This ICCV paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

GSLAM: A General SLAM Framework and Benchmark

Yong Zhao*[,1], Shibiao Xu†[,2], Shuhui Bu[1], Hongkai Jiang[1], and Pengcheng Han[1]

[1]Northwestern Polytechnical University
[2]NLPR, Institute of Automation, Chinese Academy of Sciences

Abstract

SLAM technology has recently seen many successes and attracted the attention of high-technological companies. However, how to unify the interface of existing or emerging algorithms, and effectively perform benchmark about the speed, robustness and portability are still problems. In this paper, we propose a novel SLAM platform named GSLAM, which not only provides evaluation functionality, but also supplies useful toolkit for researchers to quickly develop their SLAM systems. Our core contribution is an universal, cross-platform and full open-source SLAM interface for both research and commercial usage, which is aimed to handle interactions with input dataset, SLAM implementation, visualization and applications in an unified framework. Through this platform, users can implement their own functions for better performance with plugin form and further boost the application to practical usage of the SLAM.

1. Introduction

Simultaneous Localization and Mapping (SLAM) is a hot research topic in computer vision and robotics for several decades since the 1980s [3, 10, 14]. SLAM provides fundamental function for many applications that need real-time navigation like robotics, unmanned aerial vehicles (UAVs), autonomous driving, as well as virtual and augmented reality. In recent years, SLAM technology develops rapidly and a variety of SLAM systems have been proposed, including monocular SLAM system (key-point based [12, 37, 49], direct [15, 16, 53] and semi-direct methods [22, 23]), multi-sensor SLAM (RGBD [7, 36, 68], Stereo [17, 23, 51] and inertial aided methods [45, 56, 66]), learning based SLAM (supervised [6, 55, 67] and unsupervised methods [71, 72]).

However, with the rapidly developing SLAM technology, almost all the researchers focus on the theory and implementation of their own SLAM systems, which makes it difficult to exchange ideas and not easy to port the implementation to other systems. This prevents the quick apply to various industry fields. Currently, there exist many implementations of SLAM systems, how to effectively perform benchmark about the speed, robustness and portability is still a problem. Recently, Nardi *et al.* [52] and Bodin *et al.* [4] proposed uniform SLAM benchmark systems to perform quantitative, comparable and validatable experimental research for investigating trade-offs among various SLAM systems. Through these systems, the evaluation experiments can be easily performed by using the dataset, and metric evaluation modules.

As those systems only provide evaluation benchmarks, we consider it is possible to build a platform to serve the whole life-circle of SLAM algorithms including development, evaluation and application stages. In addition, deep learning based SLAM has achieved remarkable progress in recent years, it is necessary to create a platform which not only supports C++ but also Python for better supporting integration for geometric and deep learning based SLAM system. Therefore, in this paper we introduce a novel SLAM platform which provides not only evaluation functionality, but also useful toolkit for researchers to quickly develop their own SLAM systems. Through this platform, frequently used functions are provided with plugin forms, therefore, users could implement their own projects with directly using them or creating their own functions for better performance. We hope this platform could further boost the SLAM systems to practical applications. In summary, the main contributions of this work are as follows:

1. We presented an universal, cross-platform and full open-source SLAM platform for both research and commercial usages, which is beyond that of previous benchmarks. The SLAM interface is consisted by several lightweight, dependency-free headers, which makes it easy to interact with different datasets, S-

*Yong Zhao and Shibiao Xu contributed equally to this work and share the first authorship.

†Shibiao Xu and Shuhui Bu are joint corresponding authors (emails: shibiao.xu@nlpr.ia.ac.cn; bushuhui@nwpu.edu.cn).

LAM algorithms and applications with plugin forms in an unified framework. In addition, both JavaScript and Python are also provided for web based and deep learning based SLAM applications.

2. We introduced three optimized modules as utility classes including Estimator, Optimizer and Vocabulary in the proposed GSLAM platform. Estimator aims to provide a collection of close-form solvers cover all interesting cases with robust sample consensus (RANSAC); Optimizer aims to provide an unified interface for popular nonlinear SLAM problems; Vocabulary aims to provide an efficient and portable bag of words implementation for place recolonization with multi-thread and SIMD optimization.

3. Benefit from the above interface, we implemented and evaluated plugins for existing datasets, SLAM implementations and visualized applications in an unified framework, and emerging benchmark or applications could be further integrated easily in the future.

The source code of GSLAM with documentation wiki has been released, which can be found at our GitHub[1].

2. Related Works

In this section, we will briefly review the SLAM techniques including methods, systems and benchmarks.

2.1. Simultaneous Localization And Mapping

SLAM techniques build a map of an unknown environment and localize the sensor in the map with a strong focus on real-time operation. Early SLAM are mostly based on extended kalman filter (EKF) [12]. The 6 DOF motion parameters and 3D landmarks are probabilistically represented as a single state vector. The complexity of classic EKF grows quadratically with the number of landmarks, restricting its scalability. In recent years, SLAM technology develops rapidly and lots of monocular visual SLAM systems including key-point based [12, 37, 49], direct [15, 16, 53] and semi-direct methods [22, 23] are proposed. However, monocular SLAM systems lack scale information and are not able to handle pure rotation situation, then, some other multi-sensor SLAM systems including RGBD [7,36,68], Stereo [17,23,51] and inertial aided methods [45,56,66] are being studied for higher robustness and precision.

While a large number of SLAM algorithms have been presented, there has little effort to unify the interface of such algorithms, or to perform a holistic comparison of their capabilities. Implementations of these SLAM algorithms are often released as standalone executables rather than as libraries, and often do not conform to any standard structure.

