TrimBot2020 Deliverable D3.4

Algorithmic Speedup and Final Evaluation

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

Abstract: This report describes the efforts on algorithmic speedup as well as the final evaluation of the outcome of WP3. The algorithmic speedup, mainly done by University of Groningen (RUG), focused on the DUGMA algorithm which is used for point cloud alignment. We also investigated using faster local feature extractors in the visual SLAM as well as using pre-computations for accelerating the motion planning for the robotic arm. Since most of the individual parts of WP3 have have been evaluated in previous deliverables (see D3.2 and D3.5), we focus on evaluating the final pipeline that was used for the final demonstrator.

Deliverable due: Month 42

Contents

1 Algorithmic Speedup

In this deliverable, we report on the progress in algorithmic speedup within the TrimBot2020 project. This is part of *Task 3.6 - Algorithmic Speedup*, which is headed by RUG. Most of the work in this task was done speeding up the the point-cloud alignment algorithm DUGMA, which was developed within the TrimBot2020 project. The original formulation and implementation of DUGMA, published in 3DV 2018, did not reach sufficient computational efficiency to be deployed on the TrimBot2020 demonstrator. The demonstrator requires interactive frame rate ($\sim$ 1 Hz), but the original implementation needed more than 10 s to register a single point cloud pair from the sensor input.

In the following of the sections, we describe the optimization choices and report the results that we achieved on the TrimBot2020 benchmark point-cloud data sets published together with the DUGMA paper. We focus on optimization of the formulation of some parts of the algorithm and on technological improvements to the implementation code.

In addition to the improvements made to the DUGMA algorithm, we also present some work done on speeding up the visual SLAM and the motion planning for the robotic arm. The outline of the algorithmic speedup is as follows:

- Algorithmic speedup of the DUGMA algorithm (Section 1.2)
- Faster local image features for visual SLAM (Section 1.4)
- Pre-computations for arm motion planning (Section 1.5)

1.1 The DUGMA Algorithm

DUGMA stands for Dynamic Uncertainty-based Gaussian Mixture Alignment. It is an algorithm for point-cloud matching based on combining error estimation from sample covariances and dynamic global probability alignment using the convolution of uncertainty-based Gaussian Mixture Models (GMM). Given two point clouds $\mathbf{X}$ and $\mathbf{Y}$ with point-wise uncertainty, DUGMA models them as two GMMs, namely $G_{\mathbf{X}}^{\mathbf{I},\mathbf{0}}(\tau)$ and $G_{\mathbf{Y}}^{\mathbf{R},\mathbf{t}}(\tau)$, which are defined as follows:

$$G_{\mathbf{X}}^{\mathbf{I},\mathbf{0}}(\tau) = \sum_{n=1}^{N} w_{x_n}(\tau, cp_{x_n}) g_{x_n}(\tau), \qquad G_{\mathbf{Y}}^{\mathbf{R},\mathbf{t}}(\tau) = \sum_{m=1}^{M} w_{y_m}(\tau, cp_{y_m}) g_{y_m}(\tau),$$

where the gaussian kernels $g_{x_n}(\tau)$ and $g_{y_m}(\tau)$ indicate the probability of finding x_n or y_n at the position τ in their own coordinate system, while $w_{x_n}(\tau, cp_{x_n})$ and $w_{y_m}(\tau, cp_{y_m})$ potential corresponding point cp_{x_n} (or cp_{y_n}) from point cloud $\mathbf{Y}$ (or $\mathbf{X}$) is near the position τ. These quantities are formally defined as:

$$\begin{aligned}
g_{x_n}(\tau) &= \frac{1}{\sqrt{(2\pi)^D |\Sigma_{x_n}|}} e^{-\frac{1}{2}(\tau - x_n)^T \Sigma_{x_n}^{-1} (\tau - x_n)} , \\
g_{y_m}(\tau) &= \frac{1}{\sqrt{(2\pi)^D |\Sigma_{y_m}|}} e^{-\frac{1}{2}(\tau - y_m)^T \Sigma_{y_m}^{-1} (\tau - y_m)} , \\
w_{x_n}(\tau, cp_{x_n}) &= e^{-\frac{1}{2}(cp_{x_n} - \tau)^T \Sigma_{x_n}^{-1} (cp_{x_n} - \tau)} , \\
w_{y_m}(\tau, cp_{y_m}) &= e^{-\frac{1}{2}(cp_{y_m} - \tau)^T \Sigma_{y_m}^{-1} (cp_{y_m} - \tau)} .
\end{aligned}$$

In DUGMA, the problem of aligning the uncertainty-based point clouds is formulated as minimizing an energy function computed by integrating the product of the two dynamic GMMs $G_{\mathbf{X}}^{\mathbf{I},\mathbf{0}}(\tau)$ and $G_{\mathbf{Y}}^{\mathbf{R},\mathbf{t}}(\tau)$. The energy function is defined as:

$$E = \int P(\tau) \log \left[G_{\mathbf{X}}^{\mathbf{I},\mathbf{0}}(\tau) G_{\mathbf{Y}}^{\mathbf{R},\mathbf{t}}(\tau) \right] \mathrm{d}\tau, \tag{1}$$

where τ integrates over all the domain of the point clouds; $P(\tau)$ is the joint probability that there is a point from X or Y at the position τ and is defined as:

$$P(\tau) = \sum_{i=1}^{N} \sum_{j=1}^{M} P(\tau, x_i, y_j) = \sum_{i=1}^{N} \sum_{j=1}^{M} g_{x_i}(\tau) g_{y_j}(\tau). \tag{2}$$

Equation (1) was thus rewritten as

$$E = \int P(\tau) \log \left[\sum_{i=1}^{N} \sum_{j=1}^{M} F^{\mathbf{R},\mathbf{t}}(\tau, x_i, y_j) \right] \mathrm{d}\tau, \tag{3}$$

where

$$F^{\mathbf{R},\mathbf{t}}(\tau, x_i, y_j) = w_{x_i}(\tau, cp_{x_i}) g_{x_i}(\tau) w_{y_j}(\tau, cp_{y_j}) g_{y_j}(\tau). \tag{4}$$

Observation 1. The weight term $w_{x_i}(\tau, cp_{x_i})$ is nearly zero when point x_i is far from any point in $\{y_j\}$. This avoids having to compute correspondences by using all y_j in place of cp_{x_i} (and similarly for cp_{y_j}) and simplify $F^{\mathbf{R},\mathbf{t}}$ with:

$$\tilde{F}(\tau, x_i, y_j) = w_{x_i}(\tau, y_j) g_{x_i}(\tau) w_{y_j}(\tau, x_i) g_{y_j}(\tau). \tag{5}$$

The optimization process for matching the point clouds is based on EM maximization of Equation (3) to get the estimated rotation and translation. It guesses the values of the parameters firstly in the previous iteration (denoted by 'old') and then calculates a posteriori probability $P^{old}(x_i, y_j|\tau)$ using Bayes' theorem, which corresponds to the expectation stage. Neglecting the constant term and using $P(\tau) \approx P^{old}(\tau)$, the computation of the target function was simplified as follows:

$$\mathcal{L} = \sum_{i=1}^{N} \sum_{j=1}^{M} \int P^{old}(\tau, x_i, y_j)\, Mah^{new}(\tau, x_i, y_j)\, \mathrm{d}\tau, \tag{6}$$

where a term similar to Mahalanobis distance is obtained:

$$Mah^{new}(\tau, x_i, y_j) = \frac{1}{2}(\tau - x_i)^T (\Sigma_{x_i}^{-1} + \Sigma_{y_j}^{-1})(\tau - x_i) + \frac{1}{2}(\tau - y_j)^T (\Sigma_{x_i}^{-1} + \Sigma_{y_j}^{-1})(\tau - y_j)\ .$$

There is no real benefit to integrate over the whole 3D space, because the values of the Gaussian functions are only significant near the data points themselves. In fact, because most values are quite low, the authors stated that *(Observation 2) the integral was approximated by a sum at only the data points to speed up the algorithm drastically*. The time complexity of the algorithm is estimated as $\mathcal{O}(MN)$. The approximated energy function becomes:

$$\tilde{\mathcal{L}} = \sum_{i=1}^{N} \sum_{j=1}^{M} \sum_{\tau \in \{x_i, y_j\}} P^{old}(\tau, x_i, y_j)\, Mah^{new}(\tau, x_i, y_j). \tag{7}$$

For more details about the mathematical formulation of the original DUGMA, we refer to [1]. In the above paragraphs, we highlighted the main parts of the algorithm, which we optimized. We provide insights about the algorithmic optimization process in the following section.

1.2 Algorithm Analysis and Optimization of DUGMA

The algorithmic speedup that we performed on DUGMA was focused on two aspects. On one hand, we examined the mathematical formulation of the algorithm, matching it with the implementation. On the other hand, we explored technical solutions for optimizing repeated operations and the use of specific data types on the GPUs used for the project experiments. It is worth pointing out that the optimization of Equation 6 is enabled by the computation of 637 coefficients, whose value computations are determined by the joint probability and distance measures of all the possible point pairs. This exhaustive process is computed by means of GPU kernels, which are the subject of the optimization described in this document.

1.2.1 Method optimization

The formulation of the joint probability of finding corresponding pairs of points at location τ (see Eq. 2) and the target function of the optimization (see Eq. 6) have a missing normalization factor $A = \frac{1}{N \cdot M}$. Although the optimization output does not change, the benefit of using the normalization factor A is two-fold: on one side, the numerical complexity of the problem is constrained, allowing for optimized computations on floating-point units; and on the other side, it allows to scale the algorithm to be used with point clouds of increasing size. The original equations were, thus, modified as follows:

$$P(\tau) = \frac{1}{NM} \sum_{i=1}^{N} \sum_{j=1}^{M} P(\tau, x_i, y_j) = \sum_{i=1}^{N} \sum_{j=1}^{M} g_{x_i}(\tau) g_{y_j}(\tau). \tag{8}$$

$$\tilde{\mathcal{L}} = \frac{1}{NM} \sum_{i=1}^{N} \sum_{j=1}^{M} \sum_{\tau \in \{x_i, y_j\}} P^{old}(\tau, x_i, y_j)\, Mah^{new}(\tau, x_i, y_j). \tag{9}$$

Although *Observation 1* and *Observation 2*, marked in the previous section, are correct in principle, the original implementation did not take them completely into account, as far as it was understood from the simulations. The computation of the joint probability in Eq. 8 is performed for each possible pair of points in the point clouds X and Y. For many point pairs, it is very close to 0, as they are in very different parts of the clouds and their uncertainty distributions do not overlap. To strengthen the above observations and fully exploit the information about the distributions of the uncertainty of position of the points in the point clouds, we ensure that the computation of the coefficients needed for the optimization in Eq. 7 is done only in the case the joint probability $P(\tau)$ of Eq. 8 is non-negligible. This reduces greatly the amount of GPU computations, which are however not contributing to the calculation of the coefficients, being then multiplied by a close to 0 probability value.

1.2.2 Implementation optimization

The second part of the DUGMA speedup focused on the optimization of the implementation with respect to the GPU characteristics and the required computation precision.

Version 2 The original implementation used the *double* datatype to represent the intermediate coefficients of the optimization, which we found do not require such fine precision. The first modification to the algorithm was the use of *single-precision* datatype, which allowed to gain

about 15% of the processing time. Subsequently, we implemented a mechanism to optimize the number of threads that are spawned to compute the optimization coefficients: the number of threads that run in parallel is computed in relation to the architecture of the GPU (i.e. available cores), the size of the point clouds to align and the size of the GPU kernel used to compute the coefficient (fixed to 32×32 GPU threads). These modifications were coupled with those presented in Section 1.2.1.

Version 2.1 This version, which is the one used in the demonstrator, further improves the performance of version 2, by tackling repeated partial computation of the optimization coefficients. From a deep analysis of the code, we noticed that several coefficients shared their mathematical formulation. This means that the same values were computed multiple times within the same GPU kernel and assigned to different locations in the GPU central memory (by memory synchronization operations).

In order to avoid repeated computations and excessive memory synchronization, we strongly modified the implementation of the GPU kernel that computes the partial values of the optimization coefficients. Only unique, in terms of mathematical formulation, coefficients are computed within the GPU kernel and their values are synchronized in the central GPU memory. The value of the duplicated coefficients and their synchronization in the central memory of the GPU is done only once, when all the GPU kernel instances have terminated their execution, before the final coefficient matrix is delivered to the EM optimization procedure.

This version is used in the demonstration TrimBot2020 prototype.

Version 3 We developed a further optimized version of the algorithm, which builds directly on the implementation of Version 2.1. It is based on the use of the *half-precision* datatype to represent the DUGMA optimization coefficients. The implementation was thus refactored to use native half-precision arithmetic directives of the GPU firmware. The use of the *half* datatype, however, impose limits on the type of GPU architecture and firmware that the algorithm can run on. The possibility of using half-precision arithmetics and memory synchronization (for 16bit memory chunks) is guaranteed on architectures with compute capability higher than 7.0. We experimented with an Nvidia V100 GPU card, but this version is not yet deployable on the demonstrator.

In the following section, we report the results of the speed-up optimization in terms of gain in processing time, rotation and translation error difference with respect to the original implementation.

1.3 Evaluation of DUGMA

We run the evaluation on the TrimBot2020 point clouds released with the original DUGMA paper [1]. There are 30 pairs of point clouds, with ground truth translation and rotation matrices, registered with Kinect sensors. For each point, a distribution of its position uncertainty is provided. Only one point cloud (no. 26) is excluded from evaluation as the results of DUGMA were not reliable (it is excluded also in the original paper).

We carried out experiments on two GPU cards, namely a GTX1080 and a V100 card. In the following, we present plots of the achieved speed-up, and rotation and translation errors.

1.3.1 NVIDIA GTX 1080

In Figure 1, we show the speed-up achieved on the 29 valid point clouds. The V2.0 algorithm achieved an average speed-up of $3.71x$, while the V2.1 achieved a speed-up of $7.79x$. We also report the partial results achieved in two phases of the algorithm, namely the coefficient computation and aligning the energy minimization. The extra modification made in V2.1 regarding the optimization of the duplicate computations shows the strongest contribution to the overall algorithmic speedup.

