

SensorClouds: a framework for real-time processing of multi-modal sensor data for human-robot-collaboration

Alexander Poeppel, Christian Eymüller, Wolfgang Reif

Angaben zur Veröffentlichung / Publication details:

Poeppel, Alexander, Christian Eymüller, and Wolfgang Reif. 2023. "SensorClouds: a framework for real-time processing of multi-modal sensor data for human-robot-collaboration." In *2023 9th International Conference on Automation, Robotics and Applications (ICARA), 10-12 February 2023, Abu Dhabi, United Arab Emirates*, edited by Anthony Tzes, Ramesh K. Agarwal, Xin-She Yang, and Farshad Khorrami, 294–98. Piscataway, NJ: IEEE.
<https://doi.org/10.1109/icara56516.2023.10125740>.

Nutzungsbedingungen / Terms of use:

licgercopyright

Dieses Dokument wird unter folgenden Bedingungen zur Verfügung gestellt: / This document is made available under these conditions:

Deutsches Urheberrecht

Weitere Informationen finden Sie unter: / For more information see:

<https://www.uni-augsburg.de/de/organisation/bibliothek/publizieren-zitieren-archivieren/publiz/>



SensorClouds: A Framework for Real-Time Processing of Multi-modal Sensor Data for Human-Robot-Collaboration

Alexander Poeppel, Christian Eymüller, Wolfgang Reif
Institute for Software and Systems Engineering
University of Augsburg
Augsburg, Germany
{poeppel, eymueller, reif}@isse.de

Abstract—Human-robot-collaboration (HRC) requires fast and reliable sensor data to ensure the safety of humans in the workspace. Current solutions for processing multi-modal sensor data in HRC are either highly performant in specific scenarios or offer more flexibility at the cost of decreased performance. Our GPU accelerated *SensorClouds* framework, however, combines both high flexibility and real-time performance. The architecture aids developers in quickly implementing complex HRC applications with multiple sensors by encapsulating all functionality into reusable modules. The resulting pipeline is optimized by the framework and executed in real-time.

Index Terms—human-robot-interaction, point clouds, voxel maps, multi-modal sensor data, sensor fusion, GPU computing, real-time computing, robotics

I. INTRODUCTION

Collaborative robots (so called *cobots*) have been gaining importance rapidly over the past years and are projected to continue to do so in the future [1]. Commercially available cobots nowadays ensure the safety of humans in their workspace in two different ways. On the one hand, they are typically of a lighter construction than classic industrial robots and on the other, they have built-in sensors to detect collisions with humans or the environment.

However, such cobots only allow the detection of a collision once it has happened and thus still pose a residual risk of injury to humans in the collaborative workspace. While these can be further reduced by e.g. switching the robot to a compliant impedance mode after collision [2] or adding a soft rubber coating [3] to the exterior of the robot, preventing collisions altogether is preferable. Not only does this avert bodily harm but it also increases the efficiency of operation by minimizing downtime due to mandatory safety stops after collisions.

Much research has focused on improving the collaboration between humans and robots by employing new sensors or new processing methods for existing sensors. OpenPTrack [4], for example, uses multiple cheap Microsoft Kinect sensors which produce aligned RGB-D images (2D image with additional depth information) in order to track people in the workspace of a robot. With this precise information on the exact positions of people and their extremities a much closer collaboration with

robots becomes feasible, since these can now navigate around or away from humans in the workspace before a collision can even occur. Camera sensors are typically deployed in a manner such that they capture the robot's workspace in its entirety or at least a large portion of it. Sensors proposed for measuring from the robot's structure outwards, also called *robot skins*, range from simple tactile [5] or force [6] over infrared [7] to capacitive [8], [9] sensors.

While all these proposed sensor systems for human-robot-collaboration (HRC) offer clear advantages in certain scenarios, they all suffer from distinct disadvantages in others. Camera sensors, for example, have the great advantage of being able to capture a wide field of view with a single sensor, yet their disadvantage lies in the possible obstruction of view through people or objects entering the workspace, which is of serious concern in HRC applications. Sensors that can be mounted directly on the robot structure do not suffer from this drawback, yet they typically possess maximum detection ranges far inferior to optical sensors.

