in the reconstructed trajectory and the ground truth, we estimate a 3D similarity transformation using a LO-RANSAC [51] routine (with an alignment threshold of 0.2m) to account for potential outliers in the reconstructed trajectory or the ground truth (in very rare occasions the laser tracker failed during our experiments).

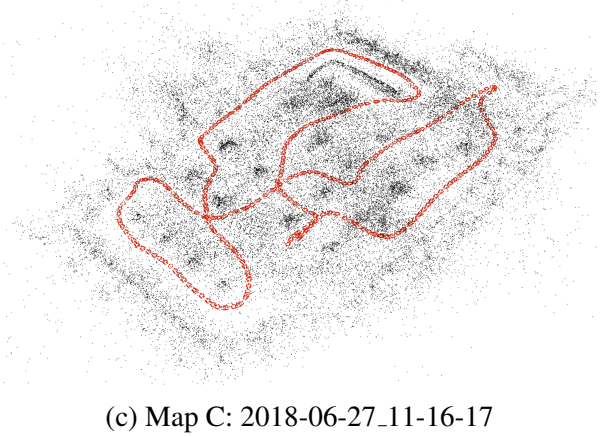
 



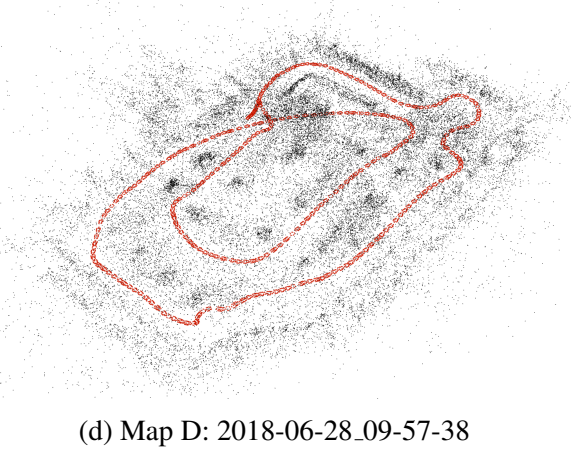
(a) Map A: 2018-06-26_12-00-42



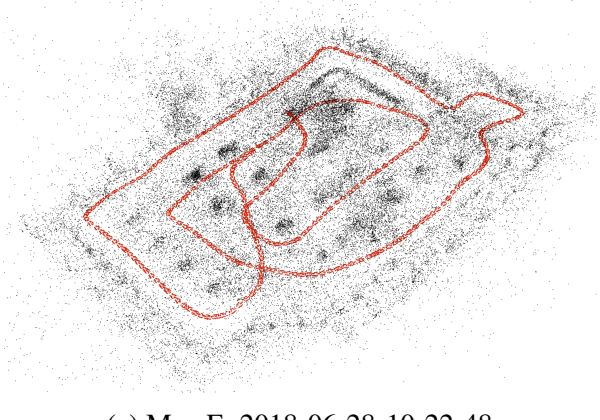
(b) Map B: 2018-06-26_14-04-57



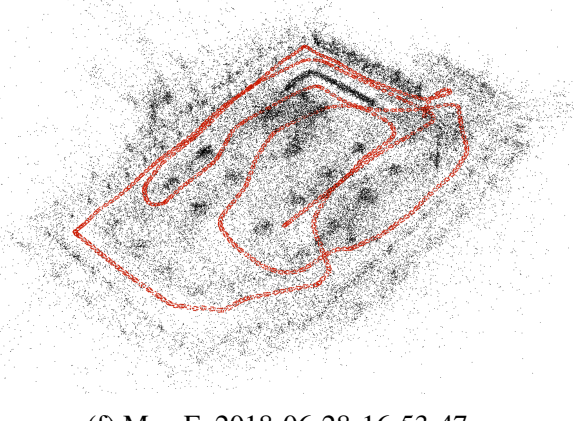
(c) Map C: 2018-06-27_11-16-17



(d) Map D: 2018-06-28_09-57-38



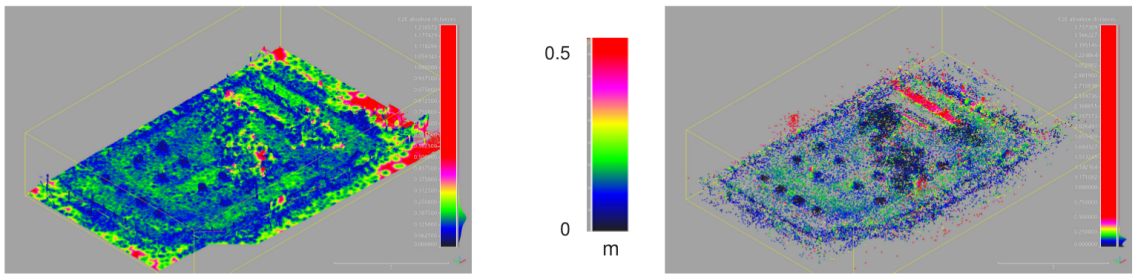
(e) Map E: 2018-06-28_10-22-48



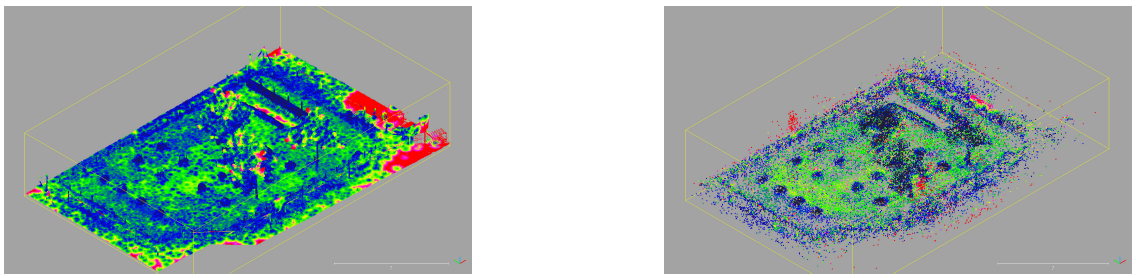
(f) Map F: 2018-06-28_16-53-47

Figure 14: Visualization of the 6 maps built offline by the SLAM system. The 3D point cloud is shown in black while the estimated camera trajectory is shown in red. For each map, we specify the recording date of its trajectory.

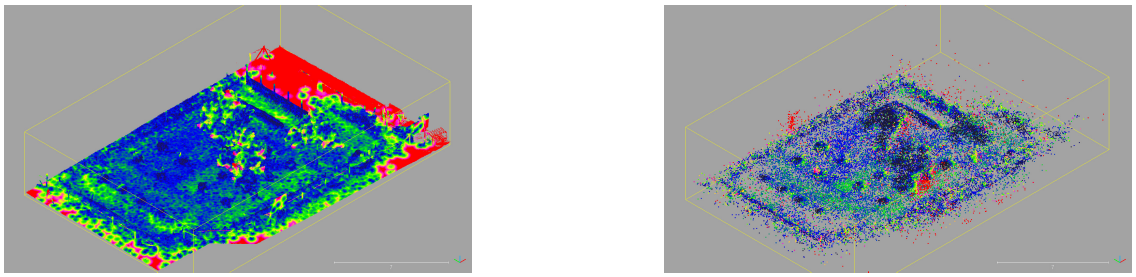
 




(a)



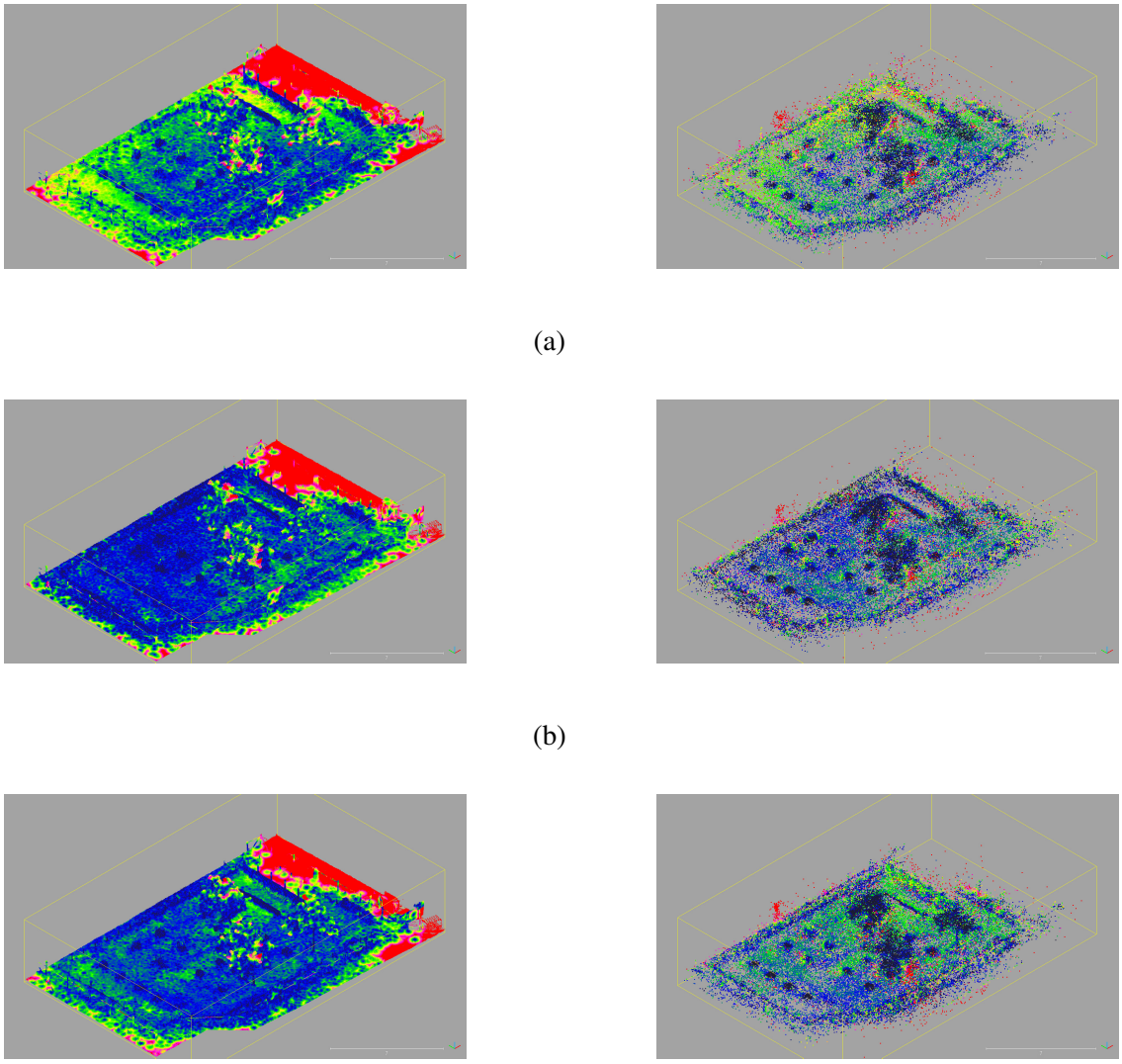
(b)



(c)

Figure 15: Visualization of the accuracy and completeness of the SLAM maps for the first three of the 6 mapping runs. For each subfigure, the left plot shows the completeness of the map, visualized by color-coding the distance of each laser scan point to the closest SLAM point. The right plot shows the accuracy of the SLAM point cloud, again color-coded according to the distance to the nearest laser scan point. Warm colors correspond to large, cold colors to small distances. Completeness (esp. near garden borders) largely depends on the trajectory of the robot, ie. if the scene part was visible from the robot and from how far.



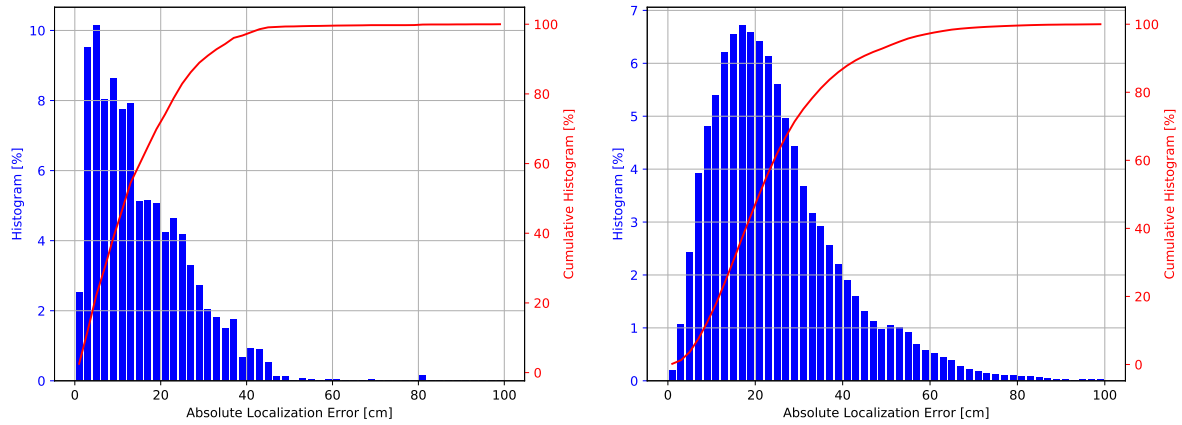
(a)

(b)

(c)

Figure 16: Visualization of the accuracy and completeness of the SLAM maps for the last three of the 6 mapping runs. For each subfigure, the left plot shows the completeness of the map, visualized by color-coding the distance of each laser scan point to the closest SLAM point. The right plot shows the accuracy of the SLAM point cloud, again color-coded according to the distance to the nearest laser scan point. Warm colors correspond to large, cold colors to small distances (see Fig. 15 for the mapping from colors to values).




(a) (b)

Figure 17: Accuracy of the SLAM (17a) and localization (17b) trajectories compared to the Topcon tracker ground truth. For the SLAM trajectories, we show the distribution of errors over all 6 mapping runs. For the localization trajectories, we localize 25 sequences captured at different times against maps recorded at different times of the day (see Sec. 5.1.2).

We the compute the alignment error between the reconstructed trajectory and the ground truth as the spatial distance between corresponding locations. Fig. 17a shows histograms for the results from all 6 mapping runs. As can be seen, the position estimated by the SLAM system is typically within 20cm of the ground truth position. Larger errors are not results from sudden jumps in the trajectory, but are caused by drift. One possibility to reduce the errors would be to record the mapping trajectories more carefully to ensure that many loop-closures can be detected. Another possibility would be to include the semantic constraints described in Sec. 2.8. However, the ability of Semantic SLAM to reduce drift strongly depends on the quality of the semantic segmentation. Unfortunately, we haven't been able to evaluate Semantic SLAM on TrimBot2020 data so far as publicly available semantic segmentation algorithms do not cover the classes relevant for the project.

### 5.1.2 Vehicle Localization

We also evaluate the performance of the localization component of our SLAM system in terms of the accuracy of the estimated robot location. Given an existing map, built offline using the SLAM system, the localization part aims to localize the vehicle inside this garden. It then combines the localization results with SLAM-based camera pose tracking.

For the evaluation, we use the already aligned maps from the evaluation of the mapping component as an input to the localization system. Localization computes the pose of the robot with respect to these maps, which are already in the same reference system as the ground truth. Thus, we can directly compare the localization results with the ground truth positions provided by the Topcon tracker and no further alignment is needed for this evaluation. Equivalent to the evaluation of the accuracy of the reconstruction trajectory during the mapping, we determine corresponding ground truth locations by linear interpolation of the two closest laser tracker locations in time. We measure the localization accuracy as the spatial distance between the so determined corresponding locations.

We evaluate all combinations of 6 maps and 25 localization scenarios captured at varying times of the day, i.e., a total of 150 scenarios. This is done to measure the robustness of the localization system to appearance changes over time. We categorized the different times of the day into morning (before 11:00am), midday (11:00am-01:00pm), and afternoon/evening (after 01:00pm) in order to gather summary statistics for cross-time localization.

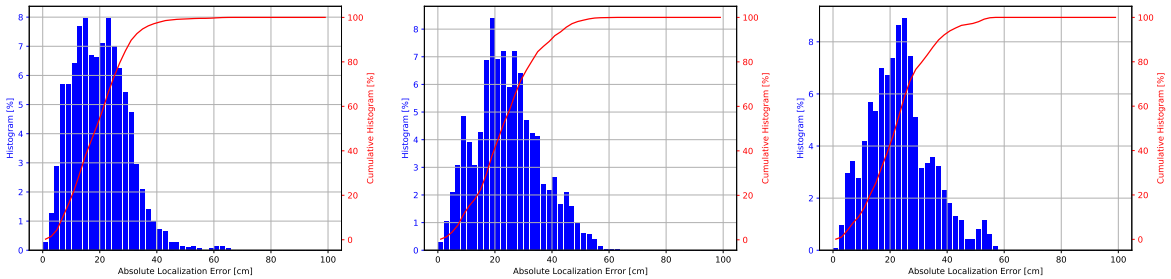
Fig. 17b shows histograms for the position accuracy measured over all 150 scenarios. Fig. 18 provides a more detailed evaluation, showing the errors in the estimated positions based on the times at which the query sequence and the map were recorded. Naturally, we see that localizing images, which were taken at the same time of the day as the map, leads to more accurate estimates. This is due to a smaller variation in scene appearance, which typically results in more matches between the query sequence and the map. In turn, more matches typically translate to a higher localization accuracy. We observe that localizing against maps built during midday is harder than localizing maps recorded during the morning or the evening in Fig. 18). At the same time, localizing sequences recorded during midday are harder to localize than sequences recorded during other times of the day, independently of the time of the map (cf. middle column in Fig. 18). This behavior is due the strong appearance changes, e.g., shadows caused by the sun.

To develop a better understanding of the localization procedure, we analyze some of the scenarios for which large localization errors occur. Fig. 19-24 show two scenarios each. The first scenario corresponds to a hard case, where the position error is high (typically 20cm or more). The second scenario shows the result of the same query sequence on another map, selected such that the error is as small as possible. While there is some drift in the trajectories, e.g., in Fig. 21, the largest part of the error seems to be systematic. This can be seen from the fact that the accuracy of the estimated positions depends on the map used for localization. As such, part of the localization error seems to come from a registration error between the map and the ground truth. In order to reduce such errors, one could try to build the maps using more features per camera, which typically translates to more loop closure detections and stronger constraints between generalized images. However, it currently also results in longer online run-times as the localization system needs to localize against a larger map with more features. Compressing the map after its creation by selecting a subset of all points, e.g., using a method recently proposed by ETH [52], might help avoid this problem.

