YaleNUSCollege

Real-time Filtering for Multi-sensory SLAM Benchmarking (Report 1)

Jianglong Liao

Capstone Final Report for BSc (Honours) in

Mathematical, Computational and Statistical Sciences

Supervised by: Dr. Bruno Bodin

AY 2019/2020

Contents

Chapter 1

Introduction and Motivation

The simultaneous localization and mapping (SLAM) problem questions the autonomy of a mobile robot in an unfamiliarised environment: would a robot be able to explore and incrementally construct the information of its surroundings, yet at the same time accurately locate its position in the complex, realistic environment? Solutions to the SLAM problem would imply that a robot has become "truly autonomous" as it no longer requires arbitral instructions or fixed input in order to understand its relation with its situated context [1]. Over the decades, multiple solutions have been proposed in different technical and mathematical forms, and have been applied to various fields of robots, including group aerial, underwater, indoor, etc. From a theoretical perspective, SLAM is deemed as a solved problem [1].

However, in a realistic environment, different solutions will showcase their advantages and disadvantages. For example, algorithms such as iterative closet point (ICP) perform exceptionally well in an indoor context, but when it comes to complex, outdoor environment, they may fail to reach the expectation for accuracy [2]. Moreover, substantial issue remains in implementing these solutions. Problems such as the trade-off

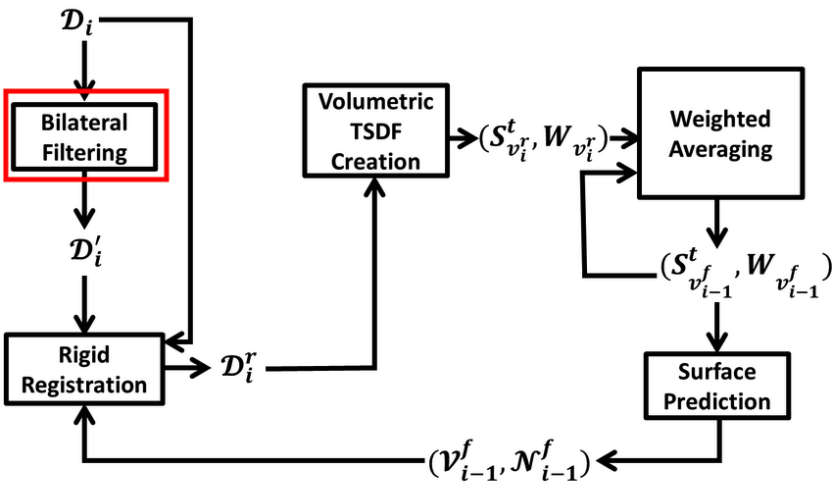
between accuracy, speed, memory and power consumption continue to challenge the researchers and practical commercialization. Hence, there has been sustained research in practical implementation and evaluation of different SLAM solutions.

In recent year, the creation of new sensors and increase in processing power have enabled the releases of newer, more accurate and efficient SLAM systems. As SLAM systems continue to evolve and develop, needs for evaluating diverse SLAM algorithms surge exponentially. Unfortunately, despite heated discussion over SLAM, there has been little attention devoted to standardizing the evaluation of all SLAM systems. Different approaches have been used by researchers to test and experiment the SLAM algorithms, making it difficult to compare the performance of various SLAM systems. Furthermore, most of the testing and evaluation can take considerable effort and have been done in a specific, manual procedure. Researchers need a plug and play system that can easily evaluate the performance different SLAM systems and streamline the testing process.

Hence, SLAMBench has been created by numerous SLAM researchers to streamline and standardize the benchmarking process of SLAM systems. The program is structured to be dataset-agnostic and SLAM-system-agnostic, so that researchers can conveniently input their own SLAM model and testing data. In addition, by having a standard SLAM benchmarking program, researchers would also be able to rest assured that the quantitative testing results are reproducible [3]. The program now is developed in multiple languages and supports eight state-of-the-art SLAM systems.

Yet, an essential component is missing in the current SLAMBenck software architecture: a real-time filter. While undoubtedly extracting as much information from sensors as possible would render the most accurate solution, fixed computational bounds and energy consuming limitation of a mobile robot will restrict the quantity and quality of sequential inputs that an algorithm can ingest [4]. Therefore, before feeding the sensor data into the correspondence SLAM system, real-time filtering is an indispensable step to improve the computational speed and lower the energy consumption.

FIGURE 1.1: Example of filtering in KinectFusion [5]



Currently, the SLAMBench framework consists of two configurable parts – dataset and SLAM algorithms. To test the performance of filtering and other preprocessing method, researchers are forced to manually customized their filter with deliberate calibration, for which these tinkering approaches will defeat the purpose of SLAMBench and complicate the benchmarking process. In addition, current SLAMBench framework only allows one data pipeline to deliver sensor data sequentially from the

I/O system, where data are ingested, to the loader, where the SLAM algorithms are evaluated. This simplistic design of data pipeline does not account for concurrent data delivery from multiple sensors and does not provide room for filtering and preprocessing multi-sensory data based on sequential time frames. Hence, restructuring the software architecture is required to decouple and modularize the new filtering module and ensure its compatibility with the rest parts of SLAMBench.

In this capstone project I introduce real-time filtering system for the current SLAMBench program. This filtering system enables researchers to evaluate the impact of various filtering and preprocessing techniques on different open source or proprietary SLAM systems. It is a tool that ensures the reproducibility of results for existing filtering methods and allows the integration and evaluation of new filters for SLAM algorithms. Through a series of testing and experimentations, we demonstrate its configurability, extensibility and its ease of use to benchmark different filters for SLAM systems. In summary, we present the following contributions:

- A publicly available filtering system, integrated with SLAMBench, that supports quantitative, reproducible comparison of different filtering techniques for SLAM systems.
- A filtering system that is configurable, extensible and enables plug and play of new filters.
- Various experimenting cases of different filters (resize, blur, random drop, etc.) on state-of-art SLAM systems.

Chapter 2

Background and Related Work

This chapter serves to introduce key and essential concepts of robotics system benchmarking and SLAM systems. Prior knowledge of SLAM and relevant technical understanding is not presupposed for reading this chapter, so the internal workings of specific SLAM systems, including the applied mathematics and codes, will not be elaborated. Rather, we introduce modular conceptualization of robotics system benchmarking and SLAM to assist the reader in understanding the structure of our filtering system. In-depth knowledge of SLAM systems is not strictly required, but we reference some of them in evaluating the results of our testing and experiments.

