SLAMBench2: Multi-Objective Head-to-Head Benchmarking for Visual SLAM

Bruno Bodin[†], Harry Wagstaff[†], Sajad Saeedi[‡], Luigi Nardi[•], Emanuele Vespa[‡], John Mawer[⋆], Andy Nisbet[⋆], Mikel Luján[⋆], Steve Furber[⋆], Andrew J. Davison[‡], Paul H. J. Kelly[‡], and Michael F. P. O'Boyle[†],

Abstract— **SLAM is becoming a key component of robotics and augmented reality (AR) systems. While a large number of SLAM algorithms have been presented, there has been little effort to unify the interface of such algorithms, or to perform a holistic comparison of their capabilities. This is a problem since different SLAM applications can have different functional and non-functional requirements. For example, a mobile phone-based AR application has a tight energy budget, while a UAV navigation system usually requires high accuracy. SLAMBench2 is a benchmarking framework to evaluate existing and future SLAM systems, both open and close source, over an extensible list of datasets, while using a comparable and clearly specified list of performance metrics. A wide variety of existing SLAM algorithms and datasets is supported, e.g. ElasticFusion, InfiniTAM, ORB-SLAM2, OKVIS, and integrating new ones is straightforward and clearly specified by the framework. SLAMBench2 is a publicly-available software framework which represents a starting point for quantitative, comparable and validatable experimental research to investigate trade-offs across SLAM systems.**

I. Introduction

Over the last decade there has been sustained research interest in Simultaneous Localisation and Mapping (SLAM). Every year, new SLAM systems are released, each containing new ideas, supporting new sensors, and tackling more challenging datasets. As the area matures and the number of systems increases, the need to objectively evaluate them against prior work intensifies. However, robust quantitative, comparable and validatable evaluation of competitive approaches can take considerable effort, delaying innovation.

One of the main difficulties in robust evaluation is the numerous different software platforms in which research is embedded. Each has to be installed and setup with the appropriate libraries and device drivers. More important is the variations in input formats, hardware setups, and various parameters such as calibration and algorithmic parameters. These variations must be carefully managed to provide the desired results. For instance, applying similar parameters to different datasets may lead to failure. Trying to compare two different algorithms with the same sensors, datasets, and hardware platform may involve a significant amount of redundant software engineering effort.

In this paper we present the SLAMBench2 benchmarking framework [1], which aims to provide reproducibility of results for different datasets and SLAM algorithms. It provides a single plug and play environment where users can try different algorithms, sensors, datasets, and evaluation metrics, allowing robust evaluation.

The need to systematise and reduce redundant evaluation effort is not new and has already been recognised, e.g. the SLAM Evaluation 2012 from the KITTI Vision Benchmark Suite [1] which is limited to a specific dataset and environment. More recently, SLAMBench [2] provided multiple implementations (C++, OpenMP, OpenCL and CUDA) of a single algorithm, KinectFusion, and dataset, ICL-NUIM [3].

In this paper we introduce SLAMBench2, which is designed to enable the evaluation of open source or proprietary SLAM systems, over an extensible sets of datasets and clearly specified metrics. SLAMBench2 is inspired by the work in [2], [4], [5]. It currently supports eight different algorithms (dense, semi-dense, and sparse) and three datasets. It is a tool that enables the reproducibility of results for existing SLAM systems and allows the integration and evaluation of new SLAM results. Through a series of use-cases (Sec. V-B), we demonstrate its simplicity of use as well as a variety of results that have already been collected. We make the following contributions:

- A publicly-available framework that enables quantitative reproducible comparison of SLAM systems.
- A system that is *dataset-agnostic*, supports *plug and play* of new algorithms, and provides a *modular user interface*.
- A quantitative comparison between eight state-of-the-art SLAM systems.

II. Background and Related Work

A. Simultaneous Localisation And Mapping

Simultaneous Localisation and Mapping is the process of localising a sensor in a previously unexplored environment while building a map of its environment incrementally. The interdependence of localisation and mapping makes the problem more complex, and researchers have proposed various solutions such as joint optimisation using Kalman filters [6] or parallel tracking and mapping [7].

While in theory one may claim that SLAM is a solved problem [8], in practice it is still considered challenging [9] when a specific level of performance is required (e.g. frame rate, accuracy). For example, in visual SLAM, when a camera moves rapidly or the environment is highly dynamic, SLAM may simply fail to produce consistent maps. Therefore, improving the performance and robustness of SLAM algorithms and their applications is still an important research topic [9], [10], [11].

[†] School of Informatics, University of Edinburgh, UK
[‡] Department of Computing, Imperial College London, UK
[⋆] School of Computer Science, University of Manchester, UK
[•] Electrical Engineering - Computer Systems, Stanford University, USA
[1]https://github.com/pamela-project/slambench2

In visual SLAM, various discrete paradigms exist for the algorithms. For instance, **'direct'** algorithms use all pixels' intensity values directly [12], while **'indirect'** methods extract features and process those [13]. Orthogonal to the direct/indirect paradigm is the reconstruction spectrum [14]. On one end of the spectrum are **'sparse'** methods, where only a subset of features are used for reconstruction [13]. These methods, also called feature-based, tend to use less resources in terms of power and system memory. Sparse methods limit the amount of information extracted from images. However, they may be useful in situations where a detailed map is not required. On the other end are **'dense'** methods where all pixels are used for reconstruction [15]. Additionally, in sparse method points are treated independently, while in dense methods, geometric dependencies are used. Semi-dense algorithm [16] belong to the middle of this spectrum. Within each category, various algorithms exist, depending on the details of the design. For example, in sparse visual SLAM, ORB-SLAM [17] uses ORB features [18] and OKVIS [19] relies on BRISK features [20]. This diversity makes evaluation and comparison difficult.

