Distributed 3D TSDF Manifold Mapping for Multi-Robot Systems

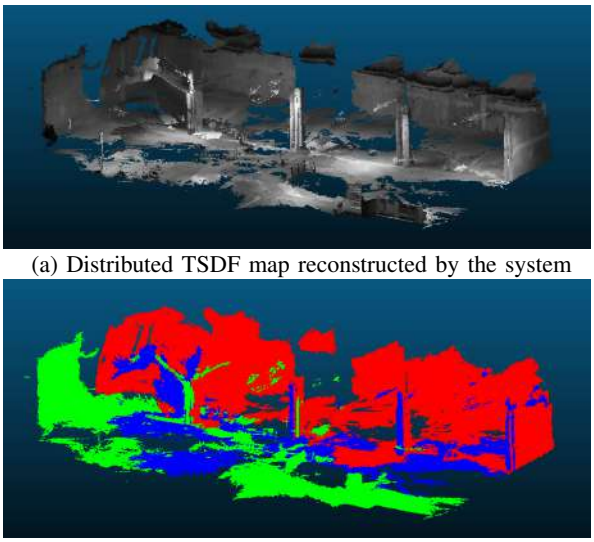
Thibaud Duhautbout[1], Julien Moras[2] and Julien Marzat[2]

***Abstract*— This paper presents a new method to perform collaborative real-time dense 3D mapping in a distributed way for a multi-robot system. This method associates a Truncated Signed Distance Function (TSDF) representation with a manifold structure. Each robot owns a private map which is composed of a collection of local TSDF sub-maps called *patches* that are locally consistent. This private map can be shared to build a public map collecting all the patches created by the robots of the fleet. In order to maintain consistency in the global map, a mechanism of patch alignment and fusion has been added. This work has been integrated in real-time into a mapping stack, which can be used for autonomous navigation in unknown and cluttered environment. Experimental results on a team of wheeled mobile robots are reported to demonstrate the practical interest of the proposed system, in particular for the exploration of unknown areas.**

I. Introduction

Autonomous robot navigation is a complex task that requires a set of distinct capabilities. Among them, the construction of a dense map of the environment is particularly important for safe path planning. Several studies have demonstrated the ability to make a single robot navigate autonomously in unknown and congested environments. The robustness to drift for simultaneous localization and mapping (SLAM) has also seen a lot of research effort, and several methods have been developed to ensure the consistency of the map over time. Although one of the next steps is to improve the reliability of these approaches, another one is to scale to multi-robot systems. Indeed, one way to increase the performances of the system is to use a team of robots working together (e.g. for coverage or exploration tasks). In this case, the mapping problem becomes harder since each robot should build a map from its own measurements, share this information within the team and fuse information coming from the other robots. The main challenges are to ensure the consistency of the map and to build a useful representation for autonomous navigation with limited computational resources.

We propose a distributed 3D mapping approach combining a TSDF representation and a manifold map structure composed of a set of local sub-maps called *patches* (main components described in Section III). The mapping system presented in Section IV includes the simultaneous management of private and public maps to deal with the multiple sources of information. In particular, a strategy to realign and fuse patches on the fly is proposed to improve the global map consistency; this could be used to perform posterior corrections on the robots' estimated trajectories. An interface has also been designed to locally check collisions on request, which is well suited for path planning with collision avoidance constraints. These features were tested with a team of wheeled mobile robots equipped with stereo-vision sensors and real-time processing capabilities, in remote operation and for autonomous exploration of unknown areas (Section V).



(a) Distributed TSDF map reconstructed by the system

(b) Contributions of three robots (different colors) to the global map

Fig. 1: Example of distributed multi-robot TSDF 3D map

II. Related Work

Environment mapping for robot navigation has been studied for a few decades and multiple approaches have been proposed relying on different paradigms. The development of 3D mapping methods has expanded in the last ten years thanks to the simultaneous progress in computational capabilities and availability of 3D sensors. In the following of 2D Occupancy Grid (OG) maps [1] that have been widely studied in previous decades, the first significant 3D modeling approach named *Octomap* was proposed in [2]. It relies on a 3D occupancy grid with an internal Octree model. This representation adapts the level of detail of the map to the environment, which reduces memory usage. The approach was implemented as an open-source library, which is particularly optimized to run in real time on CPU. In addition, since the occupancy grid representation is not very convenient for path planning algorithms, an Euclidean Distance Transform (EDT) module has been added [3]. It

[1]T. Duhautbout is with Sorbonne Universités, Université de technologie de Compiègne, CNRS, Heudiasyc UMR 7253, F-60203 Compiègne, France, thibaud.duhautbout@hds.utc.fr

[2]J. Moras and J. Marzat are with DTIS, ONERA, Université Paris Saclay, F-91123 Palaiseau, France, julien.moras@onera.fr, julien.marzat@onera.fr

978-1-7281-3605-9/19/$31.00

allows to compute the distance to the closest obstacles with a refresh rate of 1 to 2 Hz on a usual embedded CPU. Newcombe et al. [4] proposed another 3D reconstruction method entitled *Kinect Fusion*. This approach was not directly intended for robotic navigation. It uses measurements of a RGB-D sensor to calculate the TSDF (Truncated Signed Distance Function) [5] over a grid. The main limitation of this method comes from the need to run on a GPU, so the volume mapped is limited to a few meters by the available memory. In [6], the method was improved to map larger areas thanks to a moving TSDF volume. Areas that fall out of the volume are exported into a mesh representation and stored in the system memory. Since the TSDF map is mostly empty (within usual truncation distances), it was proposed to allocate voxels on the fly and to access them using a spatially-hashed index [7]. This method allows to drastically reduce the memory size occupied by the TSDF and access time, with limited effects on performances. This idea was kept in the development of the open-source *Chisel* library[1], which provides a TSDF approach for 3D reconstruction onboard of mobile devices [8]. More Recently, Oleynikova et al. proposed *Voxblox* [9], a 3D mapping method for robotic navigation. This method was directly derived from *Chisel*, with the addition of an Euclidean distance field estimation to produce a similar output as *Octomap/EDT*. An Octree-based mapping stack that is able to build either a TSDF or a probabilistic occupancy map has also been presented in [10]. The *C-Blox* method recently proposed in [11] adapted the concept of manifold mapping from [12] within the TSDF framework to obtain a mapping approach robust to localization drift. The idea consists in replacing a monolithic map linked to a single fixed frame by a map splitted into a set of *patches* (i.e. local sub-maps).