2.1 Robotics System Benchmarking

2.1.1 Background

Collaborative operations between man and machine has not only improve productivity, but also in some cases provide solutions not yet available to humans alone (disaster search and rescue robots) [6]. Co-development

between engineering and computer sciences have started to lay the foundations for mechanical hardware and software systems of generic and domain-specific robots. Yet without standardization, developing robotic technologies becomes not only ineffective, but also unprofessional and dangerous. For example, with proper standardization of video medium, there was a wide confusion and frustration of choosing HD DVD or Blue Ray for home-video appliance among consumers. On the other hand, successful standardization would bring synergies among different fields and cohesion between various research projects. For instance, the immense progress in the wireless communication technology has largely attribute to the establishment of widely accepted industrial standards such as IEEE 802.11. [6].

2.1.2 Methodology

Objective performance evaluation is required for each field of robotic systems to continue the progress of research and to facilitate the acceptance of robotic technologies. Yet it is not a trivial task to ensure that the evaluation is repeatable, unambiguous and holistic. Specifically, in the field of robot navigation and mapping, distinct research projects may take different approaches to measure the accuracy of mapping [7] [8]. However, even if a standard for accuracy comparison do exists, this benchmark is not enough to characterise the mapping system as a whole: there are still considerations such as energy consumption and performance. Recently, there has been initiative, such as RoboBench, that intends to construct a framework that allows researchers to holistically benchmark all aspects of robotic systems and form a set of comparable, reproducible measure.

Hence, when building a benchmarking framework for robots, systematic approach needs to be taken to account the cost of computing over the task, the performance of robotics algorithms, as well as the costs incurred in infrastructure such as energy and networks [9].

To ensure the completeness of Robotic system benchmarking, two aspects of benchmark categories need to be considered: the method that determines how we benchmark, and the focus that determines what we benchmark [10]. In general, there are two methods of benchmarking: analytical approach that concerns with evaluation of a robotics system on its own, and functional approach that test how a system solves a specific problem. Similarly, in terms of the focus, there are also two targets of benchmarking: component – evaluation on one specific part of the system, and system – evaluation on the overall system. By combining these criteria, we are able to form a matrix of our robotics benchmark system:

TABLE 2.1: Robotics benchmark matrix.

Analytical	1	3
Functional	2	4
	Component (filter)	System (SLAM system)

2.1.3 Application

In the case of our filtering system in the SLAMBench, to apply the framework of holistic evaluation, we identify four potential areas of benchmark that we intend to explore based on Table 2.1.

1. **Use the same dataset and apply different filters.**
   Since we already understand the task of each filter, this process merely allows us to observe whether data set has been correctly filtered.

2. **Use different datasets and apply the same filter.**
   Targeting problem: Supposed task of a filter
   Benchmark purpose: what is the performance of a filter at carrying out its filtering tasks.

3. **Use the same filter and apply to different SLAM systems.**
   Benchmark purpose: observe how the same filtering technique would impact different SLAM systems.

4. **Use different filters and apply to the same SLAM systems.**
   Targeting problem: SLAM benchmarking outcome (accuracy, computational speed, energy consumption)
   Benchmark purpose: how does different filters affect a specific SLAM system? What is the optimal filtering technique for a SLAM system?

These benchmarking methods are designed to be holistic to test the filtering system and its correlation with the SLAM systems. In addition, to

ensure that the benchmarking results are repeatable, only minimal configuration should be required throughout the system. Therefore, modularization of each part of the system, where every module functions independently from each other, is the key to protect the integrity of the whole benchmarking system. Further details with regards to benchmarking and system design are included in the following chapters.

2.1.4 Extension: Reinforcement Learning

Beyond the purpose for standardizing industrial performance for robotic tasks, what makes benchmarking ever more essential and exciting is the emergence of reinforcement learning. Similar to the SLAM problem and numerous other continuous control robotic tasks, in the paradigm of reinforcement learning, a robot is not told which action to take, but instead "must discover which actions yield the most reward by trying them out" [11]. By doing so, the robot will need to maximize a numerical reward signal. In the case the SLAM problem, such reward signal could be accuracy of trajectory, power consumption, speed or combined metrics. However, reinforcement learning does not stop at the end of benchmarking. The result of the benchmark will then be fed back to the computational loop to further optimize the efficiency of robotic actions.

This evolutionary learning method seems promising as it may finally address some of the robotic problems that have been solved theoretically, but still lack real-world applicability. However, researchers quickly realize the problem of applying reinforcement learning to robotic tasks: **lack of benchmark tasks and supporting tools** [12]. For more advanced tasks beyond SLAM such as path planning, robots not only needs to map and

localize itself, but also need to reach a certain goal through optimum path [13]. Robots need to actively make decisions under different scenarios and what assists them to improve their decision-making capability (in other words, fine-tuning their hyper-parameter configurations) is their benchmark result of previous actions.

In addition, some robotic tasks may be resource and time consuming. In order to expedite reinforcement learning process, simulation of scenarios, where robotic actions are reproduced and tested for thousands of times, is required. Therefore, there have been collective efforts to gather all sampling-based (instead of modelling-based) algorithms for reproducible simulations [14]. And SLAMBench, as part of the initiative of standardizing SLAM benchmarking criteria, procedure, and repeatable simulation, is laying foundation to enable other advanced research, such as reinforcement learning to further improve the state-of-art of robotic tasks in the real world.

2.2 Simultaneous Localization and Mapping

2.2.1 SLAM Background

As mentioned in the Background section, we provide a conceptual state-of-art description of the Simultaneous Localization and Mapping (SLAM) system. The system, as the name suggests, consists of two inseparable parts: mapping and localization. Mapping refers to the process of incrementally building a map representation of an environment, while localization denotes the method to locate the robot in the current map in order to minimize the error [15]. SLAM can be implemented by combination of

hardware and software, including laser, monocular vision, and visual-inertial SLAM system, which leverages on RGB-D camera and inertial measurement unit (IMU).

For now, whether SLAM is solved is still hard to answer in reality [16], since under the SLAM domain, there are still various subdomains that are categorized by combining specific type of robot, operational environment and performance requirements. Some, such as deterministic navigation environment and highly robust laser robot, can largely considered solved [17]. On the other hand, for a visual-inertial SLAM system, the accuracy of mapping still varies dramatically with regards to the motion of robot and environment, including fast movement and highly dynamic and challenging environment. Some small alternation in the navigation environment can still easily induce failure and largely drop the accuracy of mapping. In addition, performance of a SLAM system is also directly determined by the computational bound and system requirement, in which case the evaluation of a SLAM system goes beyond theoretical mathematical investigation and aims to provide practical systematic optimization.

