An evaluation of real-time RGB-D visual odometry algorithms on mobile devices

Vincent Angladon, Simone Gasparini, Vincent Charvillat, Tomislav Pribanić, Tomislav Petković, Matea Đonlić, Benjamin Ahsan, Frédéric Bruel

An Evaluation of Real-Time RGB-D Visual Odometry Algorithms on Mobile Devices

Vincent Angladon[1,2], Simone Gasparini[1], Vincent Charvillat[1], Tomislav Pribanić[3], Tomislav Petković[3], Matea Đonlić[3], Benjamin Ahsan[2], and Frédéric Bruel[2]

[1]Université de Toulouse; INPT – IRIT; 118 Route de Narbonne, F-31062 Toulouse, France,
{vincent.angladon, simone.gasparini, vincent.charvillat}@irit.fr
[2]Telequid; Toulouse, France,
{benjamin.ahsan, frederic.bruel}@telequid.fr
[3]Faculty of Electrical Engineering and Computing, University of Zagreb, Unska 3, HR-10000 Zagreb, Croatia,
{tomislav.pribanic, tomislav.petkovic.jr, matea.donlic}@fer.hr

Abstract

We present an evaluation and a comparison of different Visual Odometry algorithms, selected to be tested on a mobile device equipped with a RGB-D camera. The most promising algorithms from the literature are tested on different mobile devices, some equipped with the Structure Sensor. We evaluate the accuracy of each selected algorithm as well as the the memory and CPU consumption, and we show that even without specific optimization, some of them can provide a reliable measure of the device motion in real time.

Index Terms— RGB-D Camera; Visual Odometry; Benchmark; Mobile device

Todo list

1 Introduction

In the past years, we have witnessed the development of consumer grade depth sensors such as the Kinect for Xbox 360 (Kinect^{SL}). Although these sensors were initially meant for gaming and entertainment, they found a large interest in various communities, such as computer vision, robotics, and biomechanic communities, as they can provide 3D data at relatively low cost. Khoshelham *et al.* [24] evaluated experimentally the accuracy of this technology and proposed a noise model which explains why the depth error grows quadratically with the distance to the objects. The accuracy also depends on the tilt of the surface normal w.r.t. the camera viewpoint [33] and the properties of the object material.

In an escalating trend, manufacturers are now focusing their efforts on reducing the size of these sensors in order to offer mobile devices the possibility to better sense and understand the world around them. PrimeSense proposed first the now discontinued Capri 1.25 embeddable depth sensor, which was followed by the Structure Sensor [37] and now the Structure Core from Occipital both. Other companies including Intel ®, Mantis Vision, SoftKinectic, Inuitive and Oriense Tech, started at the same time to follow this direction. Some early prototypes and products with integrated depth sensors were already released, such as the Google Tango Peanut and Yellowstone devices [16], the Dell Venue 8 700 serie tablet, Intel ® RealSense ™ Smartphone, and the now discontinued Aquila tablet from Mantis Vision.

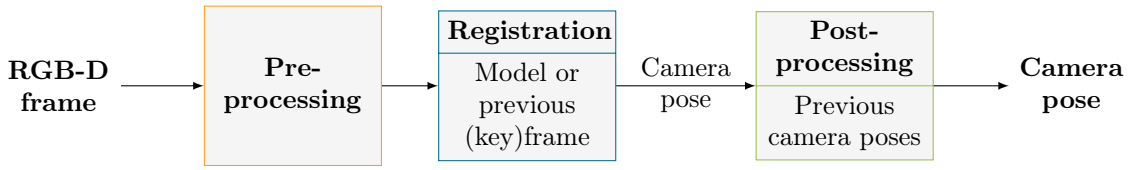

Figure 1: The different components of a VO pipeline.

Embedding depth sensors on everyday mobile devices can foster a whole new range of consumer applications, from 3D reconstruction to augmented and virtual reality. While these problems have been thoroughly addressed in the literature using desktop computers, the limited computational power and hardware of mobile devices set new challenges for adapting or designing new and more efficient algorithms able to process the depth data of these sensors. In this paper we propose a benchmark and an evaluation of state-of-the-art Visual Odometry (VO) algorithms suitable for running in real time on mobile devices equipped with an RGB-D sensor. Visual odometry is the task of estimating the 3D pose (*i.e.*, its position and orientation) of the camera from visual data [34] and, more in general, from RGB-D data exploiting the 3D information provided by depth sensors [32]. VO is a key component for any computer vision and robotic application requiring 3D reconstruction of the surrounding environment, such as augmented reality or mobile navigation. It is then interesting to evaluate the performances of current algorithms on mobile settings. In particular, we selected five real-time VO algorithms among the most accurate in the literature and for which an implementation was available. We first evaluated and compared their performances in terms of accuracy and resource consumption on desktop settings using a standard dataset for the evaluation of RGB-D VO algorithms [57]. We then performed a similar evaluation on mobile settings, specifically on a Structure Sensor [37]. To this purpose we recorded a new dataset with various luminosity levels and assessed the accuracy as well as the memory and CPU usage of each of the selected algorithm. Although similar benchmarks can be found in the literature for desktop environment [31, 13], to the best of our knowledge this is the first attempt at benchmarking state-of-the-art algorithms on a mobile device equipped with a depth sensor.

The paper is organized as follows: Section 2 gives a brief background on VO and a taxonomy of the different approaches. Section 3 reviews some related works on benchmarking VO algorithms, while Section 4 briefly introduces the methods used for this evaluation. Section 5 presents the process of algorithms selection, followed by a mobile benchmark in Section 6. Finally, Section 8 concludes the paper with some final remarks and perspectives.

2 Visual odometry

Visual Odometry is an incremental process which estimates the 3D pose of the camera from visual data. The latest visual frame is registered against previous data to compute a rigid body transformation between the current and the previous pose. Similarly to wheel odometry, the accuracy of the current pose depends on the reliability of the previous pose. Therefore VO is prone to accumulation of errors and the resultant trajectory can only be considered locally consistent.

A VO algorithm can be qualified small or large-baseline depending on its ability to handle large changes of camera viewpoint between consecutive frames. Large camera displacement can occur during fast camera movements or with a low camera frame rate. In this paper, we will focus on small-baseline RGB-D VO using Primesense-based depth sensors with the assumption of static environments.

Generally, a VO algorithm can be seen as a registration procedure with a preprocessing and a post-processing step as depicted in Figure 1. The depth maps provided by depth sensors are noisy and can have missing values due to occlusions. Some VO implementation [32, 46] add a filtering preprocessing step in order to enhance the depth maps and improve the registration step. To this end, bilateral filters [59] can be applied to the raw depth data to reduce the noise and the missing data, yet preserving the discontinuities.

The registration process takes as input the latest RGB-D frame and a previous frame (or a model) to compute the current pose of the camera. There are different strategies for aligning two frames. In

the *frame-to-frame* matching strategy [23, 56], the current RGB-D frame is aligned with the previous one. This strategy quickly leads to a large drift of the estimated trajectory as the pose errors are cumulated. To mitigate this effect, the *frame-to-keyframe* strategy samples the sequence of RGB-D frames into keyframes, usually having a larger spatial displacement among them [20, 35, 36]. The current frame is then aligned w.r.t. to the previous keyframe, until a new keyframe is selected. The selection of the keyframe is important in order to have an homogeneous sampling of the scene, and it often relies on an heuristic evaluating the image quality (*e.g.* no motion blur) or the redundancy of the visual information. For example, in [20] a threshold on the number of matched visual features is used as an indicator of the overlapping part of the scene between the frames and of the movement of the camera. Other methods [37] uses the IMU sensor and a threshold on the estimated rotations and translations to select the keyframe. Another strategy, called *frame-to-model*, consists in building a model of the explored scene and using this model to align the new frames. The model can be a sparse 3D point cloud, as *e.g.* for CCNY [11], or a voxel grid of TSDF (Truncated Signed Distance Function) for KinectFusion [32]. For the latter model, a synthetic view of the surface is generated, usually in the form of a depth map at a predicted camera pose to perform the registration. This strategy significantly reduces the small-scale drift and it is more accurate than the *frame-to-keyframe* strategy [32]. Moreover, *frame-to-model* strategy allows to recover after tracking failures and relocalize the device w.r.t. the model [6]. On the other hand, they still suffer from large-scale drift and may require a heavy memory usage, which can be reduced by using only a subset of the model. For example, CCNY [11] subsamples the 3D point cloud and Kintinuous [60] only loads the part of the scene taken into consideration. Aligning a frame to a model requires more computation than aligning a frame with another one. To speed up the registration process, it is necessary to take into consideration only the part of the model that has an overlap with the current frame. A common approach for the voxel grid of TSDF is to generate a depth map from the model at a predicted pose and to compare the current frame with this depth map [32, 38]. We refer the reader to [31] for more detailed information on the different registration methods.

On some *(key)frame-to-frame* VO approaches [64], a local optimization post-processing step is added to refine the latest camera pose and reduce the trajectory drift. It cannot be applied on *frame-to-model* strategies, unless the model can be updated when the previous camera poses are refined by the optimization process.

During the registration process, only two frames are taken into account in order to efficiently compute the camera pose at the current time with a closed form expression. The idea is that the current pose could have been computed with earlier frames than the previous (key)frame. The local optimization process takes several (key)frames as input of an optimization problem, often a windowed bundle adjustment problem, and returns refined camera poses [64]. When this optimization is performed on the whole trajectory, the purpose is to ensure a global consistency of the trajectory, and the algorithm enters in the visual Simultaneous Localization and Mapping (vSLAM) category. As the name suggests, vSLAM algorithms aim at building a map of the environment and, at the same time, localizing the device. Typically, this is achieved thanks to a place recognition module that can identify previously visited areas and perform the loop closure, thus ensuring global consistency of the map and the trajectory. VO can be considered as vSLAM without place recognition (and, hence, unable to perform loop closure). We refer the reader to [52] for a more detailed survey on camera pose optimization for monocular VO.

A taxonomy for VO registration approaches has been proposed in [13], classifying the approaches into three main categories: *image-based*, *depth-based* and *hybrid-based* (see Figure 2). In the following we briefly summarize each category w.r.t. the methods evaluated in this paper. We also recommend Yousif *et al.* review [62] on RGB-D VO algorithms which includes monocular VO and vSLAM algorithms.

