TrimBot2020 Deliverable D3.1

Data representation design and implementation, sensor calibration

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

Abstract: This report describes the data structures for representing the fused 3D sensor data, hypothesized objects, and garden structures, including the software interfaces to these structures. In addition, the report describes and evaluates the sensor calibration, especially the intrinsic and extrinsic calibration of the multi-camera rig used in the project.

Deliverable due: Month 18

1 Introduction

This report consists of two parts. The first one (Section 2) details the data structures used to represent the results of fusing the 3D sensor data, including a description of its implementation. The second part of this report (Section 3) describes the calibration of the multi-camera system mounted on the chassis of the robot used in the Trimbot2020 project.

2 Data Structures for 3D Data Fusion

This section describes the data structures used for representing the fused 3D shape data, hypothesized objects, and garden structures. As the individual components are developed by different partners and since these individual components need to interact with each other, it was decided that each partner implements their component as a ROS (Robot Operating System) package. These packages communicate with each other by sending messages that contain the data they provide. As such, the data structures used to represent the different objects are implemented as messages in ROS. In the following, we give a brief overview of the components relevant for this deliverable and then discuss the individual ROS messages defined for this project. When possible, we utilize the standard messages pre-defined in ROS and augment them with additional customary messages.

2.1 Relevant Packages

The `Camera` package (developed by ETH Zurich) provides camera images together with the intrinsic calibration of the cameras. A FPGA is used to synchronize data capture for the 10 cameras mounted on the Trimbot platform (cf. Section 3). In addition, the FPGA also computes depth maps for the 5 stereo pairs that make up the multi-camera system.

The (semantic) `SLAM` package (developed by ETH Zurich) is responsible for simultaneously estimating a 3D map of the garden (in the form of a sparse point cloud) and the position of the robot with respect of this map. To this end, the images and calibration data provided by the `Camera` package are used. In addition, the package is responsible for localizing the robot with respect to a pre-built 3D map from a previous run. Besides publishing messages that describe the current poses of the multi-camera and vehicle frames as well as an odometry message, the `SLAM` package also provides the sparse point cloud, detected obstacles, and a grid for static obstacles. In addition, the `SLAM` package has a semantic component that uses semantic image information to support the SLAM system, e.g., by exploiting semantics to reduce drift, to semantically annotate the sparse point cloud, or to detect obstacles based on their semantics.

The `3D Data Processing` package (developed by the University of Edinburgh and the Albert-Ludwigs-Universität Freiburg) is responsible for processing the 3D sensor data generated in the project. It uses the camera images, calibrations, and depth maps from the `Camera` package and the sparse point cloud from the `SLAM` package as input and consists of multiple sub-components: The `projection` component generates a dense point cloud, including covariance estimates for the points. The `scene flow` component computes the scene flow from the camera images. The dense point cloud and the flow are then merged with the sparse SLAM point cloud in the `fusion` component, which generates a volumetric representation of the scene. The `surface extraction` component then extracts a 3D mesh from this volume.

Both the volumetric and mesh representations are published to other packages.

As part of its responsibilities, the `Garden Object Detection` package (developed by the University of Amsterdam) performs semantic segmentation and garden object detection, where the garden objects can consist of multiple prototypical objects (proto objects). These initial 2D detections are enriched by the 3D data computed in the other packages, e.g., a garden object can be associated with a 3D mesh.

2.2 Custom ROS Messages

In the following, we discuss the custom ROS messages defined for the Trimbot project and provide their definitions. The messages are ordered by package.

2.2.1 `Camera` Package Messages

The depth maps generated on the FPGA are stored and sent as disparity images, using the `DisparityImage.msg` message pre-defined in ROS (`http://docs.ros.org/api/stereo_msgs/html/msg/DisparityImage.html`).

2.2.2 `SLAM` Package Messages

Trimbot's SLAM system generates a sparse point cloud of the scene that is simultaneously used to represent the scene geometry and to estimate the pose of the robot in the scene. The point cloud is represented using the `PointCloud2.msg` message pre-defined in ROS (`http://docs.ros.org/api/sensor_msgs/html/msg/PointCloud2.html`). As the point cloud is generated by triangulating local image features during SLAM, each 3D point is also associated with two or more image features. The features extracted from an image are defined in the customary `Feature.msg` message shown below. The message contains the id (type) of features, the id of the image, the number of features found in this frame, the region of interest around each feature (defined by a 2D position, a scale, and an orientation), and the D-dimensional local descriptors associated with the features.

```
# Feature.msg
uint32 id          # feature set id
uint32 frame       # frame id
uint32 count       # number of 2D points
float32[] pos      # xy position [count 2]
float32[] scale    # roi radius [count]
float32[] orient   # roi orientation [count]
float32[] desc     # descriptors [count D]
```

In order to preserve the association between 3D points and features and enable adding proto objects (used by the semantic component of the SLAM system), the standard `PointCloud2.msg` is accompanied by a customary `SparseCloud.msg` message defined as

```
# SparseCloud.msg
uint32 seq           # header.seq of the PointCloud2 msg
uint32 id            # cloud id
uint32[] proto       # linked semantic proto ids
```



```
FeatureList[] flist # linked image feature id [count]
```

The `SparseCloud.msg` message refers to the corresponding standard point cloud and contains a list of proto objects (see Section 2.2.4 for details) and a list of features associated with the 3D points (`flist`). The latter list is represented by storing the number of linked features and a list of (image feature id, feature id) pairs per point, where the former number specifies the frame id of a `Features.msg` message and the latter defines an index into the set of features of this frame.

