Computers & Graphics 84 (2019) 199–211

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Special Section on SVR 2019

Geometrical and statistical incremental semantic modeling on mobile devices

Rafael Roberto[a,*], João Paulo Lima[a,b], Hideaki Uchiyama[c], Veronica Teichrieb[a], Rin-ichiro Taniguchi[c]

[a] Voxar Labs, Centro de Informática, Universidade Federal de Pernambuco, Brazil
[b] Departamento de Computação, Universidade Federal Rural de Pernambuco, Brazil
[c] Laboratory for Image and Media Understanding, Kyushu University, Japan

ARTICLE INFO

Article history:
Received 15 May 2019
Revised 27 August 2019
Accepted 17 September 2019
Available online 20 September 2019

Keywords:
Semantic tracking
Visual SLAM
Mobile devices
Tracking
Android

ABSTRACT

Improvements on mobile devices allowed tracking applications to be executed on such platforms. However, there still remain several challenges in the field of mobile tracking, such as the extraction of high-level semantic information from point clouds. This task is more challenging when using monocular visual SLAM systems that output noisy sparse data. In this paper, we propose a primitive modeling method using both geometric and statistical analyses for sparse point clouds that can be executed on mobile devices. The main idea is to use the incremental mapping process of SLAM systems for analyzing the geometric relationship between the point cloud and the estimated shapes over time and selecting only reliably-modeled shapes. Besides that, a statistical evaluation that assesses if the modeling is random is incorporated to filter wrongly-detected primitives in unstable estimations. Our evaluation indicates that the proposed method was able to improve both precision and consistency of correct detection when compared with existing techniques. The mobile version execution is 8.5 to 9.9 times slower in comparison with the desktop implementation. However, it uses up to 30.5% of CPU load, which allows it to run on a separate thread, in parallel with the visual SLAM technique. Additional evaluations show that CPU load, energy consumption and RAM memory usage were not a concern when running our method on mobile devices.

1. Introduction

Several applications on computer vision and augmented reality require camera pose tracking. However, determining the device position in relation to the real environment can demand a lot of computational power and memory depending on the approach and the required information. Along with all the improvement on processing power and memory on mobile devices itself, several tracking techniques that are capable to run on such devices were released in the last couple of years. There are examples in the academy and in the industry. The most distinguished ones are ARCore[1] and ARKit,[2] by Google and Apple, respectively.

Although several techniques have been released, there are still many remaining research issues in the field of tracking. One is to extract and track high-level semantic information from the environment. Different types of semantics can be collected, from geometric primitives of objects to the model of a piece of furniture. This process is referred to as semantic modeling. Shape parameters of geometric primitives and the relationship among them are valuable knowledge to be estimated especially in man-made environments such as a house and a factory. This data can also be gathered in different ways: from the input image, the scene map or a combination of both.

There are several benefits of detecting primitives from the scene map. For instance, objects are usually over-represented when defined using a point cloud because it is not necessary to have so many points to describe them. Therefore, an implicit representation can replace redundant points, which is particularly helpful when targeting devices with memory restrictions, such as robots or unmanned aerial vehicles (UAV). Additionally, a tracking system can use these primitives to denoise the reconstructed map or constrain its optimization [1], which can reduce tracking errors. Furthermore, it can be used to provide haptic feedback on augmented reality applications [2].

* Corresponding author.
E-mail address: rar3@cin.ufpe.br (R. Roberto).

[1] https://developers.google.com/ar/.
[2] https://developer.apple.com/arkit/.

https://doi.org/10.1016/j.cag.2019.09.003
0097-8493/

Mobile devices are reducing the gap to desktop computers in terms of processing power and memory space [3]. This improvement allowed the development of more complex algorithms for mobile devices, including tracking techniques, which are one of the foundations of augmented reality. This can be linked to the improvement and popularization of augmented reality solutions for such devices.

Some of these recent advancements in tracking techniques involve removing the necessity of any external marker, improving the execution time to achieve real-time performance and having a more stable tracking that is not harmed by jitter or drift. For instance, the improvement of the device's sensors allowed the development of visual-inertial trackers. However, there is not much progress regarding the extraction of different kinds of semantics from the 3D map of the scene.

Considering all the benefits mentioned above, the main goal of this study is to model and track primitives aiming to obtain semantics from sparse point clouds. In order to do that, we present Geometric and Statistical Incremental Semantic Tracking, or simply GS-IST, which is a method for incrementally modeling and tracking planes, spheres, and cylinders on sparse point clouds. In summary, this method uses the generating process of point clouds on SLAM effectively and relies on geometric and statistical analyses to filter unreliable shapes. Additionally, this method was ported and evaluated on mobile devices, showing that it was feasible to extract and track basic primitives on such platforms. This paper is an extended version of the work published in Roberto et al. 2018 [4] and its main contributions are summarized as follows:

- A technique that uses geometric and statistical evaluation to incrementally perform semantic modeling and tracking of primitives on sparse point clouds (Section 3);
- Evaluations of the proposed method in comparison with existing techniques, showing that it improves semantic modeling precision. This evaluation also includes a dataset with sparse point clouds of primitives and a metric precision evaluation that was not available in the preliminary work (Section 4);
- The port and evaluation of the proposed approach to mobile devices and a guideline to evaluate computer vision techniques on such platform, which is also an enhancement from our previous paper (Section 5).

2. Related works

Automatic reconstruction of 3D object shapes is useful for several applications, such as blueprint generation for architecture. Although useful for 3D measurement and visualization, there are some aspects to be improved. For instance, the scene is usually represented by using a point cloud or a mesh computed from it. The latter simply consists of connected neighbor points with little information about the semantic structure.

Several methods have been proposed in the literature to determine semantics in point clouds, and most of them are based on dense clouds. One conventional approach is to use reverse engineering techniques to estimate geometric primitives, such as region growing [5]. It can efficiently deal with large amounts of data because it makes simple comparisons using the normals to determine if a set of points belongs to the same group. However, this approach is not robust to noisy point clouds, leading to wrong classifications. The robustness has been improved using both Hough Transform [6] and RANSAC [7]. Some methods are able to detect only specific primitives, such as planes [8–10] or cylinders [11,12]. Some detect a set of primitives, such as planes, cylinders, cones and spheres [13] and others identify premodeled complex shapes along with planes and cylinders in industrial scenarios [14]. Another approach is based on machine learning techniques, which combine local features and AdaBoost to detect complex objects [15]. Support Vector Machine (SVM) and RANSAC are also used to extract semantics from a point cloud [16,17]. Recent works used Convolutional Neural Network (CNN) to perform a semantic classification using the keyframes of a dense monocular SLAM and then apply this result to the point cloud [18].

One advantage of having dense data extracted by laser or infrared sensors is that the acquired point cloud contains several information that the algorithm can use for the detection. Moreover, the data is relatively noiseless, which means that a large number of points can stably fit primitive shapes. Therefore, it is more challenging to extract primitive shapes in a noisy and sparse point cloud of a partially-observed object computed from image-based approaches with mobile devices. Some studies and systems have tackled this issue by focusing on specific situations, like detecting only one shape class, such as ARKit and ARCore that detect only planes. Another example is [19], which estimated planes based on their reconstruction to create textured models. In this context, RANSAC-based methods are very promising to work with sparse point clouds. This is because they estimate primitives by initially picking a minimal group of points for each shape and detecting the one that approximates the maximum number of points [20]. Besides, they can also work with data containing a large number of outliers [21]. However, the performance of such approach for sparse point clouds was not sufficiently investigated yet.

Another aspect of existing semantic methods is that they usually work in batch, which means that input data is analyzed all together only once. This is consistent with the generation method. Usually, the dense sensors generate the entire point cloud at once. However, the performance of visual SLAM systems regarding both the accuracy of the reconstruction and the computational cost for real-time applications improved drastically in recent years [22,23]. Several of these SLAM methods generate the 3D map incrementally, which can provide valuable information for a semantic approach in addition to the point cloud itself.

Until 2015, tracking techniques on mobile devices had a substantial improvement on both number of publications but, most importantly, on tracking capabilities [24]. However, none of these techniques were able to retrieve any type of semantics from the scene.

There are examples of geometric techniques for object recognition too. One is [25], which created a distributed approach that uses a cache of frames scheme to improve the method performance. Their method is able to identify from faces to traffic signs. Recent developments of machine learning techniques allowed the extraction of semantics from images using neural networks. The architectures of these CNN are designed to deal with the constraints of mobile devices [26]. For instance, Tobas et al. 2016 [27] use deep learning to perform domain-specific object recognition. They achieved high classification accuracy and near real-time execution time running on an iPad. However, no work was found that is able to use machine learning to extract primitive parameters on mobile devices. One challenge is that running neural networks to address this level of semantics often requires a lot of processing power and powerful GPUs that are not available on mobile devices at this moment [28].