The *image-based* methods rely on the information of the RGB image [20, 23, 45] and it can be further divided into *feature-based* methods and *direct* methods. The formers are sparse methods as they use local image features to register the current frame w.r.t. a previous (key)frame. On desktop computers, SIFT [29] and SURF [3] features are commonly used for their high robustness [12]. On the other hand, their computational cost makes them unsuitable for mobile devices Hudelist *et al.* [21], and other computationally cheap features such as BRISK [27], BRIEF [8] and ORB [49] must be used. These methods perform well in highly textured scenes while they tend to fail in poor light conditions

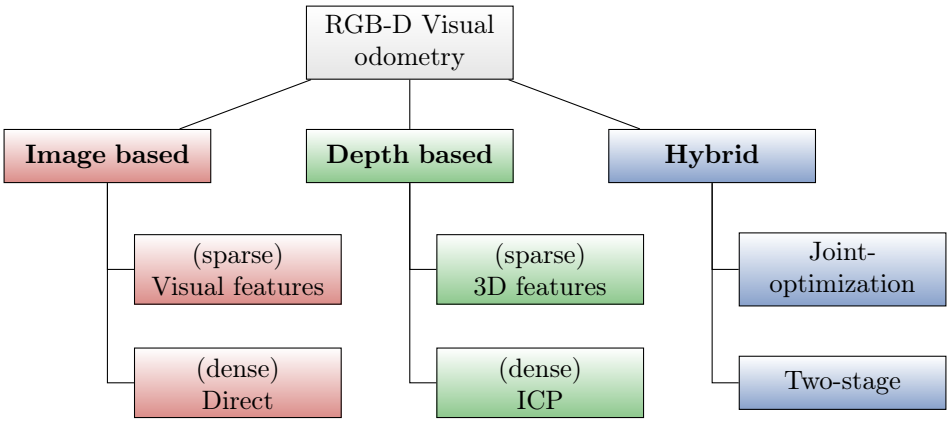

Figure 2: Summary of the three classes taxonomy of registration approaches proposed for RGB-D VO.

and under fast motion of the camera, as the features are not robust to motion blur. Moreover, the features are generally located at objects boundaries where the depth information provided by sensors based on Structured Light technology [37] is the least reliable, thus affecting the registration accuracy. The *direct* methods [23] are instead dense method, as the registration uses all the pixels of the images. Under the assumption that the luminosity of the pixels is invariant to small viewpoint changes, they estimate the camera motion that maximizes a photo-consistency criterion between the two considered RGB-D frames. These methods works even in poor light conditions and low textured scenes, and they can handle object occlusions. On the other hand, the viewpoint displacement between the considered frames must be small, thus limiting the range of application to smooth and relatively slow movements.

The *depth-based* algorithms rely mostly on the information of the depth images [56, 45, 35]. The sparse *3D feature-based* methods rely on the extraction of salient features on the 3D point clouds. The rigid body transform can be computed by matching the descriptors associated to the features extracted in two frames. As for the feature-image-based algorithms, the majority of these features are located at objects boundaries and areas with high curvatures. Again, due to the limitations of the Structured Light technology, the depth values have low accuracy or can be missing in these areas, leading to bad repeatability of the features and poor registration accuracy. The Iterative Closest Point (*ICP*) methods refer to a class of registration algorithms which try to iteratively minimize the distance between two point clouds without knowing the point correspondences [9, 5]. The alignment error is computed with a given error metric such as point-to-point or point-to-plane distance, and the process is repeated until this error converges or the maximal number of iterations is reached. Each iteration improves the point clouds alignment, which in return enables the heuristic association function to output more correct matches, and so on. Weighting strategies [50] are used for robust registration and filtering outliers due to sensor noise or the non overlapping parts of the 3D point clouds. Similarly to the direct-image-based methods, ICP converges well under the assumption of small viewpoint changes, as it avoids local minima and converges to desired solution. Coarse-to-fine approach have also been proposed to improve the convergence [32, 44]. For an exhaustive review of ICP algorithms, we refer the reader to [39]. *Depth-based* algorithms can work well in poor light conditions as they rely on the 3D data, but on the other hand they might fail with scenes having low structure (*e.g.* only few planar surfaces).

Finally, *hybrid* algorithms try to combine the best of the two worlds in order to handle scenes having either low structure or little texture [43, 36]. They can be divided into *two-stage* methods and *joint-optimization* methods. The two-stage methods use one approach (usually a sparse method) to compute an initial guess of the registration, and use a second approach (usually a dense method) to refine the transformation or just compute it in case of failure of the first approach [11]. The joint-optimization strategy consists in designing an optimization problem which combines equations from depth-based and image-based approaches [43, 60].

2.1 VO on mobile devices

Developing real-time VO algorithms is more challenging on mobile devices due to their limited memory and processing power. Fine optimizations can be performed using SIMD instructions of the embedded CPU and OpenGL ES shaders can be used for processing parallelizable tasks on the GPU. However this highly increases the complexity of the implementation and requires low-level programming skills. On modern mobile devices, one can also take advantage of the Inertial Motion Unit (IMU), and eventually integrate the estimated rotation as a prior knowledge into the registration algorithm.

Regarding monocular VO algorithms, Schöps *et al.* [53] achieved a 30 FPS tracking performance with a partial porting of the semi-dense LSD-Slam algorithm on a Sony Xperia Z1 phone. No code was publicly released though. Commercial solutions also emerged in the past years, proposed by 13th Lab, Metaio and RealityCap, before their recent acquisition by Occulus VR, Apple and Intel ® respectively. The Google Project Tango [16] also proposes a proprietary monocular Visual and Inertial Odometry (VIO) algorithm. It is designed for dedicated hardware using in particular a fisheye camera such as the Tango Yellowstone tablet and the Intel RealSense® smartphone™.

Lately, with the recent development of depth sensors for mobiles such as the Structure Sensor [37] and Mantis Vision MV4D [30], new proprietary RGB-D VO algorithms for mobile devices have been developed and they are available through their relevant SDKs. Presumably, these advances will lead to more interests in the academic research on mobile RGB-D VO, even if developing algorithms that fully exploit the low level hardware capabilities of the device is challenging. For example, Brunetto *et al.* proposed a RGB-D vSLAM algorithm based on SlamDunk which can run on a Samsung Galaxy Tab Pro 10.1 tablet [7], but due to the lack of low level optimizations it could only reach 10 FPS.

Visual Odometry is however an important component to enable computer vision applications on mobile devices. Klingensmith [25] proposed a real-time mapping solution for indoor scenes that takes advantage of the depth sensor and the VIO algorithm by Google Tango. Instead of allocating a fixed grid of 3D voxels as in the traditional approaches, he creates on demand chunks of voxels according to the observations of the scene, which is appropriate for indoor environments as they contain a lot of free space. Schöps [54] addresses the outdoor mapping problem using the same hardware, which prevents the use of the depth sensor. With the help of the provided VIO algorithm he computes and filters depth maps from the fisheye camera and fuse them with a TSDF approach too in order to reconstruct the scene. Live 3D reconstruction is a very challenging problem which requires to perform the VO and the mapping in real time. Prisacariu *et al.* [40, 41] jointly estimate the pose and the visual hull of the model with a probabilistic framework. The system is robust to motion blur, lack of texture and can run at 20 FPS on an iPhone 5, but the resulting model is coarse and cannot contain concavities. Tanskanen *et al.* [58] propose a monocular visual features tracking approach combined with a multi-resolution stereo depth map estimation. The tracking runs at 15-30 FPS on a Samsung Galaxy S3 but the generated dense 3D point cloud is only refreshed at 0.3-0.5 FPS while being GPU optimized. Kolev *et al.* [26] improve the accuracy of the reconstructed model with a surfel approach combined with weighted depth maps, but at the cost of a lower frame rate. Ondrúška *et al.* [38] propose a faster solution (25 FPS on an Apple iPhone 6) which can generate medium accuracy 3D reconstructed models. They use a direct method for the tracking, compute the depth maps by dense stereo matching and perform the mapping with a TSDF approach.

3 Related works

Assessing and comparing the quality and the accuracy of VO algorithms is an important task. This work relies on previous benchmarks that have been published in the last years, mostly in the robotics community. Sturm *et al.* [57] introduced and publicly released the TUM dataset, a collection of different RGB-D image sequences meant to benchmark the SLAM and visual odometry algorithms. Even if in the paper no algorithms evaluation is carried out, it has become a seminal work as the dataset has become a sort of standard for benchmarking new algorithms in the spirit of other computer vision datasets, such as *e.g.* the KITTI dataset [14].

Morell-Gimenez *et al.* [31] performed a comparison of registration methods on scenes mapping and object reconstruction scenarios. For the scenes mapping scenario, which is our topic of interest, they evaluated five different algorithms: DVO [23], KinFu (an implementation of KinectFusion [32]), an

ICP approach, an imaged-based visual feature approach using a combination of FAST [48] keypoints and BRIEF [8] descriptors, and an hybrid two-stage approach combining the two last ones where the refinement step is provided by the ICP algorithm. The last three approaches were implemented by Morell-Gimnenez *et al*. using the Point Cloud Library [51]. The results show that DVO and Kinfu are the most accurate algorithms on the "fr1" scenes of the TUM dataset. The paper does not report any information about the computational time and the memory consumption as the main objective of the work was to assess the quality and the accuracy of each method.

Handa created the ICL-NUIM dataset [18] composed of synthetic images of indoor scenes generated with POVRay. Although the main focus of the dataset is to provide a method to benchmark the surface reconstruction accuracy, it has been used to evaluate different VO algorithms, thanks to the ground truth provided by the synthetic data. The following algorithms are compared on a desktop environment: DVO [23], Fovis [20], RGB-D [55], ICP KinectFusion flavour [32] and Kintinuous [60]. The evaluation on all scenes from ICL-NUIM with the ATE metric showed a clear advantage to KinectFusion ICP registration while Fovis gives the less accurate results.

More recently, Fang and Zhang [13] compared different open-source VO implementations: Libviso2 [15], Fovis [20], DVO [23], FastICP, Rangeflow [22], 3D-NDT [1], CCNY [11] and DEMO [63]. The evaluation is performed on two scenes of the TUM dataset and on a challenging dataset created by the authors with illumination changes, fast motion and long corridors. The metrics taken into consideration are the accuracy of the estimated camera motion and the performances of the algorithms (runtime and CPU usage). The authors [13] provide an analysis of the success and failure cases of the different algorithms w.r.t. the environment. In particular the study shows that there is no algorithm performing well in all environments and some guidelines to choose a VO algorithm depending on the environment are proposed. For example, when the scene is well illuminated, image-based and hybrid methods are recommended, whereas depth-based methods are only really interesting in low light environments.

The evaluation we are proposing in this paper is similar in spirit to the mentioned works, but our comparison is focused on the evaluation of algorithms for the mobile experiment, in which computational cost and memory consumption are strong constraints. To the best of our knowledge this is the first attempt at benchmarking state-of-the-art algorithms on mobile devices equipped with a depth sensor. Our benchmark is similar in spirit to [13], but aimed at testing VO algorithm on mobile devices. For this reason, we tested some algorithms that were not considered in [13] and, due to our needs, we considered both the CPU and memory usage of the algorithms (whereas [13] only assesses the CPU usage).

4 Tested visual odometry algorithms

For our evaluation we selected the algorithms to test based on two main criteria. Firstly, we only considered the methods that performed better in other benchmark studies (see Section 3). Secondly, the most important criteria was the availability of the code (or a SDK in the case of [37]), so that it could be ported and tested on a mobile device. According to these criteria we selected DVO [23], Fovis [20], MRSMAP [56], the 3 algorithms of the OpenCV RGB-D module [46] , and the VO algorithms that come with the Occipital sensor [37]. For brevity purpose, we denote OCV (ICP, RGB-D, RgbdICP) the three OpenCV algorithms we took into consideration. As pointed out in Section 3, only DVO and Fovis were considered for the benchmark in [13]. Table 1 provides a classification of the considered methods according to the taxonomy described in Section 2, and Table 2 collects more technical details about the code available for each method.