Besides the sparse 3D point cloud generated as part of the SLAM system, the `SLAM` package also provides an occupancy grid containing static obstacles. This is implemented using the pre-defined `OccupancyGrid.msg` message (`http://docs.ros.org/kinetic/api/nav_msgs/html/msg/OccupancyGrid.html`). In addition, the package can provide messages for individual object detections, where each obstacle (given with respect to the current pose of the robot frame) is represented by a `PointCloud2.msg`.

2.2.3 `3D Data Processing` Package Messages

The `projection` component takes the dense depth maps generated by the camera or the `scene flow` component and creates a dense 3D point cloud. This point cloud is again represented by a `PointCloud2.msg`. In order to enrich this representation, a customary `DenseCloud.msg` message is defined as

```
# DenseCloud.msg
uint32 seq        # header.seq of the PointCloud2 msg
uint32 id         # unique cloud id
uint32 source     # 1=depth map, 2=scene flow
uint32 from       # map or flow id
string covf       # covariance function spec
float32[C] covp   # covariance function params
```

The message stores the type of data source used to create the point cloud and the id of the corresponding raw data (`from`) together with covariance estimations for the 3D points. In order to remain flexible, the message defines the covariance function in the `covf` field.

The individual depth maps generated by the `projection` component are then fused into a single volumetric representation by the `fusion` component. Each voxel is represented by a 3D position, a normal, a RGB color, the probability of being occupied, and a signed distance value that gives an estimate for the distance to the closest surface. In addition, each voxel stores covariance matrix estimates for the position, normal, and color. The volumetric grid is then represented by a list of voxels, where each voxel can have a separate size (scale). This allows the use of hierarchical data structures such as octrees that take the scale of individual measurements into account during fusion. The resulting `Volume.msg` message is shown below. Notice that the volumetric fusion can also be linked to a set of proto objects, stored through their ids.

```
# Volume.msg
uint32 id         # volume id
uint32 count      # number of voxels
float32[] scale   # voxel size [count]
Point32[] pos     # xyz position [count]
```



```
float32[] pcov  # pos covariance [count 3 3]
Vector3[] norm  # unit normal [count]
float32[] ncov  # norm covariance [count 3 3]
Vector3[] color # rgb color [count]
float32[] ccov  # color covariance [count 3 3]
float32[] occ   # occupancy probability [count]
float32[] dist  # signed dist. to surface along normal [count]
uint32[] proto  # linked semantic proto ids [count]
```

Given the volumetric representation defined above, the `surface extraction` component generates a 3D surface mesh, for example through applying a variant of the Marching Cubes algorithm. This triangle mesh is represented using the pre-defined `Mesh.msg` message provided by ROS (`http://docs.ros.org/kinetic/api/shape_msgs/html/msg/Mesh.html`). In order to include additional measurements such as an estimate of the covariance at each vertex of the mesh, triangle colors, or semantic labels, the custom `MeshEx.msg` shown below is used, where `V` is the number of mesh vertices and `T` is the number of mesh triangles.

```
# MeshEx.msg
uint32 id           # mesh id
float32[] vcov      # vertex position covariance [V 3 3]
Vector3[] tcolor    # rgb face colors [T]
uint32[] tlabel     # optional face label [T]
uint32[] tshape     # optional face shape [T]
```

2.2.4 `Garden Object Detection` Package Messages

The `Garden Object Detection` package provides 2D image detections of objects. These detections are represented through a semantic image segmentation that consists of a set of segments. Each segment is defined as a set of pixels with an associated semantic label and a shape type. The resulting `Segments.msg` message is defined as

```
# Segments.msg
uint32 id          # segmentation id
uint32 frame       # image frame id
uint32 count       # number of segments
uint32[] seg       # pixelwise segment map [h w]
uint32[] label     # labels for segments [count]
uint32[] shape     # shape types for segments [count]
```

Using the 3D information generated by the other packages, 2D image detections (potentially from multiple images) can be upgraded to so-called *proto objects*. These objects reference their object type (in the form of a semantic label) and shape type, as well as lists of corresponding 2D segments and 3D surface meshes. In addition, each proto object has a 3D position and orientation and defines a bounding box for the object. In addition, the resulting `ProtoObject.msg` message also contains a volumetric representation for the object:

```
# ProtoObject.msg
```