2.2.2 SLAM Benchmarking

Therefore, benchmarking a SLAM system now requires a renewed set of requirements to fulfill the current research demand and development of SLAM [18]. First, SLAM system needs to be tested on the robustness of performance. To what extent can a SLAM system operate with a large variety of dataset, yet maintain low failure rate for a long period of time? This benchmarking requirement is essential for generic SLAM system

that would be applied to different scenarios and conditions of robots. In this regard, SLAMBench provides interfaces that allow researchers to test a SLAM system with different standardized datasets. Second, SLAM system needs to benchmarked on a high-level understanding. Basic geometry reconstruction is no longer the only measure to represent the robot's understanding of the environment. High-level geometry and semantics now become the main objective measure for accuracy of mapping. To address this requirement, SLAMBench runs iterative closest point (ICP) algorithm on the point cloud models to gain a high-level understanding of differences between reconstruction and groud truth. Third, SLAM system will need to be evaluated on its efficient usage of limited resources. These resources include type and number of sensors, computational power, memory and energy storage. Beyond algorithmic accuracy, SLAMBench provides performance metrics that consists of computation speed, power consumption and memory usage to evaluate SLAM system's performance with limited resources.

Finally, a SLAM system should be tested on specific task-driven functionality. This test will evaluate the capability of a SLAM system to process the relevant information and filter out inessential sensor data before feeding these data through core algorithms. For this task, a decoupled filtering system is required to allow real-time prepossessing of sensor data from multiple channels. The filtering system could be utilized for the following purposes.

- **Expose the advantage and disadvantage of certain SLAM.** In visual SLAM, there are "direct" algorithms that directly use all intensity values [19], and "indirect" methods that first extract features

before ingestion [20]. On the other hand, there are "sparse" method that only uses a subset of features [19], and "dense" method that ingest all pixel values [21]. By using a filtering system, characteristics of a dataset (brightness, sharpness, etc.) could be extended to amplify strengths and weaknesses of all methods;

- **Explore methods to optimize SLAM systems.** The filtering system allows real-time preprocessing of dataset to test certain parameters and improve the robustness of SLAM system when applied to different situations. Moreover, applying a specific filter has the potential to lessen power consumption, memory storage, and at the same time, improve computational speed.

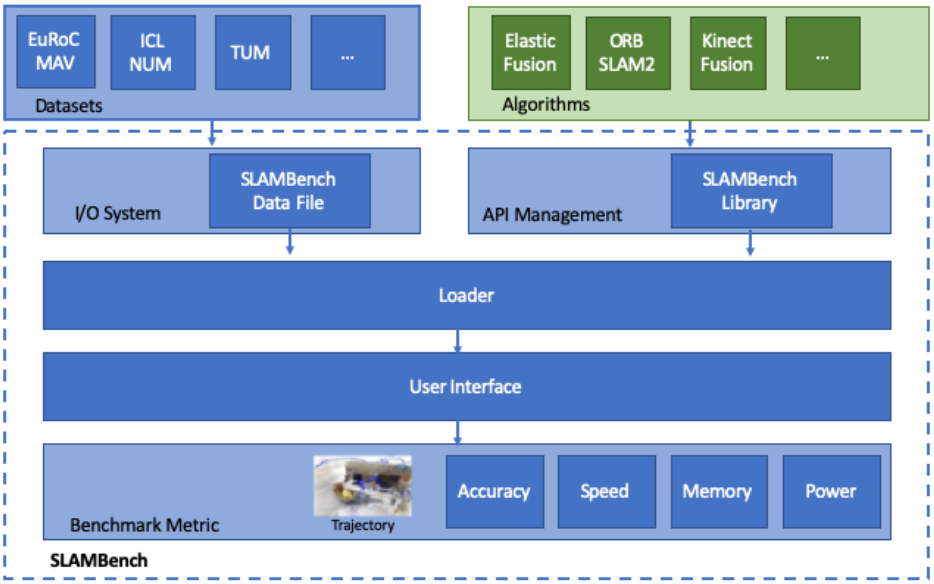
However, this filtering system is not present in the SLAMBench framework. Currently, there are other general-purpose SLAM benchmarking tools such as KITTI Benchmark Suite [22] and TUM RGB-D benchmarking [7]. Yet compared to SLAMBench, these benchmark tools do not allow flexible integration of real-world datasets. In addition, these systems do not entirely address the SLAM benchmarking requirements as mentioned above. Hence, an addition of filtering system to the SLAMBench framework makes the streamlined benchmarking procedure even more comprehensive, and at the same time, maintains the flexibility and scalability of SLAMBench.

2.3 SLAMBench Architecture and Evaluation

2.3.1 Overview of Core Components

The main framework of SLAMBench consists of four main components: I/O (data ingesting system), API (library integration), Loader (middleware), and User Interface (visualization). The architectural components and their relations are shown in the following figure 2.1.

FIGURE 2.1: SLAMBench Design Architecture [3]



I/O – Data Ingestion System

If the SLAMBench directly ingest datasets from various sources that may have different data formats and file types, it would be hard to allow different algorithms to perform benchmarking on the same dataset. Without multiple data formats available for the dataset, comparison between different algorithms would not be possible. Hence, I/O System translates different datasets into uniformly formatted SLAMBench datafiles.