In the remaining part of this section we briefly review the algorithms considered for our analysis. For each algorithm we present a block diagram of the main pipeline. In the diagrams, blocks that are vertically aligned in the pipeline are blocks that can potentially run in parallel.

4.1 Fovis

Fovis [20] is a fast visual odometry library developed for micro aerial vehicles (MAV). The visual odometry (frontend) represented by the Figure 3 is performed on the MAV and the global consistency of the trajectory (backend) is enforced off-board. The registration is feature-based with a frame-to-keyframe matching strategy, employing FAST keypoints computed on multiple scales and on subdivisions of the

Algorithm	Method class	Registration	Matching Strategy	Local optimization
Fovis [20]	Image-based	Feature-based	frame-to-keyframe	No
OCV RGB-D [45]	Image-based	Direct	frame-to-frame	No
DVO [23]	Image-based	Direct	frame-to-frame	No
OCV ICP [44]	Depth-based	ICP	frame-to-frame	No
MRSMAP [56]	Depth-based	Feature-based	frame-to-frame	No
STTracker depth [35]	Depth-based	ICP	frame-to-keyframe	Unknown
STTracker color [36]	Hybrid	Unknown	frame-to-keyframe	Unknown
OCV RgbdICP [43]	Hybrid	Joint-optimization strategy	frame-to-frame	No

Table 1: Overview of the different approaches proposed in the evaluated VO algorithms.

Algorithm	Release year	License	ROS binding	SW dependencies	HW dependencies
Fovis [20]	2011	GPLv3	Yes	Eigen	(x86 SSE2)
OCV RGB-D [45]	2012	MIT	No	(Eigen)	No
DVO [23]	2013	GPLv3	Yes	Eigen, OpenCV, (PCL)	x86 SSE2
OCV ICP [44]	2012	MIT	No	(Eigen)	No
MRSMAP [56]	2012	BSD	Yes	GSL, TBB, OpenCV, Boost, PCL	No
STTracker depth [35]	2014	Closed	No	No	iPhone, iPad
STTracker color [36]	2014	Closed	No	No	iPhone, iPad
OCV RgbdICP [43]	2012	MIT	No	(Eigen)	No

Table 2: Technical overview of the evaluated VO algorithms. Dependencies in brackets are optional.

images to ensure a uniform repartition of the keypoints over the image. Each feature is assigned to a descriptor containing the pixel values of the 9×9 patch centred in the keypoint. The descriptors are matched across frames using a L1 distance. The matches are then validated using the associated 3D points: for each frame, the distances among the associated 3D points are calculated and compared with those of the other frame. This allow to retain the inlier features used to estimate the rigid body motion with Horn *et al.* method [19]. Several refinement are then applied to improve the robustness of the computed camera pose.

4.2 OpenCV RGB-D module

Maria Dimashova developed the OpenCV RGB-D module which is available in the `opencv_contrib` repository [46]. It offers a visual odometry algorithm which comes into three flavours : ICP, RGB-D and RgbdICP.

OCV RGB-D Figure 4 illustrates the RGB-D flavour [45], which is based on a direct image-based approach inspired by Steinbrucker *et al.* works [55] with a frame-to-frame matching strategy. Two hypotheses are made. First, the light intensity of a 3D point is considered to be constant among successive frames. Then, the angular and translational speed are supposed to be constant between two frames. The algorithm finds the transformation relating two frames by minimizing the difference in intensity between the warped current RGB-D frame and the previous one. The first hypothesis enables to define the objective function as the sum of the square pixel intensities between the back-projected frame and the previous one. Thanks to the second hypothesis, it is then possible to reduce the minimization problem to a linear least square problem. Finally, to ensure better robustness with large motion change, the authors apply a coarse to fine approach by working on an image pyramid.

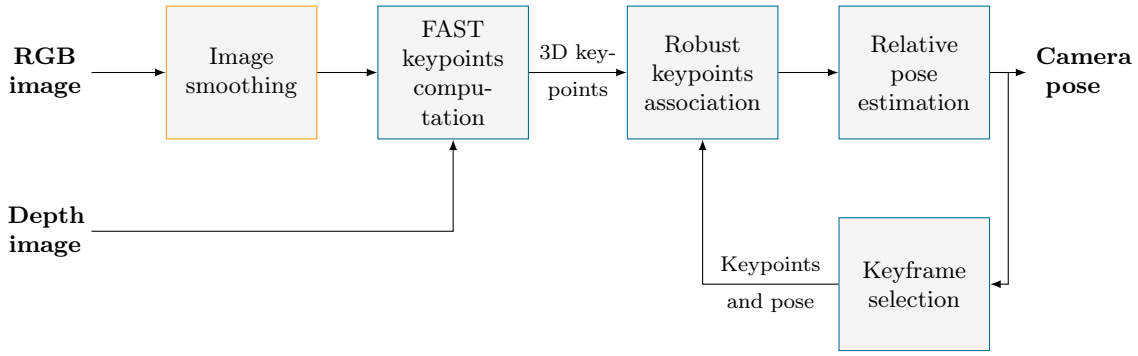

Figure 3: The pipeline of the Fovis algorithm. The preprocessing and registration steps are displayed in orange and blue respectively.

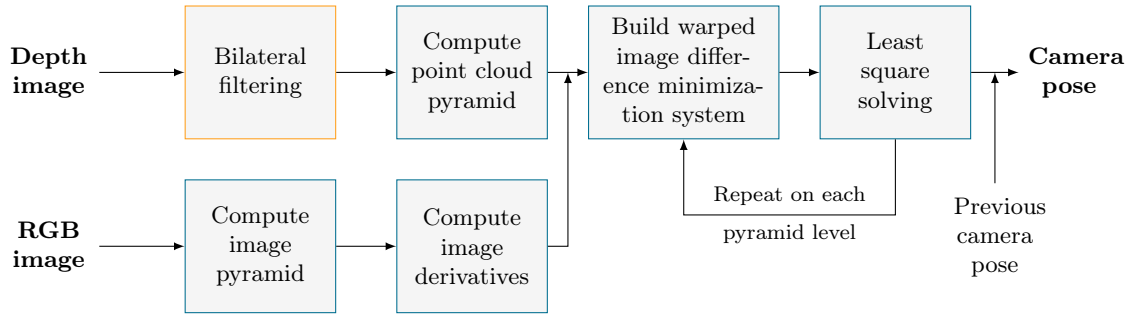

Figure 4: The pipeline of the OCV RGB-D algorithm.

OCV ICP The ICP flavour [44], shown in Figure 5, is inspired by the point cloud registration algorithm of KinectFusion [32]. KinectFusion ICP variant is based on a projection based heuristic association function with a point-to-plane error metric. Assuming a small rotation between the two frames, the minimization of the point-to-plane error is reduced to a linear least square problem. A coarse-to-fine scheme is used to speed up the point cloud registration. It requires the computation of image pyramids for the depth frames and the normal maps. A notable difference with KinectFusion point cloud registration is that OpenCV is frame-to-frame whereas the other is frame-to-model.

OCV RgbdICP We have seen previously OCV RGB-D and OCV ICP were reduced to linear least square problems. As illustrated in Figure 6, the joint-optimization hybrid approach of OCV RgbdICP takes into consideration the concatenation of the equations of the two problems and solves it. It is the same scheme Whelan proposed with his RGB-D and ICP Integration [60].

4.3 Dense Visual Odometry

Dense Visual Odometry (DVO) [23] depicted in Figure 7 is a direct image-based method with a frame-to-frame matching strategy. As in Steinbrucker *et al.* works [55] described earlier, a residual is defined with the difference of pixel intensities between the registered RGB-D frames. The minimization is performed with a coarse-to-fine approach in a probabilistic way, defining a likelihood of the transformation given the residual, and with the use of a sensor model and a motion model. The sensor model takes the form of a weighting function giving more or less importance to a given residual which partially originates from sensor noise. The motion model expresses the probability of a transformation. It can depends on IMU data if available. The proposed motion model assumes a constant velocity in order to avoid jitter in the estimated trajectory.

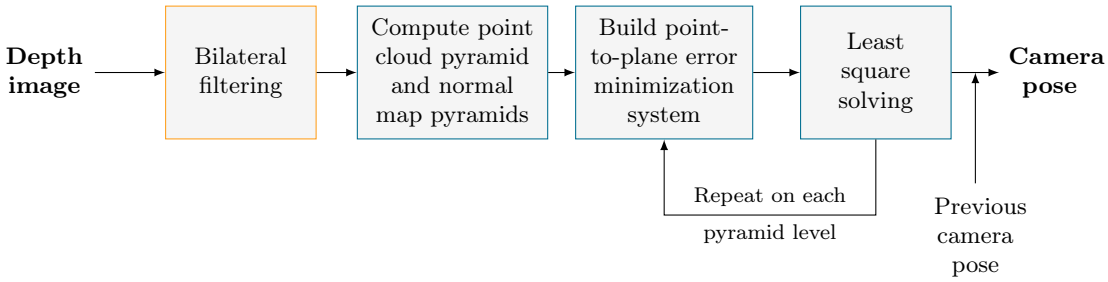

Figure 5: The pipeline of the OCV ICP algorithm.

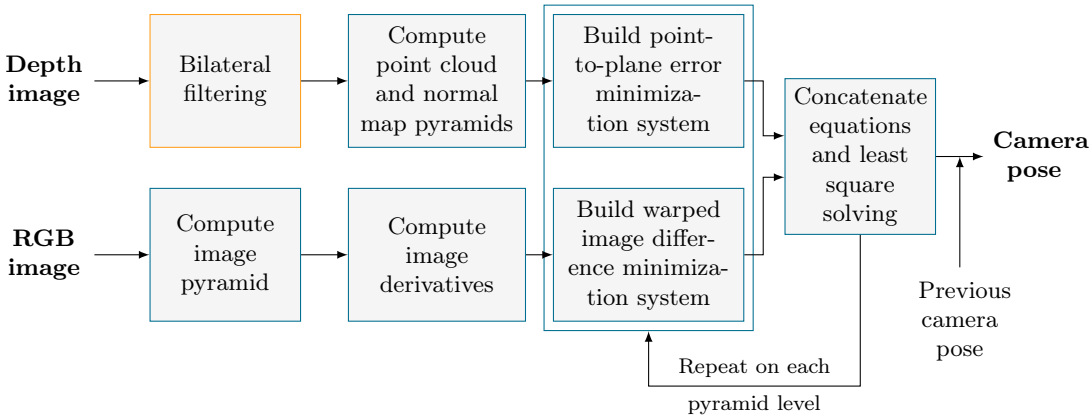

Figure 6: The pipeline of the OCV RgbdICP algorithm.