In Figure 2 and Figure 3, we report an analysis of the translation and rotation errors respectively achieved on the point clouds of the TrimBot2020 data set. In the bottom figures, when the bars are above the x-axis, the optimized algorithms achieve less error than the original implementation. However, the differences are very small (also when the optimizations perform less precisely), indicating that the sped-up version achieves accuracy that is usable in practice,

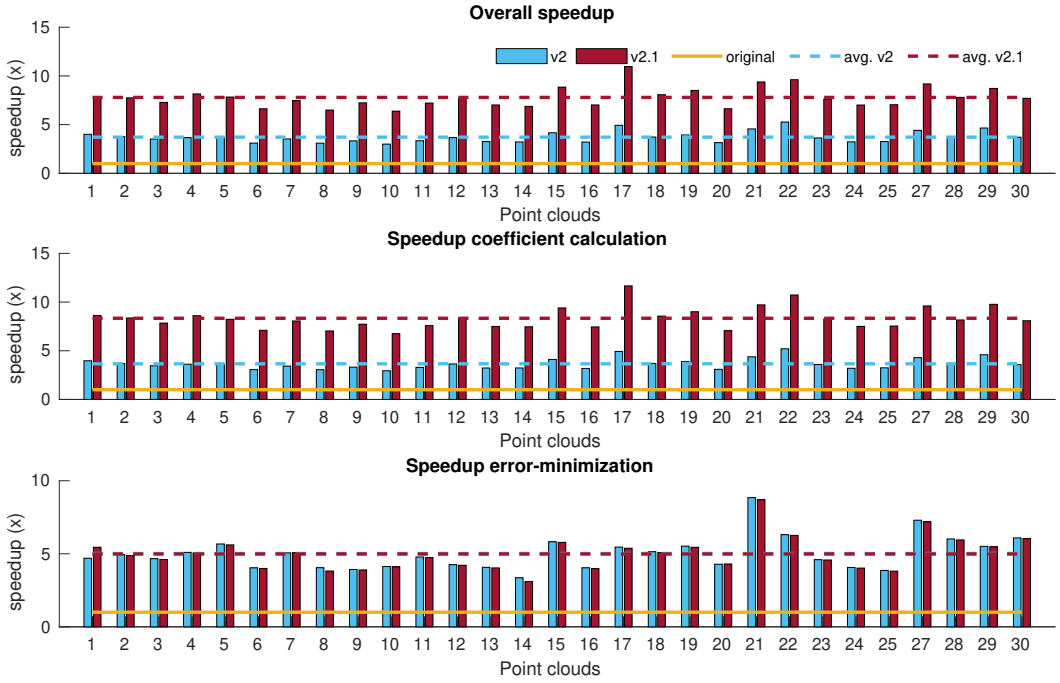


Figure 1: Speed-up achieved per point cloud of the TrimBot2020 data set. Average results are given by the horizontal lines. Compared to the original implementation, the improved versions are significantly faster.

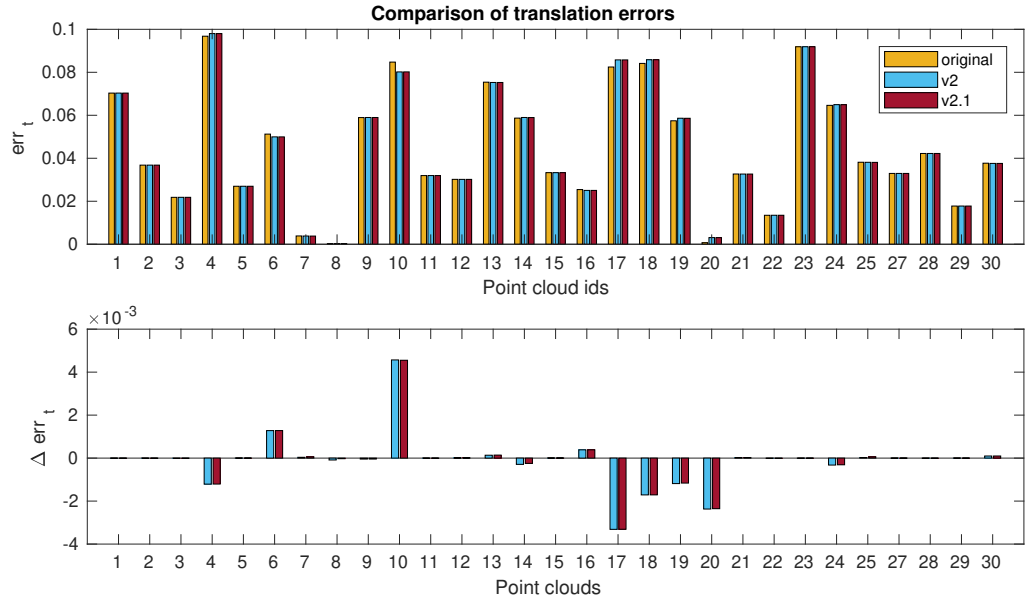


Figure 2: Absolute translation errors (top). Difference between the translation error of the original and sped-up algorithms (bottom).

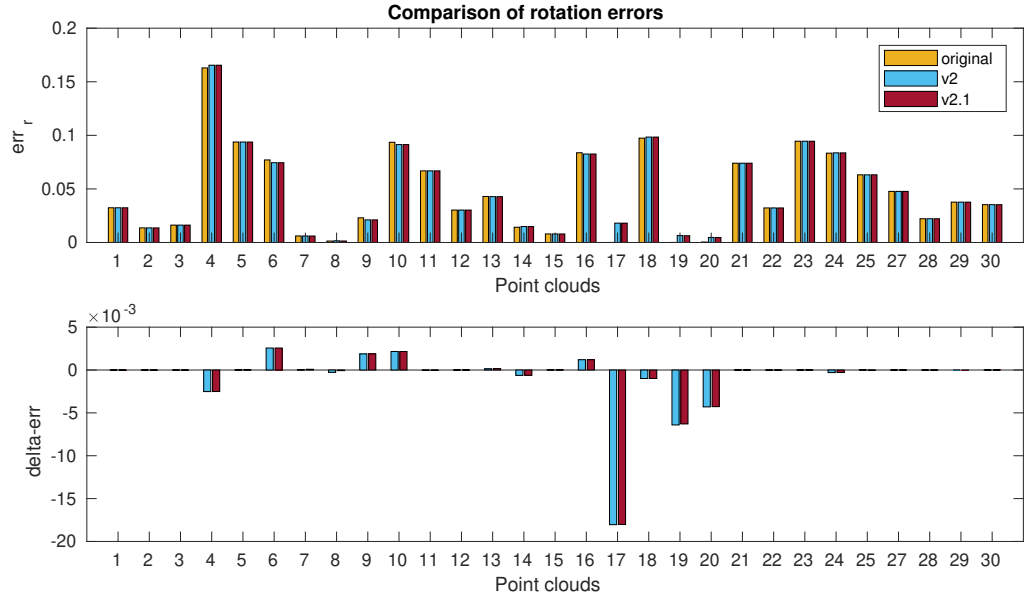


Figure 3: Absolute rotation errors in degrees (top). Difference between the rotation error of the original and sped-up algorithms (bottom). For practical purposes the difference in the rotation estimates are negligible.

without losing the precision of the original implementation.

1.3.2 NVIDIA V100

In Figure 4, we show the speed-up achieved on the 29 valid point clouds. The V2.0 algorithm achieved an average speed-up of $1.03x$, while the V2.1 achieved a speed-up of $2.51x$. The lower speed-up with respect to the GTX runs is due to the fact that the V100 card has double-precision units, which are able to process quite efficiently the operations of the original implementation. The half precision implementation, instead, achieves an average speed-up of $6.29x$.

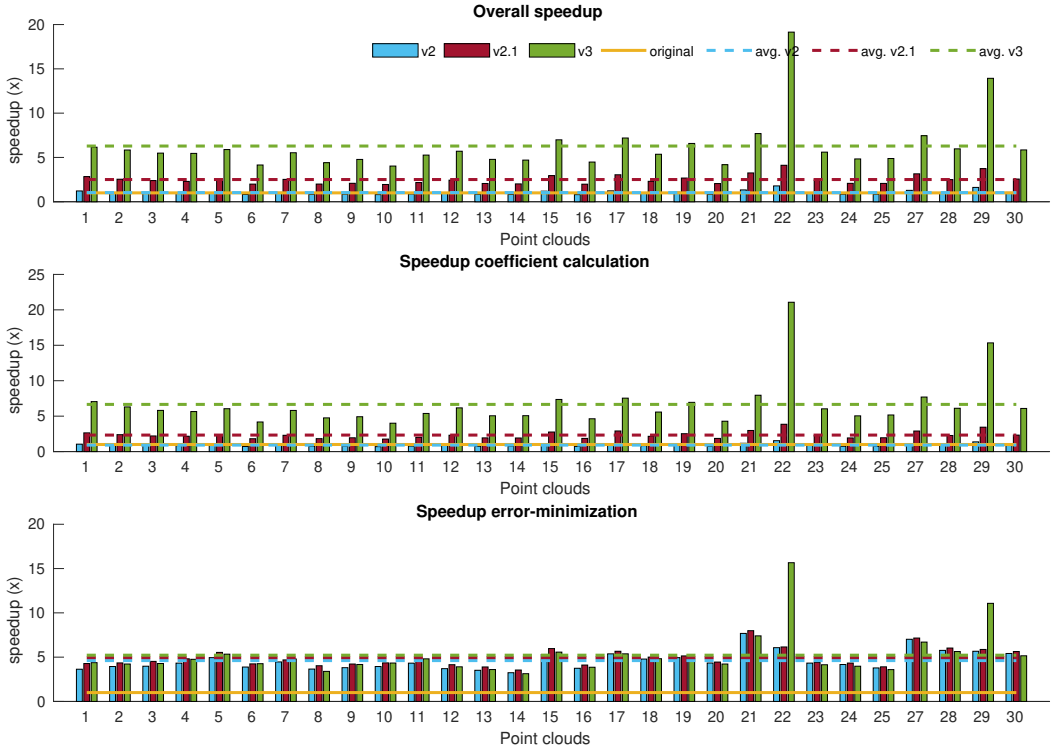


Figure 4: Speed-up achieved per point cloud of the TrimBot2020 data set using the V100 GPU. Average results are given by the horizontal lines.

In Figure 5 and Figure 6, we report an analysis of the translation and rotation errors respectively achieved on the point clouds of the TrimBot2020 data set using the NVIDIA V100 card to run the algorithm. It is worth noticing that the *half-precision* implementation (V3) obtains larger errors than the single precision version. However, the error is at most $8mm$ for the estimated translation matrix and $0.04°$ for the estimated rotation matrix. These errors do not harm the use of this version in case a suitable GPU is deployed on the robot.

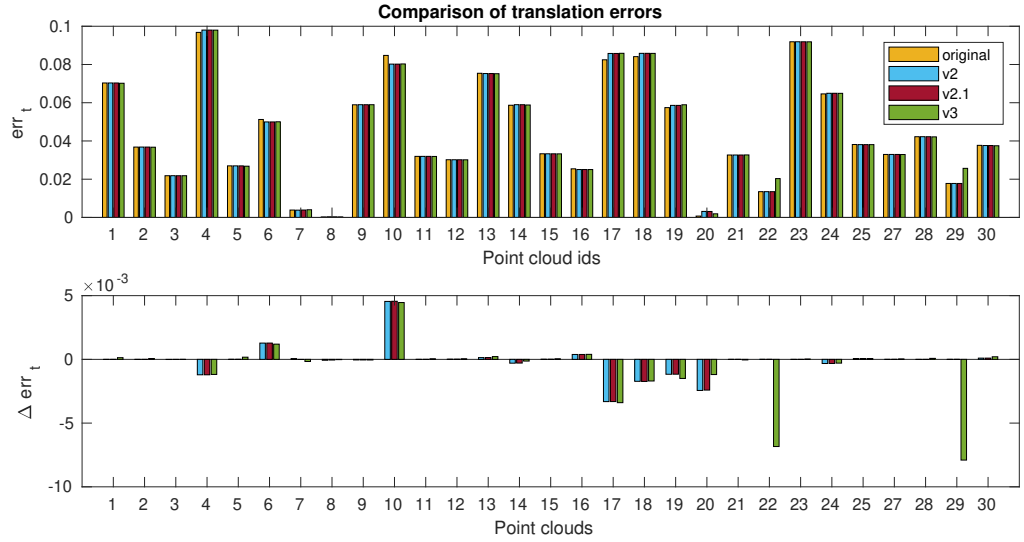


Figure 5: Run on the V100 GPU card. Difference between the translation error of the original and sped-up algorithms (bottom).

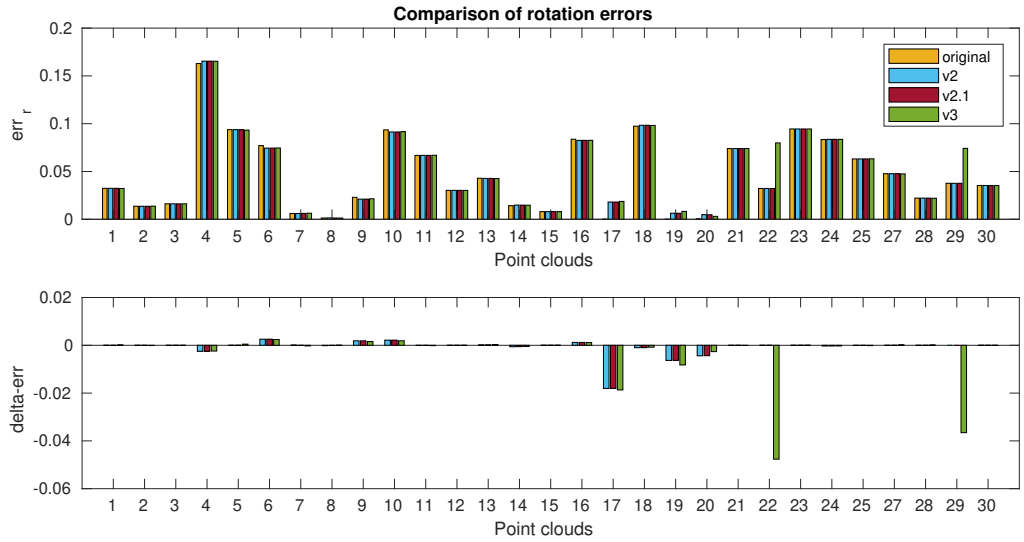


Figure 6: Run on the V100 GPU card. Absolute rotation errors in degrees (top). Difference between the rotation error of the original and sped-up algorithms (bottom).

1.3.3 Summary of processing time

In Table 1, we report the overall processing times achieved by the original implementation of DUGMA compared to those recorded when running the optimized versions. The reported results are average times (in seconds) for the processing of the 29 valid point clouds of the considered dataset.

	GTX1080	**V100**
Original	171.1	73.2
V2.0	46.5	72.6
V2.1	21.9	29.7
V3.0	–	12.6

Table 1: Processing times (in seconds) of DUGMA.

1.4 Faster Local Features for Visual SLAM

The current visual SLAM system relies on SIFT features for finding sparse correspondences between images. Since the feature extraction and matching takes a non-trivial part of the per-frame runtime, we investigated the possibility of using a faster local feature extractor. In particular we decided to compare against SURF (Speeded-Up Robust Features) [2]. Similarly to SIFT, SURF can extract 128 dimensional features, but utilizes integral images to speed up the feature and descriptor extraction. The current pipeline uses the SiftGPU implementation from Wu [3]. For a fair comparison we use the GPU accelerated CudaSURF from OpenCV [4]. In our experiments the average runtime for the feature extraction was circa 45 ms (SIFT) and 35 ms (SURF). For the experiment, we chose parameters such that the SURF extractor returned approximately the same number of features as the SIFT implementation. For the particular GPU implementations that we compare, the SURF-based feature extraction is only slightly faster compared to the SIFT-based approach.

While SURF features are less computationally expensive, we found that they provide significantly worse matching accuracy in our settings compared to SIFT. Figure 7 provides qualitative results for one of the localization experiments from Section 2.1. Using SURF features, large parts of the trajectories were not be able to be localized at all. The dataset shown in Figure 7 is also one of the easier cases since the map and localization datasets were recorded on the same day. For many of the cross-day datasets, the visual pipeline did not manage to localize, even a single time, using SURF features. Since the localization performance was already a limiting factor in navigation, we decided to not pursue this direction any further.