Compared to the single robot case, the multi-robot case has been little studied, especially for 3D mapping. Previous work has mainly focused on matching and merging mono-robot maps. Such methods have been proposed in [13], [14] using occupancy grids. In [15], a collaborative mapping approach has been designed for multiple Micro Aerial Vehicles (MAV) performing a matching and merging process relying on a point cloud representation. An important step was achieved with the work of Howard et al. [12] where the concept of Manifold Mapping was introduced in 2D. This approach aims at enhancing map consistency using additional dimensions (e.g. *time*). The proposed implementation splits the map into a set of *patches* (corresponding to the new dimension). Using this method, the authors were able to successfully build an OG to map a large environment (up to $600\,\mathrm{m}^2$) using four robots equipped with laser sensors. A 2D multi-robot SLAM approach built from laser data and relying on signed distance functions has also been proposed in [16].

We propose a distributed mapping approach for multi-robot systems based on a structure similar to the one of *C-Blox* [11] with a 3D extension of the manifold concept from [12]. The main objective is to allow several robots to share *patches* so as to be able to plan their own trajectory using information from the rest of the fleet, including alignment correction between patches. The focus is put on the multi-robot mapping system, therefore it is assumed that the localization of the robots is computed by another algorithm, e.g. [17], [18] (the latter was used in the experiments). The other main input of the proposed system is depth information, which can be retrieved either from stereo-vision, RGB-D or laser sensors (here, stereo-vision data was processed with ELAS [19]).

[1] https://github.com/personalrobotics/OpenChisel

III. Manifold TSDF Mapping

A. TSDF mapping

The environment is modeled as a *truncated signed distance function* (TSDF) [5]. In this representation, space is discretized using a given *resolution* into a set of *voxels*. Each voxel v observed by a range sensor is updated to store the signed distance $\boldsymbol{d}(v)$ between the voxel center and the 3D point measured on the nearest surface. A weight $\boldsymbol{w}(v)$ is also associated to each voxel in order to fuse the measurements depending on the sensor precision. A quadratic-decay method is used to compute the weight, as in [9]. The distance and the weight of each voxel are updated with new data as follows:

$$\begin{cases} \boldsymbol{d}(v) \leftarrow \dfrac{\boldsymbol{d}(v) \cdot \boldsymbol{w}(v) + d \cdot w}{\boldsymbol{w}(v) + w} \\ \boldsymbol{w}(v) \leftarrow \boldsymbol{w}(v) + w \end{cases} \tag{1}$$

where d corresponds to the distance value computed for the voxel v and w is the associated weight. The distance stored in a voxel is *signed*: by convention, it is positive if the voxel is located between the sensor and the surface, and negative if the voxel is behind the surface. The distance function is also *truncated* at a given threshold d_{max} – if the distance d measured for a voxel is such that $|d| > d_{max}$, the measurement is not integrated. The implementation used in this work is based on a spatially-hashed TSDF [7]. In this case, the whole TSDF grid is not allocated at the beginning. During the TSDF update process, the system queries the voxels within truncation distance of the measured surface. If a voxel has not been allocated yet, a chunk (i.e. a small set of voxels) is allocated at this location. The chunk is then added to a list and can be accessed using a key generated with its 3D coordinates. The signed distance function implicitly defines the surface as the zero-crossing (also called zero-level-set) of the function. The marching cubes algorithm [20] is used to compute the zero-crossing and extract the surface as a polygonal mesh.

B. Manifold map structure

The manifold map structure introduced in [12] is dedicated to multi-robot mapping in a 2D space. The manifold is defined as a continuous 3-dimensional representation of the map built by a robot, composed of the 2D plane and time. The time dimension is added to ensure self-consistency and to allow place revisiting without corrupting the whole map. Indeed, the same place mapped at two distinct moments will

not overlap on the manifold. In practice, the manifold is discretized into successive 2-dimensional *patches* corresponding to the fusion of data integrated during a time interval. We extend this representation in a 3D space to manage a TSDF-based map. Under the hypothesis that the drift of the localization algorithm is small, we can consider that data integrated during a limited time/space window will remain consistent enough to provide a good local reconstruction. With this idea in mind, a patch can be seen as a temporal sub-map with a finite lifetime. During this lifetime, the active patch will be used to integrate sensor data before being deactivated, stored and replaced by a new patch. The lifetime of a patch can be defined as the number of depth maps integrated in the TSDF, the distance or the angular range covered by the sensor. A 3D pose (6 DoF) $T_{\mathcal{W}\leftarrow i} \in \mathbb{SE}(3)$ is associated to each patch $\mathcal{P}_i$, to define a local coordinate system with respect to a global reference frame $\mathcal{W}$. This pose is fixed until the patch becomes inactive and does not receive data anymore. It can then be moved freely without affecting the other parts of the map, either to correct an inaccurate pose estimation, or to change the reference frame of the map.