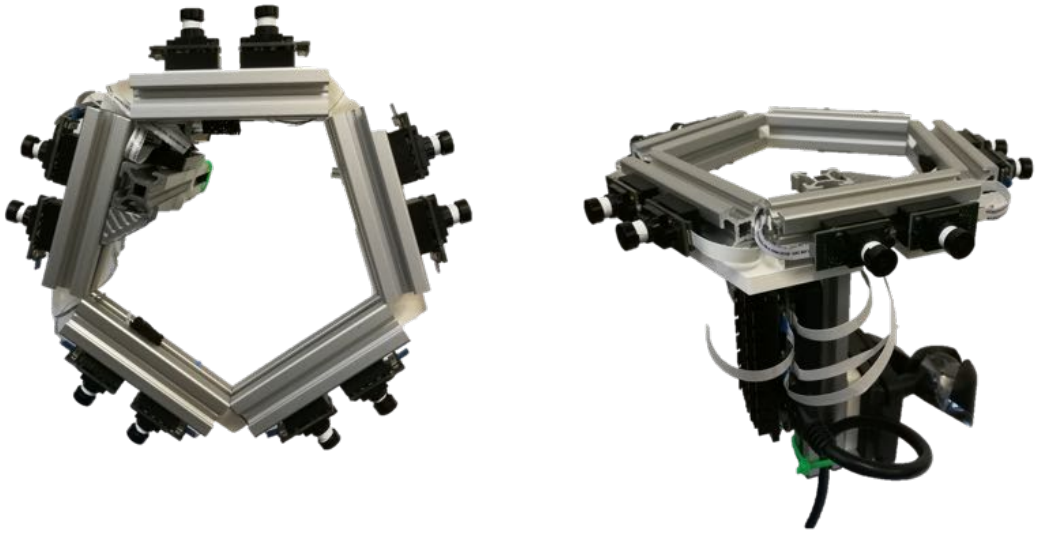
```
uint32 id          # proto id
uint32 label       # object type
uint32 shape       # shape type
uint32[] seg       # image segment ids(1) and idx(2) [N 2]
uint32[] mesh      # surface mesh ids(1) and face idx(2) [M 2]
Point32 pos        # xyz position
Vector3 orient     # orientation vector
Vector3 box        # bounding box size
uint32[3] dim      # grid dimensions [H,W,D]
bool[] occup       # occupancy grid [H W D]
```

The proto objects can be used to construct more complex *garden objects*, which consist of multiple proto objects (and potentially a detailed geometric representation in the form of a triangle mesh). In addition, the `GardenObject.msg` message also contains the position and orientation of the object, as well as a bounding box and a volumetric representation:

```
# GardenObject.msg
uint32 id          # garden object id
uint32 map         # map object id
uint32[] proto     # proto object ids
Point32 pos        # xyz global position
Vector3 orient     # orientation vector
Vector3 box        # bounding box size
uint32[3] dim      # grid dimensions [H,W,D]
bool[] occup       # occupancy grid [H W D]
Mesh geometry      # optional detailed geometry
```

3 Sensor Calibration

There are 10 cameras mounted on the Trimbot platform. Each camera features an image sensor and an inertial measurement unit (IMU). The cameras are arranged in pairs for stereo vision, the left camera is a color camera, the right is greyscale camera. However the color and greyscale cameras are fully compatible and can easily be replaced to support a uniform 10 color camera or 10 greyscale camera setup. The color cameras are slightly less light sensitive than the greyscale cameras and have less spatial resolution due to the bayer filter. The current setup with mixed pairs allows to take advantage of capturing color and full spatial resolution images. The five pairs are mounted in a pentagon shape and create together a 360 degree field of view.
A camera calibration contains the camera intrinsic parameters, which are lens and image sensor specific values as well as extrinsic parameters, that describe the position and orientation of a camera. The intrinsic camera parameters are similar among all cameras as all of them are equipped with the same lens but due to small production inaccuracies, slightly different positions of the image sensor relative to the lens and other mechanical reasons, the intrinsic values are not the same. A calibration routine therefore needs to estimate intrinsic parameters for all cameras individually. Figure 1 shows the camera configuration with five stereo pairs in a pentagon shape.



(a) (b)

Figure 1: Top view in (a) and side view in (b) of the 10 camera setup. The cameras are mounted on an aluminium profile to maintain a rigid camera setup. The 10 cameras are forming 5 stereo pairs that are arranged in a pentagon setup. The baseline between the left and right camera of a stereo pair is 30mm.

The cameras are hardware synchronized and provide images at up to 12 frames per second each. Figure 2 shows all 10 images of the pentagon setup in a garden scene. The top row corresponds to the images of the 5 left color cameras. The bottom row corresponds to the images of the 5 right greyscale cameras. There is an overlap in the field of view of neighbouring cameras as the field of view is 82 degrees for the individual cameras while the pentagon shape has only 72 degrees corners.