1.5 Pre-computations for Arm Motion Planning

The motion planning for the bush trimming used in the TrimBot2020 project was developed by WUR. To generate good arm trajectories for bush trimming, the problem is posed as a Generalized Traveling Salseman Problem (GTSP) where the nodes encode solutions to the inverse kinematics problems for the arm. The edges between the nodes are weighted by how easy it is to move between the two poses. Solving the GTSP, even approximately, is very

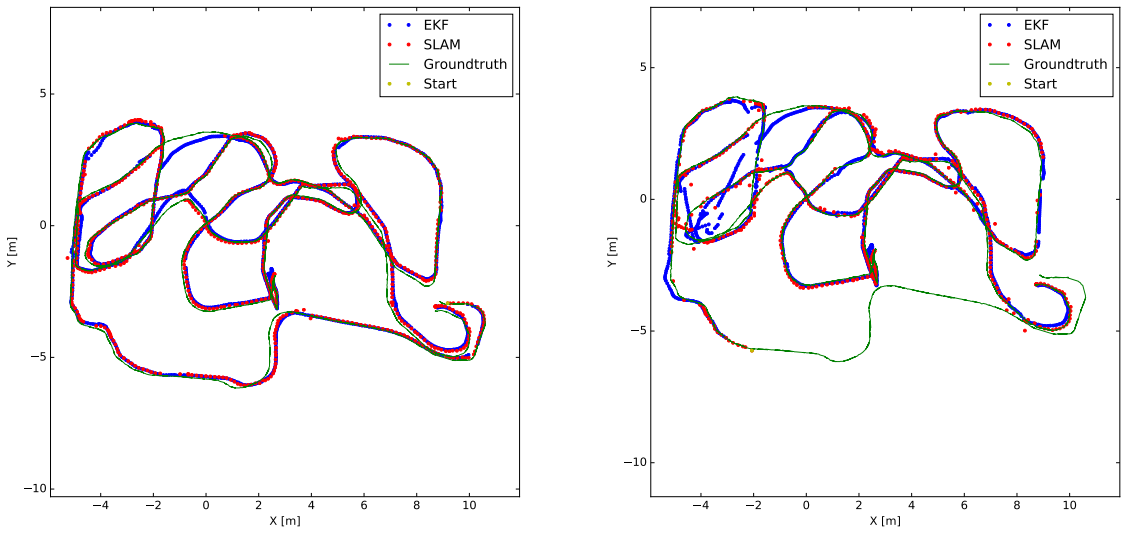


Figure 7: Qualitative comparison between using SIFT *(Left)* and SURF *(Right)* for local features in the visual SLAM pipeline. Using the less computationally expensive SURF features the localization performance is significantly worse. For large parts of the trajectory the localization failed completely.

expensive. To make this approach tractable to be run in the garden, WUR developed a method relying on pre-computing solutions to the GTSP problem and storing them in a database. During planning, the closest pre-computed instance is retrieved and then locally optimized to fit the current scenario. For more details about this approach can be found in deliverable *D2.6 - Final Manipulator, tools and algorithms* and [5]. Using the pre-computed solutions the runtime of the arm planned was reduced from multiple minutes (as was the case during the second demonstrator) to a couple of seconds. This allowed for the planning to be used in the garden for the final demonstrator.

2 Final Evaluation

In this section, we present the final evaluation of the outcome of WP3. Since most of the individual parts have been evaluated in previous deliverables (D3.2 and D3.5), the focus will be to evaluate the methods on the platform in the state it was in for the final demonstrator. Compared to previous platforms, the base camera-ring is now placed slightly lower. In Section 2.1 we evaluate the accuracy of the localization pipeline with newly collected datasets with the final prototype. Section 2.2 provides an evaluation of the localization for datasets collected where the TrimBot was navigating in the garden autonomously. In Section 2.3, we provide a comparison of the feature based image retrieval currently used in the visual localization pipeline and the learned place-recognition proposed by RUG. Finally, in Section 2.5 we evaluate and compare different methods for dense reconstruction.

2.1 Evaluation of Final Visual SLAM and Localization Pipeline

During September 2019, multiple datasets were recorded in the Wageningen test garden. These datasets were recorded with the final demo platform, except for the robotic arm not being attached. This was to allow room for a laser prism which was used to generate the ground truth data. Ground truth position and orientation measurements were obtained using a combination of LiDAR and TopCon laser tracker as described in *D7.4 - Ground-truth data definitions and acquisition*.

We selected 3 datasets for mapping and 11 datasets for evaluating the vehicle localization pipeline. The mapping datasets were collected on different days in the first week of September 2019. The datasets were all recorded in the afternoon to be similar to the conditions of the final demonstrator. In Deliverable D3.2, we previously performed evaluation of the localization performance when the mapping and localization datasets were recorded during different times of day (e.g. mapping during the morning and localizing in the evening). Note that for some datasets the laser tracker briefly lost track of the TrimBot due to occlusion. The images where no ground truth position is available are excluded from the evaluation.

We evaluate both the raw output from the visual localization pipeline, as well as the smoothed output from the EKF (see Deliverable D3.5 for more details) which fuses the wheel odometry and on-board IMU with the SLAM poses. Table 2 shows the statistics for the absolute pose errors for the localization datasets. The EKF produces slightly larger errors in average which is to be expected since it extrapolates between the images using more noisy sensors, wheel odometry and IMU. This extrapolation is necessary for the navigation which requires pose estimates with a lower latency than the visual SLAM pipeline is able supply. Figure 10 shows the distribution of the processing time for the visual SLAM system. While most frames can be processed in ≈ 150 ms, there are some outliers where the processing takes significantly longer. During this time period, the robot at normal speed ($v = 25$ cm/s) will move typically 4 cm away (or up to 25 cm in the worst case). This latency-induced error is to a large extent mitigated using inertial measurement buffer in the EKF. See Section 2.2 for more details about this and an evaluation of the localization pipeline during autonomous navigation. Furthermore, if the robot is incorrectly localized it takes the EKF some additional frames to trust the new position once SLAM correctly re-localizes in the map, which also leads to higher errors. Figure 8 shows the error distributions for the absolute localization errors. We also show some qualitative results of the localization performance in Figure 9. The EKF correctly filters spurious pose estimates returned by the visual localization pipeline and in general provides a smoother trajectory which is more suitable for navigation tasks.

Note that the errors statistics above only account for the localized frames, i.e. when the pipeline returned an estimate. In the last column of Table 2, we also present the statistics of how often the robot was localized in the garden. This was measured as the percentage of ground truth measurements from the laser tracker, which has a pose estimate within 0.5 seconds. For this evaluation, we started counting after the first pose estimate was returned from the localization pipeline. Using the fused output from the EKF, the robot is able stay localized for longer periods of time.

2.1.1 Effects of Rotation-variant Local Features

One of the changes made to the to visual localization pipeline during the preparation for the final demonstrator was switching to rotation-variant local features. The SIFT algorithm normally

	Localization error (*cm*)		Orientation error (*deg*)		Loc. rate
	Average	Median	Mean	Median	
GC-SLAM	17.66	12.53	6.09	5.05	92.3%
EKF	22.83	16.55	7.20	5.40	96.1%

Table 2: Localization errors for the evaluation in Section 2.1.

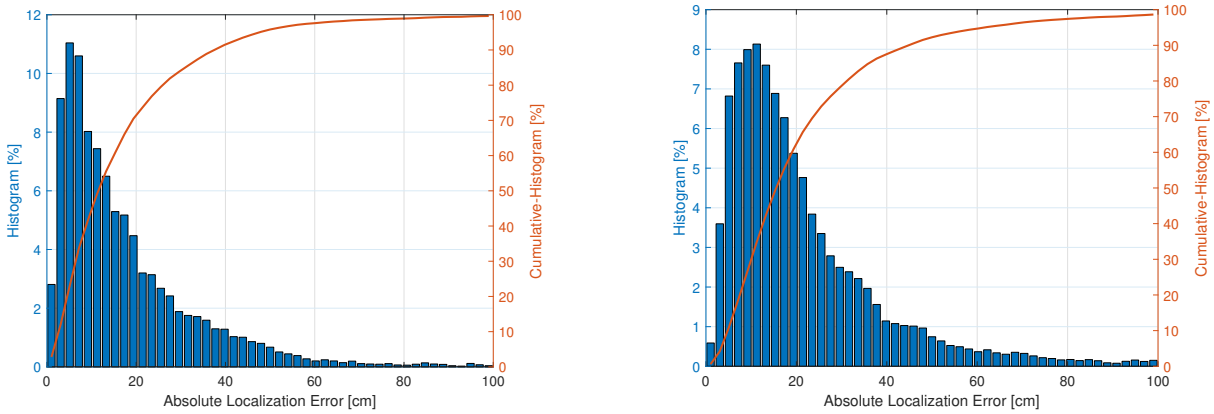


Figure 8: Error distributions for the absolute localization errors in centimetres from the evaluation in Section 2.1. Note that while the errors for the fused EKF measurements are slightly larger, the EKF provides measurements with significantly lower latency. *Left:* Visual SLAM. *Right:* EKF.

tries to compute rotation invariant descriptors by computing the descriptors with respect to the dominant gradient direction. While this provides some invariance to in-plane rotations, it also results in less discriminative descriptors. Since in-plane rotations are unlikely in the TrimBot scenario (the robot having close to planar motion) we experimented with using Upright-SIFT [6], which is a variant of the SIFT feature descriptor which skips the orientation alignment step. In our experiments, this turned out to greatly improve the ability to relocalize in the pre-built maps. Figure 11 show qualitative results for one of the localization experiments from the previous section where we compare SIFT and Upright-SIFT. Note that all experiments in the previous section is using the latter.

2.2 Visual Localization in Live Navigation Scenarios

The datasets used for evaluation in Section 2.1 were all recorded by manually driving the TrimBot in the test garden. In this section, we instead evaluate the performance of the localization pipeline during autonomous navigation. During the integration meetings preceding the final demonstrator, we found that the latency of the visual localization pipeline was leading to unstable navigation behaviour where the robot would drive too quickly close to obstacles before getting an updated pose estimate. To avoid this problem, we decided to reduce the driving speed of the platform, which gave more time for the visual localization to provide a pose, but it also turned out to improve the accuracy of the localization.

One benefit of the performing evaluation with autonomous navigation is that we can more

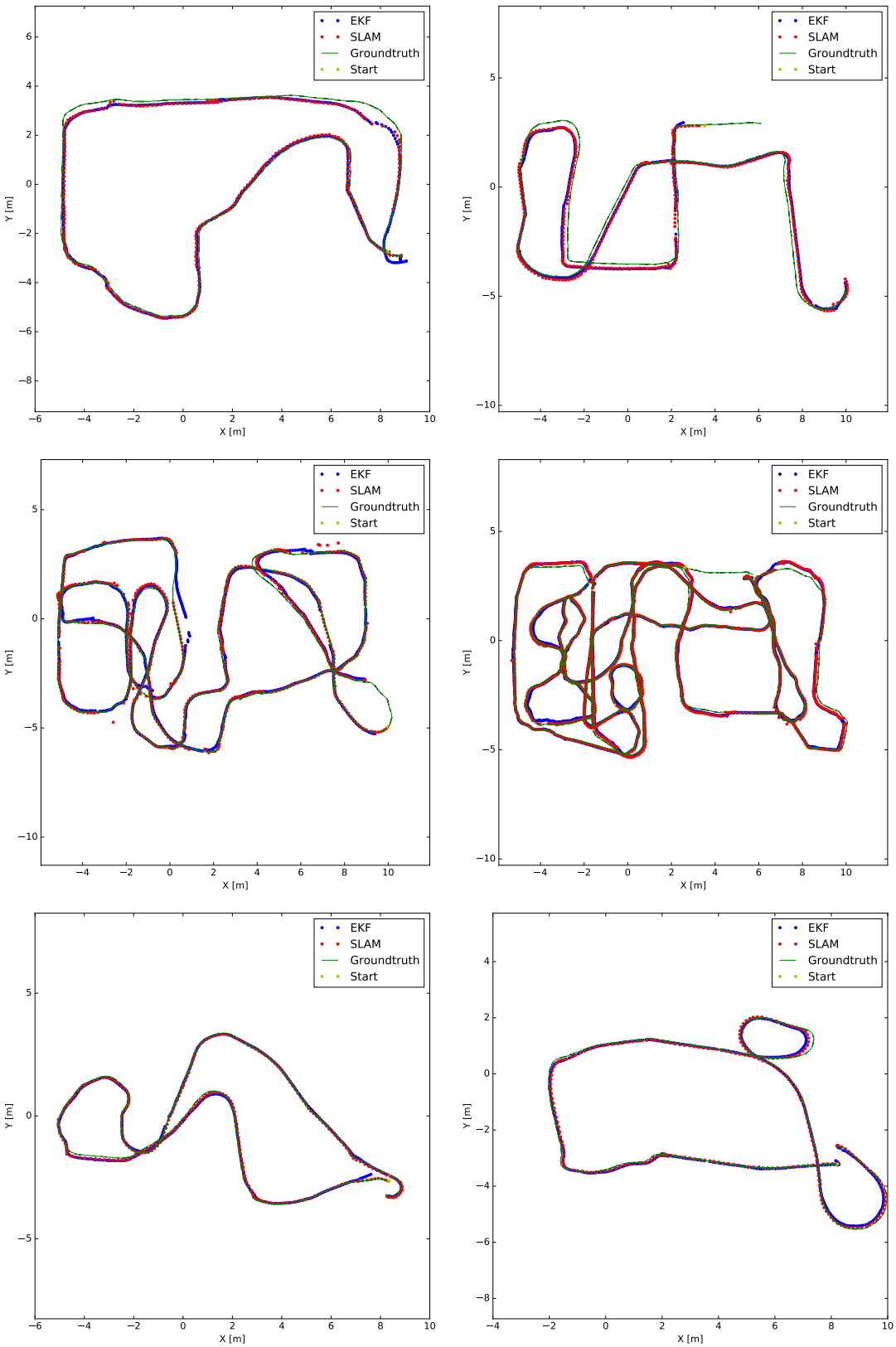


Figure 9: Qualitative examples of the localization results for some of the sequences considered in Section 2.1. The visual localization results mostly agree with the ground trajectory provided by the laser tracker.

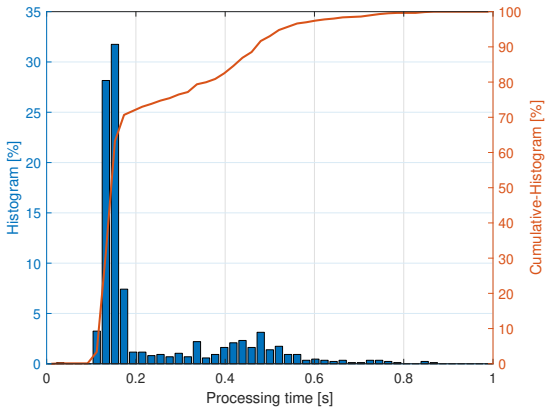


Figure 10: Histograms of the processing time for the visual SLAM system. While most frames are processed in $\approx$ 150 ms (camera framerate is about 5 Hz), some take significantly longer.

easily setup repeatable experiments to test the reliability of the localization pipeline. To this end, we setup an experiment where we manually specified four navigation goals in the garden, which creates a loop. We then let the robot navigate autonomously to each of these goals while recording the ground truth position using a laser tracker. We specified two different sets of navigation goals and repeated the experiment six times for each set. Figure 13 and Figure 14 shows the resulting trajectories of the experiments. In each of the 12 trials the robot was able to successfully navigate to each of the specified goals. Note that for the second trajectory there is no straight path between the navigation goals, but the robot needs to actively avoid the obstacles (bushes) to reach the goals. The error distributions for the two experiments are shown in Figure 12. In general, the localization errors are lower for the datasets recorded with the slower driving speed. Again, EKF is able to filter our spurious outlier poses produced by the visual SLAM. In Figure 14, we can see an instance where the localization failed and the navigation started the recovery behaviour due to being too close to an obstacle. Nonetheless, the system managed to correctly relocalize and was able to continue successfully towards the navigation goal. Table 3 shows the error statistics for the navigation experiments. Note that since we did not record the LiDAR information for these datasets, we are only able to evaluate the error in the position which we get from the TopCon laser tracker.