[1] https://github.com/zdzhaoyong/GSLAM

Recently, supervised [6,55,67] and unsupervised [71,72] deep learning based visual odometers (VO) present novel ideas compared to traditional geometry based methods, but it is still not easy to optimize the predicted poses further for consistencies of multiple keyframes. GSLAM could help them for obtaining better global consistency, it is more easier to visualize or evaluate the results, and further be applied to various industry fields.

2.2. Computer Vision and Robotics Platform

Within the robotics and computer vision community, robotics middle-ware (e.g., ROS [57]) presents a very convenient communication way between nodes and is favored by most robotics researchers. Lots of SLAM implementations provide ROS wrapper to subscribe sensor data and publish visualization results. But it does not unify the input and output of SLAM implementations and is hard to further evaluate different SLAM systems. In this paper, GSLAM provides an alternative option to replace ROS inside the SLAM implementation, and maintains the compatibility.

2.3. SLAM Benchmarks

Currently, there exist several SLAM Benchmarks, including KITTI Benchmark Suite [28], TUM RGB-D Benchmarking [62] and ICL-NUIM RGB-D Benchmark Dataset [32], which only provide evaluation functionality. In addition, SLAMBench2 [4] expanded these benchmarks into algorithms and datasets, which requires users to make released implementation SLAMBench2-compatible for evaluation and it is difficult to extend to further applications. Different from these systems, the proposed GSLAM platform provides a solution which serves the whole life-circle of the SLAM implementation from development, evaluation to application. We provide useful toolkit for researchers to quickly develop their own SLAM system, and further visualization, evaluation and applications are developed based on an unified interface.

3. General SLAM Framework

The core work of GSLAM is to provide a general SLAM interface and framework. For better experience, the interface is designed to be lightweight, which is consisted by several headers and only relies on the C++11 standard library. And based on the interface, script language like JavaScript and Python are supported. In this section, the GSLAM framework is presented and a brief introduction of several basic interface classes is given.

3.1. Framework Overview

The framework of GSLAM is shown in Fig. 1, generally speaking, the interface is aimed to handle interaction with three parts:

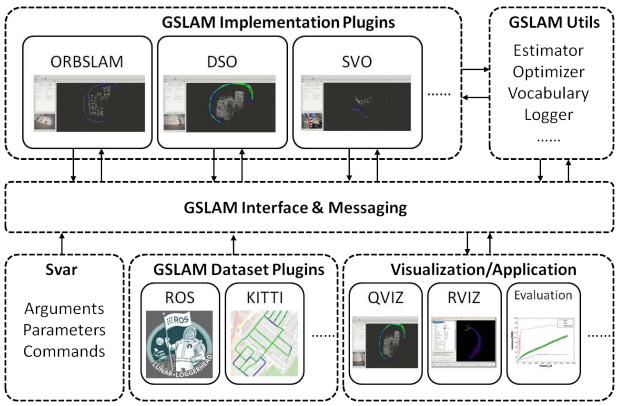


Figure 1: The framework overview of GSLAM.

1. The input of a SLAM implementation. When running a SLAM, the sensor data and some parameters are required. For GSLAM, a Svar class is used for parameters configuration and command handling. And all sensor data required by SLAM implementations are provided by a Dataset implementation and transfered using the Messenger. GSLAM implemented several popular visual SLAM datasets and users are free to implement his own dataset plugins.

2. The SLAM implementation. GSLAM treats each implementation as a plugin library. It is very easy for developers to design a SLAM implementation based on the GSLAM interface and utility classes. Developers can also wrap the implementation using the interface without extra dependency imported. Users can focus on the development of core algorithms without caring the input and output which should be handled outside the SLAM implementation.

3. The visualization part or applications using SLAM results. After SLAM implementations handled the input frames, users probably want to demonstrate or utilize the results. For generality, SLAM results should be published in a standard format. Default GSLAM uses Qt for visualization, but users are free to implement a customized visualizer and add application plugins such as an evaluation application.

The framework is designed to be compatible with different kinds of SLAM implementations include but not limited to monocular, stereo, RGBD and multiple camera visual inertial odometer with multi-sensor fusion. And now it best match feature based implementations while direct or deep learning based SLAM systems are also supported. As modern deep learning platforms and developers prefer Python for coding, GSLAM provides Python binding and thus developers are able to implement a SLAM using Python and call it with GSLAM or call a C++ based SLAM implementation with Python. Moreover, GSLAM could be used to train SLAM-modules, the supervised procedure can be summarized as: 1) compute sparse depth maps and camera poses with traditional SLAM plugin; 2) use the depth maps and camera poses as supervision to train estimators. GSLAM can also apply an unsupervised method to jointly learns depth and pose estimators, which only requires image sequence without ground truth depth for training via dataset plugins. Then, multi-view geometry constrains as losses are employed to train the network.

Table 1: Transform comparison with three popular implementations. The table statistics the time usage to run 1e6 times of transform multiply, point transform, exponential and logarithm in Milli seconds on an i7-6700 CPU running 64bit Ubuntu.

Method		GSLAM	Sophus	TooN	Ceres
$SO(3)$	*mult*	**14.9**	34.3	17.8	159.1
	trans	15.4	17.2	**14.5**	90.4
	exp	**80.7**	98.4	106.8	-
	log	**55.7**	72.5	63.8	-
$SE(3)$	*mult*	**28.6**	55.2	29.3	-
	trans	19.3	19.8	**12.1**	-
	exp	152.4	249.2	**99.2**	-
	log	**152.7**	194.0	205.8	-
$SIM(3)$	*mult*	**33.2**	58.5	34.5	-
	trans	16.9	17.2	**13.7**	-
	exp	**180.2**	286.8	229.0	-
	log	**202.5**	341.6	303.6	-

3.2. Basic Interface Classes

There are some data structures that are often used by the SLAM interface, including the parameter setting/reading, image format, pose transformation, camera model and map data structures. Here is going to give a brief introduction of some basic interface classes.