4.4 MRSMAP VO

Stückler *et al.* [56] proposed a 3D feature-based approach with a frame-to-frame matching strategy in which each frame is viewed as an octree of surfels. The originality of the approach is that multiple levels of resolution can be used simultaneously since each parent node of the octree encodes the information of their children node. The uncertainty of 3D points w.r.t. the camera is modelled by using smaller surfels for points closer to the camera. For optimization purpose, the coloured 3D points are not stored in the nodes: the local geometry and the colour distribution of the 3D points are instead encoded by a 6D multivariate normal distribution of 3D points coordinates and the three components of the colour in the L$\alpha\beta$ space. Each surfel is associated to a shape-texture descriptor which encodes the difference of colour and normal orientations between the adjacent surfels in the form of three bins histograms. The registration of a RGB-D frame illustrated in Figure 8 is performed at the level of their octree representation. The surfels of the two octrees are first associated with a coarse-to-fine approach using the shape-texture descriptor of the surfels. Then a likelihood based on the difference of the local geometry encoded by the associated surfels is maximized in order to compute the transformation between the two frames.

4.5 Occipital STTracker

Structure is a depth sensor manufactured by Occipital using Primesense's technology and it uses structured light to estimate the depth. The sensor does not support any RGB camera and it has to take advantage of the mobile device rear camera to retrieve the RGB frames. Occipital provides an iOS SDK with a VO algorithm in two flavours: depth-based [35] and hybrid [36].

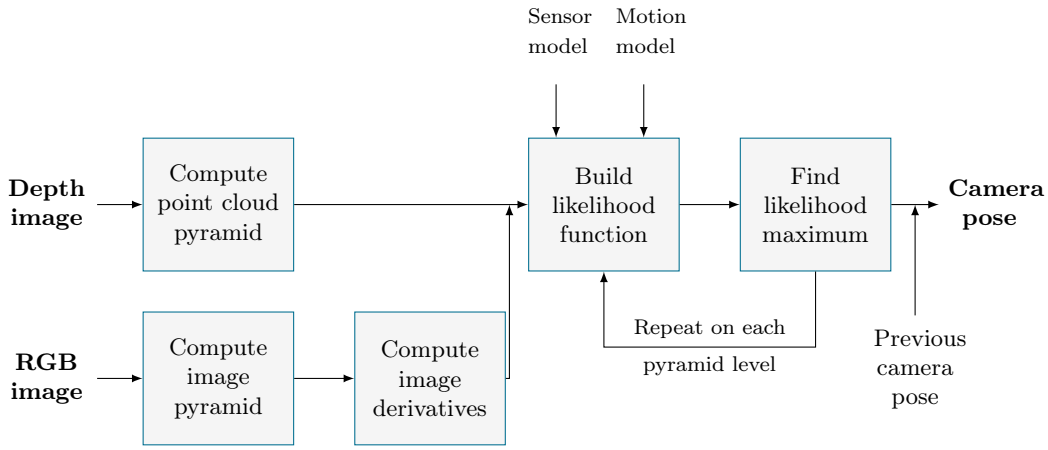

Figure 7: The pipeline of the DVO algorithm.

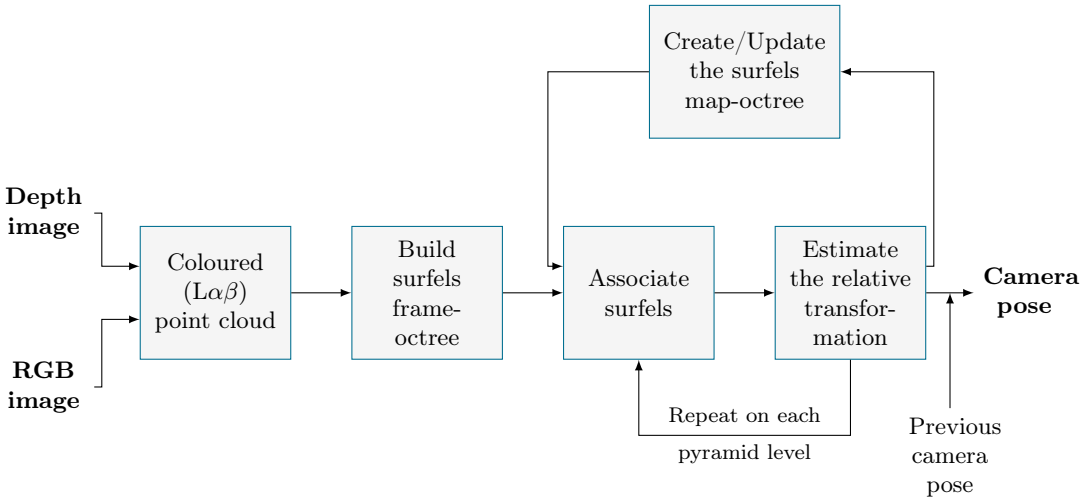

Figure 8: The pipeline of the MRSMap algorithm. The nodes association and transformation estimation steps are parallelized for each node and association respectively.

5 Algorithms selection

In order to limit the number of algorithms evaluated on the mobile devices, we performed a selection based on three criteria: the accuracy, the runtime and the memory footprint. As mentioned in Section 3, most of the algorithms that we are considering were not used in previous benchmarks [13, 31]. For these reasons, we needed an assessment of their performances in terms of accuracy and resources consumption. Since all the algorithms mentioned earlier were designed for embedded or desktop computers, we chose the latter platform which also enabled us to easily perform memory monitoring. On the other hand, accuracy evaluations are not dependent of the computing platform.

5.1 Description of the dataset

As we mentioned earlier, the RGB-D TUM dataset for SLAM evaluation was interesting for the evaluation because it offers various indoor acquisitions scenarios with ground truth trajectories. It is divided into three sets of sequences: “fr1”, “fr2” and “fr3”.

The “fr1” sequences provide various scenes recorded in an office environment. They include two simple scenes for debugging purpose: “xyz” and “rpy” with respectively translation only and rotation

only sensor movements, and two very challenging scenes : "floor" which as the name suggests has low structure, and "360" with a high rotational motion and, thus motion blur.

The "fr2" sequences were recorded in a large industrial hall. Compared to the "fr1" sequences they are generally longer and have a slower camera motion. It also contains two debugging series and a "desk" scene. Three scenes are very challenging: "360 hemisphere", "large no loop" and "large with loop", due to the low texture and the distant 3D points.

Finally, the "fr3" sequences feature a scene with a desk and various evaluation series to evaluate the performances of algorithms on scenes with structure and/or texture.

At the time of writing, the `STTracker` class which implements the VO algorithm is designed to be used with the RGB-D frames of the Structure Sensor only. The evaluation on this algorithm on the TUM RGB-D SLAM Dataset was very difficult and required us to write an intermediate software layer which supplied the Structure SDK with the required data.

5.2 Description of the metrics

Parameters All VO algorithms have parameters which must be tuned in order to give the best results. To simplify the experiments, we took the parameters recommended by the author's algorithms in their respective articles. For the Structure STTracker, we took the parameters `STTrackerQualityAccurate` and `STTrackerDepthAndColorBased`.

Accuracy evaluation There are two well know metrics that can be used to estimate the accuracy of the estimated camera poses over time, the Absolute Translational Error (ATE) and the translational Relative Pose Error (RPE) [57]. They both assume that the ground truth and the estimated trajectory are aligned, time-synchronized and equally sampled. At a given time step ATE computes the euclidean distance between the estimated camera position and its ground truth. The ATE is then defined as the mean squared error (RMSE) of these distances all along the trajectory. This metric is more suitable for vSLAM evaluation because it assesses the global consistency of the estimated trajectory relatively to the ground truth.

The RPE is instead used to measure the local accuracy of the estimated trajectory over a fixed time interval Δ. Considering a sequence of estimated camera poses (rototranslations) $\mathtt{P}_i \in SE(3), i = 1, \ldots, n$ and their corresponding ground truth $\mathtt{Q}_i \in SE(3), i = 1, \ldots, n$, the relative pose error $\mathtt{E}_i$ at time i is defined as

$$\mathtt{E}_i = \left(\mathtt{Q}_i^{-1}\,\mathtt{Q}_{i+\Delta}\right)\left(\mathtt{P}_i^{-1}\,\mathtt{P}_{i+\Delta}\right)^{-1}$$

The overall RPE of the sequence is then defined as the RMSE of the translation of each $\mathtt{E}_i$. The RPE better represents the drift of the trajectory over time, which is useful for the evaluation of visual odometry systems.

The accuracy comparison of the algorithms was performed during the desktop experiment. We used the RPE metric with a time interval $\Delta = 1\,\mathrm{s}$. For each experiment, we computed and plotted the RMSE, the mean and the standard deviation of the RPE values. We also plotted the graphs of the RPE over the time for visual inspection purpose, in order to highlight the experiments with high and narrow error peaks which would be masked by the RMSE measure.

As shown in Section 2, many parameters influence VO algorithms performances. For a proper comparison, we should compare individually, for each VO algorithm, its registration performance, using the same rigid body transformation estimation function, the same pre-processing and post-processing steps. This would be unpractical, for this reason we only compared the full pipeline of the algorithms, as an end-user would use it.

Performance evaluation Memory consumption evaluation can be quite controversial. It can be heavily impacted by the optimizations performed by the kernel and the presence of a garbage collector. For this reason, we provided values intended to give a general idea of the memory consumption of the evaluated algorithms.

We used a computer with an Intel ®Core™i7-2600 CPU and 6 GB of RAM for the desktop experiment. We monitored the performances of the VO algorithms by recording every second the process information status given on GNU Linux operating systems by the files `/proc/pid/stat` and `/proc/pid/statm`. To evaluate the memory consumption, we took into consideration the maximum

value of the Resident Set Size, also called virtual memory high water mark (VmHWM), and the maximum value of the program data (Pgm Data).

The Resident Set Size is the actual part of the virtual memory used by the process which is mapped into the RAM. Therefore it is a good indicator of the RAM requirements of the target platform. The program data is the sum of the stack size, the heap size, and the size of the global plus static variables (data+bss), in other words it is the sum of VmData and VmStck. It is mainly affected by the heap size and may be partially mapped into the RAM.

We did not take into account the Virtual Memory Size, which is the total amount of virtual memory used by a program. It includes the size of the binary and its linked shared libraries, the stack and heap usage. Unused shared libraries can dramatically increase the Virtual Memory Size, leading to misinterpretations.

To monitor the runtime performances, we took into consideration the number of processed frames per seconds and the CPU usage[1]. Due to the Quad-Core CPU of our desktop computer, our CPU load value are between 0 and 400 %.

In order to ensure the runtime performances were not affected by intensive I/O operations, we also monitored the total I/O delays provided by `delayacct_blkio_ticks`. Since the I/O delays of all the monitored algorithms was negligible compared to execution time, we did not include their values in our results.

5.3 Accuracy results

Figure 9, Figure 10 and Figure 11 represent the bar graphs of the RPE of the evaluated algorithms on the different scenes of the RGB-D TUM dataset for SLAM. In order to ease the comparison, the different classes of VO algorithms are clustered with different hues: shades of red, green, and blue for the image-based, depth-based, and hybrid algorithms.