	GC-SLAM		EKF		Successful trials
	Average	Median	Mean	Median	
Rectangle	8.37	6.37	12.24	10.54	6 / 6
Hourglass	4.91	4.04	8.67	6.94	6 / 6
Total	6.83	5.22	10.61	8.74	12 / 12

Table 3: Absolute Localization Errors (in centimetres) for the evaluation in Section 2.2

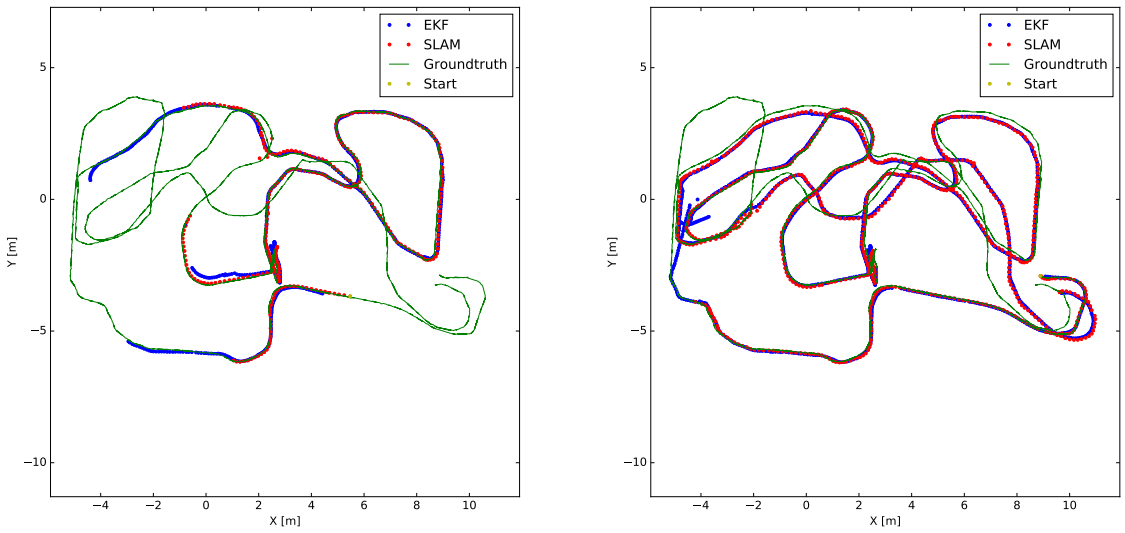


Figure 11: Qualitative comparison of the effects of rotation-variant local features in the visual localization pipeline. *Left:* SIFT. *Right:* Upright-SIFT. Using the rotation-variant features (*Right*) the visual localization pipeline is able to correctly localize more often.

2.3 Comparison of Image Retrieval Methods

The visual localization system uses an image retrieval method to find constraints to the pre-built map. The image retrieval method currently being used in the SLAM is based on the COLMAP [7] vocabulary matching functionality and relies on a visual vocabulary. In a first step, each feature descriptor in the query image is assigned to five visual words by fast approximate nearest neighbor search (see [8] for details). Features of the map images are similarly assigned to their nearest vocabulary words and stored efficiently using Hamming embedding [9]. For each of the query image features and each of their assigned visual words, the corresponding map features vote for their respective images. The impact of each vote is weighted by the Hamming distance between the query and map feature. To increase robustness of the retrieval pipeline, votes are additionally weighted using burst normalization [10] to limit the impact

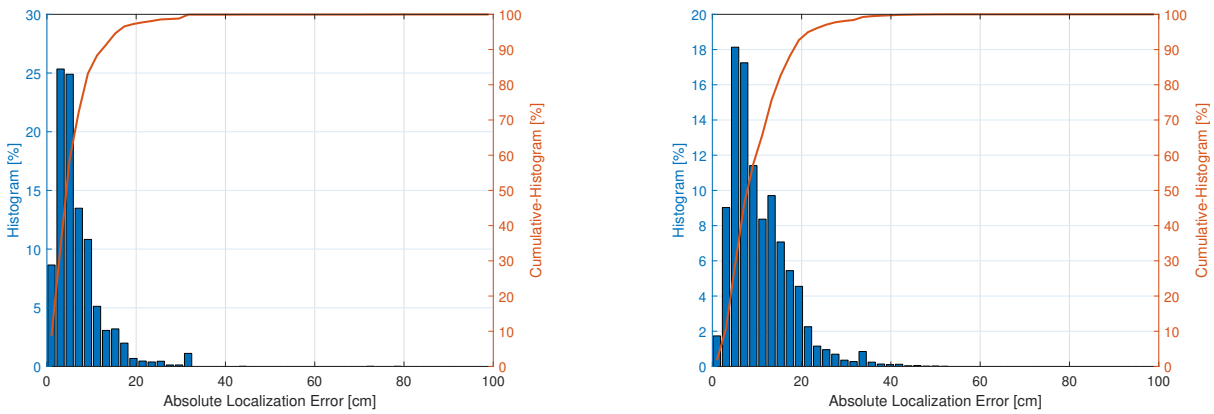


Figure 12: Error distributions for the absolute localization errors in centimetres from the live navigation evaluation in Section 2.2. *Left:* Visual SLAM. *Right:* EKF.

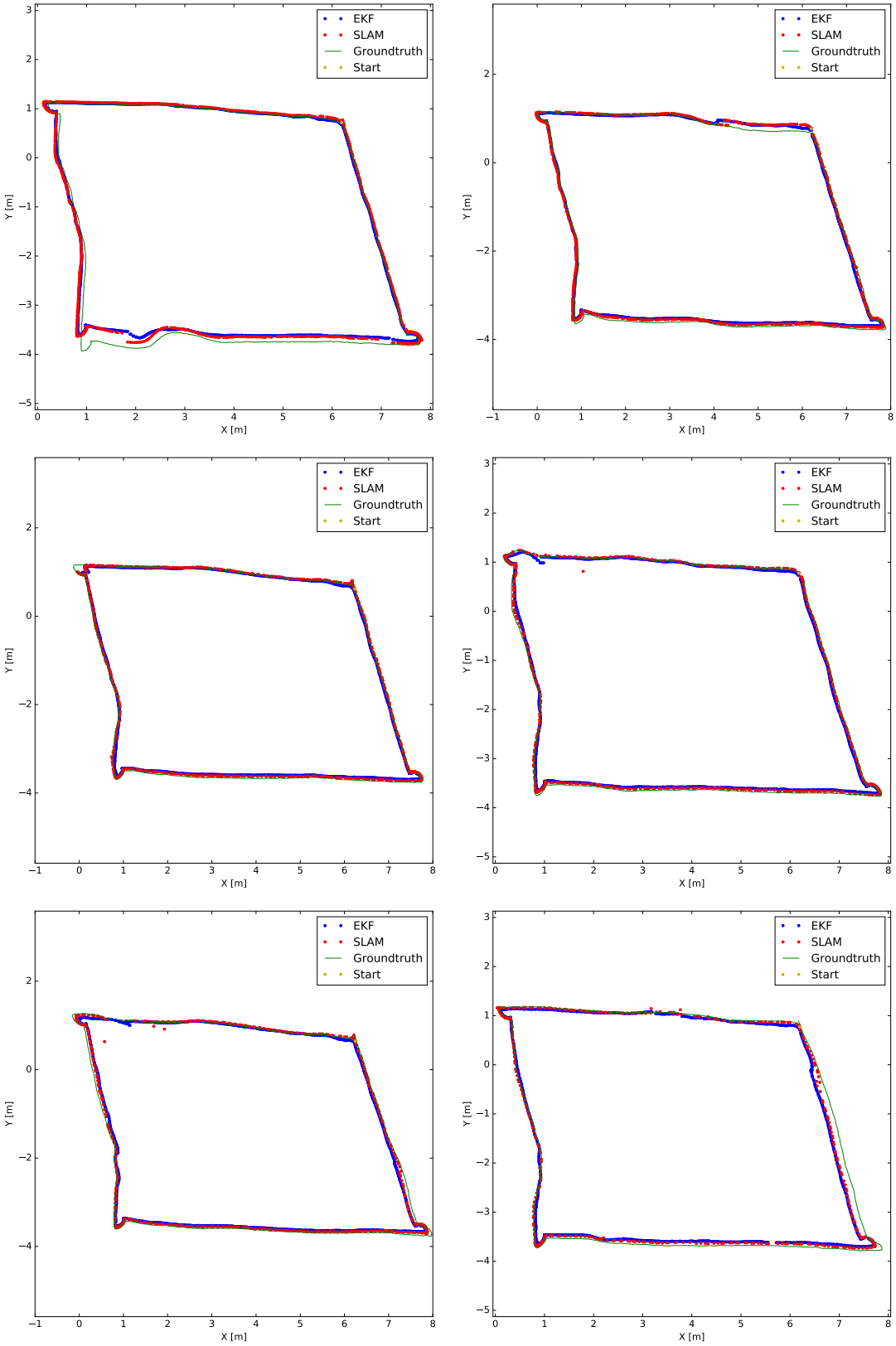


Figure 13: Qualitative results for the first path for the navigation/localization evaluation.

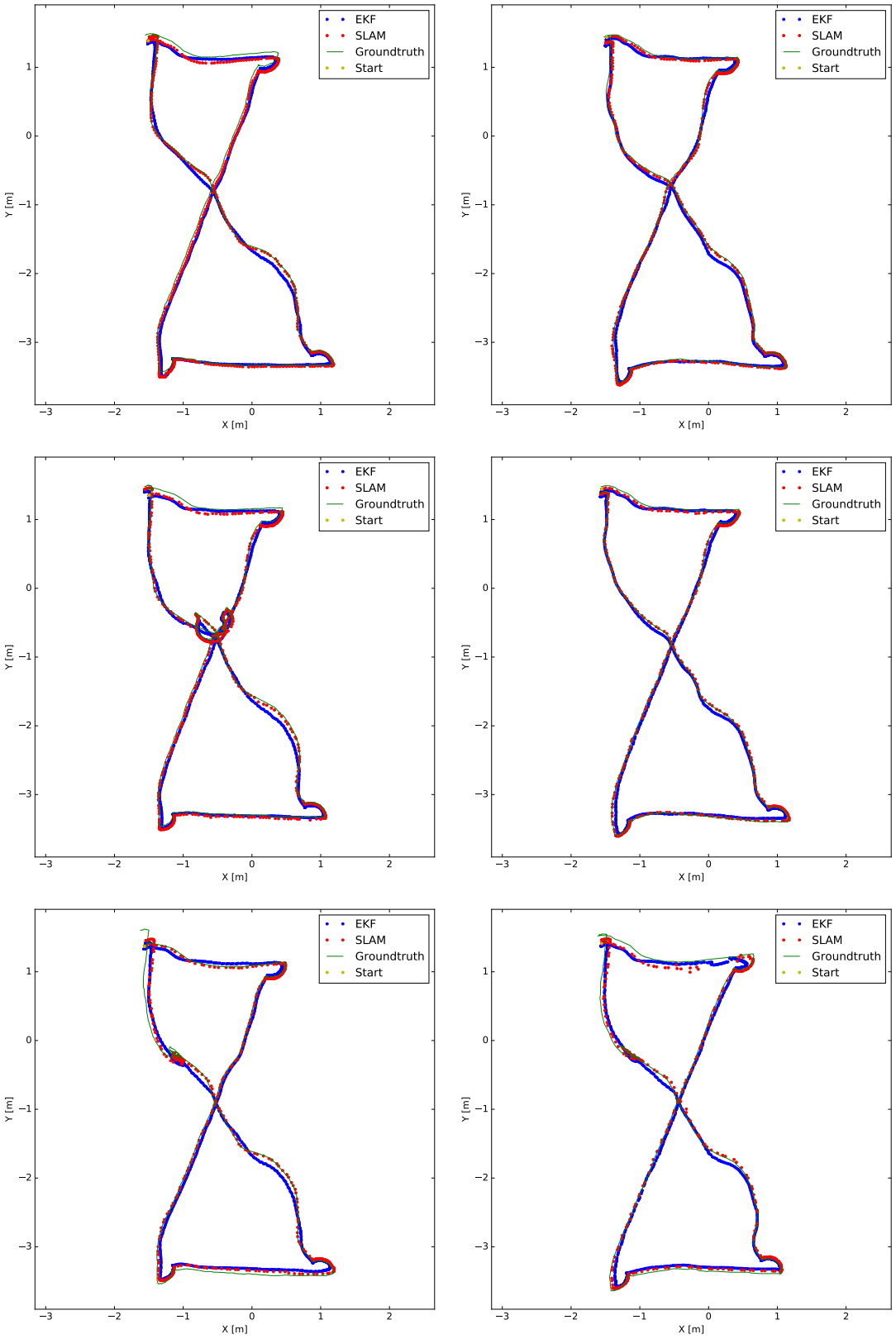


Figure 14: Qualitative results for the second path for the navigation/localization evaluation. In the third run (left in second row) the robot drove too close to an obstacle and the navigation initiated the recovery behaviour. The robot was able to relocalize correctly and could successfully continue to the target goal.

of densely populated and therefore non-distinctive regions of the feature space. We keep the images with the highest number of votes as candidates for subsequent pairwise feature matching.

One possible direction for improving the performance of the SLAM would be to use a better image retrieval. ETHZ experimented with simply retrieving the map images with most similar camera poses to the previous pose estimate. This does not consider the image content at all and relies on a previous localization being correct. The idea was to use this together with the original image retrieval to get more robustness. Due to time constraints and since we were able to reach satisfactory results without it, this method was not integrated in the pipeline used for the final demonstrator.

In Section 2.3.2 we show a comparison of the image retrieval method used in the visual SLAM with two other approaches; the position prior based approach described above and the CNN-based place recognition developed by RUG.

2.3.1 CNN-based Place Recognition (RUG)

We employed a Siamese CNN architecture with two backbone networks that share their weights to compute the descriptors of two input images (see Figure 15). During training, the weights of the networks are updated so as to optimize a Contrastive Loss function that compares the computed descriptors, f_0 and f_1. As initial conditions for the learning process, we selected a DenseNet model pre-trained on ImageNet. We learn garden-specific representations by training the considered models with images of the map sequences [11].