3.2.1 Parameter Setting

GSLAM uses a tiny arguments parsing and parameter setting class Svar, which only consists of a single header file depending on C++11 with the following features:

1. Arguments parsing and configure loading with help information. Similar to popular argument parsing tools like Google gflags[2], the variable configuration could be loaded from arguments, files and system environment. Users could also define different types of parameters with introduction which will be shown in help.

[2]https://github.com/gflags/gflags

2. A tiny script language with variable, function and condition, which makes configure file more powerful.

3. Thread-safe variable binding and sharing. Variables used with very high frequency are suggested to bind with pointer or reference, which provides high efficiency along with convenience.

4. Simple function definition and calling from both C++ or plain script. A binding between command and function helps developers decouple the file dependencies.

5. Support tree structure presentation, which means it is easy to load or save configuration with XML, JSON and YAML formats.

3.2.2 Intra-Process Messaging

As ROS presents a very convenient communication way between nodes and is favored by most robotics researchers. Inspired by the ROS2 messaging architecture, GSLAM implements a similar intra-process communication utility class named Messenger. This provides an alternative option to replace ROS inside the SLAM implementation and maintains the compatibility. Due to the intra-process design, the Messenger is able to publish and subscribe any class without extra cost. More features are listed below:

1. The interface keeps ROS style and easily for users to get started. And all ROS defined messages are supported, which means very few works are needed to replace the original ROS messaging.

2. Since there is no serialization and data transferring, messages can be sent without latency and extra cost. Meanwhile the payload is not limited to ROS defined messages but any copyable data structures are supported.

3. The source are header files only based on C++11 with no extra dependency, which makes it portable.

4. The API is thread-safe and supports multi-thread condition notify when the queue size is greater than zero. Both topic name and RTTI data structure check are done before a publisher and subscriber are connected from each other to ensure correct calls.

3.2.3 3D Transformation

Rotation, rigid and similarity are three of the most used transformations in SLAM research. A similarity transformation of a point $\mathbf{p} = (x, y, z)^T$ is common to use a 4×4 homogeneous transformation matrix or decompose such a matrix into rotational and translational components:

Table 2: Algorithms which the GSLAM Estimator implemented.

Algorithm		Ref.	Model
2D-2D	*F8-Point*	[20]	Fundamental
	F7-Point	[33]	Fundamental
	E5-Stewenius	[61]	Essential
	E5-Nister	[54]	Essential
	E5-Kneip	[42]	Essential
	H4-Point	[33]	Homography
	A3-Point	[5]	Affine2D
2D-3D	*P4-EPnP*	[44]	SE3
	P3-Gao	[27]	SE3
	P3-Kneip	[41]	SE3
	P3-GPnP	[40]	SE3
	P2-Kneip	[38]	SE3
	T2-Triangulate	[39]	Translation
3D-3D	*A4-Point*	[5]	Affine3D
	S3-Horn	[34]	SIM3
	P3-Plane	[41]	SE3

$$\begin{bmatrix} \mathbf{p}' \\ 1 \end{bmatrix} = \begin{bmatrix} s\mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix}. \tag{1}$$

Here $\mathbf{R} \in \mathbb{R}^{3\times3}$ represents the rotation matrix, which is given as a member of the $SO(3)$ Lie group [31] with three unit direction axises. $\mathbf{t} \in \mathbb{R}^{\mathbf{3}}$ means the translation and s is the scale factor. The similarity transform matrix belongs to the $SIM(3)$ group. When the scale $s = 1$, the transform becomes a rigid transform and belongs to the $SE(3)$ group.

For the rotational component, there are several choices for representation, including the matrix, Euler angle, unit quaternion and Lie algebra $so(3)$. For a given transformation, we can use any of these for representation and can convert one to another. However, we need to pay close attention to the selected representation when we consider multiple transformations and manifold optimization. The matrix representation is overparamatrized with 9 parameters where as the rotation only has 3 degrees of freedom (DOF). The Euler angle representation uses 3 variable and is easy to understand but faces the well-known gimbal lock problem and not convenience to multiple transformations. The unit quaternion is the most efficient way to perform multiple and Lie algebra is the common representation to perform manifold optimization. The matrix representation of rotation $\mathbf{R}$ is calculated from $\phi \in \mathbb{R}^3$ using the exponential function according to Lie algebra $so(3)$:

$$\mathbf{R} = \exp(\phi^\wedge) = \exp(\theta\mathbf{a}^\wedge) \tag{2}$$

$$= \cos\theta\mathbf{I} + (1 - \cos\theta)\mathbf{a}\mathbf{a}^T + \sin\theta\mathbf{a}^T. \tag{3}$$

Where $\mathbf{a}$ is the rotation axis and θ is the angle to rotate. $\phi^\wedge$ is the skew-symmetric matrix of ϕ.

Similarly the Lie algebra of rigid and similarity transformation $se(3)$ and $sim(3)$ are defined. GSLAM uses quaternion to represent the rotational component and provide functions converting from one representation to other representations. Table 1 demonstrates our transforms implementation with comparison to three other popular implementations Sophus, TooN and Ceres. Since Ceres implementation uses the angle axis representation, the rotation exponential and logarithm are not needed. As the table demonstrates, the GSLAM implementation outperforms due to the use of quaternion and better optimization, while TooN utilizes the matrix implementation and outperforms on point transformation.

3.2.4 Image Format

Image data storing and transferring are two of the most important functions for visual SLAM. For efficiency and convenience, GSLAM utilizes a data structure GImage which is compatible to cv::Mat. It has a smart point counter for safely memory free and is easy to be transfered without memory copy. And the data pointer is aligned so that it would be easier for single instruction multiple data (SIMD) speed up. Users can convert between GImage and cv::Mat seamlessly and safely without memory copy.