In SLAMBench2, we support several different algorithms. Amongst sparse algorithms, MonoSLAM [13], PTAM [7], OKVIS [19], and ORB-SLAM2 [17] are supported. The first two algorithms support only monocular systems, OKVIS supports only stereo and RGB-D, and ORBSLAM2 support all. These algorithms are indirect algorithms. Amongst dense methods, three algorithms are supported. The previous release of SLAMBench integrated an implementation of KinectFusion [12], a dense algorithm only using depth frames to produce its map. In SLAMBench2 in addition to KinectFusion, we also integrated ElasticFusion [15] and InfiniTAM [21], two recent dense systems with fundamentally different data structures and operators than KinectFusion. Finally LSD-SLAM, a semi-dense SLAM system using only one monocular camera, is also supported in SLAMBench2. These algorithms are summarised in Table II.

B. Computer Vision and Robotics Middleware

With new algorithms being proposed, it is clear that SLAM designers need to compare different algorithms efficiently and systematically. Additionally, with this large variety of algorithmic choices, it is a key concern for SLAM users how to compare different algorithms/parameters, and select the most appropriate. SLAMBench2 presents a platform that researchers in industry and academia can harness to efficiently and fairly compare different algorithms. Within the robotics and computer vision community, the closest related works to SLAMBench2 are various types of middleware such as Robot Operating System (ROS) [22], Gazebo [23], Orca [24], OpenRDK [25], YARP [26], MOOS [27], and many more. These systems have been designed to facilitate design, simulation, and integration of sensors, algorithms, and actuators. SLAMBench2 is not a robotic middleware, but rather it is a benchmarking framework with generic APIs concerned about SLAM and computer vision algorithms. SLAMBench2 aims to help designers and developers to design and compare their algorithms against the state-of-the-art with minimum overhead. In addition, this enables fair comparison between different hardware to be performed on many state-of-the-art SLAM algorithms.

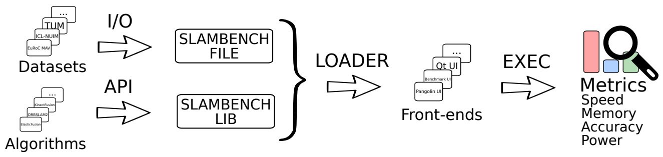

Fig. 1. Overview of the SLAMBench2 structure. This picture shows the four main components of the framework (I/O, API, Loader, and User Interface) as well as its three main characteristics (Dataset-agnostic, Plug and Play Algorithms, and Modular User Interfaces).

C. SLAM Benchmarks

Our work is closely related to KITTI Benchmark Suite [1] and TUM RGB-D benchmarking [28]. However we differentiate from these systems by expanding in two directions: algorithms and datasets. SLAMBench2 allows users to evaluate a new trajectory, real-world or synthetic, against existing algorithms available in the platform. Additionally, the diversity of the datasets allows researchers to evaluate new performance metrics such as reconstruction error and reconstruction completeness.

III. SLAMBench2

A. Overview

SLAMBench2 goes beyond previous frameworks [2] in that it is *dataset-agnostic*, supports *plug and play algorithms*, and provides *modular user interfaces*. See Fig. 1 for a summary.

Dataset-agnostic: It includes interfaces to use ICL-NUIM, TUM RGB-D, and EuRoC MAV (described in Sec. III-C.1), and can be extended to any dataset using compatible sensors. Datasets consisting of new sensors can also be supported by extending the SLAMBench2 library.

Plug and Play Algorithms: It provides a generic API to interface with SLAM algorithms and integrate new SLAM algorithms. There are currently eight different SLAM systems available (described in Sec. III-C.2). SLAMBench2 also works with closed source SLAM systems which can be distributed to the community without sharing source code. This feature is expected to raise the level of collaboration with industry in SLAM research.

Modular User Interface: SLAMBench2 also makes it easy to select between different user interface systems. Evaluation metrics can be changed and customised, as can the Graphical User Interface (GUI), while maintaining independence of the datasets and SLAM algorithms. For instance, the native visualiser, based on the Pangolin library [29], can easily be replaced with ROS visualiser [30]. Although this modularity is useful for advanced evaluations, we provide and recommend default user interfaces to maximise the reproducibility of experimental results.

B. Framework Components

The framework has four main components: **I/O System** (Sec. IV-A), **Integration API** (Sec. IV-B), **Loader** (Sec. IV-C) and **User Interfaces** (Sec. IV-C).

1) The I/O System translates existing datasets into SLAMBench2 datafiles which contain a description of the dataset sensors, frames, and ground-truth data when available (trajectory and 3D point cloud). For calibration, currently pinhole camera model with radial distortion is supported. The model includes two parameters for focal length and two parameters for optical center along x and y axes, and five parameters for lens distortion following the OpenCV format.
2) The API is used to interface to any SLAM algorithm implementation, and includes features to:
    - describe algorithmic parameters,
    - stream frames into the algorithm,
    - collect results (camera pose and reconstruction).
3) The Loader is then used to load the datafile and SLAM algorithm using the API. The loader executes these SLAM algorithms using the specified dataset.
4) Finally User Interfaces are used to dynamically analyse the algorithm; these can be command-line tools for accurate experiments, or GUI for demonstrations and debugging.