For all models, we evaluate the descriptor that we compute by means of a *global AvgPool* layer that we add in the end of the network. We use these descriptors to optimize a Contrastive Loss function, formally defined as:

$$L(f_0, f_1) = \frac{1}{2N} \sum_{n=0}^{N-1} \left(y^{(n)} d(f_0^{(n)}, f_1^{(n)})^2 + (1 - y^{(n)}) \cdot \max(\alpha - d(f_0^{(n)}, f_1^{(n)}), 0)^2 \right) ,$$

where $d(f_{n0}, f_{n1}) = ||f_{n0} - f_{n1}||_2$ is the Euclidean distance between the feature vectors f_0 and f_1, y_n is the ground truth label (0: dissimilar, 1: similar) [12]. The hyperparameter α is the loss margin, i.e. a threshold value of the descriptor distance above which a pair of images is not considered as depicting the same place. We set α as the value of the threshold that contributes to the highest F_1-score on the training set (map sequences). We trained the considered model for 30 epochs, using the images included in the map sequences. The sets contains an average of 50k image pairs with positive class label. We set the batch size equal to 16 pairs and included in each training epoch the positive pairs and an equal number of random negative pairs, in order to avoid class unbalance while training. The total amount of training iterations is thus approximately 100k. Stochastic Gradient Descent was used for training, with an initial learning rate of 0.01, which was decreased by 10^1 every 5 epochs.

As stated in [13], fine-tuning all the convolutional kernels in a CNN is not necessary to learn effective descriptors for visual place recognition. Early layers of a CNN, indeed, learn low level visual features (i.e. gabor-like filters) [14], while deepers layers learn to detect more complex, task-specific features. We thus train the parameters of the deeper layers, which compute semantically rich features. In particular, for the DenseNet-161 backbone CNNs, we train the last residual block (block5). This choice allows to learn the value of a reduced set

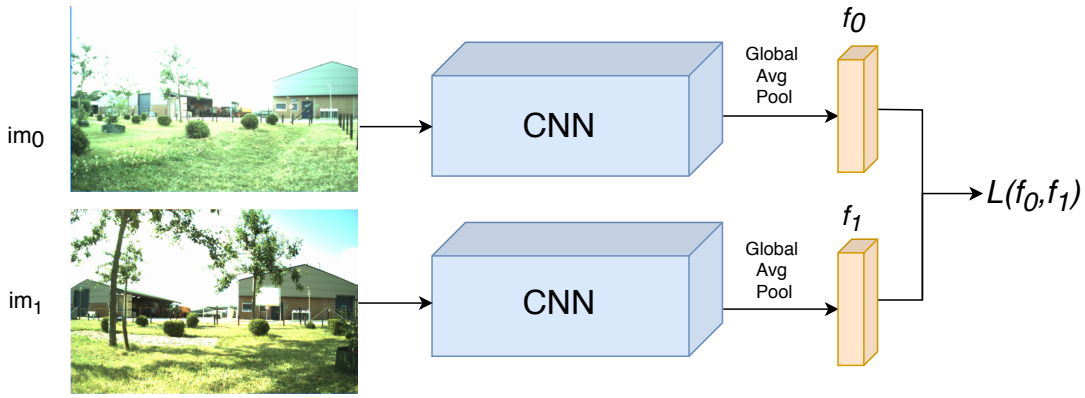


Figure 15: Sketch of the employed architecture for learning garden-specific descriptors for place recognition. We feed a pair of images to a siamese CNN architecture and compute their representation with a Global Average Pool layer. The training is guided by optimizing a contrastive loss function $L(f_0, f_1)$.

of parameters, i.e. only those of the deeper convolutional layers, simplifying the optimization problem and reducing the training time.

2.3.2 Evaluation of Image Retrieval Methods

For the experiments, we chose three pairs of mapping and localization datasets from the evaluation in Section 2.1. From the three mapping datasets, we extracted the images that were used as keyframes (circa 5k images per dataset) and then circa 1.5k query images from the corresponding localization datasets. For each image, we retrieved the top-5 best candidates from the mapping set using the two methods. Since the end-goal is to find additional correspondences to the map, we evaluate the number of geometrically verified matches between the map and the retrieved images. Table 4 shows the percentage of retrieved images with at least k verified matches. The method from RUG was developed and trained with color images. To be able to compare the methods, we use the corresponding color image for each query image in the dataset (note that the SLAM only use the grayscale images). The effect of this should be negligible since the baseline between the color and grayscale cameras is small ($\approx$ 3 cm). Figure 16 provides qualitative results of the image retrieval methods.

Table 4 shows that the baseline image retrieval already works very well. It should be noted that the method from RUG was not directly developed for this task and does not try to maximize the matchability of the retrieved images. In contrast, the image retrieval in the visual SLAM, directly uses the local features which we try to match later, which biases the results towards this method.

2.4 Evaluation of Visual Servoing

In the previous evaluations of the visual SLAM system (Deliverable D3.2), it became apparent that it would not be able to reach the accuracy needed for performing bush-trimming. It was decided to instead only rely on the visual SLAM to navigate to a position near the bush (approximately 1 meter away) and then use visual servoing for making the final approach to the bush. The visual servoing only needs estimates of the relative position (distance and angle) of

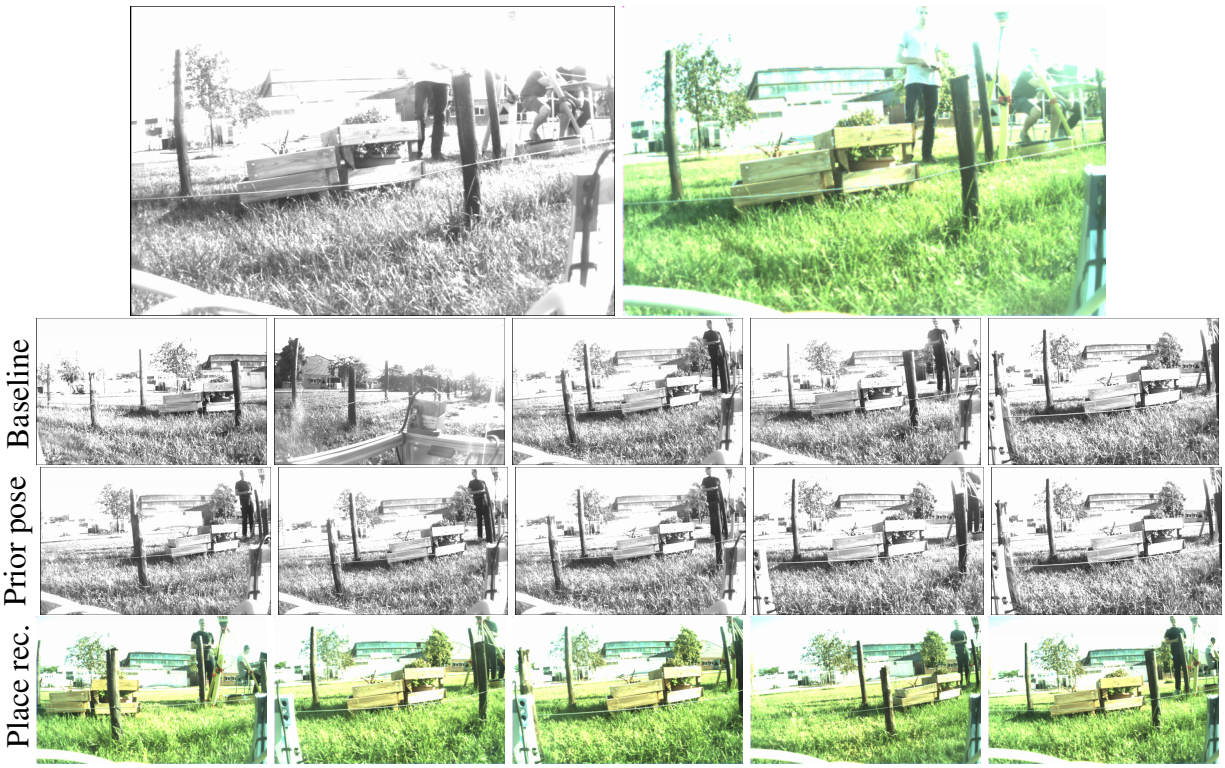


Figure 16: Qualitative example for the image retrieval. Top row shows the query images and the following three rows show the returned images from the baseline method (ETHZ), the pose prior (ETHZ) and the learned place recognition (RUG). The color images are used for the RUG place recognition algorithm. In this example each of the three methods return reasonable images. Only the baseline method returned one image that was not relevant.

	Single Image [%]			Generalized Image [%]		
Retrieval method	≥ 50	≥ 100	≥ 200	≥ 50	≥ 100	≥ 200
Baseline (ETHZ)	55.27	35.84	14.32	99.11	97.22	94.67
Pose prior (ETHZ)	76.11	53.54	24.01	99.83	99.83	99.57
Place Recognition (RUG)	30.43	15.74	6.08	96.44	92.56	87.33

Table 4: Comparison of different image retrieval methods. Table shows the percentage of retrieved images with at least k geometrically verified matches (for $k = 50,\ 100,\ 200$). The three right-most columns shows the statistics for the camera rigs instead of individual images, i.e. using the total number of matches found from all cameras.

the robot w.r.t. a particular bush, so it is not affected by any error in the pre-built map or its alignment with the sketchmap. In this section we evaluate the visual servoing which was jointly developed by ALUF and BOSCH.

The visual servoing uses DispNet [15] for 3D perception and DeepTAM [16] for visual tracking. To enable the evaluation, we mount a Velodyne on the robot platform which is considered as pseudo ground truth. We compare the estimated distance and angle of the robot relative to the target bush against the ones computed from the Velodyne point clouds. For the experiment, we collected 5 recordings in the Wageningen test garden where we manually drove the robot approaching a target bush and then circling around.

Figure 17 shows the estimated distance to the center of the bush as well as the angle of the bush w.r.t. the robot. The estimates from the visual servoing (using DeepTAM and DispNet) closely match the Velodyne-based ground truth. Figure 18 shows the average errors for the five experiments. Note that the ground truth values computed from the Velodyne pointclouds are not perfect. This can be seen in Figure 19 that shows the estimated bush size from the Velodyne measurements, which varies significantly throughout the sequence. Finally, in Figure 20 we show the errors for the first target position in each sequence which gives some idea about the localisation accuracy for the bush approach navigation.

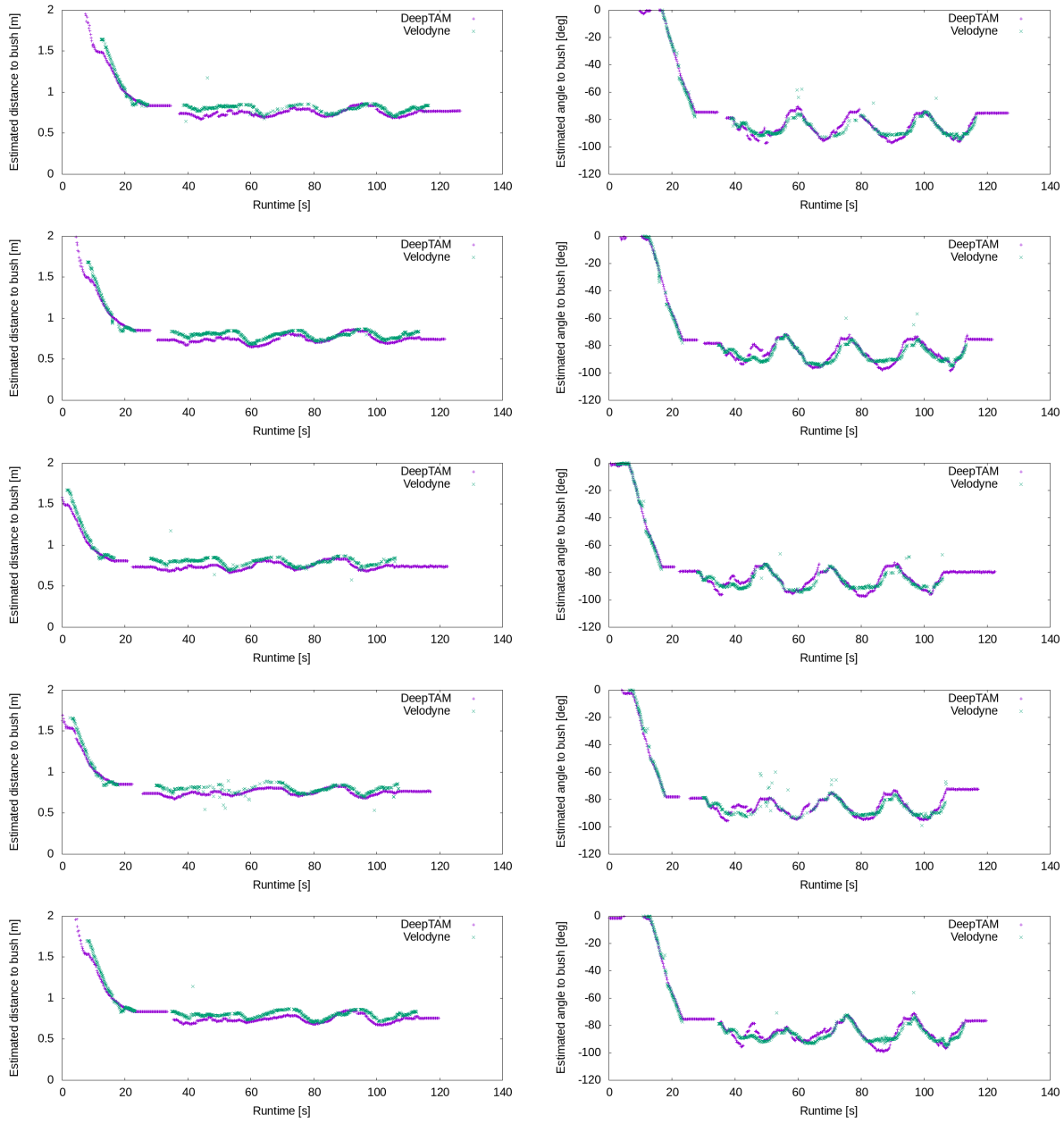


Figure 17: Estimated distance to bush-center (left) and angle robot-to-bush (right), using visual tracking and DispNet pointclouds (”DeepTAM”) vs using Velodyne pointclouds (”Velodyne”). Each row is a separate experiment run. Note that the ground truth measurements from the Velodyne pointclouds are not perfect and contain a few outliers.

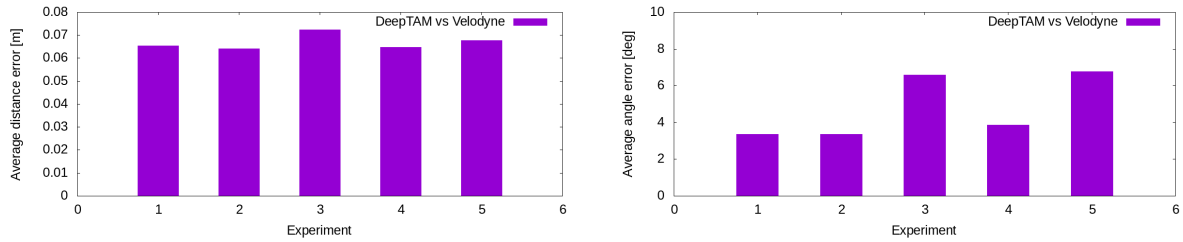


Figure 18: Average errors for bush distance and angle in the 5 experiments shown in Figure 17.

