Online Calibration for Networks of Cameras and Depth Sensors

Filippo Basso, Riccardo Levorato and Emanuele Menegatti

Abstract— **This paper presents a novel approach to the sensor network calibration problem. Its aim is to easily calibrate a network composed by heterogeneous sensors, taking advantage of the Robot Operating System (ROS) framework. The proposed approach is able to calibrate – in a unique and consistent reference frame – the extrinsic parameters of a network composed by standard cameras and depth sensors. Compared to other state-of-the-art implementations, the presented algorithm performs an online calibration and optimization with minimal human intervention. Results of both simulation and real experiments are provided.**

I. INTRODUCTION

Robotic systems and sensor networks consist of many heterogeneous sensors. The estimation of the poses of all such sensors with respect to a unique consistent world frame is a challenging and well-known problem. As a matter of fact, a good calibration of different sensors can be a useful starting point for several applications both in the field of sensor networks (e.g. 3D mapping, people recognition and tracking [1], microphone calibration for audio localization) and in many robotics applications (e.g. simultaneous localization and mapping (SLAM) applications, grasping and manipulation).

Researchers proposed lots of techniques for calibrating specific sensors or pairs of sensors, even of different types; however, there are only few methods specifically developed to simultaneously calibrate an heterogeneous sensor network, e.g. [2]. As stated by Le et al. [2], the most followed approach is to divide the system into pairs of sensors and calibrate each pair independently, even using different algorithms for each one. In this context, our aim is to develop an easy-to-use algorithm for the simultaneous calibration of many different kinds of sensors considering them as a unique group. Moreover, since calibration is a complex and time-consuming task, a fast procedure would be a very useful tool, especially when the involved sensors need often to be moved (and therefore re-calibrated). To achieve both these goals, we:

1) assume the sensors' intrinsic parameters are already calibrated;
2) perform the calibration online, that is while the data are being acquired;
3) take advantage of all the tools offered by the Robot Operating System (ROS) [3].

This research has been partially supported by Telecom Italia S.p.A. with the grant "Service Robotics", by University of Padova with the grants "DVL-SLAM" and "TIDY-UP: Enhanced Visual Exploration for Robot Navigation and Object Recognition". Basso, Levorato and Menegatti are with the Department of Information Engineering (DEI), University of Padova, Italy. Email: {filippo.basso, riccardo.levorato, emanuele.menegatti}@dei.unipd.it.

The choice to deal with sensors whose intrinsic parameters have already been calibrated comes from the fact that the intrinsic parameters need to be estimated only once, they usually do not change over time. This assumption allows us not only to drastically reduce the computation time, but also makes the extrinsic calibration procedure less sensor-dependent. For what concerns the online calibration, it lets us reduce the time for the whole procedure to complete. Moreover it allows us to give the user a real-time feedback about the calibration status: the interface becomes user-friendly. Finally, the calibration of a multi-sensor system requires the program to deal with real sensors and, above all, the data provided by every sensor needs to be synchronized with all the other sensors' data. In this context, ROS gives an easy way to deal with hardware while hiding all the low-level network synchronization procedures.

We propose a simple calibration procedure for heterogeneous sensor networks that:

- deals with different kinds of sensors (e.g. standard cameras, Kinect-like depth sensors, time-of-flight cameras, omnidirectional cameras, actuated laser scanners, etc.) in a uniform way;
- is fast and able to perform the calibration in an online way with a minimal user supervision.

Our system is an extension of the classical single camera calibration procedure. We move a checkerboard pattern in front of the each camera [4] and depth sensor. As soon as some of the sensors see the checkerboard, the calibration starts. Then whenever the pattern is visible by at least two sensors simultaneously, we add a constraint to the calibration problem. At the end, all the data are processed inside an optimization framework that improves the quality of the initial estimation. The pattern is easily extracted from both images with a state-of-the-art corner finder algorithm, and from depth data by means of a segmentation algorithm.

Results based on simulation and online real tests are also presented and compared to those given by a state-of-the-art toolbox.

A. Related Work

In the robot vision field, RGB cameras have been a key technology in the development of visual perception. In the very last years, the introduction of depth sensors contributed deeply to the advancement of sensor fusion in practical applications. Auvinet et al. [5] proposed a new method for calibrating multiple depth cameras for body reconstruction using only depth information. They do not use a particular object but rely on plane intersections. For the Kinects' data synchronization, they exploit the NTP protocol and the

OpenNI library [6]. Their calibration achieves good results: even if the depth (z-axis) error of the sensor is 10 mm, the reconstruction error with 3 depth cameras is, in the best case, less than 6 mm. A drawback of their implementation is that they have to manually select the plane corners and, above all, they only deal with depth sensors avoiding the possibility to add the color information to the fused data.