Although most of the studies for mobile devices find semantics from images, it is also possible to retrieve these information from point clouds. This is more complex to achieve on such devices due to the overload of data to process, especially when dealing with dense point sets. However, some systems that are able to extract information from dense point clouds have been developed recently. These techniques usually generate the point cloud using a Google Tango device, such as [29] that apply an iterative RANSAC approach to segment planes and model walls of indoor structures. Using a Tango as well, Sankar et al. 2017 [30] also detect planes and model

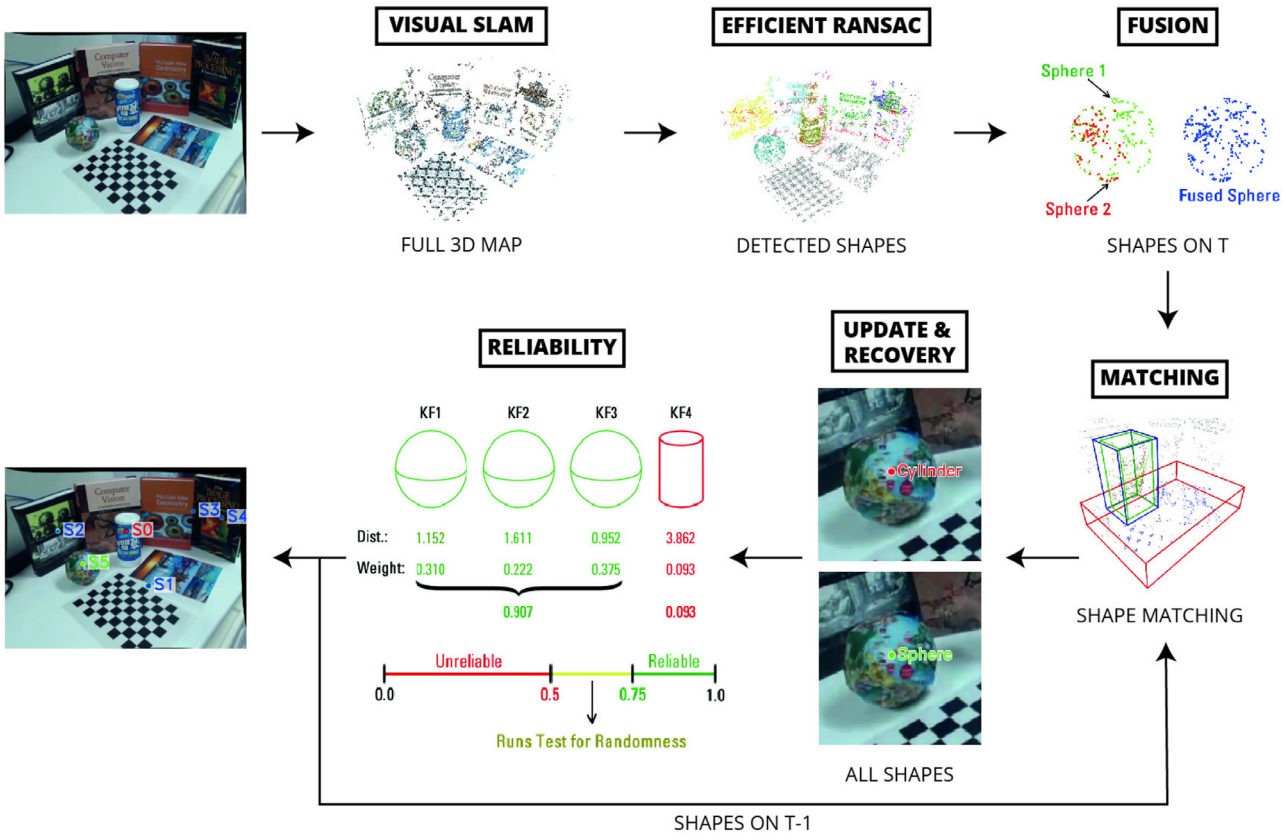

Fig. 1. Flow of the GS-IST approach. The boxes have the steps of our method and bellow them are the outputs of each step.

indoor environments. Nevertheless, they also detect planes to segment more complex objects and based on their set of points and visual appearance the authors are able to match the object with a 3D model available in the library.

Extracting semantic information using sparse point clouds should be simpler when concerning the amount of data to process. However, this is the same reason why this task is more challenging. Nevertheless, both Google's ARCore and Apple's ARKit incorporate semantic modeling as part of their scene understanding feature. They both extract planes based on the scene 3D reconstruction in order to have a more stable positioning of the virtual objects.

3. Geometric and statistical incremental semantic tracking

Previous evaluations indicate that Efficient RANSAC achieves good results when detecting primitives in a sparse point cloud [31]. However, it still requires improvements in consistency and precision for use in various applications. Therefore, it is possible to use the primitive estimation from Efficient RANSAC and the generating process of point cloud from visual SLAM systems to perform an incremental semantic modeling. This approach can improve both the precision and stability of the primitive detection. Moreover, it also allows the tracking of these primitives through the scene.

Fig. 1 shows the GS-IST execution flow. In summary, first it gets the sparse point clouds that are incrementally generated by a visual SLAM system and runs Efficient RANSAC to detect primitives. Then, it uses the history information of the estimated primitives and their parameters to match the shapes over time. Also, it estimates the reliability of the detected primitive using the geometry of the shape. When this estimation is not reliable enough, it performs a statistical evaluation using the detection history to eliminate random detected shapes. The remaining steps are detailed in the following subsections.

3.1. Efficient RANSAC

When a visual SLAM system reaches a keyframe, it updates the map adding new points to it. Then, GS-IST detects the shapes for every new map. We modified the Efficient RANSAC configuration to reduce the number of types of primitives detected. Instead of five, we track three classes of shapes: planes, spheres and cylinders. They were selected because they can be used to model most of the objects. In fact, when modeling industrial scenarios, almost 80% of the scene can be represented only by planes and cylinders [16].

In Fig. 1, every primitive detected using Efficient RANSAC has a different color. The points in red are unassigned points that were not fit to any shape. All these primitives are passed to the subsequent modules, which will identify and eliminate the unreliable shapes and track the remaining ones.

3.2. Shape fusion

In keypoint-based visual SLAM systems, most of the characteristics are clustered at highly textured areas. For example, spheres 1 and 2 in Fig. 1 are the same primitive that Efficient RANSAC detected as two because they have distinct clusters of features. In order to improve the results, we fuse different shapes that belong to the same primitive. This fusion not only provides a more representative primitive but also increases the overall information regarding the shape. Since a primitive with a small number of points can be unreliable, it is better to discard it. However, if more than one shape with similar parameters is detected, even if they are small, it is safe to assume that they correspond to the same element in the scene. The reason is that it is unlikely that two primitives

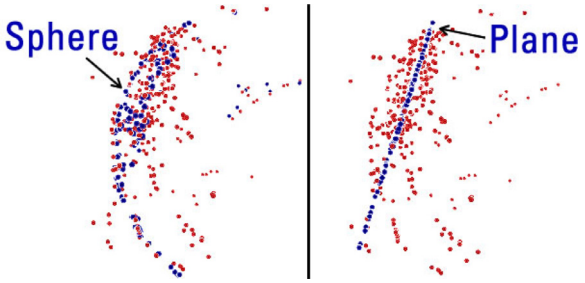

Fig. 2. Difference between the input points and their correspondent projected points on a shape estimated correctly (left) and incorrectly (right).

are wrongly detected with the same parameters. Therefore, this parameters-based fusion of primitives helps in the history evaluation. This process uses the similarity of the shape parameters to decide whether to fuse two shapes or not:

- **Plane:** the planes are parallel given an angle threshold α_t and the distance between them is smaller than the distance threshold d_t;
- **Sphere:** the distance between their centers is lower than d_t and the difference between their radii is smaller than the radius threshold r_t;
- **Cylinder:** the angle between the axis direction of both cylinders is smaller than α_t, the distance between them is smaller than d_t and the difference between their radii is smaller than r_t.

While d_t and r_t are controlled for each case such that they are 2% of the largest size of the point cloud bounding box, α_t is always set to 5°.

Additionally, our method considers the proximity between the primitives to restrict or widen the similarity thresholds. The principle is that two distant shapes have a smaller chance to be the same than closer ones. Experimentally, the thresholds are widened by 25% when evaluating the fusion of primitives that have an intersection. Otherwise, it is restricted by 25%. This means that distant shapes have to be more alike to be merged. On the other hand, closer primitives are more likely to be the same and the threshold can be less restricted.

3.2.1. Parameter computation

Similar shapes are fused according to the aforementioned criteria. GS-IST sets the parameters of the resulting fused shape as a weighted average between the parameters of both primitives. The weight is based on the geometric analysis of the points in the detected shape. The main idea is to use the average Euclidean distance between each input point that was used to estimate the primitive and the resulting shape. Even for noisy data, this distance will be smaller on correct estimations than on wrong ones. For instance, consider a globe being tracked that was correctly modeled as a sphere or incorrectly detected as a plane. The average distance of the 3D points in the globe to their projection in the sphere will be smaller when compared to the distance of the same input points to their projection in the wrong plane. Fig. 2 illustrates this idea.