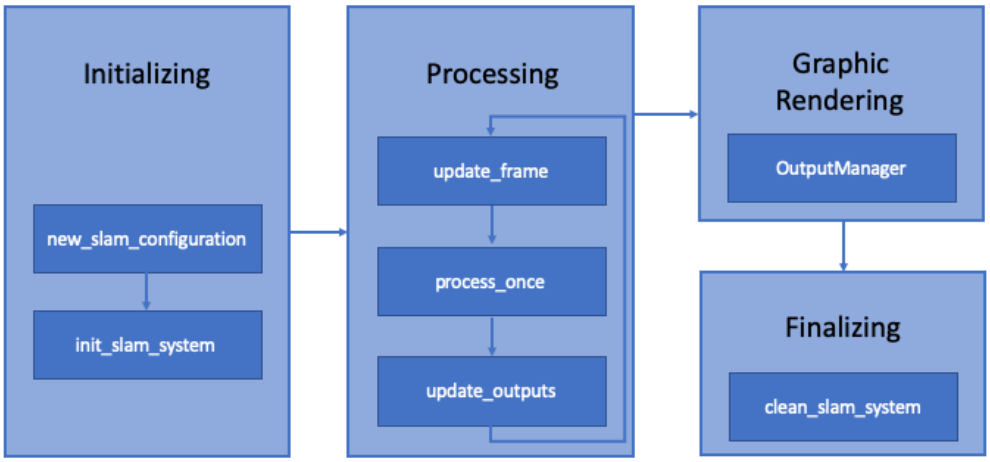
```
DATAFILE = <HEADER><SENSORS><GT_FRAMES><IN_FRAMES>
HEADER   = <VERSION:4B><SENSOR_COUNT:4B>
SENSORS  = <SENSOR 1>...<SENSOR N>
SENSOR   = <TYPE:4B><PARAMETERS>
GT_FRAMES= <EMPTY>|<GT_FRAME 1>...<GT_FRAME N>
IN_FRAMES= <IN_FRAME 1>...<IN_FRAME N>
GT_FRAME = <TIMESTAMP:8B><GT_TYPE:4B><DATA>
IN_FRAME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

By doing so, integrating new dataset will be seamless as long as the input frame types (`IN_FRAMES`) are supported by the collection of helper functions for various sensor types (indicated in the `SENSOR` object), including RGB, greyscale, depth images, IMU data and pixel events. SLAMBench data file also allows users to input ground truth (`GT_FRAME` and `GT_TYPE` as pose or point cloud) for each frame, so that these parameters can later be utilized for the accuracy evaluation.

API – Library Integration

The main challenge for SLAMBench is to integrate different SLAM algorithms that exist in various libraries. Most libraries do not conform to the same API format, preventing SLAMBench from directing interfacing with all of them. Therefore, in SLAMBench, a standalone API, along with a collection of helper objects, is created to interface with SLAM algorithms in different libraries. This API functionally abstract the general process of a SLAM algorithm into three phases: initializing, processing, and finalizing phases.

FIGURE 2.2: API Workflow [3]



In the initializing phase, `sb_new_slam_configuration` sets the user configuration for the SLAM algorithm. Then `sb_init_slam_system` initializes the SLAM system with certain memory allocation. In the processing phase, `sb_update_frame` will deliver the frame to SLAM algorithm and let the algorithm to compute the framework with `sb_process_once` function. Afterwards, estimation of pose and mapping is obtained through `sb_update_outputs`. Finally, in the finalizing phase, after all frames have been processed, `sb_clean_slam_system` release the previously allocated memory. Mapping rendering and extraction are complex, as the method is required to deal with a large number of different data structures. Since this part is not directly related to the filtering system, I will not discuss this part in detail.

Loader

Loader is a crucial middleware that connect everything together and execute the experiment. It is the main program that ingests the SLAMBench data files translated by the I/O System, triggers the loop which sends each frame to the SLAM algorithms, and outputs the trajectory map.

User Interface

The user interface produces the mapping result and benchmark metrics, including trajectory, accuracy, reconstruction, power consumption, memory usage etc. The GUI uses Pangolin library to provide a visualization of trajectory, point cloud and ground truth. The text interface produces other non-visual metrics.

2.3.2 Strengths of SLAMBench Architecture

There are a few strengths that make SLAMBench stand out from other benchmarking tools for SLAM systems: extensibility, centralized configuration, and modularity.

Extensibility with Multiple Datasets

By having a I/O system that can convert different data frames into a unified SLAMBench data files, SLAMBench is able to ingest any type of datasets that are in different formats, including ICL-NUIM, TUM RGB-D, and EuRoC MAV. Even the new dataset may contain frames that are not able to be ingested by current sensor types in SLAMBench, SLAMBench allows the user to extend the library to accommodate any kind of sensor format.

Centralized Configuration for Various Algorithms

Since the API management system of SLAMBench extrapolates the generic process of SLAM systems, SLAMBench, together with the collection of helper objects specific to each SLAM system, is able to interface with multiple types of algorithms, resulting in centralized configuration for the user in a single general API. New SLAM system can also be integrated through connecting its specific API with the API of SLAMBench.