Since no single sensor modality can ensure continuously accurate measurement results in every instance, multiple sensor modalities should be integrated into and, where applicable, fused in HRC applications to ensure operator safety and smooth interaction at all times. The most basic implementation of a multi-modal sensor system entails defining specific processing pipelines for each sensor and subsequently different thresholds (e.g., react when human is closer than 30 cm) and reaction strategies (e.g., move the robot in the opposite direction) and finally determining a consensual reaction to pass on to the robot controller. This, however, is tedious and error prone, since any change in the overall application behavior or the desired reaction parameters must be reflected in each processing pipeline.

II. STATE OF THE ART

Much research has been conducted into sensor fusion for HRC in recent years, yet these works mainly focus on highly specific use cases and combinations of sensor modalities. Ong et al. [10], for example, proposed a system for tracking humans in proximity of a robot by fusing data from a 2D laser range finder and a common RGB camera. By calculating

This work partly presents results of the research project *SINA* which was funded by the German Federal Ministry of Education and Research (BMBF)

the covariance intersection between hypotheses derived from the respective input data sets, the three-dimensional position of humans in the workspace can be estimated. Al Nasr et al. presented a static pipeline for the fusion of depth, color and thermal images in order to detect humans in the workspace and estimate their current poses.

Fusion of sensor data can naturally also be performed by machine learning algorithms. Wang et al. [11] use data from an RGB-D camera and wearable motion trackers to predict the intentions of humans performing collaborative assembly tasks with the robot. Zhou et al. [12] utilized the same sensors to predict future actions of a surgeon during an operation and assist them with a robotic nurse handing over surgical instruments pertinent to the next step. Such approaches, however, are inherently locked into a predetermined set of sensor modalities and usually can't generalize to others without expensive retraining.

Especially with the advent of cheaper RGB-D sensors such as the Microsoft Kinect, more and more research focused on the processing of so called point clouds, which is a simple data format describing three-dimensional sensor measurements by their coordinates in space and optional additional data such as RGB color values. The most commonly known framework for processing these is the aptly named *Point Cloud Library* (PCL) [13]. It provides a host of functionalities for the processing of point clouds such as feature and keypoint extraction or registration of point clouds to each other. Since the PCL is mostly aimed at CPU processing, processing of large datasets is not very performant.

The most closely related project to this work however, is GPUvoxels [14] by the FZI in Karlsruhe, a framework for processing large three-dimensional point cloud data sets on the GPU. Voxels are a very efficient representation of three-dimensional data and thus have been used in a wide array of applications ranging from the use in game engines for illumination rendering [15] over collision detection in physics engines [16] to the navigation of autonomous robots [17]. The internal data model of GPUvoxels, however, merely supports binary, counted or probabilistic representations of sensor data to express occupancy of a volume. Adding additional sensor modalities to the internal representation is not supported.

In summary, existing solutions offer many desirable traits such as support for multi-modal data, real-time performance, or dynamic fusion of sensor data, yet none of them support all at once. Hence, we propose a novel architecture which combines all these traits while being highly flexible and modular. Pursuant to these goals, we define the following functional requirements (FR) for such an architecture:

FR 1 Integration of all sensor data into a global model

All sensor data must be combined in a global, three-dimensional model such that global reaction strategies and thresholds can be defined.

FR 2 Real-time execution

Quick reaction times are paramount in HRC applications, since humans in the workspace must be accurately detected before collisions can occur. We target overall execution times per

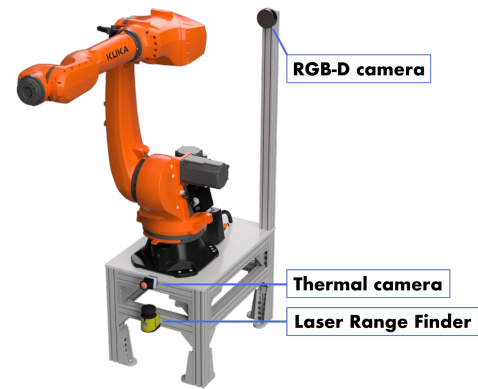


Fig. 1. Schematic of the exemplary robot cell

cycle of under 100 ms.