Thus, the parameters P_f of the fused shape will be:

$$P_f = \sum_{i=1}^{n} w_i P_i, \tag{1}$$

where n is the number of similar shapes to be fused and P_i are their parameters. The weight w_i is:

$$w_i = \frac{m}{\sum_{j=1}^{m} d_j}, \tag{2}$$

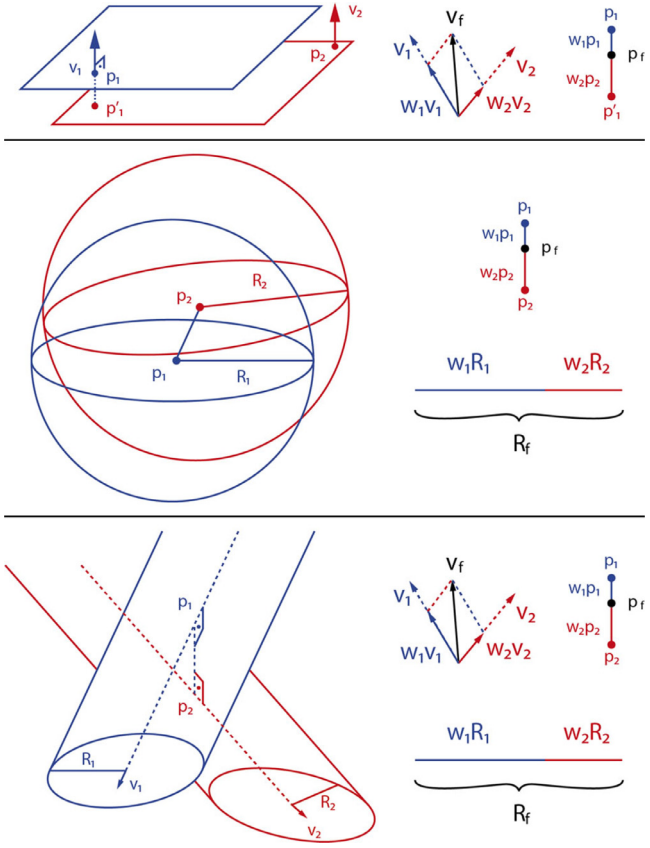

Fig. 3. Fusion of parameters for different classes of shapes.

where d_j is the Euclidean distance of the input points to their projection in the estimated shape and m is the number of points in the estimated primitive. The weights are normalized and $\sum_{i=1}^{n} w_i = 1$.

Using Eq. (1), every fused shape will influence the resulting primitive. However, it will be more similar to the one with the smaller error. This average can be applied to every parameter except the plane and the cylinder position. The point that represents the plane position is projected in all others and then the weighted average is computed, as illustrated in Fig. 3. This will also work in case of parallel planes. As for the cylinder position, this parameter will be the axes lines intersection. In case of concurrent or parallel axes lines, we fuse the cylinders in pairs when there are three or more. We select the closest points of the axes lines and the resulting cylinder position will be the weighted average of these points.

3.2.2. Inclusion criteria

In case a shape is not fused with any other, GS-IST performs an initial reliability evaluation to decide whether to keep this detected primitive or not. We consider four geometric characteristics to make this assessment, as described below:

- **Number of Points:** shapes with more points are usually more reliable because the estimation is based on a large amount of data. The number of points in good primitives is *larger than 2%* of the whole point cloud;
- **Dispersion:** it measures how spread are the points in the primitive when dividing the number of points by the volume it occupies. Since the keypoints are clustered around highly textured areas, small regions tend to concentrate most of the shape points. The dispersion of reliable shapes is *smaller than 20%* of the dispersion of the whole point cloud;

- **Distance:** it is the same Euclidean distance mentioned previously. Reliable shapes have an average distance *smaller than 5%* of the largest size of the entire point cloud bounding box;
- **Radius:** noisy planes can be estimated as spheres and cylinders with a large radius. Therefore, good spheres and cylinders have a radius *smaller than* the largest size of the entire point cloud bounding box. It should be noted that this criterion is not applied to planes.

The system only keeps shapes that pass in all of these criteria. These values were determined experimentally and they are based on the dimension of the input point cloud because it puts all thresholds in proportion to the scene scale.

3.3. Shape matching

Due to several factors, such as inconsistency or the shape volume, a primitive on a given keyframe can match with several others on the previous one, including shapes of a different type. Therefore, it is necessary to detect the shape on the previous keyframe that is the most likely to be the correspondent on the current one. To match the primitives between consecutive keyframes, GS-IST uses the intersection of the 3D bounding box and the distance between the center of mass. For instance, in Fig. 1, a cylinder in the current keyframe (in blue) intersects with two primitives in previous keyframes: another cylinder (in green) and a plane (in red). Thus, we compute a score s for each primitive that a given shape on the current detection intersects on the previous estimation. This score is proportional to the intersection volume and inversely proportional to the distance to the center of mass:

$$s = \frac{\sum_{i=1}^{k} |\psi_i| p_i}{\sum_{i=1}^{k} |\psi_i| d_i} \frac{|\psi_s|}{\sum_{i=1}^{k} |\psi_i|}, \tag{3}$$

where k is the number of shapes with intersection on previous keyframes, $|\psi_i|$ is the number of points of that shape, p_i and d_i are the intersection ratio and distance between centers of mass, respectively, and $|\psi_s|$ is the number of points in the primitive on the current keyframe. The shape with the maximum score is selected as correspondence.

3.4. Shape update and recovery

The shape detected on the current frame inherits the history data of the one it matched on the previous detection. These data contain the primitive class that was detected on every keyframe, as well as the average distance to the original point cloud at that detection. With that information, GS-IST can verify if the current estimation is following the historical data.

The system checks the class that this primitive was detected over time. If the current shape has the same type of the primitive that appears in more than half of the detection, including the current one, its parameter is updated. This new parameter will be the weighted average of each detection over time. With this update, every previous detection will influence the final shape, which results in primitives that are more stable over time rather than using just the parameters from the last detection.

The process to update the new parameter P_u is similar to the one explained in Section 3.2:

$$P_u = \sum_{i=1}^{n-1} w_i P_{u-1} + w_n P_f, \tag{4}$$

where n is the number of shapes detected in the past, including the current one, and w_i are their respective weights, which are normalized. P_{u-1} and P_f are, respectively, the parameters in the previous detection and the current parameter after fusion.

On the other hand, if the current shape has a type that is different from the one that appears most of the time, it is changed to that class of primitive. As for the parameters, it will be the same as the previous P_u from that type. This is illustrated in Fig. 1, in which a sphere was detected as a cylinder and, by looking at the history data, GS-IST was able to replace it by the correct primitive.

In this step, GS-IST also evaluates shapes that were not detected on the current keyframe, but appeared previously. It recovers these shapes with the same parameters from the last appearance. The history data will be updated using the average distance of the recovered shape, but it will not have a primitive class associated at this particular moment. This shape will eventually disappear when not detected anymore because there will be no class of primitives that appears in more than 50% of the time.

3.5. Reliability computation

At this point, GS-IST has a set of detected shapes and it is necessary to determine which of them are reliable. This reliability computation is based on a geometric and statistical evaluation.

3.5.1. Geometric analysis

Each shape has a history information since its first appearance, which is a list of each primitive it matched in the past keyframes. However, a given shape may have different classes over time due to imprecision or inconsistency. Therefore, to perform the geometric analysis, the weight w_c is computed for each class of primitive that appears in the history data:

$$w_c = \frac{1}{\sum_{i=1}^{h} a_i}, \tag{5}$$

where h is the number of times that each class of primitive appears in the history data and a_i is the average distance of the points in that shape to the original point cloud. Fig. 1 exemplifies this with a sphere that was detected as a cylinder in the fourth keyframe. The average distance of the points is much larger when compared to the three previous detections.

The weights are normalized and the one with the maximum value is the dominant class. GS-IST judges as reliable shapes whose dominant primitives have a weight higher than 0.75. On the other hand, it considers unreliable those in which all weights are smaller than 0.5. When the weight of the dominant shape is between these two values, its classification will be determined by the statistical analysis. If this evaluation shows that the detection class through history is random, the primitive will be unreliable. Otherwise, it will be set as reliable.

3.5.2. Statistical analysis

GS-IST performs a runs test for randomness to determine if the estimation history is random [32]. Basically, this non-parametric test uses the expected value and standard deviation to estimate the minimum number of runs that a sample can have to be considered random. A run means a sequence of consecutive estimates of one particular class of primitive. However, the system looks at the history of classification as binary data because the convergence is faster. Therefore, a + sign is assigned to the first primitive detected. Then, the sign is repeated if the shape class is the same as the previous one. Otherwise, it is inverted. For a 5% level of significance, the sample is random if the number of runs is greater than:

$$N(R) = \mu - 1.65\sigma, \tag{6}$$

where the expected value μ and the standard deviation σ for the total number of samples n are:

$$\mu = \frac{2n-1}{3}, \tag{7}$$

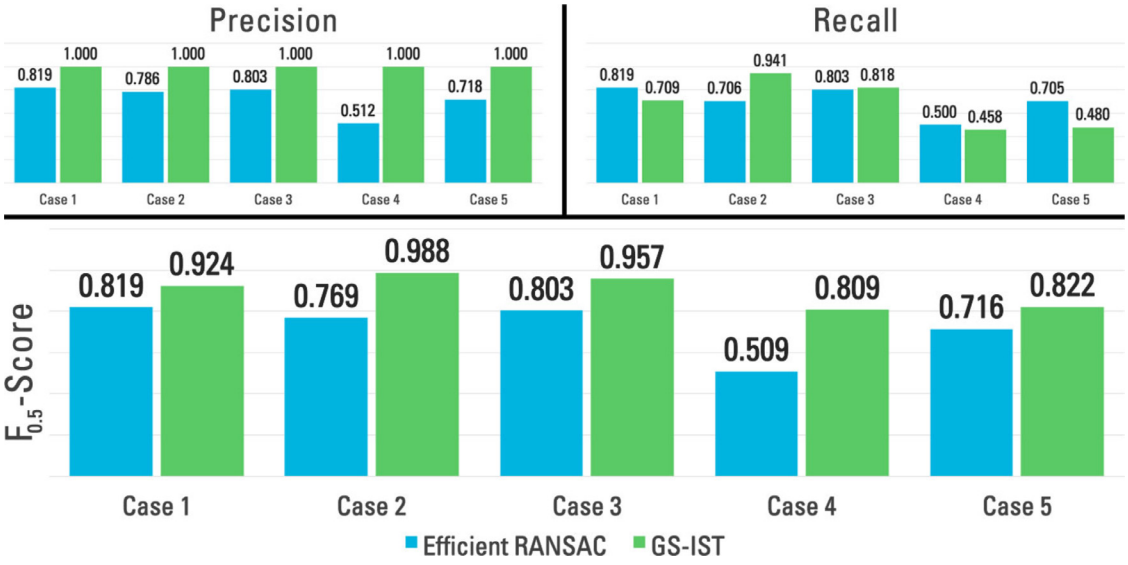

Fig. 4. Comparison of precision, recall and $F_{0.5}$-Score between Efficient RANSAC [20] and GS-IST.