Modularity of the Framework

The framework of SLAMBench is highly modular, as each part of system performs dedicated functions. I/O system for dataset ingestion, API management for SLAM system integration, Loader, and User Interface are all separated functions that allow researchers to modified one part of the system without changing the whole structure. In addition, modularity is also exhibited in each part of the system itself: I/O system is able to call different sensor types for different data frames; API management allows changes for one SLAM system without impacting others; User interface enables the user to change the default Pangolin library [23] to other visualization tool such as ROS visualizer [24].

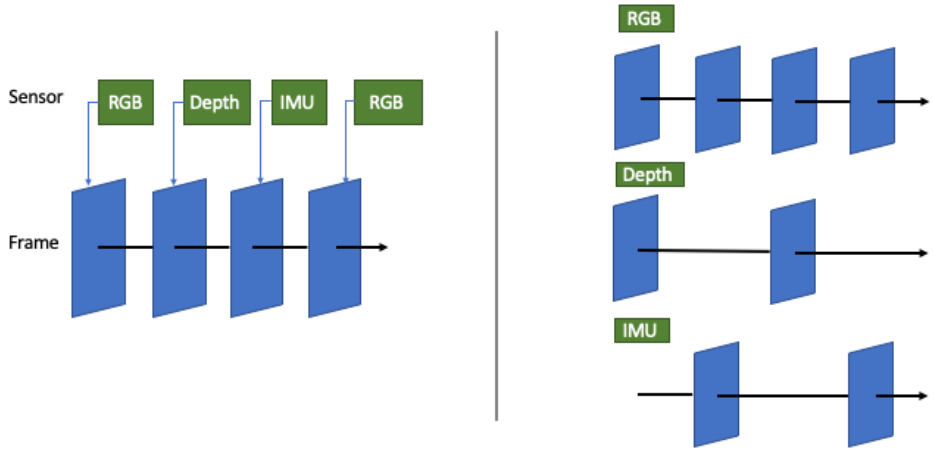
2.3.3 Weaknesses and Obstacles

However, there are also some weaknesses of the framework that brings us obstacles to integrate the new filtering system to SLAMBench.

Singular & Nonconcurrent Data Pipeline

Currently, the data ingestion process is a for-loop that iteratively ingest frame (SLAM data file) one by one. Specific sensor type will then be trigged to process the frame and send the data to SLAM algorithm for further processing and estimation. However, such singular, nonconcurrent data pipelining is different from how actually multiple sensors ingest data during mapping and localization. In reality, the processing of frames is asynchronous, as each sensor performs independently to ingest, process and deliver data to the SLAM system.

FIGURE 2.3: Example of Singular and Concurrent Input Stream



The simple, singular frame input stream may perform properly without any problem if a combined computational task of frames from multiple various sensors is not required. However, some scenarios, such as the filtering system I intend to construct, require frames from multiple sensors at the same time to perform statistical analysis. For example, for

the filtering system to drop frames from both the RGB sensor and depth sensor, a combined analysis of previous and current frames from both RGB and depth sensors is required to make a drop decision. A singular input stream of frames would not enable such analysis unless some data buffering mechanism is built.

Pre-determined Sensor Setting

Until now the sensor configuration has been predetermined by the initial input frame. For example, if the input RGB frame is 250×250, then the RGB sensor object will exclusively process the frame at 250×250. This static configuration is not a problem if the specification of the input frame does not change in the run time. However, if a filtering system is put in place to change the size of each frame at the run time, then statically configured sensor object would not be able to properly ingest the resized frame.

Difficult Installation and Interaction

Installation of SLAMBench is not a smooth process because the framework relies on specific and multiple dependencies. There is no integrated installer for SLAMBench framework, which would allow the user to perform one-click installation. There is an attempt to allow users to download all dependencies through `make dep`, but unfortunately, `make dep` does not perform properly until the time of writing this paper. In addition, although the configuration of all SLAM system is done through centralized API, for someone who is not familiar with SLAMBench, it is

still troublesome to navigate and set specific configurations through terminal.

Chapter 3

Filtering System in SLAMBench

In this chapter, we propose two semantics of filtering system in SLAMBench to help us process input sensor frames. We describe the high-level architecture and how to perform a filtering task with two proposed filtering systems.

3.1 Overview

Our filtering system takes two forms of a module which is separately defined for different filtering tasks (since we have found two types of filtering tasks that we could implement in SLAMBench). In C++, this can be achieved cleaning by declaring a header file (e.g., SLAMBenchFilterLibraryHelper.h) and using it to parameterize a filter module (e.g. identity filter). We conceive two filtering systems since so far, the filtering task for input sensor frames can be categorized into two forms: flow control and sensor modification.