FR 3 Dynamic fusion of various sensor modalities

In order to increase the robustness of the HRC application, the architecture shall facilitate simple fusion of multiple sensor modalities.

III. MOTIVATING EXAMPLE

For illustrative purposes, the following sections will detail various aspects of the proposed architecture with the help of a simple application example. Imagine, if you will, a basic collaborative robot application in which a robot performs an independent pick-and-place task and a human operator is required to enter the workspace at certain times in order to assist the robot in its task or receive completed products from it. The application designer has previously defined the following three sensors as sufficient to ensure operator safety and efficient collaboration:

RGB-D camera

A camera that provides standard 2D RGB images as well as depth information and is positioned in a slight overhead position in order to maximize its viewing area.

2D laser range finder

A planar laser range finder (LRF) positioned close to the floor in order to detect the legs of humans approaching the robot.

Thermal camera

Positioned above the LRF and intended to identify humans in the workspace more easily, since they will typically be distinguishable from the environment by their body temperature.

In a typical application this setup would require a developer to process each data stream individually and define thresholds and reactions according to the intended behavior of the robot. If a suitable sensor fusion algorithm is available it could be integrated into the application, yet these typically only target a highly specific set of sensors (e.g., RGB camera and LRF) and would leave the remaining sensor data to manual integration or fusion.

IV. ARCHITECTURE OF *SensorClouds*

The main goal of *SensorClouds* is to provide a highly modularized and runtime efficient framework for processing multi-modal point clouds in real-time. Integration and fusion

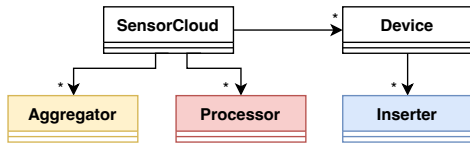


Fig. 2. Overview over the main components of the *SensorClouds* architecture.

of sensor data into a single global model shall be facilitated in order to enable globally defining thresholds and reaction strategies. To this end the architecture defines the following key concepts also shown in fig. 2:

SensorCloud

This is the main environment model and central to the entire framework in that it coordinates all necessary steps in the compilation and execution of applications. It is also responsible for data storage for the resulting model. This addresses **FR 1**.

Device

Input data streams are not configured manually but through device drivers. Additional information on the sensor providing the input data is stored here as well and can be accessed by other components at runtime. *Devices* are then associated with the desired *SensorCloud* their data is to be added to. All the sensors in the motivating example are represented by custom *Device* drivers.

Inserter

An insertion module is responsible for adding input data to the global environment model and optionally transform it beforehand. *Inserters* are always associated with *Devices* as their purpose is the transfer of data from a specific *Device* to the global model. In the motivating example multiple different *Inserters* are needed: the simple *Inserters* which perform direct insertion of data which is already located in three-dimensional space and one *Inserter* which performs a projection of the thermal image onto three-dimensional data from other sensors and thus introduces a data dependency to other modules. This is the first of two methods for implementing **FR 3**.

Processor

Processing modules operate on the global model after all input data has been inserted and can provide a wide range of functions including filtering, detection and export of the model data. A *SensorCloud* can have any number of *Processors* associated with it. In the motivating example this could be a simple filter, which extracts all data points where the measured temperature is within the typical range of a human body.

Aggregator

Aggregation modules perform the fusion of multiple data points inserted into the same voxel and are applied to *Inserters* and *Processors*. *Aggregators* are associated with a *SensorCloud* since they are intended to govern the aggregation of data points in the model for any subsequent data write access. Consequently, write access to the global model is never direct, but always cached by the aggregator, which then inserts the resulting value into the model. In the motivating example an *Aggregator* is needed to fuse the data points from the RGB-D

sensor and the other three-dimensional sensors for one location (more details in section IV-B). This fusion can be implemented as simple calculation of the mean of the inserted data points or determine occupation probabilities, trust values, etc. and thus is the second method for implementing **FR 3**.