IV. Mapping System

A. Mono-robot mapping process

At each integration step, range measurements and their associated pose $T_{\mathcal{W}\leftarrow DM}$ are sent to a *Patch Manager* which integrates the information in the corresponding patch. If the current patch is terminated, a new patch i is created and its pose $T_{\mathcal{W}\leftarrow i}$ is set to $T_{\mathcal{W}\leftarrow DM}$. The pose of the measurement is transformed into the reference frame of the patch as

$$T_{i\leftarrow DM} = (T_{\mathcal{W}\leftarrow i})^{-1} \cdot T_{\mathcal{W}\leftarrow DM} \tag{2}$$

and the TSDF is updated by projecting the voxels in the depth map using (1). When a patch reaches the end of its lifetime and is deactivated, its TSDF is filtered and all voxels with a weight value under a given threshold are deleted. This filter aims at discarding artifacts caused by sensor errors, in order to get the best reconstruction possible and to reduce memory usage. It is worth noting that during the mapping process, surface extraction or TSDF interpolation are not required, which is a key point to ensure real-time performances.

B. Multi-Robot distributed mapping

We take advantage of the manifold representation to propose a multi-robot distributed mapping system, whose architecture is depicted in Fig. 2. This system relies on the exchange of aligned and co-localized patches between the different robots. The mapping stack is based on two *Patch Managers*, one is called *public* and the other *private*. Each Patch Manager stores a set of patches as defined in Section III-B, using the OpenChisel library to handle the TSDF part. The private map stores the patches created by its host robot. The public map is composed of all the patches created by all the other robots, expressed in the same reference frame. When a patch is broadcasted by a robot, each receiving one adds this patch to its public map. Each patch is identified by a couple (R_{ID}, P_{ID}), where

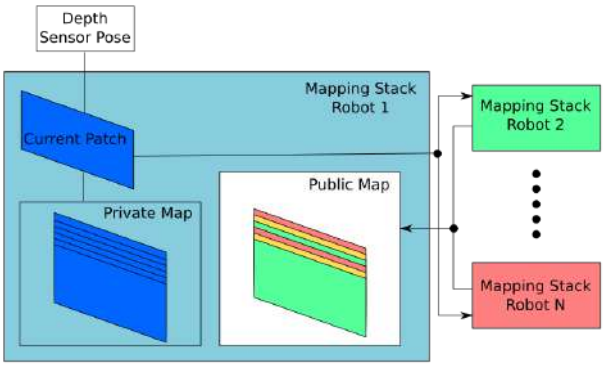


Fig. 2: Architecture of the multi-robot mapping system

R_{ID} is a unique robot ID and P_{ID} is the patch ID, unique in the robot's private map. To broadcast the update of a patch, we serialize only updated voxels of the TSDF. When deserializing the TSDF, if the couple (R_{ID}, P_{ID}) already exists in the public map, the corresponding patch is updated. This reduces the size of the messages sent over the network and prevents multiple propagation of the same information. If for some reason the robots do not share the same reference frame initially, the alignment process is carried out on the private patches only. When a common reference frame is set, all the corrected patches can then be serialized and sent to the other robots over the fleet network for integration in each public map.

C. Patch fusion

When revisiting an area that has already been mapped by the distributed system, the drift accumulated by the external localization module may yield an inconsistent reconstruction of the scene. When a patch is terminated, axis-aligned bounding boxes (AABB) are used to find the patch from the public map with the most important overlapping area, then a model-to-model alignment [21] based on the iterative closest point algorithm [22] is applied to correct the pose of the aligned patch. Using the public map to find the best patch means that a robot can apply a correction based on what the other robots have mapped, thus improving global consistency. Algorithm 1 summarizes the patch alignment procedure, which runs in parallel of the mapping process.

To propagate the corrections to further patches, we combine all corrections into a single transformation T_{corr}. Before alignment, each patch is pre-corrected by T_{corr} to help the alignment process to converge. This method only applies *posterior* corrections to the map, leaving localization unchanged. T_{corr} could also be used in a more integrated SLAM scheme so as to be consistent with the reconstructed map, e.g. for obstacle detection and path planning.

Since each patch is expressed in a local reference frame, an interpolation of the candidate patch is required before running the fusion process. Indeed, to perform the fusion of patch $\mathcal{P}_j$ into patch $\mathcal{P}_i$, the system has to find the distance and weight values of $\mathcal{P}_j$ at the centers of the voxels of $\mathcal{P}_i$. These values can then be processed with the standard

Algorithm 1: Patch alignment procedure

T_{corr} is initialized to the identity;
foreach *new candidate patch* $\mathcal{P}_i$ **do**
  Correct $\mathcal{P}_i$ by T_{corr};
  Find the best reference patch $\mathcal{P}_j$ with AABB;
  Interpolate $\mathcal{P}_i$ into $\mathcal{P}_j$;
  Extract the surfaces of each part in the overlapping area using marching cubes [20];
  Compute the correction transform $T_{\mathcal{W}\leftarrow\mathcal{W}'}$ by ICP;
  if *ICP converges* **then**
    Correct $\mathcal{P}_i$ by $T_{\mathcal{W}\leftarrow\mathcal{W}'}$;
    $T_{corr} \leftarrow T_{\mathcal{W}\leftarrow\mathcal{W}'} \cdot T_{corr}$;
    if $\mathcal{P}_i$ *and* $\mathcal{P}_j$ *have enough overlap* **then**
      Fuse $\mathcal{P}_i$ into $\mathcal{P}_j$;

integration model (1). Each voxel center $\zeta_i^i \in \mathbb{R}^3$ of $\mathcal{P}_i$ expressed in its own reference frame is projected into the reference frame of $\mathcal{P}_j$ to obtain ζ_j^i as follows:

$$\begin{bmatrix} \zeta_j^i \\ 1 \end{bmatrix} = (T_{\mathcal{W}\leftarrow j})^{-1} \cdot T_{\mathcal{W}\leftarrow i} \cdot \begin{bmatrix} \zeta_i^i \\ 1 \end{bmatrix} \quad (3)$$

Then the values of distance and weight of ζ_j^i in $\mathcal{P}_j$ are computed by trilinear interpolation and integrated into $\mathcal{P}_i$. $\mathcal{P}_j$ is then deleted to limit memory usage. This process can run in background at a lower frequency: delayed correction and fusion of the patches do not affect the overall consistency of the map.