A first simple observation of the different graphs is that the accuracy results significantly vary from a scene to another. As stated by Fang [13], there is no algorithm which outperforms the others in all environments. The results have to be analysed w.r.t. the scene characteristics. Therefore the choice of VO algorithm depends on the target environment. Apart from the challenging scenes we described earlier and correspond to higher RPE values, the slower "fr2" scenes obtain better results than the "fr1" scenes. This illustrates well the importance of speed on the VO performances.

As the intuition suggests, the hybrid and image-based methods are the most accurate when the environment has texture and no structure such as the scenes "fr1 floor", "fr3 nostructure texture near withloop" and "fr3 nostructure texture far". Similarly, the environments with structure and low texture favour the hybrid and depth-based algorithms as shown by the scene "fr3 structure notexture near". Nevertheless, with the scene "fr3 structure notexture far", which has noisier depth data, the accuracy of the ICP algorithm is comparable to the image-based algorithms. On this scene, the 3D feature-based approach of MRSMAP enables to achieve the lowest RPE. When the environment is neither flat nor textureless, *e.g.* the "fr3 structure notexture near" scene, we reproduced Fang [13] results, in which image-based or hybrid-based methods are more robust than depth-based methods. However, surprisingly the addition of texture on the "fr3 structure notexture near" scene deteriorated the results of the depth-based methods. It must be noted that the results reported in [31] for the "fr3" scenes show that the image-based methods have a higher RPE than the depth-based methods on the textured scenes with low structure and vice-versa for the untextured scenes with low structure. After a comparison with our DVO results, we found out that this discrepancy of results was due to the inversion of the plot labels and to an incorrect scaling on the y-axis ticks in the paper of [31]. Also the accuracy difference between the depth-based and image-based methods is very important, which might be explained by the lack of structure in the scene. In contrast, the scenes recorded in the office also show the hybrid and image-based methods are more robust, but the accuracy difference with the depth-based methods is slighter. A trend emerge if we compare the most accurate algorithm on each scene of the "fr1" and "fr2" series: OCV RgbdICP and Fovis have the lowest RPE on the scenes "fr1" and "fr2" respectively.

[1] We use the UNIX definition for CPU usage as $\sum_{c \in NumCores} \frac{time\_spent\_on\_core_c}{elapsed\_time}$, where $elapsed\_time$ is the delta of system clock between the start and the end of the execution, and $time\_spent\_on\_core_c$ are the number of system clocks spent on each of the $NumCores$ of the machine [17].

DVO and OCV RGB-D generally come behind or between. There are some exceptions to this trend, but not enough to draw conclusions on them.

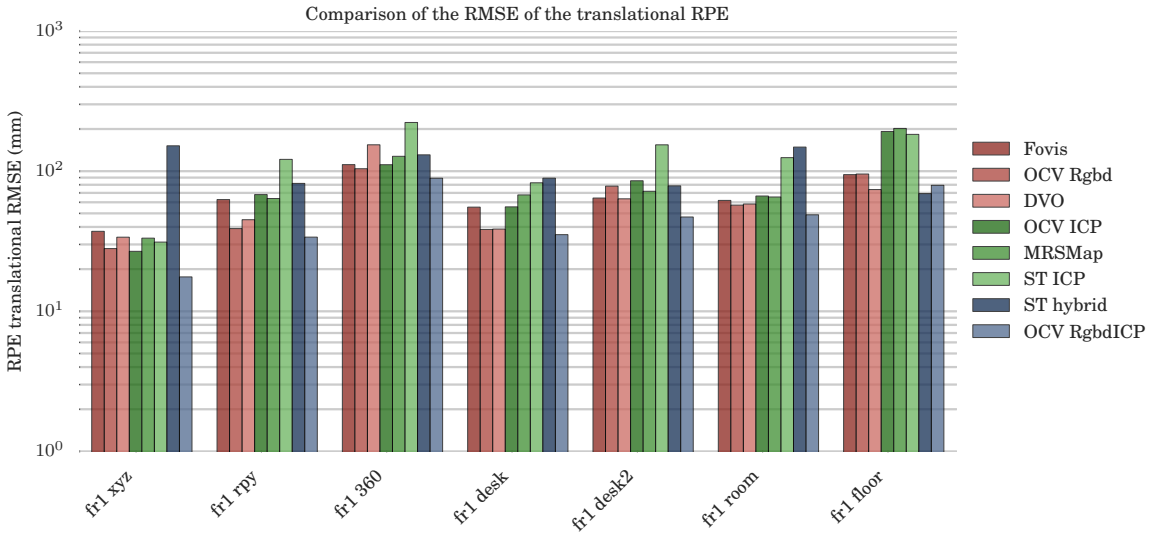

Figure 9: RPE comparison on the fr1 sequences of the TUM dataset.

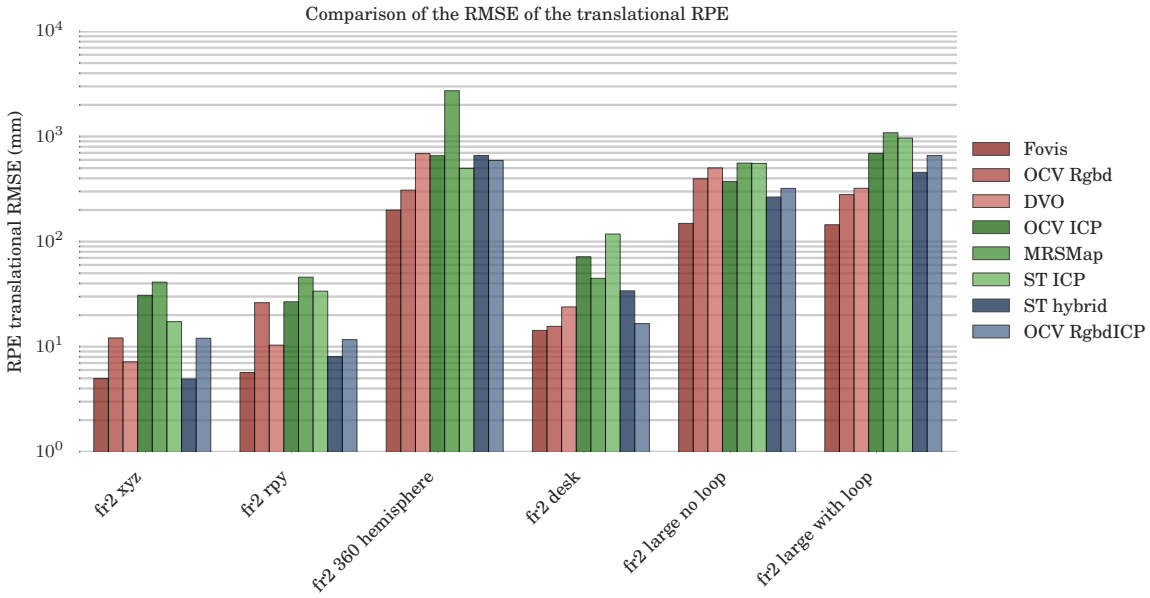

Figure 10: RPE comparison on the fr2 sequences of the TUM dataset.

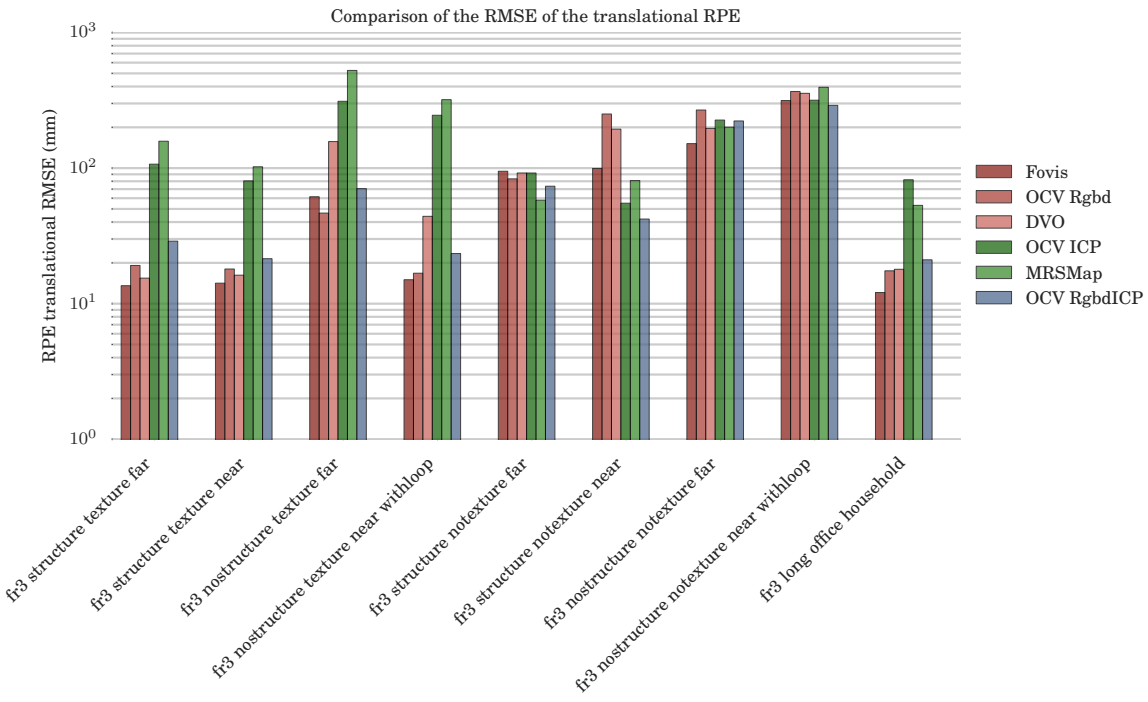

Figure 11: RPE comparison on the fr3 sequences of the TUM dataset.

5.4 Performance results

Runtime performances Table 3 illustrates the runtime performances performed with VGA frames on the TUM fr1 desk scene. It shows that only Fovis can run at the rate of the depth sensor which is 30 FPS. All the imaged-based algorithms, DVO, Fovis and OCV RGB-D, can run at a frame rate superior to 20 FPS, which is fast enough for real-time applications such as augmented reality.

The CPU load column from the Table 3 illustrates that all the algorithms do not fully take advantage of the multiple cores of the CPU. Surprisingly, the fastest algorithm, Fovis only uses one thread, while the slowest, MRSMAP use several ones. Also the hybrid method OCV RgbdICP does not take well advantage of threads, while its image-based and its depth-based approaches could be run in parallel.

Memory consumption The memory performance evaluation illustrated by Table 3 reveals that several algorithms require more than 500 MB of program data. In contrast the peak value of the Resident Set Size (VmHWM) is generally below 100 MB which is low enough for mobile devices. Again, Fovis is the least demanding algorithm with only 25 MB of maximal memory mapped into the RAM while MRSMap, the most demanding uses 300 MB. This comparison between the program data and the VmHWM also demonstrates that generally only a small amount of the program data is mapped into the RAM.