Le and Ng [2] jointly calibrates groups of sensors. Their groups are formed using a unifying principle: they all output 3D data. More specifically, each group is composed by a set of sensors that can provide a 3D representation of the world (e.g. a stereo camera, an RGB camera and a depth camera, etc.). They make use of the fact that a sensor can appear in different groups to have redundancy. So, first of all they calibrate the intrinsics of each sensor, secondly they calibrate the extrinsic parameters of each group of sensors and then they calibrate the extrinsic parameters of each group with respect to all the others. Finally they calibrate the entire system in one optimization step. Their experiments show that this method not only reduces the calibration error, but also requires a little human intervention. An advantage of having groups that output 3D data is that the same calibration objective can be used to calibrate each group with respect to the others, regardless of the type of sensor. Also, a joint calibration with redundancy has higher accuracy. An explanation is that different pairs see different views of the world, so the algorithm can make use of more constraints. Another reason is that a joint calibration does not accumulate errors like a calibration based on sensor pairs do. However they also state that they should combine this two steps and jointly calibrate all parameters at once, as we propose in our work. The main drawback of this approach is that they always need to group the sensors beforehand in order to have 3D data outputs – that it's not always possible.

Jointly calibrating the intrinsic and extrinsic parameters of cameras has been studied as well; for example, Zhang [7] proposed to jointly estimate the intrinsics and extrinsics using one objective function. Starting from this approach, a multi-camera version has been implemented: the Automatic Multi-Camera Calibration Toolbox [8]. Obviously, approaches like the one just mentioned are not easily extendable to other kinds of sensors rather than standard cameras.

B. Notations

We use non-bold characters x to represent scalars, bold lower case letters $\mathbf{x}$ to represent vectors with no distinction between cartesian coordinates and homogeneous coordinates. Bold upper case letters $\mathbf{M}$ represent matrices. Note that matrices can be seen as ordered lists of vectors, one for each column. A coordinate frame belonging to a body B is denoted by $\mathcal{B}$. The coordinates of an entity e with respect to the reference frame $\mathcal{F}$ are denoted by ${}^{\mathcal{F}}e$. In this context, the pose of A in B's coordinate system $\mathcal{B}$ is denoted as ${}^{\mathcal{B}}\mathcal{A}$ and the relative homogeneous transformation matrix is ${}^{\mathcal{B}}_{\mathcal{A}}\mathbf{T}$. As an example, a point $\mathbf{p}$ in A's coordinate system can be transformed to B's coordinate system as following:

$$ {}^{\mathcal{B}}\mathbf{p} = {}^{\mathcal{B}}_{\mathcal{A}}\mathbf{T} \cdot {}^{\mathcal{A}}\mathbf{p}. $$

II. CALIBRATION PROBLEM

Let $\overline{\mathsf{S}} = \{\mathsf{S}_1, \mathsf{S}_2 \dots \mathsf{S}_L\}$ be the set of sensors, either cameras or depth sensors, in the network. For each sensor $\mathsf{S}_i \in \overline{\mathsf{S}}$, $i = 1 \dots L$, the goal is to find its pose ${}^{\mathcal{W}}\mathcal{S}_i$ with respect to a common reference frame $\mathcal{W}$, namely the *world*.

In the following we assume that all the sensors' intrinsic parameters are already calibrated. Moreover, for what concerns depth sensors, since we want also to deal with "a posteriori" calibrated sensors (cfr. [9], [10]), we assume they cannot provide any intensity information of the scene (e.g. Kinect's infrared images are not considered to be available). In fact, the above-mentioned calibration methods estimate the intrinsic parameters using 3D data and therefore the intensity images cannot be corrected according to them.

A. Camera-to-Camera Calibration

A solution to the calibration problem for camera-only networks is really simple to achieve. A common way is to use a specific pattern, e.g. a checkerboard, visible by (at least) two cameras at a time. First, the pose of the pattern with respect to each camera (using for example OpenCV [11] `solvePnP()` method) is estimated, then the transformation between the two camera frames is computed by means of simple geometrical operations.

