On the Fly Plane Detection and Time Consistency for Indoor Building Wall Recognition Using a Tablet Equipped With a Depth Sensor

Adrien Arnaud, Michèle Gouiffès, Mehdi Ammi

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 24, 2018, accepted March 13, 2018, date of publication March 21, 2018, date of current version April 23, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2817838

On the Fly Plane Detection and Time Consistency for Indoor Building Wall Recognition Using a Tablet Equipped With a Depth Sensor

ADRIEN ARNAUD, MICHÈLE GOUIFFÈS, AND MEHDI AMMI
LIMSI-CNRS, Universitè Paris-Saclay, F-91405 Orsay, France
Corresponding author: Adrien Arnaud (adrien.arnaud@limsi.fr)

ABSTRACT This paper presents an on the fly planar segmentation algorithm that runs on a tablet equipped with a depth sensor and which uses a motion tracking algorithm. Our algorithm segments each incoming point cloud from the depth sensor and it then updates a global model containing all of the previously identified planes. Consequently, identical planes are identified in successive frames. We use a fast segmentation algorithm that generates and merges smaller intermediate clusters to identify all of the planes contained in the incoming point cloud. We then give each plane a unique ID by computing an histogram of its parameters. These IDs are used as a key for storage in a hash map. Identifying similar planes in different frames will enable us to update the plane's borders and will serve to identify the walls of an indoor scene. Our algorithm enables us to perform a 3-D planar segmentation of a point cloud that is issued from a depth sensor in less than 200 ms. Moreover, we are able to estimate the maximal size of a room with a mean error inferior at 10%. This will serve as a basis to develop a 3-D reconstruction algorithm that can automatically generate, in real time a 3-D editable model of an existing building. The generated 3-D model will contain the principal structural elements (i.e., walls, doors, and windows) of the building. This algorithm has a a number of applications, from simple 3-D modeling to building energetic performance assessment.

INDEX TERMS Computer vision, mobile devices, planes detection, time consistency.

I. INTRODUCTION

The digital mockup of buildings (or BIM: Building Information Modeling [1]) has become a key element of the lifecycle of buildings. This includes all of the related information, from the design to the management of the buildings. Recently, BIM has became the norm in the industry and it is particularly suited for energetic renovation works. While a BIM model is made at the conception stage for newer buildings, it also has to be done for old buildings for which construction plans have not been kept. However, generating a 3D model of a building can become a fastidious task to perform when manually making the measurements. Consequently, the 3D modeling process tends to be automatized.

Professional solutions already allow us to perform a 3D scan of an existing indoor building environment, such as DotProduct's DPI-8 scanner[1] and FARO's Focus 3D series of sensors.[2] Highly accurate point clouds of an existing indoor building environment can be generated with these tools. These point clouds can then be processed to extract the building's geometry, which can then be enhanced to generate a complete BIM mockup.

Previous works have already studied the automation of BIM mockup generation from an indoor building point cloud [2], [3]. However, the process is partially automatic and only the principal edges of the building are detected to simplify the modeling process. These solutions rely on professional tools that are not affordable for small companies or individuals. Moreover, constructing a highly accurate 3D model often takes a lot of time. For some applications, there is no need for a highly detailed 3D model and only the main structural elements (i.e., walls, windows, and doors) are needed, such as for the energetic performance evaluation of a building or to take measurements to do subsequent works.

[1] http://www.dotproduct3d.com/DPI8.php

[2] http://www.faro.com/en-us/products/3d-surveying/faro-focus3d/overview

The development of more affordable depth sensors, such as the Microsoft Kinect, allows the modeling of smaller scale buildings at a lower cost. Depth sensors are now even available on tablets, such as Apple's StructureSensor,[3] Stereolabs's Zed sensor,[4] and the Google Tango project.[5]

Our work focuses on using a tablet equipped with a depth sensor to generate a 3D editable model of an existing indoor environment. Our basic approach consists in generating a 3D mesh of the building in real time. This approach has been developed using a Kinect sensor [4], [5] and it has been adapted with a tablet [6], [7]. A 3D segmentation of the generated mesh can then be performed to identify the structure of the building. This approach has been investigated in [8]. Unfortunately, in this previous study, to make the 3D mesh generation run in real time, the accuracy of the mesh generation had to be reduced, which affects the results of the post-processing. Moreover, the identification of the openings is limited because these are viewed as empty rectangular shapes on a wall, which cannot be discriminated clearly enough in some scenarios.

From this observation, our reflection has turned to a new methodology which, instead of generating a 3D mesh and then using it to identify the structural elements, performs the identification on the fly from the rough data provided by the device. We can use all of the information available from the tablet's sensors, including the depth map, pose estimation, and the full RGB image. To make this work, the different structural elements observed by the tablet have to be identified in each frame and then matched to previously identified structural elements. This approach has two advantages: first, a larger amount of information is available to distinguish the structural elements; and second, the storage of a heavy 3D mesh is no longer required, which saves precious time and memory resources. The following reasonable assumption is made to simplify the analysis: the walls are planar and orthogonal to the ground.

This paper presents the first step of our 3D modeling algorithm. The algorithm performs a real-time planar segmentation of a 3D point cloud which runs on a tablet equipped with a depth sensor and it then simultaneously matches corresponding planes between successive frames. A user simply scans an existing indoor building environment (see Fig.1) and a list of all the planes contained in the building is generated. This algorithm relies on a quick planar segmentation and it uses a histogram to conduct noise resistant planes estimation. It is able to perform the planes detection for a frame in less than 200*ms* and it reaches a precision of around 10% in placing the planes. In this study, this algorithm will serve as a basis for detecting walls on the fly.

Th rest of this paper is organized as follows. A review of previous studies is made in Section II. Our algorithm is then described in Section III. Finally, the implementation of the tablet is discussed in IV and an evaluation is carried out in SectionV.

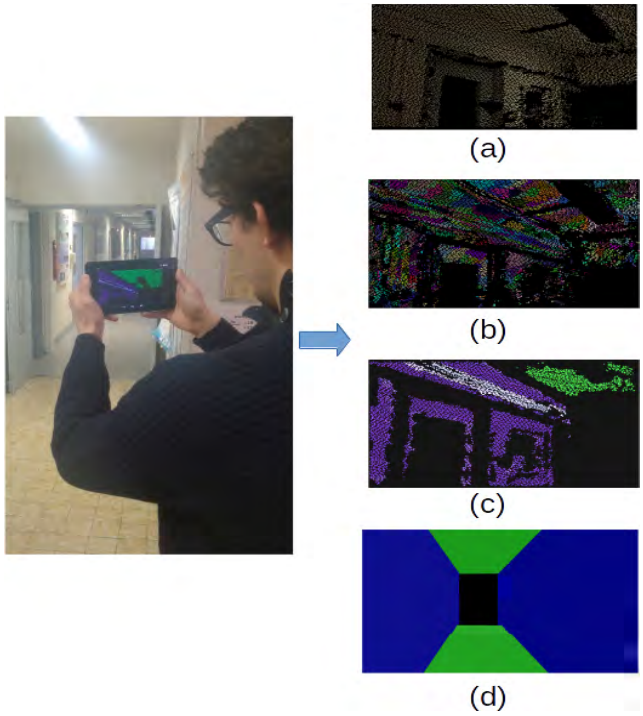


FIGURE 1. Our algorithm works on a tablet equiped with a depth sensor. Users can scan an existing environment, and at each frame, the input point cloud (a) is segmented into a set of small clusters (b), and these clusters are merged (c). Similar planes through different frames are merged so it is possible to extract the building's geometry (d).