C. Supported Artefacts

1) Datasets: SLAMBench2 supports a number of existing datasets incorporating a range of different sensors and sensor configurations (listed in Table. I). Due to the presence or absence of certain types of sensor in each dataset, most of the datasets are compatible only with a subset of the supported algorithms.

2) SLAM Algorithms: There are many existing SLAM systems and it is likely that future systems will considerably improve on the current state-of-the-art. Therefore an important feature is the easy integration of new SLAM systems into the framework. SLAMBench2 already has eight different algorithms fully integrated, which can be used as examples for the integration of new algorithms (See Table II).

ICL-NUIM Dataset [3]		
Sensors: RGB-D Ground-truth: Trajectory, 3D Point Cloud		
Name	Duration	datafile size
Living Room 0	60.4s	2.8GB
Living Room 1	38.6s	1.8GB
Living Room 2	35.2s	1.7GB
Living Room 3	49.6s	2.3GB
TUM-RGDB Dataset [28]		
Sensors: RGB-D, IMU Ground-truth: Trajectory		
Name	Duration	datafile size
Freiburg1/xyz	30.1s	1.4GB
Freiburg1/rpy	27.7s	1.3GB
Freiburg1/360	28.7s	1.3GB
Freiburg1/floor	49.9s	2.2GB
Freiburg1/desk	23.4s	1.0GB
Freiburg1/desk2	24.9s	1.1GB
Freiburg1/room	48.9s	2.4GB
Freiburg2/xyz	122.7s	6.6GB
Freiburg2/rpy	110s	5.9GB
Freiburg2/360_hemisphere	91.8s	4.9GB
Freiburg2/360_kidnap	48s	2.5GB
Freiburg2/desk	99.4s	5.3GB
Freiburg2/desk_with_person	142.1s	7.3GB
Freiburg2/large_no_loop	112.4s	6.0GB
Freiburg2/ large_with_loop	173.2s	9.3GB
Freiburg2/pioneer_360	72.8s	2.1GB
Freiburg2/pioneer_slam	155.7s	5.2GB
Freiburg2/pioneer_slam2	115.6s	3.7GB
Freiburg2/pioneer_slam3	111.9s	4.5GB
EuRoC MAV Dataset [31]		
Sensors: Greyscale Stereo, IMU Ground-truth: Trajectory, 3D Point Cloud		
Name	Duration	datafile size
Machine Hall 1	184.1s	10.3GB
Machine Hall 2	152s	8.5GB
Machine Hall 3	135s	7.6GB
Machine Hall 4	101.6s	5.7GB
Machine Hall 5	113.6s	6.4GB
Vicon Room 1/1	145.6s	8.2GB
Vicon Room 1/2	85.5s	4.8GB
Vicon Room 1/3	107.5s	6.0GB
Vicon Room 2/1	114s	6.4GB
Vicon Room 2/2	117.4	6.6GB
Vicon Room 2/3	116.9s	6.0GB

TABLE I

COMPATIBLE DATASETS WITH THE SLAMBENCH2 FRAMEWORK. THE DATAFILE SIZE IS THE UNCOMPRESSED SLAMBENCH DATAFILE.

D. Performance Metrics

The original release of SLAMBench had a number of features designed around providing various metrics: frame rate (measured in frames per second), absolute trajectory error (ATE, measured in centimetres, explained in Section III-D.2), and power consumption (measured in Watts). SLAMBench2 provides these alongside several new metrics. We also extend some of the existing metrics to provide more information, and to be compatible with more experimental configurations (e.g. devices).

1) Computation Speed: Firstly, the computation speed metric provided in the original SLAMBench is preserved. For each algorithm, the total computation time per frame is available. It is also possible for an integrated implementation to provide more fine-grained timing information. For example, times for the front-end image processing, tracking, and integration are all available from KinectFusion. Many of the currently integrated algorithms provide these fine-grained metrics, which can be used to tune parameters or implementations in order to improve performance.

2) Algorithmic Accuracy: The accuracy metrics are determined by comparing estimates with ground-truth data. Absolute Trajectory Error (ATE) and Relative Pose Error (RPE) are used to measure trajectory accuracy. RPE is a measure for the drift of the trajectory over a fixed time interval. While RPE measure the local accuracy of the estimate trajectory, ATE measures the global consistency of the trajectory. For more information about ATE and RPE, refer to [28]. These trajectory metrics alongside with a mapping metric, Reconstruction Error (RER) [15], provide

Algorithm	Type	Sensors	Implementations	Large-scale	Loop-closure	Year
ORB-SLAM2 [32]	Sparse	RGB-D, Monocular, Stereo	C++	✓	✓	2016
MonoSLAM [13]	Sparse	Monocular	C++, OpenCL	✓		2003
OKVIS [19]	Sparse	Stereo, IMU	C++	✓		2015
PTAM [7]	Sparse	Monocular	C++	✓		2007
ElasticFusion [15]	Dense	RGB-D	CUDA	✓	✓	2015
InfiniTAM [21]	Dense	RGB-D	C++, OpenMP, CUDA	✓		2015
KinectFusion [12]	Dense	RGB-D	C++, OpenMP, OpenCL, CUDA			2011
LSD-SLAM [16]	Semi-Dense	Monocular	C++, PThread	✓	✓	2014

TABLE II

TABLE SUMMARISING THE ALGORITHMS CURRENTLY ADAPTED FOR USE IN SLAMBENCH2. WE MARK WITH A ✓WHEN THE FUNCTIONALITY IS INCLUDED. THE YEAR COLUMN SHOWS WHEN THE ALGORITHM APPEARED FOR THE FIRST TIME.

quantitative comparisons for various algorithms.