3.1 Calibration Parameters

A lens model that takes into account radial and tangential distortion parameters as described in [1] is used to model the lens distortion effects. The complete set of intrinsic parameters is given in Table 1.
The camera intrinsic parameters focal length $[f_{cx}, f_{cy}]$ and principal point $[c_{cx}, c_{cy}]$ are measured in pixels. The radial distortion parameters $[\kappa_1, \kappa_2]$ and tangential distortion parameters $[\xi_1, \xi_2]$ are dimensionless values.
The extrinsic parameters include a three dimensional translation vector $\mathbf{t}$ in meters and a three dimensional rotation matrix $\mathbf{R}$ in right-handed coordinates[1].
Figure 3a shows an example raw image of a window, especially in the corners the image is affected by distortion effects of the lens. In Figure 3b the image is corrected based on the radial

[1] All coordinate systems used in this report are right-handed.

Figure 2: Panoramic view composed of the five individual stereo camera views from the pentagon setup. Top row shows the five left color images, bottom row the five right greyscale images. There is an overlap in the field of view between neighbouring cameras. All images are subject to lens distortion effects.

(a) (b)

Figure 3: (a) Raw image of a single camera from the pentagon setup. The distortion effects of the lens are visible as the window frame is curved. (b) The corrected image with respect to the lens distortion based on the obtained intrinsic and extrinsic camera parameters.

tangential lens model.

3.2 Offline Calibration via Kalibr

Kalibr is a toolbox that can estimate intrinsic and extrinsic calibration parameters of a multi-camera system with non-globally shared overlapping fields of view. The toolbox is available under BSD license, implementation details are available in the corresponding publications [2, 3, 4]. Three different calibration targets are supported, a checkerboard target, a circle grid target, and an Aprilgrid target. The Aprilgrid target is recommented as it allows to use also partially visible targets for calibration and the pose of the target is fully resolved, eliminating any problems with flips. An example of an Aprilgrid target is shown in Figure 4. There are 36 AprilTags arranged in a 6x6 2D array.

To obtain calibration images, the camera system is fixed and the calibration target is moved in front of the cameras. Figure 5 shows the Trimbot prototype with the pentagon camera setup in front of the calibration target that is carried around the prototype. A ROS bag containing the

Name	Symbol	Unit
Focal Length	f_{cx} f_{cy}	[pixel] [pixel]
Principal Point	c_{cx} c_{cy}	[pixel] [pixel]
Radial Distortion	κ_1 κ_2	[1] [1]
Tangential Distortion	ξ_1 ξ_2	[1] [1]

Table 1: Intrinsic parameters for the pinhole camera model with radial tangential distortion. Focal length and principal points are measured in pixels, the distortion parameters are dimensionless values.

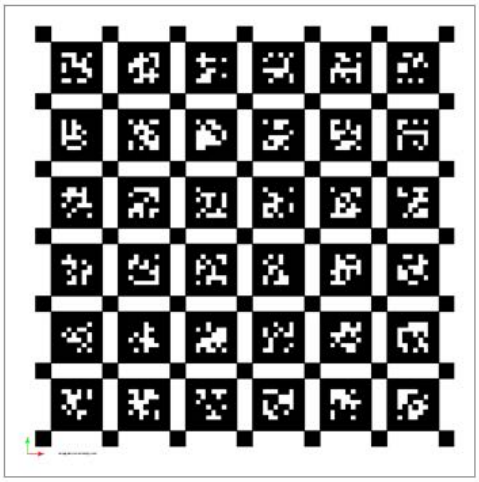


Figure 4: Calibration target pattern. A 2D array of 36 AprilTags forms the 6x6 calibration target.

images is the preferred input for the Kalibr toolbox. Kalibr first detects and extracts the position of the calibration targets in the images and uses the extracted positions to estimate the intrinsic calibration values. In a second step, the position of the calibration target is used to estimate the extrinsic parameters among views with overlapping field of view.

The calibration values are written to a .yaml file including the camera model, intrisic and extrinsic parameters, distortion model, distortion coefficients, resolution and the ROS topic name the calibration was created out of. Listing 1 shows the structure of an example .yaml calibration file for a simplified setup of only two cameras. For validation there is a PDF report available containing plots with reprojection errors for all the camera views.

```
cam0:
  cam_overlaps: [1, 2, 3, 8, 9]
  camera_model: pinhole
  distortion_coeffs: [-0.35995268092519617, 0.11030211900967231, -0.0006317626993103383,
    -0.0010628642396034863]
  distortion_model: radtan
  intrinsics: [538.3498428066779, 538.5608816406484, 388.69387892103344, 225.98034433739815]
  resolution: [752, 480]
```