II. RELATED WORKS

The following review focuses on the two main issues addressed by our algorithm, namely the planes detection in a 3D point cloud and the alignment of data from different frames.

A. PLANAR SEGMENTATION

Planar segmentation is often performed by finding the normal vectors at each point. The normal vectors estimation requires that the k-nearest neighbours (KNN) are known for each point. Many solutions exist to find the KNN of a 2D point, such as [9]–[13], but this is immediate when using an ordered point cloud, such as when using structured light sensors. Otherwise, a k-nearest neighbours or a triangulation algorithm have to be used. The main approaches to compute the normals of a cloud are averaging based methods and optimization based methods [14], [15]. With the averaging methods, the normal for a point is estimated using weighted means; see, for example, [16]–[18]. Optimization based methods mainly exploit SVD methods [19], [20] over the KNN of a point to find the statistically best normal for this point. This results in highly accurate results [15].

Although planes can be detected using a 3D generalization of the Hough transform [21], [22], such as in [23], this

[3]https://structure.io/
[4]https://www.stereolabs.com/zed/
[5]https://developers.google.com/tango/

method is costly in terms of computation time and it can provide a poor precision. Alternatively, a basic approach to perform a 3D segmentation of a point cloud consists in using a RANSAC algorithm [24]–[26]. A first planar model is fitted to the input point cloud and this method is then recursively applied to the remaining points. RANSAC algorithms are easy to implement but can lead to wrong results when the parameters are not correctly set. Tarsha-Kurdi et al. [27] suggest that RANSAC is more accurate and faster than Hough transforms.

Holz et al. [28] perform a planar segmentation by first achieving a segmentation of the normals space and then doing a refinement to distinguish the parallel planes. The authors then used these results to detect obstacles and graspable objects in a scene, with a detection rate of up to 90%.

To accelerate the normals space segmentation, the task can be divided into smaller segmentation tasks by dividing the space into seed points and adapting the superpixels technique [29]. Erdogan et al. [30] achieve a planar segmentation of a depth map issued from a Kinect sensor. They generate superpixels and then merge them into the final planes by using Barbu et al.'s [31] generalization of the Swendsen-Wang sampler [32]. Papon et al. [33] define the *supervoxels* algorithm, based on the superpixels technique [29], which is adapted to perform a planar segmentation of a point cloud. The cloud is divided into seed voxels and 3D clusters are generated around these seeds by adding points considered as similar. Similarity between two points depends on their normal, their spatial position, and their color. The type of segmentation can be chosen by adjusting the algorithm's parameters.

B. TIME CONSISTENCY

Time consistency consists in identifying similar areas in RGB-D data taken at different times or points of view. Being able to identify similar regions can either serve to correct pose estimations, or it can identify shapes or similar objects in a 3D set.

This task is often performed by finding key points in the data set and then computing local features around these points, which are scale, translation, and rotation independent. These features convey information about the geometry of the neighborhood of the key point, such as the normal vectors and the curvature. Popular 3D feature algorithms extend well known 2D image processing feature algorithms, such as SIFT [34], [35], the Harris corner detector [36], [37] or SURF [38]. Although the Harris detector provides transformation invariant features, the are not scale invariant. SIFT features provide scale invariant features by using local gradient histograms. SURF features provide transformation and scale invariant features, and they are fast to compute. Flint et al. [39] extend the 3D SIFT and SURF features, and create the THIRFT features. These features use the normal vectors computed at different scales to produce scale and transformation invariant histograms. A more complete list of 3D point features can be found in [40]. Features histograms have been shown to be resistant to translations, rotations, and noise.

Our analysis of the previous studies shows that a superpixels algorithm is the best option to perform a real time planar segmentation with a tablet. Indeed, the normals can easily be computed using an average based method when the point cloud is ordered and can easily be vectorized. A rough planar segmentation can be performed because our work focuses on the detection of walls, which are large planes. Histograms of the planes' attributes can be sufficient to identify the same planes in different frames. In fact, the planes' attributes depend on their normal vector and distance from the origin. The use of an external motion tracking algorithm ensures that we have transformation invariant plane parameters that are sufficient descriptors for a plane object.

III. ON THE FLY PLANE DETECTION

This section details our plane detection algorithm. It is based on the hypothesis that we are scanning an indoor building environment and that the building structure is aligned with a Manhattan grid [41]. The plane detection is performed on the fly by exploiting the latest complete RGB-D information that is available from the camera and the depth sensor. Given that RGB-D data can be easily ordered if we know the sensor's intrinsic parameters, the complexity of the different algorithms can be significantly simplified. With these considerations, and by optimizing the code for the target architecture, we are able to perform plane and wall recognition in less than 200ms. Although our algorithm runs fully on the device's CPU, some parts can easily be implemented on the GPU of a compatible device.

A. OVERVIEW

Our algorithm processes each incoming point cloud in three main steps. First, different maps are computed using the RGB-D data, which includes an XYZ map, a color map, a normal map, an edge map, and a D-map. When speaking about a map, we mean that each point feature (position, color, normal, on an edge or not) is indexed with a (u, v) couple of coordinates. With this, we can instantly get the nearest neighbors of a given feature. The resolution of each map is the same; that is, the resolution of the depth sensor. Several different maps, as detailed in III-B, are used as parameters for the plane detection algorithm. These maps are then used to compute the planes contained on the current point cloud. The plane detection is performed in two steps. The space is first divided into smaller regions and an algorithm similar to superpixels [33] is used to generate multiple clusters. These clusters are then merged to obtain the final planes. The whole process is detailed in III-C and the different steps are depicted in Fig.2. The detected planes are then used to update the list of all of the previously detected planes; this process is detailed in III-D. Fig.3 shows the different modules that are used in our algorithm and their order of computation.