ATE and RPE are computed at runtime, with a minimal alignment between the first closest ground-truth pose and estimated pose (in terms of timestamp). Off-line, more complex alignment techniques can be used in order to compare dense and semi-dense techniques when the map scales do not match. RER is determined by running iterative closest point (ICP) algorithm on the point cloud models of the reconstruction and ground truth. As this is computationally intense, this evaluation is also performed off-line.

3) Power Consumption: In the original SLAMBench [2], the power consumption metric was relatively ad-hoc, and only supported a specific embedded system (ODROID-XU3). We extended the metric to support PAPI [33] (a generic power sensor API). In the future we hope to provide more extensive power consumption metrics. However, fine-grained power sensors are still uncommon in embedded development boards, and tend to have an ad-hoc or vendor-specific interface, which makes integration challenging.

4) Memory Usage: Finally we include memory consumption as a new metric. The dynamic memory consumption of an algorithm can be accurately measured at a per-frame level, and overall memory consumption can also be reported. We consider this to be an important contribution, as SLAM algorithms tend to consume a large amount of memory and memory consumption is often not considered as a goal or evaluated. For implementations that use GPU acceleration, we also provide methods to measure how much on-GPU memory is consumed. Due to the multiple GPU compute APIs and GPU vendors, these techniques are ad-hoc, but can be used to provide information on the relative memory use of each algorithm and dataset.

The next sections will detail the different components of SLAMBench2, these are the I/O system, the API, the loader and user interfaces.

IV. SLAMBENCH2 CORE COMPONENTS

A. I/O System

The I/O system is used to prepare datasets to be used as inputs to the SLAMBench2 framework. As mentioned above, existing implementations of SLAM algorithms tend to each use their own dataset format. This makes it very difficult to compare the performance of two algorithms on the same dataset, since the dataset would need to be made available in multiple different formats.

SLAMBench2 attempts to solve this problem by defining a simple unified format, and then providing tools to convert from the dataset formats of popular datasets into this format. Creating such conversion tools, to support new dataset formats, is straightforward.

The SLAMBench2 datafile is defined as follows:

```
DATAFILE = <HEADER><SENSORS><GT_FRAMES><IN_FRAMES>
HEADER   = <VERSION:4B><SENSOR_COUNT:4B>
SENSORS  = <SENSOR 1>...<SENSOR N>
SENSOR   = <TYPE:4B><PARAMETERS>
GT_FRAMES= <EMPTY>|<GT_FRAME 1>...<GT_FRAME N>
IN_FRAMES= <IN_FRAME 1>...<IN_FRAME N>
GT_FRAME = <TIMESTAMP:8B><GT_TYPE:4B><DATA>
IN_FRAME = <TIMESTAMP:8B><IN_TYPE:4B><DATA>
GT_TYPE  = POSE|POINT_CLOUD
IN_TYPE  = RGB_FRAME|DEPTH_FRAME|IMU_FRAME|...
```

The datafile contains a description of the sensors used within the dataset (`SENSORS`), followed by a (possibly empty) collection of 'ground-truth' frames ordered by timestamp (`GT_FRAMES`), followed by a collection of input frames ordered by timestamp (`IN_FRAMES`). The data files can contain multiple sensors of the same type. For example, multiple RGB camera sensors can be used to make up a stereoscopic sensor. A large variety of input frame types are supported, including: RGB, Greyscale Images, Depth Images, IMU Data, and Pixel Events.

In order to allow for measurements of trajectory and reconstruction error, the SLAMBench2 data format supports several 'ground-truth' frame types. This allows a data file to contain both the input and expected output. Two such frame types are currently supported: Ground-Truth Trajectory data and Reconstruction Point Cloud data.

Image sensors present several problems due to differing image formats, pixel formats, etc. We choose to address this by using a single uncompressed raster image format with a configurable pixel format. Although using an uncompressed format means that the data size is large, we do this to avoid the runtime costs involved in uncompressing the image data. Calibration settings of cameras are also integrated in the datafile, though those parameters can be changed at runtime if needed.

Reading and writing dataset files is performed via a library provided with SLAMBench2. Several programs which can be used to convert between ICL-NUIM, TUM RGB-D, and EuRoC MAV formats are provided as examples of how to produce these files. Data is then read out of the file one frame at a time, with a variety of helper classes provided.

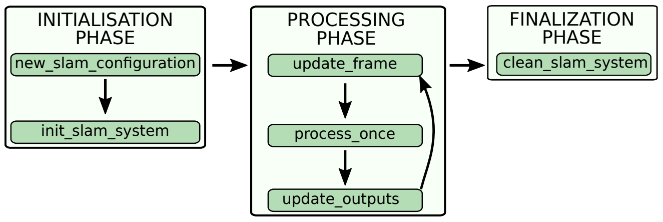

Fig. 2. Workflow of the SLAMBench API.

B. API

One of the most critical functions of the SLAMBench2 infrastructure is to actually interface with implementations of SLAM algorithms. Implementations of SLAM algorithms are often released as standalone executables (rather than as libraries), and often do not conform to any standard API or structure. A possible approach would be to use a middleware such as ROS [22], but there is no clear standard defined to declare SLAM systems in this context and using such an infrastructure may add more constraints on the system to be used. In contrast, SLAMBench2 provides a standardised API which is used as an interface between the SLAM algorithm implementation and other components of the SLAMBench2 infrastructure. This API is used to initialise the data structures used by the implementation, send input data frames into the implementation, actually execute the algorithm, and to extract the outputs, i.e. trajectory and reconstruction.