Let, for example, C_1 and C_2 be the two cameras and B the checkerboard. Let also ${}^{\mathcal{C}_1}\mathcal{B}$ and ${}^{\mathcal{C}_2}\mathcal{B}$ be, respectively, the pose of $\mathcal{B}$ with respect to $\mathcal{C}_1$ and $\mathcal{C}_2$. Then the transformation ${}^{\mathcal{C}_1}_{\mathcal{C}_2}\mathbf{T}$ can be computed as

$$ {}^{\mathcal{C}_1}_{\mathcal{C}_2}\mathbf{T} = {}^{\mathcal{C}_1}_{\mathcal{B}}\mathbf{T} \cdot {}^{\mathcal{C}_2}_{\mathcal{B}}\mathbf{T}^{-1}. \tag{1} $$

The procedure can be applied to all the camera pairs having an overlapping field of view. Imaging to create a graph with all such pairs (the cameras are the nodes while the transformations the branches), as soon as it becomes connected we have a rough calibration of the whole network. It is obvious that the computed parameters are usually far from being optimal. So at the end the estimated poses need to be adjusted by means of an optimization algorithm.

A network composed by heterogeneous sensors, such as cameras and depth sensors, can be calibrated in the same way. The procedure is not straightforward, but follows the very same workflow:

1) Roughly calibrate each possible pair.
2) Create a graph with the computed transformations.
3) Optimize.

What we need is therefore an algorithm to calibrate a camera with a depth sensor and a depth sensor with another depth sensor so that we are able to create the graph (Sec. II-B). Then we need an error function to use in our optimization framework to have the best possible results (Sec. III).

B. Camera-to-Depth Sensor Calibration

Checkerboards work really well with color cameras, so we would like to keep using them also for calibrating a camera with a depth sensor. However, because of the lack of intensity information, it is not possible to extract the

checkerboard corners' locations from the depth images and follow the approach described in section II-A. We need to exploit some other properties. One of these properties is planarity. Following the approach described in [12], we can use plane-to-plane constraints to estimate the relative poses of the two sensors by using a checkerboard.

The only difference with the previously described camera-to-camera calibration method, is the fact that this procedure needs at least three instances of the checkerboard to be able to estimate the transformation. It is also worth to mention that the very same approach can also be used to calibrate a depth sensors couple.

C. Optimization

The optimization step follows a bundle adjustment approach: both the poses of the sensors and the various locations of the checkerboard in the world are refined. The resulting parameters are the ones that most accurately predict the locations of the checkerboard in the acquired data (images and depth images). The error functions we use in the optimization framework are described in section III.

III. OPTIMIZATION ERROR

Let B be a checkerboard with reference frame $\mathcal{B}$ and let ${}^{\mathcal{B}}\mathbf{B}$ be the set of its corners (in the checkerboard's coordinate system). Let also $\overline{\mathsf{C}} = \{\mathsf{C}_1, \mathsf{C}_2 \dots \mathsf{C}_N\}$ be the set of cameras and $\overline{\mathsf{D}} = \{\mathsf{D}_1, \mathsf{D}_2 \dots \mathsf{D}_M\}$ the set of depth sensors.
According to the above notation, and supposing we have already performed K acquisition steps, we can easily enumerate the constraints the acquired data impose:

1) The pose of every camera (${}^{\mathcal{W}}\mathcal{C}_n$, $n = 1 \dots N$) and depth sensor (${}^{\mathcal{W}}\mathcal{D}_m$, $m = 1 \dots M$), with respect to the world reference frame $\mathcal{W}$, must be the same at each step k.
2) The pose of the checkerboard with respect to the world reference frame at step k, namely ${}^{\mathcal{W}}\mathcal{B}_k$, must be kept constant.

We can therefore design our error function e as

$$\mathrm{e} = \sum_{k=1}^{K} \left[\frac{1}{\sigma_{\mathrm{cam}}^2} \sum_{n=1}^{N} u_{nk} \cdot \mathrm{e}_{\mathrm{cam}}({}^{\mathcal{W}}\mathcal{C}_n, {}^{\mathcal{W}}\mathcal{B}_k) + \frac{1}{\sigma_{\mathrm{dep}}^2} \sum_{m=1}^{M} u_{mk} \cdot \mathrm{e}_{\mathrm{dep}}({}^{\mathcal{W}}\mathcal{D}_m, {}^{\mathcal{W}}\mathcal{B}_k) \right], \quad (2)$$

where $\mathrm{e}_{\mathrm{cam}}(\cdot)$ and $\mathrm{e}_{\mathrm{dep}}(\cdot)$ are the two different errors we can compute, respectively, on the image and on the depth data, while σ_{cam} and σ_{dep} are the normalization factors. u_{nk} and u_{mk} are instead two indicator functions: they are equal to 1 if at step k camera C_n or, respectively, depth sensor D_m sees the checkerboard B, otherwise they are equal to 0.