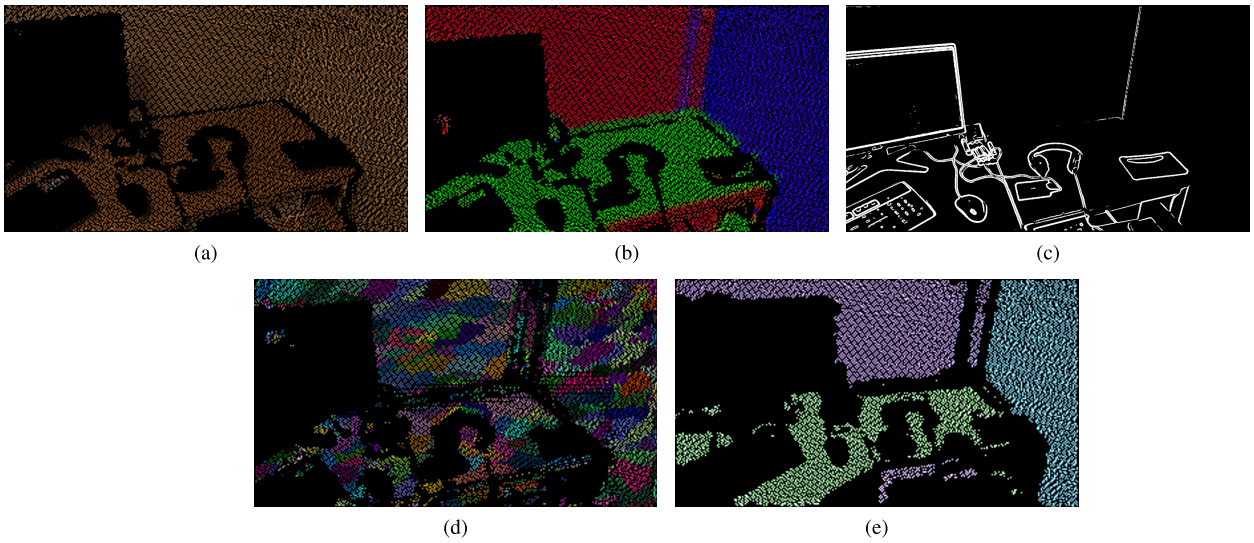


FIGURE 2. An example of input data with the resulting planar segmentation. All the data are ordered in the 2D space, that avoid the use of k-nearest neighbours algorithms. (a) Colored point cloud. (b) Computed normals map. (c) Computed edges map. (d) Generated clusters. (e) Final planes.

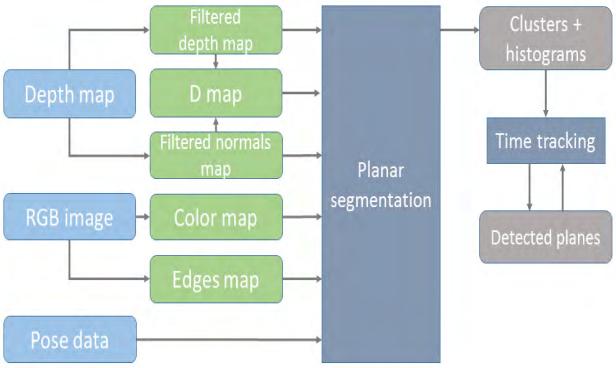


FIGURE 3. Overview of the segmentation algorithm.

B. DATA ACQUISITION

Our algorithm requires a depth map, the associated RGB frame, and a pose estimation of the device. RGB-D data are used to generate the different input maps and the pose estimation is used to estimate the different planes parameters in an absolute reference frame so that we can merge them with the planes that were previously detected. In order to keep reasonable computation time, the depth image can be downsampled, especially when there is no available GPU. The downsapling of the input depth map has no effect on the algorithm accuracy since it focuses on large planar surfaces detection. To reduce the amount of noise, a Gaussian filter is applied to the input depth map.

1) NORMAL ESTIMATION

Normal vectors are estimated using a sliding window algorithm that browses the depth map *depth_map*. The fact that the points are ordered avoids the need to use a k-nearest neighbors search algorithm and we can easily compute the neighborhood of each point of the map with a given radius *window_width*.

For each pixel $P_x(u, v)$ of map coordinates, we define a window W by :

$$W = \left\{ p_x(u,v) \in D\_map, \begin{array}{l} |u-U| \leq window\_width \\ |v-V| \leq window\_width \end{array} \right\} \quad (1)$$

W is divided into four quadrants, NW, NE, SW, SE as illustrated in Fig:5. For each $P_x \in Q_i$, where $i \in \{NW, NE, SW, SE\}$, we compute the vector $\overrightarrow{V_i}$ for each Q_i defined by :

$$\overrightarrow{V_i} = \frac{\sum_{p \in N_i} \overrightarrow{depth\_map(p)}}{|N_i|} \quad (2)$$

where $|.|$ denotes the cardinality. Then, the normal vector $\overrightarrow{N}$ for the point at $depth\_map(P_x)$ is defined by:

$$\overrightarrow{N} = \frac{(\overrightarrow{V_{SE}} - \overrightarrow{V_{NW}}) \wedge (\overrightarrow{V_{NE}} - \overrightarrow{V_{SW}})}{\| (\overrightarrow{V_{SE}} - \overrightarrow{V_{NW}}) \wedge (\overrightarrow{V_{NE}} - \overrightarrow{V_{SW}}) \|} \quad (3)$$

Although the use of a large window improves the normal estimation accuracy for planar areas, it also increases the computation times and the inaccuracies on the corner points by smoothing the normal estimation. To avoid this, we use a small window width and apply a Gaussian filter on the normal map to reduce the noise. Fig. 9b shows an example of normal map computed with our algorithm.

2) ESTIMATION OF THE EDGE MAP

Given that the segmentation algorithm that we use is a region growing algorithm, we compute an edge map to improve the segmentation. Therefore, it constrains the region on some borders and avoids some region ''leaks''. Although we can use both the depth map edges or the color image edges, in our implementation we used the color camera edges because our depth map was not continuous. Edge detection is made using a Canny edges detection [42]. The edges are stored in a byte