SLAM algorithms are typically highly configurable, so that they can be adapted to different operating conditions and to different host systems, e.g. to gracefully degrade accuracy but improve speed in order to run on a less powerful device, or to gracefully degrade speed but improve power consumption for a power-constrained device. These parameters can be exposed through the SLAMBench2 API in order to allow user interfaces to manipulate these parameters. This also means that SLAMBench2 can be used in combination with auto-tuning techniques, this point will be the subject of one use-case in Sec. V-B.2.

The SLAMBench2 API is made up of the following functions:

```
bool sb_new_slam_configuration(SBConfig*);
bool sb_init_slam_system(SBConfig*);
bool sb_update_frame(SBConfig*, SBFrame*);
bool sb_process_once(SBConfig*);
bool sb_update_outputs(SBConfig*);
bool sb_clean_slam_system();
```

This API is used by the framework through a 3-phase workflow as shown in Fig. 2. In the rest of this section we will describe this workflow.

1) Initialisation Phase: The first functions of this API are used to initialize the SLAM system.

- `sb_new_slam_configuration`: The `SBConfig` object is used to share environment information between the framework and the SLAM algorithm. The algorithm is also able to use this object to declare parameters. Later, this information enables the GUI to show these parameters, to change them on-line, or to perform auto-tuning, which is optimising parameters value using an automatic mechanism for accuracy/performance trade-offs (e.g. [34], [5]), or for active SLAM (e.g. [35]).
- `sb_init_slam_system`: This function controls the start of memory allocation and SLAM system initialisation. This time the `SBConfig` object contains user-defined values for the algorithm parameters as well as a description of the current available sensors.

2) Processing Phase: For every frame, the `sb_update_frame` function is called to share frame data with the SLAM algorithm. When the algorithm is capable of doing meaningful computation (where the data received are sufficient), then SLAMBench2 will call the `sb_process_once` function.
After calling `sb_process_once`, SLAMBench2 calls `sb_update_outputs` to get the current pose estimation, map estimation and tracking status from the SLAM system. Extraction of the map can be complex given the wide range of possible formats, this is discussed later in Sec. IV-B.4.

3) Finalisation Phase: Eventually when the user interface is closed or when the end of data is reached, SLAMBench2 will call the `sb_clean_slam_system` function. In this function the SLAM system is expected to release any memory buffers previously allocated.

4) Graphical Rendering and Map Extraction: As there are a large number of different data structures used to internally represent a map, we did not define a specific format to extract the map. Instead we defined an `OutputManager`, used to store any information from the SLAM system which could potentially be processed or visualised. This includes trajectories and maps as much as visual information for debugging purposes. When SLAMBench2 calls `sb_init_slam_system`, the SLAM system is made aware that the UI system is being used, and so memory allocation or other initialisation specific to the UI should be performed. Then, after every call of `sb_process_once`, SLAMBench2 also calls the `sb_update_outputs` function which allows the SLAM system to update the information displayed on the user interface. If the data structure used by the SLAM system is new, there is a way to include this new format. For now, SLAMBench2 already defines several types such as a 3D point cloud, a list of visible features, and RGB frames.

C. Loader and User Interfaces

The Loader connects the user interface with algorithms. The Loader is a critical part of the infrastructure in the context of collaboration with industry, since it allows SLAM libraries to be dynamically loaded without source code being made available. This means that commercially-developed SLAM algorithms can be integrated into SLAMBench2 and compared against other commercial or open source algorithm implementations in a consistent and reproducible manner.

The user interfaces display the outputs and metrics of the algorithms, such as output trajectories, accuracy, reconstruction, etc. Multiple interfaces are available, with two

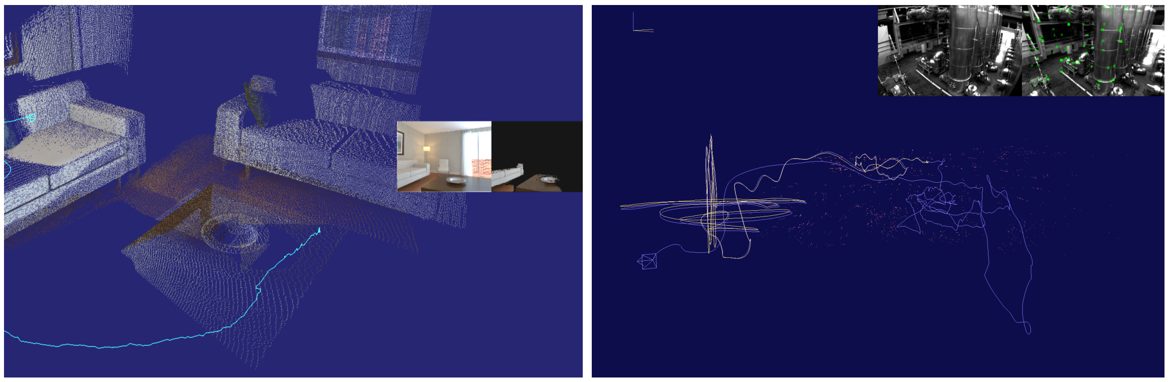

Fig. 3. Cropped screenshots of the GUI of SLAMBench2 that uses the Pangolin library [29]. On the left a point cloud extracted from ElasticFusion is shown, alongside the trajectory. The input frame and internal 3D rendering of ElasticFusion are also shown, inlaid on the right of the image. On the right a similar interface used with ORBSLAM2 and the EuRoC MAV dataset. This time the map is sparse, we can see the trajectory aligned with the ground-truth.

of the main interfaces being a GUI which can be used to view and compare trajectories and reconstructions between different algorithms and ground-truths, and a 'benchmark' interface, which has a text-only interface and outputs the desired metrics (such as timing or trajectory information). Figure 3 shows the existing GUI.