Going deeper into details, at each acquisition step $k = 1 \dots K$ we can assume that the checkerboard is visible by at least two sensors[1], one of which is a camera. The latter assumption is necessary to estimate the pose of the checkerboard in the world.

[1]If this is not the case, data are discarded. It makes no sense to keep these data, since we cannot use it to constraint the pose of a sensor with respect to another one.

A. Camera Error Function $\mathrm{e}_{\mathrm{cam}}(\cdot)$

We consider the error as the reprojection error of the checkerboard corners onto the image. Let $\mathsf{C} \in \overline{\mathsf{C}}$ be a camera with reference frame $\mathcal{C}$. Let also ${}^{\mathcal{W}}\mathcal{B}$ and ${}^{\mathcal{W}}\mathcal{C}$ be the poses, respectively, of the checkerboard B at step k (here we omit the subscript k) and the camera C with respect to the world reference frame $\mathcal{W}$. We can easily compute the transformation ${}^{\mathcal{C}}_{\mathcal{B}}\mathbf{T}$ of $\mathcal{B}$ with respect to $\mathcal{C}$ as

$${}^{\mathcal{C}}_{\mathcal{B}}\mathbf{T} = {}^{\mathcal{W}}_{\mathcal{C}}\mathbf{T}^{-1} \cdot {}^{\mathcal{W}}_{\mathcal{B}}\mathbf{T} \quad (3)$$

and therefore transform each corner ${}^{\mathcal{B}}\mathbf{b} \in {}^{\mathcal{B}}\mathbf{B}$ to camera coordinates by pre-multiplying it by the affine transformation ${}^{\mathcal{C}}_{\mathcal{B}}\mathbf{T}$, that is

$${}^{\mathcal{C}}\mathbf{b} = {}^{\mathcal{C}}_{\mathcal{B}}\mathbf{T} \cdot {}^{\mathcal{B}}\mathbf{b}. \quad (4)$$

For each corner $\mathbf{b} \in \mathbf{B}$ we call $\hat{\mathbf{b}}$ the corner extracted from the image in pixel coordinates. The reprojection error of the corners can be computed as

$$\mathrm{e}_{\mathrm{cam}}({}^{\mathcal{W}}\mathcal{C}, {}^{\mathcal{W}}\mathcal{B}) = \sum_{\mathbf{b} \in \mathbf{B}} \left\| \mathrm{repr}_{\mathsf{C}}({}^{\mathcal{C}}\mathbf{b}) - \hat{\mathbf{b}} \right\|^2, \quad (5)$$

where $\mathrm{repr}_{\mathsf{C}}(\cdot)$ is the reprojection function that returns the pixel coordinates of a 3D point using camera C intrinsic parameters.