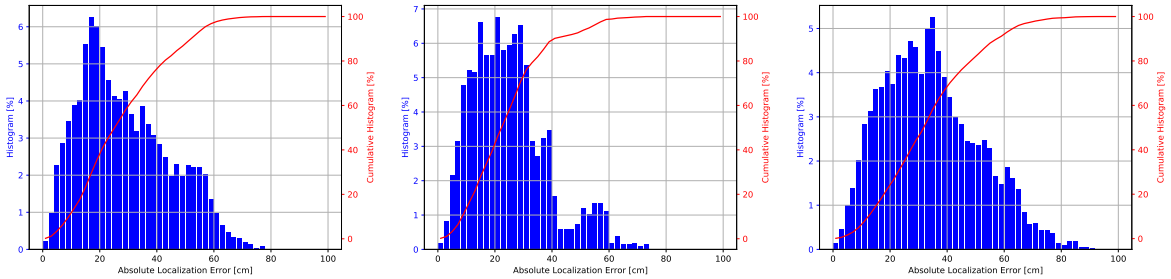
Looking at Fig. 19-24, there seem to be only small jumps in the estimated positions between frames. Fig. 25, which shows the distribution of jumps in the estimated positions between frames over all scenarios. As can be seen, the position estimates are stable up to a few centimeters.

We note that in the TrimBot2020 project, where the goal is to build an autonomous gardening robot, it is possible to choose the time of the day at which the robot operates. Based on the results, we recommend operation either in the morning or the evening/afternoon, where the appearance of the garden seems to be more stable. For problem settings where operation under all conditions is required, e.g., self-driving cars, robustness to appearance (and seasonal) changes is obviously desirable. One possible approach to robust longer-term localization is the semantic localization approach recently developed in the project [4]. This approach will be discussed in more detail in D3.5 (*Arm and vehicle localization*). Similarly, we could use the semantic constraints from Sec. 2.8 to further refine the poses obtained from the localization system, similar to [53]. Such an approach could even make use of the laser scan maps (if they were annotated with semantics) for pose refinement. One problem, illustrated in Fig. 26, that we

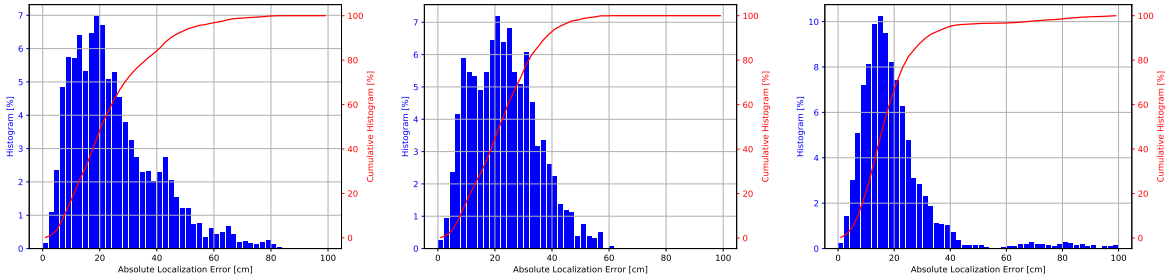
 




(a) Positional accuracy when localizing against a map built in the morning.




(b) Positional accuracy when localizing against a map built during midday.




(c) Positional accuracy when localizing against a map built during the evening.

Figure 18: Accuracy of the positions estimated by the localization system when localizing sequences taken during different times of the day against SLAM maps recorded at different times of the day. The three rows correspond to maps showing the garden at morning, midday, and evening, respectively. The three columns correspond to localizing sequences recorded during morning, midday, and evening, respectively. The diagonal thus shows results for localizing images taken under the same condition as the map.

are currently facing in this setting is that semantic segmentation algorithms are not very robust to changes in scene appearance. This indicates that further research is necessary to enable robust long-term localization based on semantic scene understanding.

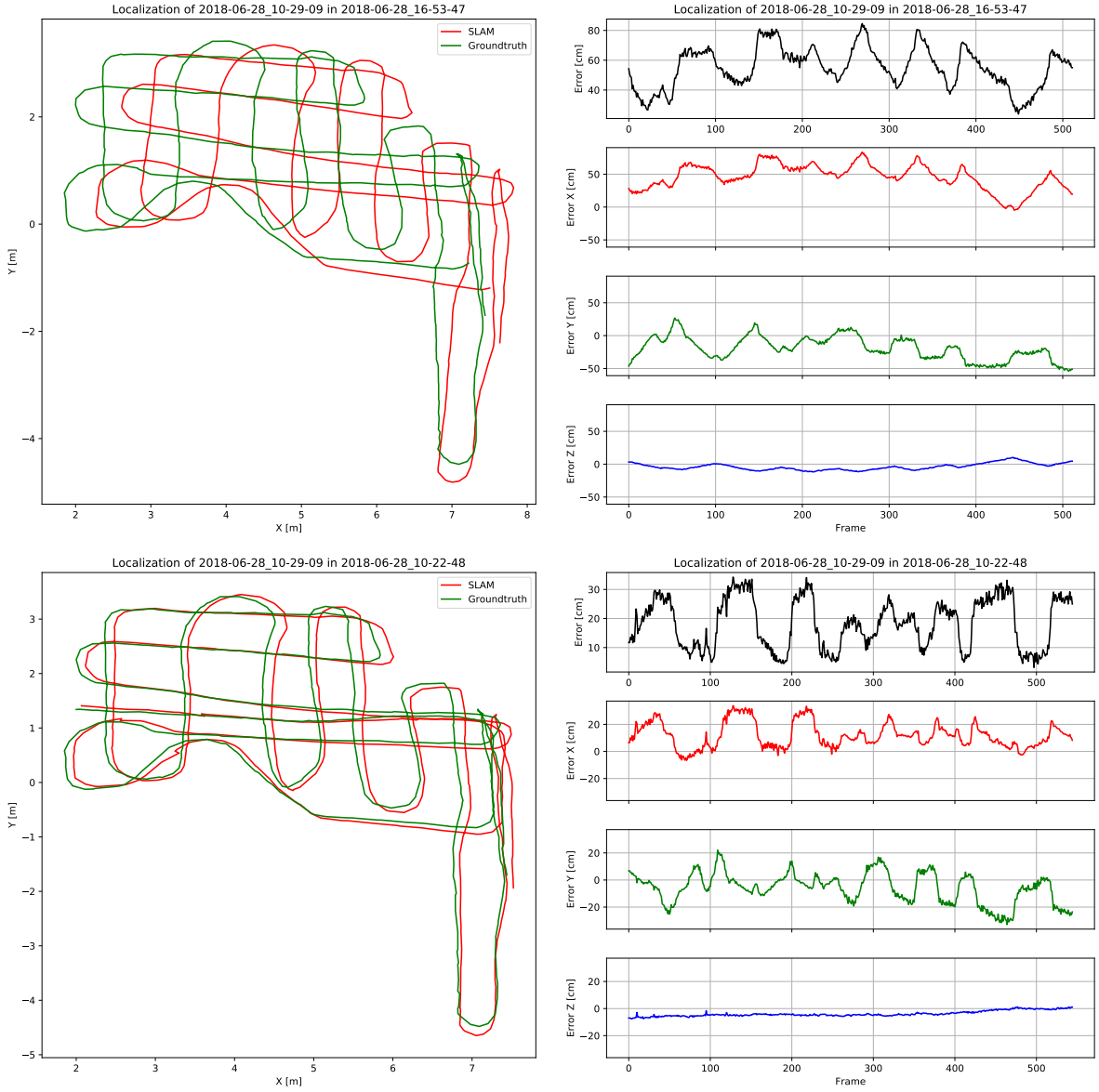



Figure 19: Example for a hard localization scenario (1): We show the localization (denoted as SLAM) and ground truth trajectories and the development of the positional error over time (the z-axis corresponds to the height of the camera). The top row corresponds to a hard localization scenario, trying to localize a morning sequence against an evening map. The bottom row shows the result for the same sequence when localizing against a morning map.

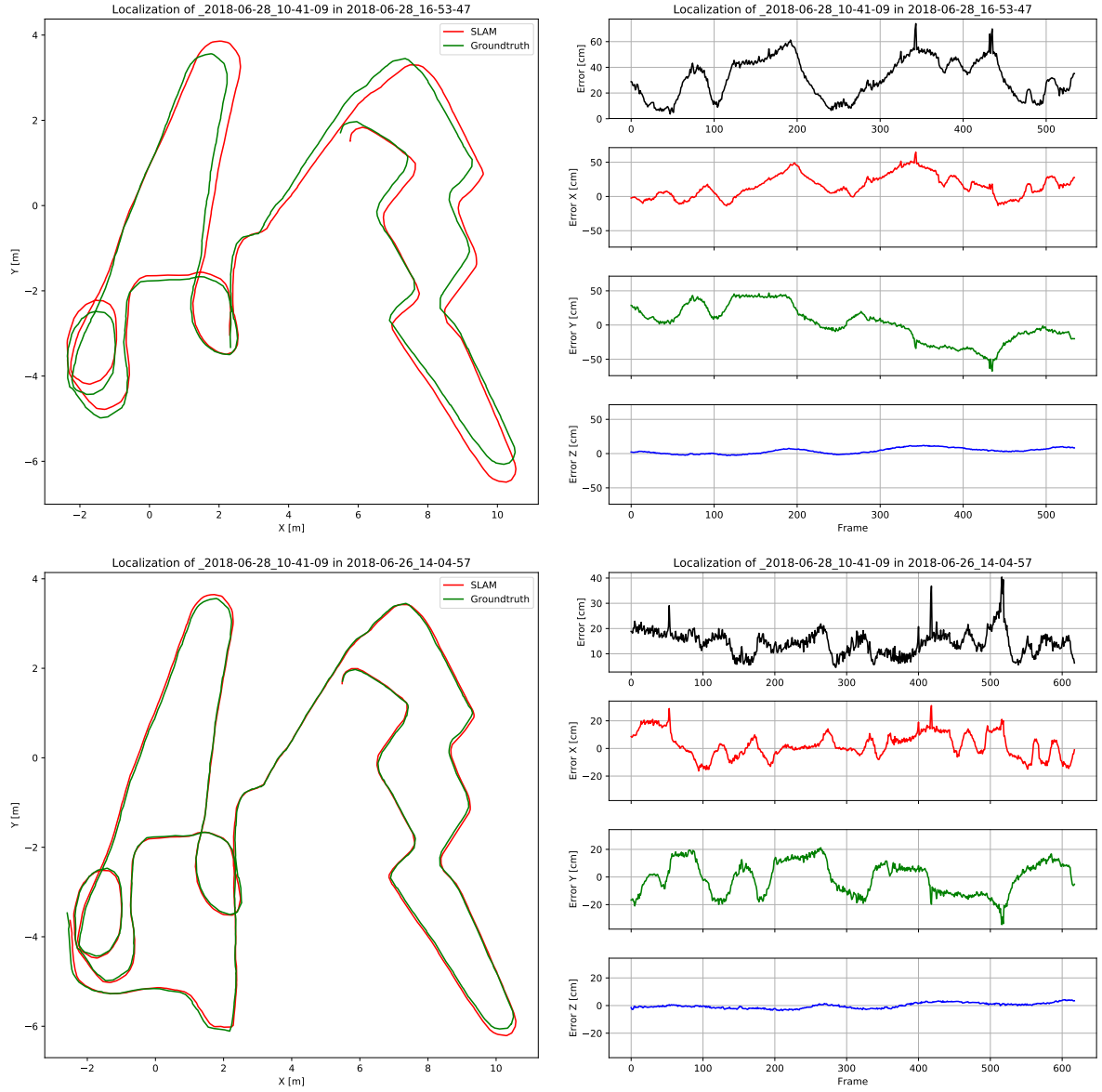



Figure 20: Example for a hard localization scenario (2): We show the localization (denoted as SLAM) and ground truth trajectories and the development of the positional error over time (the z-axis corresponds to the height of the camera). The top row corresponds to a hard localization scenario, trying to localize a morning sequence against an evening map. The bottom row shows the result for the same sequence when localizing against another evening/afternoon map.

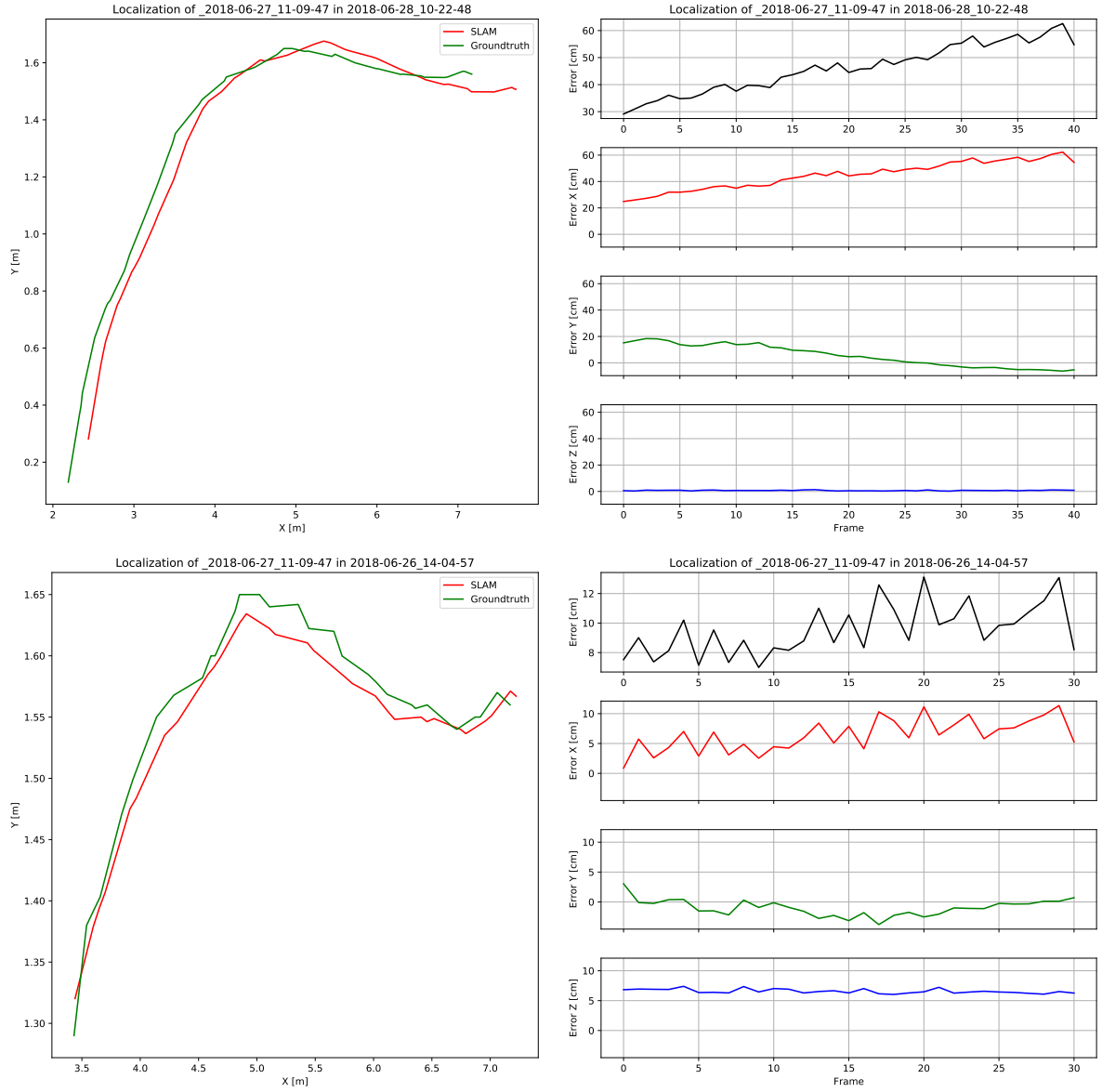



Figure 21: Example for a hard localization scenario (3): We show the localization (denoted as SLAM) and ground truth trajectories and the development of the positional error over time (the z-axis corresponds to the height of the camera). The top row corresponds to a hard localization scenario, trying to localize a midday sequence against a morning map. The bottom row shows the result for the same sequence when localizing against an evening map.

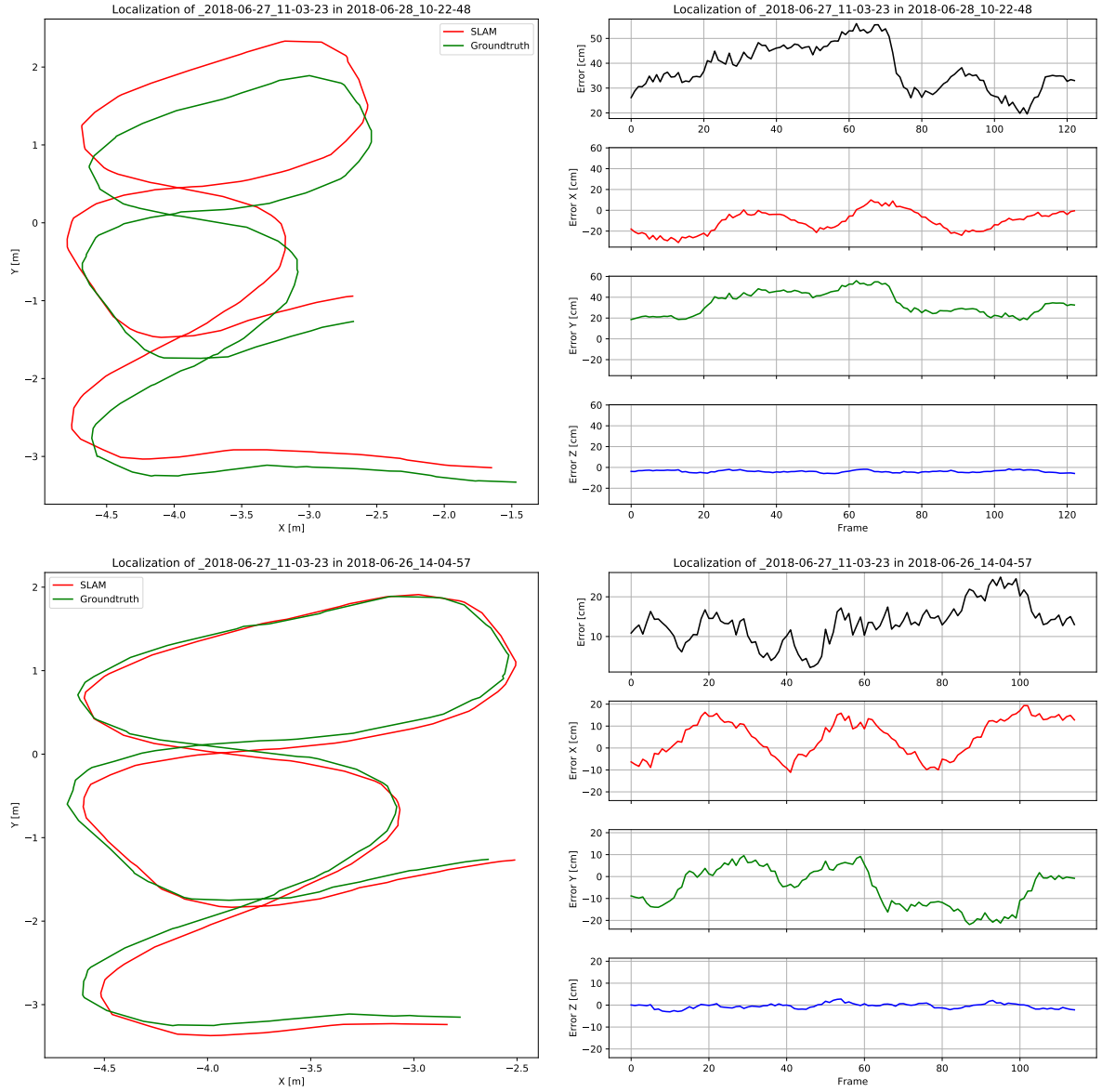



Figure 22: Example for a hard localization scenario (4): We show the localization (denoted as SLAM) and ground truth trajectories and the development of the positional error over time (the z-axis corresponds to the height of the camera). The top row corresponds to a hard localization scenario, trying to localize a midday sequence against a morning map. The bottom row shows the result for the same sequence when localizing against an evening map.