5.5 Experiments conclusion

From this evaluation, we selected the algorithms to evaluate on the iPad. Concerning the depth-based methods, we selected OCV ICP over MRSMAP for being slightly more accurate, less CPU-demanding and easier to compile on the iPad. We selected OCV RgbdICP which was our only hybrid algorithm, while for the image-based methods, we kept Fovis for its best runtime performance and high accuracy while we dropped DVO as it has many x86 optimizations and a similar accuracy to OCV RGB-D.

Name	FPS	CPU load (%)	VmHWM (MB)	Pgm Data (MB)
MRSMap	9.3	200	332	811
OCV ICP	17.8	101	70	622
OCV RgbdICP	11.9	101	74	626
OCV Rgbd	22.7	94	50	602
DVO	23.8	288	89	536
Fovis	103.9	91	25	24

Table 3: Performance evaluation on the "TUM fr1 desk" scene with VGA frames performed on a desktop computer.

6 Mobile experiments

6.1 Second accuracy experiment: Structure Sensor acquisitions

6.1.1 Description of the dataset and the metrics

The evaluated algorithms had their parameters optimized to give the best results on some series of the TUM dataset which used a Kinect^{SL}as depth sensor. While the two sensors share the same core technology, we wanted to check with a second experiment whether we could observe a different accuracy trend with the Structure Sensor. We connected the Structure Sensor on the iPad Air used in the previous experiment and we used a dedicated application to record locally the trajectory estimated from the STTracker VO algorithm, the RGB-D frames and the IMU data.

We recorded three scenes in three different rooms (r1, r2, r3) with various luminosity levels, denoted *hl*, *ml* and *ll* for high, medium and low luminosity respectively. For each scene, We also recorded different camera motions, which we denoted *hs*, *ms* and *ls* for high, medium and low camera speed respectively.

In the absence of motion capture cameras that could provide a ground truth for the device motion, we constrained the camera motion on an horizontal plane (*e.g.* like a ground floor or the surface of a table), using a home-made mount to secure the device in a vertical position. Therefore, instead of evaluating the drift w.r.t. a known pose, we will rather evaluate the planarity of the device trajectory. We also ensured the trajectories had their start and stop positions identical by putting the mount in contact with the same reference object at the beginning and the end of the recording. This guarantees almost perfect loop closure, any error made is negligible in comparison to the expected drift, as measured from the desktop experiment.

As for the metric to evaluate the accuracy of the algorithms, we use both the loop closing error and the RPE. The loop closing error is defined as the distance between the endpoints of the estimated trajectory divided by the path length, and it is used by most of the authors [13, 64] to compare VO algorithms. This metric evaluates the performances of the algorithms globally rather than locally, for each time step. On one hand, algorithms with a low RPE may be penalized by this metric because of one major drift; on the other hand, algorithms with high RPE may have a smaller distance because of compensations between the various drifts, somewhat like in a perfect random walk. Therefore RPE can give a better insight about the performance of the algorithm all over the device motion.

In our case, given that the trajectory is supposed to be planar and we cannot have a ground truth for the device pose, we instead evaluate the RPE as the drift along the z-axis component (*i.e.* the normal to the plane of the motion) to assess the quality of the estimated trajectory

6.1.2 Results and analysis

The Figure 12 represents the RPE along the z-axis for the evaluated algorithms. We compared our metric with the loop closing error. In contrast with the experiments on the TUM datasets, the OCV RGB-D and OCV ICP algorithms do not perform well. With the Structure Sensor dataset the STTracker hybrid algorithms has generally the highest accuracy, matched only by OCV RgbdICP. These results show that the trends are very different on some algorithms for the Kinect^{SL} RGB-D TUM dataset and

our Structure Sensor dataset.

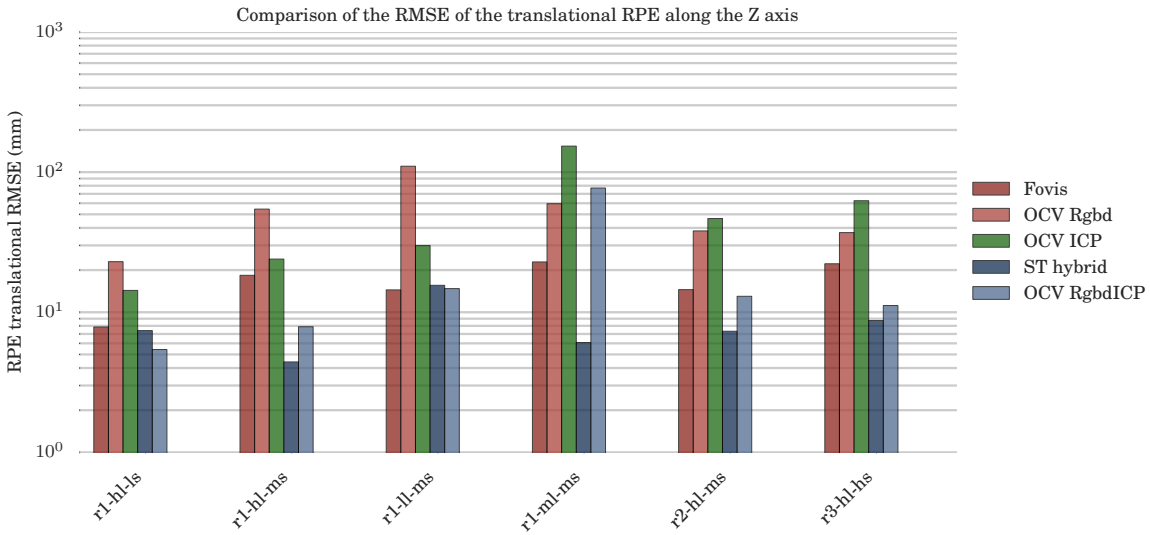

Figure 12: Comparison of the RMSE of the translational RPE along the z-axis.

Figure 13: Loop closing error defined as the distance between the endpoints of the estimated trajectory divided by the path length.

Figure 13 represents the same evaluation with the start-end distance metric. The trends between the two figures seem very similar. However, on "r1-hl-ms", "r1-ll-ms", "r2-hl-ms" and "r2-hl-ms" the ranking of the Fovis, OCV RGB-D and OCV ICP algorithms is very different with the two metrics.

6.2 Performance evaluation

The performance experiment was carried out on four mobile devices: two iOS devices, iPhone 5 and iPad Air and two Android devices, Memo Pad 7 and Tango Yellowstone. As mentioned in the Section 2, the Tango Yellowstone has a dedicated VIO module requiring different input: thus, it cannot be fairly compared with the RGB-D VO algorithms, which are the object of this article. Table 4 displays

the characteristics of these mobile devices. The PassMark CPU benchmark assesses trough various intensive parallel computational algorithms how fast a CPU is. The scores gives an indication of the CPU speed, the faster is the processor, the higher is the score. We took the value of the Android[2] and iOS[3] ranking available on November 17th, 2016. The scores indicate the iPhone 5 and the Memo Pad 7 can be considered as middle performance devices, whereas the iPad Air and the Tango Yellowstone can be considered as high performances devices.

We compiled the previously selected VO algorithms with the `-O3` optimization option and used the "TUM fr1 desk" scene to evaluate their performance, with two different image resolutions: VGA (640×480) and QVGA (320×240).

Manufacturer	Model	CPU	CPU PassMark score	RAM (GB)
Apple	iPad Air	Apple A7	37517	1.0
Apple	iPhone 5	Apple A6	23914	1.0
Asus	Memo Pad 7 K013	Intel ® Atom ™Z3745	27807	0.86
Google	Tango Yellowstone	Nvidia Tegra K1	38503	3.7

Table 4: The mobile devices used for the performance evaluation with some of their hardware specifications and their PassMark score (the faster the CPU the higher the score).

Algorithm	Device	QVGA (FPS)	VGA (FPS)
Fovis	iPad Air	92.0	24.1
	iPhone 5	38.7	10.2
	Memo Pad 7	81.6	20.1
	Tango Yellowstone	96.0	26.0
OCV RGB-D	iPad Air	28.6	6.7
	iPhone 5	13.7	3.6
	Memo Pad 7	8.9	2.4
	Tango Yellowstone	17.2	4.3
OCV ICP	iPad Air	23.8	5.5
	iPhone 5	9.7	2.5
	Memo Pad 7	7.9	2.1
	Tango Yellowstone	14.4	3.6
ST ICP	iPad Air	43.7	42.6
	iPhone 5	23.3	20.9
ST hybrid	iPad Air	36.4	28.3
	iPhone 5	19.2	16.7
OCV RgbdICP	iPad Air	14.3	3.2
	iPhone 5	6.4	1.6
	Memo Pad 7	4.3	1.2
	Tango Yellowstone	8.1	2.0

Table 5: Performance evaluation on the "TUM fr1 desk" scene with QVGA (320×240) and VGA (640×480) images performed on the four mobile devices.

[2] http://www.androidbenchmark.net/cpumark_chart.html
[3] http://www.iphonebenchmark.net/cpumark_chart.html

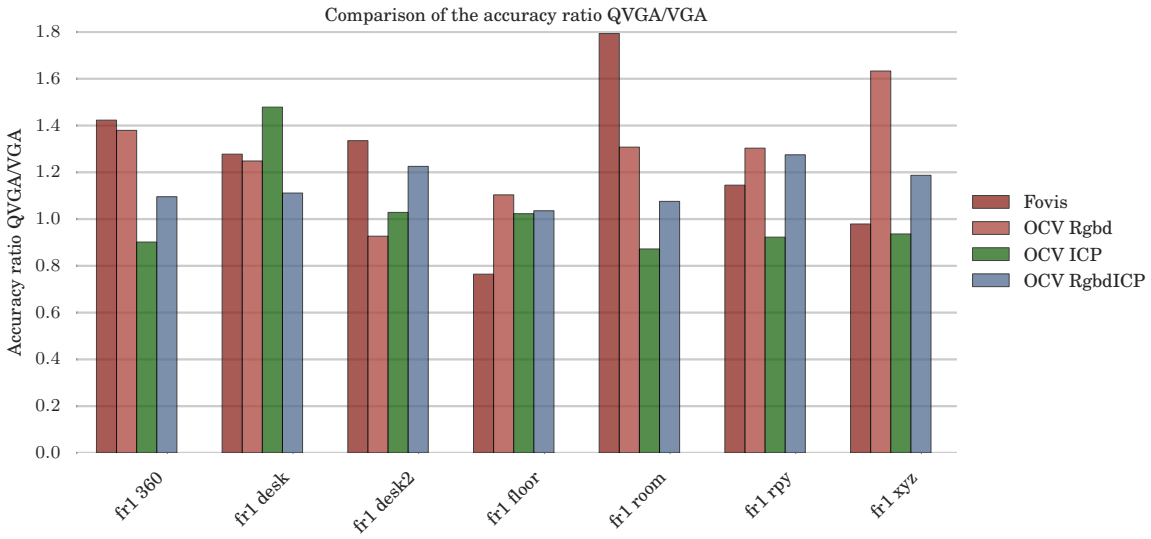

Figure 14: Comparison of the RPE ratio QVGA/VGA on the fr1 scenes of the TUM dataset.