Table 1
History of the estimated shape of a primitive. For each sample that represents a keyframe K_i, it was classified as plane (P), sphere (S) or cylinder (C).

	K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_8
Primitive	S	S	S	S	C	P	S	S
Label	+	+	+	+	−	+	−	−

$$\sigma = \sqrt{\frac{16n - 29}{90}}. \tag{8}$$

Table 1 shows the example of a history information with a sequence of four spheres, followed by one cylinder, one plane and then by two other spheres. There are $R = 4$ runs and $n = 8$ samples. In this case, the maximum number of runs for a nonrandom sample is $N(R) = 3.269$, which indicates a random detection.

4. GS-IST evaluation

GS-IST was implemented in C++ using OpenCV[3] and Efficient RANSAC[4] as libraries. This evaluation compared GS-IST with Efficient RANSAC regarding precision, recall and $F_{0.5}$-Score. To perform a fair comparison, we disabled in Efficient RANSAC the primitives we are not targeting in our method (toruses and cones). Since there was no dataset with the generating process of point clouds, we created one with five different scenarios using ATAM [23][5] in order to evaluate semantic modeling and tracking. This dataset has the RGB images, a text file containing the camera parameters and the rotation and translation for each frame, the list of keyframes and the point cloud generated on each keyframe. It has five scenes targeting distinct types of primitives and different numbers of keyframes, as illustrated with some screenshots in Fig. 8. It is worth to mention that Efficient RANSAC does not track the primitives, only detecting the shapes at each keyframe because it is a batch-based approach.

It is possible to see in Fig. 4 that GS-IST obtained 100% precision in all cases while Efficient RANSAC never achieves more than

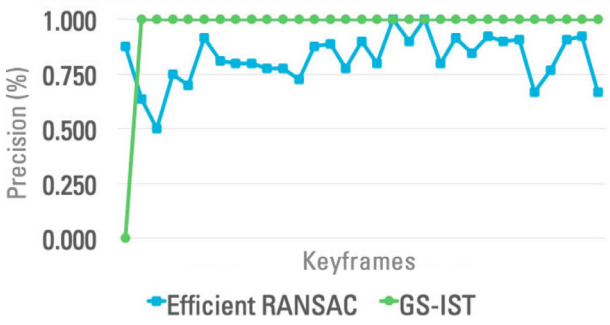

Fig. 5. Precision over time of Efficient RANSAC [20] and GS-IST on *Case 1*.

82%. It was noticed that there are more wrong estimations in the initial keyframes, as seen in the chart on Fig. 5. GS-IST uses the precision of *Case 1* over time to illustrate this behavior, which is expected since the point cloud has few points in the initial reconstructions. The geometric and statistical analyses are able to identify these early incorrect detections. For instance, in the first three keyframes of *Case 2*, the bottle in the right side is assigned as a sphere, then as a cylinder and later as a sphere again because of the small number of noisy points. For the Efficient RANSAC, the bottle is incorrectly estimated as a sphere twice in three consecutive detections. Using the proposed tracker, the bottle is assigned to the same primitives in the first three keyframes but, each one has a weight based on the geometric analysis. After normalization, the weights of detection history are 0.186 (sphere in the first keyframe), 0.537 (cylinder in the second keyframe) and 0.277 (sphere in the third keyframe). Thus, for GS-IST, the bottle will be assigned as a cylinder because its weight is higher than the 0.463 of the sphere.

Regarding the recall, Fig. 4 displays that GS-IST is worse than Efficient RANSAC in three of the five cases. This happens because Efficient RANSAC outputs twice more shapes on average than GS-IST, even though some of them are incorrect. Thus, the proposed method compromises recall in order to be entirely sure that the most reliable shapes are selected. Using the same bottle as an example, in the third keyframe the cylinder weight is 0.537, which is below the reliability threshold of 0.75. However, since it is above the 0.5 unreliability mark, it is performed a statistical analysis to verify the randomness of this detection. According to Eq. (6), this

[3] Available at http://opencv.org/.
[4] Available at http://cg.cs.uni-bonn.de/aigaion2root/attachments/Software%20v1.1.zip.
[5] Available at https://github.com/rarrafael/vSLAM-dataset.

Table 2
The influence of modifications in GS-IST on the final precision and recall in *Case 1*.

Condition changed	Precision (%)	Recall (%)
Remove geometrical analysis	−1.409	+1.846
Remove statistical analysis	−1.409	+3.139
Double elimination thresholds	−0.704	+2.602

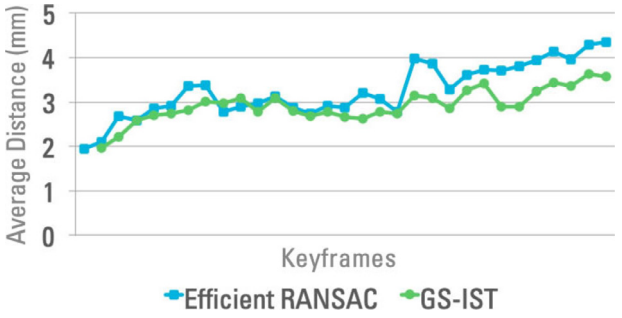

Fig. 6. Average distance in millimeters of each input point to its projection on the estimated primitive over time for *Case 1*.

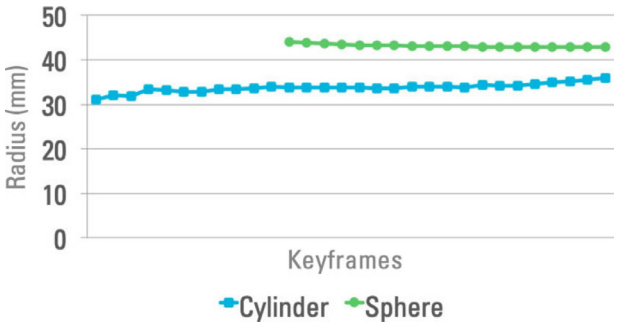

Fig. 7. Radii in millimeters estimated over time of the wipe container (40 mm) and the globe (50 mm) in *Case 1*.

Table 3
Percentage of points that were labeled to each primitive.

Plane	Cylinder	Sphere	Unreliable	Unassigned
41.32%	27.56%	1.97%	22.85%	6.30%

estimation history is random and the shape is assigned as unreliable.

Concerning $F_{0.5}$-Score, Fig. 4 shows that GS-IST presents a better result in all cases. The most significant improvement is in *Case 4*, which is very challenging because the reconstruction is very noisy. This evaluation indicates that the restriction imposed improved the precision, but with the cost of having fewer shapes detected. Therefore, it is possible to adjust the parameters to have more primitives and decrease the precision. These changes will depend on the target application. Table 2 provides some examples of possible modifications to make and the outcome for *Case 1*.

The first and third rows of Fig. 8 present one keyframe from each test case and the estimated shapes using GS-IST[6]. They are represented by the projection of the input points used to compute the primitive. The second and fourth rows display one view of the input point cloud in red and some of the estimated shapes in blue. From the last row, it is possible to see how challenging is *Case 4*. Although the books are aligned in real life, the points from the left one are not aligned with the other two.

4.1. Metric evaluation

Case 1 has a chessboard pattern, which means that the reconstruction can be calibrated to the metric scale. This allowed an accuracy evaluation of object pose and parameters for this case in particular since there is no such pattern in the other cases. Considering the absence of ground truth for pose estimation, it was measured the average distance of each input point to its projection on the estimated primitive to assess how close they were. Fig. 6 compares this distance over time between Efficient RANSAC and GS-IST. The leap in the distance is expected due to the error accumulation of the SLAM method. However, the use of history information when fusing primitives improves the overall result. The average distance was 2.925 ± 0.370 mm for GS-IST while for Efficient RANSAC it was 3.247 ± 0.611 mm. It is worth to mention that this accuracy depends on the quality of the map. In this case, the error accumulation of the SLAM method was not precisely measured but it was around 15 mm, which is compatible with the error of other systems [33].