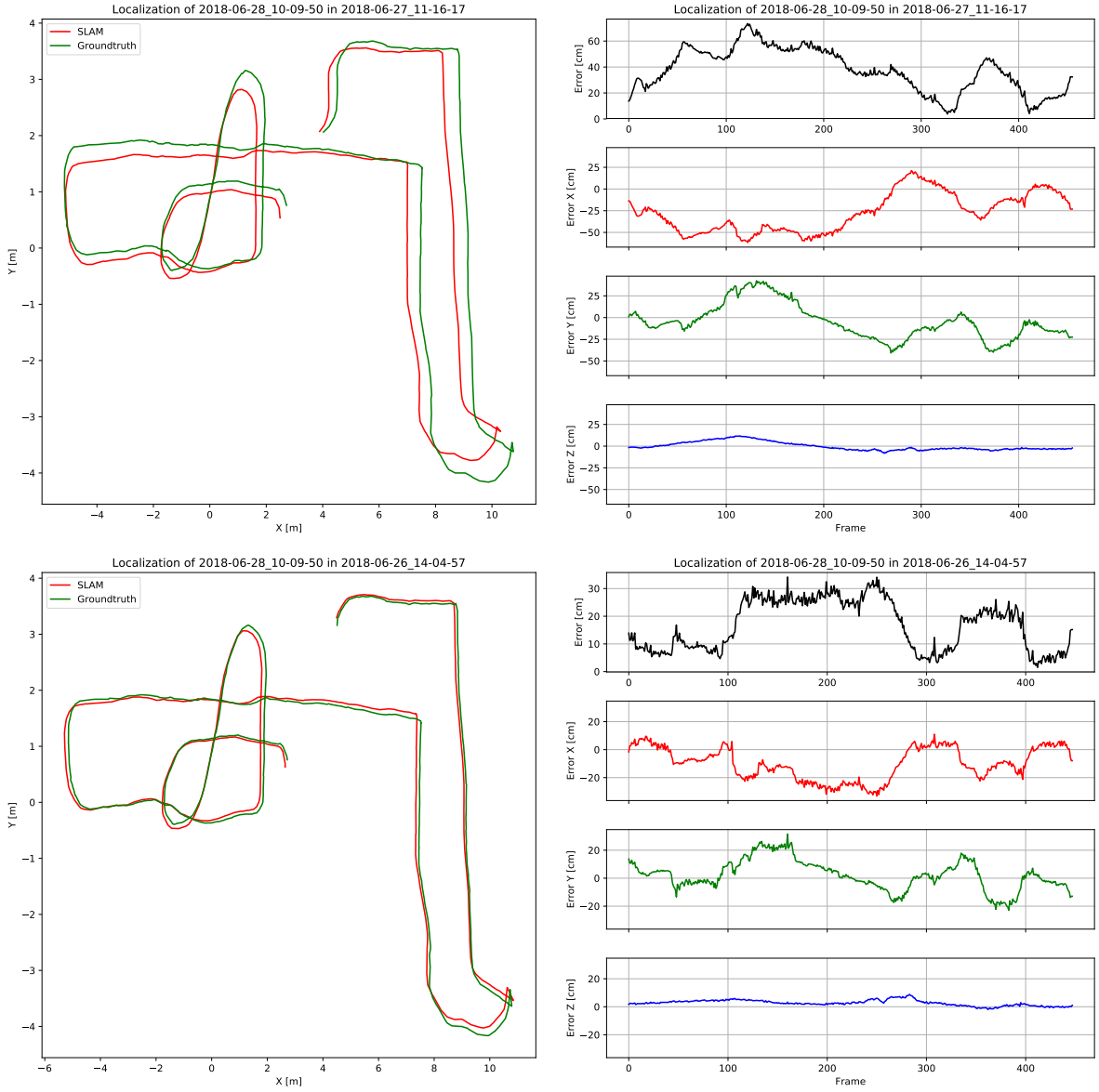



Figure 23: Example for a hard localization scenario (5): We show the localization (denoted as SLAM) and ground truth trajectories and the development of the positional error over time (the z-axis corresponds to the height of the camera). The top row corresponds to a hard localization scenario, trying to localize a morning sequence against a midday map. The bottom row shows the result for the same sequence when localizing against an evening map.

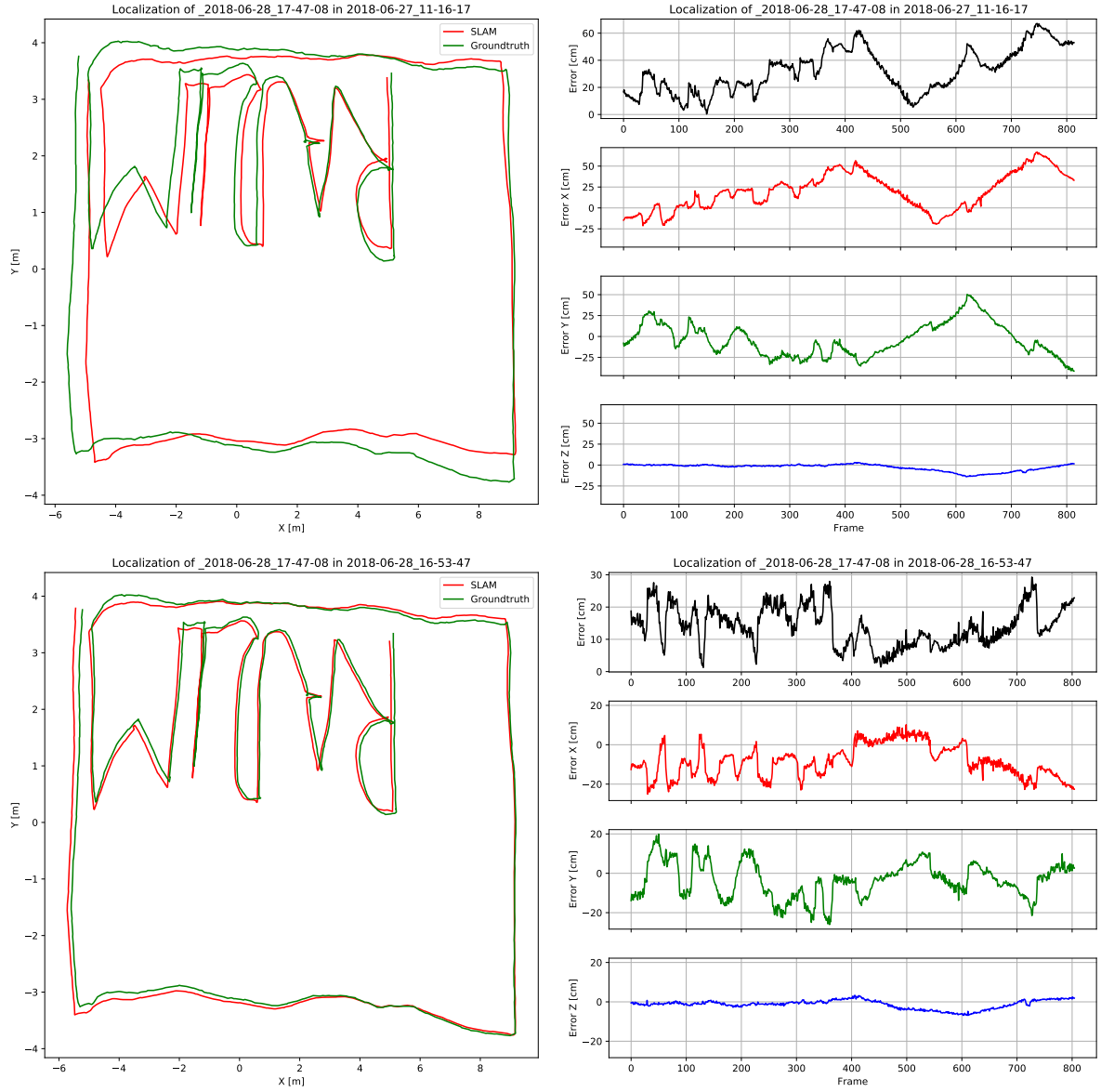



Figure 24: Example for a hard localization scenario (6): We show the localization (denoted as SLAM) and ground truth trajectories and the development of the positional error over time (the z-axis corresponds to the height of the camera). The top row corresponds to a hard localization scenario, trying to localize an evening sequence against a midday map. The bottom row shows the result for the same sequence when localizing against an evening map.

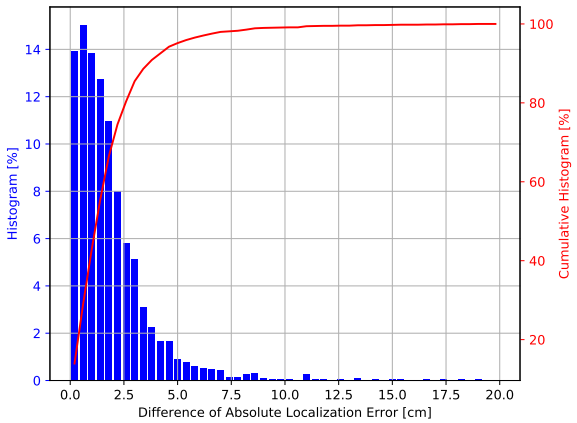



Figure 25: Histograms over the magnitude of jumps in the positions estimated by the localization algortihm between frames.

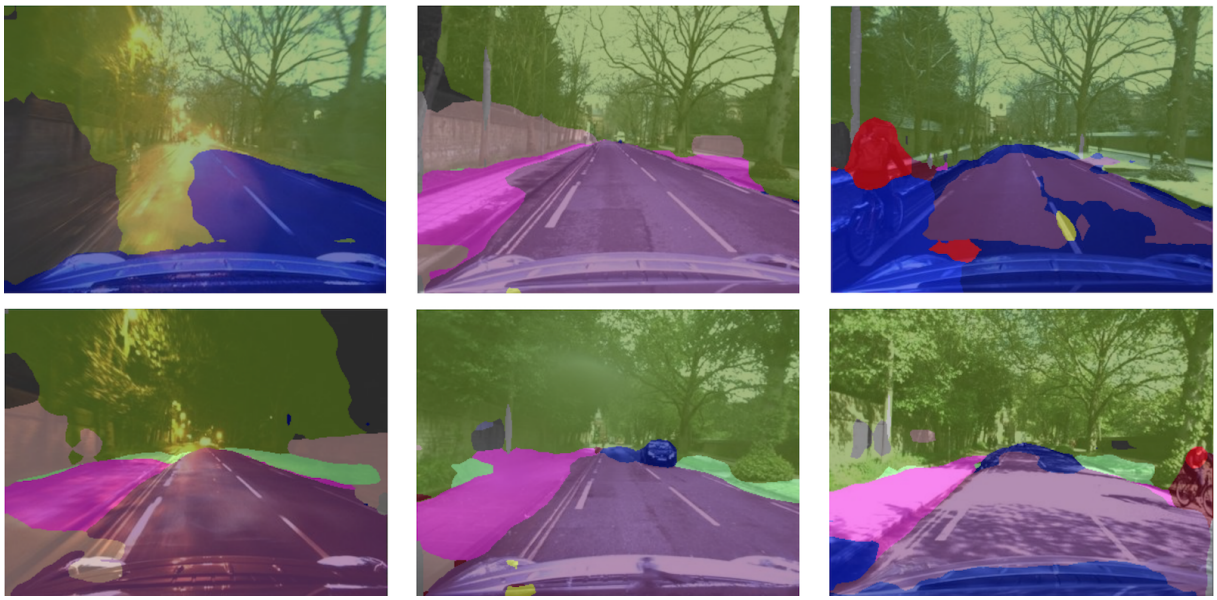


Figure 26: Result of applying a state-of-the-art semantic segmentation algorithm on images depicting the same place (captured in the RobotCar dataset [54]) under different times of the day and different seasons. As can be seen current semantic segmentation algorithms are not robust to changes in scene appearance.

## 5.2 Depth Perception

**Experimental setup** In order to evaluate the quality of the estimated depth maps, we make use of the Wageningen 2017 dataset used for the semantic reconstruction challenge of the ICCV 2017 workshop '3D Reconstruction meets Semantics'. The dataset consists of a 3D laser scan of the garden as well as multiple traversals of the robot through the garden (see Fig. 27). In addition to the challenge dataset (2 camera pairs), we included all 5 camera pairs, obtaining total 1250 sample pairs. Robot poses for the traversals were recorded in the coordinate system of the laser scanner using a Topcon TotalStation. The results were subsequently refined using Structure-from-Motion [26]. The quantitative evaluation is performed only on a subset of pixels which correspond to static non-ground areas (the grass on the ground surface yields noisy GT measurements as well as other moving parts like tree branches).

The accuracy of stereo depth map estimates depends on the distance of the cameras to the scene, with the uncertainty growing quadratically with the distance. In contrast, the uncertainty grows only linearly in the disparity space (measured in pixels). As is common [50, 55], we thus measure the accuracy of the stereo algorithms by comparing their estimated disparity values with the ground truth disparity values provided by the laser scanner.

Following [50, 55], we use the *badX* metric for evaluation, which measures the percentage of pixels whose disparity error is $X$ pixels or more among all pixels that have a ground truth disparity estimate. We use $X = 1, 2, 3, 4$ pixels.

In addition to the quantitative evaluation, we also provide qualitative examples. Besides the Wageningen 2017 dataset, we also use a Velodyne lidar scanner that was mounted on top of the TrimBot2020 robot for the qualitative evaluation. As an example, Fig. 28 shows the disparity image estimated by the FPGA SGM stereo algorithm and the points where the Velodyne sensor estimated a distance. There are about 3000 Velodyne distance measurements present in the field of view of a single camera as shown in Fig. 28b.

### 5.2.1 FPGA Stereo

**Quantitative evaluation.** Fig. 29(top left) shows the mean disparity error between the estimates of the laser scan of the Wageningen 2017 dataset and the FPGA SGM stereo algorithm. Fig. 29(top right) shows the the percentage of valid disparity matches within an error band of one, two, three und four pixel disparity difference between the laser scan and the camera disparity data. Fig. 29(bottom) shows the cumulative distribution over the bad1, bad2, bad3, and bad4 errors. For comparison, Fig. 29 also shows results obtained with DispNet and SDF-MAN.

As can be seen, the error in disparity is rather small, even though the FPGA stereo algorithm evaluates only a small, discrete set of disparity values. As a result, the FPGA stereo algorithm is able to provide coarse obstacle measurements. Again, notice that the FPGA stereo algorithm provides depth estimates without inducing any load on the CPU and GPU of the TrimBot. This is in contrast to, e.g., DispNet, which makes heavy use of the robot's GPU.

**Qualitative evaluation.** Considering the garden scene shown in Fig. 6 and Fig. 7, Fig. 30a shows a close up of the point cloud data at a tree trunk. The disparity data matches with the Velodyne lidar distance measurement. Due to the discrete disparity steps, the accuracy is limited to $\pm$ one disparity value. The tree trunk has a disparity value of 16. The distance uncertainty

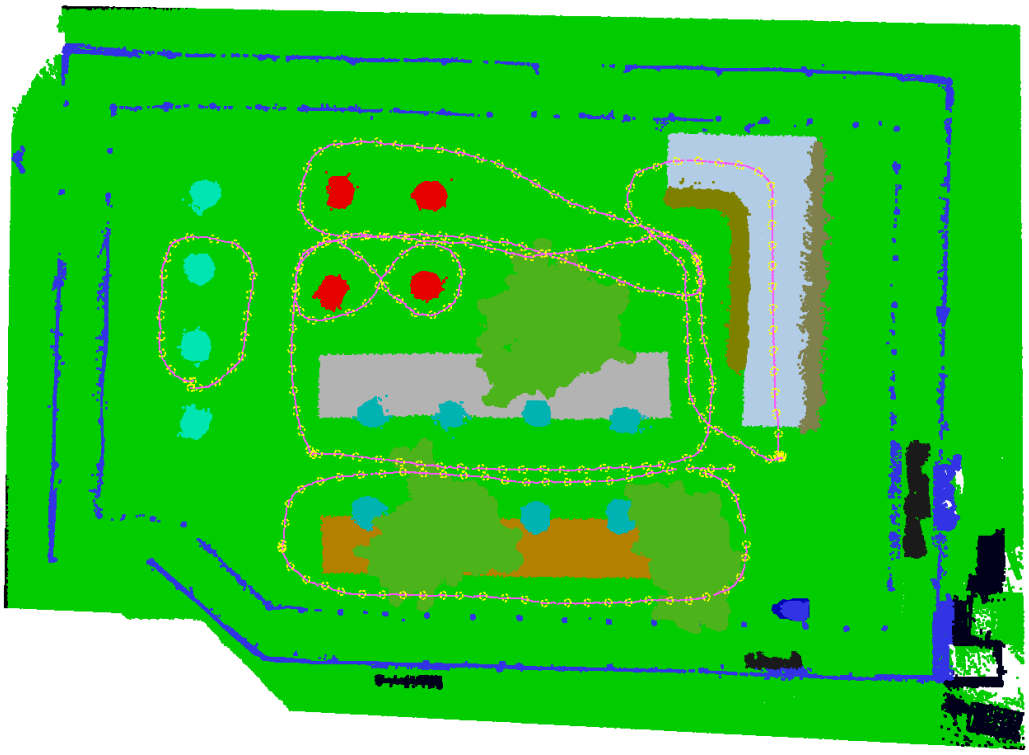
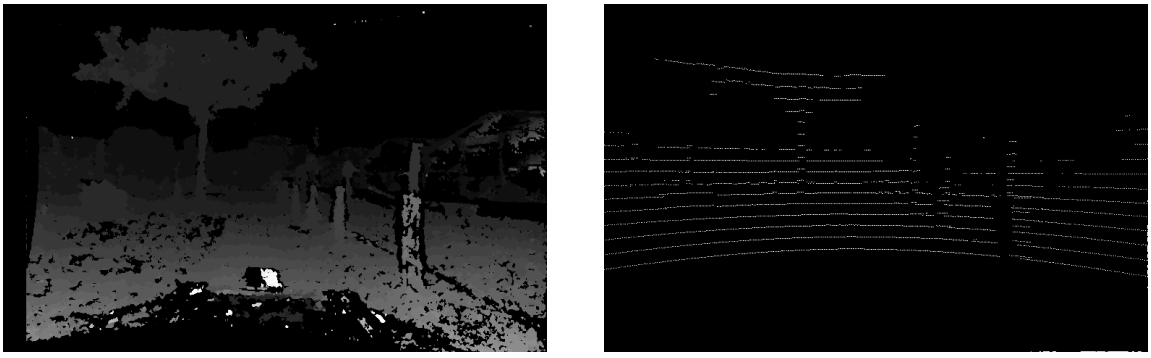


Figure 27: Wageningen Garden 2017 (WUR2) GT dataset with semantic point cloud, trajectories (magenta line) and camera centers (yellow).