3.2.5 Camera Models

A camera model should be defined to project a 3D point $\mathbf{p}_c$ from camera coordinates to 2D pixel $\mathbf{x}$. One most popular camera model is the pinhole model where the projection can be represented by multiply an intrinsic matrix K known as:

$$\mathbf{x} = \mathbf{K}\mathbf{p}_c = \begin{bmatrix} f_x & & c_x \\ & f_y & c_y \\ & & 1 \end{bmatrix} \mathbf{p}_c \tag{4}$$

As images for SLAM possibly contain radial and tangential distortion due to imperfect manufacturing or are captured with a fish-eye or panorama camera, different camera models are proposed to describe the projection. GSLAM provides implementations including the OpenCV [24] (used by ORBSLAM [51]), ATAN (used by PTAM [37]) and O-CamCalib [59] (used by MultiCol-SLAM [65]). Users are also easy to inherit the class and implement some other camera models like Kannala-Brandt [35] and Equirectangular panorama model.

3.2.6 Map Data Structure

For a SLAM implementation, its goal is to localize the real-time poses and generate a map. GSLAM suggests an unified map data structure which is consisted by several mapframes and mappoints. This data structure is appropriate for most of the existed visual SLAM systems including both feature based or direct methods.

Table 3: Comparison of four BoW implementations in loading, saving and training a same vocabulary with memory usage statistics. The experiment is performed on a computer with i7-6700 CPU, 16GB RAM running 64bit Ubuntu. 400 and 10k images from DroneMap [8] dataset are used to train the models with 4 and 6 levels.

Implementation		Ours	DBoW2	DBoW3	FBoW
Load	*ORB-4*	**67.3us**	47.2ms	7.1ms	72.3us
	ORB-6	**7.2ms**	6.8 s	1.1 s	9.5ms
	SIFT-4	**1.0ms**	436.1ms	5.1ms	1.1ms
Save	*ORB-4*	**437.9us**	40.4ms	1.7ms	553.1us
	ORB-6	34.4ms	4.8 s	632.4ms	**20.6ms**
	SIFT-4	4.4ms	437.6ms	6.7ms	**2.7ms**
Train	*ORB-4*	**7.6 s**	24.8 s	23.6 s	8.5 s
	ORB-6	**230.5 s**	1.1Ks	911.4 s	270.4 s
	SIFT-4	23.5 s	327.7 s	299.0 s	**18.7 s**
Trans	*ORB-4*	**615.5us**	2.1ms	1.9ms	862.4us
	ORB-6	**723.7us**	6.0ms	4.9ms	1.2ms
	SIFT-4	**1.1ms**	10.3ms	9.2ms	11.5ms
Mem	*ORB-4*	**0.44MB**	2.5MB	2.5MB	0.45MB
	ORB-6	**44.4MB**	247.1MB	246.5MB	45.3MB
	SIFT-4	**5.8MB**	7.8MB	7.8MB	5.8MB

Mapframes are used to represent location statuses in different times with various information captured by sensors or estimated results including IMU or GPS raw data, depth information and camera models. Relationships between them are estimated by SLAM implementations and their connections form a pose graph.

Mappoints are used to express the environment observed by frames, which are generally used by feature based methods. However, a mappoint could not only represents a keypoint but also a GCP, edge line or 3D object. Their correspondences with mapframes form an observation graph which are often called as bundle graph.

4. SLAM Implementation Utilities

For making things easier to implement a SLAM system, GSLAM provides some utility classes. This section will briefly introduce three optimized modules named Estimator, Optimizer and Vocabulary.

4.1. Estimator

The purely geometric computation remains a fundamental problem that requires robust and accurate real-time solutions. Both classical visual SLAM algorithms [22, 37, 49]) or modern visual-inertial solutions [45, 56, 66] rely on geometric vision algorithms for initialization, relocalization and loop-closure. OpenCV [5] provides several geometry

Table 4: Dataset plugins build-in implemented until now.

Dataset	Year	Environment	Type
KITTI [29]	2012	outdoors	multi-cam, imu
TUMRGBD [63]	2012	indoors	RGBD
ICL [32]	2014	simulation	RGBD
TUMMono [18]	2016	indoors	mono
Euroc [9]	2016	indoors	stereo, imu
NPUDroneMap [8]	2016	aerial	mono
TUMVI [60]	2018	in/outdoors	stereo, imu
CVMono [5]	-	-	mono
ROS [57]	-	-	-

algorithms and Kneip presents a toolbox for geometric vision OpenGV [39] which is limited to camera pose computation. Estimator of GSLAM aims to provide a collection of close-form solvers cover all interesting cases with robust sample consensus (RANSAC) [19] methods.

Table 2 lists the algorithms supported by the estimator. They are divided into three categories according to the given observations. 2D-2D matches are used to estimate epipolar or homography constraints and relative pose could be decomposed from them. 2D-3D correspondences are used to estimate both central or non-central absolute pose for monocular or multiple camera systems, which is the famous PnP problem. 3D geometry functions such as plane fitting, and estimating the $SIM3$ transformation of two point clouds are also supported. Most algorithms are implemented depending on the linear algebra library Eigen, which is header-only and readily for most platforms.

4.2. Optimizer

Nonlinear optimization is the core part of state-of-the-art geometric SLAM systems. Due to the high dimensionality and sparseness of Hessian matrix, graph structures are used to modeling complex estimation problems for SLAM. Several frameworks including Ceres [1], G2O [43] and GTSAM [13] are proposed to solve general graph optimization problems. These frameworks are popular used by different SLAM systems. ORB-SLAM [49, 51], SVO [22, 23] use G2O for bundle adjustment and pose graph optimization. OKVIS [45], VINS [56] use Ceres for graph optimization with IMU factors and sliding window is used to control the computation complex. Forster *et al.* present a visual-initial method [21] based on SVO and implement the back-end with GTSAM.