Concerning the parameter accuracy, it was used the radius of the globe and the wipe container, which are 50 and 40 mm, respectively. The average radius of the sphere detected with the globe points was 43.147 ± 0.318 mm, resulting in an error of 6.853 mm. As for the cylinder estimated based on the wipe container, the 33.723 ± 1.001 mm radius is 6.277 mm smaller than the real object. Fig. 7 shows that in the first detection the cylinder radius is 31.051 mm and it gradually increases closer to the actual measurement over time, ending with 35.952 mm. These are the largest and smallest error in comparison with the ground truth for both the cylinder and the sphere. This can be credited to parameter update over time and the increase in the number of points that, even with the error accumulation, adds more data for the shape extractor. The sphere radius, on the other hand, decreases around 1 mm from the first to the last keyframe, going to the opposite direction of the actual radius. This can also be credited to error accumulation.

4.2. Runtime evaluation

Concerning the computational cost, Efficient RANSAC takes on average 25.474 ± 10.380 ms to estimate the primitives in a computer with a Core i7-6820 (2.70 GHz) and 16GB of RAM. The other steps combined run in 14.995 ± 3.019 ms on average. The bottleneck is the *shape fusion* step, which takes 9.982 ± 2.957 ms of that time. It is worth mentioning that the execution time is related to the number of input points. Therefore, the measurements, which are the averages of all five test cases, were normalized to a group of thousand points.

4.3. Segmentation evaluation

It was also evaluated how GS-IST segments the point cloud, which is a natural outcome of semantic modeling. Several objects have the form of the basic primitives tracked with this method. Looking at an average from all five test cases, 70.85% of the points can be assigned to a plane, sphere or cylinder. Even though the dataset deals with scenes designed with this type of primitives, the number is similar to the study that claims that 78% of all elements in an industrial scenario can be modeled using these three shapes [16]. Moreover, Table 3 shows that only 6.30% of the remaining points were not labeled as any primitive. The other 22.85% points come from primitives that were discarded because they were unreliable.

[6] A video with this result is also available at https://goo.gl/5RGrYm.

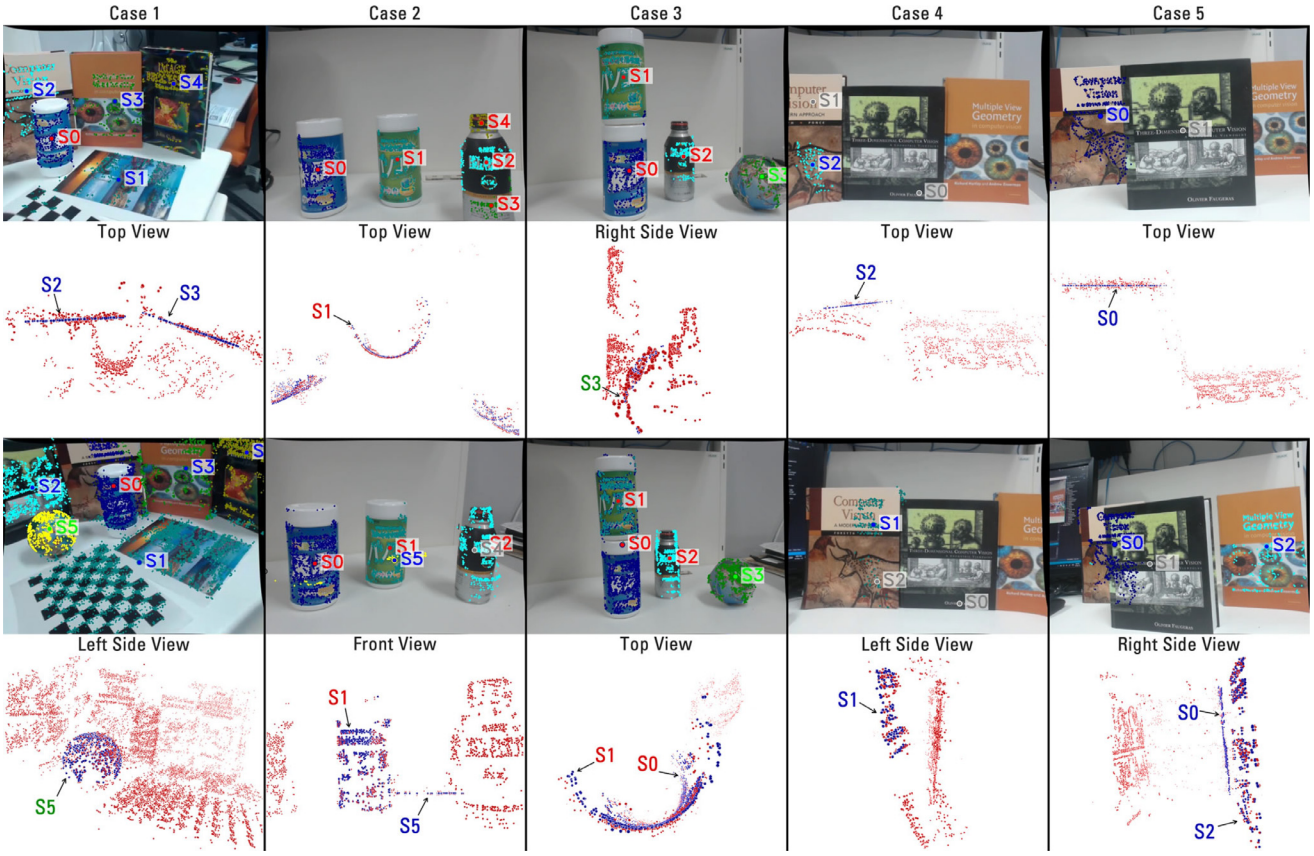

Fig. 8. The first and third rows show the result of GS-IST on each test case. Blue labels represent planes, green ones are for spheres and red for cylinders. The second and fourth rows show one particular view of the input point cloud (in red) and some of the estimated primitives (in blue). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

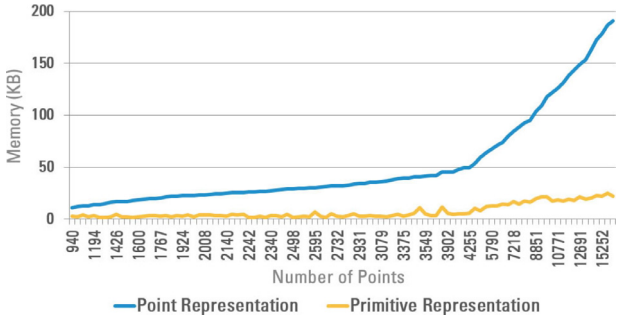

Fig. 9. Memory (in KB) required to describe a scene using the point cloud and the data structure of the detected primitives for *Case 1*.

4.4. Point cloud representation evaluation

Finally, the scene representation using the point cloud was compared to the modeled primitives. The scene is usually overrepresented when it is described using the points because there are many redundant points. The memory necessary to represent the reconstruction of each test case using the point cloud was measured and, later, it was compared with the description of the same map using the data structure of the primitives modeled with GS-IST. Fig. 9 shows this difference in KB between them, ordered by the number of points from the reconstructed map. The most significant difference is in the last keyframe of *Case 1*, which has 16,302 points. Using the point cloud requires 8.69 times more memory than describing the same map using the six detected primitives plus the 3,915 unreliable and unassigned points.

Moreover, describing a scene using points commonly results in over and underrepresentation at the same time. For instance, considering only the cylinder detected in the last keyframe of *Case 1*, it can be noticed that there are more points than necessary to describe the textured front side, but none to represent the back side. Thus, it is possible to use much less information to define this shape while filling the missing parts. Using this cylinder as an example, it is necessary 24 times more memory to describe it using the points than using the data structure of the detected primitive.

5. Mobile implementation

In order to evaluate how GS-IST would perform running on mobile devices, the technique was ported to the Android platform. It was also necessary to compile the libraries used in the project to Android: OpenCV and Boost[7]. Efficient RANSAC was treated as a library as well, but the compilation process was a little bit different due to the fact that it was added to the project in order to facilitate modifications in the source code.

5.1. Evaluation

The two most critical aspects of mobile devices are the energy consumption and the temperature. Although the number of cores and CPU clock have increased lately, the processors' architecture compromise on speed to be more efficient on these two facets [34]. Most of the studies aiming mobile devices focus their evaluation only on execution time along with any qualitative assessment that

[7] Available at https://www.boost.org.

Fig. 10. Results of GS-IST running on a Samsung Galaxy S8 and an ASUS ZenFone 3. Blue labels represent planes, green ones are for spheres and red for cylinders. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

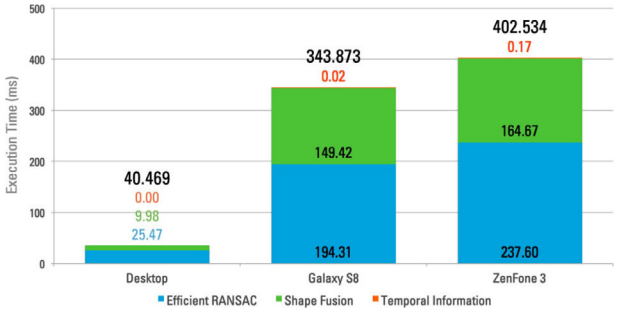

Fig. 11. GS-IST execution time in milliseconds divided by stages on desktop and two different mobile devices.

is adequate for the proposed method. Using the 10 mobile studies cited in Section 2 and the 25 relevant studies listed in Roberto et al. 2016 [24] as a sample, 62.9% measured execution time and 45.7% evaluated only this criteria. Energy consumption was evaluated in 17.1%, RAM memory usage in 8.6% and 28.6% performed qualitative assessments alone. Only Li et al. 2016 [35] evaluate execution time, energy consumption and memory usage. However, they did not detail the methodology used to perform these measurements. These three criteria were used to evaluate GS-IST along with CPU load, which is a good indicator of the potential of parallel execution of a certain application.