(a) (b)

Figure 28: Disparity image of the front stereo camera pair, estimated by the FPGA SGM stereo algorithm, in 28a and the Velodyne 3D points projected in the same camera view in 28b. There are about 3000 Velodyne measurements present in the corresponding camera view.

between the disparity values of 15 and 17 is 0.12 meter (cf. Tab. 1). The Velodyne lidar system allows for a much finer distance measurement with less than 0.03 meter accuracy.

The upper part of the tree trunk is slightly bent towards the robot system. This feature is not visible in the camera disparity values as the difference is within the uncertainty of the disparity values. However the Velodyne lidar data clearly shows the shape of the tree trunk. With a bigger search range beyond 32 disparity values, including sub-pixel accuracy for finer distance resolution, also smaller details of the scene can be captured in the disparity data. However, the extended search range would need additional computing resources to provide real-time processing. This would mean a more powerful FPGA with a higher energy consumption would be needed. As the existing FPGA stereo algorithm already provides reasonably good depth maps, we decided to keep using the existing FPGA board.

Fig. 30b shows the scene with the tree trunk from the perspective of the left camera. The Velodyne lidar measurements are shown as an overlay of orange dots. While the Velodyne sensor is most accurate in distance, the vertical resolution is limited to 16 channels with a vertical field of view of only 30 degrees. The cameras allow for a vertical field of view of 52 degrees with 480 pixel resolution. The ten camera system in the pentagon shape and the Velodyne sensor produce a full horizontal field of view. The horizontal resolution of a single camera is 752 pixels and in a pentagon shape the full horizontal field of view has a resolution of $5 * 752 = 3760$ pixels. With a uniform distribution this corresponds to a horizontal angular resolution of $360/3760 = 0.096$ degree. The Velodyne Lidar sensor has a horizontal angular resolution of 0.1 degree in best case.

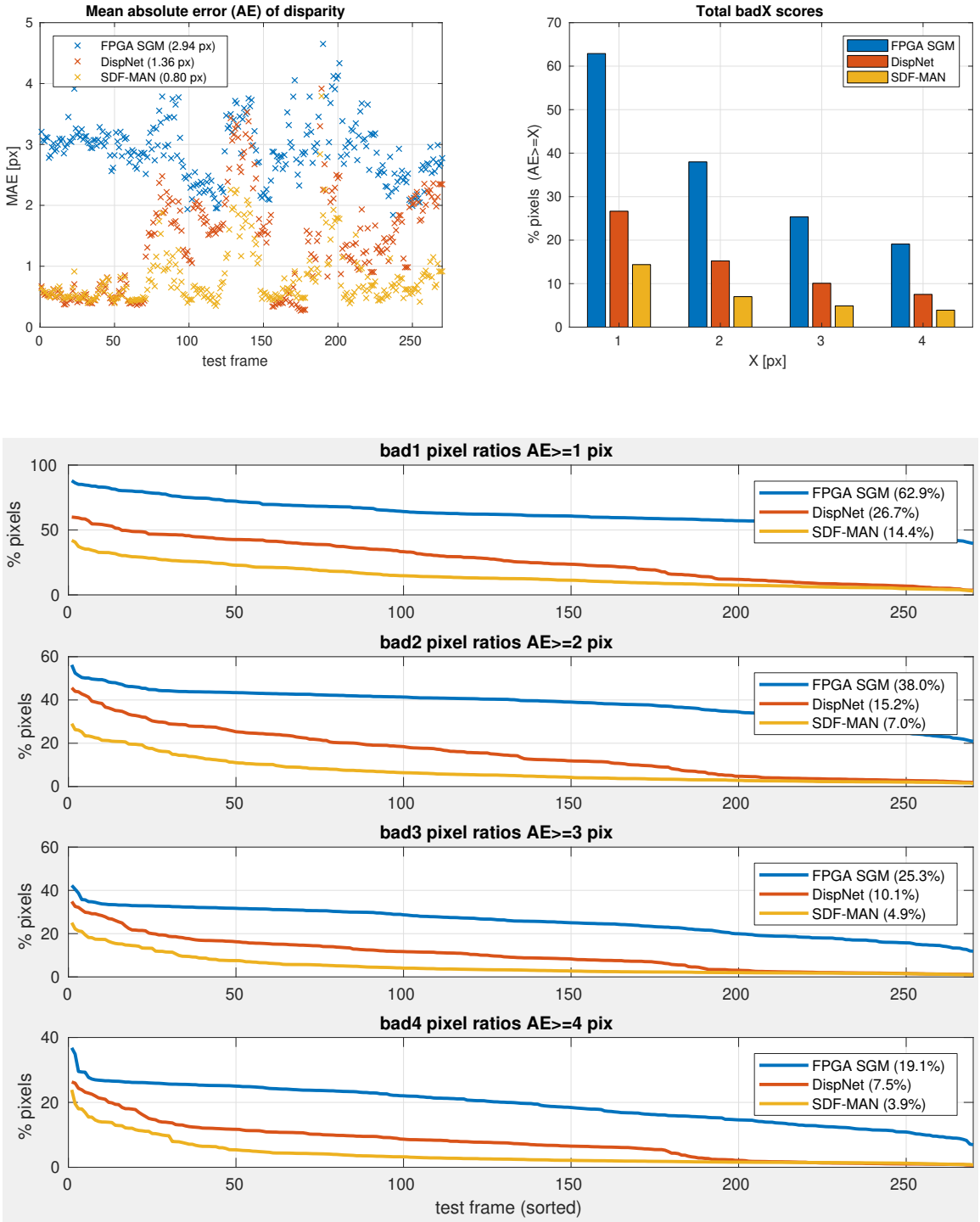
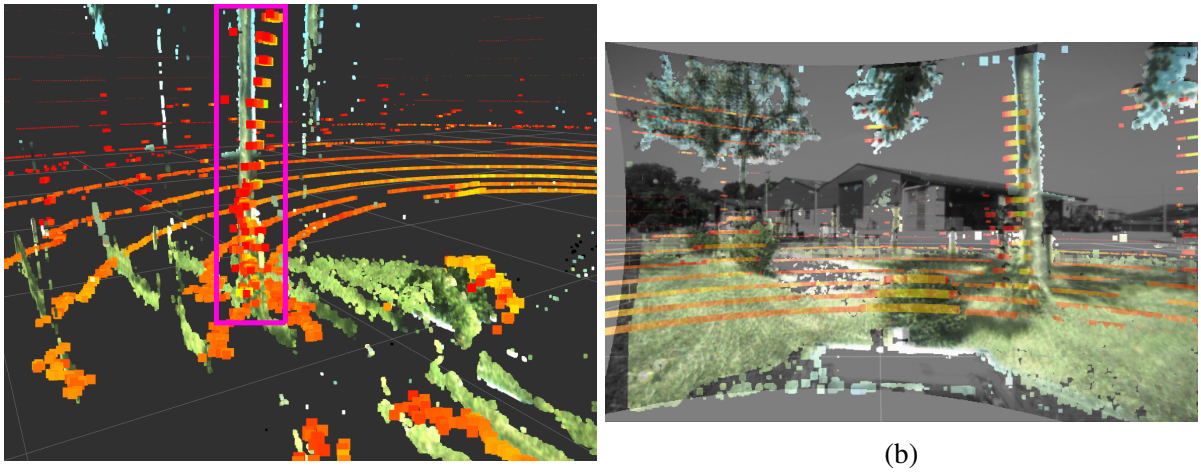



Figure 29: Comparison of absolute disparity errors (AE) of two inputs (FPGA SGM, DispNet) and the fused result (SDF-MAN) on the real garden test set. Numbers in brackets are overall (mean) values over all frames. BadX pixel ratio plots show the distribution of per-frame results (sorted independently for each method) with difficult instances to the left and easy ones to the right (lower error ratio is better).



(a) (b)

Figure 30: Qualitative evaluation of the FPGA stereo algorithm: Side view of a point cloud based on the disparity values of camera pair 0/1 and the Velodyne 3D points. The overall scale factor of the camera 3D data can be verified with the Velodyne measurements that present a similar distance of the tree trunk as visible in 30a (box). A view from the left camera perspective with an overlay of the Velodyne points is shown in 30b.

### 5.2.2 3D from DispNet

We evaluated the DispNet in 3 forms: the originally published, unchanged version; a version finetuned using the original training data but adapted using knowledge of the robot's imaging system; and a heavily retrained version which uses the small amount of available garden groundtruth data to specialize to this specific setting. Fig. 31 compares the outputs of the 3 versions. The original version is clearly inferior, but which of the finetuned versions should be prefered depends on how much generalization ability to non-plant objects the network is required to keep.

**DispNet 'vanilla'.** The DispNet in its currently best-performing published form [57] is trained on *FlyingThings3D*, an abstract, task-agnostic synthetic dataset [43]. It offers one advantage: its generic nature means that this network can be directly applied to any change in the robot's configuration, without a need for adjustment or retraining. However, it is also the worst performer.

**Finetuning without adjusted synthetic data.** In [56] it was demonstrated that a network can (to some extend) be adjusted to a specific setting by reasoning about the data characteristics of the imaging setup. We applied this to the robot base cameras by creating a new version of the *FlyingThings3D* dataset [43] with narrower stereo baseline: the stereo setup on the robot leads to the expected disparities being significantly smaller than those in the original dataset, and neural networks generally perform best when they are trained to the same data distributions as they are expected to encounter in the test setting [58]. The 'left eye color, right eye grayscale' setting is also a unique characteristic which is easily applied to the training data.

While this network does not show any quantitative improvement (see Fig. 33 (a) vs Fig. 34 (a)), Fig. 32 shows that its qualitative results are significantly better. It is likely that the test data (projected laserscan) is not accurate enough to capture relatively small improvements,

especially at object boundaries which are clearly more accurate in the finetuned network's outputs.

**Finetuning with specialized data from the garden.** Generating (pseudo)-groundtruth to train machine learning methods for real-world depth tasks is notoriously difficult and expensive [55, 59], it is prone to errors, and it generally yields small quantities of data; but on the other hand, it is particularly valuable as it can entirely avoid differences between the training and testing settings. Since an accurate 3D garden model was needed for general evaluation purposes in this project, highly accurate and dense laser scans of the entire garden were generated—data that we can also use for network training. For this purpose, the garden pointcloud is reprojected into the camera views of the robot, at laser-tracked positions (an example is shown in Fig. 32, bottom left). The resulting depth maps are not perfectly accurate and do not view the garden at the same time as the visible-light cameras, but Fig. 31 (bottom row) shows that the DispNet can use it well to focus on the 'garden' setting.

However, the constrained nature of this data (and its small number of data points) leads to overfitting: Fig. 36 shows that the finetuned network 'unlearns' to do general disparity estimation after only having seen 'garden' data for a training period, and in Fig. 31 the network outputs show no sign of objects outside of the garden. The network that has never seen garden data does not exhibit this forgetting behaviour. Whether not being unable to measure the depth of objects that are not bushes, trees, or fenceposts is acceptable remains to be seen. It is also possible to mix training datasets: a weighted combination of original synthetic data and garden-specific laser scan data could offer a favourable tradeoff between generalizability and overspecialization.

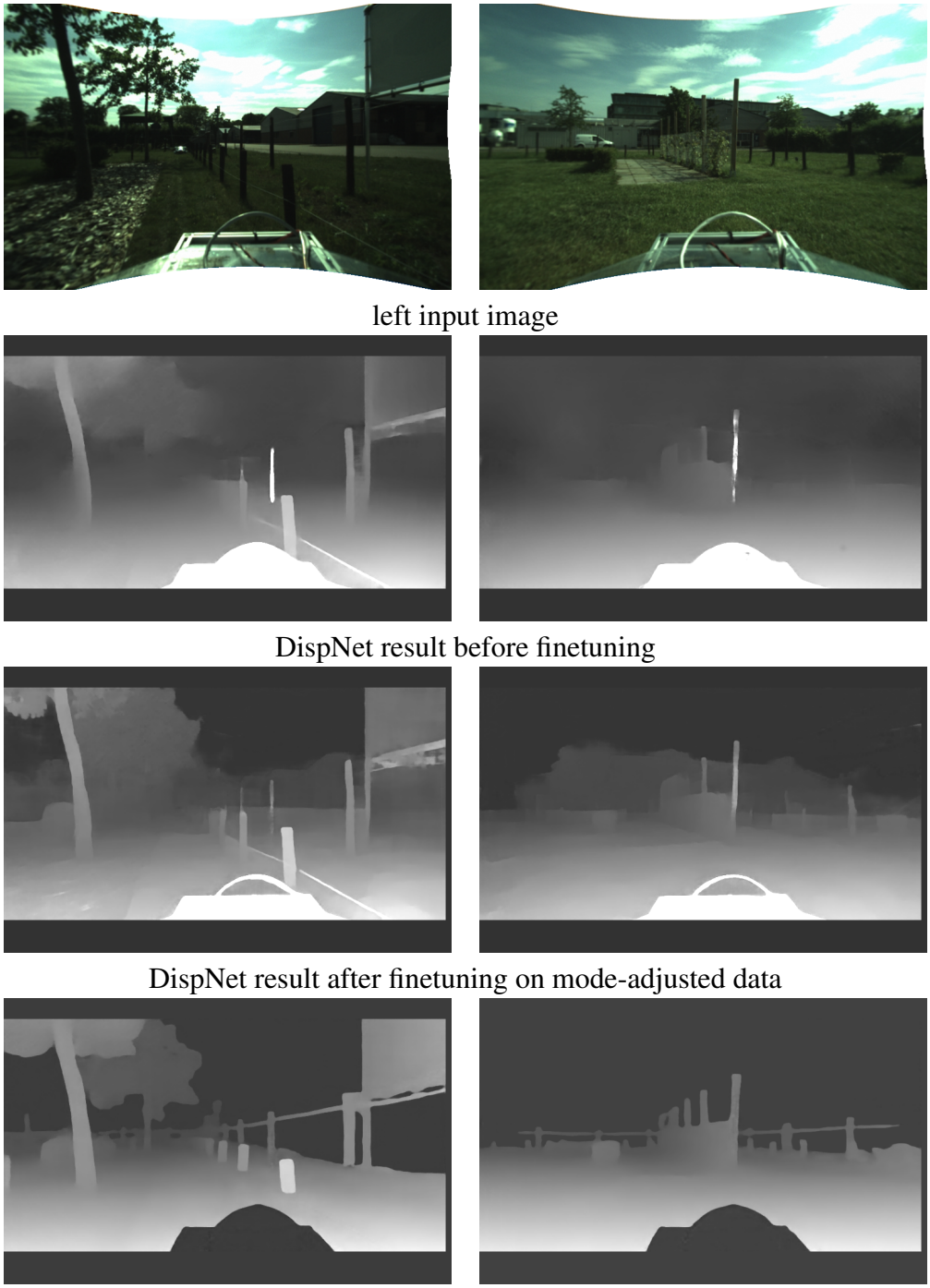
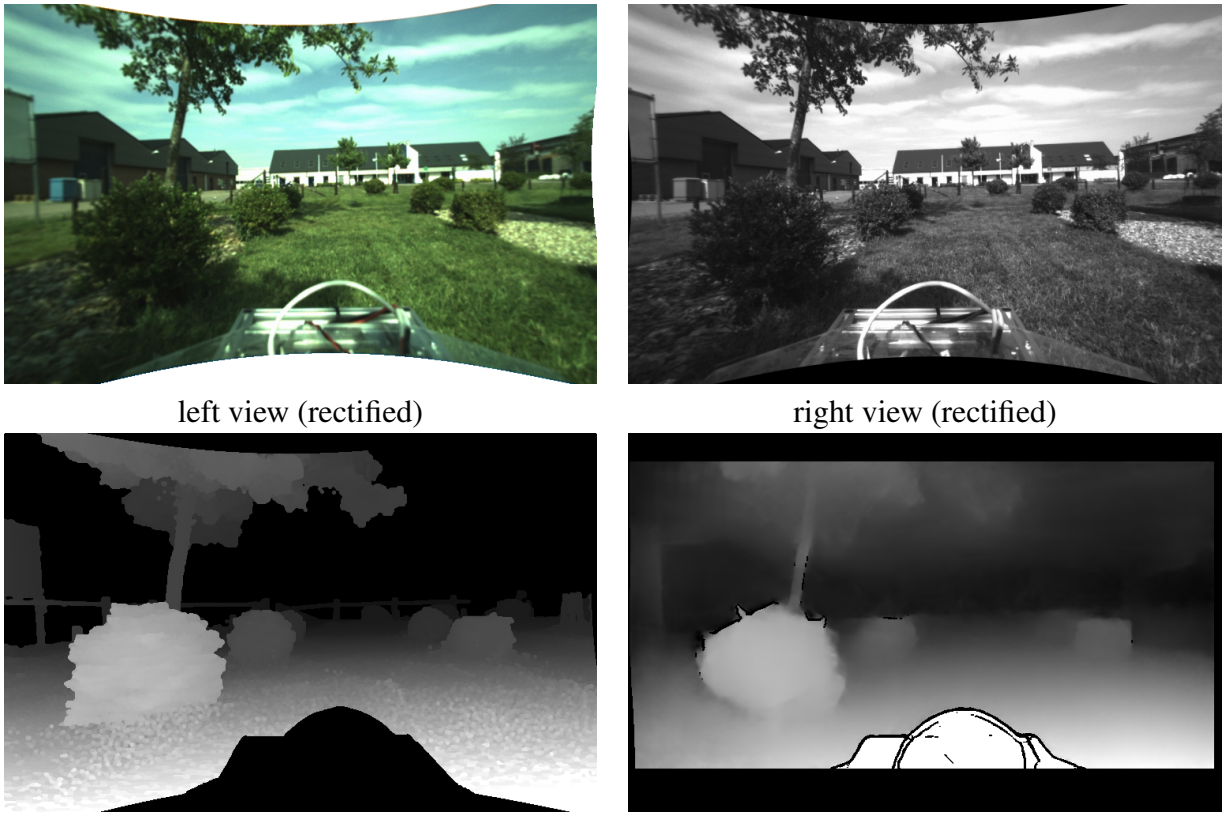



Figure 31: DispNet finetuning: [56] demonstrated that network finetuning can be done on a low data level, by reasoning about the characteristics of the imaging system (and without access to scenario-specific groundtruth). We applied this and finetuned a DispNet using FlyingThings3D data [57] re-rendered with a narrower stereo baseline, and with the TrimBot base camera RGB-Grayscale camera combination. The finetuned network produce fewer false matches and manages to capture object contours at greater distances. Note that the finetuning procedure uses a small dataset with little variance which leads to a loss in generalization ability (see Fig. 36).



left view (rectified) right view (rectified)

offline laser scan DispNet estimate (no finetuning)

Figure 32: *top row:* input stereo view from one of the robot base camera pairs; *bottom row:* pseudo-groundtruth and DispNet prediction of the stereo pair's disparity map. The groundtruth map excludes the sky and the robot itself. The DispNet result excludes invalid pixels from undistortion/rectification.