B. Depth Sensor Error Function $\mathrm{e}_{\mathrm{dep}}(\cdot)$

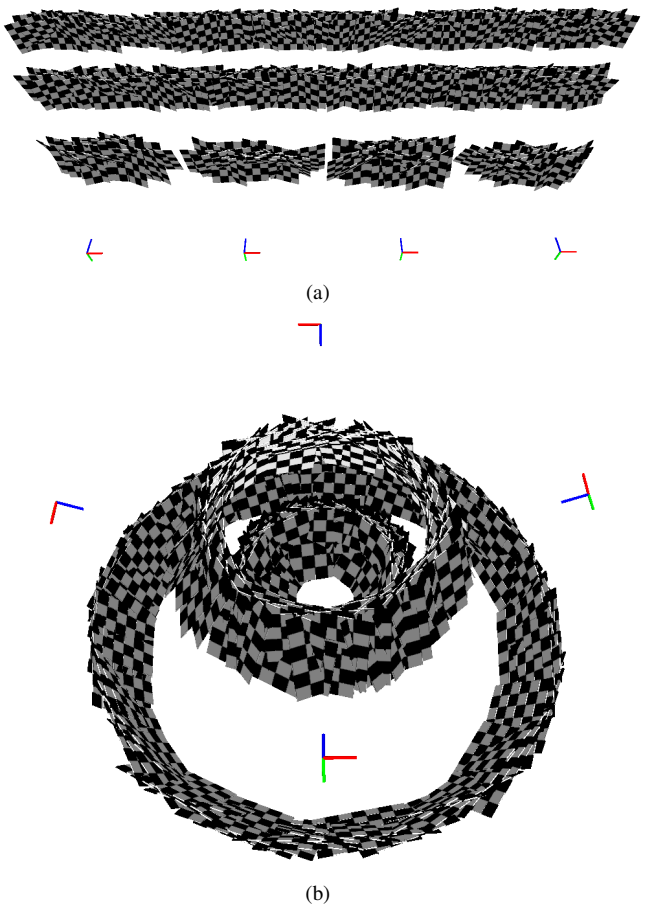
We calculate the error as a sort of distance between the plane defined by the checkerboard and the portion of depth data belonging to the checkerboard. Let $\mathsf{D} \in \overline{\mathsf{D}}$ be a depth sensor with reference frame $\mathcal{D}$ and let ${}^{\mathcal{W}}\mathcal{D}$ be the pose of D with respect to the world reference frame $\mathcal{W}$. Let also ${}^{\mathcal{D}}\pi$ be the plane fitted to the portion of depth data belonging to the checkerboard in sensor coordinates. We can estimate the location of each checkerboard corner in depth coordinates ${}^{\mathcal{D}}\mathbf{B}$ by using equation (3) and (4) and therefore express the error as

$$\mathrm{e}_{\mathrm{dep}}({}^{\mathcal{W}}\mathcal{D}, {}^{\mathcal{W}}\mathcal{B}) = \sum_{\mathbf{b} \in \mathbf{B}} \left\| {}^{\mathcal{D}}\mathbf{b} - \mathrm{los\_repr}_{\pi}({}^{\mathcal{D}}\mathbf{b}) \right\|^2, \quad (6)$$

where $\mathrm{los\_repr}_{\pi}(\cdot)$ is the function that project a 3D point to the plane π along its line-of-sight as described in [13].

The normalization factor σ_{dep} in equation (2) is the error of the depth measurements. If it depends on the depth value, it must be removed from equation (2) and added to equation (6) that becomes

$$\mathrm{e}_{\mathrm{dep}}({}^{\mathcal{W}}\mathcal{D}, {}^{\mathcal{W}}\mathcal{B}) = \sum_{\mathbf{b} \in \mathbf{B}} \frac{1}{\sigma_{\mathrm{dep}}^2({}^{\mathcal{D}}\mathbf{b})} \left\| {}^{\mathcal{D}}\mathbf{b} - \mathrm{los\_repr}_{\pi}({}^{\mathcal{D}}\mathbf{b}) \right\|^2.$$



(a)

(b)

Fig. 1. The two test scenarios we simulated. (a) N aligned Kinects at a fixed distance. (b) N Kinects in a circle of a fixed radius.

IV. Experiments

A. Simulation

We simulated two different scenarios to test our algorithm:

1) N aligned Kinects (RGB + Depth sensor) at a fixed distance (Fig. 1(a)).
2) N Kinects in a circle of a known radius (Fig. 1(b)).

We supposed that the location of a corner c in an image is estimated with a normally distributed error with mean $\mu_c = 0$ and standard deviation $\sigma_c = 0.5$ pixels along both axes. Then, dealing with the error on the depth estimation, it is well known that it depends on the real depth of the considered pixel. In particular, it is a function of the square of the depth measurement[2]. So, for each pixel p in the depth image belonging to the checkerboard with a ground truth depth value d, we assumed the error to be normally distributed with mean $\mu_p = 0$ and a standard deviation $\sigma_p = 0.0035d^2$ meters (we performed some tries with $\sigma_p = 0.005d^2$ pixels, but results were pretty much the same).