Optimizer of GSLAM aims to provide an unified interface for most of nonlinear SLAM problems such as PnP solver, bundle adjustment, pose graph optimization. A general implementation plugin for these problems is carried out based on the Ceres library. For a particular problem such as bundle adjustment, some more efficient implementations

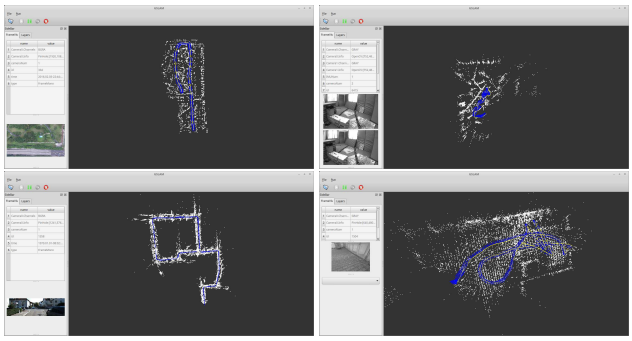


Figure 2: ORBSLAM [49, 51] can run on different datasets with only one parameter modified in GSLAM, including NPUDroneMap [8] (left-top), Euroc [9] (right-top), KITTI [29] (left-bottom) and TUMMono [18] (right-bottom).

such as PBA [70] and ICE-BA [46] could also be provided as a plugin. With the optimizer utility, developers are able to access different implementations with an united interface, particularly for deep learning based SLAM systems.

4.3. Vocabulary

Place recognition is one of the most important part for SLAM relocalization and loop detection. Bag of words (BoW) approach is popular used in SLAM systems since its efficiency and performance. FabMap [11] [30] propose a probabilistic approach to the problem of recognizing places based on their appearance, which is used by RSLAM [47], LSD-SLAM [16]. As it uses float descriptors like SIFT and SURF, DBoW2 [25] builds a vocabulary tree for training and detection, which supports both binary and float descriptors. Rafael presents two improved versions of DBoW2 named DBoW3 and FBoW [48], which simplify the interface and accelerate the training and loading speed. After ORB-SLAM [49] adopts the ORB [58] descriptor and uses DBoW2 for loop detection [50], relocalization and fast matching, a varies of SLAM systems such as ORB-SLAM2 [51], VINS-Mono [56] and LDSO [26] use DBoW3 for loop detection. It has become the most popular tool to implement place recognition for SLAM systems.

Inspired by the above works, GSLAM carries out a header only implementation of DBoW3 vocabulary with the following features:

1. Removed OpenCV dependency and the all functions are implemented within one single header file only depending on C++11.

2. Combined the advantages of both DBoW2/3 and FBoW [48] which are extremely fast and easy to use. Interface similar to DBoW3 is provided and both binary and float descriptors are accelerated using SSE and AVX instructions.

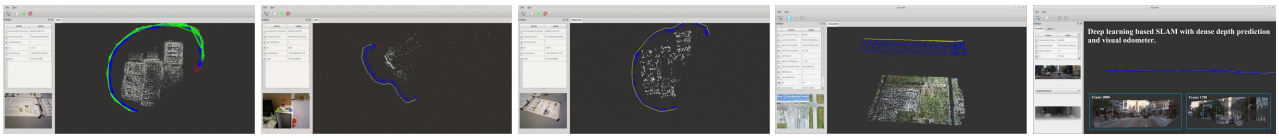


Figure 3: Screenshots of some SLAM and SfM plugins implemented, including direct DSO [15], semi-direct visual odometer SVO [22, 23], feature based ORBSLAM [49, 51], global SfM system TheiaSfM [64] and dense depth learning [2].

3. We improved the memory usage and accelerated the speed of loading, saving or training a vocabulary and transformation from images features to a BoW vector.

A comparison of the four implementations is demonstrated in Table 3. In the experiment, each parent node has 10 children, and for ORB feature detection we use the ORB-SLAM [51], and SiftGPU [69] is used for SIFT detection. Two ORB vocabularies with 4 and 6 levels and one SIFT vocabulary are used in the results. Both FBoW and GSLAM use multi-thread for vocabulary training. Our implementation outperforms in almost all items including loading and saving the vocabulary, training a new vocabulary, transforming a descriptor list to a BoW vector for place recognition and a feature vector for fast feature matching. Furthermore, the GSLAM implementation uses less memory and allocates less pieces of dynamic memories as we found that the fragmentation problem is the main reason that DBoW2 needs lots of memory.

5. SLAM Evaluation Benchmark

Existed benchmarks [29, 63] need users download test datasets and upload results for precision evaluation, which are not able to unify the running environment and evaluate a fairy performance comparison. Benefit from the unified interface of GSLAM, the evaluation of SLAM systems becomes much more elegant. With help of GSLAM, developers just need to upload the SLAM plugin and various evaluations on speed, computation cost and accuracy could be done in a dockerlized environment with fixed resources.

In this section, an evaluation is carried out with three representative SLAM implementations on speed, accuracy, memory and CPU usages, which is performed to demonstrate the possibility of an united SLAM benchmark with different SLAM implementation plugins.

5.1. Datasets

A sensor data stream with corresponding configuration is always needed to run a SLAM system. For letting developers focus on the development of the core SLAM plugins, GSLAM provides a standard dataset interface where developers do not need to take care of the SLAM inputs. Both online sensor input and offline data are provided through different dataset plugins, and correct plugin is dynamic loaded by identify the given dataset path suffix. A dataset implementation should provide all sensor stream requested with related configurations, thus no extra setting is needed for different datasets. All different sensor streams are published through Messenger introduced in Sec. 3.2.2 with standard topic names and data formats.