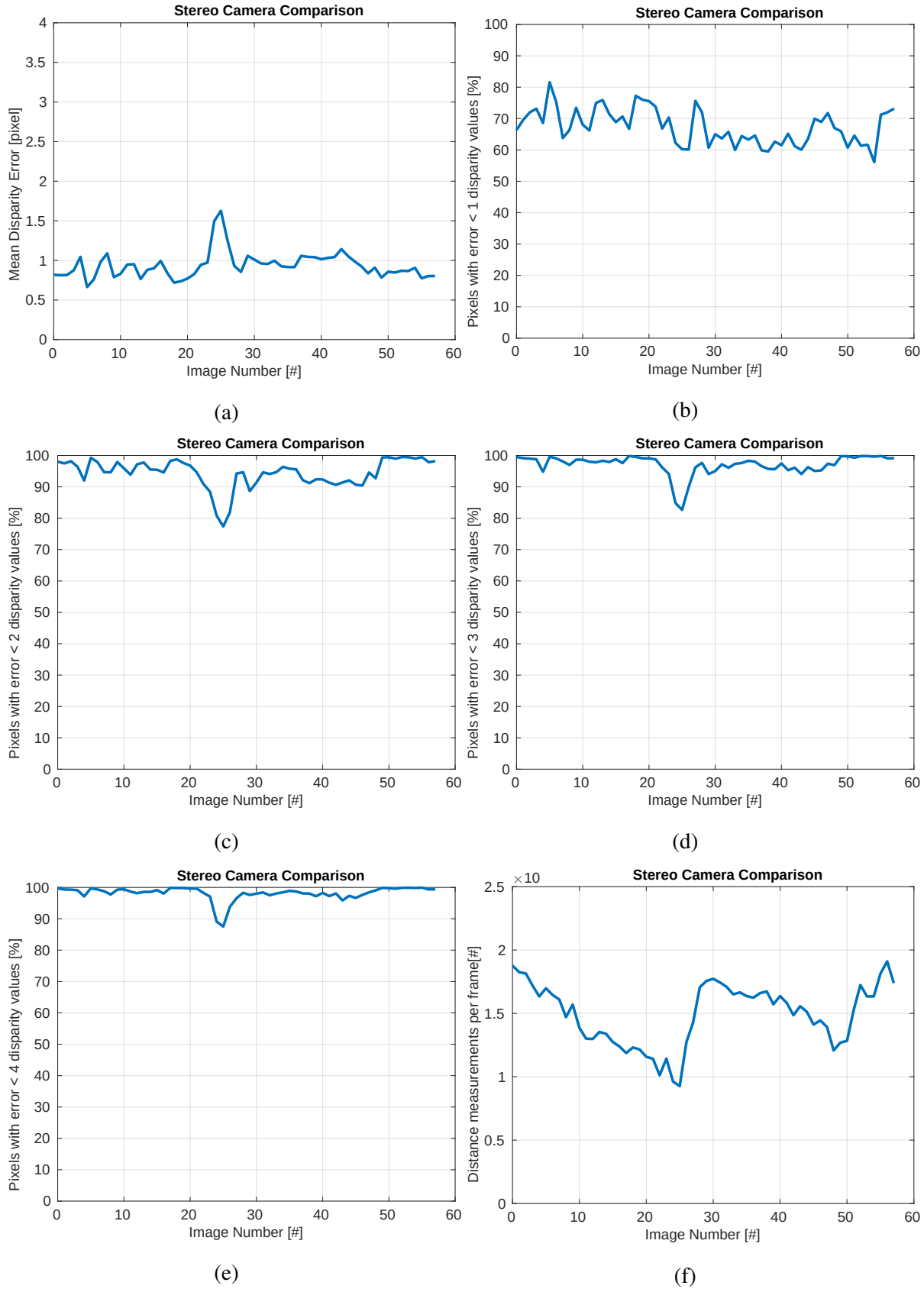



Figure 33: Quantitative evaluation of the DispNet stereo network in its 2018 version [57] on the Wageningen 2017 dataset. The subplots show the percentage of valid disparity matches within an error band of {1,2,3,4} pixel disparity difference compard to the laser scan.

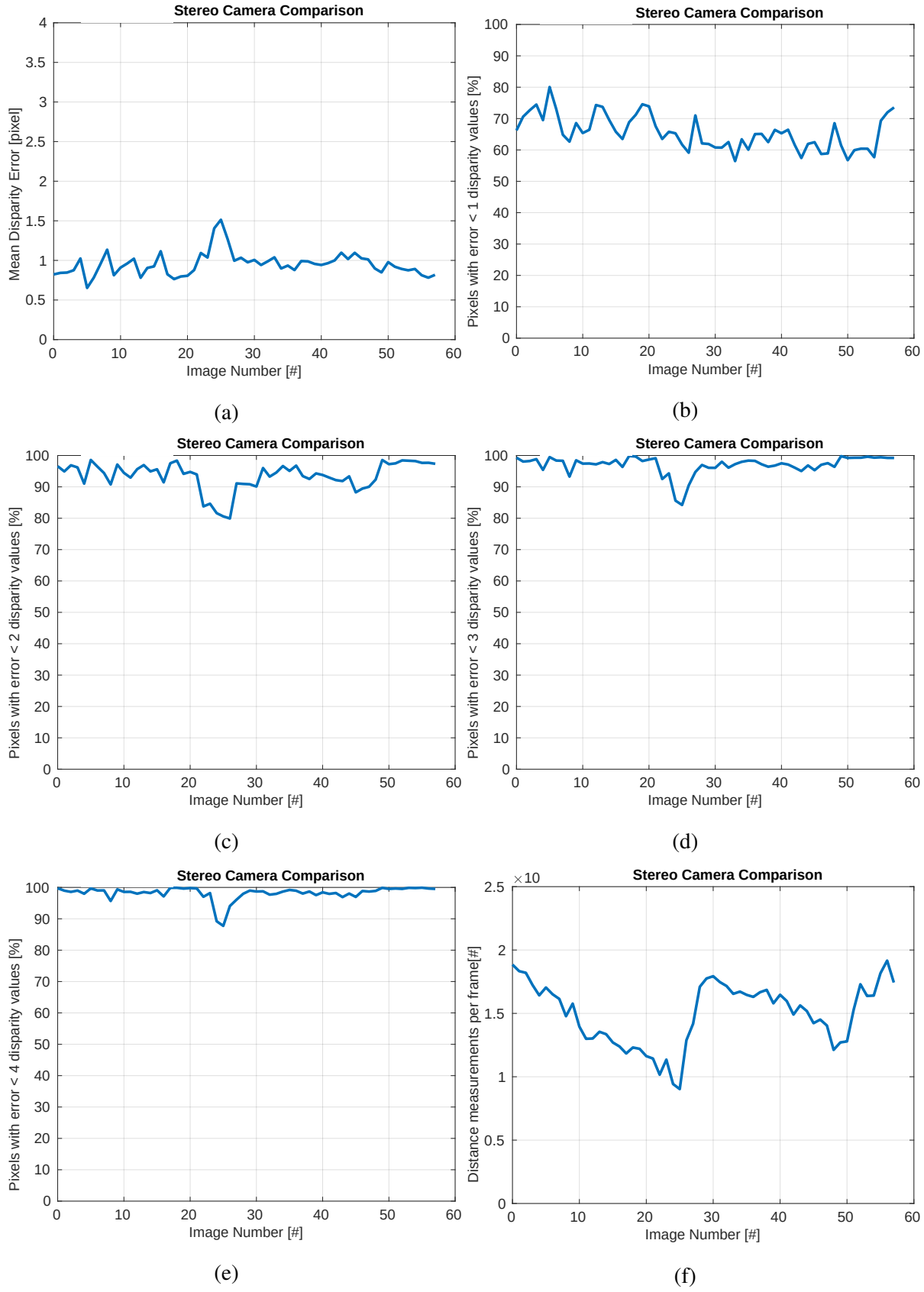
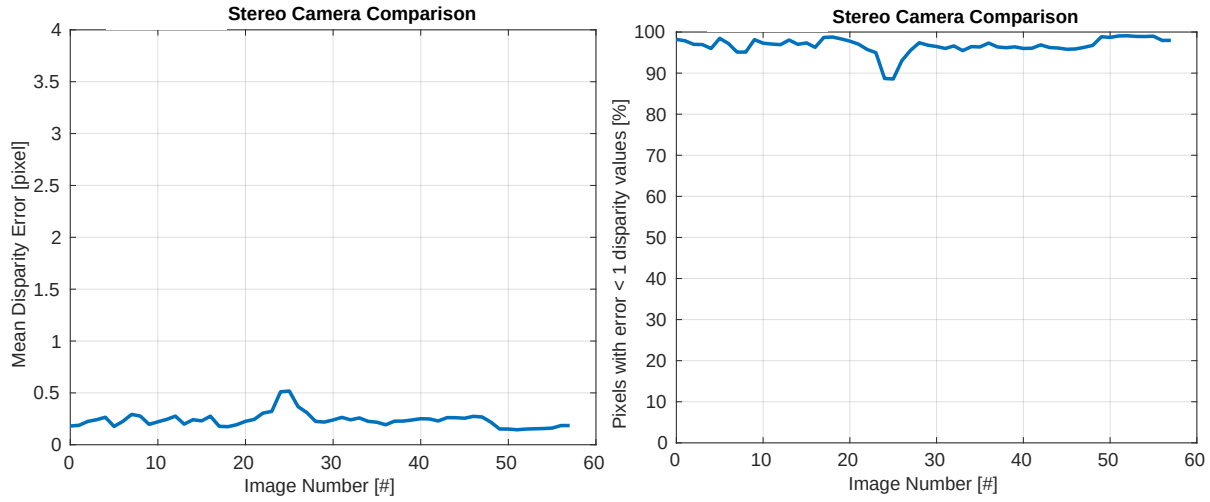



Figure 34: Quantitative evaluation of the DispNet as in Fig. 33, after finetuning on low-level adjusted data. Despite the qualitative improvement shown in Fig. 31, this pseudo-groundtruth test data cannot capture the relatively small changes, especially at larger distances.

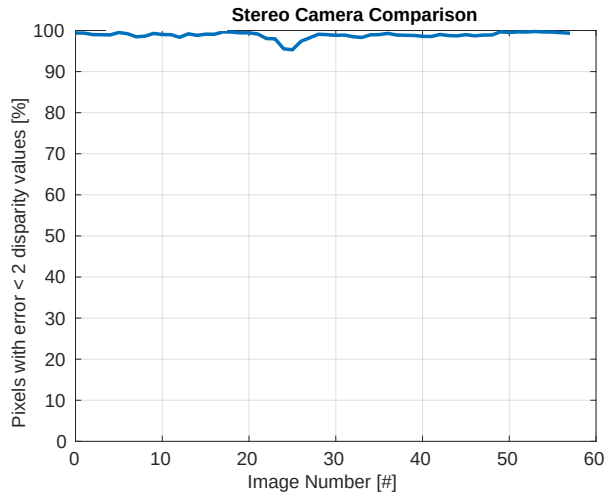
 



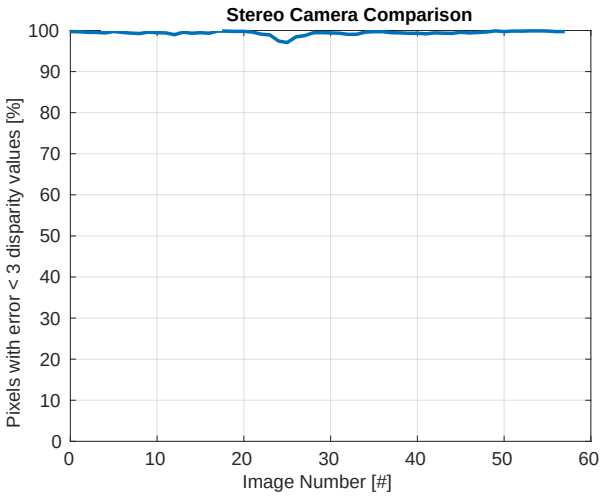

(a)



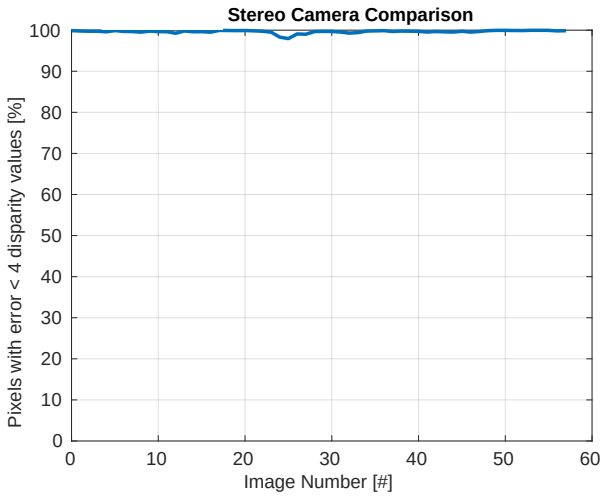

(b)



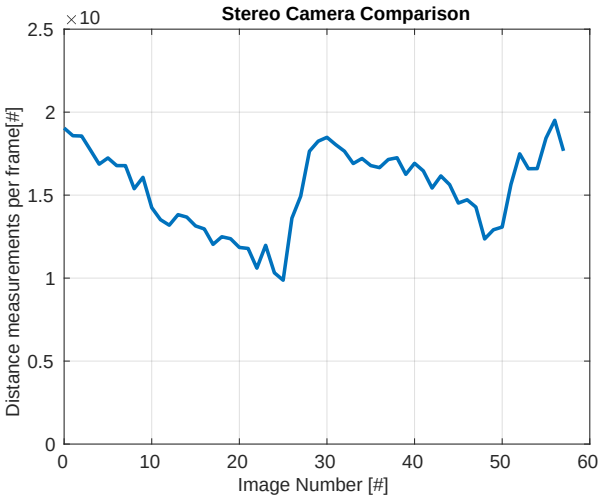

(c)




(d)




(e)




(f)

Figure 35: Quantitative evaluation of the DispNet stereo network as in Fig. 34, with the difference that the network has been finetuned using data of the same mode as the test data. This represents the best-case training scenario in terms of accuracy, but the network also strongly overfits to the 'garden objects' setting as demonstrated in Fig. 36.

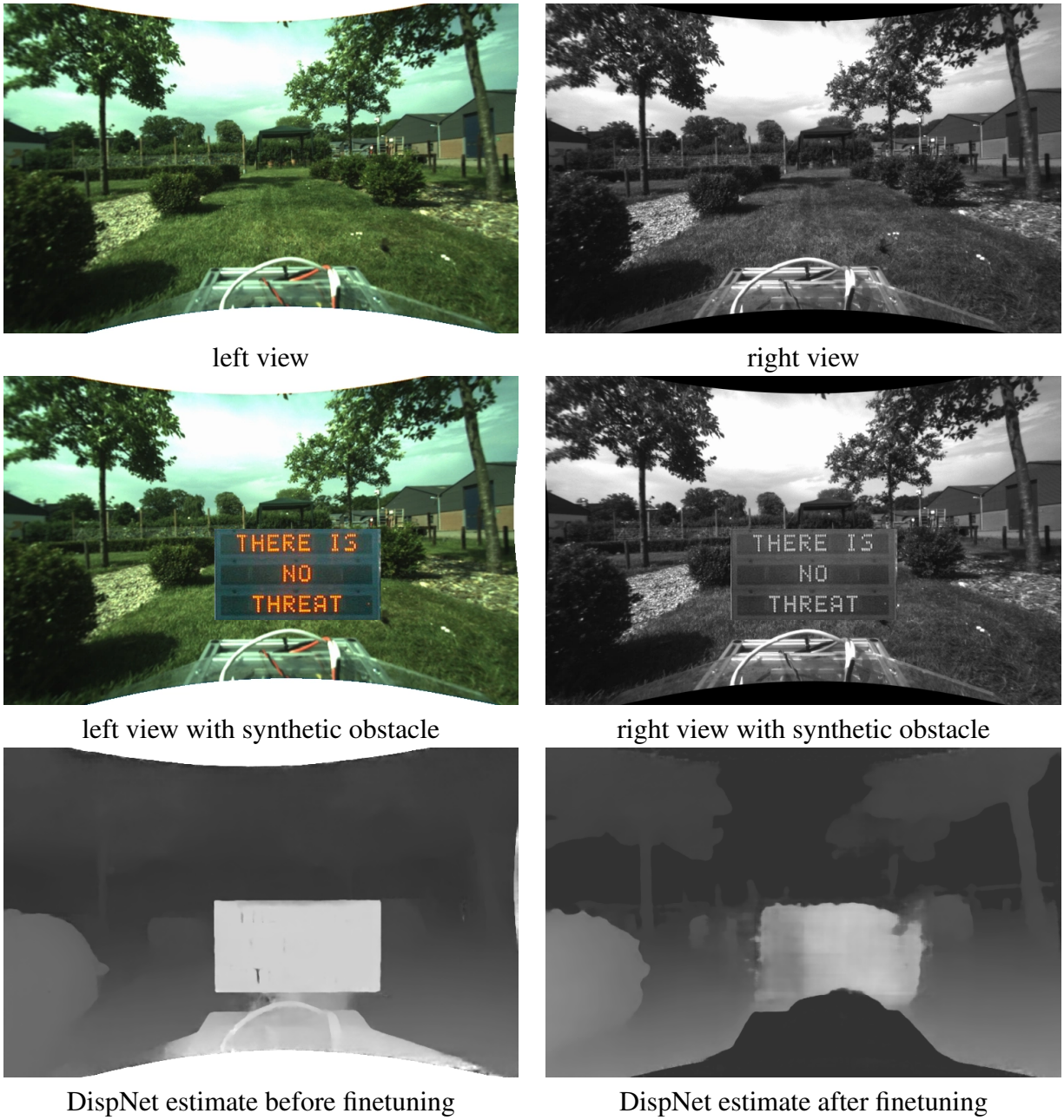



Figure 36: Overfitting: As demonstrated in Fig. 32, the DispNet easily accepts the 'garden' training data and learns it well. However, in the process the network unlearns how to deal with 'non-garden' objects. **Middle row:** Test sample with added 'foreign' object. **Bottom row:** The general-purpose DispNet faithfully reproduces the sign's edges. After finetuning to the garden setting, the network can only produce bush-style object contours. This behaviour is acceptable only in tightly controlled environments. (The change in appearance of the robot platform in the disparity images is a coincidental side effect of the training data.)