D. Occupancy estimation

In order to be used for obstacle detection and path planning by each robot, the map should provide information about occupied and free space. The TSDF can be used to derive this information and the mapping system exposes a set of services providing the distance and direction to the closest object from a given position. An ellipsoidal formulation is proposed to deal with anisotropic robot characteristics. A local volumetric treatment is necessary to provide the distance to the nearest obstacle (equivalent to an EDT [3] or an ESDF [9]). An AABB is defined around the queried position, with horizontal semi-axis d_{des} and vertical semi-axis $\alpha_z d_{des}$, where d_{des} is the desired safety distance (robot size with an additional margin) and α_z the ratio between horizontal and vertical dimensions of the robot. A collision is then reported if the following Mahalanobis distance condition is met for at least one voxel:

$$n_{obs}{}^T \cdot \mathrm{diag}\left[d_{des}^{-2}, d_{des}^{-2}, (\alpha_z d_{des})^{-2}\right] \cdot n_{obs} \leq d_{obs}^{-2} \quad (4)$$

where n_{obs} and d_{obs} are the unit direction and the distance from the queried position to the closest obstacle, given by the TSDF within truncation distance (either directly or by the closest occupied voxel). This procedure was interfaced with a LazyPRM* planner from the *OMPL* library [23] and was able to compute feasible paths within 1 second on the embedded platforms (see Section V-C). The query process runs in parallel of the main mapping thread to prevent any access conflict.

V. Experimental results

A. Platforms, dataset and map configuration

The proposed mapping system was tested experimentally with a fleet of Wifibot Lab V4 wheeled terrestrial robots (see left of Fig. 8). They were equipped with Intel NUC 7 embedded computers and a stereo-rig of baseline 26 cm between two synchronized calibrated cameras (IDS UL124xLE) providing 640x512 px images at 20 Hz. The robots were connected together using the 5 Ghz band of a WiFi router (TP-Link Archer-C7). The same ROS software stack was run in real time by each robot on its on-board computer: eVO [18] for visual odometry at 20 Hz (less than 2% drift on the Kitti benchmark) and image rectification, ELAS [19] for computing dense depth maps at 5 Hz (distances between 0.5 m and 5 m were taken into account), and the distributed mapping system which used these inputs. The TSDF parameters were set to a voxel resolution of 5 cm and a truncation distance of 50 cm. The creation of new local patches was triggered by displacement thresholds of 3 m in translation or 90° in rotation. The depth integration takes about 100 ms per frame and the *patch* alignment and fusion takes between 1 and 15 s, but this process is not time critical and is processed in another thread. To facilitate information exchange between the robots, the multi-master node manager system [24] was used. This software shares ROS topics between different computers, with clock synchronization from a common NTP reference.

The system was tested under two representative use cases. In the first one, two robots were remotely operated in a large SNCF (French National Railway Corporation) storage area (50 m x 25 m) in Sotteville-Lès-Rouen, France. In the second case, three robots cooperated in order to realize a fully autonomous exploration mission in an underground car park (22 m x 15 m). In this scenario, the mapping stack was connected to autonomous navigation algorithms achieving real-time path planning. In order to define a common world frame shared by all the robots, a vision-based co-localization method was applied once in each mission. We used a cube of AprilTag [25] with known relative poses between its sides (see Fig. 8 bottom right), but any other strategy could be employed. The robots were co-localized at the start of the mission in the second use case, while in the first one this was carried out around the middle of the trajectory. This demonstrated that the mapping system, which maintains local patches, can work locally and then reconfigure itself in a global frame when co-localization is achieved on the fly.

B. Map alignment and cooperative mapping

A single-robot loop was replayed twice to simulate place revisiting and evaluate the ability of the system to align the map in order to achieve a consistent fusion. Fig. 4 shows the reconstruction obtained on the double loop with the initial trajectory and the one corrected by back-propagating the map alignment transforms on visual odometry data. As

Fig. 3: Snapshot of *Colmap* ground-truth reconstruction

shown in Fig. 4, the initial trajectory (in red) suffers from accumulated drift which can be critical in a narrow zone. As seen on Fig. 5, a successful alignment between the patches created during the first loop and the new patches created during the second loop was achieved. This is used to correct the pose for the second loop, leading to a more consistent reconstruction. The drift of the original trajectory is of about 1 m, and the drift of the corrected trajectory is reduced to 0.35 m, for a global length of 104 m. A ground-truth 3D reconstruction of the environment (Fig. 3) was realized with *Colmap* [26], a *structure from motion* software which operated on a complete set of 1150 images acquired in high-definition (1280x1024 px) for this purpose. Fig. 6 presents the mapping accuracy compared with the GT using CloudCompare[2]. The resulting mean error is about 7 cm.