For each test we computed both the translation error

$$\mathrm{e_{tra}} = \frac{1}{2 \cdot N - 1} \sum_{i=2}^{2 \cdot N} \|t_i - \hat{t}_i\| \tag{7}$$

[2] http://wiki.ros.org/openni_kinect/kinect_accuracy

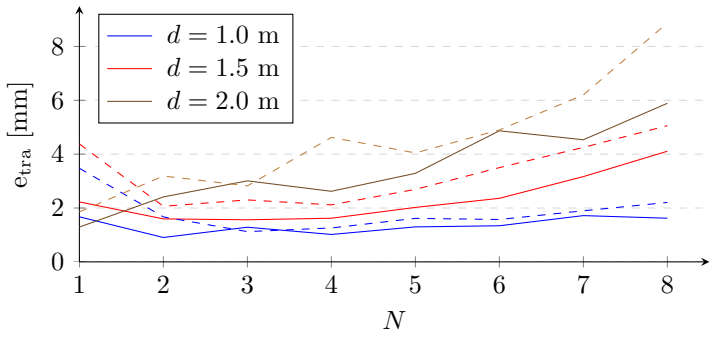


Fig. 2. Translation error $\mathrm{e_{tra}}$ computed both varying the number of sensors N and the distance d between them. Continuous lines report results of tests with $60 \cdot N$ checkerboards, while dashed lines report results of tests with $30 \cdot N$ checkerboards.

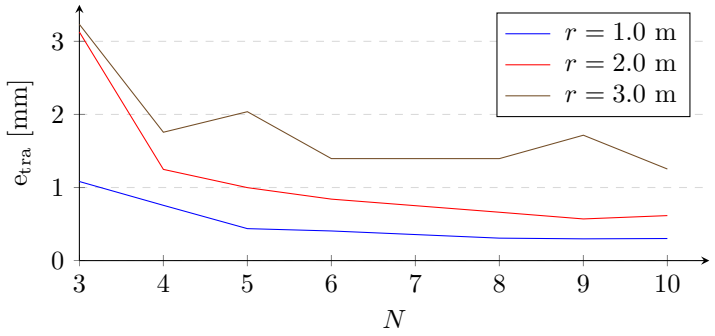


Fig. 3. Translation error $\mathrm{e_{tra}}$ computed both varying the number of sensors N and the radius r of the circle.

and the rotation error

$$\mathrm{e_{rot}} = \frac{1}{2 \cdot N - 1} \sum_{i=2}^{2 \cdot N} 2 \cdot \arccos(|{q_i}^* \cdot \hat{q}_i|). \tag{8}$$

Here t_i and q_i are respectively the estimated translation and rotation (expressed as a quaternion) of sensor i while $\hat{t}_i$ and $\hat{q}_i$ are the real ones. All the sensor poses are referred to sensor 1 that we consider the world reference frame. Therefore, remembering that each Kinect is composed by a camera and a depth sensor, we have $2 \cdot N - 1$ sensors overall for which to calculate the two errors. Fig. 2 shows the translation error $\mathrm{e_{tra}}$ computed both varying the number of sensors N and the distance d between them. As we expected, the closer the sensors, the better the calibration. In fact, when the distance is small there are more checkerboards in the overlapping field of view of neighboring sensors, especially close range ones, and therefore there are more (strong) constraints for the pose estimation.

In Fig. 3 is shown the translation error $\mathrm{e_{tra}}$ of the sensor poses when sensors are in a circle. In this case, an increase in the number of sensors makes the error decrease, while an increase in the radius of the circle makes the error increase. Results about the rotation error are not reported as they are very small (usually less than $0.1°$) and not that significative.

Fig. 4. One of the test scenarios. Four markers are placed on each kinect in order to recover their locations by means of a motion capture system.

TABLE I

COMPARISON BETWEEN THE CALIBRATION ESTIMATED WITH THE AMCCTOOLBOX AND WITH OUR PACKAGE.

		translation error [mm]				
TEST	STEPS	AMCC	OUR_D	OUR_A	OUR_BA	MUTUAL
1	130	16.755	16.230	**13.205**	**13.205**	11.5761
2	106	**10.371**	17.592	16.493	13.542	7.5705
3	104	**10.319**	18.118	12.512	13.949	6.0740
4	112	45.938	**29.029**	50.483	32.269	11.0807
5	101	37.628	**28.777**	43.043	32.489	8.3238
6	101	26.739	**22.744**	26.253	27.474	7.4885

B. Real Tests

To prove the validity of our approach, we also performed some tests in a real scenario. We measured the real poses of the sensors (i.e. the ground truth) by means of a motion capture system (Fig. 4) and compared these measures with the estimation obtained with our method and with the one given by a state-of-the-art Matlab toolbox (amcctoolbox) [8]. The comparison is made without taking into account the depth sensors, since the amcctoolbox is not meant to deal with such kind of sensors. Moreover its worth to notice that such package estimates both the intrinsic and the extrinsic parameters of all the cameras.