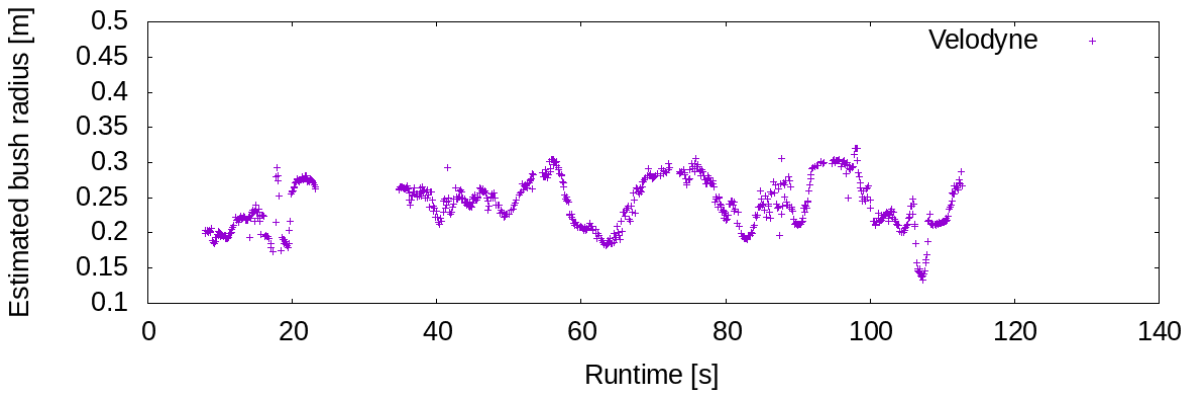


Figure 19: Variance in estimated bush radius when estimating from Velodyne pointclouds during approach (see Figure 17). Even with a high-precision LiDAR and a supposedly spherical bush, the bush's size cannot be reliably estimated from a single viewpoint.

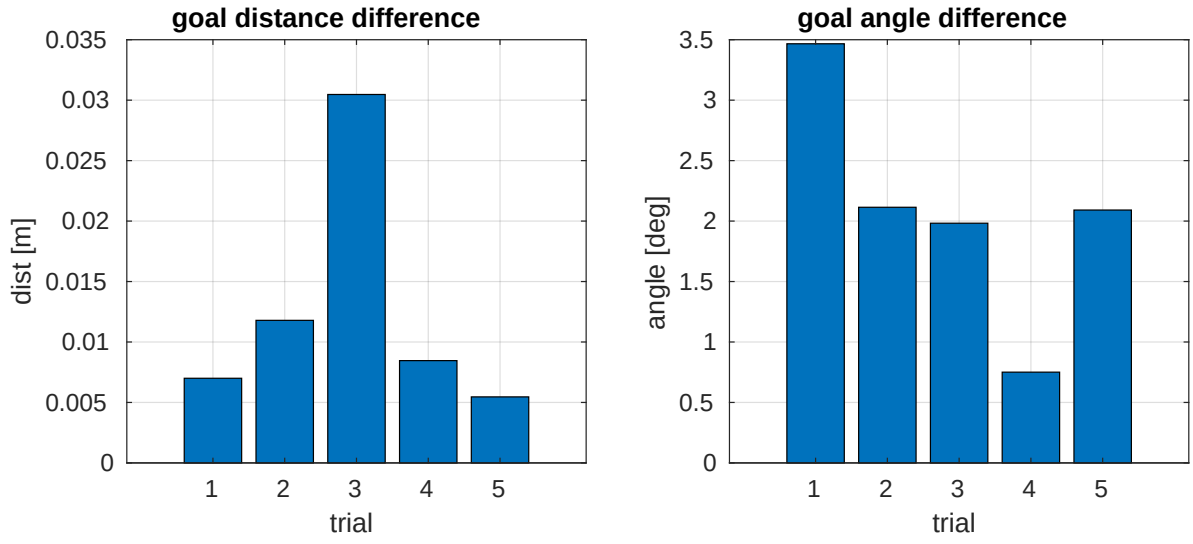


Figure 20: Difference of angle and distance for the first position target of Velodyne and DeepTAM. Mean values of 5 trials are 2.08 deg for angle and 0.0126 m for distance.

2.5 Evaluation of the Dense 3D Reconstructions

The task for dense 3D reconstruction in the project is to estimate the detailed shape of bushes for trimming and object detection.

The local dense reconstruction from stereo matching was successfully tested and integrated for both topiary and rose bush trimming (see *D2.6 - Final manipulator*), and instant obstacle detections (see *D4.3 - Drivable garden regions*).

The global 3D fusion and reconstruction was however not fully tested with the final prototype system, because its primary use case of extended object (hedge) trimming was not reached. Results on more recent data below give insights on how this could work in practice.

2.5.1 Evaluation of Stereo Matching Algorithms

For the final evaluation, we have included more variants of the algorithms developed in the project:

FPGA SGM (ETHZ) - onboard stereo with fixed 32 disparity levels.

DispNet (ALUF) - deep net stereo fine-tuned on garden data.

SDF-MAN (UEDIN) - learned fusion of the above two methods (fully *supervised* and *semi-supervised* variants).

MaxTree (RUG) - fast stereo running on CPU, with variants (*sparse*, *semi-dense* and *dense* versions)

Similar to popular stereo benchmarks, dense and sparse results are reported independently. This allows to examine performance of sparse methods that give disparity estimates only for some areas of the image. In this case, the trade-off between accuracy (error) and completeness (predicted area) is observed.

Quantitative dense evaluation in Figure 21 confirm previous results, where supervised fusion of DispNet and FPGA using SDF-MAN gives best performance with mean disparity error of 0.97 px.

Results for sparse evaluation in Figures 22 and 23 give the same conclusion, while providing further insight that the semi-supervised variant of SDF-MAN gives higher completeness (88%) compared to supervised case (78%), while keeping similar accuracy.

While FPGA SGM and MaxTree are deterministic methods, DispNet and SDF-MAN are CNN-based methods. The performance of CNN-based methods is influenced by the datasets on which they were trained and fine-tuned. CNN-based methods can only obtain accurate results on a test dataset when the datasets on which they were trained are of sufficient size, and these training datasets allow them to learn general concepts which can be applied to the test dataset. MaxTree, on the other hand, is a deterministic and almost non-parametric method, making it nearly plug-N-play.

CNN-based methods can only quickly process the large number of convolutions they are composed of when they are run on GPUs. GPUs cannot be easily used on embedded or power-constrained systems such as battery-powered robots or drones. MaxTree was designed to be run on CPUs, making it appropriate for applications in which limited computational and energy resources are available.

While each of the three variants of MaxTree provide less accurate disparity maps than the CNN-based methods, MaxTree is the best performing non-learning based method. The dense version of MaxTree (MaxTree HD) achieved very similar mean absolute error, yet much higher completeness than FPGA SGM. Furthermore, the semi-dense (MaxTree SD) and sparse (MaxTree S) version of MaxTree achieved lower mean absolute error than FPGA SGM, although completeness was lower.

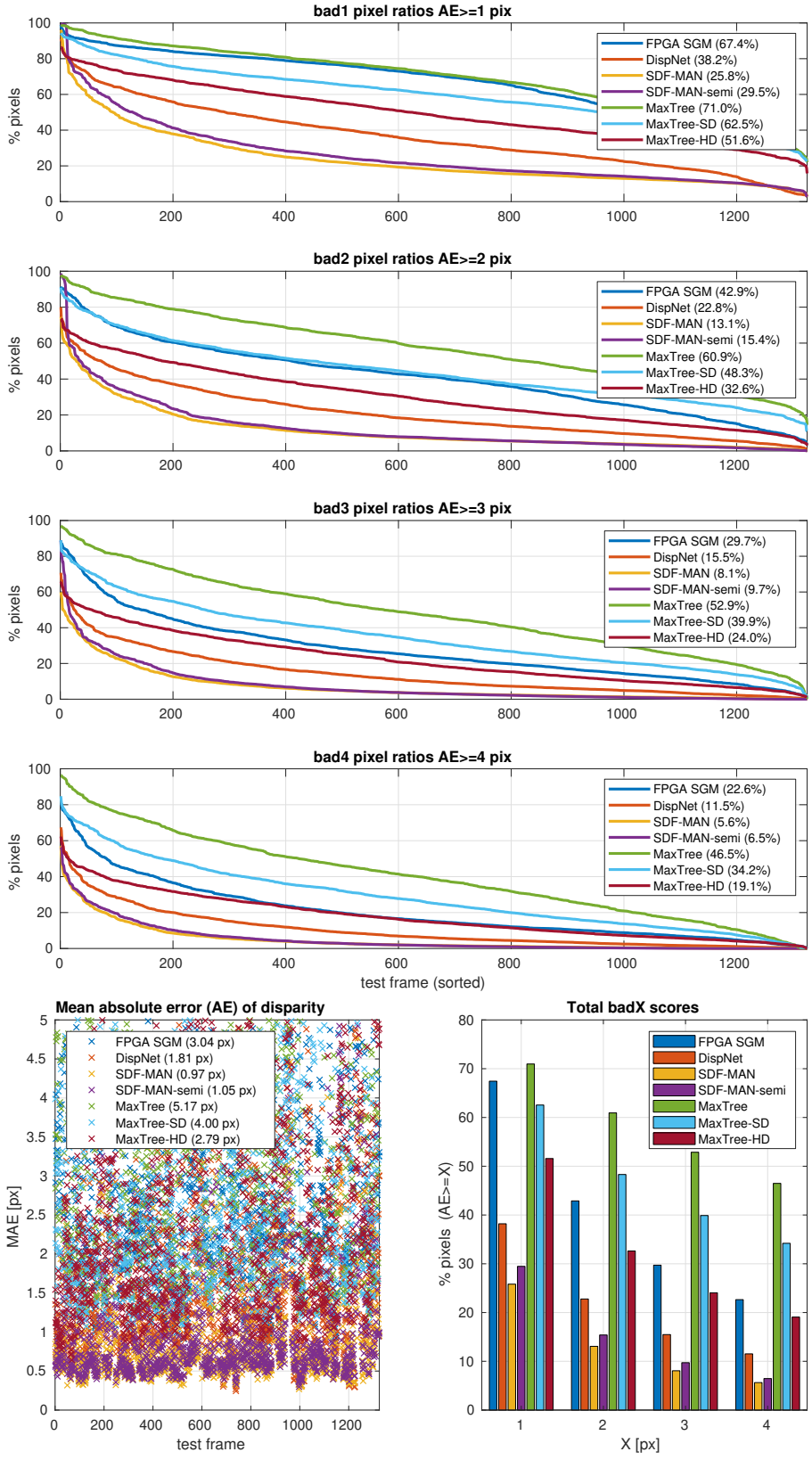


Figure 21: Dense comparison of stereo algorithms developed in the project on WUR2 dataset. All image pixels are included in the evaluation.

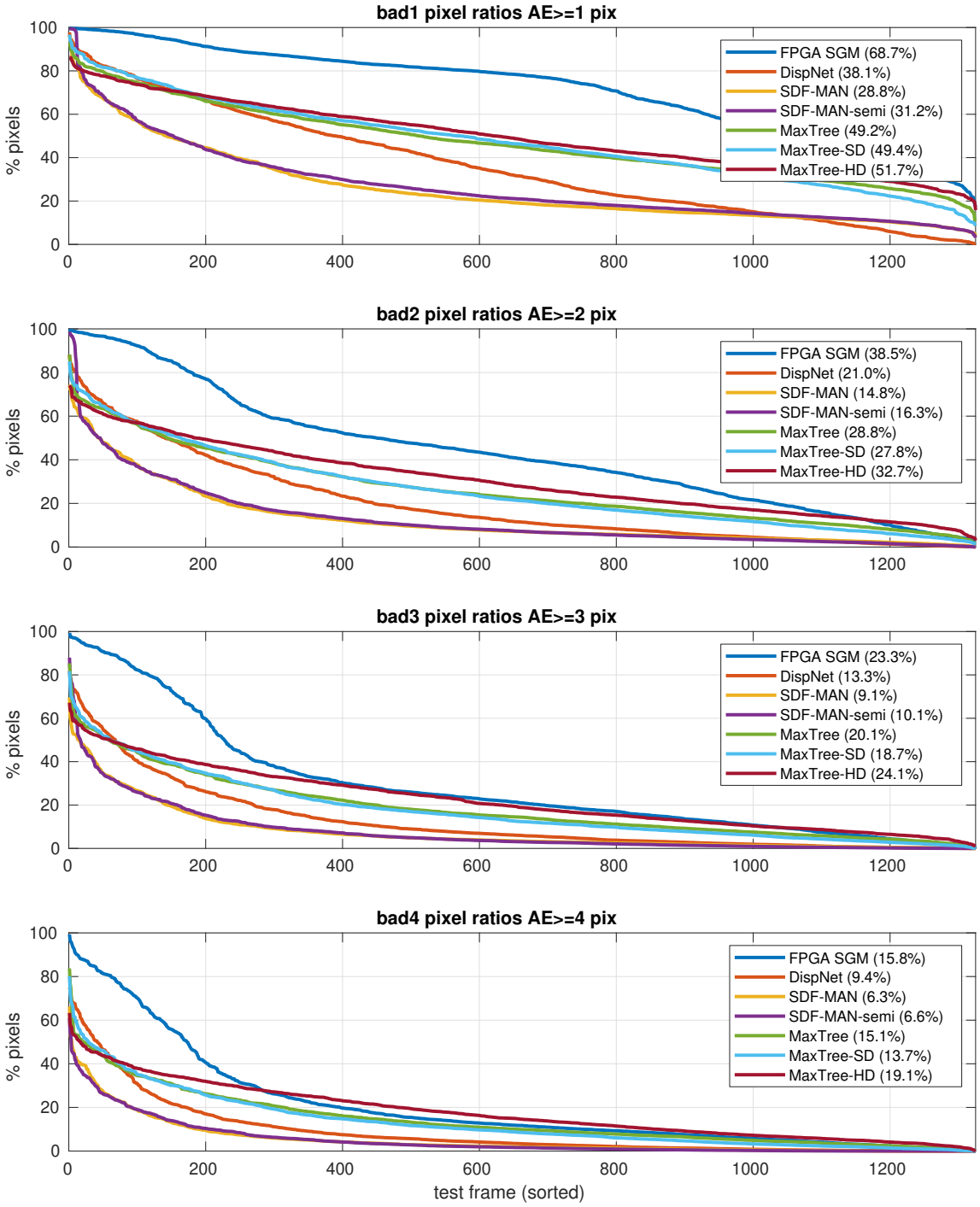


Figure 22: Sparse comparison of stereo algorithms developed in the project based on badX score on WUR2 dataset. Only foreground pixels that algorithm predicted are included.