This evaluation was executed in devices with different capabilities and from distinct manufacturers, being important to assess how GS-IST performs in dissimilar conditions. The chosen devices were Samsung Galaxy S8 and ASUS ZenFone 3. They were selected using ARCore as a reference. Galaxy S8 is in the list of the supported devices[8] while ZenFone 3 was selected to stress GS-IST since it has a configuration inferior to those in that list. All the tests were executed with the device fully charged, on airplane mode, with all other applications closed and connected to the computer via the USB cable. The only exception was the energy measurement in which the device was disconnected from the computer.

Regarding the dataset, it was used the same five scenes generated in the previous section. The images and pose files were stored in the device and loaded on every frame. The same happened for the map files, which were loaded on every keyframe. Since the codes are identical, precision and recall are equal to the ones presented in the last section. Fig. 10 shows a few keyframes of GS-IST running on both phones[9].

5.1.1. Execution time

The execution time is proportional to the number of points processed and all time measurements, which are the averages of all five test cases, were normalized to a group of thousand points. Fig. 11 shows that the average execution time of GS-IST on ZenFone 3 is 9.9 times slower in comparison with the desktop implementation and 8.5 times slower on the Galaxy S8. For this test, the same desktop computer with Core i7-6820 (2.70 GHz) processor and 16GB of RAM was used. The *shape fusion* was slower than the average on mobile devices. The desktop implementation is 16.5 (ZenFone 3) and 15.0 (Galaxy S8) times faster.

The CPU load is an important measure because it indicates how much room the GS-IST leaves to perform other processing, such as the SLAM technique. For that evaluation, it was used Qualcomm's Trepn Profiler[10]. This application, available in the Play Store, samples the desired information in a constant time interval. In this test, both the CPU load and the normalized CPU load were sampled every 100 milliseconds. The operating system imposes a limit on how much processing an application can use. The CPU load represents how much of that limit is being used by the application while the normalized CPU load indicates how much processing is being used in relation to the total processing power of the device.

It is possible to observe in Fig. 12 that GS-IST presents some execution peaks. These apexes coincide with the keyframes, which are moments in which the technique extracts the primitives. The maximum normalized load for ZenFone 3 was 90% of CPU, similar to the 89% value for Galaxy S8. The average normalized load was also similar for both devices, in which the Samsung device was 30.545% ± 25.119% of normalized CPU load while the ASUS mobile phone presented 27.128% ± 16.353%. This happens because for most of the time the execution is in between keyframes, a moment in which the device is not processing much data. The median of the normalized CPU load is a numerical indication for that and it was 19% and 21% for Galaxy S8 and ZenFone 3, respectively. However, there was a difference in the average CPU load, which was 39.283% ± 27.131% for the Samsung phone and 46.483% ± 25.040% for the ASUS device. Smaller differences between the CPU load and the normalized CPU load suggest that the device is allowed to use the full potential of the processor. In that case, this value was 8.738% for Galaxy S8, but it was 19.355% for ZenFone 3.

5.1.2. Energy consumption

The most precise method to evaluate energy consumption is by using external instruments that can directly measure the current drained by the device. However, this equipment requires opening the device to be attached to the physical battery, which is difficult for most smartphones nowadays since their batteries are not easily accessible. An alternative is to use profiler tools that use the battery API to assess the voltage and the state of charge at certain intervals. This procedure is much more accessible, but is not as accurate as using these external instruments. The latter approach was selected for this evaluation and Trepn Profiler was also used for this task. Qualcomm's tool has an accuracy of 99%, which is reported to be one of the highest for such profilers [36].

As expected, Fig. 13 shows that energy consumption on both mobile devices follows the same pattern of the CPU load since more energy is required in those most computational-intense moments. The average consumption on ZenFone 3 was 1.776 ± 1.162 W every 100 ms. Since the battery capacity of this device

[8] Available at https://developers.google.com/ar/discover/supported-devices.

[9] A video with this result is also available at https://goo.gl/A6T51d.

[10] Available at https://developer.qualcomm.com/software/trepn-power-profiler.

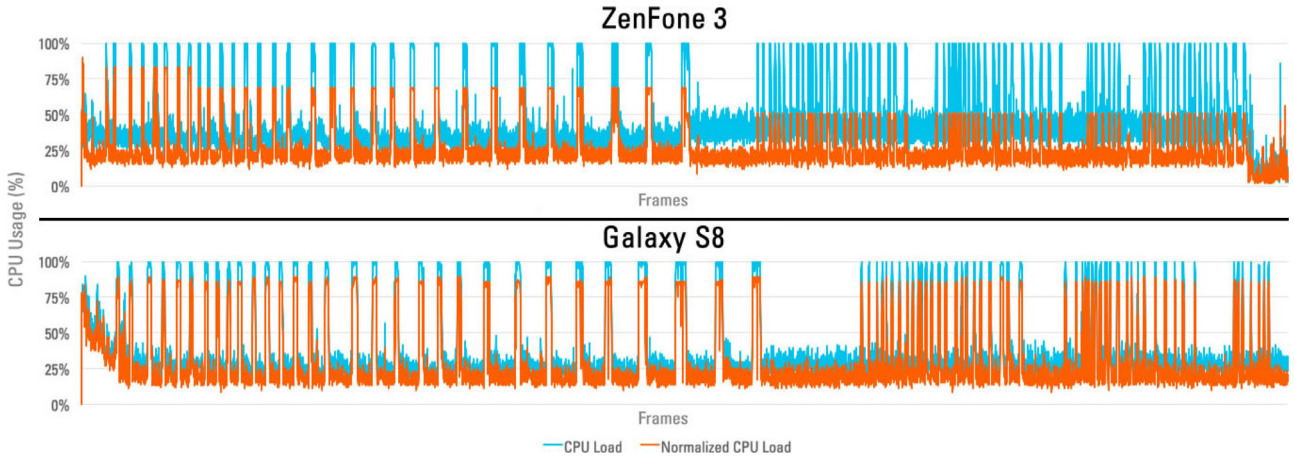

Fig. 12. CPU load and normalized CPU load over time of all five test cases on a ZenFone 3 (top) and Galaxy S8 (bottom).

Fig. 13. Energy consumption (in W) over time of all five test cases on ZenFone 3 (top) and Galaxy S8 (bottom).

is 3000 mAh with 3.85 V, this means that this device could run GS-IST for 5 hours and 12 minutes before draining all the battery when it is fully charged considering an energy efficiency of 80% [37]. The Galaxy S8 battery has the same characteristics (3000 mAh and 3.85 V), which determines that its average 2.271 ± 1.998 W consumption would drain a fully charged battery in 4 hours and 04 minutes.

5.1.3. Memory usage

Concerning the memory usage, two measurements can be done. One is the storage space the sample APK requires when installed and the other is the RAM memory it uses when running. The former value can easily be found in the device settings. In ZenFone 3, GS-IST used 45.14 MB of the storage space while in Galaxy S8 it occupied 47.38 MB.

To evaluate the latter it was used the Android Profiler available on Android Studio, which builds a chart with the RAM memory usage as the application is executed. Similar to the previous evaluation, Fig. 14 shows that the RAM memory has some peaks when extracting the primitives. For ZenFone 3, the moment with most memory usage is in the 28th keyframe of *Case 1* with 195.39 MB, which represents 6.4% of the device total memory. When not processing the keyframes, the sample app consumes between 26.14 MB and 38.19 MB.

As seen in Fig. 14, the memory usage for the Galaxy S8 is similar, although with higher absolute values. The memory peak was 322.97 MB in the 25th keyframe of *Case 1*, which constitutes to 7.9% of the phone RAM memory. The memory consumption when not extracting primitives was above 119.07 MB and below 151.16 MB.

5.2. Discussion

From Fig. 11 it is possible to see that GS-IST does not run in real-time on any of the devices it was tested. This is especially true when the number of points increases. However, this approach uses less than a third of the CPU power regardless of the device. There are SLAM systems for mobile devices that run in real-time, such as ARCore and ARKit. This means that GS-IST and a SLAM technique can run in different threads in a way that they can interact with each other without compromising their performance. Since this mobile version is running on Android devices, it is possible to use ARCore as a SLAM method and extract the primitives from the point cloud the SDK generates. Nevertheless, it is necessary to perform an evaluation to see if the ARCore map is too sparse. Additionally, there is room for optimization in the implementation.