V. Experiments

This section demonstrates how SLAMBench2 can be used for a range of use cases. Here. we evaluate memory size, accuracy, speed and power consumption for different algorithms, datasets, and devices. Each of these results can be reproduced with a few command lines.

A. Experimental Setup

For our experiments we used three platforms, we will refer to these machines as ODROID, TK1, and GTX1080.

ODROID: A Hardkernel ODROID-XU3 based on the Samsung Exynos 5422 SoC, featuring an ARM big.LITTLE heterogeneous multiprocessing (HMP) unit with four Cortex-A15 "big" out-of-order cores, four Cortex-A7 "LITTLE" in-order cores, and a 6-core Mali-T628-MP6 GPU.

TK1: A Jetson TK1, NVIDIA embedded Linux development platform featuring a Tegra K1 SoC. It is provided with a quad-core ARM Cortex-A15 (NVIDIA 4-Plus-1), with an on-die GPU composed of 192 CUDA cores.

GTX1080: At the desktop end of the spectrum we selected a machine featuring an Intel i7-6700K CPU and an NVIDIA GTX 1080 GPU containing 2560 CUDA cores.

B. Experimental Evaluation

1) Memory-Constrained Device: While memory consumption of SLAM systems is not typically considered, there are areas where available memory is highly constrained. For example, Augmented Reality applications that run on mobile phones often have only 4GB of RAM available, which must be shared with the operating system and other apps. We therefore evaluated a variety of SLAM algorithms and measured how memory usage varies throughout the dataset. For edge computing platforms the memory constraint is even more important allowing only few hundred of Megabytes of RAM. By introducing this performance metric in the design space SLAMBench2 pushes the envelope of deployability benchmarking of SLAM systems.

We evaluated five SLAM algorithms: ElasticFusion, InfiniTAM, KinectFusion, LSD-SLAM, and ORB-SLAM2, over trajectory 2 of the ICL-NUIM dataset. This trajectory was selected as it successfully tracked on each dataset. The results are shown in Fig. 4.

These results show that there are two classes of algorithms depending on whether or not memory is allocated during processing. For example, KinectFusion allocates a fixed-size volume used for mapping as well as for tracking (through raycast). This explains why memory usage is fixed over time. However, if the camera goes beyond this fixed sized volume, the algorithm will lose track. In contrast, an algorithm such as LSD-SLAM will continuously build up its map of key frames and so avoid this limitation, but its memory usage will grow as more key frames are discovered. A more recent algorithm such as ORB-SLAM2 prunes its list of key frames in order to reduce memory consumption. As the results show, this pruning severely reduce memory growth over time.

2) Speed/Accuracy Trade-off: One well known use case of SLAM is in UAV navigation. Consumer-grade UAVs are increasingly operated indoors or at low altitudes, and so must navigate through complex spaces and avoid obstacles. This means that highly accurate tracking is required. The speed of the device also means that the system must be responsive, and hence have a high frame-rate. Having a high accuracy and high frame-rate is challenging and in this experiment we explore the trade-off between accuracy and frame-rate for different versions of the same algorithm.

As our API exposes algorithmic parameters, it is possible to explore the parameter space using smart search algorithms (e.g. HyperMapper [34], [5], OpenTuner [36]). Figure 5 shows the result of exploring the parameter space for the KinectFusion algorithm to find accuracy/speed Pareto optimal on the ODROID embedded platform (which could be mounted on a small drone).

The parameters were "Volume resolution", "μ distance", "Pyramid level iterations", "Compute size ratio", "Track-

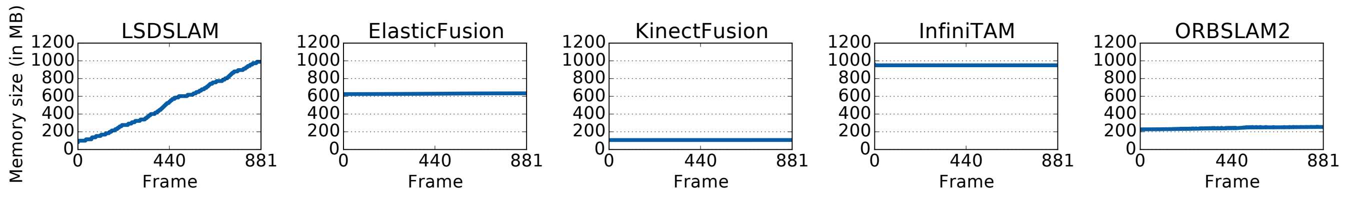

Fig. 4. Memory usage of different SLAM algorithms on the trajectory 2 of the ICL-NUIM dataset.

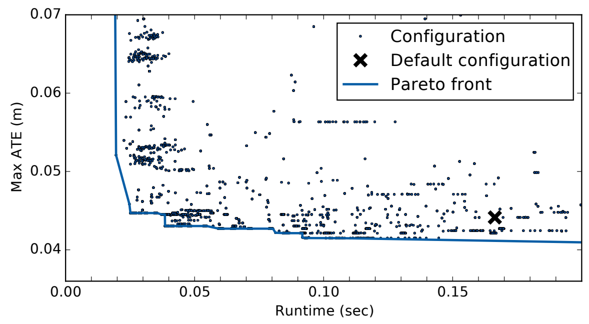

Fig. 5. Accuracy/Speed Trade-off exploration performed for KinectFusion (OpenCL) on the ODROID board.