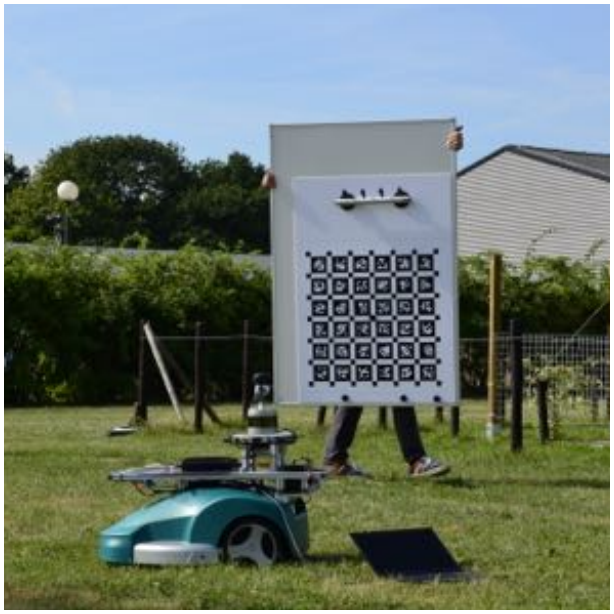



Figure 5: Calibration bag recording. The calibration board is moved in front of the cameras to obtain calibration images.

```
  rostopic: /uvc_camera/cam_0/image_raw
cam1:
  T_cn_cnm1:
  - [0.9999646166488014, -0.0027179038177023314, -0.007961058298851202, -0.031705878626870 98]
  - [0.0027635673748502222, 0.9999797618755782, 0.005730491655251141, -0.0004948946690760341]
  - [0.007945322256815766, -0.005752289812236394, 0.9999518903507988, -0.0005098018039522468]
  - [0.0, 0.0, 0.0, 1.0]
  cam_overlaps: [0, 2, 3, 8, 9]
  camera_model: pinhole
  distortion_coeffs: [-0.35609294506885897, 0.10647325852719773, -0.0010914026444042012,
    -0.0010556456670249045]
  distortion_model: radtan
  intrinsics: [541.6460202751782, 541.9113267065561, 362.16127043794364, 231.3916809318506]
  resolution: [752, 480]
  rostopic: /uvc_camera/cam_1/image_raw
```

Listing 1: Calibration values in .yaml file format

Additional to the camera calibration tool, Kalibr also provides a calibration tool to get the spatial and temporal parameters between camera and IMU. To obtain camera-IMU calibration images and calibration values, the calibration pattern is fixed and the camera-IMU system is moved in front of the Aprilgrid target to excite all IMU axes. Good illumination to keep the shutter times low to avoid motion blur is crucial. Details about the camera-IMU calibration method are desribed in [2].

3.3 Online Calibration

The offline calibration of the intrinsics and extrinsics of our camera array (see Figure 1) is highly accurate but requires a specialized calibration pattern. As such, this step would be carried out once before deploying the robot for operation. However, even though the camera array is rigidly mounted, external forces, such as vibrations, temperature changes, etc., can significantly alter the intrinsic and extrinsic parameters of the setup. Since the entire operational functionality of

the robot relies on an accurate localization within the map, which itself relies on an accurate calibration, it is therefore essential to maintain an accurate calibration of the camera parameters during operation.
In addition, we strive to avoid the tedious offline calibration process before each deployment of the robot. Therefore, we developed a system that continuously refines and updates its camera calibration parameters to account for small variations during operation [5]. Furthermore, our system can self-calibrate the camera array extrinsics from scratch in order to account for bigger variations. As an ongoing effort, we are currently also extending our system to support self-calibration of the intrinsic parameters, thereby superseding the need for offline calibration entirely. In the following, we first describe the technical details of the online calibration system and then evaluate our approach using real-world datasets.

3.3.1 Calibration Problem

Given the intrinsic and extrinsic parameters $\mathbf{p}_{C_l}$ of camera C_l from the offline calibration, the goal is to continuously refine these parameters during operation to account for small variations caused by external physical forces. While the robot is in operation, our system continuously estimates its location $\mathbf{t}_{I_i}$ and orientation $\mathbf{R}_{I_i}$ for every camera frame I_i and builds a local map of its environment using the sparse points $\mathbf{X}_k$. This problem is called Simultaneous Localization and Mapping (SLAM) [6] and, in order to give context for calibration, we formulate this problem as a joint optimization problem minimizing the reprojection error

$$e = \sum_{i,j} \rho \left(\left\| \mathbf{x}_j^{I_i} - \pi \left(P(\mathbf{X}_k, \mathbf{R}_{I_i}, \mathbf{t}_{I_i}), \mathbf{p}_{C_l} \right) \right\|^2 \right) \ , \tag{1}$$

which is typically called the bundle adjustment problem [7]. Here, we sum over all image observations j in all camera frames i, while ρ denotes a loss function (e.g., the L_2 loss or the robust Huber loss), $\mathbf{x}_j^{I_i}$ denotes an image observation of a 3D point $\mathbf{X}_k$, and π defines the projection function from world to camera coordinates given the intrinsic and extrinsic calibration $\mathbf{p}_{C_l}$ of the camera array as well as its location $\mathbf{t}_{I_i}$ and orientation $\mathbf{R}_{I_i}$ in the world. The typical SLAM problem optimizes this objective function over the location and map parameters $\{\mathbf{X}_k, \mathbf{R}_{I_i}, \mathbf{t}_{I_i}\}$ and keeps the calibration parameters $\mathbf{p}_{C_l}$ constant, as initialized by an offline calibration routine.