Table 5 shows the results of the performance experiment. For each algorithm the frame rate (fps) is reported for each device for which it was possible to port the algorithm. It must be noted that we chose to report the frame rate as a performance measure as it can be used as a reference for assessing the suitability of the algorithm for a given application. In general, computer vision applications are usually considered to be real-time when they are able to assure a minimal throughput in terms of images processed per second. This clearly depends on the target application and the type of time constraints that must be guaranteed [4]. Visual odometry can be used to enable applications like augmented reality or more general robotic applications, and in general, for this range of applications a frame rate of 15 fps is commonly considered a minimal threshold to assure the responsiveness of the application.

As it can be noted from the table, downsampling the images from a VGA resolution to QVGA clearly improves the computational performances up to a factor of 4. However, this has sometimes an impact on the estimated trajectory and the accuracy of the results. Figure 14 shows the ratio between the VGA and the QVGA accuracy: using half resolution generally worsens the achievable throughput (ratio greater than 1), with some exceptions in which we observed a slight improvement of accuracy (ratio lesser than 1).

In the case of the STTracker, it can be noted that resolution does not affect the performances, as there is only a slight difference between the two resolutions. Since the code is not available and the documentation is not clear on this point, we can only speculate that the algorithm always downsample input VGA images to ensure a high frame rate.

More generally, Fovis is the only algorithm that can achieve high frame rates at VGA resolution on high-end devices, and it anyway outperforms the other algorithms on iPhone 5. It is worth noting that Fovis' SSE2 optimizations were disabled when running the tests on the Memo Pad 7 as they leaded to a slight loss of performance. All the other algorithms fail to reach real-time performances at VGA resolution, even on high end devices: OCV RGB-D and OCV ICP are the only ones passing 5 fps on iPad Air. When using QVGA resolution, the performances of all algorithms improves and generally iPad Air is the device getting higher frame rates for every considered algorithm. As for the OpenCV family of algorithms, OCV RGB-D only fails to achieve real-time performances on Memo Pad 7, OCV ICP can provide high frame rates only on the most powerful devices, iPad Air and Tango Yellowstone, while OCV RgbdICP only comes close to the threshold of 15 fps on iPad Air.

Concerning the devices, the iPad Air is twice as faster as the iPhone 5, and the Memo Pad 7 is twice as slower as the Tango Yellowstone, with the exception of the Fovis algorithm.

7 Discussion

Even if modern mobile devices can sport CPUs with 2 or 4 cores up to 2.3 GHz, their computational power cannot be exploited at their full potential for long period of time without draining the battery and risking some over-heating of the device. Since they are designed to be power efficient, their frequency is often throttled down and their instructions set is reduced. Moreover the current hardware architectures of mobile devices have reduced L1 and L2 cache and a reduced instruction set. Therefore, when optimizing the implementation, developers should pay particular attention to the memory accesses, for example taking advantage of the pre-fetching and maximizing the processing on small blocks of data.

As a general rule, polymorphism should be limited or used carefully, as it may introduce performance overheads and it may lead to indirect function calls which are less likely to be optimized at compile time [61]. For example, in a study over a large set of programs Driesen *et al.* [10] showed that, in average, 5.3 % of the time is used to deal with polymorphism, and 13.7 % for "all virtual" versions of the program. For the examined algorithms, we can note that OpenCV highly uses polymorphism both for data structures and algorithms, while at the other end, Fovis uses polymorphism only for the abstraction of the input data source.

Standard computer vision libraries such as OpenCV and ROS [42] are extremely useful tools for developing, prototyping and testing algorithms. On the other hand, these libraries were originally designed mostly for desktop environments, and only recently the porting to mobile environment has been started. Despite these efforts, at the moment of writing, they still lack of adequate and complete code optimization, supporting *e.g.* specific instruction set like ARM-NEON that could fully exploit the specific hardware of modern mobile devices. Moreover, the use of `float` over `double` data type is recommended for runtime and memory performance: even if the most recent ARM processors are 64 bit CPUs, for the time being there are no instructions that support double precision operations [2], which introduces type conversion overheads that significantly affects the performances [28].

Parallelization can also improve the performances of the algorithm when ported in the mobile environment. Computationally intensive algorithms for image processing can be parallelized by means of shaders that can speed-up the computation by running on the GPU of the device. For example, all the pre-processing blocks of Figures 3–6 used to pre-process the RGB-D image using classic image processing algorithm (bilateral filtering, image smoothing, *etc.*) can be implemented as a shader. Multithreading is also another natural way to optimize the pipeline execution. As showed in Section 4 when describing the algorithm pipelines, all the blocks that are vertically aligned can be actually run in parallel by different threads. This is especially adapted for the algorithms that require to process both the depth and the RGB image, such as OCV ICP, OCV RGB-D and DVO. Splitting the processing of each input image into different threads will certainly benefit the performance of the algorithm, and optimize the resource consumption. More sophisticated parallelization can also be employed combining pipelining and look-ahead strategies [47], so that the device resources can be fully exploited. For example, a pipelined version of OCV RGB-D (*c.f.* Figure 4) could reserve two threads for processing the two input images as they are available instead of waiting for the whole pose estimation process to end. This allows to improve the throughput of the algorithm at the cost of more complexity of the implementation.

An IMU offers a good combination with VO algorithms because of its complementary with visual sensors. Inertial data is computationally cheap, it deals well with rapid movements, but it suffers from drift and measurement noise. On the other hand, visual data can provide more precise and stable measurements, which can be used as reference to prevent the inertial measurements from drifting. Inertial data are particularly adequate for algorithms based on iterative solvers that need a first initial solution, and algorithms where a rough estimate of the motion is used. For the Fovis algorithm for example, the "features matching" step takes advantage of the knowledge of the rotation between the frames: the method uses pixel errors between images to infer the rotation, whereas as stated by the authors, the IMU data could provide the same kind of information at a lower computational cost. The DVO algorithm is designed to be used with a motion prior, again an inertial sensor can fulfil this task. Brunetto *et al.* [7] demonstrated that higher robustness of the pose estimation of their features image-based vSLAM algorithm SlamDunk can be achieved with the use of IMU data from a Samsung tablet.

Considering the scenario of a mobile augmented reality application where 24 fps or higher is rec-

ommended, Fovis with QVGA images appears to be the best choice since it can runs at high frame rates on middle and high performance mobile devices. In the case iOS, only the devices with high-end specifications such as the iPad Air can achieve real-time performances, with the exception of the STTracker algorithms, which are specifically optimized for this environment.

8 Conclusions

In this paper we presented a classification and a theoretical review of RGB-D VO algorithms. We tested and analysed the performances on a mobile device of 6 visual odometry algorithms designed for RGB-D sensors on different mobile devices covering the two most common mobile operating systems, iOS and Android. We selected the most promising algorithms to test based on previous benchmarks found in the literature, and our tests on desktop environment to assess the computational and memory performances before porting them to the mobile environment. The performances of each algorithm were analysed in terms of accuracy and time and memory consumption, which is a fundamental aspect when deploying a visual odometry algorithm on a device with limited resources. We assessed and confirmed the algorithms accuracy on the state-of-the-art RGB-D TUM dataset and we collected the time and memory consumption of the algorithms to get a first rough estimation of the resources needed.

After selecting the most promising algorithms in terms of accuracy and resources consumption, we run several tests on mobile devices to assess both the actual performances on the mobile devices and the accuracy on our own dataset.

In general, results showed that only high-end devices such as iPad Air can guarantee some adequate frame rate at normal resolution (VGA). Reducing the resolution of the input image proved to increase the throughput, yet sometimes at the expense of the accuracy. On the other hand, algorithms provided with the Structure SDK, the ST hybrid [36], can achieve a good accuracy and faster execution times even at full resolution. Since the code for the latter is closed-source, we can only expect that the code is specifically designed and optimized for the mobile settings. As for the open-source algorithms, only Fovis could achieve frame rates up to 24 fps and perform adequately on all the four available mobile devices. This could be explained by the fact the algorithm was already designed for running on micro aerial vehicles, thus privileging a simpler implementation adapted to limited resources environment. Results also shown that implementation relying on standard computer vision libraries such as OpenCV, are still lacking a proper support for mobile architectures. These tools are quite useful for quickly prototyping the implementation of an algorithm, but, for the time being, they are still oriented to the desktop environment and their complexity is not well suited for mobile environments.

In the light of the evidences showed in this paper, it appears clearly that designing a VO algorithm for mobile environments requires to thoughtfully adapt the implementation to the limited resources available to achieve a good trade-off between accuracy and throughput. The VO algorithms provided by the hardware makers such as Occipital are the best bases upon which building an application, as they are finely tuned for the specific environment. On the other hand, developing an original VO algorithm requires a thorough design of the algorithm from the ground-up.

Acknowledgements

This research was supported by the CIFRE grant ANRT contract #2014/0014 funding the collaboration between the French company Telequid and INPT-IRIT laboratory, and in part by the Croatian Science Foundation's funding of the project IP-11-2013-3717. The authors also acknowledge the French-Croatian Program "Cogito", Hubert Curien partnership, funding the project "3D reconstruction using smartphone" (project code 33059RF).

References

[1] H. Andreasson and T. Stoyanov. Real time registration of RGB-D data using local visual features and 3D-NDT registration. In *Proceedings of International Conference on Robotics and Automation (ICRA) Workshop on Semantic Perception, Mapping and Exploration (SPME)*. IEEE Computer Society, 2012. 6

[2] ARM. Arm compiler toolchain assembler reference v5.0 - neon instructions. http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0489c/CJAJIIGG.html, 2016. 19

[3] H. Bay, T. Tuytelaars, and L. Van Gool. SURF: Speeded up robust features. In A. Leonardis, H. Bischof, and A. Pinz, editors, *Proceedings of the 2006 European Conference on Computer Vision (ECCV 2006)*, volume 3951 of *Lecture Notes in Computer Science*, pages 404–417, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. 3

[4] M. Ben. *Principles of Concurrent and Distributed Programming, Second Edition*. Addison-Wesley, second edition, 2006. 18

[5] P. J. Besl and H. D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, Feb 1992. 4

[6] J. Borenstein, H. R. Everett, and L. Feng. *Navigating Mobile Robots: Systems and Techniques*. A. K. Peters, Ltd., Natick, MA, USA, 1996. 3

[7] N. Brunetto, S. Salti, N. Fioraio, T. Cavallari, and L. Di Stefano. Fusion of Inertial and Visual Measurements for RGB-D SLAM on Mobile Devices. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshops*. IEEE Computer Society, December 2015. 5, 19

[8] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. BRIEF: Binary robust independent elementary features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, volume 6314 LNCS, pages 778–792. Springer Verlag, 2010. 3, 6

[9] Y. Chen and G. Medioni. Object modeling by registration of multiple range images. In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2724–2729 vol.3, Apr 1991. 4

[10] K. Driesen and U. Hölzle. The direct cost of virtual function calls in c++. *SIGPLAN Not.*, 31(10):306–323, Oct. 1996. 19

[11] I. Dryanovski, R. G. Valenti, and J. Xiao. Fast visual odometry and mapping from RGB-D data. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2305–2310, 2013. 3, 4, 6