### 5.2.3 3D from (Trinocular) Stereo

As we do not plan to integrate the 3D plane stereo matching algorithms from Section 3.3 on our robot platform due to long computation time, we have performed only early qualitative comparison on garden data (without GT).

Fig. 37 shows disparity maps for two views. The floating point implementation of 3D plane stereo gives naturally smoother results than discrete FPGA SGM stereo and also produces less noise and outliers. It however does not cope fully with the repetitive structures in the background (left column), e.g. notice bright green and blue patches in the upper part of the disparity maps (c,d). The boundaries of the objects are sometimes not clear, e.g. the pole in the right image, likely due to the lack of salient texture in the corresponding image areas.

Evaluation on standard datasets can be found in [44, 45], where the two approaches ranked among the top performing results in the Middlebury and KITTI 2012/2015 benchmarks.

## 5.3 3D Data Fusion

This section will evaluate the performance of three steps in the fusion pipeline (Sec. 4 - instant, temporal and global fusion) on the Wageningen garden 2017 dataset. The test part consists of several trajectory segments, as shown in Fig. 38.

### 5.3.1 Instant Fusion

The SDF-MAN fusion of the two input methods (FPGA SGM stereo and DispNet - Sections 3.1 and 3.2) was fine-tuned on 1000 training images from Wageningen garden 2017 dataset. The GT disparity comes from a static off-line scan of the garden using a laser scanner, which does not correspond exactly to the actual images of moving dynamic objects taken at a later time during actual drive of the platform through the garden. Along with the limited accuracy of GT camera poses used for the point cloud projection, this resulted in invalid GT disparities near object boundaries.

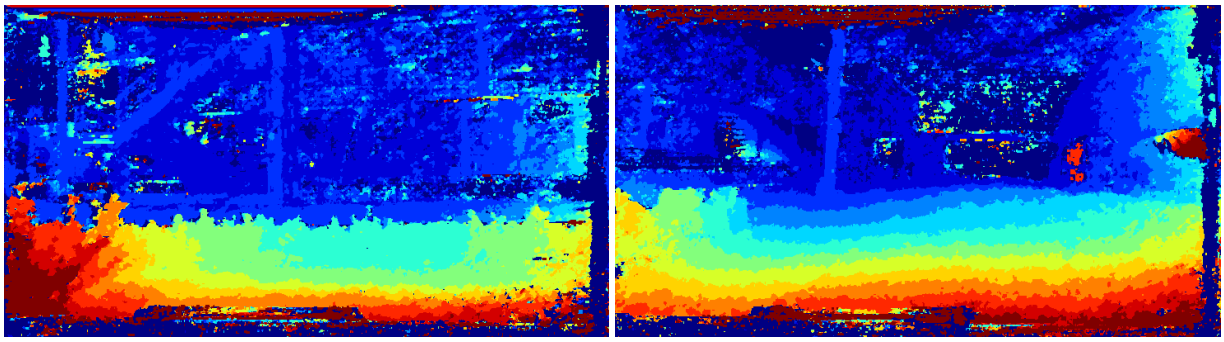
Two test cases comparing the inputs and output can be seen in Figures 39 and 40 on the test set. The SDF-MAN [46] algorithm provides overall smoother results and has learned small corrections of DispNet, which are visible in the far range (background). In Fig. 40 it is also able to reasonably extrapolate disparities outside of the undistorted visible area near borders (tree crown and close ground). The extent of how the coarse FPGA SGM input (a) actually contributes to the fusion was not systematically investigated, but in this example it seems to help to rectify the uneven (bent) ground surface estimation of DispNet (b), resulting in flat ground in the fused output (c).

The mentioned inaccuracy of GT object boundaries impacted the evaluation (errors indicated there can be falsely reported) but also the fine-tuning of networks. In the case of SDF-MAN, we observe excessive smoothing of discontinuities, even after pixels with steep disparity gradients were removed from the output maps during post-processing. Smoothing near boundaries reduces the overall disparity error cost for which the network is optimized, but can negatively impact the subsequent reconstruction steps by adding noise to the free space behind objects (‘curtain’ artefacts). DispNet alone appears to handle disparity discontinuities better, with only one pixel wide gaps at the object boundary or sharper discontinuity steps.

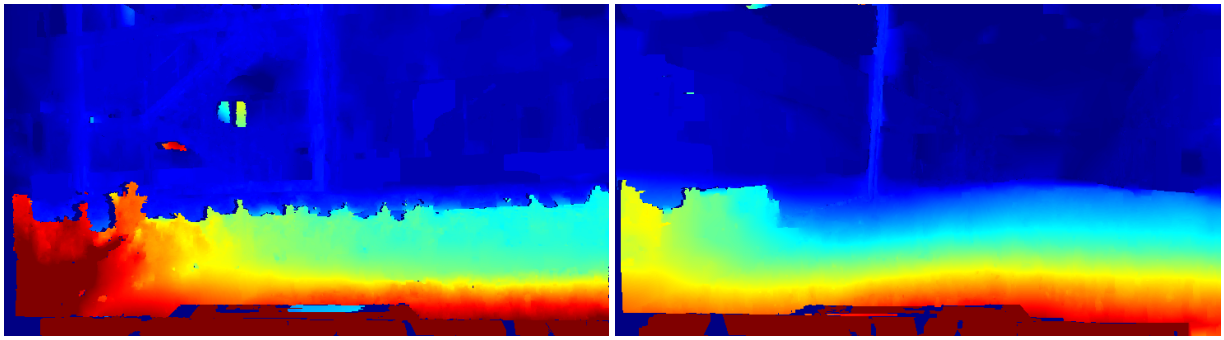
Fig. 29 shows the absolute disparity error of the static pixel subset and the distribution of error magnitudes (badX). The overall statistics suggest that the mean absolute error of the two

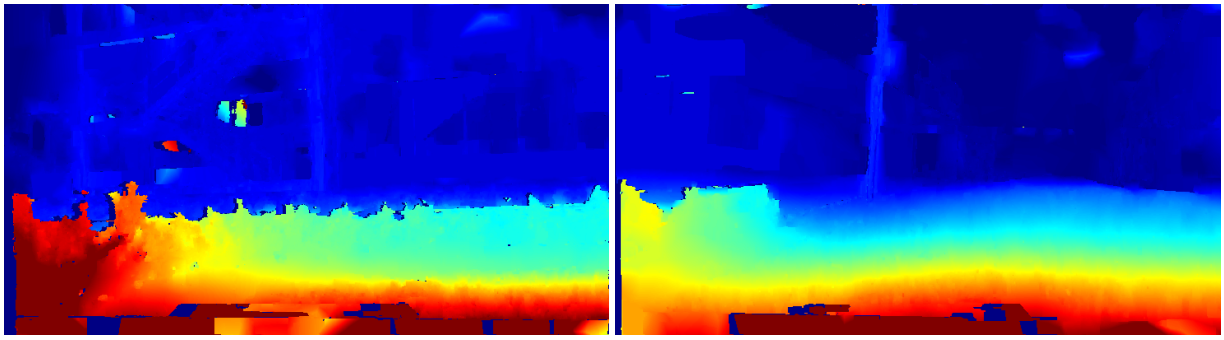
(a) Left rectified images



(b) FPGA SGM disparity



(c) Binocular stereo disparity



(d) Trinocular stereo disparity

Figure 37: Binocular [44] and trinocular stereo [45] compared to FPGA SGM [42] on two test images pairs from Renningen garden. Depth is shown in pseudo-color: blue corresponds to far objects (small disparities), red to near objects (large disparities).

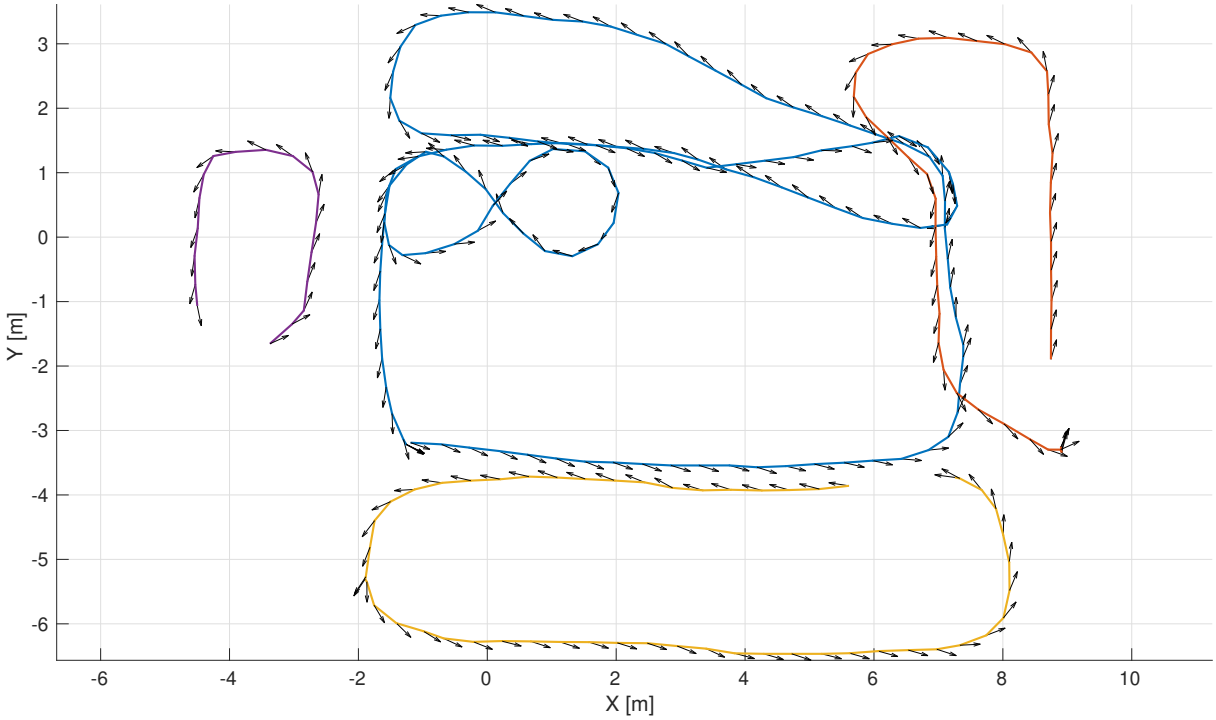



Figure 38: Test trajectories of the robot used to capture data for the Wageningen garden 2017 dataset. Arrows indicate the camera orientation. For testing 250 out of total 1200 poses was selected (see Figure 27).

inputs, FPGA SGM with 2.94 px and DispNet 1.36 px, was successfully reduced by SDF-MAN to 0.8 px. The same proportion of the performance gain is seen in total badX scores (ratios of pixels with absolute error $\geq X$ px) for all $X = 1, \ldots, 4$. The distribution of badX per-frame results shown in four plots in Fig. 29 bottom suggest that the fusion helps particularly with difficult instances, while the easier ones were already solved by DispNet fully (to the GT accuracy limit).

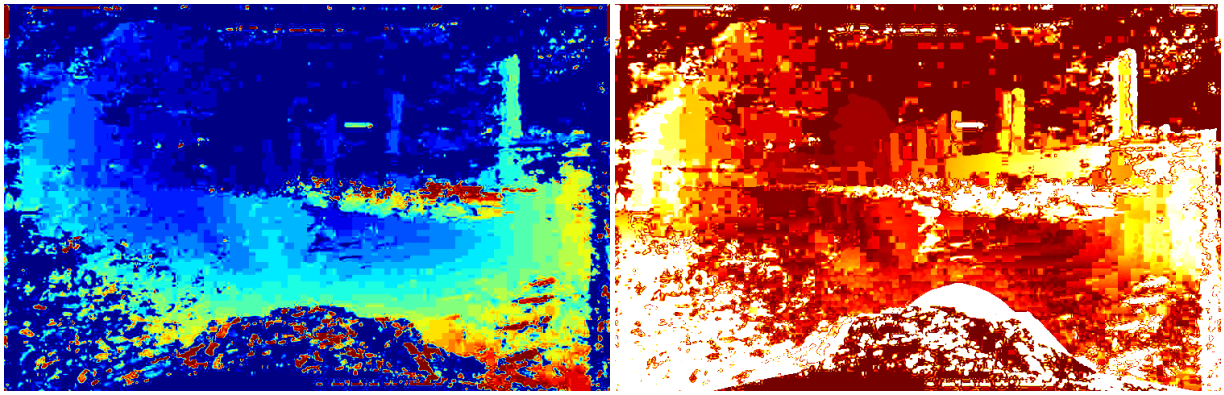
Detailed analysis of the mean absolute error of individual image frames revealed that the performance was improved by fusion in 60% of all cases, 30% of all cases saw comparably similar performance ($\pm 0.2$ px) and in 10% of all cases the fusion impaired the DispNet result. The performance decrease can be attributed to instances where the two inputs were contradictory rather than complementary, ie. violating the assumption for successful fusion.

### 5.3.2 Temporal Fusion

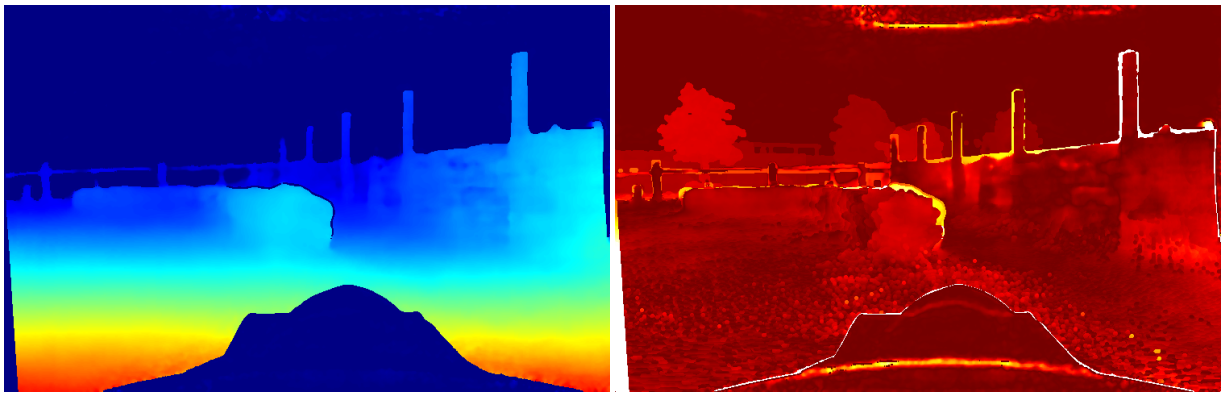
The goal of relative pose estimation with DUGMA is to provide refined poses for alignment of point clouds prior to volumetric reconstruction. The performance was evaluated on four trajectories as shown in Figure 38. The average distance between two consecutive poses is around 0.5 m and rotation up to 45 degrees. The poses in the dataset correspond to every 10th originally captured image frame.

Registration was computed independently for all pairs of consecutive camera poses, based on the point clouds generated by backprojection of fused SDF-MAN disparities. The older point cloud $t$ in the pair was considered fixed and the 6 DOF relative pose of the other (moving) point

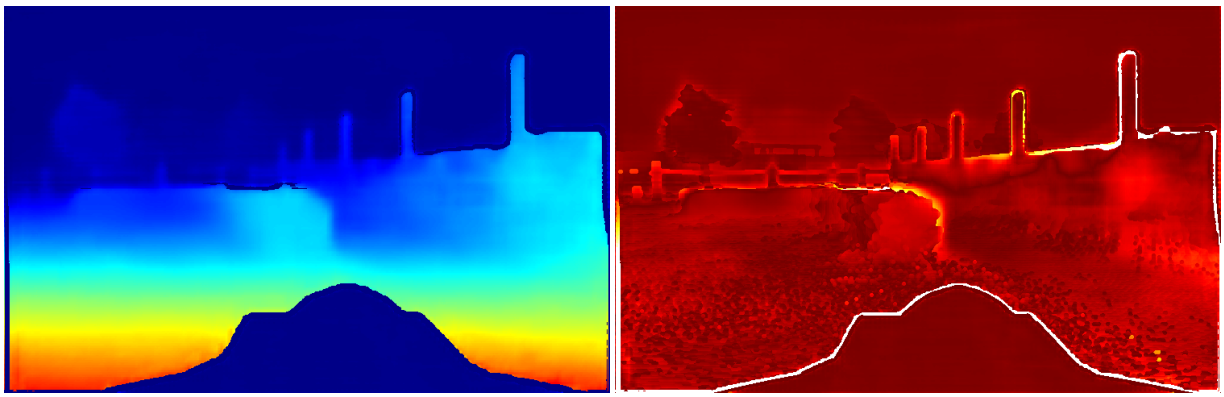
 



(a) FPGA SGM



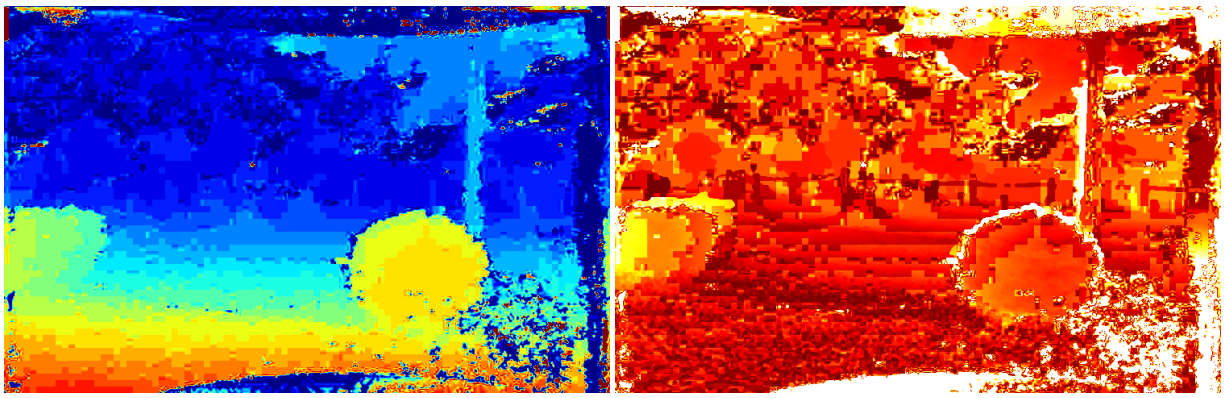
(b) DispNet



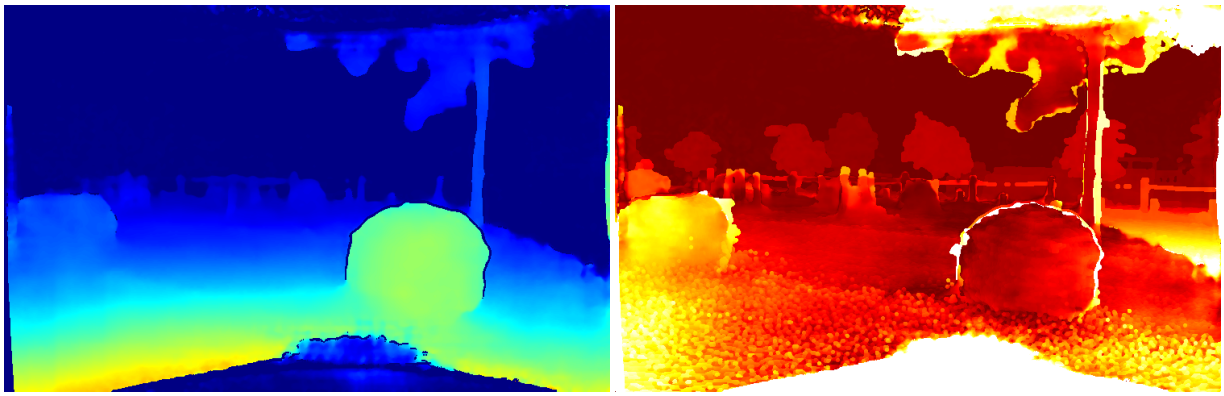
(c) SDF-MAN

Figure 39: Disparity maps and errors of two inputs (FPGA SGM, DispNet) and the fused result (SDF-MAN) from the front-view stereo pair. *Left:* color-coded disparity (cold = far, warm = near). *Right:* color-coded accuracy (dark = small error, bright = large error, white = 5+ pixels error). The GT provided by static laser point cloud can be inaccurate near dynamic object boundaries.