3.3.2 Continuous Refinement

In order to continuously refine the calibration parameters, we therefore also optimize over the intrinsic and extrinsic calibration parameters $\mathbf{p}_{C_l}$ in SLAM. We optimize the objective function using the Levenberg-Marquardt algorithm and use preconditioned conjugate gradients as an iterative method to solve the system of equations. To ensure a well-conditioned system for calibration, we only optimize over the calibration parameters if we detect a reliable geometric configuration with enough constraints. A reliable configuration is typically achieved when every camera has common observations from sufficiently different viewpoints with many other cameras in the array, leading to a relatively dense system of equations. For this configuration, the robot must both rotate and change its position. The optimization approach requires a good initialization in order to converge and we use standard two-view Structure-from-Motion techniques to obtain initial estimates for $\{\mathbf{X}_k, \mathbf{R}_{I_i}, \mathbf{t}_{I_i}\}$. To initialize the intrinsic and extrinsic calibration parameters $\mathbf{p}_{C_l}$, we use the values obtained from the offline calibration, as described in the previous section.

3.3.3 Extrinsic Self-Calibration

This section describes how we can self-calibrate the extrinsic parameters from scratch without any prior information. While the continuous calibration refinement operates online, it requires good initialization which is either tedious due to the offline calibration routine or is not accurate enough due to larger changes to the camera array configuration caused by external physical forces. In this section, we describe a system to self-calibrate the extrinsic parameters from scratch to overcome these limitations. Our method requires the robot to move for a short distance in order to obtain enough constraints for self-calibration before initializing the SLAM pipeline. In contrast to the calibration refinement in SLAM, which uses an incremental Structure-from-Motion approach, we parameterize a global Structure-from-Motion approach to incorporate the rigid extrinsic configuration of the camera array.
In global Structure-from-Motion, one first estimates the relative transformations $\{\mathbf{R}_{I_{ij}}, \mathbf{t}_{I_{ij}}\}$ between image I_i and image I_j. These pairwise transformations are estimated using projective geometry and define a viewing graph (see Figure 6) with images as vertices and transformations as edges. We estimate a homography for pure camera rotation or unconstrained motion in planar scenes and an essential matrix for unconstrained motion in non-planar scenes. Global Structure-from-Motion then aims to assign a global position $\{\mathbf{R}_{I_i}, \mathbf{t}_{I_i}\}$ to each image. First, one typically solves for globally consistent rotations $\mathbf{R}_{I_i}$ using robust rotation averaging [8]. In the second stage, one solves for globally consistent translations [9] followed by triangulation of points $\mathbf{X}_k$ and integration of point-to-camera constraints [10]. In the last stage, one usually uses the global Structure-from-Motion results as an initialization for bundle adjustment as a refinement. We follow the standard approach and, to estimate the extrinsic calibration of the camera array, we build the pairwise transformation graph over multiple consecutive frames at different time steps. In the optimization, we then constrain the relative transformations to be equal between the same cameras in the camera array at different times (see Figure 7). This modified parameterization of global Structure-from-Motion allows us to obtain the relative rigid geometry of the camera array. Similar to the continuous refinement of the calibration, we only self-calibrate the extrinsics once the viewing graph is well-conditioned. In our experiments, this is usually the case after a small translational and rotational motion of the robot, which provides enough multi-view constraints for the optimization. Furthermore, since the initial set of extrinsic parameters is likely to be inaccurate, the triangulation of points is inaccurate or fails for some points completely. Hence, we perform bundle adjustment and triangulation in an alternating manner until convergence. In the next section, we evaluate the accuracy of our calibration approach.

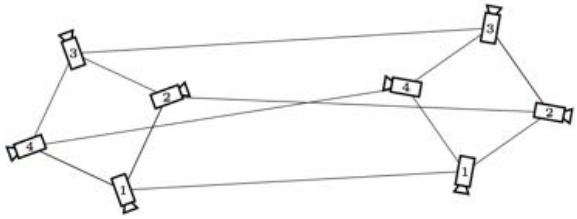


Figure 6: Viewing graph with camera images as vertices and relative transformations as edges.

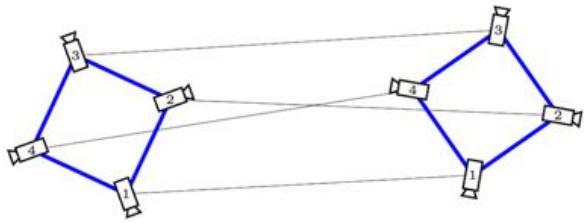


Figure 7: Viewing graph augmented with rigid camera constraints. We enforce the relative transformations between cameras in the array (blue) and between time steps (gray) to be consistent across time since the cameras are rigidly mounted.