GSLAM has already implemented several popular visual SLAM dataset plugins which are listed in Table. 4. It is also very easy for users to implement a dataset plugin based on the header-only GSLAM core and publish it as a plugin or compile it along with the applications. Furthermore, We provide the screenshots that ORBSLAM can run on different datasets with only one parameter modified in Fig. 2.

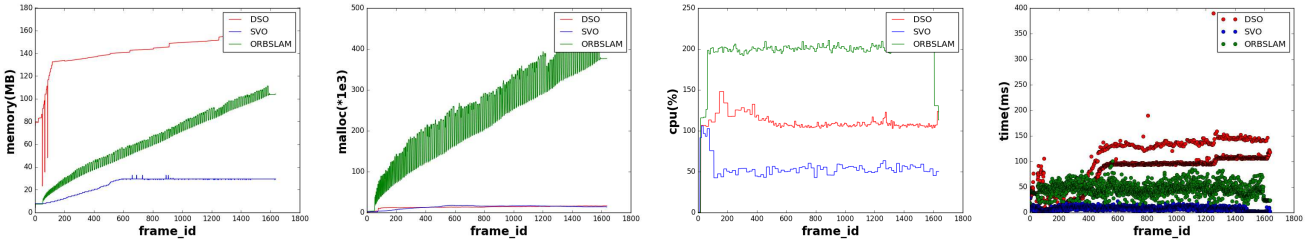
5.2. SLAM Implementations

Fig. 3 demonstrates the screenshots of some open-source SLAM and SfM plugins running with build-in Qt visualizer. Different architectures of SLAM systems including direct, semi-direct, feature based, even SfM methods and learning based dense depth estimation are supported by our framework. It should be mentioned that since SVO, ORB-SLAM and TheiaSfM utilize the map data structure introduced in Sec. 3.2.6, the visualization is auto supported. The DSO implementation needs to publish the results such as pointcloud, camera poses, trajectory and pose graph for visualization just like the ROS based implementation does. Users are able to access different SLAM plugins with the unified framework and it is very convenient to develop a SLAM based applications depending on the C++, Python and Node-JS interfaces. As many researchers use ROS for development, GSLAM also provides the ROS visualizer plugin to transfer the ROS defined messages seamlessly, and developers could utilize Rviz for display or continue to develop other ROS based applications.

5.3. Evaluation

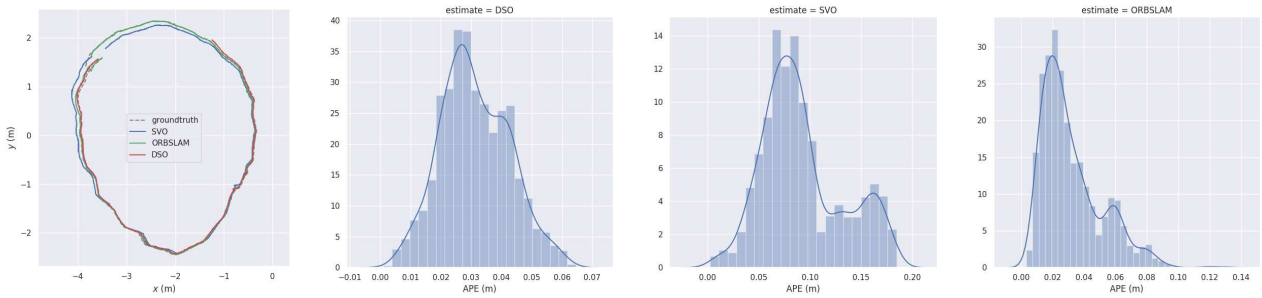
As most benchmarks only provide datasets with or without ground-truth for users to perform evaluations by themselves. GSLAM provides a build-in plugin and script tools for both computation performance and accuracy evaluation.

The sequence *nostructure-texture-near-withloop* from TUMRGBD dataset is used to demonstrate how the evaluation performs. And three open-source monocular SLAM plugins DSO, SVO and ORBSLAM are adopted for the following experiments. A computer with i7-6700 CPU, GTX



(a) *Memory usage* (b) *Memory malloc number* (c) *CPU usage* (d) *Frame duration*

Figure 4: Computation performance statistics of three monocular implementations integrated within the evaluation tool. The recordings of memory usage and memory allocated numbers are started after the SLAM application loaded, and updated after every frame processed. CPU usage is updated when the process occupied CPU time increases a curtain value. Frame duration is measured by the time between current frame published and processed.



(a) *Trajectory aligned* (b) *DSO* (c) *SVO* (d) *ORBSLAM*

Figure 5: Odometer trajectories aligned with ground-truth (left) and absolute pose error (APE) distributions of DSO, SVO and ORBSLAM. The odometer trajectories published lively instead of final results are used.

1060 GPU and 16GB RAM running 64bit Ubuntu 16.04 is used for all the experiments.

The computation performance evaluation including memory usage, malloc numbers, CPU usage and time used by every-frame statistics are shown in Fig. 4. The results demonstrate that SVO uses the least memory, CPU resources and obtains fastest speed. And all cost keeps stable since SVO is a visual odometer and just a local map is maintained inside the implementation. DSO mallocs fewer memory block numbers, but consumes more than 100MB RAM which increases slowly. One problem of DSO is that the processing time increases dramatically when frame number is below 500, in addition, the processing times for key-frames are even longer. ORBSLAM uses the most CPU resources and the computation time is stable, but the memory usage increases fast and it allocates and frees a lot of memory blocks since the bundle adjustment uses the G2O library and no incremental optimization approach is used.

The odometer trajectory evaluation is presented in Fig. 5. As we can see, SVO is faster but have much higher drift, while ORBSLAM achieves the best accuracy in terms of absolute pose error (APE). The relative pose error (RPE) are also provided, however due to the limitation of the paragraph, more experimental results are provided in the supplementary materials. Since the integrated evaluation is a pluggable plugin application, it can be reimplemented with more evaluation metrics such as the precision of pointcloud.