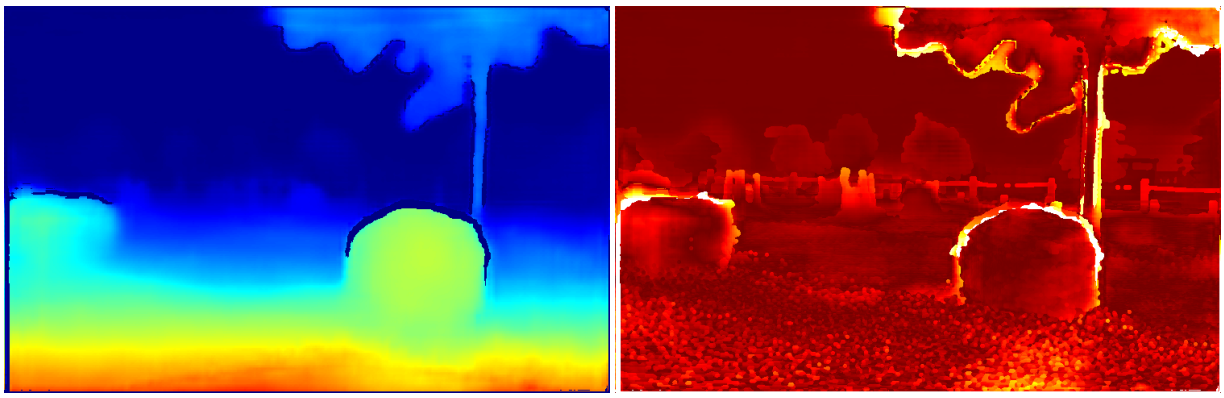
 



(a) FPGA SGM



(b) DispNet



(c) SDF-MAN

Figure 40: Disparity maps and errors of two inputs (FPGA SGM, DispNet) and the fused result (SDF-MAN) from the side-view stereo pair. *Left:* color-coded disparity (cold = far, warm = near). *Right:* color-coded accuracy (dark = small error, bright = large error, white = 5+ pixels error). The GT provided by static laser point cloud can be inaccurate near dynamic object boundaries.

cloud $t+1$ was estimated by DUGMA. Both point clouds were subsampled to 5000 points based on normal vector segmentation of the point cloud: only 1000 points were uniformly sampled from the horizontal ground segment to limit its dominance in the set, and the remaining 4000 points were sampled from vertical objects. Objects are salient features of the point cloud and allow to establish the right correspondences between the two point clouds to estimate Z axis rotation (yaw), unlike the planar ground part that only stabilizes the X-Y plane. The initial estimate was set to identity and the algorithm was run for 50 iterations, with an average run-time of 50s on a GPU.

Some typical cases of registration results are shown in Fig. 41, where the robot is moving to the left with the side-view stereo pair facing down, as indicated by color arrows. The results were better in the case of pure lateral translation as shown in Fig. 41a. The algorithm however typically failed to find rotation when it was present, such as in Fig. 41b, except for instances when large objects were in range. The case of small overlap between the point clouds however led to incorrect matches of wider planar objects (hedges), such as in Fig. 41c. In such cases, the algorithm will get stuck in a local minimum on the noisy surface.

A quantitative evaluation is shown in Fig. 42 for pairs with a rotation difference up to 20 degrees (as for larger rotations, the overlap was apparently too small for a possible match). The mean translation error of 33.2 cm (black line) was high compared to the mean GT motion of 45 cm. The mean rotation error of 6.4 deg with the mean GT rotation of the same magnitude indicates the estimates were incorrect except for a few cases. We attribute this to the 'curtain' artifacts observed in the previous section, that extend beyond the object away from the camera into free space and discourage rotational motion by 'locking' the orientation according to the artifact.

This and other problems (eg. slow convergence) will be investigated further and the sub-sampling strategy will be modified appropriately. Also, the case of more frequent relative pose estimation (ie. every 5th or 2nd frame) will be evaluated, along with the different initializations (e.g., SLAM poses or wheel odometry).

### 5.3.3 Global Fusion

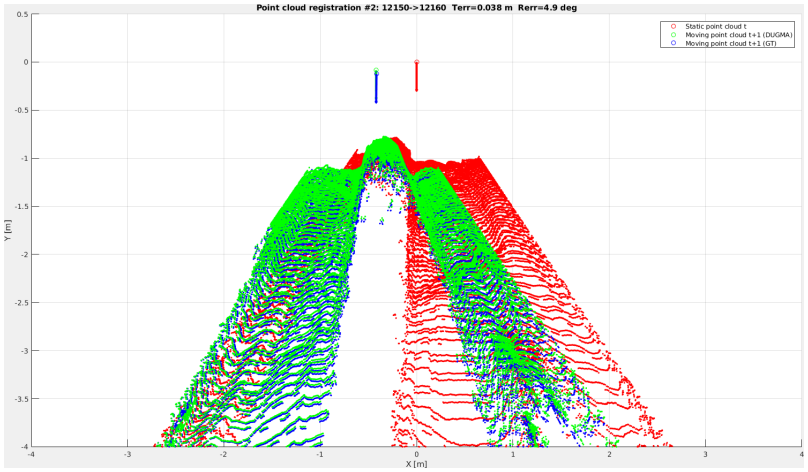
A set of 250 point clouds from SDF-MAN was fused to evaluate the global fusion. From each view, only a subset of points was used: Points further than 10 m from the camera were discarded due to low accuracy. Similarly points in the top 1/3 of the image (moving tree branches or sky) were ignored to focus on reconstruction of the ground surface and bushes, which are of interest for navigation and trimming in this project. Normals were estimated for all points based on local plane fitting (using 8 neighbors) and oriented towards the cameras. All point clouds were then merged into one using the GT pose estimates, as shown in Fig. 43. As can be seen, hedges and topiary bushes can clearly be identified. However, tree trunks and other thin structures are not well represented as most of there corresponding points are noisy and thus were removed by the statistical outlier filter.

The original input point cloud (8.1M points) was sub-sampled to 1 cm inter-point distance (1.3M points), then volumetric point fusion (Sec. 4) was run with the point scale set uniformly to 3 cm. The run-time of the fusion was around 10 minutes using 4 CPU cores. The resulting surface mesh can be seen in Fig. 44, where individual bushes can be recognized. The shape accuracy is around 5-10 cm, depending on the distance from which it was observed by the robot.

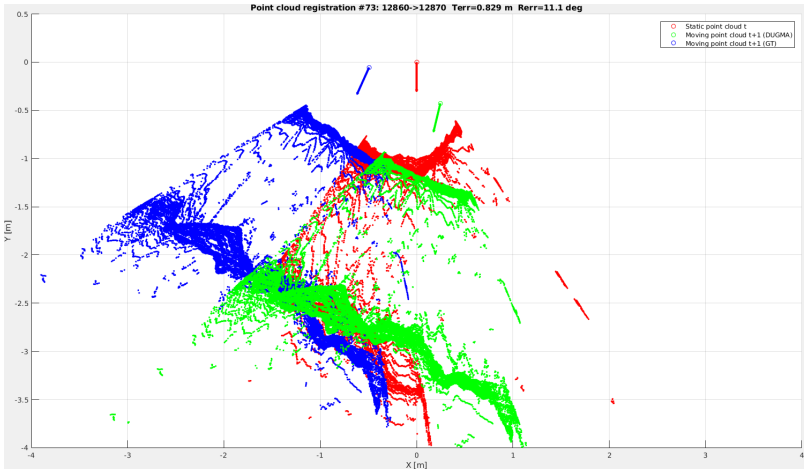
 

Note that only one out of five camera pairs mounted on the robot were utilized in this initial evaluation. In the next steps we will add point clouds from all cameras and evaluate the result according to the expected use-case in trimming of extended bushes.

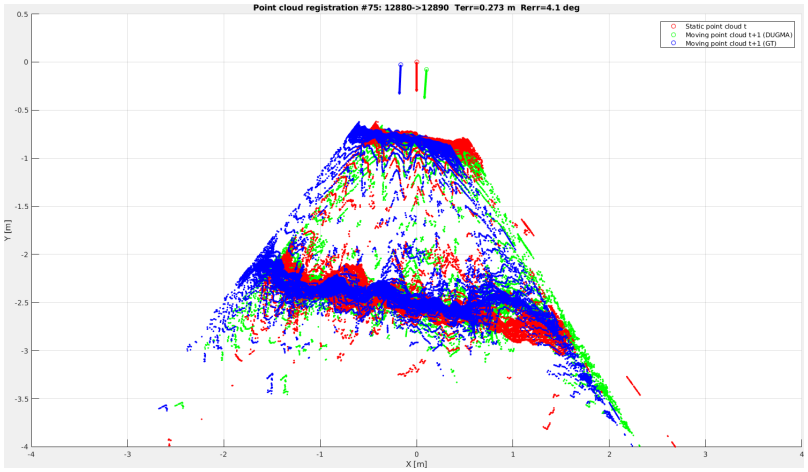
 




(a) Successful registration




(b) Failed translation estimate due to a large rotation (small overlap)




(c) Failed translation estimate in front of planar objects

Figure 41: Point cloud registration with DUGMA using the SDF-MAN output. In each case there is a top view of 3 point clouds with camera poses as arrows above (circle is the optical center, arrow points in the yaw direction). The red point cloud is a static reference frame $t$, the green one is a moving point cloud from $t+1$ with its relative pose estimated by DUGMA. The blue point cloud is the $t+1$ point cloud visualized using the GT relative pose. Overlapping blue and green points indicate a good DUGMA estimate.

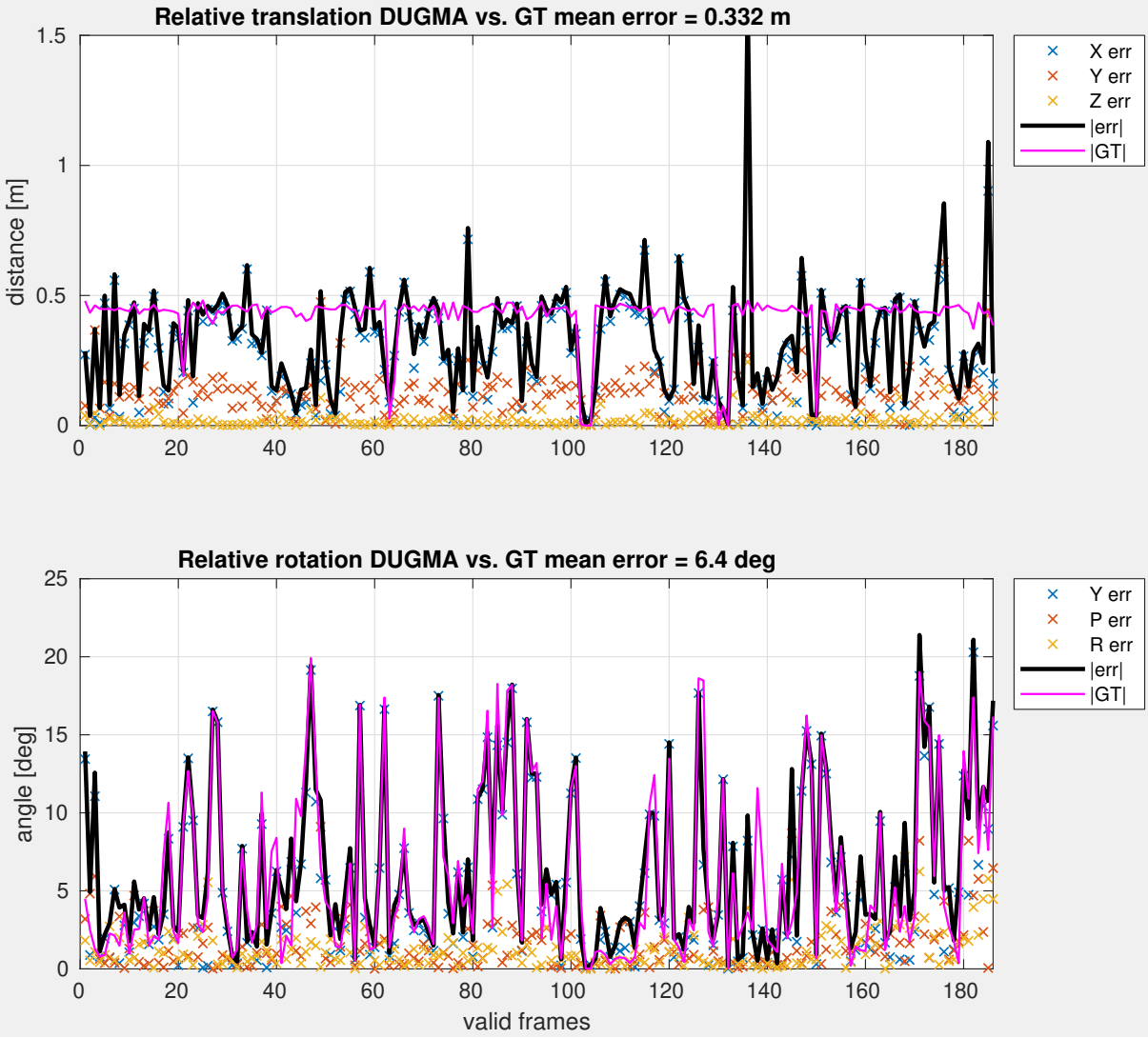
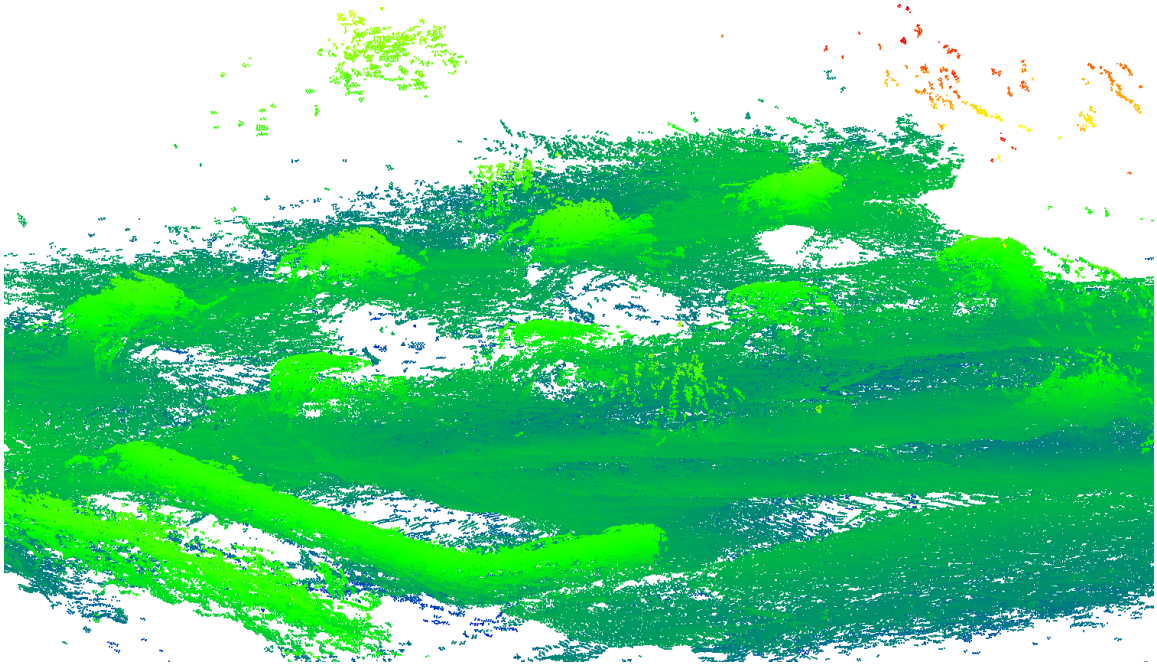
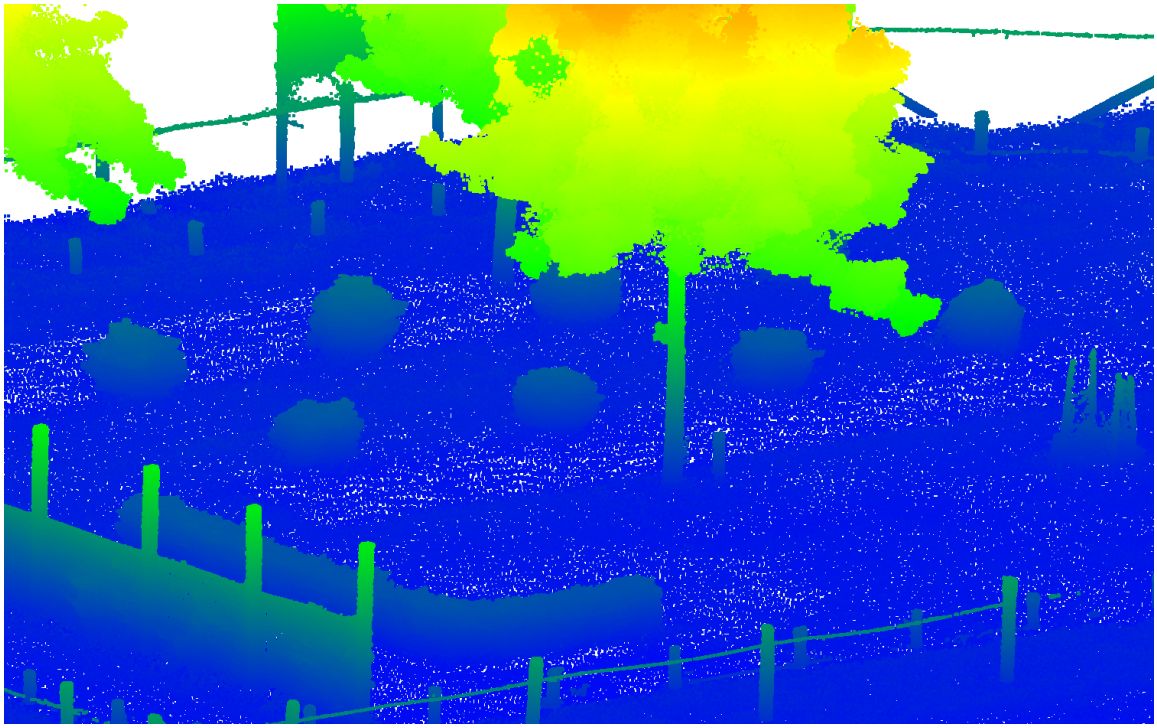



Figure 42: Relative pose error plots for DUGMA and valid frame pairs with relative GT rotation $\leq$ 20 deg. The black line shows the absolute error (translation distance or minimal rotation angle) with components in color $\times$. The magenta line shows the magnitude of GT motion, ie. allows to compare how large is the error compared to the actual motion.
*Top:* Lateral translation along the $X$ direction and its error is dominant in the shown camera frame (oriented sideways on the robot).
*Bottom:* Yaw rotation $Y$ is dominant as the robot moves on an approximately planar surface.