The distributed multi-robot mapping system was tested in the same environment. Two robots initially built their own map in their relative frame, until the co-localization AprilTag cube was detected around the middle of the trajectories. Fig. 8 shows the two trajectories. Both robots start from the top right corner of the zone and perform a loop of about 65 m length. The initial trajectory of Robot 1 is displayed in black and its corrected trajectory in blue. The initial trajectory of Robot 2 is in red and its corrected trajectory in green. In this example, it can be seen that the alignment module takes advantage of the patches sent by the other robot, since significant corrections are applied after the co-localization event by aligning the reconstruction with the other robot map.

C. Multi-robot autonomous exploration

In the second use case, a fleet of three robots exploited the obstacle distance service described in Section IV-D to carry out a fully autonomous exploration. Here, the robots shared a common AprilTag initial reference frame and then followed a frontier-based exploration strategy under the constraint of obstacle avoidance to cover the entire area.[3]

A dedicated multi-robot system (Fig. 7) has been designed, with the same suite of distributed algorithms executed on-board of each robot. The exploration strategy relies on

[2]CloudCompare: `https://www.danielgm.net/cc/`

[3]A video can be found at https://tinyurl.com/CopernicExplo

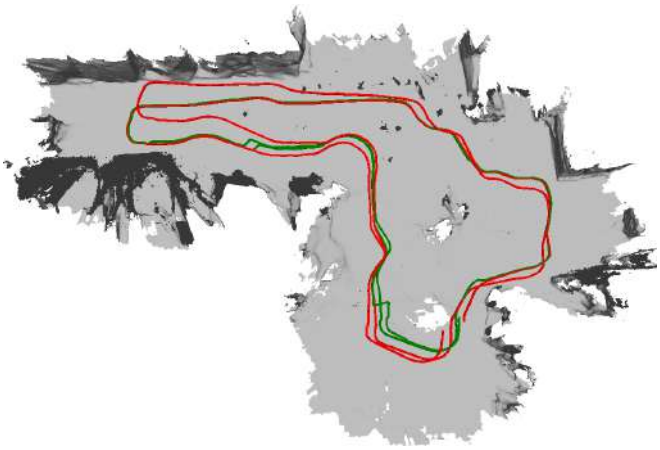
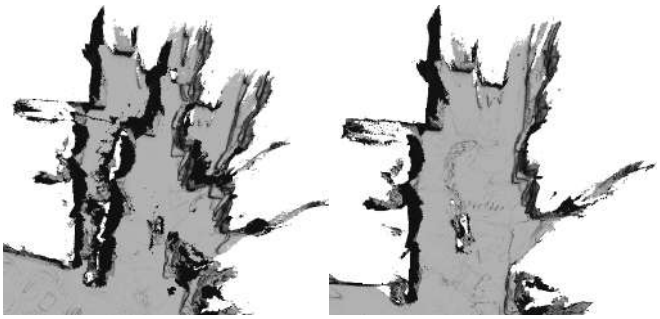


Fig. 4: Reconstruction and correction of the trajectory of one robot on a repeated loop over the same area. In red : original eVO trajectory. In green : corrected trajectory.



(a) (b)

Fig. 5: Detail of a corridor mapped twice sequentially. (a) Before patch alignment, the corridor is inconsistent due to localization drift. (b) After patch alignment, the corrected map is much more consistent.

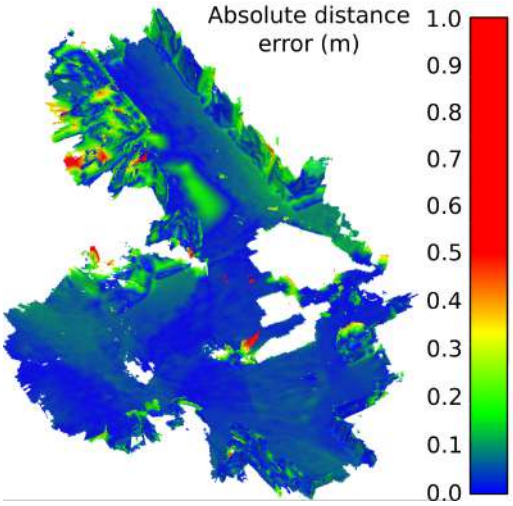


Fig. 6: Comparison of reconstructed map and GT mesh

the TSDF environment model proposed, which has been associated to a 3D grid to determine *next best views* (NBV) and find a feasible path at the same time. The exchange of information between robots is managed at the level of each algorithm and is mostly event-based:

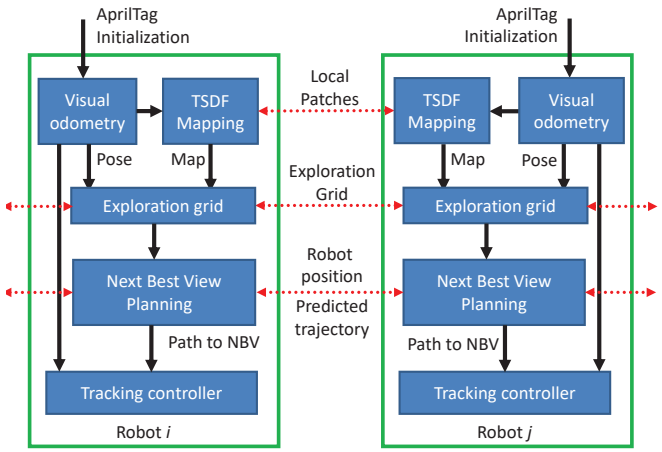


Fig. 7: Architecture of the multi-robot exploration system

- The local patches of the TSDF map are broadcasted by each robot to the rest of the team as soon as they are completed (under distance or orientation conditions).
- The exploration grid is broadcasted by each robot to the team once a new NBV has been planned.
- The robot current position and predicted trajectory are exchanged at a fixed high frequency (control loop rate) in order to be taken into account in planning and control.