Originally, in the header file `SLAMBenchFilterLibraryHelper.h`, there are mainly three functions that help the user to configure a filter and run the filtering task

1. `c_sb_new_filter_configuration` - initialize the filter with the user-defined threshold parameter

2. `c_sb_init_filter` - initialize sensors with `get_sensors()` function to make sure that all sensors are found in the sensor library

3. `c_sb_process_filter` - ingest frame one by one with the sensors defined in the sensor library, process the frame in real time, and return the processed frame to the SLAMBench. The frame is then ready to be sent to SLAM algorithms for mapping and localization.

If the user does not require the frame to be processed by batch (for example, the filtering system processes RGB frame, intensity frame and depth frame iteratively), or the user does not require the frame to be processed by updated sensors (for example, processing an resized frame requires a different sensor configuration from the original one), then this simple setting of filter (3.1) works fine.

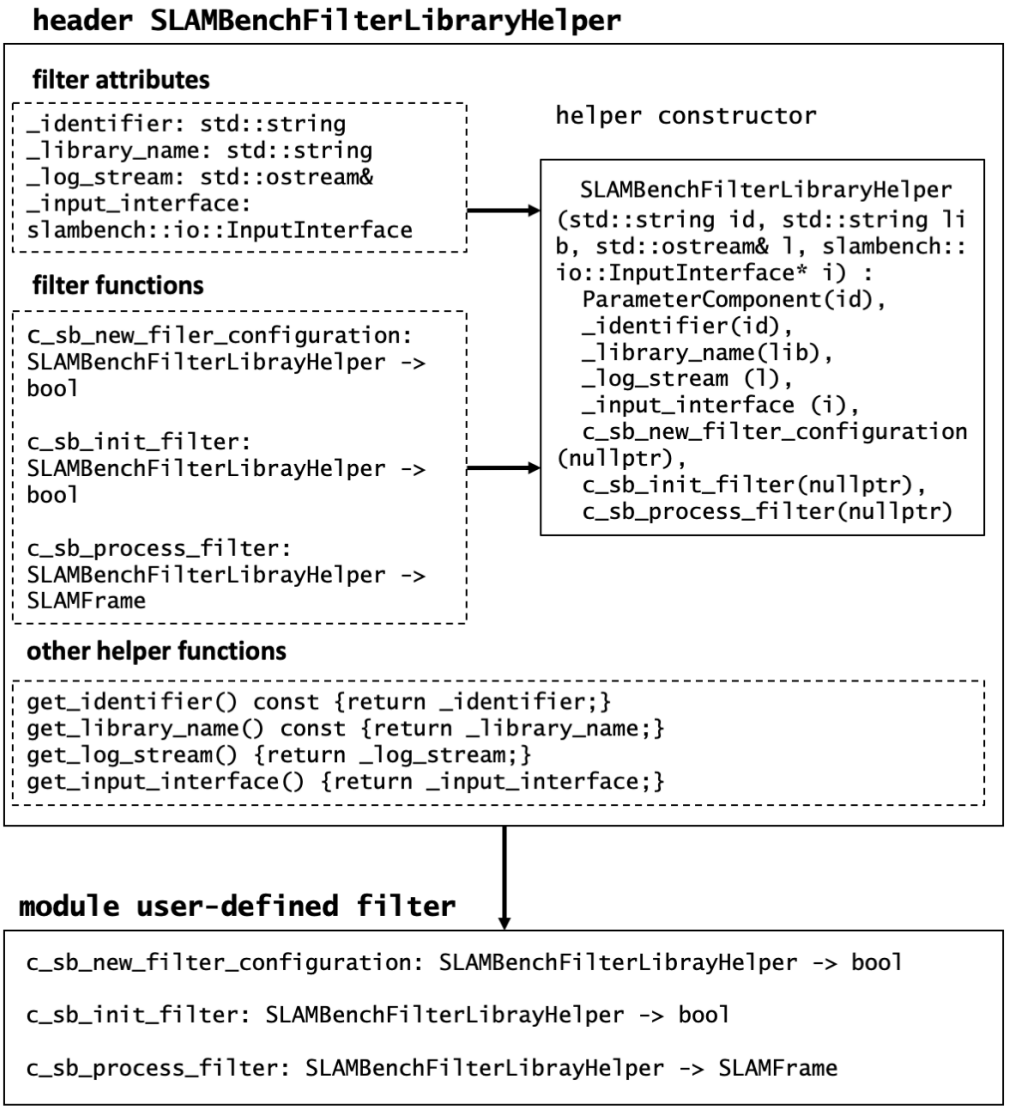
However, for more complex (beyond single, size-fixed filtering task) filtering tasks, our filtering system cannot handle them. We need to restructure our filtering system, so that it can:

1. filter more than one frame at once, and

2. modify the frame and ingest it with a software-defined sensor.

Therefore, we have designed two filtering systems: flow filtering system and sensor filtering system.

FIGURE 3.1: SLAMBench Original Filter Architecture



3.2 Flow Filter

Flow filter, as the name suggests, controls the flow of ingested frames. Before computing the frame, the SLAMBench loader calls functions in SLAMBench configurations to load libraries, datasets and sensors, retrieve ground truth and add performance metrics. After relevant objects of SLAMBench have been instantiated, the loader will start to compute frames in the `compute_loop_algorithm` function and iterate through frames one by one.

Then the current frame is passed to the filter. However, instead of processing the frame and directly returning the filtered frame to SLAMBench loader, the flow filter needs to hold the frame and decides whether the filter has enough frames to do multi-sensory filtering. If there are enough frames, then the flow filter starts to process all "buffered" frames (buffered in the sense that these frames have been stored in the flow filter with a time delayed state), and calls the SLAM algorithm to update filtered frames one by one. However, after a frame has been updated, the SLAM algorithm will also decide whether it has enough frames to process and generate a mapping output, so the SLAM algorithm will also hold on to the frames sent by filtered until sufficient frames have been passed in.

Therefore, our flow filter needs to control the occurrence of four operations:

1. `input_stream_->GetNextFrame()` - when does the SLAMBench loader pass a new frame to filter;

2. `c_sb_process_filter` - when does the filter execute filtering task (this function will be created in the actual implementation;

3. `c_sb_update_frame_filter` - when does the filter call the SLAM algorithm (lib) to update a frame;

4. `c_sb_process_once_filter` - when does the filter call the SLAM algorithm (lib) to process the frames.

And our flow filter needs to two conditional statements to control the flow of frames:

1. `c_sb_buffer_frame_filter` - a function that ensures the filter has enough frames to do the processing, and returns a Boolean value;

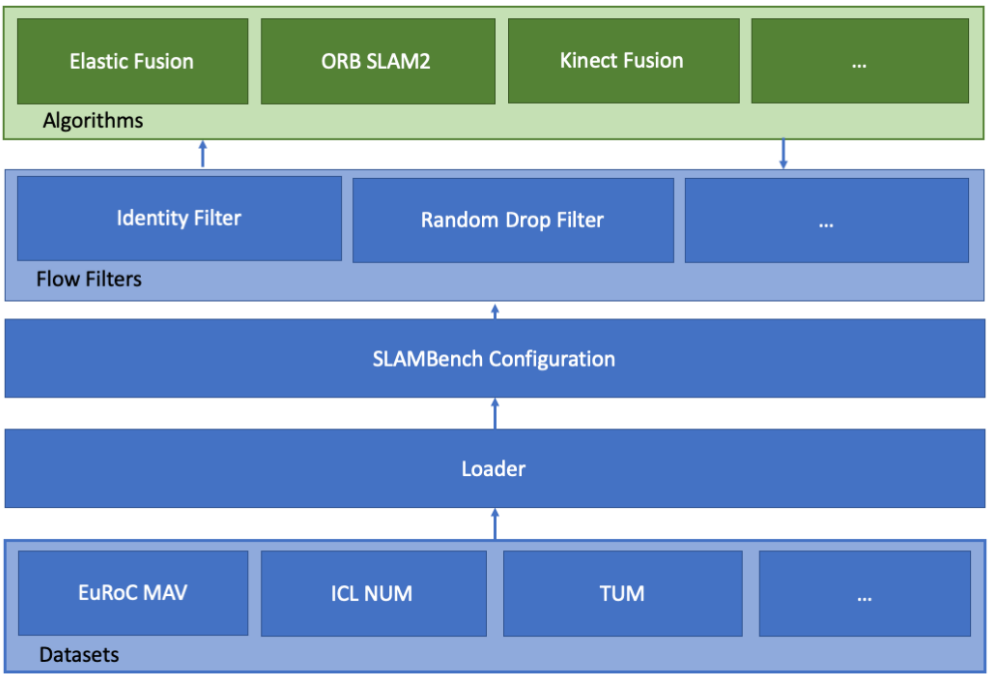
2. `c_sb_update_frame_filter` -– a function that returns Boolean value, notifying whether the SLAM algorithm has enough frames to start the processing.

3.2.1 Architectural Design & Control Flow