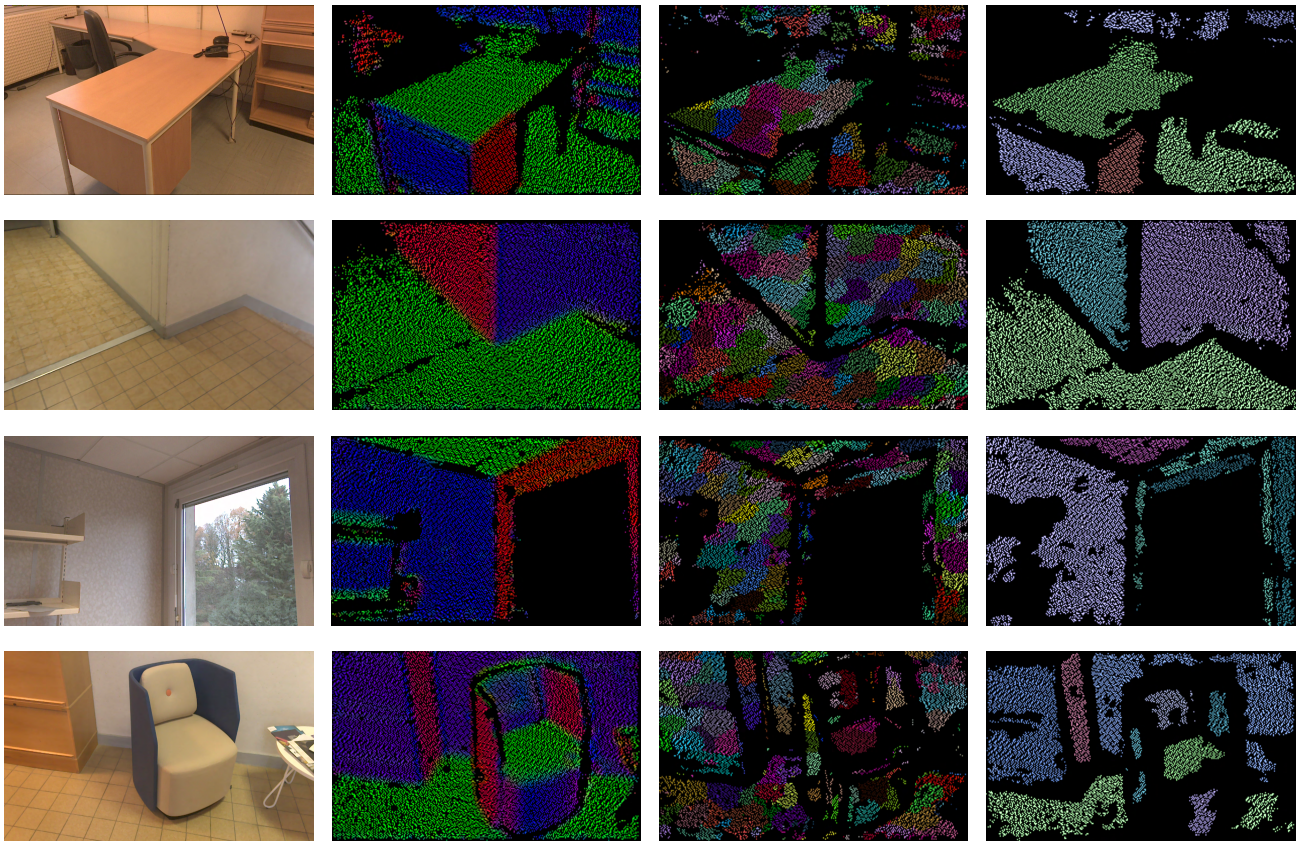


FIGURE 4. Some examples of the planes detection algorithm applied on different areas. From left to right : RGB images, map of the normals, generated clusters, and final planes.

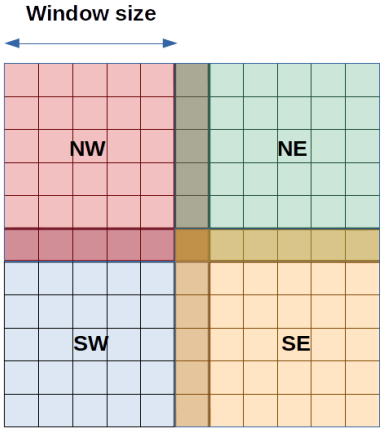


FIGURE 5. Example of a sliding window with a size of 5. The four quadrants are identified by North West (NW), South West (SW), North East (NE) and South East (SE).

map, where each byte is set to 0 if the point indexed by (u, v) is supposed to be on an edge, and it is 0 elsewhere. An example of the edges computed is shown in Fig.9h.

3) D-MAP COMPUTING

The D-map is constructed by computing for each point $P(x, y, z)$ the d parameter of the plane passing by P and verifying $n_x x + n_y y + n_z z + d = 0$, where $\overrightarrow{n}(n_x, n_y, n_z)$ represents the unitary normal vector computed at this point. The D-value of each point P of the RGB-D data set is then defined by :

$$d = -(n_x x + n_y y + n_z z). \tag{4}$$

This D-map is used in combination with the normal and the depth map to perform the planar segmentation. Each d value is computed to avoid the need to recompute them during the plane detection phase.

C. PLANAR SEGMENTATION

Our point descriptor is defined as a vector that embeds the spatial, normal, and d information computed for a point, and also a weight which indicates the confidence given to the descriptor. The planar segmentation uses a set of point descriptors as an input and it returns a set of clusters defined by a set of clusters defined by a structure *superpixels_cluster_t*. This structure contains two vectors: the actual plane parameters and the list of the point descriptors belonging to this plane.

When a point descriptor is used as an input, the weight values are set as 1. When a cluster has to be updated, its descriptor weight is updated whenever a new point is added to this cluster. The cluster's references are supposed to

represent the parameters of the plane containing the cluster's points.

The algorithm performs a planar segmentation of the input descriptors in two steps. The points descriptors map is first divided into a grid of seed descriptors and a subset of clusters is generated around each seed descriptor, as depicted in Fig.6. These clusters are then merged to generate the final segmentation. This method is similar to the supervoxels method [33]. Examples of planar detection performed with our algorithm are shown in Fig.4.

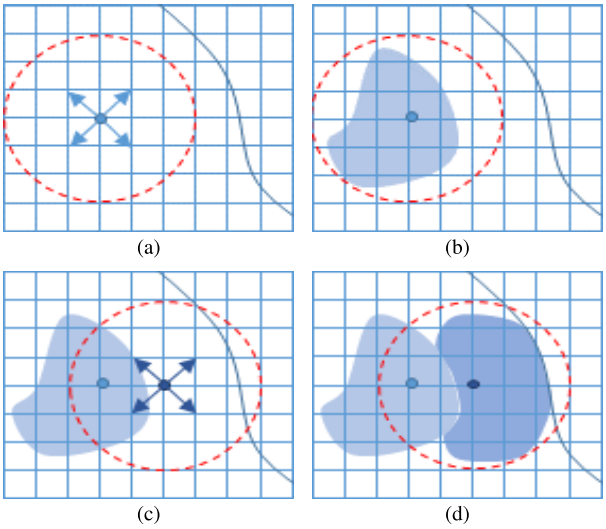


FIGURE 6. Clusters generation process : a grid of seed points is applied on the point descriptors map, with a fixed interval. The algorithm processes line by line: it selects the next available seed point (a), and generates a cluster with the neighbour points in a given max radius (in red) having the same characteristics (b). Then the next non observed seed point is used to generate the next cluster (c), each point already placed in a cluster is ignored, and the spreading stops if an edge is detected (d).