During execution, all configured modules are added to a global pipeline and subsequently ordered according to their step in the execution order (*Inserters* must be executed before *Processors* can be executed) and within those steps by their data dependencies to other modules, if any.

The *SensorClouds* architecture is designed to be utilized by two distinct types of developers:

Application Developer

Application developers can use *SensorClouds* to develop collaborative robotics applications. They can add an arbitrary number of devices to the model and define the desired *Inserter* and *Processor* modules for the *Devices* and the global *SensorCloud*, respectively.

Module Developer

In order to provide functionality which can later be used by application developers, module developers create *Inserter*, *Processor* and *Aggregator* modules which contain small reusable code fragments for processing individual data points.

This separation allows application developers to focus entirely on their application through using predefined algorithmic fragments implemented by module developers. In order to guarantee expected behavior, modules in the *SensorClouds* framework are specified following the *design-by-contract* [18] paradigm such that modules must explicitly specify which data fields they read and write as well as which data types they can process. A module is also explicitly prohibited from accessing any data fields not specified in its contract from any source.

A. Module Definition

Every module regardless of its type (*Inserter*, *Processor* or *Aggregator*) defines an atomic operation on a single data point from the respective data source it is associated with and as such is conceptually akin to shading languages utilized in computer graphics programming. This enables *SensorClouds* to automatically optimize the execution of all modules in the pipeline for real-time processing.

As depicted in fig. 3, the definition of a module must specify an executable method which performs the actual computation as well as the fields in the data point that are read from and written to by name and type. An *Aggregator* must only specify which intermediate variables it requires within the model and is solely eligible to specify distinct methods for various data types within a single module definition. An example of this is a simple mean calculation, which may define different implementations of its arithmetic operation for various primitive types. The set of data fields which is written into the model by a module is referred to as *Model Output*. *Processors* must additionally define which values they need to read from the global model in order to perform their calculations (*Model Input*). Finally, *Inserters* incur the most data dependencies and in addition to *Model Input* and *Model*

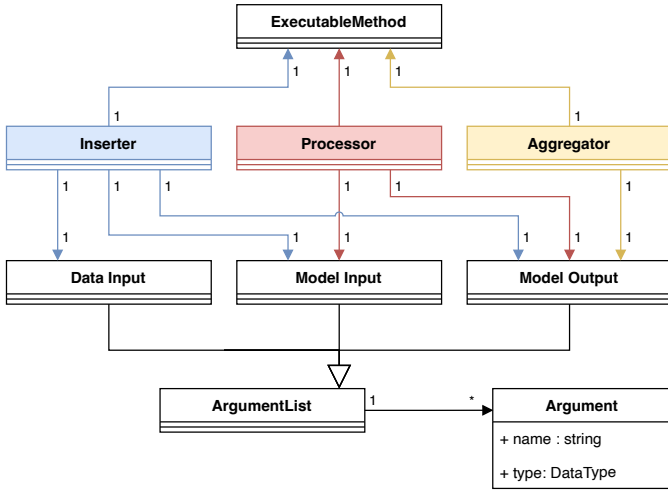


Fig. 3. Definition of modules in the *SensorClouds* architecture.

Output must specify which fields of an incoming point cloud they are able to insert into the model (*Data Input*).

These specifications of read and written data comprise the contracts to which each individual module is bound by the *SensorClouds* framework.

B. Data Model

The basic underlying data structure upon which *SensorClouds* operates is a *voxel map*. In *SensorClouds* the voxel map is initially configured by three parameters which are immutable at runtime:

- voxel resolution in m (identical for x , y and z)
- number of voxels in every dimension (n ; total number of voxels is consequently n^3)
- origin point of the voxel map

All data inserted into the global model is discretized according to the voxel resolution. This necessitates the use of *Aggregators* to fuse multiple data points that fall within the same voxel boundaries, but consequently results in an already fused and reduced dataset, since the number of voxels is always the same and an application developer can explicitly specify the required resolution of the voxel map a priori.