A complete communication graph was used during the experiments, however the architecture is fully distributed therefore it remains valid if some of the robots cannot directly exchange information. The only prior information available is the dimensions of the axis-aligned box to be covered with respect to an initial global frame. A single 3D grid manages the exploration progress, the computation of the frontier, the occupancy (in relation with the TSDF environment model) and the intentions of the robots of the team for better cooperation. The following four concurrent tasks are carried out by each robot:

- At a fixed frequency [10 Hz]: Update newly explored cells in the grid using current pose and sensor geometry.
- When the time t_{go} to reach the current NBV has elapsed: Compute a new NBV using Algorithm 2.
- On reception of an exploration grid from another robot: merge information in its own grid.
- At control rate [20 Hz]: Follow the path to the NBV.

The exploration progress is monitored in 3D space, this way, it can be used for either 2D or 3D navigation (and it is thus ready for heterogeneous teams). The evolution of the exploration is tracked using a 3D dense voxel map that encodes the status of each cell among the following set of tags: {Explored, Unexplored, Occupied, Border, Intentions}. Each voxel stores a status S represented by a 32-bit vector, where each bit is a flag defined as follows:

bit #	0	[1, 26]	[27, 30]	31
Flags	Explored	Border	Robot Intention	Occupied

Cells are initially unexplored and are set as explored if they are located inside the polyhedral model of the embedded sensor (e.g. a square pyramid) located at the current pose

Algorithm 2: Computation of the NBV

```
Update obstacles in exploration grid by requests (4);
Compute the new set B of frontier voxels;
foreach Frontier position p_i in border set B do
    Call Lazy-PRM* to find a path from the current
     robot position in free explored space[4];
    if Feasible path to p_i found then
        foreach Candidate orientation ψ_j do
            Evaluate utility U of view ξ_ij = [p_i, ψ_j]^T;
            Set cost C(ξ_ij) as the path length to p_i;
Select the NBV ξ* in the list of feasible views with (5);
Smooth the path corresponding to this NBV and send it
  to trajectory tracking;
Clean the old and set the new self-intentions in 3D grid;
Share 3D grid with the other robots;
```

of the robot. The exploration utility is computed with the exact same function at a given candidate view, and is equal to the number of unexplored voxels that can be observed (considering the intentions of the other robots as already explored). The border set (i.e. the free cells that are located at the border of any unexplored cell) is managed by updating the 26 neighbours (in 3D) of a cell when it becomes explored. Each time a NBV is computed, the intention bit fields corresponding to the current robot are updated and the grid is broadcasted to the other robots of the team.

The exploration strategy is frontier-based [27], and the following cost function is considered to choose the NBV:

$$J(\xi) = U(\xi)\exp(-\beta C(\xi)) \tag{5}$$

where $\xi = [x, y, z, \psi]$ is a *view* on the exploration frontier comprising a 3D position and a yaw in inertial frame, U the utility or gain, C the cost and β a weighting coefficient. This cost function, initially defined in [28], achieves a trade-off between the utility of the view (here, the new volume seen given the sensor model and the intentions of the other robots) and the cost to reach this location (here, the distance to reach the view given the obstacle map). The cost to each border voxel is evaluated using the Lazy-PRM* path planning algorithm [29]. By construction, this strategy is efficient for multi-query requests using a single graph, therefore it is better suited for the frontier-based approach considered (which can be seen as a single-start multiple-goals problem), than e.g. its RRT* counterpart [30]. The any-time characteristics and the low computational cost of Lazy-PRM* are also relevant, since we want to check the feasibility and compare the path lengths towards all the border nodes within a limited computation time. A unique PRM graph is thus generated in the position space (2D at the current altitude for a mobile robot in this work, but the 3D case for aerial vehicles can also been taken into account), and multiple queries are performed from

[4]Lazy-PRM* is granted 1 s for the first frontier point (when the graph is built), then 1 ms for the remaining ones

the start location (current robot position) to all the current candidate border points (Fig. 9a). The planner thus returns the feasibility to find a path toward each border point ξ_i and for the feasible ones, the length of the path is taken as cost value $C(\xi_i)$. The collision checks are performed from the states of the exploration grid, a position is considered feasible if it lies in a non-occupied explored voxel. This ensures a safe navigation, in the spirit of the frontier-based approach where the robot navigates only in the known free space. The current positions of the other robots of the team are also taken into account in collision checking. The current intentions of the other robots are considered as already explored by the robot computing its NBV (Fig. 9b). For each reachable cell of the border determined as described above, a discretized number of candidate orientations are evaluated (8 yaws ψ in our tests). Following these discretized evaluations of distance cost and utility, the pose ξ^* containing the coordinates of the border cell and the orientation which maximizes the cost function (5) is selected as the NBV to be reached. The path from the start to this position is smoothed using B-splines to better respect motion constraints, and sent to a robot trajectory tracking controller (derived from [31]). This way, the trajectory to be followed (with a reference speed chosen equal to 0.4 m/s) is consistent with the current state of the exploration grid.

Five experiments were performed for the case of a single robot (to serve as a baseline), and then for teams of two and three robots, with randomized initial positions. The mission was stopped when the volume coverage reached 90%. The metrics computed are those reported in previous works: evolution of the coverage with time and distance covered by each robot. The performances and gains (around 1.5 for each new robot added to the team) reported in Table I compare favourably with those reported in real experiments (in 2D with laser scanners [32]) or in simulation [33]. This confirms the applicability and relevance of this distributed multi-robot vision-based system with the proposed TSDF model in the loop. Fig. 1 shows the mesh reconstructed by the TSDF and in particular the respective contribution of each robot to the shared global map, which was reconstructed by each of them on its own embedded computer (nothing was processed on a ground station). The cooperation strategy allowed the robots to share the task of information gathering, and the mapping system thus allowed them to plan their own path using locations previously covered by the other robots. Fig. 11 presents the equivalent occupancy grid (with safety margins for autonomous navigation) generated by querying the TSDF map over a 20 cm resolution grid, and an example of planned trajectory avoiding several obstacles. Fig. 10 presents the supervision view of the exploration mission.