Moreover, Fig. 12 shows that Android 7.0 on ZenFone 3 imposed a more strict limit on how much processing GS-IST could use. In fact, that limit increased over time. For instance, GS-IST could use 90% of the CPU at the beginning of the execution and that ceiling decreased to approximately 75% in the 9th keyframe of *Case 1* and to 50% from *Case 2* and beyond. On the other hand, Android 8.0

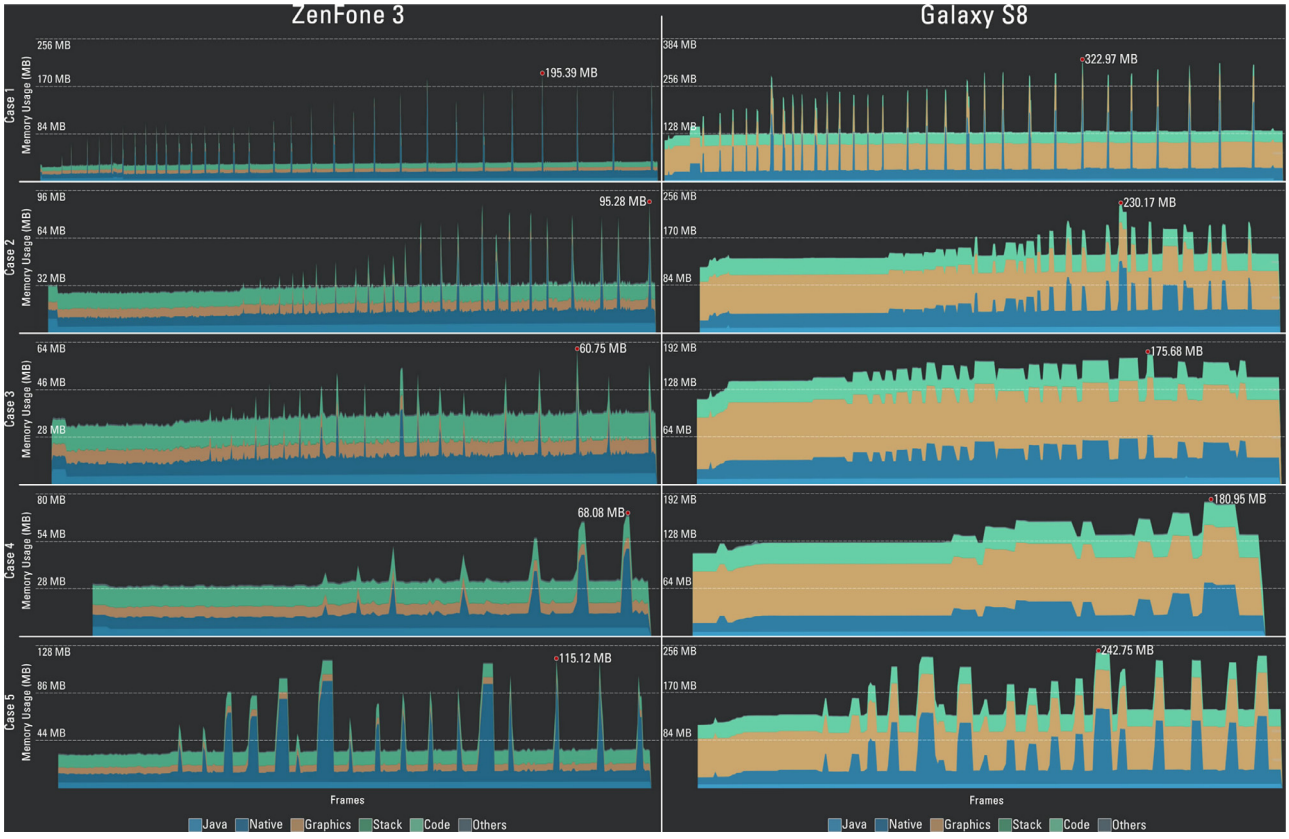

Fig. 14. Memory usage over time of GS-IST running on ZenFone 3 (left) and Galaxy S8 (right). Each test case has a different time scale.

Table 4
Time that different applications take to fully drain a battery fully charged on ZenFone 3 and Galaxy S8 devices.

Application	ZenFone 3	Galaxy S8
1080p video	8h07min	6h31min
Google Maps	6h32min	4h29min
GS-IST	5h12min	4h04min
Fifa Soccer	5h09min	4h47min
Measure (ARCore)	Not supported device	3h01min

on Galaxy S8 allowed GS-IST to use around 90% of the CPU from beginning to end.

This difference in the CPU usage impacts energy consumption because they are directly related, as seen in Figs. 12 and 13. Thus, the ZenFone 3 saves battery when the operating system limits the processing capabilities at the coast of taking more time to extract the primitives. The energy saved when reducing the CPU load is larger than the energy necessary to execute the method slower for more time. The 4 hours GS-IST needs to drain the battery in the worst case is sufficient to use this technique without having extra concerns about the battery. Moreover, considering the habits and the average time users spend on mobile devices [38], it is unlikely that a person would use a specific app for such a long period. In order to put this in perspective, Table 4 compares the time other common applications take to fully discharge the battery, such as watching a 1080p video, navigating with Google Maps using 4G network and playing FIFA Soccer using WiFi connection. There is also a comparison with Google's Measure app, which is essentially ARCore with a very simple rendering. Besides the video activity, GS-IST has an energy consumption similar to the other applications.

The evaluation showed a noticeable difference in RAM memory usage between both devices. The ASUS phone used approximately 100 MB less memory than the Samsung one. It is possible to see in Fig. 14 that graphics uses much more memory in Galaxy S8 than in ZenFone 3. The graphics are responsible for more than 70% of that difference. It was not found any precise information explaining this increase in the graphics memory consumption. However, based on observation using other devices, one possibility is that the ASUS device delegates the rendering activity to the GPU, therefore, graphics-related structures use the GPU memory.

The memory usage of GS-IST was not a concern since, in the worst case, it used less than 8% of the total RAM memory of the device. This is due to the fact that modern smartphones have a fair amount of RAM memory. For them, replacing the representation from point clouds to primitives does not have any impact on the execution time. However, this change in representation can be important in some situations. Complex algorithms can use a lot of memory and perform several computations for each element in the scene, which can overload the powerful devices even for a sparse map. Tracking optimization methods have that characteristic. In fact, some developers reported that version 1.2.1 of ARCore can run out of memory when it has to perform bundle adjustment with a map whose size is that of a large room. Rendering algorithms are another case in which having a large number of elements can cause the usage of most of the device's resources. Therefore, a more efficient representation can be very helpful in similar circumstances.

6. Conclusion

This work presents a new technique that detects and tracks geometric primitives, called GS-IST. This method uses the generating process of sparse point clouds of visual SLAM systems and applies

geometrical and statistical analyses to incrementally estimate and track planes, spheres and cylinders. The evaluation indicated that GS-IST improved precision in all test cases, outperforming existing methods in this criteria. The developed approach focuses on precision and for that, it compromises recall to assure we have the correct shapes. However, we can modify the parameters to increase recall when necessary. Additionally, this technique was ported to the Android platform and evaluated to assess how it performed running on mobile devices. The evaluation showed that the mobile version is slower when compared with the desktop implementation, but it can be executed on a separate thread of the SLAM technique because the CPU load is not so high. Finally, the energy consumption and memory usage were not a concern.

As future studies, we are working on the integration of GS-IST with Google's ARCore and Apple's ARKit, which will allow the creation of new test cases in order to perform evaluation on more complex environments. This integration allows us to use the generated 3D map to compare GS-IST and ARCore plane detection. Further activities include performing a more extensive evaluation of the accuracy of object tracking and its parameters. In order to accomplish that, it is necessary to have a dataset with ground truth pose and measurements. This can be achieved with the creation of other scenes that would include a chessboard pattern or other means to recover the scale. Another idea for future work is to use the semantic knowledge of the scene to improve the tracking results of the visual SLAM system. There are some ways to achieve this goal. One is to constrain the map 3D points during the bundle adjustment to move only over the surface of the shape it belongs. A different strategy is to optimize the map using the primitive parameters instead of the points.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

The authors would like to thank Clemens Arth and Dieter Schmalstieg for the valuable discussions. This work was partially funded by JSPS KAKENHI (grant number JP17H01768).

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.cag.2019.09.003.

References

[1] Ramadasan D, Chateau T, Chevaldonn M. Dcslam: A dynamically constrained real-time slam. In: Proceedings of the 2015 IEEE international conference on image processing (ICIP); 2015. p. 1130–4. doi:10.1109/ICIP.2015.7350976.

[2] Hettiarachchi A, Wigdor D. Annexing reality: Enabling opportunistic use of everyday objects as tangible proxies in augmented reality. In: Proceedings of the 2016 CHI conference on human factors in computing systems. CHI '16. New York, NY, USA: ACM; 2016. p. 1957–67. ISBN 978-1-4503-3362-7. doi:10.1145/2858036.2858134.

[3] Halpern M, Zhu Y, Reddi VJ. Mobile CPU's rise to power: quantifying the impact of generational mobile CPU design trends on performance, energy, and user satisfaction. In: Proceedings of the 2016 IEEE international symposium on high performance computer architecture (HPCA); 2016. p. 64–76. doi:10.1109/HPCA.2016.7446054.

[4] Roberto R, Lima JP, Uchiyama H, Arth C, Teichrieb V, Taniguchi Ri, Schmalstieg D. Incremental structural modeling based on geometric and statistical analyses. In: Proceedings of the 2018 IEEE Winter conference on applications of computer vision (WACV); 2018. p. 955–63. doi:10.1109/WACV.2018.00110.

[5] Holz D, Holzer S, Rusu RB, Behnke S. Real-time plane segmentation using RGB-d cameras. In: RoboCup 2011: robot Soccer World Cup XV. Berlin, Heidelberg: Springer Berlin Heidelberg; 2012. p. 306–17. ISBN 978-3-642-32060-6. doi:10.1007/978-3-642-32060-6_26.