3.4 Evaluation

Continuous Refinement We evaluate the accuracy of our calibration routine using the dataset captured during the ground-truthing session at Wageningen. In this session, data was recorded using the 10 camera array visualized in Figure 1. Moreover, the dataset contains an accurate ground-truth trajectory of the robot pose acquired with a high precision laser tracker. We are interested in evaluating the accuracy of the calibration with respect to the accuracy of the trajectory estimated via SLAM as this is the main purpose of the multi-camera rig. To evaluate the impact of the continuous calibration refinement during the operation of the robot, we thus compare the average localization error over the entire trajectory computed using our SLAM system [5]. We initialized the system using the values obtained from the offline calibration. Without continuous refinement of the calibration, our system achieves an average localization error of 0.18m over the complete sequence of 80m. If we only refine the rigid extrinsics of the camera array, we achieve an average error of 0.16m over the complete sequence, while a joint refinement of both the intrinsics and extrinsics leads to an error of 0.15m. We conclude that, for this sequence, the refinement of parameters slightly improves the localization results. Due to relatively short length of the sequence, we believe that this effect becomes more pronounced when evaluating the continuous refinement over longer sequences and time frames. Note that the continuous refinement of the extrinsic parameters has only a small performance impact and does not impair the real-time capability of the SLAM system. As more ground-truth datasets become available during the Trimbot project, we will re-evaluate the accuracy on the new datasets as well.

Extrinsic Self-Calibration We evaluate the accuracy of our extrinsic self-calibration routine by comparing it against the obtained extrinsics from the offline calibration as a ground-truth. We compare the obtained extrinsic calibration both in terms of positional and rotational error. Figure 8 visualizes the obtained extrinsic calibration for the ground-truth dataset from Wageningen for different steps in the self-calibration. Given no prior information about the extrinsics of the calibration, we obtain an accurate estimation of the camera extrinsics. Figure 9 quantifies the calibration accuracy after the global Structure-from-Motion initialization and after the bundle adjustment. We observe that while the initial estimate after rotation and translation averaging is typically rather inaccurate, the bundle adjustment step improves the results significantly and leads to a very accurate calibration of around 0.5° rotational and 1.9mm translational error. To evaluate the robustness of the self-calibration routine, we ran experiments on several other datasets. Whenever there is well-constrained robot motion, our calibration routine yields

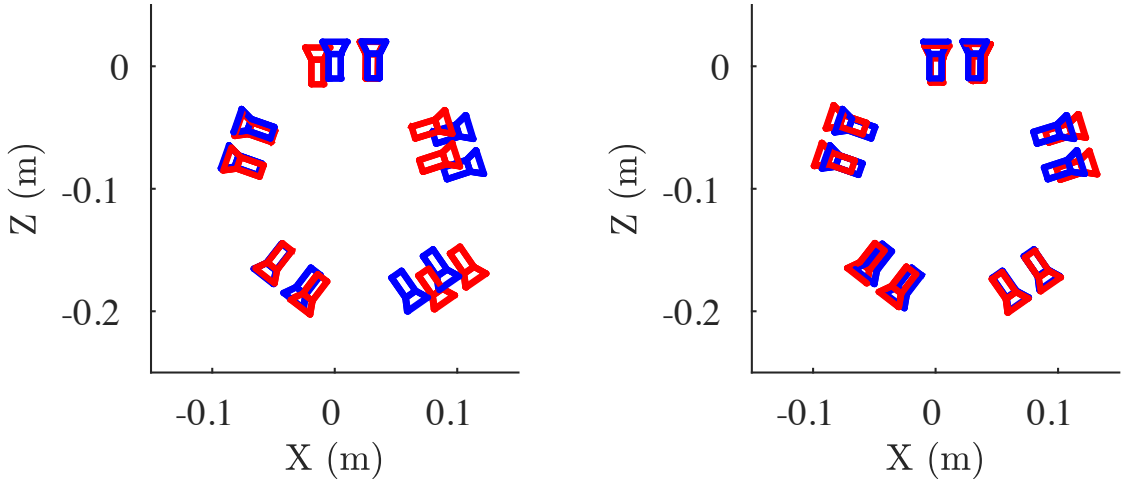


Figure 8: Visualization of the estimated camera array configurations after different steps in the online calibration (red) compared to the reference (offline) calibration (blue). Left: estimated configuration after the global rotation and translation averaging steps. Right: estimated configuration after the global bundle adjustment.

accurate results. In practice, we envision the robot to move in a pre-defined pattern for a few seconds in order to obtain a reliable self-calibration. As more datasets become available and as the calibration system is integrated into the robot, we will re-evaluate the performance of the extrinsic self-calibration.

3.5 Outlook

In order to supersede the offline calibration entirely, there are two problems that need to be addressed: First, it is necessary to also estimate the camera intrinsics. We are currently working on this point to develop a full self-calibration pipeline. Second, the online calibration returns the extrinsics of the camera up to an arbitrary scaling factor due to being based on images. This scaling factor can be recovered through knowing a distance in the scene.
In order to recover this scale, we will place markers with known positions on the chassis of the robot. As shown in the middle of Figure 2, one of the camera pairs observes the front of the robot as the camera system is mounted on its back (cf. Figure 5). As such, these marker points will always be visible in at least two cameras of the multi-camera rig. Both points will be addressed in Deliverable D3.2, which evaluates the SLAM system.