6. Conclusions

In this paper, we introduce a novel and generic SLAM platform named GSLAM, which provides support from development, evaluation to application. Frequently used toolkits are provided by plugin form, and users can also easily develop their own modules. To make the platform easy to be used, we make the interfaces only depend C++11. In addition, Python and JavaScript interfaces are provided for better integrating traditional and deep learning based SLAM or distributed on the web.

7. Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grants 61573284, 61620106003, 91860124, 61671451, 51875459 and 61971418. This work was also supported by the National Key R&D Program of China (Grant 2018YFB2100602).

References

[1] Sameer Agarwal, Keir Mierle, and Others. Ceres solver. http://ceres-solver.org.

[2] Ibraheem Alhashim and Peter Wonka. High quality monocular depth estimation via transfer learning. *CoRR*, abs/1812.11941, 2018.

[3] Tim Bailey and Hugh Durrant-Whyte. Simultaneous localization and mapping (slam): Part ii. *IEEE Robotics & Automation Magazine*, 13(3):108–117, 2006.

[4] Bruno Bodin, Harry Wagstaff, Sajad Saecdi, Luigi Nardi, Emanuele Vespa, John Mawer, Andy Nisbet, Mikel Luján, Steve Furber, Andrew J Davison, et al. Slambench2: Multi-objective head-to-head benchmarking for visual slam. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[5] Gary Bradski et al. The opencv library (2000). *Dr. Dobbs Journal of Software Tools*, 2000.

[6] Samarth Brahmbhatt, Jinwei Gu, Kihwan Kim, James Hays, and Jan Kautz. Mapnet: Geometry-aware learning of maps for camera localization. *arXiv preprint arXiv:1712.03342*, 2017.

[7] Shuhui Bu, Yong Zhao, Gang Wan, and Ke Li. Semi-direct tracking and mapping with rgb-d camera for mav. *Multimedia Tools and Applications*, 75:1–25, 2016.

[8] Shuhui Bu, Yong Zhao, Gang Wan, and Zhenbao Liu. Map2dfusion: real-time incremental uav image mosaicing based on monocular slam. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 4564–4571. IEEE, 2016.

[9] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The euroc micro aerial vehicle datasets. *The International Journal of Robotics Research*, 35(10):1157–1163, 2016.

[10] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

[11] Mark Cummins and Paul Newman. Fab-map: Probabilistic localization and mapping in the space of appearance. *The International Journal of Robotics Research*, 27(6):647–665, 2008.

[12] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1052–1067, 2007.

[13] Frank Dellaert. Factor graphs and gtsam: A hands-on introduction. Technical report, Georgia Institute of Technology, 2012.

[14] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[15] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *IEEE transactions on pattern analysis and machine intelligence*, 40(3):611–625, 2018.

[16] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *Computer Vision–ECCV 2014*, pages 834–849. Springer, 2014.

[17] Jakob Engel, Jorg Stuckler, and Daniel Cremers. Large-scale direct slam with stereo cameras. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 1935–1942. IEEE, 2015.

[18] Jakob Engel, Vladyslav Usenko, and Daniel Cremers. A photometrically calibrated benchmark for monocular visual odometry. *arXiv preprint arXiv:1607.02555*, 2016.

[19] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[20] Martin A Fischler and Oscar Firschein. *Readings in computer vision: issues, problem, principles, and paradigms*. Elsevier, 2014.

[21] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. On-manifold preintegration for real-time visual–inertial odometry. *IEEE Transactions on Robotics*, 33(1):1–21, 2017.

[22] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. S-vo: Fast semi-direct monocular visual odometry. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 15–22. IEEE, 2014.

[23] Christian Forster, Zichao Zhang, Michael Gassner, Manuel Werlberger, and Davide Scaramuzza. Svo: Semidirect visual odometry for monocular and multicamera systems. *IEEE Transactions on Robotics*, 33(2):249–265, 2017.

[24] John G Fryer and Duane C Brown. Lens distortion for close-range photogrammetry. *Photogrammetric engineering and remote sensing*, 52(1):51–58, 1986.

[25] Dorian Gálvez-López and J. D. Tardós. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, October 2012.

[26] Xiang Gao, Rui Wang, Nikolaus Demmel, and Daniel Cremers. Ldso: Direct sparse odometry with loop closure. In *International Conference on Intelligent Robots and Systems (IROS)*, October 2018.

[27] Xiao-Shan Gao, Xiao-Rong Hou, Jianliang Tang, and Hang-Fei Cheng. Complete solution classification for the perspective-three-point problem. *IEEE transactions on pattern analysis and machine intelligence*, 25(8):930–943, 2003.

[28] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[29] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3354–3361. IEEE, 2012.

[30] Arren Glover, William Maddern, Michael Warren, Stephanie Reid, Michael Milford, and Gordon Wyeth. Openfabmap: An open source toolbox for appearance-based loop closure detection. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 4730–4735, May 2012.

[31] Jose Manuel Pardos Gotor. Lie groups and lie algebras in robotics. In *Computational Noncommutative Algebra and Applications*, pages 101–125. Springer, 2004.

[32] Ankur Handa, Thomas Whelan, John McDonald, and Andrew J Davison. A benchmark for rgb-d visual odometry, 3d reconstruction and slam. In *Robotics and automation (ICRA), 2014 IEEE international conference on*, pages 1524–1531. IEEE, 2014.

[33] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[34] Berthold KP Horn. Closed-form solution of absolute orientation using unit quaternions. *JOSA A*, 4(4):629–642, 1987.

[35] Juho Kannala and Sami S Brandt. A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses. *IEEE transactions on pattern analysis and machine intelligence*, 28(8):1335–1340, 2006.