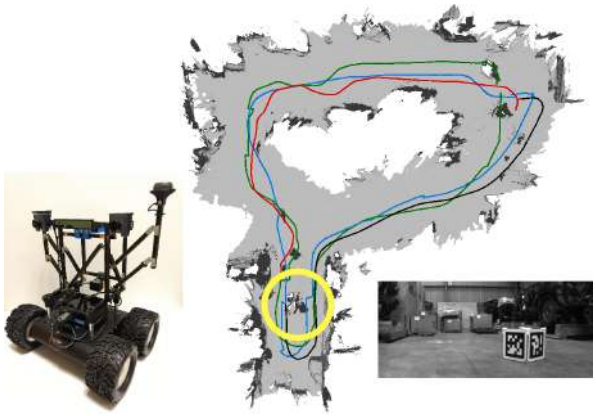
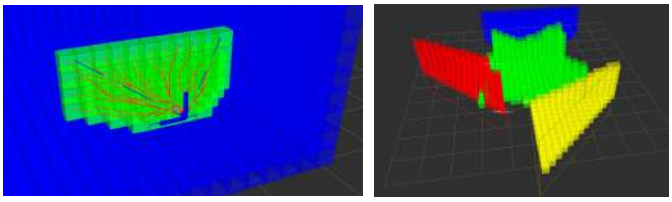


Fig. 8: Global reconstruction of the same zone mapped in parallel by two robots (Wifibot Lab V4, on the left)



(a) Lazy-PRM* path generation (red) towards all border cells through explored space (green), unexplored space in blue.

(b) NBV intentions of robots 2 and 3 (blue and yellow) received by robot 1 (red). Explored space in green.

Fig. 9: Exploration grid: planning and intentions

TABLE I: Metrics and gains w.r.t. 1-robot baseline (Averaged over five experiments)

	Time (s) to cover 90%	Gain w.r.t. 1 robot	Distance (m) per robot	Gain w.r.t. 1 robot
1 robot	281.11	1	77.53	1
2 robots	168.59	1.67	48.95	1.58
3 robots	129.25	2.17	33.92	2.29

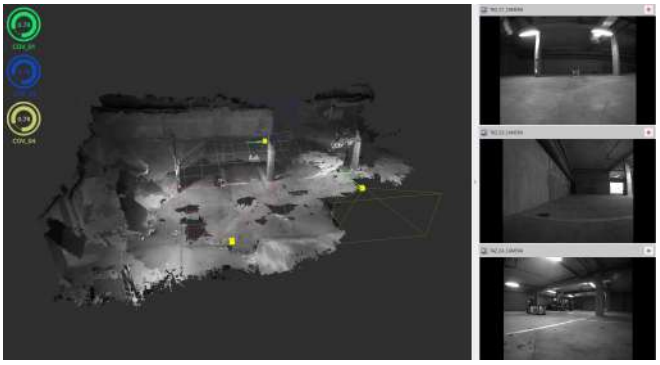
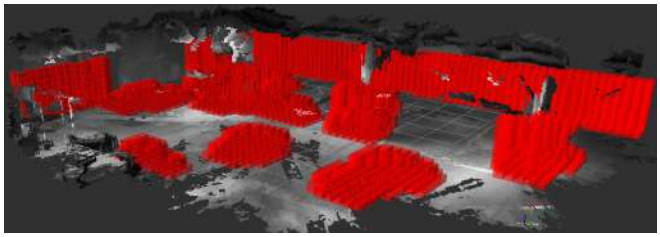


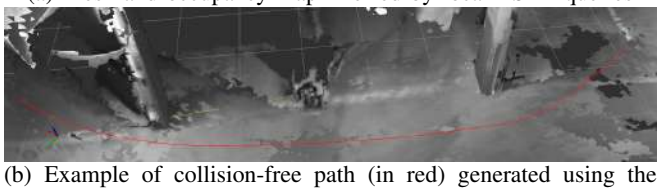
Fig. 10: TSDF environment model built on-board during exploration. Embedded camera views on the right side.

VI. Conclusions

This paper has described a multi-robot distributed 3D TSDF mapping approach based on a manifold structure and a representation as a collection of local *patches*. It contains a patch-to-patch alignment strategy coupled with a fusion algorithm, which are able to improve the consistency of the map and can also be used to carry out posterior corrections on the localization of the robots. The distributed nature of



(a) Mesh and occupancy map inferred by local TSDF queries



(b) Example of collision-free path (in red) generated using the proposed mapping system

Fig. 11: TSDF map for autonomous robot navigation

the system is provided by the management – at the level of each robot – of a private map where the patches are created using depth information from the embedded sensors, and a public map where all the patches received from the other robots are integrated. Experiments conducted with a fleet of wheeled terrestrial robots successfully validated the abilities to perform map correction and also interact with planning algorithms for autonomous navigation and exploration in areas mapped cooperatively. Future work will consider the integration of additional sensors (RGB-D, laser) and field tests in larger areas with heterogeneous fleets comprising terrestrial and aerial robots.

ACKNOWLEDGMENT

This work was supported by ONERA projects Carnot-CODA and PR-GUIMAUVE. The authors thank SNCF Réseau for granting access to the experimental area.

REFERENCES

[1] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, pp. 46–57, June 1989.