[12] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard. An evaluation of the RGB-D SLAM system. In *Proceedings of the 2012 IEEE International Conference Robotics and Automation (ICRA)*, volume 1, pages 1691–1696. IEEE Computer Society, 2012. 3

[13] Z. Fang and Y. Zhang. Experimental Evaluation of RGB-D Visual Odometry Methods. *International Journal of Advanced Robotic Systems*, 12(3):1–16, 2015. 2, 3, 6, 10, 12, 15

[14] A. Geiger. Are We Ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2012)*, CVPR '12, pages 3354–3361, Washington, DC, USA, 2012. IEEE Computer Society. 5

[15] A. Geiger, J. Ziegler, and C. Stiller. StereoScan: Dense 3D Reconstruction in Real-time. In *Proceedings of the Intelligent Vehicles Symposium (IV 2011)*. IEEE Computer Society, 2011. 6

[16] Google. ATAP Project Tango – Google. http://www.google.com/atap/projecttango/, 2014. 1, 5

[17] N. J. Gunther. White paper: UNIX Load Average – Part 1: How It Works. White paper http://www.teamquest.com/pdfs/whitepaper/ldavg1.pdf, TeamQuest Corporation, 2010. 12

[18] A. Handa, T. Whelan, J. Mcdonald, and A. J. Davison. A Benchmark for RGB-D Visual Odometry, 3D Reconstruction and SLAM. In *Proceedings of the 2014 IEEE International Conference on Robotics and Automation (ICRA 2014)*, pages 1524–1531. IEEE Computer Society, 2014. 6

[19] B. K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4):629–642, 1987. 7

[20] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy. Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera. In *Proceedings of the International Symposium of Robotics Research (ISRR 2011)*, pages 1–16, 2011. 3, 6, 7

[21] M. A. Hudelist, C. Cobârzan, and K. Schoeffmann. OpenCV Performance Measurements on Mobile Devices. In *Proceedings of International Conference on Multimedia Retrieval - ICMR '14*, pages 479–482, New York, New York, USA, 2014. ACM Press. 3

[22] G. Jones. Accurate and Computationally-inexpensive Recovery of Ego-Motion using Optical Flow and Range Flow with Extended Temporal Support. In *Procedings of the British Machine Vision Conference 2013*, pages 75.1–75.11. British Machine Vision Association, 2013. 6

[23] C. Kerl, J. Sturm, and D. Cremers. Robust odometry estimation for RGB-D cameras. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2013)*, pages 3748–3754. IEEE Computer Society, 2013. 3, 4, 5, 6, 7, 8

[24] K. Khoshelham and S. O. Elberink. Accuracy and resolution of Kinect depth data for indoor mapping applications. *Sensors (Basel, Switzerland)*, 12(2):1437–54, Jan. 2012. 1

[25] M. Klingensmith, I. Dryanovski, S. Srinivasa, and J. Xiao. Chisel: Real time large scale 3d reconstruction onboard a mobile device. In *Proceedings of the 2015 Robotics Science and Systems Conference*, July 2015. 5

[26] K. Kolev, P. Tanskanen, P. Speciale, and M. Pollefeys. Turning mobile phones into 3d scanners. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3946–3953. IEEE Computer Society, June 2014. 5

[27] S. Leutenegger, M. Chli, and R. Y. Siegwart. BRISK: Binary Robust invariant scalable keypoints. In *Proceedings of the 2011 IEEE International Conference on Computer Vision (ICCV2011)*, pages 2548–2555. IEEE Computer Society, nov 2011. 3

[28] N. Limare. Integer and floating-point arithmetic speed vs precision. http://nicolas.limare.net/pro/notes/2014/12/12_arit_speed/#index3h2, 2014. 19

[29] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004. 3

[30] Mantis Vision. MV4D depth sensor for mobile devices. http://www.mv4d.com/mobile.php, 2014. 5

[31] V. Morell-Gimenez, M. Saval-Calvo, J. Azorin-Lopez, J. Garcia-Rodriguez, M. Cazorla, S. Orts-Escolano, and A. Fuster-Guillo. A comparative study of registration methods for RGB-D video of static scenes. *Sensors (Switzerland)*, 14(5):8547–8576, 2014. 2, 3, 5, 10, 12

[32] R. A. Newcombe, D. Molyneaux, D. Kim, A. J. Davison, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion: Real-Time Dense Surface Mapping and Tracking. In *Proceedings of the IEEE International Symposium on Mixed and Augmented Reality (ISMAR 2011)*, pages 127–136. IEEE Computer Society, 2011. 2, 3, 4, 5, 6, 8

[33] C. V. Nguyen, S. Izadi, and D. Lovell. Modeling kinect sensor noise for improved 3d reconstruction and tracking. In *Proceedings of the 2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization & Transmission*, 3DIMPVT '12, pages 524–530, Washington, DC, USA, 2012. IEEE Computer Society. 1

[34] D. Nister, O. Naroditsky, and J. Bergen. Visual odometry. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2004)*, volume 1, pages I–652–I–659 Vol.1. IEEE Computer Society, June 2004. 2

[35] Occipital. ST depth. `STTracker` instance with the `kSTTrackerTypeKey` set to `STTrackerDepthBased`, 2013. 3, 4, 7, 9

[36] Occipital. ST hybrid. `STTracker` instance with the `kSTTrackerTypeKey` set to `STTrackerDepthAndColorBased`, 2013. 3, 4, 7, 9, 20

[37] Occipital Inc. The structure sensor. http://structure.io, 2014. 1, 2, 3, 4, 5, 6

[38] P. Ondrúška, P. Kohli, and S. Izadi. Mobilefusion: Real-time volumetric surface reconstruction and dense tracking on mobile phones. *IEEE Transactions on Visualization and Computer Graphics*, 21(11):1251–1258, Nov 2015. 3, 5

[39] F. Pomerleau, F. Colas, and R. Siegwart. A Review of Point Cloud Registration Algorithms for Mobile Robotics. *Foundations and Trends in Robotics*, 4(1-104):1–104, 2015. 4

[40] V. Prisacariu, O. Kähler, D. Murray, and I. Reid. Simultaneous 3d tracking and reconstruction on a mobile phone. In *Proceedings of the IEEE International Symposium on Mixed and Augmented Reality (ISMAR 2013)*, pages 89–98. IEEE Computer Society, Oct 2013. 5

[41] V. A. Prisacariu, . Kähler, D. Murray, and I. Reid. Real-time 3d tracking and reconstruction on mobile phones. *IEEE Transactions on Visualization and Computer Graphics*, 21(5):557–570, May 2015. 5

[42] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009. 19

[43] V. Rabaud and M. Dimashova. OpenCV ICP and RGBD visual odometry. Class `RgbdICPOdometry` and function `RGBDICPOdometryImpl()` in the file https://github.com/Itseez/opencv_contrib/tree/master/modules/rgbd/src/odometry.cpp, 2012. 4, 7

[44] V. Rabaud and M. Dimashova. OpenCV ICP-based visual odometry. Class `ICPOdometry` and function `RGBDICPOdometryImpl()` in the file https://github.com/Itseez/opencv_contrib/tree/master/modules/rgbd/src/odometry.cpp, 2012. 4, 7, 8

[45] V. Rabaud and M. Dimashova. OpenCV RGBD-based visual odometry. Class `RgbdOdometry` and function `RGBDICPOdometryImpl()` in the file https://github.com/Itseez/opencv_contrib/tree/master/modules/rgbd/src/odometry.cpp, 2012. 3, 4, 7

[46] V. Rabaud and M. Dimashova. OpenCV RGBD module. https://github.com/Itseez/opencv_contrib/tree/master/modules/rgbd, 2012. 2, 6, 7

[47] M. Raynal. *Concurrent Programming: Algorithms, Principles, and Foundations*. Springer Publishing Company, Incorporated, 2012. 19

[48] E. Rosten, R. Porter, and T. Drummond. Faster and Better: A Machine Learning Approach to Corner Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1):105–119, jan 2010. 6

[49] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: An efficient alternative to SIFT or SURF. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2564–2571. IEEE Computer Society, 2011. 3

[50] S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. In *Proceedings of the third International Conference on 3-D Digital Imaging and Modeling.*, pages 145–152. IEEE, 2001. 4

[51] R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *Proceedings of the 2011 IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011. IEEE Computer Society. 6

[52] D. Scaramuzza and F. Fraundorfer. Visual Odometry [Tutorial]. *IEEE Robotics & Automation Magazine*, 18(4):80–92, 2011. 3

[53] T. Schöps, J. Engel, and D. Cremers. Semi-dense visual odometry for AR on a smartphone. In *Proceedings of the IEEE International Symposium on Mixed and Augmented Reality (ISMAR 2014)*. IEEE Computer Society, September 2014. 5

[54] 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 *Proceedings of the International Conference on 3D Vision*, Piscataway, NJ, 2015. IEEE Computer Society. 5

[55] F. Steinbrücker, J. Sturm, and D. Cremers. Real-time visual odometry from dense RGB-D images. In *Proceedings of the IEEE International Conference on Computer Vision*, number 3, pages 719–722. IEEE Computer Society, 2011. 6, 7, 8

[56] J. Stückler and S. Behnke. Multi-resolution surfel maps for efficient dense 3D modeling and tracking. *Journal of Visual Communication and Image Representation*, 25(1):137–147, 2014. 3, 4, 6, 7, 9

[57] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of RGB-D SLAM systems. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS 2012)*, pages 573–580. IEEE Computer Society, 2012. 2, 5, 11

[58] P. Tanskanen, K. Kolev, L. Meier, F. Camposeco, O. Saurer, and M. Pollefeys. Live metric 3d reconstruction on mobile phones. In *Proceedings of the 2013 IEEE International Conference on Computer Vision (ICCV)*, pages 65–72. IEEE Computer Society, Dec 2013. 5

[59] C. Tomasi and R. Manduchi. Bilateral Filtering for Gray and Color Images. In *Proceedings of the 1998 IEEE International Conference on Computer Vision (ICCV)*, pages 839–846. IEEE Computer Society, 1998. 2

[60] T. Whelan, H. Johannsson, M. Kaess, J. Leonard, and J. McDonald. Robust real-time visual odometry for dense RGB-D mapping. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2013)*, Karlsruhe, Germany, May 2013. IEEE Computer Society. 3, 4, 6, 8

[61] D. Yang. *C++ and Object-Oriented Numeric Computing for Scientists and Engineers*. Springer New York, New York, NY, 2001. 19

[62] K. Yousif, A. Bab-Hadiashar, and R. Hoseinnezhad. An overview to visual odometry and visual slam: Applications to mobile robotics. *Intelligent Industrial Systems*, 1(4):289–311, 2015. 3

[63] J. Zhang, M. Kaess, and S. Singh. Real-time depth enhanced monocular odometry. In *Proceedings of the 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4973–4980. IEEE Computer Society, sep 2014. 6

[64] J. Zhang, M. Kaess, and S. Singh. A real-time method for depth enhanced visual odometry. *Autonomous Robots*, 41(1):31—43, 2015. 3, 15