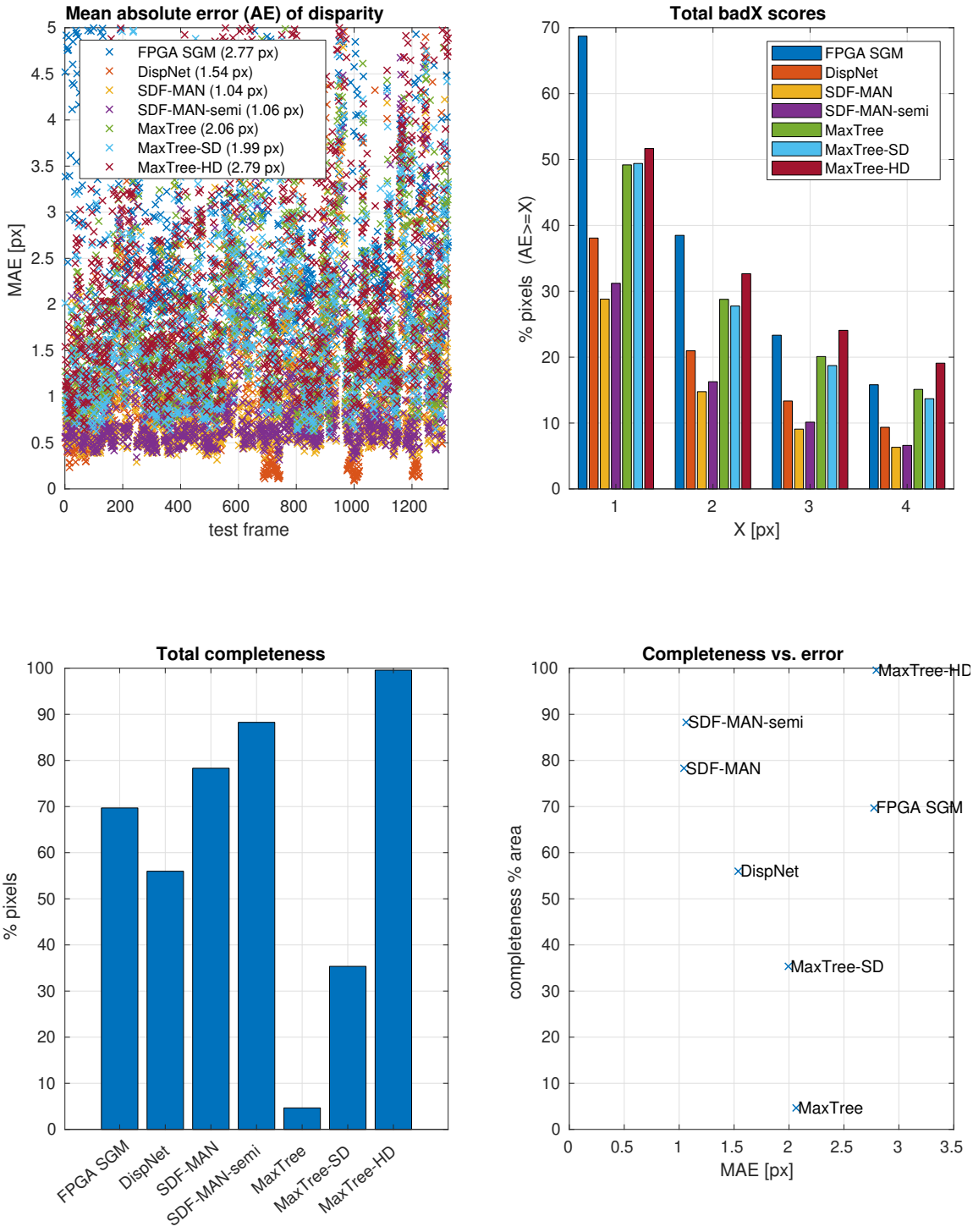


Figure 23: Sparse comparison of stereo algorithms developed in the project based on total error and completeness on WUR2 dataset.

2.5.2 Evaluation of local 3D fusion

We have investigated options to improve performance of local 3D fusion based on registration of point cloud using DUGMA. The results presented previously in D3.2 on WUR2 dataset showed a considerable number of failures in the case of large motions and lack of distinctive objects in the view of the single camera pair. Based on this analysis, we have implemented several improvements to the method initialisation and data pre-processing:

- Generate 360 point cloud inputs by merging 5 pairs of stereo vision cameras to increase number of objects in the view.
- Use semantic segmentation (based on D4.2) for different sampling of objects and ground. Objects are kept in full resolution but less informative ground points are down-sampled to reduce imbalance and run-time.
- Use improved visual SLAM pose for initialization (based on Section 2.1).

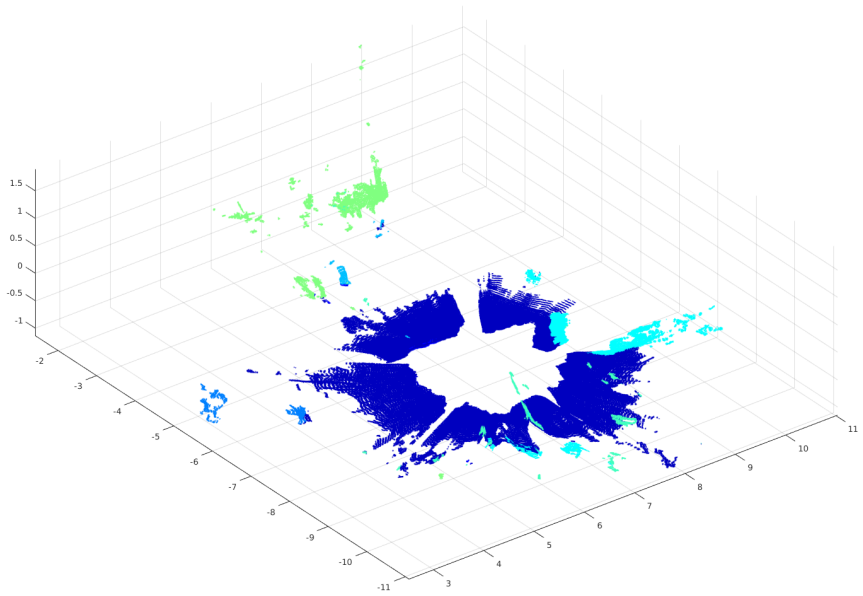
The performance of the improved pipeline was evaluated on WUR3 dataset scene 1 with 212 samples to test. Sampling rate for the ground (including grass and pavement) is 0.1%. The sampling rate for the remaining objects (bushes, trees) was 100%. See Figure 24 for examples.

Quantitative evaluation in Figures 25 and 26 shows that on average DUGMA registration refines the translation component of initial relative pose from SLAM by reducing its error by 50% (down to 5.8 cm from 11.4 cm). At the same time the rotation error is however increased by 35% (up to 1.46 deg from 1.08 deg), which is caused by 6 samples failing due to under-sampling. The rotation error on the remaining samples is about the same (i.e. unchanged).

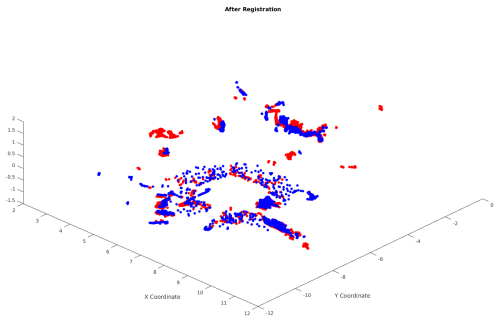
2.5.3 Evaluation of dense global 3D fusion

We have performed evaluation of global 3D fusion based on the final version of localisation with additional filtration of outliers. This step utilises the semi-dense version of the SLAM map to constrain the fully dense reconstruction as follows. Disparity result of SDF-MAN fusion is used to create local point clouds, which are transformed to global coordinates using camera poses provided by SLAM and merged in a global point cloud. The result is processed using statistical outlier filter and sub-sampled using a grid filter (grid size 10 mm). The result is shown in Figure 27a. The SLAM map is also filtered for outliers, see Figure 27b. Subsequently, the distance d between SLAM points and merged SDF points is calculated as shown in Figure 27c, and merged points that are further than $d_{max} = 2$ cm from the map are removed as outliers in Figure 27d.

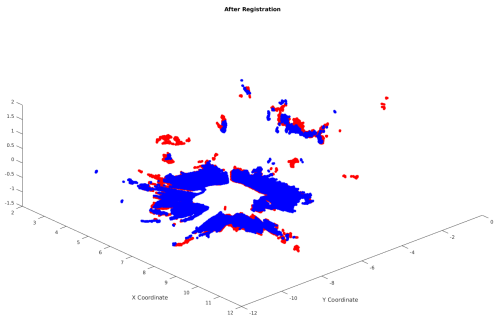
The evaluation was performed on two scenes from the WUR3 dataset. Qualitative results are shown in Figure 27 and Figure 28 suggest that the merged point cloud is cleaned up while the shapes of objects are preserved. Quantitative evaluation was not possible, because the dataset does not have corresponding dense ground truths.



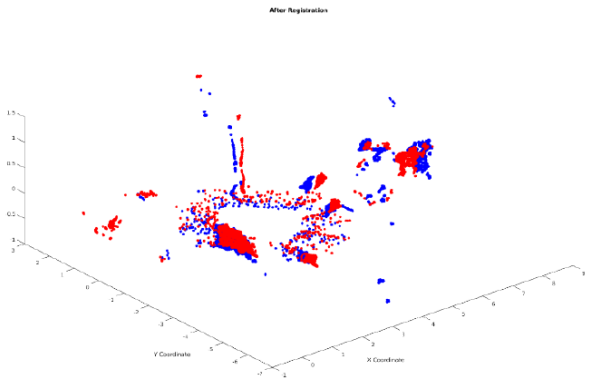
a) semantic segmentation of the point cloud (blue=ground, other=objects)



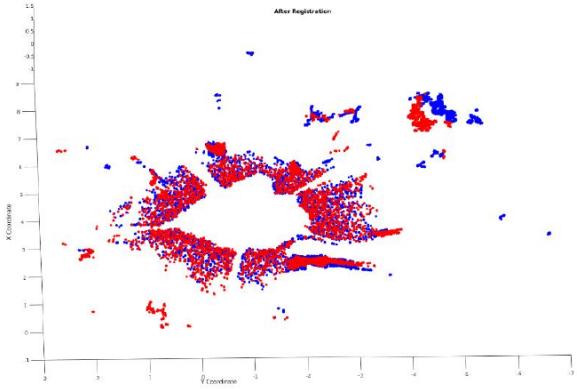
b) result with 0.1% ground points



c) result with 20% ground points



d) failure case with 0.1% ground points



e) success with 1% ground points

Figure 24: Results of DUGMA point cloud registration using different ground sampling rates for segmented ground (a). Number of ground points can be reduced without significant impact on the result (b,c). In some cases like (d) the resulting number of points is not sufficient and has to be increased (e).

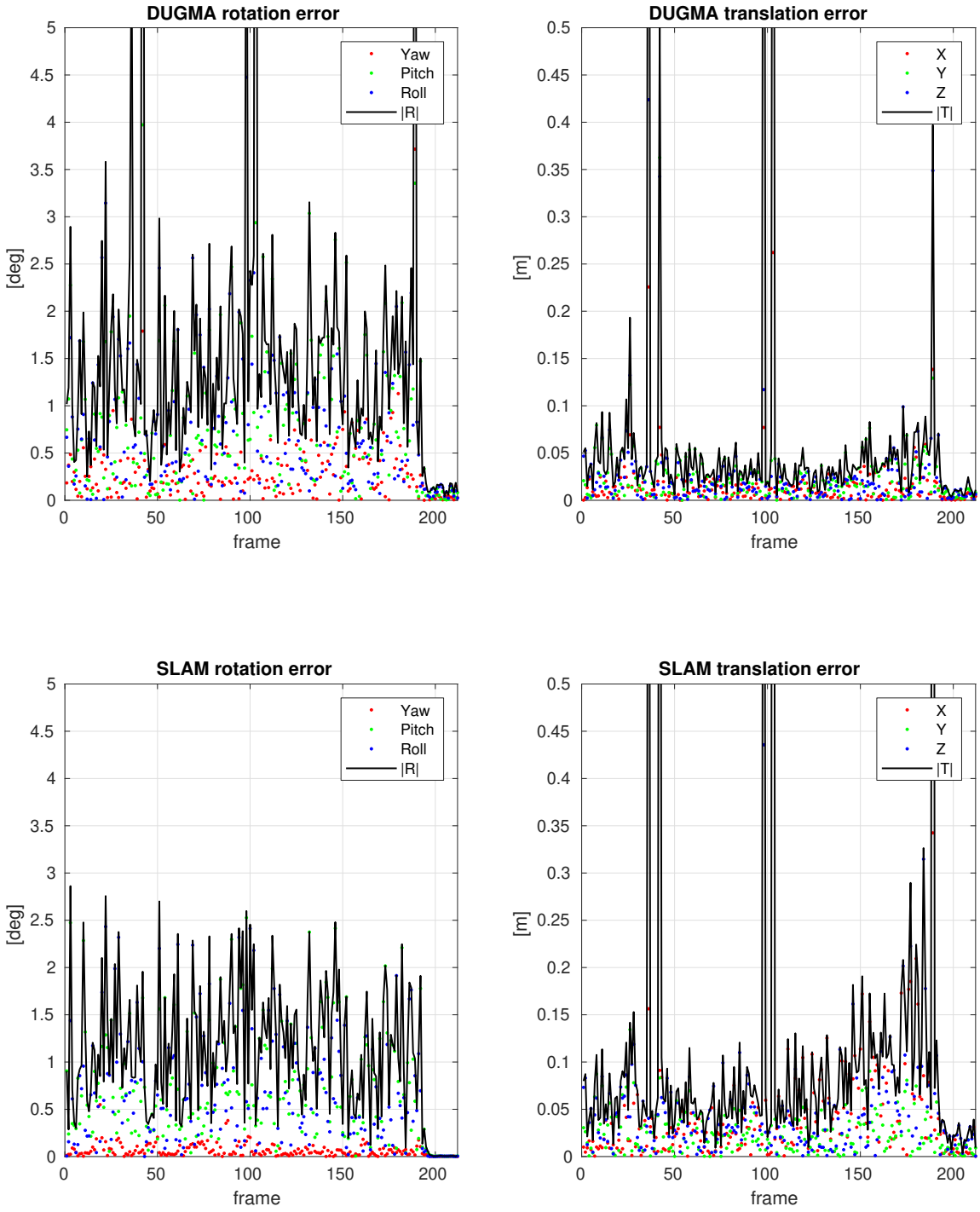


Figure 25: Results of multi-camera DUGMA registration on WUR3 dataset (s1). $|R|$ is minimal rotation angle and $|T|$ is translation norm (error distance).