[6] Drost B, Ilic S. Local hough transform for 3d primitive detection. In: Proceedings of the 2015 international conference on 3D vision; 2015. p. 398–406. doi:10.1109/3DV.2015.52.

[7] Lopez-Escogido D, de la Fraga LG. Automatic extraction of geometric models from 3d point cloud datasets. In: Proceedings of the 2014 11th international conference on electrical engineering, computing science and automatic control (CCE); 2014. p. 1–5. doi:10.1109/ICEEE.2014.6978316.

[8] Nguyen T, Reitmayr G, Schmalstieg D. Structural modeling from depth images. IEEE Trans Visual Comput Gr 2015;21(11):1230–40. doi:10.1109/TVCG.2015.2459831.

[9] Oehler B, Stueckler J, Welle J, Schulz D, Behnke S. Efficient multi-resolution plane segmentation of 3d point clouds. In: Proceedings of the 4th International Conference, ICIRA 2011 intelligent robotics and applications, Aachen, Germany, December 6–8, 2011, Proceedings, Part II. Berlin, Heidelberg: Springer Berlin Heidelberg; 2011. p. 145–56. ISBN 978-3-642-25489-5. doi:10.1007/978-3-642-25489-5_15.

[10] Kim YM, Dolson J, Sokolsky M, Koltun V, Thrun S. Interactive acquisition of residential floor plans. In: Proceedings of the 2012 IEEE international conference on robotics and automation; 2012. p. 3055–62. doi:10.1109/ICRA.2012.6224595.

[11] Liu YJ, Zhang JB, Hou JC, Ren JC, Tang WQ. Cylinder detection in large-scale point cloud of pipeline plant. IEEE Trans Visual Comput Gr 2013;19(10):1700–7. doi:10.1109/TVCG.2013.74.

[12] Qiu R, Zhou Q-Y, Neumann U. Pipe-run extraction and reconstruction from point clouds. In: Proceedings of the computer vision – ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6–12, 2014, Proceedings, Part III. Cham: Springer International Publishing; 2014. p. 17–30. ISBN 978-3-319-10578-9. doi:10.1007/978-3-319-10578-9_2.

[13] Li Y, Wu X, Chrysathou Y, Sharf A, Cohen-Or D, Mitra NJ. Globfit: consistently fitting primitives by discovering global relations. ACM Trans Graph 2011;30(4):52:1–52:12. doi:10.1145/2010324.1964947.

[14] Pang G, Qiu R, Huang J, You S, Neumann U. Automatic 3d industrial point cloud modeling and recognition. In: Proceedings of the 2015 14th IAPR international conference on machine vision applications (MVA); 2015. p. 22–5. doi:10.1109/MVA.2015.7153124.

[15] Pang G, Neumann U. Training-based object recognition in cluttered 3d point clouds. In: Proceedings of the 2013 international conference on 3D vision - 3DV 2013; 2013. p. 87–94. doi:10.1109/3DV.2013.20.

[16] Huang J, You S. Detecting objects in scene point cloud: a combinational approach. In: Proceedings of the 2013 international conference on 3D vision - 3DV 2013; 2013. p. 175–82. doi:10.1109/3DV.2013.31.

[17] Stanescu A, Fleck P, Schmalstieg D, Arth C. Semantic segmentation of geometric primitives in dense 3d point clouds. In: Proceedings of the 2018 IEEE international symposium on mixed and augmented reality adjunct (ISMAR-Adjunct); 2018. p. 206–11. doi:10.1109/ISMAR-Adjunct.2018.00068.

[18] Tateno K, Tombari F, Laina I, Navab N. Cnn-slam: Real-time dense monocular slam with learned depth prediction. In: Proceedings of the 2017 IEEE conference on computer vision and pattern recognition (CVPR); 2017. p. 6565–74. doi:10.1109/CVPR.2017.695.

[19] Sinha SN, Steedly D, Szeliski R, Agrawala M, Pollefeys M. Interactive 3D architectural modeling from unordered photo collections. ACM Trans Graph 2008;27(5):159:1–159:10. doi:10.1145/1409060.1409112.

[20] Schnabel R, Wahl R, Klein R. Efficient RANSAC for point-cloud shape detection. Comput Gr Forum 2007;26(2):214–26. doi:10.1111/j.1467-8659.2007.01016.x.

[21] Roth G, Levine MD. Extracting geometric primitives. CVGIP: Image Underst 1993;58(1):1–22. doi:10.1006/ciun.1993.1028.

[22] Mur-Artal R, Montiel JMM, Tard?s JD. ORB-SLAM: a versatile and accurate monocular slam system. IEEE Trans Robotics 2015;31(5):1147–63. doi:10.1109/TRO.2015.2463671.

[23] Uchiyama H, Taketomi T, Ikeda S, Lima JPSd M. [POSTER] abecedary tracking and mapping: a toolkit for tracking competitions. In: Proceedings of the ISMAR; 2015. p. 198–9. doi:10.1109/ISMAR.2015.63.

[24] Roberto R, Lima JP, Teichrieb V. Tracking for mobile devices: a systematic mapping study. Comput Gr 2016;56:20–30. https://doi.org/10.1016/j.cag.2016.02.002.

[25] Chen TY-H, Ravindranath L, Deng S, Bahl P, Balakrishnan H. Glimpse: continuous, real-time object recognition on mobile devices. In: Proceedings of the 13th ACM conference on embedded networked sensor systems. SenSys '15. New York, NY, USA: ACM; 2015. p. 155–68. ISBN 978-1-4503-3631-4. doi:10.1145/2809695.2809711.

[26] Sandler M, Howard A, Zhu M, Zhmoginov A, Chen L. Mobilenetv2: inverted residuals and linear bottlenecks2018:4510–4520, doi:10.1109/CVPR.2018.00474.

[27] Tobas L, Ducournau A, Rousseau F, Mercier G, Fablet R. Convolutional neural networks for object recognition on mobile devices: a case study. In: Proceedings of the 2016 23rd international conference on pattern recognition (ICPR); 2016. p. 3530–5. doi:10.1109/ICPR.2016.7900181.

[28] Wang J, Cao B, Yu P, Sun L, Bao W, Zhu X. Deep learning towards mobile applications. In: Proceedings of the 2018 IEEE 38th international conference on distributed computing systems (ICDCS); 2018. p. 1385–93. doi:10.1109/ICDCS.2018.00139.

[29] Runceanu L, Becker S, Haala N, Fritsch D. Indoor point cloud segmentation for automatic object interpretation. Wissenschaftlich-Technische Jahrestagung der DGPF in Wrzburg 2017:147–59. https://www.semanticscholar.org/paper/Indoor-Point-Cloud-Segmentation-for-Automatic-Runceanu-Becker/181fd778e36b935479a3eb33abe2d68dbc876f12.

[30] Sankar A, Seitz SM. Interactive room capture on 3d-aware mobile devices. In: Proceedings of the 30th annual ACM symposium on user interface software

and technology. UIST '17. New York, NY, USA: ACM; 2017. p. 415–26. ISBN 978-1-4503-4981-9. doi:10.1145/3126594.3126629.

[31] Roberto RA, Uchiyama H, Lima Ja P S M, Nagahara H, Taniguchi R-i, Teichrieb V. Incremental structural modeling on sparse visual SLAM. IPSJ Trans Comput Vis Appl 2017;9(1):5. doi:10.1186/s41074-017-0018-3.

[32] Sheskin DJ. Handbook of parametric and nonparametric statistical procedures, Fifth Edition. Taylor & Francis; 2011. ISBN 9781439858011. https://books.google.com.br/books?id=YDd2cgAACAAJ.

[33] Roberto R, Lima JP, Arajo T, Teichrieb V. Evaluation of motion tracking and depth sensing accuracy of the tango tablet. In: Proceedings of the 2016 IEEE international symposium on mixed and augmented reality (ISMAR-Adjunct); 2016. p. 231–4. doi:10.1109/ISMAR-Adjunct.2016.0082.

[34] Reiner H. The paramountcy of reconfigurable computing. In: Energyfficient distributed computing systems; chap. 18. Wiley-Blackwell; 2012. p. 465–547. ISBN 9781118342015. doi:10.1002/9781118342015.ch18.

[35] Li D, Salonidis T, Desai NV, Chuah MC. Deepcham: collaborative edge-mediated adaptive deep learning for mobile object recognition. In: Proceedings of the 2016 IEEE/ACM Symposium on Edge Computing (SEC); 2016. p. 64–76. doi:10.1109/SEC.2016.38.

[36] Hoque MA, Siekkinen M, Khan KN, Xiao Y, Tarkoma S. Modeling, profiling, and debugging the energy consumption of mobile devices. ACM Comput Surv 2015;48(3):39:1–39:40. doi:10.1145/2840723.

[37] Valoen L, Mark IS. The effect of PHEV and HEV duty cycles on battery and battery pack performance. In: Plugin Highway 2007 Conference; 2007. p. 1–9. http://umanitoba.ca/outreach/conferences/phev2007/PHEV2007/proceedings/PluginHwy_PHEV2007_PaperReviewed_Valoen.pdf.

[38] Noon H. How much time do people spend on their mobile phones in 2017?. 2017. https://goo.gl/bGLVSZ [Online; last access: 21-May-2018].