Results from 6 different test scenarios are provided in Table I. Test 1, 2 and 3 are grouped together since they have been performed on the same scene, same for test 4 and 5. For each test we report:

- STEPS — The number of acquisition steps, as described in section III.
- AMCC — The mean translation error (cfr. equation (7)) after the calibration with the amcctoolbox.
- OUR_D — The mean translation error after the calibration with our algorithm using as intrinsic parameters the default ones provided by ROS.
- OUR_A — The mean translation error after the calibration with our algorithm using as intrinsic parameters the ones provided by the amcctoolbox.
- OUR_BA — The mean translation error after the calibration with our algorithm using as intrinsic parameters the best ones provided by the amcctoolbox (actually the ones calculated during test 1).
- MUTUAL — The mean translation error between the two calibration with the same intrinsic parameters.

Looking at the results, we can state that none of the different setups clearly outperforms the others. Also, when using the same intrinsic parameters, the camera poses estimated by the two algorithms do not differ that much (MUTUAL). Moreover, we can notice that the results highly depend on the intrinsic parameters estimation: keeping them constant (OUR_D and OUR_BA) makes the results more stable.

What makes the difference between our algorithm and the others is the computational time, i.e. on a modern quad-core laptop[3], we are able to perform the whole calibration procedure during the GiB acquisition – 2 minutes in total – without the need to save all the images and process them back offline – 2 + 20 minutes in total for the amcctoolbox [8].

During the development of the toolbox, several real tests have been made with different sensor types (Microsoft Kinect, PrimeSense Xtion, standard and omnidirectional cameras, etc.). One of the most challenging test we performed was the calibration between an actuated laser rangefinder and an omnidirectional camera. We arranged a robotic platform (Fig. 5) with a Sick LMS200 laser rangefinder mounted on a Directed Perception pan-tilt unit (PTU46) and a catadioptric camera above. We then performed our calibration and obtained the results visible in Fig. 6. Despite the number of images and 3D scans was around 10, the results were really satisfactory. Note that the PTU46-LMS200 couple was already calibrated [13] and let us have a 3D view of half of our laboratory.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a camera-depth sensor network calibration algorithm. During its development, we focused on an approach as independent as possible from the involved sensor types and the outcoming implementation turned out to be both fast and, as proved by some tests, correct. The algorithm relies on previously calibrated sensors and takes advantage of highly optimized C++ code and libraries, e.g. ceres-solver [14]. Moreover, the choice of using ROS as the developing framework gives our implementation lots of advantages in managing the sensors and their synchronization.

One of the main drawbacks of the presented approach is that it highly depends on the intrinsic parameters provided. If they are not well estimated, the calibration results are not really accurate.

A first version of the code is available on GitHub at `https://github.com/iaslab-unipd/multisensor_calibration`. This is a ROS package intended to foster the development of new (and hopefully more accurate) calibration algorithms developed by the ROS community.

As a future work, we envision to use this toolbox in an multi-sensor speaker localization and tracking application for robotic purposes including audio, video and depth sensors.

[3]CPU: Intel® Core™ i7-4700MQ CPU @2.40GHz × 8. RAM: 12 GiB.