The flow filter is called when the SLAMBench loader calls the compute loop algorithm inside the SLAMBench Configuration. Therefore, the flow filter acts as a buffered interface between the SLAMBench Configurtation and a SLAM algorithm (3.2).

To control the flow of each sensor frame, and when it is filtered and processed, the flow filtering system executes the program in the following control flow diagram 3.3.

FIGURE 3.2: Flow Filter Architecture



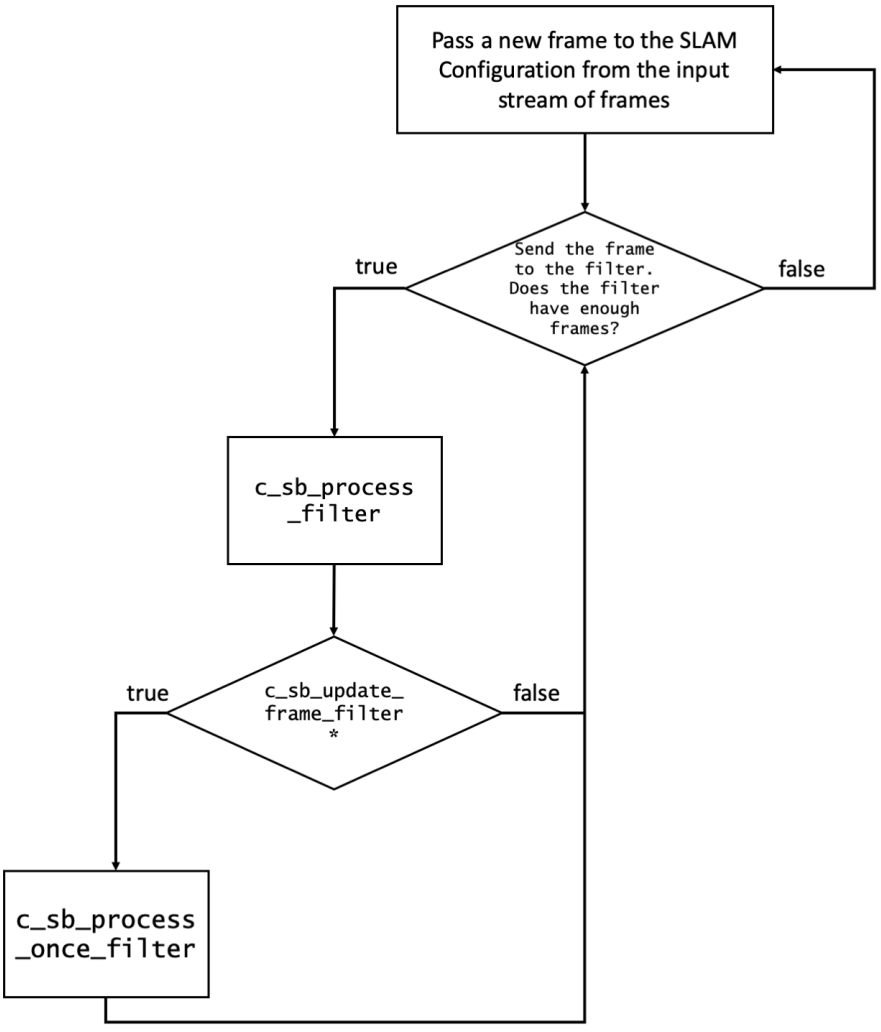
3.2.2 Multi-sensory Processing

The main important reason to have a flow filter is to allow our filtering system to perform real-time multi-sensory processing. For our flow filter, it should be able to handle multiple frames from different sensors and process them at once in a delayed time state, since our flow filter can temporarily store frames, signals the SLAMBench Configuration and calls SLAM algorithm when stored frames have met a certain condition (e.g. sufficienet number of frames / types of frames, etc.).

3.2.3 Implementation

In order to ensure that we can implement a flow filtering system inside the SLAMBench framework, we first construct an identity filter as

FIGURE 3.3: Flow Filter Control Flow Diagram

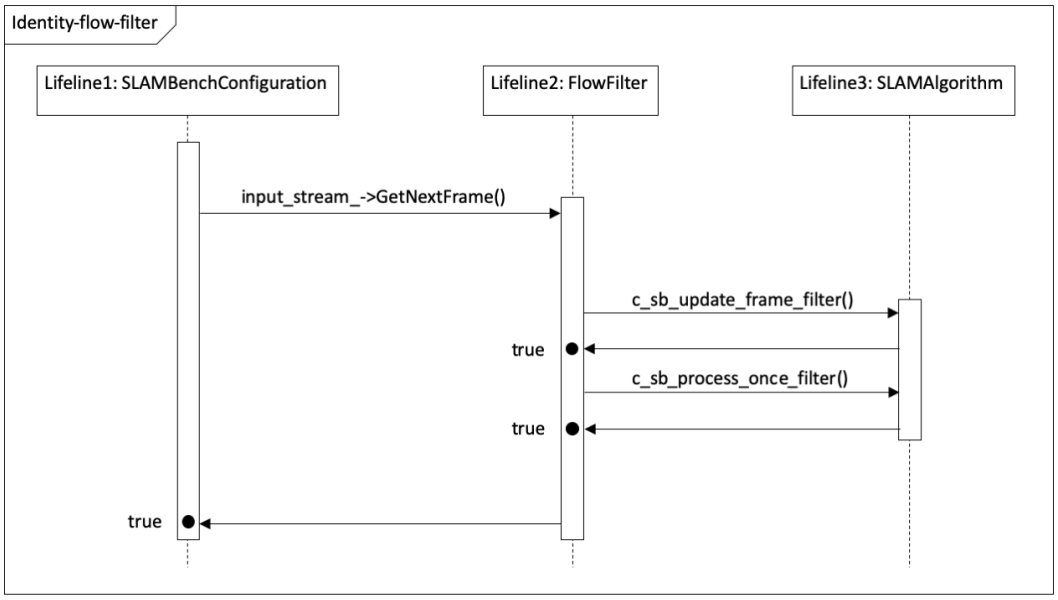


* this function not only updates the current frame for the SLAM algorithm, but also returns a boolean value, notifying whether the SLAM algorithm has enough frames

a proof of concept. The identity filter is a middleware that ingests a sensor frame from the loader, make a copy of the frame and directly pass the copied frame to update the SLAM algorithm. Most of the work has been dedicating to revamping the `SLAMBenchFilteringLibraryHelper.h` and `SLAMBenchAPI.h` so that our identity filter can utilize functions from SLAMBench API, including `c_sb_update_frame` , and `c_sb_process_once` functions.

The interaction of flow filtering for the identity filter is shown in the interaction diagram 3.4. We note that there is complete transparency between the SLAMBench Configuration, Flow Filter, and the SLAM algorithm, in the sense that each component is informed the status of other components.

FIGURE 3.4: Identity Flow Filter Interaction Diagram



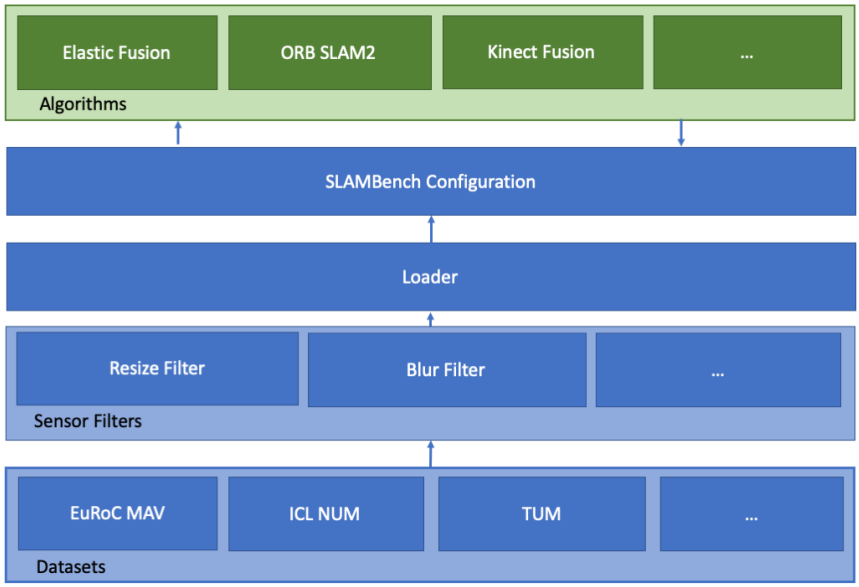
Now, to see that whether the identity filter actually produces the same

frame, we have run the SLAMBench with different data sets and algorithms to test whether SLAMBench produce the same benchmarking results with and without the identity filter. Print message is also added in the identity filter to ensure that SLAMBench actually has passed frames to the identity filter. Indeed, after thorough testing, SLAM has passed the frame to the identity filter, and it produces the same result with and without the identity filter.

3.3 Sensor Filter

Sensor filter facilitates filtering tasks (e.g., resized, blurred, etc.) that requires another set of sensor, which allows the filtered frames to be processed by the SLAM algorithm. Therefore, for the sensor filter, we set the sensor configuration before initializing and configure the rest of the SLAMBench configuration.

FIGURE 3.5: Sensor Filter Architecture



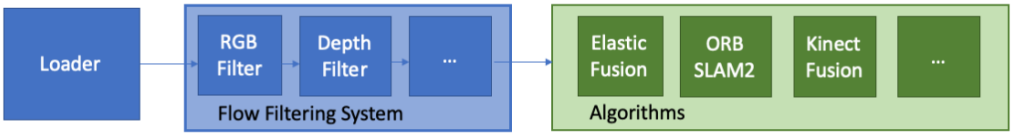
However, this architecture would require the sensor filter to have a independent set of sensor object, which is challenging to implement within in the capstone period. Hence I will move on to implement the flow filter further and come back to the sensor filter if any other alternative architecture is possible.

Chapter 4

Next Steps

The progress has been on track so far, with regard to the semester 1 plan in the project proposal. The immediate next step is to further expand the flow filtering system, so that we can implement different filters for different sensors (4.1). An preliminary graph of a complete flow filter is shown below, where each filter connects to one after the other to process frames with different sensors.

FIGURE 4.1: Flow Filter System with Multiple Filters