1) CLUSTER GENERATION

Algorithm 1 shows how a point descriptor P at the index (u, v) can be added to a *cluster*. Because a descriptor can belong to only one cluster at one time, a Boolean map is used. Therefore, we used the noted *seen* map, which contains `true` if the descriptor at any index (u, v) has been already integrated. Let P_{ref} be the descriptor of *cluster*. If P has not been previously added, then if *cluster* is empty, we initialize it with P. Otherwise, we compute the distance $dist(P, P_{ref})$ as the Euclidean distance between the four parameters: n_x, n_x, n_x, d. If the distance is inferior to an ϵ value, then P_{ref} is updated with P by adding (u, v) to the cluster's indices list and then updating its reference. Let α be a real number. Then, we compute :

$$\alpha = \begin{cases} 1 & \text{if } \delta \geqslant 1 \\ \dfrac{|P.d - P_{ref}.d|}{|P_{ref}.d|} & \text{otherwise} \end{cases} \tag{5}$$

Let ω be the weight of P_{ref}, then we update P_{ref} with :

$$P_{ref} \leftarrow \frac{\omega P_{ref} + \alpha P}{\alpha + \omega} \tag{6}$$

Algorithm 1 Cluster Generation

```
P ← features.at(u, v)
if seen(u, v) then
    return
end if
if cluster.size = 0 then
    seen(u, v) ← true
    P_ref ← P
    cluster.indices.add(u, v)
else
    d ← superpixels_dist(P, P_ref)
    if d ≤ ε then
        seen(u, v) ← true
        update_reference(P_ref, P)
        cluster.indices.add(u, v)
    end if
end if
if depth < DEPTH_MAX then
    for (u', v') ∈ neighborhood(u, v) do
        add_point(u', v', cluster, depth + 1)
    end for
end if
```

and then

$$\omega = \alpha + \omega \tag{7}$$

If the algorithm is recursively applied to the neighbour descriptors of P, it stops when a descriptor is not integrated or when the maximum recursive depth $DEPTH\_MAX$ has been reached. This process is depicted in Fig.6.

Algorithm 2 Fusion of Clusters

```
merged ← ∅
planes ← ∅
for base ∈ clusters, base ∉ merged do
    plane ← base
    for c ∈ clusters, c ∉ merged, c ≠ base do
        if dist(c.ref, base.ref) ≤ ε then
            plane+ = c
            plane.ref ← (ω_ref plane.ref + ω_c c.ref) / (ω_ref + ω_c)
            ω_plane ← ω_ref + ω_c
            merged+ = c
        end if
    end for
    planes+ = plane
end for
```

2) PLANE GENERATION

Algorithm 2 describes the merging process of the clusters that were previously detected. Once again, the Euclidean distance is computed between the different clusters. If two clusters are considered to be similar, then they are merged

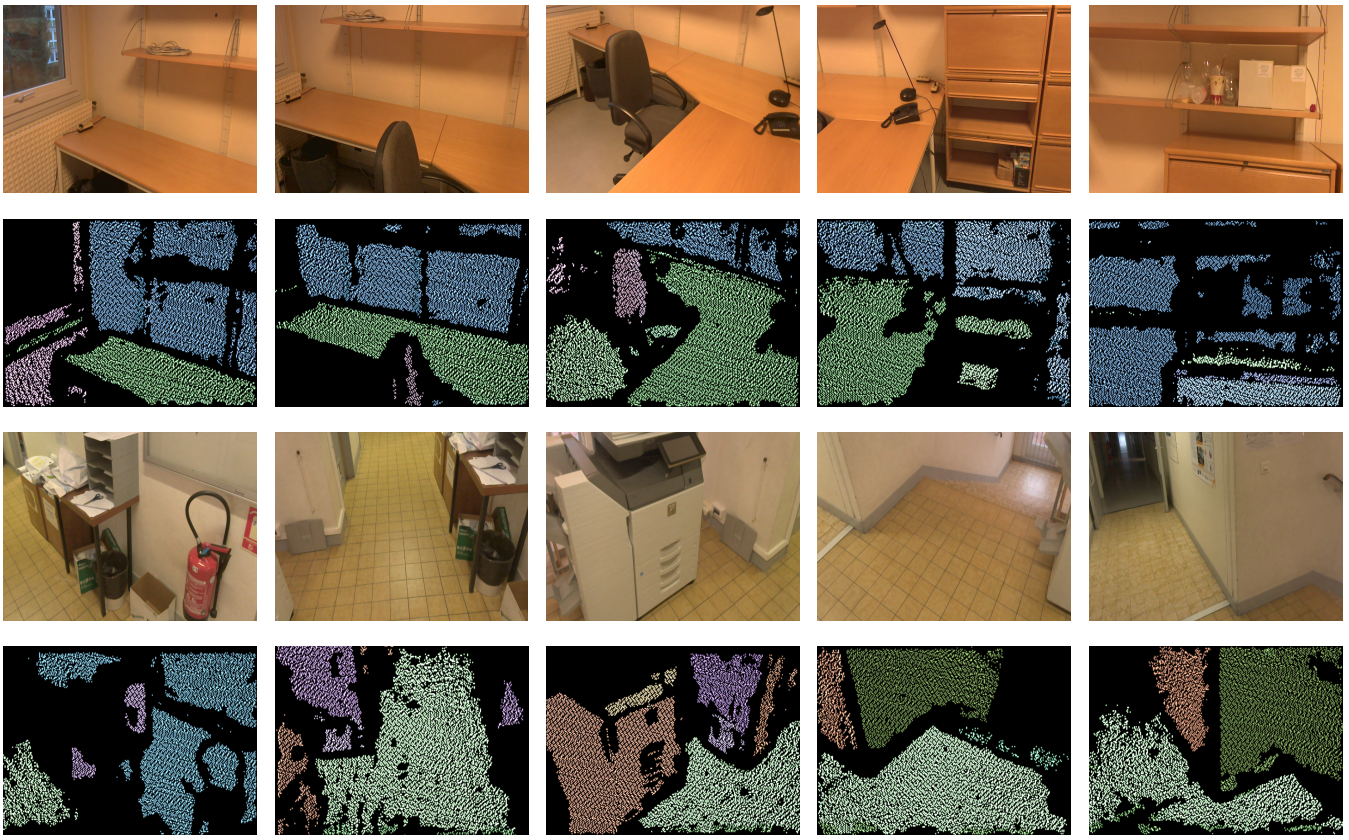


FIGURE 7. Time consistency: simular planes have the same color through different frames.

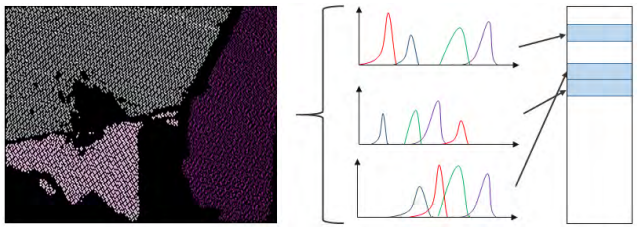


FIGURE 8. Planes storage mechanism : for each identified plane, we compute an histogram for each of its parameter. This histogram serves to compute an ID that will be used as a key for storing the plane in a hash map.

together. The parameters of the resulting cluster are computed using a weighted mean of the parameters of the two clusters.

D. PLANE STORAGE

In this section, we describe how the planes are actually stored in memory and how we detect similar planes in different frames. The planes storage mechanism is depicted Fig.8. It is assumed here that a motion tracking algorithm is already provided by the device. Alternatively, some existing tracking algorithms, such as Visual Inertial Odometry [43], can be used instead.

Our algorithm is developed to serve as a basis for on the fly interior building wall detection. Consequently, we focus on detecting large planes, which are supposed to be wall candidates. Focusing on large planes ensures us that we have planes with a sufficient number of points to use a statistical description of the plane parameters. In practice, we keep each previously detected plane if it has more than 100 points, while the other planes are considered as irrelevant.