References

[1] Brown, D.: Decentering distortion of lenses. Photogrammetric Engineering (1966) 444–462

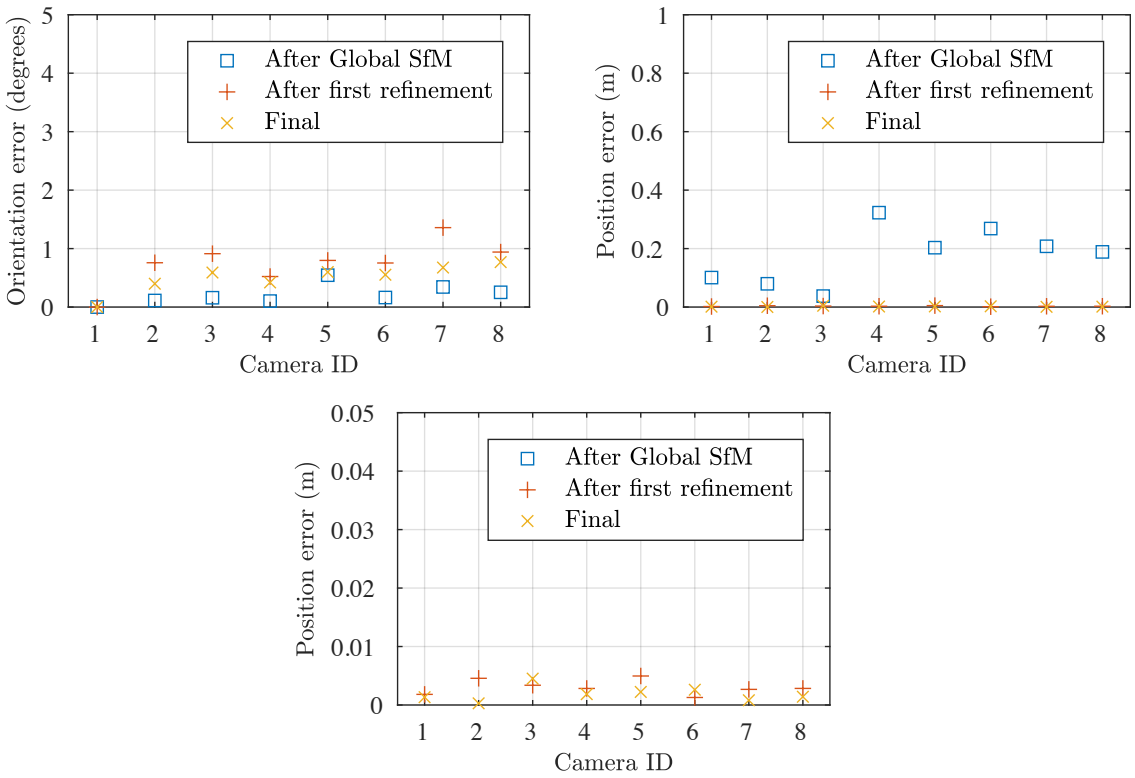


Figure 9: The calibration errors after calibration with the 'around garden' dataset. Top left the orientation errors. Top right the orientation errors with adapted plot scale to show the large position errors after Global SfM. At the bottom the position errors again with the usual plot scale for readability and better comparison with previous results. Interestingly, Global SfM returns very small orientation errors of $0.2°$ in average, while the position errors are large with an average error of 17 cm. After the final refinement, the mean error is $0.50°$ for orientations and 0.19 cm for positions.

[2] Furgale, P., Rehder, J., Siegwart, R.: Unified temporal and spatial calibration for multi-sensor systems. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems. (2013) 1280–1286

[3] Furgale, P., Barfoot, T.D., Sibley, G.: Continuous-time batch estimation using temporal basis functions. In: 2012 IEEE International Conference on Robotics and Automation. (2012) 2088–2095

[4] Maye, J., Furgale, P., Siegwart, R.: Self-supervised calibration for robotic systems. In: 2013 IEEE Intelligent Vehicles Symposium (IV). (2013) 473–480

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

[6] Smith, R.C., Cheeseman, P.: On the representation and estimation of spatial uncertainty. The international journal of Robotics Research **5** (1986) 56–68

[7] Triggs, B., McLauchlan, P., Hartley, R., Fitzgibbon, A.: Bundle adjustmenta modern synthesis. Vision algorithms: theory and practice (2000) 153–177

[8] Chatterjee, A., Madhav Govindu, V.: Efficient and robust large-scale rotation averaging. In: Proceedings of the IEEE International Conference on Computer Vision. (2013) 521–528

[9] Wilson, K., Snavely, N.: Robust global translations with 1dsfm. In: ECCV. (2014)

[10] Crandall, D.J., Owens, A., Snavely, N., Huttenlocher, D.P.: Sfm with mrfs: Discrete-continuous optimization for large-scale structure from motion. IEEE transactions on pattern analysis and machine intelligence **35** (2013) 2841–2853