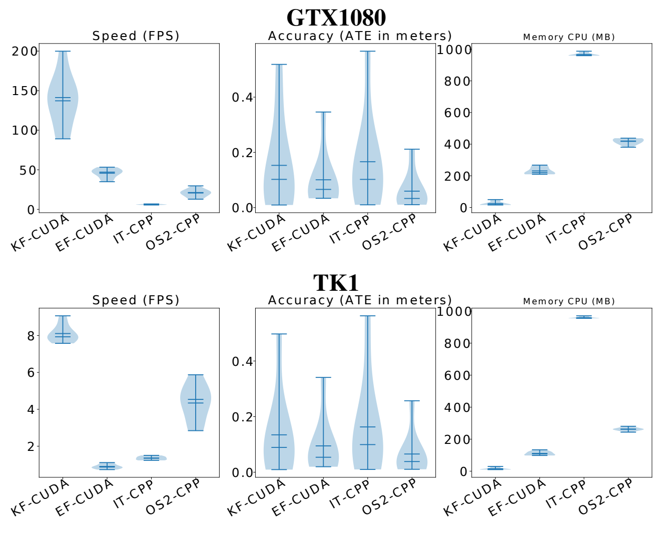

Fig. 6. Comparison of four algorithms run on TK1 and GTX1080, with eight different datasets and over three different metrics. KF: KinectFusion (CUDA), EF: ElasticFusion (CUDA), IT: InfiniTAM (C++), OS2: ORB-SLAM2 (C++). The datasets used were ICL-NUIM (LV0, LV1, LV2, LV3) and TUM (fr1_xyz,fr1_rpy,fr2_xyz,fr2_rpy).

ing rate", "ICP threshold", and "Integration rate" [2]. The original performance is denoted by the X. By exploring the parameter space, we can see that there are many other parameter settings that lead to either faster or more accurate results. In this figure it is clear that by only using different parameters, there is a potential speed-up of 8x compared to the original configuration. SLAMBench2 allows us to generalise such an experiment over a wider range of datasets and algorithms.

3) Productivity Cost of Introducing a New Algorithm: We evaluate the integration of an algorithm in the SLAMBench2 framework. Integrating into the UI allows both the use of all supported datasets, and the fair and direct comparison of the algorithm against other integrated algorithms. Integrating a new algorithm requires implementing a library compatible with the SLAMBench2 API. For example, we need to implement the `sb_new_slam_configuration` function, which declares the algorithm parameters:

```
bool sb_new_slam_configuration(SBConfig * sbc) {
  sbc->addParameter(TypedParameter<int>(
  "mf", "max-features",
  "Maximum number of features",
  &max_features, &default_mf));
  ....
```

For ORB-SLAM2, the required functions can be implemented using around 350 lines of code. This is certainly a straightforward task for the authors of the algorithm, and a simple task for a person familiar with SLAM algorithms. Once compiled, this library (i.e. liborbslam2.so) can be directly used with the SLAMBench2 loader:

```
> sb_loader -i tum_F2_xyz.slam -load liborbslam2.so
frame ...  orbslam2_ATE
...   ...  ...
3665  ...  0.0088654254
```

4) Multi-objective Comparison of Multiple SLAM Systems: Fig. 6 shows the result of evaluating ORB-SLAM2, KinectFusion, ElasticFusion, and InfiniTAM on the TK1 and GTX1080 platforms for eight datasets and for three metrics.

The top three diagrams are for the GTX1080 platform, the bottom three are for the TK1. Each algorithm is labelled along the x-axis and each metric is measured along the y-axis. The violin of results shows the behaviour of each algorithm over the full data set.

These violin plots show the variation of metrics over the different datasets. For example, the memory consumption of KinectFusion and InfiniTAM are static and thus do not change over the data, while the memory consumption of ORB-SLAM2 is variable even though it is quite reasonably kept in the limited range of 300 MB. Similarly, the accuracy (Mean ATE) of KinectFusion and InfiniTAM can vary over the datasets from 1 cm up to 60 cm while ElasticFusion seems to be more accurate in general with no more than 30 cm drift. On the other hand, ElasticFusion is around 50% slower than KinectFusion.

In this context, ORB-SLAM2 seems to be a good trade-off, keeping its mean ATE down to 30 cm, while being only twice as slow as KinectFusion on the TK1. This is significant as ORB-SLAM2 does not use the GPU.

VI. Conclusions

SLAMBench2 provides a significant advantage in terms of evaluating existing SLAM systems and comparing new systems with the state-of-the-art. The SLAMBench2 methodology has been demonstrated across a wide range of algorithms, datasets, and devices. Useful insights can be gained.

We hope that the authors of future SLAM algorithm implementations will find SLAMBench2 to be useful for evaluation, and that they will make their released implementations SLAMBench2-compatible. So far eight of the most relevant SLAM algorithms are integrated into SLAMBench2, and we hope that this number will increase in the future. There also are still a large number of datasets that have not been integrated (such as New College [37], SFU Montain [38] and Stereo KITTI [39]). We believe that these datasets will soon be supported, in order to provide reproducible and consistent experimental results when using these artefacts.

We hope to eventually provide something similar to the KITTI benchmarking platform, where SLAM implementations could be uploaded to a cloud service and the SLAMBench2 metrics provided automatically, in order to avoid the effort that is usually required to obtain such results.