1) PLANES HISTOGRAMS

Once the segmentation has been achieved, we use the device's pose data to transform our cluster descriptors into the world's frame coordinates.

Let E be a countable set of $\mathbb{R}$, and $\epsilon \in \mathbb{R}$. We define the histogram $H_\epsilon(E)$ by $\forall i \in \mathbb{Z}$:

$$H_\epsilon(E)(i) = |\{x \in E / x \in [i\epsilon, (i+1)\epsilon]\}|, \tag{8}$$

where $|.|$ denotes the cardinality of a set. Its median value $\mu_{i,\epsilon}(E)$ with $i \in \mathbb{N}$ is then defined by:

$$\mu_{i,\epsilon}(E) = \frac{\Sigma_{p=-i}^{i}(pH_\epsilon(E)(p))}{\Sigma_{p=-i}^{i}(H_\epsilon(E)(p))} \tag{9}$$

Let P be a plane that was previously detected, and N_x, N_y, N_z, D, the sets of the different values of the planes parameters for each point descriptor are included in P. We then compute the associated histograms $H_\epsilon(N_x)$, $H_\epsilon(N_y)$, $H_\epsilon(N_z)$ and $H_\epsilon(D)$, and we compute an identifier (ID) $I_{i,\epsilon}(P) \in \mathbb{Z}^4$ defined by :

$$I_{i,\epsilon}(P) = (\mu_{i,\epsilon}(N_x), \mu_{i,\epsilon}(N_y), \mu_{i,\epsilon}(N_z), \mu_{i,\epsilon}(D)) \tag{10}$$

Because P has been constructed by gathering descriptors that have similar planar parameters, (n_x, n_y, n_z, d), it can be assumed that their distribution follows a binomial law. Then, $I_{i,\epsilon}(P)$ can serve to retrieve the actual parameters of P. Histograms provide a strong estimation of the cluster's plane parameters and they are resistant to estimation errors. We computed the histograms for our parameters using $\epsilon = 0.001$, and $i = 100$. Therefore, to simplify the notations, we will call H_x, H_y, H_z and H_d the different computed histograms for our planes, and $I_p = I_{i,\epsilon}(P)$ its associated ID. The planes are defined with their set of four histograms, their ID and their weight, which increases each time the parameters of the plane are updated. In addition to the histogram and to the ID, we compute a weight for the plane. This weight will be increased each time that the plane's parameters are updated.

2) PLANES LIST STORAGE

A hash map is used to store each identified plane P, using $I_p = ((id_0, id_1, ud_2, id_3))$ as a key. The index of a plane in the map is computed similarly to the method used by Niessner et al. [44]. Given four large primers p_0, p_1, p_2, p_3, we compute an index $i = (p_0 id_0 \oplus p_1\ id_1 \oplus p_2\ id_2 \oplus p_3\ id_3)[n]$, where n is the size of the hash map and $\oplus$ denotes the *xor* operator. Planes with similar IDs are stored in lists. When inserting a P, its hash entry is computed and compared with the IDs of the planes previously stored at this entry. If a similar plane is found, then they are merged; otherwise, P is added at the queue of the list. Using a hash map allows us to have a $O(1)$ access time to an element, independent of the number of stored planes.

Let p_1 and p_2 be two planes. To compare them, their parameters are first retrieved using their IDs and a Euclidean distance is used. When inserting a plane P_{ref}, with a weight ω_{ref} in our hash table T, if we find an already stored plane P with a weight ω having similar parameters, then P_{ref} is merged with P. The histogram of P, its ID, and its weight are updated using the corresponding data of P_{ref}. Let H be the histogram of P and H_{ref} be the histogram of P_{ref}. Then, H is updated in the following way:

$$H \leftarrow \frac{\omega H + \omega_{ref} H_{ref}}{\omega + \omega_{ref}} \tag{11}$$

The ID of P is then computed and its weight is updated with $\omega + \omega_{ref}$, and its position is updated in T. The storage update process is shown in algorithm.3.

Algorithm 3 Planes List Update

```
P ← find_if_exists(P_ref)
if P ≠ ∅ then
    merge_planes(P_ref, P)
    insert(P_ref)
else
    insert(P)
end if
```

E. DRIFT CORRECTION

Although the use of histograms provides a robust and time consistent estimation of the planes parameters, some drifts in the motion tracking can occur and they can affect the final results. An Iterative Closest Points (ICP) algorithm [45] is used to correct small frame-to-frame drifts. ICP is an iterative process that follows two steps: first, correspondences are found between a source and a target point cloud; and second, an estimate is made of a transformation that minimizes the overall distance (often the least square distance) between them. However, finding correspondences can be costly in terms of computing resources, even though the use of ordered clouds as an input can simplify the process.

Let S be a source point cloud and T be the target cloud that we want to align with. These clouds are ordered, so each point can be referenced as $S(i, j)$ or $T(i, j)$ where i and j are integers. Assuming that there is a small movement between two successive point clouds, it can be assumed that for each point $S(i, j)$ in the input point cloud, a corresponding point, if it exists, can be found in the neighborhood of $T(i, j)$.

So for each point $S(i, j)$, a corresponding point is searched in a fixed size window around the associated point $T(i, j)$.

Once the correspondences are found, Arun et al.'s [46] SVD method developed is used to find a transformation that minimizes the least square distance between the correspondences.

IV. IMPLEMENTATION

We used a Google Project Tango Yellowstone tablet for our developments.[6] This tablet has an Nvidia Tegra K1 processor with 4 GM of RAM, and it integrates a depth and a motion tacking sensor. The Google Tango API already provides motion tracking and callbacks to use the depth data issued from the depth sensor.

The different processes to generate the different data maps used for the planar segmentation can easily be vectorized to optimize the performance. The Arm NEON intrinsic [47] has been used to optimize these phases. Some other parts were vectorized, such as the histograms merging parts. This process is kept as linear as possible to avoid the use of mutual exclusions. Given the fact that several tasks are repetitive and independent, OpenMP [48] is used in most of the steps to parallelize these tasks.

The tablet's depth sensor and the Google Tango API provide ordered point clouds with a resolution of(640×480), each $200ms$.

Our implementation allows us to process each of these incoming point clouds in less than $200ms$, using only the device's CPU capabilities.