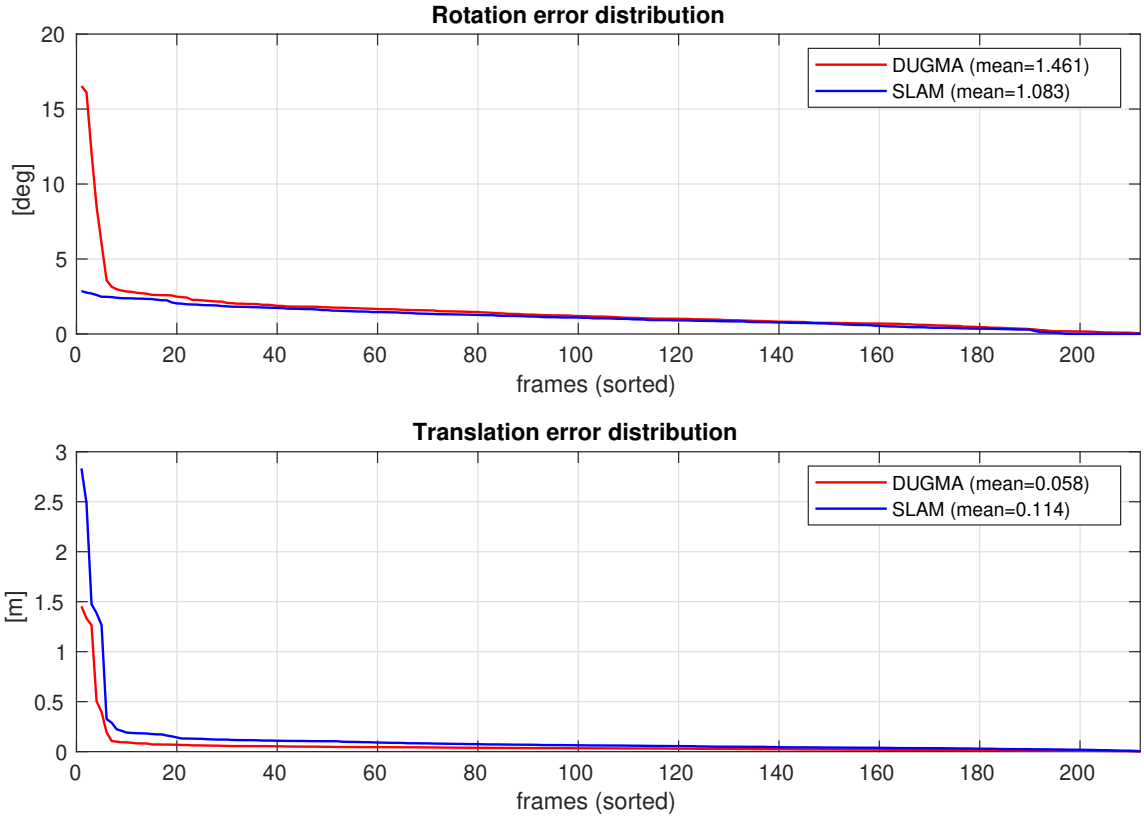
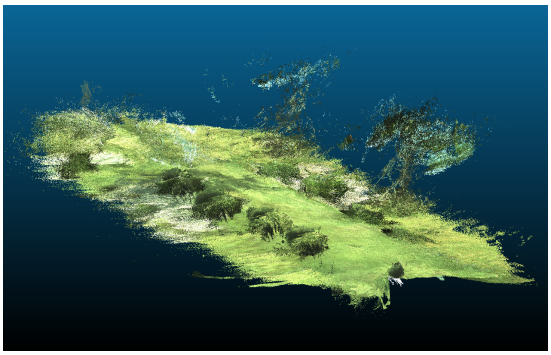
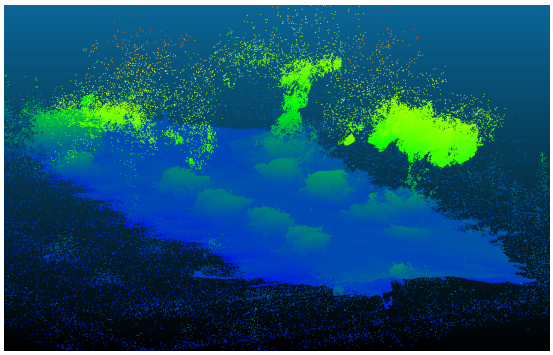


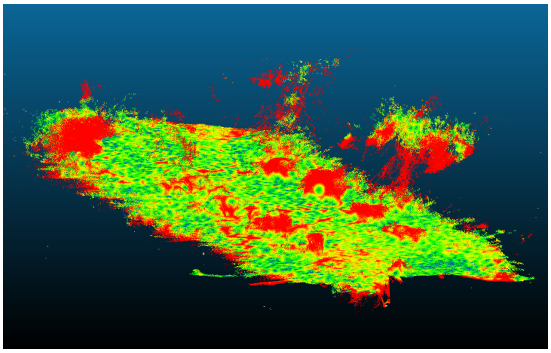
Figure 26: Results of multi-camera DUGMA registration on WUR3 dataset (s1).



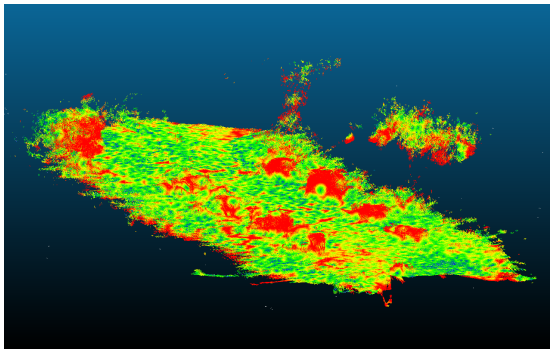
a) merged RGB points



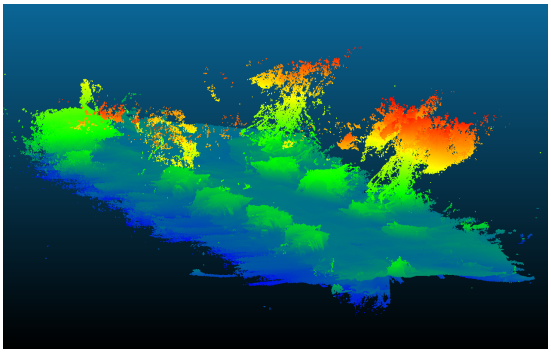
b) merged height-coloured with SLAM points



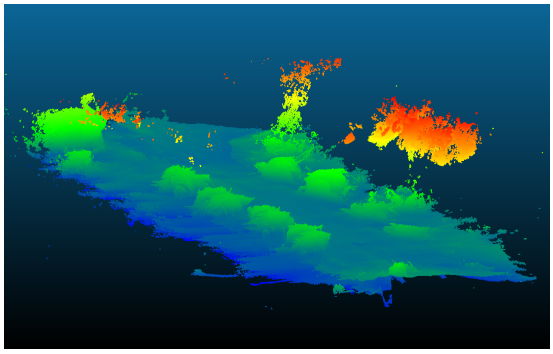
c) merged distance-coloured



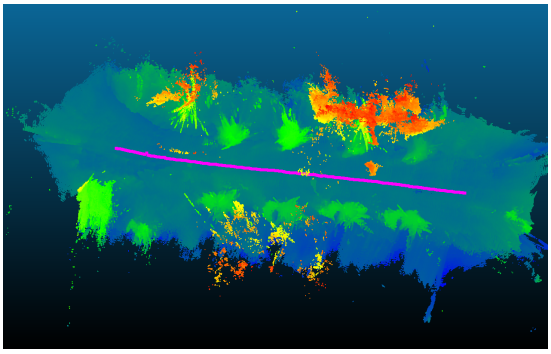
d) filtered distance-coloured



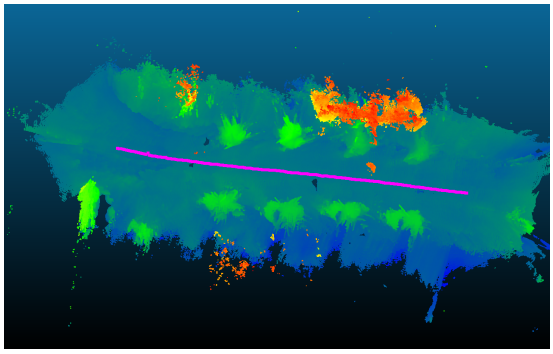
e) merged height-coloured



f) filtered height-coloured

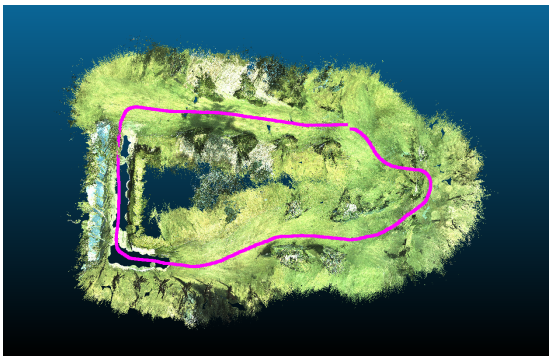


g) merged height-coloured with trajectory

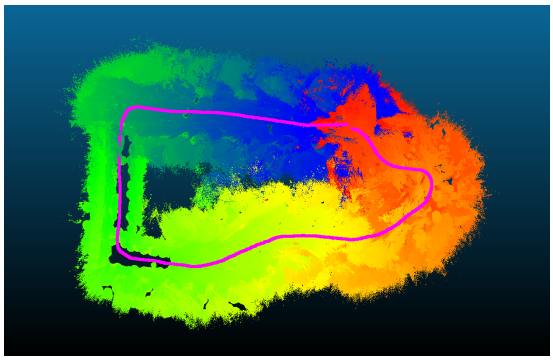


h) filtered height-coloured with trajectory

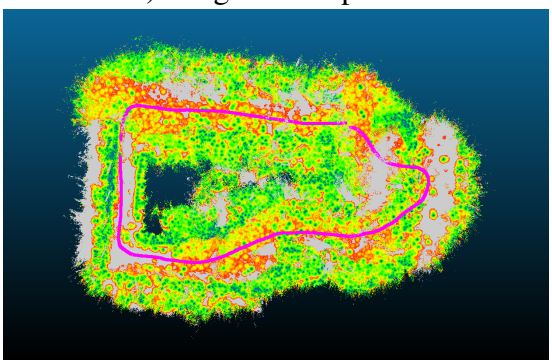
Figure 27: Dense global reconstruction of WUR3-s8 scene from GC-SLAM positions and fused disparity maps using SDF-MAN. Dense merged point cloud (left) is filtered using sparse map features (right) to reduce outliers based on distance (green=near, red=far).



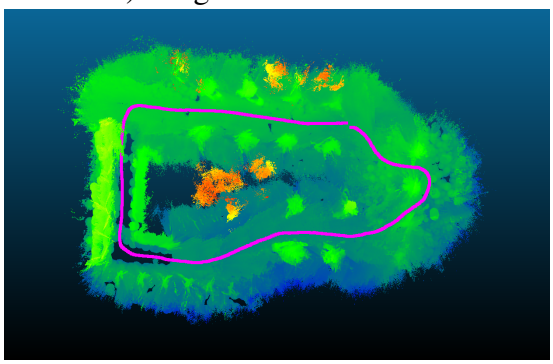
a) merged RGB points



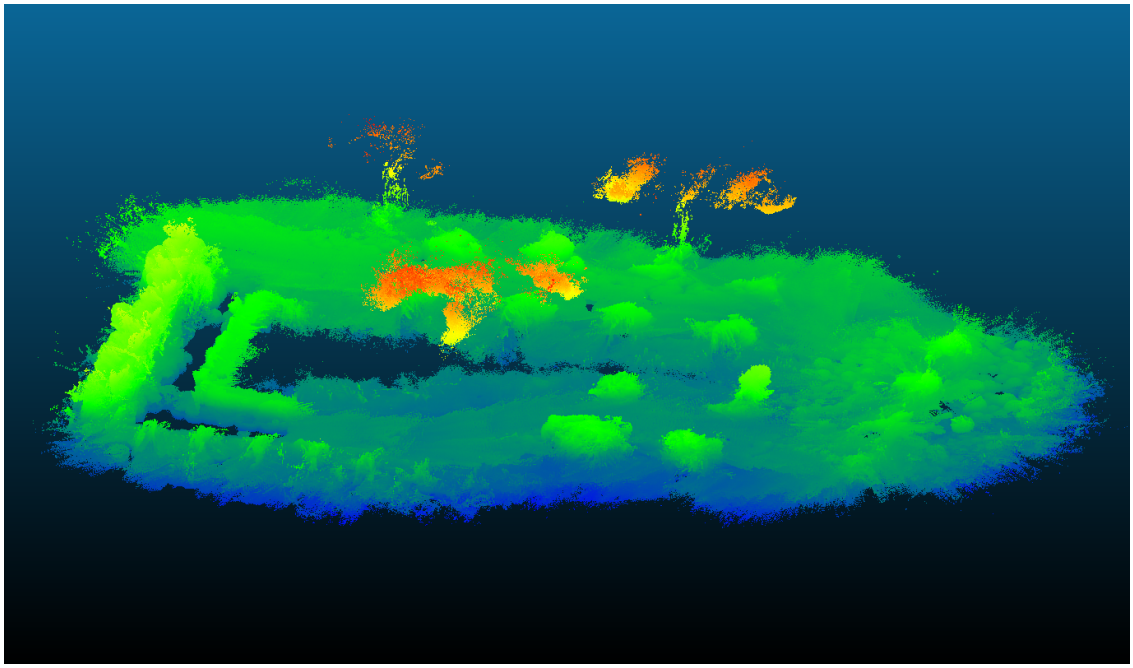
b) merged frame-coloured



c) merged distance-coloured



d) filtered height-coloured with trajectory



e) filtered height-coloured point cloud (side view)

Figure 28: Dense global reconstruction of WUR3-s8 scene from GC-SLAM positions and fused disparity maps using SDF-MAN. Dense merged point cloud (a) merged from multiple frames (b) is filtered using sparse map features to reduce outliers based on distance (c) into resulting point cloud (d,e). *Magenta = trajectory. Distance: green=near, red=far, grey=outlier.*

3 Conclusions

In this report, we have presented the efforts on the algorithmic speedup performed in the TrimBot2020 project. The work was focused on the DUGMA algorithm for which we showed significant improvement over the baseline version. The speed-ups came from avoiding unnecessary and duplicated computations, making the implementation more GPU-friendly and utilizing less precise floating point representations.

In the second part of the report, we presented the final evaluation of WP3. We have evaluated the main parts of the visual pipeline; visual localization, visual servoing and dense 3D reconstruction. We have shown, both in the evaluations above and the final demonstrator, that the methods developed in the project have reached satisfactory performance. Whenever applicable, we have compared the methods developed by different partners, e.g. image retrieval (ETHZ and RUG) and dense reconstruction (ETHZ, ALUF, UEDIN and RUG).

References

[1] Pu, C., Li, N., Tylecek, R., Fisher, B.: Dugma: Dynamic uncertainty-based gaussian mixture alignment. In: International Conference on 3D Vision (3DV). (2018)

[2] Bay, H., Tuytelaars, T., Van Gool, L.: Surf: Speeded up robust features. In: European Conference on Computer Vision (ECCV). (2006)

[3] Wu, C.: Siftgpu: A gpu implementation of scale invariant feature transform. `http://cs.unc.edu/~ccwu/siftgpu` (2011)

[4] Bradski, G.: The OpenCV Library. Dr. Dobb's Journal of Software Tools (2000)

[5] Kaljaca, D., Vroegindeweij, B., van Henten, E.: Coverage trajectory planning for a bush trimming robot arm. Journal of Field Robotics (2019)

[6] Baatz, G., Köser, K., Chen, D., Grzeszczuk, R., Pollefeys, M.: Leveraging 3d city models for rotation invariant place-of-interest recognition. International Journal of Computer Vision (IJCV) (2012)

[7] Schönberger, J.L., Frahm, J.M.: Structure-from-motion revisited. In: Conference on Computer Vision and Pattern Recognition (CVPR). (2016)

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

[9] Jegou, H., Douze, M., Schmid, C.: Hamming embedding and weak geometric consistency for large scale image search. In: European Conference on Computer Vision (ECCV). (2008)

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

[11] Leyva-Vallina, M., Strisciuglio, N., Petkov, N.: Place recognition in gardens by learning visual representations: data set and benchmark analysis. In: International Conference on Computer Analysis of Images and Patterns, Springer (2019) 324–335

[12] Leyva-Vallina, M., Strisciuglio, N., López-Antequera, M., Tylecek, R., Blaich, M., Petkov, N.: Tb-places: a data set for visual place recognition in garden environments. IEEE Access (2019)

[13] Arandjelovic, R., Gronat, P., Torii, A., Pajdla, T., Sivic, J.: Netvlad: Cnn architecture for weakly supervised place recognition. In: Conference on Computer Vision and Pattern Recognition (CVPR). (2016)

[14] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems (NIPS). (2012)

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

[16] Zhou, H., Ummenhofer, B., Brox, T.: Deeptam: Deep tracking and mapping. In: European Conference on Computer Vision (ECCV). (2018)