In the future, SLAMBench2 could also be extended to allow the integration of more realistic and application-specific evaluation metrics. For example, certain classes of algorithm, such as 3D object recognition and scene segmentation, could provide an interesting environment to evaluate the quality of 3D maps generated by SLAM algorithms.

Acknowledgement

This research is supported by Engineering and Physical Sciences Research Council (EPSRC), grant reference EP/K008730/1.

References

[1] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[2] L. Nardi *et al.*, "Introducing SLAMBench, a performance and accuracy benchmarking methodology for SLAM," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, May 2015.

[3] A. Handa *et al.*, "A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM," in *ICRA*, 2014.

[4] M. Z. Zia *et al.*, "Comparative design space exploration of dense and semi-dense slam," in *ICRA*, 2016, pp. 1292–1299.

[5] L. Nardi *et al.*, "Algorithmic performance-accuracy trade-off in 3d vision applications using hypermapper," in *Intl. Workshop on Automatic Performance Tuning (iWAPT)*. IEEE, 2017.

[6] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA, USA: The MIT press, 2005.

[7] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE, 2007, pp. 225–234.

[8] U. Frese, "Interview: Is SLAM solved?" *KI - Kunstliche Intelligenz*, vol. 24, no. 3, pp. 255–257, 2010.

[9] C. Cadena *et al.*, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[10] S. Saeedi *et al.*, "Multiple-robot simultaneous localization and mapping: A review," *Journal of Field Robotics*, vol. 33, no. 1, 2016.

[11] E. Vespa *et al.*, "Efficient octree-based volumetric slam supporting signed-distance and occupancy mapping," *IEEE RA-L*, 2018.

[12] R. A. Newcombe *et al.*, "KinectFusion: Real-time dense surface mapping and tracking," in *International Symposium on Mixed and augmented reality (ISMAR)*. IEEE, 2011, pp. 127–136.

[13] A. J. Davison *et al.*, "MonoSLAM: Real-time single camera SLAM," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 6, pp. 1052–1067, 2007.

[14] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 3, pp. 611–625, 2018.

[15] T. Whelan *et al.*, "ElasticFusion: Dense SLAM without a pose graph," in *Robotics: Science and Systems (RSS)*, 2015.

[16] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-scale direct monocular SLAM," in *European Conference on Computer Vision*. Springer, 2014, pp. 834–849.

[17] R. Mur-Artal, J. M. M. Montiel, and J. D. Tards, "ORB-SLAM: A versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[18] E. Rublee *et al.*, "ORB: An efficient alternative to SIFT or SURF," in *International Conference on Computer Vision*, 2011, pp. 2564–2571.

[19] S. Leutenegger *et al.*, "Keyframe-based visual–inertial odometry using nonlinear optimization," *The International Journal of Robotics Research*, vol. 34, no. 3, pp. 314–334, 2015.

[20] S. Leutenegger, M. Chli, and R. Y. Siegwart, "BRISK: Binary robust invariant scalable keypoints," in *2011 International Conference on Computer Vision*, 2011, pp. 2548–2555.

[21] O. Kahler *et al.*, "Very High Frame Rate Volumetric Integration of Depth Images on Mobile Device," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 11, 2015.

[22] M. Quigley *et al.*, "ROS: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[23] J. Hsu and N. Koenig, "The gazebo simulator as a development tool in ros," in *ROSCon*, 2012.

[24] "Orca: Components for robotics," http://orca-robotics.sourceforge.net/.

[25] D. Calisi *et al.*, "OpenRDK: A modular framework for robotic software development," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept 2008, pp. 1872–1877.

[26] G. Metta, P. Fitzpatrick, and L. Natale, "YARP: Yet another robot platform," *International Journal of Advanced Robotic Systems*, vol. 3, no. 1, p. 8, 2006.

[27] P. M. Newman, "MOOS-mission orientated operating suite," 2008.

[28] J. Sturm *et al.*, "A benchmark for the evaluation of rgb-d slam systems," in *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[29] "Pangolin," https://github.com/stevenlovegrove/Pangolin.

[30] "Ros vizualizer," http://wiki.ros.org/rviz.

[31] M. Burri *et al.*, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, 2016.

[32] R. Mur-Artal and J. D. Tards, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, Oct 2017.

[33] P. J. Mucci *et al.*, "PAPI: A portable interface to hardware performance counters," in *In Proceedings of the Department of Defense HPCMP Users Group Conference*, 1999, pp. 7–10.

[34] B. Bodin *et al.*, "Integrating algorithmic parameters into benchmarking and design space exploration in 3d scene understanding," in *International Conference on Parallel Architectures and Compilation*, 2016.

[35] S. Saeedi *et al.*, "Application-oriented design space exploration for slam algorithms," in *Int. Conf. on Robotics and Automation*, 2017.

[36] J. Ansel *et al.*, "Opentuner: An extensible framework for program autotuning," in *PACT*, 2014.

[37] M. Smith *et al.*, "The new college vision and laser data set," *International Journal of Robotics Research*, vol. 28, pp. 595–599, 2009.

[38] J. Bruce, J. Wawerla, and R. Vaughan, "The SFU mountain dataset: Semi-structured woodland trails under changing environmental conditions," in *Int. Conf. on Robotics and Automation, Workshop on Visual Place Recognition in Changing Environments*, 2015.

[39] M. Menze and A. Geiger, "Object scene flow for autonomous vehicles," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.