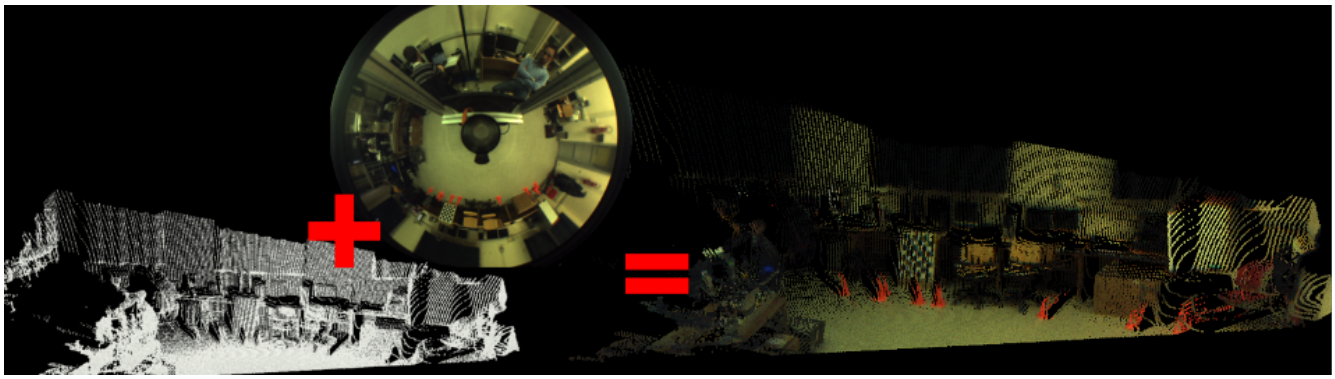


Fig. 6. Results of the calibration between a laser rangefinder and an omnidirectional camera. Bottom left: the 3D scan obtained by the actuated laser. Top center: the omnidirectional image. Right: the fused data after the calibration.

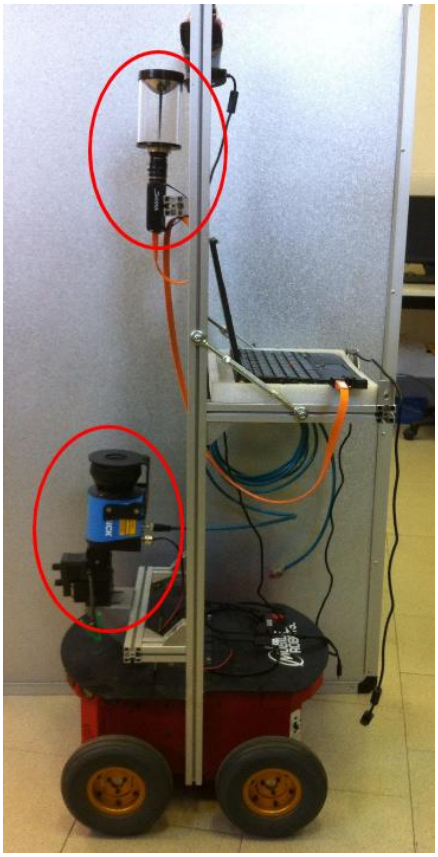


Fig. 5. The platform we used in our test. On the bottom you can see the laser rangefinder on the pan-tilt unit, while on the top there is the omnidirectional camera.

REFERENCES

[1] F. Basso, M. Munaro, S. Michieletto, and E. Menegatti, "Fast and robust multi-people tracking from RGB-D data for a mobile robot," in *Proceedings of the 12th Intelligent Autonomous Systems (IAS) Conference, Jeju Island (Korea)*, vol. 193, June 2012, pp. 265–276.

[2] Q. Le and A. Ng, "Joint calibration of multiple sensors," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, Oct 2009, pp. 3651–3658.

[3] "Robot Operating System (ROS)," http://www.ros.org/.

[4] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, 2004.

[5] E. Auvinet, J. Meunier, and F. Multon, "Multiple Depth Cameras Calibration and Body Volume Reconstruction for Gait Analysis," in *Information Science, Signal Processing and their Applications (ISSPA), 2012 11th International Conference on*, July 2012, pp. 478–483.

[6] "OpenNI," http://www.openni.org/.

[7] Z. Zhang, "Flexible Camera Calibration by Viewing a Plane from Unknown Orientations," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 1, 1999, pp. 666–673.

[8] M. Warren, D. McKinnon, and B. Upcroft, "Online Calibration of Stereo Rigs for Long-Term Autonomy," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, Karlsruhe, Germany, 2013.

[9] A. Teichman, S. Miller, and S. Thrun, "Unsupervised Intrinsic Calibration of Depth Sensors via SLAM," in *Proceedings of Robotics: Science and Systems*, Berlin, Germany, June 2013.

[10] F. Basso, A. Pretto, and E. Menegatti, "Unsupervised Intrinsic and Extrinsic Calibration of a Camera-Depth Sensor Couple," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, Hong Kong, China, June 2014.

[11] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[12] R. Unnikrishnan and M. Hebert, "Fast Extrinsic Calibration of a Laser Rangefinder to a Camera," Carnegie Mellon University, Tech. Rep., 2005.

[13] E. So, F. Basso, and E. Menegatti, "Calibration of a Rotating 2D Laser Range Finder using Point-Plane Constraints," *Journal of Automation, Mobile Robotics & Intelligent Systems*, vol. 7, no. 2, pp. 30–38, 2013.

[14] S. Agarwal and K. M. et al., "Ceres Solver," https://code.google.com/p/ceres-solver/.