V. EXPERIMENTAL VALIDATION

We tested our plane detection algorithm by measuring the maximal dimensions of different places. We chose three different places, including rooms with a lot of furniture along the

[6] https://developers.google.com/tango/

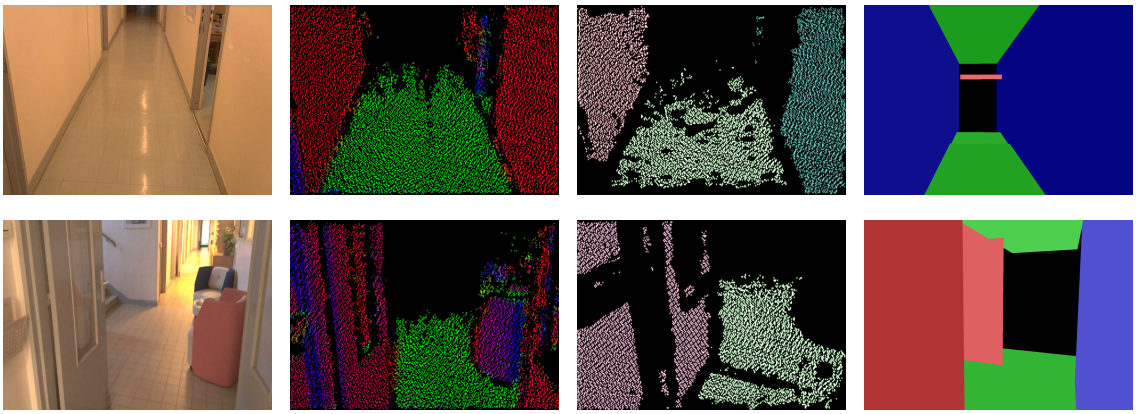


FIGURE 9. Application: the detected planes are merged frame to frame and will serve to identify the building walls.

TABLE 1. Comparison between computed and real dimensions in meters.

Measure	Office			Corridor		Meeting room		
	L	H	W	L	H	L	H	W
Real dim (.m)	5.80	2.70	3.30	2.70	1.64	6.80	2.70	5.60
Mean (.m)	5.82	2.73	3.35	2.70	1.62	6.76	2.89	5.64
Std dev. (.m)	0.08	0.13	0.04	0.04	0.04	0.12	0.25	0.21
Std dev. (%)	1	5	1	1	2	2	9	4
Max error (%)	3	11	4	3	5	4	25	11

walls, each having three dimensions to measure (length (L), height (H) and width (W)), and a corridor with two measures (height (H) and width(W)). For each room, we made a complete scan using the tablet to ensure that we had detected all of the main planes. We started the scan so that we were sure that each wall would be aligned on our base frame of reference. We then looked to the planes with the most important weight. We computed the distances between parallel planes to extract the room's maximal dimension. The measures were repeated 10 times for each room. The results are shown in Table 1, where the first line shows the real dimensions and the second shows the mean distance provided by our algorithm. The next two lines show the standard deviation for each measure, in meters and in percentages. The last line shows the maximum error for each measure.

Our tests show that the error is on average below 5%, except for one case where there were a number of lights on the roof that added some noise to the sensor. Fig.9 shows an example of a complete output, including an RGB image of the considered scene and the output planes in a 3D model.

Errors are mainly caused by the noises in the depth data and small pose estimation errors. Normals estimation can produce some errors around edge areas in the depth map, but doesn't affect the planar segmentation since we are looking for large planar areas. Because of the noise, parallel planes can be dinstinguished if they are sufficiently distant, othrewise, they could be merged into one single plane.

VI. CONCLUSION AND FUTURE WORK

We developed a plane detection algorithm that runs entirely on a tablet that is equipped with a depth sensor and a self-contained motion tracking algorithm. Our algorithm performs on-the-fly plane detection whenever a point cloud is available from the depth sensor. The segmentation is achieved by creating clusters of points that have the same supposed plane parameters. Once the segmentation is achieved, an histogram of the plane parameters is computed for each cluster after being transformed into the world's coordinates. These histograms serve to give a unique ID to each plane and to detect similar planes through different frames.

Our algorithm will serve as a basis for an application that will perform 3D modeling of an existing indoor building on the fly. Our future work will focus on updating the border of the detected planes at each new frame and identifying the walls. We will then focus on using the RGB image to detect the building's openings and attach them to a wall element.

REFERENCES

[1] C. Eastman, C. M. Eastman, P. Teicholz, and R. Sacks, *BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors*. Hoboken, NJ, USA: Wiley, 2011.

[2] X. Xiong, A. Adan, B. Akinci, and D. Huber, "Automatic creation of semantically rich 3d building models from laser scanner data," *Autom. Construction*, vol. 31, pp. 325–337, May 2013.

[3] J. Jung *et al.*, "Productive modeling for development of as-built BIM of existing indoor structures," *Autom. Construction*, vol. 42, pp. 68–77, Jun. 2014.

[4] S. Izadi *et al.*, "KinectFusion: Real-time 3D reconstruction and interaction using a moving depth camera," in *Proc. 24th Annu. ACM Symp. User Interface Softw. Technol.*, 2011, pp. 559–568.

[5] R. A. Newcombe *et al.*, "Kinectfusion: Real-time dense surface mapping and tracking," in *Proc. 10th IEEE Int. Symp. Mixed Augmented Reality (ISMAR)*, Oct. 2011, pp. 127–136.

[6] 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 *Proc. Robot., Sci. Syst.*, Rome, Italy, Jul. 2015.

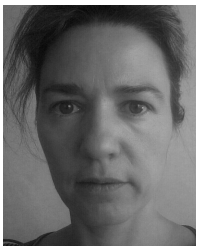
[7] T. Schöps, T. Sattler, C. Häne, and M. Pollefeys, "3D modeling on the go: Interactive 3D reconstruction of large-scale scenes on mobile devices," in *Proc. Int. Conf. 3D Vis. (DV)*, Oct. 2015, pp. 291–299.

[8] A. Arnaud, J. Christophe, M. Gouiffès, and M. Ammi, "3D reconstruction of indoor building environments with new generation of tablets," in *Proc. 22nd ACM Conf. Virtual Reality Softw. Technol.*, 2016, pp. 187–190.