Further next steps include organizing the flow filters into a module, and writing a documentation for the module. Ideally these tasks could be completed by the first half of semester 2, so that focus can be placed on experimenting the filtering system with different data sets and SLAM algorithms, as well as preparing capstone deliverable such as the thesis.

Bibliography

[1] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: Part i", *IEEE robotics & automation magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[2] K. Krinkin, A. Filatov, A. yom Filatov, A. Huletski, and D. Kartashov, "Evaluation of modern laser based indoor slam algorithms", in *2018 22nd Conference of Open Innovations Association (FRUCT)*, IEEE, 2018, pp. 101–106.

[3] B. Bodin, H. Wagstaff, S. Saecdi, L. Nardi, E. Vespa, J. Mawer, A. Nisbet, M. Luján, S. Furber, A. J. Davison, *et al.*, "Slambench2: Multi-objective head-to-head benchmarking for visual slam", in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 1–8.

[4] H. Strasdat, J. Montiel, and A. J. Davison, "Real-time monocular slam: Why filter?", in *2010 IEEE International Conference on Robotics and Automation*, IEEE, 2010, pp. 2657–2664.

[5] H. Afzal, K. Al Ismaeil, D. Aouada, F. Destelle, B. Mirbach, and B. Ottersten, "Kinect deform: Enhanced 3d reconstruction of non-rigidly deforming objects", in *2014 2nd International Conference on 3D Vision*, IEEE, vol. 2, 2014, pp. 7–13.

[6] R. Madhavan, R. Lakaemper, and T. Kalmár-Nagy, "Benchmarking and standardization of intelligent robotic systems", in *2009 International Conference on Advanced Robotics*, IEEE, 2009, pp. 1–7.

[7] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems", in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2012, pp. 573–580.

[8] A. Handa, T. Whelan, J. McDonald, and A. J. Davison, "A benchmark for rgb-d visual odometry, 3d reconstruction and slam", in *2014 IEEE international conference on Robotics and automation (ICRA)*, IEEE, 2014, pp. 1524–1531.

[9] J. Weisz, Y. Huang, F. Lier, S. Sethumadhavan, and P. Allen, "Robobench: Towards sustainable robotics system benchmarking", in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 3383–3389.

[10] A. P. del Pobil, R. Madhavan, and E. Messina, "Benchmarks in robotics research", in *Workshop IROS*, Citeseer, 2006.

[11] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[12] A. R. Mahmood, D. Korenkevych, G. Vasan, W. Ma, and J. Bergstra, "Benchmarking reinforcement learning algorithms on real-world robots", *arXiv preprint arXiv:1809.07731*, 2018.

[13] G. "Klančar, A. Zdešar, S. Blažič, and I. Škrjanc, "Path planning", in *Wheeled Mobile Robotics: From Fundamentals Towards Autonomous Systems*, Butterworth-Heinemann, 2017, ch. 4, pp. 161–206.

[14] I. A. Sucan, M. Moll, and L. E. Kavraki, *The open motion planning library*. [Online]. Available: https://ompl.kavrakilab.org/.

[15] S. Perera, D. N. Barnes, and D. A. Zelinsky, "Exploration: Simultaneous localization and mapping (slam)", *Computer Vision: A Reference Guide*, pp. 268–275, 2014.

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

[17] K. Robotics, *Kuka navigation solution, 2016*.

[18] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "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.

[19] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. W. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking.", in *ISMAR*, vol. 11, 2011, pp. 127–136.

[20] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "Monoslam: Real-time single camera slam", *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 6, pp. 1052–1067, 2007.

[21] T. Whelan, S. Leutenegger, R Salas-Moreno, B. Glocker, and A. Davison, "Elasticfusion: Dense slam without a pose graph", Robotics: Science and Systems, 2015.

[22] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite", in *2012 IEEE Conference*

on Computer Vision and Pattern Recognition, IEEE, 2012, pp. 3354–3361.

[23] *Pangolin*, https://github.com/stevenlovegrove/Pangolin.

[24] *Ros visualizer*, http://wiki.ros.org/rviz.