[2] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[3] B. Lau, C. Sprunk, and W. Burgard, "Efficient grid-based spatial representations for robot navigation in dynamic environments," *Robotics and Autonomous Systems*, vol. 61, no. 10, pp. 1116–1130, 2013.

[4] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "KinectFusion: Real-time dense surface mapping and tracking," in *10th IEEE ISMAR, Basel, Switzerland*, pp. 127–136, 2011.

[5] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *SIGGRAPH*, (New York, NY, USA), pp. 303–312, ACM, 1996.

[6] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald, "Kintinuous: Spatially extended KinectFusion," in *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, (Sydney, Australia), 2012.

[7] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, "Real-time 3D reconstruction at scale using voxel hashing," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 6, 2013.

[8] M. Klingensmith, I. Dryanovski, S. Srinivasa, and J. Xiao, "Chisel: Real time large scale 3D reconstruction onboard a mobile device using spatially hashed signed distance fields," in *Robotics: Science and Systems XI, Roma, Italy*, 2015.

[9] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox: Incremental 3D Euclidean signed distance fields for on-board MAV planning," in *IEEE/RSJ IROS, Vancouver, Canada*, pp. 1366–1373, 2017.

[10] E. Vespa, N. Nikolov, M. Grimm, L. Nardi, P. H. J. Kelly, and S. Leutenegger, "Efficient octree-based volumetric SLAM supporting signed-distance and occupancy mapping," *IEEE Robotics and Automation Letters*, vol. 3, pp. 1144–1151, April 2018.

[11] A. Millane, Z. Taylor, H. Oleynikova, J. Nieto, R. Siegwart, and C. Cadena, "C-blox: A scalable and consistent TSDF-based dense mapping approach," in *IEEE/RSJ IROS, Madrid, Spain*, pp. 995–1002, 2018.

[12] A. Howard, "Multi-robot mapping using manifold representations," in *IEEE ICRA, New Orleans LA, USA*, vol. 4, pp. 4198–4203, 2004.

[13] A. Birk and S. Carpin, "Merging occupancy grid maps from multiple robots," *Proceedings of the IEEE*, vol. 94, pp. 1384–1397, July 2006.

[14] Y. Liu, X. Fan, and H. Zhang, "A fast map merging algorithm in the field of multirobot SLAM," in *The Scientific World Journal*, 2013.

[15] C. Forster, S. Lynen, L. Kneip, and D. Scaramuzza, "Collaborative monocular SLAM with multiple micro aerial vehicles," in *IEEE/RSJ IROS, Tokyo, Japan*, pp. 3962–3970, 2013.

[16] P. Koch, S. May, M. Schmidpeter, M. Kühn, *et al.*, "Multi-robot localization and mapping based on signed distance functions," *Journal of Intelligent & Robotic Systems*, vol. 83, no. 3-4, pp. 409–428, 2016.

[17] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[18] M. Sanfourche, V. Vittori, and G. Le Besnerais, "eVO: A realtime embedded stereo odometry for MAV applications," in *IEEE/RSJ IROS, Tokyo, Japan*, pp. 2107–2114, 2013.

[19] A. Geiger, M. Roser, and R. Urtasun, "Efficient large-scale stereo matching," in *ACCV, Queenstown, New Zealand,*, pp. 25–38, 2010.

[20] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," in *SIGGRAPH Computer Graphics*, vol. 21, pp. 163–169, ACM, 1987.

[21] N. Fioraio, J. Taylor, A. Fitzgibbon, L. Di Stefano, and S. Izadi, "Large-scale and drift-free surface reconstruction using online subvolume registration," in *IEEE CVPR, Boston, MA, USA*, 2015.

[22] R. Rusu and S. Cousins, "3D is here: Point cloud library (PCL)," in *IEEE ICRA, Shanghai, China*, 2011.

[23] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012. `http://ompl.kavrakilab.org`.

[24] A. Tiderko, F. Hoeller, and T. Röhling, "The ROS multimaster extension for simplified deployment of multi-robot systems," in *Robot Operating System (ROS)*, pp. 629–650, Springer, Cham, 2016.

[25] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *IEEE ICRA, Shanghai, China*, pp. 3400–3407, 2011.

[26] J. L. Schönberger and J.-M. Frahm, "Structure-from-motion revisited," in *IEEE CVPR, Las Vegas, NV, USA*, 2016.

[27] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *IEEE International Symposium on Computational Intelligence in Robotics and Automation, CIRA'97*, pp. 146–151, 1997.

[28] H. H. González-Banos and J.-C. Latombe, "Navigation strategies for exploring indoor environments," *The International Journal of Robotics Research*, vol. 21, no. 10-11, pp. 829–848, 2002.

[29] K. Hauser, "Lazy collision checking in asymptotically-optimal motion planning," in *IEEE International Conference on Robotics and Automation, Seattle, WA, USA*, pp. 2951–2957, 2015.

[30] S. Song and S. Jo, "Online inspection path planning for autonomous 3D modeling using a micro-aerial vehicle," in *IEEE International Conference on Robotics and Automation, Singapore*, pp. 6217–6224, 2017.

[31] M. Bak, N. K. Poulsen, and O. Ravn, *Path following mobile robot in the presence of velocity constraints*. IMM, Informatics and Mathematical Modelling, The Technical University of Denmark, 2001.

[32] W. Burgard, M. Moors, C. Stachniss, and F. E. Schneider, "Coordinated multi-robot exploration," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 376–386, 2005.

[33] A. Mannucci, S. Nardi, and L. Pallottino, "Autonomous 3D exploration of large areas: a cooperative frontier-based approach," in *International Conference on Modelling and Simulation for Autonomous Systems, Rome, Italy*, pp. 18–39, 2017.