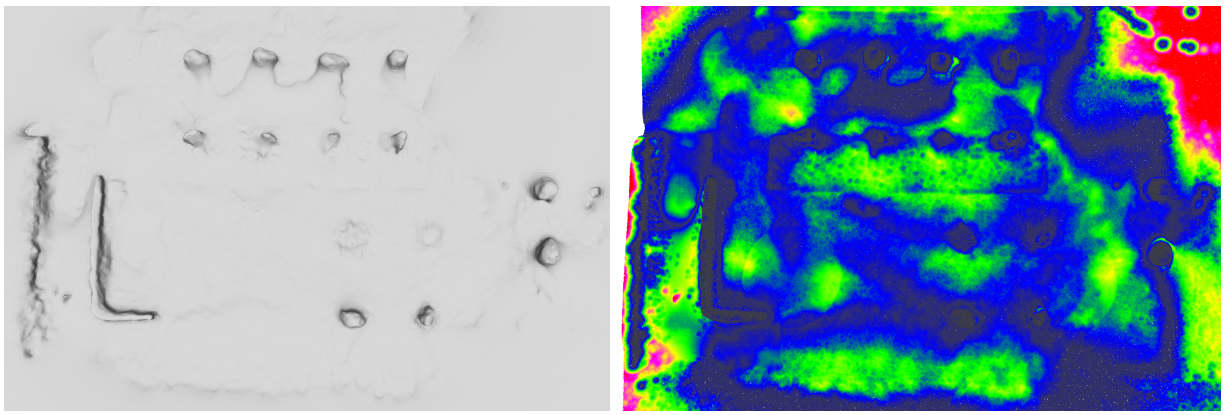
(a) Merged point cloud.



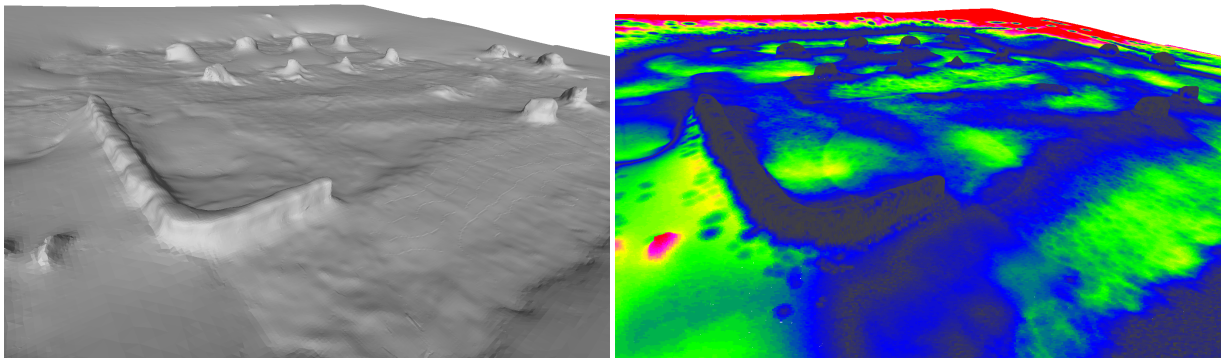
(b) Corresponding GT point cloud.

Figure 43: Result of merging 250 fused point clouds from SDF-MAN using GT poses. The color coding indicates the relative height in the point cloud.

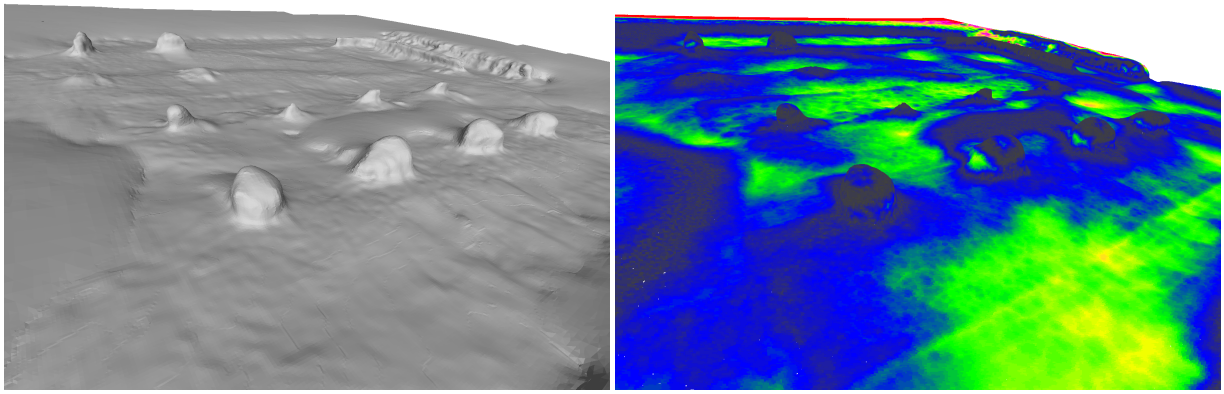
 



(a) Top view of reconstructed garden



(b) Side view with buxus hedge



(c) Side view with topiary bushes

Figure 44: PointFusion surface reconstruction of 250 fused point clouds using SDF-MAN disparities and GT poses. *Left:* shaded mesh. *Right:* color-coded accuracy (distance from GT point cloud): black = 0 cm, blue = 5 cm, green = 10 cm, red = 25+ cm.

# 6 Summary & Outlook

This report has described the algorithms used in the TrimBot2020 for SLAM, visual localization, and 3D reconstruction. We have focussed on providing technical details for parts that have not yet been published, e.g., the SLAM and localization components. In addition, the report has provided an experimental evaluation of all components on data recorded in the TrimBot2020 project (with additional experimental results available in the corresponding publications).

As shown in Sec. 5.1, the localization system works best if the 3D map used for localization is recent and was captured under similar illumination conditions. Clearly, this is a limitation in the context of building an autonomous gardening robot as this robot will need to be able to operate over longer periods of time. To this end, deliverable *D3.5* will describe an approach for *semantic visual localization* designed for localization over longer periods of time. Further potential improvements and future work include experimenting with different types of local features and using intrinsic image decomposition to better handle illumination changes.

Further future work include improving the fusion stage of the 3D processing pipeline, both in terms of processing speed and reconstruction accuracy.

# References

[1] Svärm, L., Enqvist, O., Kahl, F., Oskarsson, M.: City-Scale Localization for Cameras with Known Vertical Direction. PAMI **39** (2017) 1455–1461

[2] Li, Y., Snavely, N., Huttenlocher, D., Fua, P.: Worldwide Pose Estimation Using 3D Point Clouds. In: ECCV. (2012)

[3] Sattler, T., Leibe, B., Kobbelt, L.: Efficient & Effective Prioritized Matching for Large-Scale Image-Based Localization. PAMI **39** (2017) 1744–1756

[4] Schönberger, J.L., Pollefeys, M., Geiger, A., Sattler, T.: Semantic Visual Localization. In: CVPR. (2018)

[5] Sattler, T., Maddern, W., Toft, C., Torii, A., Hammarstrand, L., Stenborg, E., Safari, D., Okutomi, M., Pollefeys, M., Sivic, J., Kahl, F., Pajdla, T.: Benchmarking 6DOF Outdoor Visual Localization in Changing Conditions. In: CVPR. (2018)

[6] Davison, A.J., Reid, I.D., Molton, N.D., Stasse, O.: MonoSLAM: Real-Time Single Camera SLAM. TPAMI **29** (2007) 1052–1067

[7] Klein, G., Murray, D.: Parallel tracking and mapping for small AR workspaces. In: ISMAR. (2007)

[8] Newcombe, R.A., Lovegrove, S.J., Davison, A.J.: DTAM: Dense tracking and mapping in real-time. In: ICCV. (2011)

[9] Mur-Artal, R., Montiel, J.M.M., Tards, J.D.: Orb-slam: A versatile and accurate monocular slam system. T-RO **31** (2015)

[10] Middelberg, S., Sattler, T., Untzelmann, O., Kobbelt, L.: Scalable 6-DOF Localization on Mobile Devices. In: ECCV. (2014)

[11] Lynen, S., Sattler, T., Bosse, M., Hesch, J., Pollefeys, M., Siegwart, R.: Get Out of My Lab: Large-scale, Real-Time Visual-Inertial Localization. In: RSS. (2015)

[12] Arth, C., Klopschitz, M., Reitmayr, G., Schmalstieg, D.: Real-Time Self-Localization from Panoramic Images on Mobile Devices. In: ISMAR. (2011)

[13] Wang, R., Schwörer, M., Cremers, D.: Stereo DSO: Large-Scale Direct Sparse Visual Odometry with Stereo Cameras. In: International Conference on Computer Vision (ICCV). (2017)

[14] Kaess, M., Dellaert, F.: Visual SLAM with a multi-camera rig. Technical Report GIT-GVU-06-06, Georgia Institute of Technology (2006)

[15] Kim, J.S., Hwangbo, M., Kanade, T.: Motion estimation using multiple non-overlapping cameras for small unmanned aerial vehicles. In: ICRA. (2008)

[16] Lee, G.H., Li, B., Pollefeys, M., Fraundorfer, F.: Minimal solutions for the multi-camera pose estimation problem. IJRR **34** (2015) 837–848

[17] Lee, G.H., Pollefeys, M., Fraundorfer, F.: Relative Pose Estimation for a Multi-Camera System with Known Vertical Direction. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). (2014)

[18] Kneip, L., Li, H.: Efficient Computation of Relative Pose for Multi-Camera Systems. In: IEEE International Conference on Computer Vision and Pattern Recognition. (2014)

[19] Heng, L., Lee, G.H., Pollefeys, M.: Self-Calibration and Visual SLAM with a Multi-Camera System on a Micro-Aerial Vehicle. In: Robotics: Science and Systems (RSS). (2014)

[20] Harmat, A., Trentini, M., Sharf, I.: Multi-camera tracking and mapping for unmanned aerial vehicles in unstructured environments. JIRS **78** (2015)

[21] Urban, S., Hinz, S.: MultiCol-SLAM - a modular real-time multi-camera slam system. arXiv preprint arXiv:1610.07336 (2016)

[22] Liu, P., Geppert, M., Heng, L., Sattler, T., Geiger, A., Pollefeys, M.: Towards robust visual odometry with a multi-camera system. In: IROS. (2018)

[23] Geppert, M.: Simultaneous Localization and Mapping using Generalized Cameras. Master's thesis, ETH Zürich (2017)

[24] Pless, R.: Using many cameras as one. In: CVPR. (2003)

[25] Lowe, D.G.: Distinctive Image Features from Scale-Invariant Keypoints. IJCV **60** (2004) 91–110

[26] Schönberger, J.L., Frahm, J.M.: Structure-From-Motion Revisited. In: CVPR. (2016)

[27] Triggs, B., McLauchlan, P.F., Hartley, R.I., Fitzgibbon, A.W.: Bundle adjustment — a modern synthesis. In Triggs, B., Zisserman, A., Szeliski, R., eds.: Vision Algorithms: Theory and Practice, Berlin, Heidelberg, Springer Berlin Heidelberg (2000) 298–372

[28] Sattler, T., Havlena, M., Schindler, K., Pollefeys, M.: Large-scale location recognition and the geometric burstiness problem. In: CVPR. (2016)

[29] Sivic, J., Zisserman, A.: Video Google: A Text Retrieval Approach to Object Matching in Videos. In: ICCV. (2003)

[30] Wu, C.: SiftGPU: A GPU implementation of scale invariant feature transform (SIFT). `http://cs.unc.edu/~ccwu/siftgpu` (2007)

[31] Wu, C.: Towards Linear-time Incremental Structure From Motion. In: 3DV. (2013)

[32] Muja, M., Lowe, D.G.: Fast approximate nearest neighbors with automatic algorithm configuration. In: VISSAPP. (2009)

[33] Arandjelović, R., Zisserman, A.: DisLocation: Scalable descriptor distinctiveness for location recognition. In: Asian Conference on Computer Vision. (2014)

[34] Jegou, H., Douze, M., Schmid, C.: Hamming embedding and weak geometric consistency for large scale image search. In: ECCV. (2008)

[35] Fischler, M.A., Bolles, R.C.: Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM **24** (1981)

[36] Strasdat, H., Davison, A.J., Montiel, J.M.M., Konolige, K.: Double Window Optimisation for Constant Time Visual SLAM. In: ICCV. (2011)

[37] Olson, E., Leonard, J., Teller, S.: Fast Iterative Optimization of Pose Graphs with Poor Initial Estimates. In: ICRA. (2006)

[38] Agarwal, S., Mierle, K., Others: Ceres Solver. (`http://ceres-solver.org`)

[39] Lianos, N., Schönberger, J.L., Pollefeys, M., Sattler, T.: VSO: Visual Semantic Odometry. In: European Conference on Computer Vision (ECCV). (2018)

[40] Hirschmuller, H.: Accurate and efficient stereo processing by semi-global matching and mutual information. In: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2 - Volume 02. CVPR '05, Washington, DC, USA, IEEE Computer Society (2005) 807–814

[41] Honegger, D., Oleynikova, H., Pollefeys, M.: Real-time and low latency embedded computer vision hardware based on a combination of fpga and mobile cpu. In: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems. (2014) 4930–4935

[42] Honegger, D., Sattler, T., Pollefeys, M.: Embedded real-time multi-baseline stereo. In: 2017 IEEE International Conference on Robotics and Automation (ICRA). (2017)

[43] Mayer, N., Ilg, E., Häusser, P., Fischer, P., Cremers, D., Dosovitskiy, A., Brox, T.: A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In: IEEE International Conference on Computer Vision and Pattern Recognition (CVPR). (2016) arXiv:1512.02134.

[44] Horna, L., Fisher, R.B.: 3d plane labeling stereo matching with content aware adaptive windows. In: Proc. VISIGRAPP. (2017)

[45] Horna, L., Fisher, R.B.: Plane labeling trinocular stereo matching with baseline recovery. In: Proc. International Conference on Machine Vision Applications (MVA). (2017) 17–20

[46] Pu, C., Song, R., Li, N., Fisher, R.B.: SDF-MAN: Semi-supervised depth fusion with multi-scale adversarial networks. arxiv **under review** (2018)

[47] Pu, C., Li, N., Tylecek, R., Fisher, R.B.: DUGMA: dynamic uncertainty-based gaussian mixture alignment. In: Proc. 3DV. Volume abs/1803.07426. (2018)

[48] Lynen, S., Achtelik, M., Weiss, S., Chli, M., Siegwart, R.: A robust and modular multi-sensor fusion approach applied to mav navigation. In: Proc. of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS). (2013)

[49] Ummenhofer, B., Brox, T.: Global, dense multiscale reconstruction for a billion points. Int. J. Comput. Vision **125** (2017) 82–94

[50] Schöps, T., Schönberger, J.L., Galliani, S., Sattler, T., Schindler, K., Pollefeys, M., Geiger, A.: A Multi-View Stereo Benchmark with High-Resolution Images and Multi-Camera Videos. In: Conference on Computer Vision and Pattern Recognition (CVPR). (2017)

[51] Chum, O., Matas, J., Kittler, J.: Locally optimized RANSAC. In: DAGM. (2003)

[52] Camposeco, F., Cohen, A., Pollefeys, M., Sattler, T.: Hybrid scene Compression for Visual Localization. arXiv:1807.07512 (2018)

[53] Toft, C., Olsson, C., Kahl, F.: Long-term 3D Localization and Pose from Semantic Labellings. In: ICCV Workshops. (2017)

[54] Maddern, W., Pascoe, G., Linegar, C., Newman, P.: 1 Year, 1000km: The Oxford RobotCar Dataset. IJRR **36** (2017) 3–15

[55] Geiger, A., Lenz, P., Urtasun, R.: Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In: Conference on Computer Vision and Pattern Recognition (CVPR). (2012)

[56] Mayer, N., Ilg, E., Fischer, P., Hazirbas, C., Cremers, D., Dosovitskiy, A., Brox, T.: What makes good synthetic training data for learning disparity and optical flow estimation? International Journal of Computer Vision **126** (2018) 942–960 https://arxiv.org/abs/1801.06397.

[57] Ilg, E., Saikia, T., Keuper, M., Brox, T.: Occlusions, motion and depth boundaries with a generic network for disparity, optical flow or scene flow estimation. In: European Conference on Computer Vision (ECCV). (2018)

[58] Ilg, E., Mayer, N., Saikia, T., Keuper, M., Dosovitskiy, A., Brox, T.: Flownet 2.0: Evolution of optical flow estimation with deep networks. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). (2017)

[59] Scharstein, D., Hirschmüller, H., Kitajima, Y., Krathwohl, G., Nesic, N., Wang, X., Westling, P.: High-resolution stereo datasets with subpixel-accurate ground truth. In: Pattern Recognition - 36th German Conference, GCPR 2014, Münster, Germany, September 2-5, 2014, Proceedings. (2014) 31–42