The contained data fields of each voxel are determined in the compilation phase from the requirements of the configured modules, i.e. the outputs of *Inserters*, *Aggregators* and *Processors*. Prior to the execution of the global pipeline all the modules are enumerated and their respective outputs and memory requirements are added to the global data model. Consequently, all memory allocation for the voxel map including all voxels and their contents can be handled during initialization, thus eliminating the necessity of allocating additional memory at runtime.

V. USAGE EXAMPLE

In order to achieve the best results in accordance with **FR 2** a reference implementation was created which performs the processing of sensor data on a GPU. Specifically, we chose CUDA C++ for this implementation.

```

auto sc = SensorCloud(256, 0.025, Origin::CENTER);

auto realsense = SCDeviceRSL515("rs", "/rs/points");
realsense.registerInserter<SCXYZInserter>();
auto sick = SCDeviceSickLRF("sick", "/sick/points");
sick.registerInserter<SCXYZInserter>();
auto flir = SCDeviceFLIR("flir", "/flir/imagerect");
flir.registerInserter<SCThermalInserter>();

sc.addDevices(&realsense, &sick, &flir)
.completeDeviceRegistration()
.registerAggregator<MeanAggregator>({ "x", "y", "z" })
.registerProcessor<ThermalFilter>(&out, 36.1, 37.2);

sc.buildPipeline();
sc.executePipeline();

```

Listing 1: Application structure in *SensorClouds*.

The program structure of an implementation for a configuration as in the motivating example can be seen in listing 1. The main component is the definition of a *SensorCloud* object, which in this case is configured to have a voxel resolution of 2.5 cm and a voxel number of 256 in each dimension, resulting in a total number of $256^3 = 16.777.216$ voxels. The origin is placed at the center of the overall volume. Next, the required *Device* drivers are loaded alongside their respective *Inserters* registered with the former. When all the desired devices have been configured, the registration process is completed. Then the previously instantiated *Devices* and an *Aggregator* which simply calculates the mean value of all inserted coordinates are added to the *SensorCloud*. The last required component is the *Processor* needed for extracting the results from the model where the temperature measured falls between two given values. Finally, the global pipeline can be generated, compiled and executed.

At no point in the development process must an application developer optimize for real-time execution, e.g., by directly interfacing with hardware or manually defining parallel processing rules, since all of these details are handled by the framework internally. Through the explicit specification of module contracts by the module developer, the application developer is guaranteed the desired behavior of the overall application.

VI. EVALUATION

To evaluate our approach we compared the runtimes of *SensorClouds* against those of GPUvoxels and the PCL. Since all remaining aspects of the respective architectures vary greatly, we focused solely on the runtime of the insertion operation of a point cloud into the voxel model. For the purpose of evaluating only the raw performance and eliminating the influence of different point cloud sizes and invalid measurements, the evaluation of all three frameworks was performed with a single static point cloud and repeated 10,000 times. Also, the graphical user interface was disabled in order to remove any influence updates of the display output could have on the execution of the computation on the GPU.

TABLE I. RESULTS OF THE EVALUATION (POINT CLOUD WITH 43941 POINTS; 10000 EXECUTIONS EACH).

Framework	Avg [μ s]	Median [μ s]	StdDev [μ s]	Min [μ s]	Max [μ s]
<i>SensorClouds</i>	25.460	17.632	18.821	11.392	158.080
GPUvoxels	15.270	14.496	4.124	12.672	210.496
PCL	1218.544	442.935	9052.105	248.251	171 473

We evaluated all frameworks on the same hardware consisting of an Intel Core i7-8700K and an Nvidia RTX 2080 Super. The software setup consisted of a standard Ubuntu 20.04 installation with g++ 9 and CUDA 11.2 compilers. The time measurements were captured with the default mechanisms provided by the respective standard libraries. The results of the evaluation of the insertion operations can be found in table I.