[9] V. Garcia, E. Debreuve, and M. Barlaud, "Fast k nearest neighbor search using GPU," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2008, pp. 1–6.
[10] A. Nuchter, K. Lingemann, and J. Hertzberg, "Cached k-d tree search for ICP algorithms," in *Proc. 6th Int. Conf. 3-D Digit. Imag. Modeling (DIM)*, 2007, pp. 419–426.
[11] M. Greenspan and M. Yurick, "Approximate k-d tree search for efficient ICP," in *Proc. 4th Int. Conf. 3-D Digit. Imag. Modeling (DIM)*, 2003, pp. 442–448.
[12] M. Connor and P. Kumar, "Fast construction of k-nearest neighbor graphs for point clouds," *IEEE Trans. Vis. Comput. Graphics*, vol. 16, no. 4, pp. 599–608, Apr. 2010.
[13] J. Sankaranarayanan, H. Samet, and A. Varshney, "A fast all nearest neighbor algorithm for applications involving large point-clouds," *Comput. Graph.*, vol. 31, no. 2, pp. 157–174, 2007.
[14] S. Holzer, R. B. Rusu, M. Dixon, S. Gedikli, and N. Navab, "Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2012, pp. 2684–2689.
[15] K. Klasing, D. Althoff, D. Wollherr, and M. Buss, "Comparison of surface normal estimation methods for range sensing applications," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2009, pp. 3206–3211.
[16] H. Gouraud, "Continuous shading of curved surfaces," *IEEE Trans. Comput.*, vol. COM-100, no. 6, pp. 623–629, Jun. 1971.
[17] G. Thürrner and C. A. Wüthrich, "Computing vertex normals from polygonal facets," *J. Graph. Tools*, vol. 3, no. 1, pp. 43–46, 1998.
[18] S. Jin, R. R. Lewis, and D. West, "A comparison of algorithms for vertex normal computation," *Vis. Comput.*, vol. 21, no. 1, pp. 71–82, 2005.
[19] R. Hoffman and A. K. Jain, "Segmentation and classification of range images," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-9, no. 5, pp. 608–620, Sep. 1987.
[20] J. Huang and C.-H. Menq, "Automatic data segmentation for geometric feature extraction from unorganized 3-D coordinate points," *IEEE Trans. Robot. Autom.*, vol. 17, no. 3, pp. 268–279, Mar. 2001.
[21] R. O. Duda and R. E. Hart, "Use of the Hough transformation to detect lines and curves in pictures," *Commun. ACM*, vol. 15, no. 1, pp. 11–15, Jan. 1972.
[22] P. V. C. Hough, "Method and means for recognizing complex patterns," U.S. Patent 3 069 654, Dec. 18, 1962.
[23] D. Borrmann, J. Elseberg, K. Lingemann, and A. Nüchter, "The 3D Hough transform for plane detection in point clouds: A review and a new accumulator design," *3D Res.*, vol. 2, no. 2, pp. 1–13, 2011.
[24] R. Raguram, J.-M. Frahm, and M. Pollefeys, "A comparative analysis of RANSAC techniques leading to adaptive real-time random sample consensus," in *Computer Vision—ECCV*. Berlin, Germany: Springer, 2008, pp. 500–513.
[25] M. A. Fischler and R. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, 1981.
[26] R. Schnabel, R. Wahl, and R. Klein, "Efficient ransac for point-cloud shape detection," *Comput. Graph. Forum*, vol. 26, no. 2, pp. 214–226, 2007.
[27] F. Tarsha-Kurdi *et al.*, "Hough-transform and extended RANSAC algorithms for automatic detection of 3D building roof planes from lidar data," in *Proc. ISPRS Workshop Laser Scanning*, vol. 36. 2007, pp. 407–412.
[28] D. Holz, S. Holzer, R. B. Rusu, and S. Behnke, "Real-time plane segmentation using RGB-D cameras," in *RoboCup: Robot Soccer World Cup XV*. Berlin, Germany: Springer, 2011, pp. 306–317.
[29] B. Fulkerson, A. Vedaldi, and S. Soatto, "Class segmentation and object localization with superpixel neighborhoods," in *Proc. IEEE 12th Int. Conf. Comput. Vis.*, Oct. 2009, pp. 670–677.
[30] C. Erdogan, M. Paluri, and F. Dellaert, "Planar segmentation of RGB-D images using fast linear fitting and Markov chain Monte Carlo," in *Proc. 9th Conf. Comput. Robot Vis. (CRV)*, 2012, pp. 32–39.
[31] A. Barbu and S.-C. Zhu, "Generalizing Swendsen-Wang to sampling arbitrary posterior probabilities," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 8, pp. 1239–1253, Aug. 2005.
[32] R. H. Swendsen and J.-S. Wang, "Nonuniversal critical dynamics in Monte Carlo simulations," *Phys. Rev. Lett.*, vol. 58, no. 2, p. 86, 1987.
[33] J. Papon, A. Abramov, M. Schoeler, and F. Worgotter, "Voxel cloud connectivity segmentation-supervoxels for point clouds," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2013, pp. 2027–2034.
[34] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2. Sep. 1999, pp. 1150–1157.
[35] P. Scovanner, S. Ali, and M. Shah, "A 3-dimensional sift descriptor and its application to action recognition," in *Proc. 15th ACM Int. Conf. Multimedia*, 2007, pp. 357–360.
[36] C. Harris and M. Stephens, "A combined corner and edge detector," in *Proc. Alvey Vis. Conf.*, vol. 15. Manchester, U.K., 1988, p. 5244.
[37] I. Sipiran and B. Bustos, "Harris 3D: A robust extension of the harris operator for interest point detection on 3D meshes," *Vis. Comput.*, vol. 27, no. 11, pp. 963–976, 2011.
[38] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (SURF)," *Comput. Vis. Image Understand.*, vol. 110, no. 3, pp. 346–359, 2008.
[39] A. Flint, A. Dick, and A. Van Den Hengel, "Thrift: Local 3D structure recognition," in *Proc. 9th Biennial Conf. Austral. Pattern Recognit. Soc. Digit. Image Comput. Techn. Appl.*, 2007, pp. 182–188.
[40] Y. Guo, M. Bennamoun, F. Sohel, M. Lu, and J. Wan, "3D object recognition in cluttered scenes with local surface features: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 11, pp. 2270–2287, Nov. 2014.
[41] J. M. Coughlan and A. L. Yuille, "Manhattan world: Compass direction from a single image by Bayesian inference," in *Proc. 7th IEEE Int. Conf. Comput. Vis.*, vol. 2. Oct. 1999, pp. 941–947.
[42] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-8, no. 6, pp. 679–698, Nov. 1986.
[43] M. Li and A. I. Mourikis, "Improving the accuracy of EKF-based visual-inertial odometry," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2012, pp. 828–835.
[44] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, "Real-time 3D reconstruction at scale using voxel hashing," *ACM Trans. Graph.*, vol. 32, no. 6, p. 169, 2013.
[45] P. J. Besl and D. N. McKay, "A method for registration of 3-D shapes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, no. 2, pp. 239–256, Feb. 1992.
[46] K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-squares fitting of two 3-D point sets," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-9, no. 5, pp. 698–700, Sep. 1987.
[47] V. G. Reddy, *Neon Technology Introduction*. Cambridge, U.K.: ARM Corporation, 2008.
[48] L. Dagum and R. Menon, "OpenMP: An industry standard API for shared-memory programming," *IEEE Comput. Sci. Eng.*, vol. 5, no. 1, pp. 46–55, Mar. 1998.

ADRIEN ARNAUD is currently pursuing the Ph.D. degree with the Laboratory LIMSI-CNRS, where he conducts research in 3-D reconstruction on mobile devices.

He is particularly interested in computer graphics and algorithmics.



MICHÈLE GOUIFFÈS received the Ph.D. degree from Poitiers Université in 2005. She has been an Engineer with INSA Rennes since 2002. She has been an Assistant Professor with the University Paris Sud (Paris Saclay) since 2006 where she conducts her research in image processing at laboratory LIMSI-CNRS. She is particularly interested in image feature descriptors and matching for interaction.

MEHDI AMMI is an Associate Professor with Paris-Sud University specializing in HCI, virtual reality, and Internet of Things. His research interests include the design and the study of smart systems and smart environments that consider the human capacities and the technological dimension. He is the Leader of a research team at LIMSI-CNRS. He was/is involved in several scientific organizations, such as IEEE Technical Committee on Haptics and the EuroVR Special Interest Group on Haptics.