[36] Christian Kerl, Jurgen Sturm, and Daniel Cremers. Dense visual slam for rgb-d cameras. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 2100–2106. IEEE, 2013.

[37] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234. IEEE, 2007.

[38] Laurent Kneip, Margarita Chli, and Roland Y Siegwart. Robust real-time visual odometry with a single camera and an imu. In *Proceedings of the British Machine Vision Conference 2011*. British Machine Vision Association, 2011.

[39] Laurent Kneip and Paul Furgale. Opengv: A unified and generalized approach to real-time calibrated geometric vision. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 1–8. IEEE, 2014.

[40] Laurent Kneip, Paul Furgale, and Roland Siegwart. Using multi-camera systems in robotics: Efficient solutions to the npnp problem. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3770–3776. IEEE, 2013.

[41] Laurent Kneip, Davide Scaramuzza, and Roland Siegwart. A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation. 2011.

[42] Laurent Kneip, Roland Siegwart, and Marc Pollefeys. Finding the exact rotation between two images independently of the translation. In *European conference on computer vision*, pages 696–709. Springer, 2012.

[43] Rainer Kummerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g2o: A general framework for graph optimization. In *IEEE International Conference on Robotics and Automation*, 2011.

[44] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: An accurate o (n) solution to the pnp problem. *International journal of computer vision*, 81(2):155, 2009.

[45] Stefan Leutenegger, Simon Lynen, Michael Bosse, Roland Siegwart, and Paul Furgale. Keyframe-based visual–inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, 34(3):314–334, 2015.

[46] Haomin Liu, Mingyu Chen, Guofeng Zhang, Hujun Bao, and Yingze Bao. Ice-ba: Incremental, consistent and efficient bundle adjustment for visual-inertial slam. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[47] Christopher Mei, Gabe Sibley, Mark Cummins, Paul Newman, and Ian Reid. Rslam: A system for large-scale mapping in constant-time using stereo. *International journal of computer vision*, 94(2):198–214, 2011.

[48] Rafael Muoz-Salinas. FBox fast bag of words, 2017.

[49] Raul Mur-Artal, JMM Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *arXiv preprint arXiv:1502.00956*, 2015.

[50] Rául Mur-Artal and Juan D. Tardós. Fast relocalisation and loop closing in keyframe-based slam. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, 2014.

[51] Raul Mur-Artal and Juan D Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[52] Luigi Nardi, Bruno Bodin, M Zeeshan Zia, John Mawer, Andy Nisbet, Paul HJ Kelly, Andrew J Davison, Mikel Luján, Michael FP O'Boyle, Graham Riley, et al. Introducing slambench, a performance and accuracy benchmarking methodology for slam. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 5783–5790. IEEE, 2015.

[53] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. Dtam: Dense tracking and mapping in real-time. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2320–2327. IEEE, 2011.

[54] David Nistér. An efficient solution to the five-point relative pose problem. *IEEE transactions on pattern analysis and machine intelligence*, 26(6):756–770, 2004.

[55] Gabriel L Oliveira, Noha Radwan, Wolfram Burgard, and Thomas Brox. Topometric localization with deep learning. *arXiv preprint arXiv:1706.08775*, 2017.

[56] Tong Qin, Peiliang Li, and Shaojie Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, 2018.

[57] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Ng Andrew. Ros : an open-source robot operating system. *Proc. IEEE ICRA Workshop on Open Source Robotics*, 2009.

[58] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: an efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564–2571. IEEE, 2011.

[59] Davide Scaramuzza, Agostino Martinelli, and Roland Siegwart. A flexible technique for accurate omnidirectional camera calibration and structure from motion. In *Computer Vision Systems, 2006 ICVS'06. IEEE International Conference on*, pages 45–45. IEEE, 2006.

[60] David Schubert, Thore Goll, Nikolaus Demmel, Vladyslav Usenko, Jorg Stckler, and Daniel Cremers. The tum vi

benchmark for evaluating visual-inertial odometry. In *International Conference on Intelligent Robots and Systems (IROS)*, October 2018.

[61] Henrik Stewenius, Christopher Engels, and David Nistér. Recent developments on direct relative orientation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 60(4):284–294, 2006.

[62] Jrgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 573–580, Oct 2012.

[63] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 573–580. IEEE, 2012.

[64] Chris Sweeney. Theia multiview geometry library: Tutorial & reference. *University of California Santa Barbara*, 2, 2015.

[65] Steffen Urban and Stefan Hinz. Multicol-slam-a modular real-time multi-camera slam system. *arXiv preprint arXiv:1610.07336*, 2016.

[66] Lukas von Stumberg, Vladyslav Usenko, and Daniel Cremers. Direct sparse visual-inertial odometry using dynamic marginalization. *arXiv preprint arXiv:1804.05625*, 2018.

[67] Sen Wang, Ronald Clark, Hongkai Wen, and Niki Trigoni. Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 2043–2050. IEEE, 2017.

[68] Thomas Whelan, Renato F Salas-Moreno, Ben Glocker, Andrew J Davison, and Stefan Leutenegger. Elasticfusion: Real-time dense slam and light source estimation. *The International Journal of Robotics Research*, 35(14):1697–1716, 2016.

[69] Changchang Wu. Siftgpu: A gpu implementation of scale invariant feature transform (sift)(2007). *URL http://cs. unc. edu/˜ ccwu/siftgpu*, 2011.

[70] Changchang Wu, Sameer Agarwal, Brian Curless, and Steven M Seitz. Multicore bundle adjustment. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3057–3064. IEEE, 2011.

[71] Zhichao Yin and Jianping Shi. Geonet: Unsupervised learning of dense depth, optical flow and camera pose. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, 2018.

[72] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. Unsupervised learning of depth and ego-motion from video. In *CVPR*, volume 2, page 7, 2017.