Comparing the three frameworks, it should be noted that only *SensorClouds* and the PCL are capable of processing multi-modal input, whereas GPUvoxels merely updates a single value for each voxel, either probability or count, according to the coordinates of a point. Especially considering this, *SensorClouds* reaches highly competitive performance compared to GPUvoxels, while the PCL is orders of magnitude slower since it cannot benefit from GPU acceleration. It was however included in this evaluation since it is the predominant framework for processing of point clouds.

The average execution time for a simple filter operation (*Processor*) which iterates over every voxel in the model (for 256^3 voxels) is 1.77ms. This execution time can, however, be greatly reduced by optimizing the operation to only consider voxels actually relevant to it. A simple check for the occupancy of a voxel is insufficient to accomplish this, since branch-divergent code actually executes more slowly on GPUs. Indexing the voxel map during or after insertion of the sensor data could be a possible solution to improve the execution time of *Processors* in sparsely populated models.

VII. CONCLUSION

SensorClouds is a novel architecture for processing multi-modal sensor data in real-time. We have demonstrated how our approach enables developers of applications to quickly integrate multiple heterogeneous sensors and process as well as fuse their respective data with the help of modules implemented by module developers. An evaluation of the predominant competitors to our approach showed that *SensorClouds*, while arguably more complex and processing intensive, exhibits highly competitive performance figures. In future research we plan to further optimize the execution and extend the functionality of the framework. Faster search in and retrieval of data from the global model are key points in this respect. To this end we plan to develop a query language for the retrieval of relevant data. Also, we plan to investigate the integration of further, not natively three-dimensional sensor data into our model.

REFERENCES

- [1] T. M. Anandan. Robot industry trends and potential for the future. [Last accessed 2022-09-16]. [Online]. Available: <https://www.controleng.com/articles/robot-industry-trends-and-potential-for-the-future/>
- [2] S. Haddadin, A. Albu-Schaffer, A. De Luca, and G. Hirzinger, "Collision detection and reaction: A contribution to safe physical human-robot interaction," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 3356–3363.
- [3] G. Pang, G. Yang, W. Heng, Z. Ye, X. Huang, H.-Y. Yang, and Z. Pang, "Coboskin: Soft robot skin with variable stiffness for safer human–robot collaboration," *IEEE Transactions on Industrial Electronics*, vol. 68, no. 4, pp. 3303–3314, 2021.
- [4] M. Munaro, F. Basso, and E. Menegatti, "Openprtrack: Open source multi-camera calibration and people tracking for rgb-d camera networks," *Robotics and Autonomous Systems*, vol. 75, pp. 525–538, 2016. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0921889015002304>
- [5] T. Mukai, M. Onishi, T. Odashima, S. Hirano, and Z. Luo, "Development of the tactile sensor system of a human-interactive robot "ri-man";," *IEEE Transactions on Robotics*, vol. 24, no. 2, pp. 505–512, 2008.
- [6] M. Makihata, M. Muroyama, S. Tanaka, T. Nakayama, Y. Nonomura, and M. Esashi, "Design and fabrication technology of low profile tactile sensor with digital interface for whole body robot skin," *Sensors*, vol. 18, no. 7, p. 2374, 2018.
- [7] V. J. Lumelsky, M. S. Shur, and S. Wagner, "Sensitive skin," *IEEE Sensors Journal*, vol. 1, no. 1, p. 41, 2001.
- [8] A. Hoffmann, A. Poeppel, A. Schierl, and W. Reif, "Environment-aware proximity detection with capacitive sensors for human-robot-interaction," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 145–150.
- [9] A. Poeppel, A. Hoffmann, M. Siehler, and W. Reif, "Robust distance estimation of capacitive proximity sensors in hri using neural networks," in *2020 Fourth IEEE International Conference on Robotic Computing (IRC)*, 2020, pp. 344–351.
- [10] K. S. Ong, Y. H. Hsu, and L. C. Fu, "Sensor fusion based human detection and tracking system for human-robot interaction," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 4835–4840.
- [11] W. Wang, R. Li, Y. Chen, Z. M. Diekel, and Y. Jia, "Facilitating human–robot collaborative tasks by teaching-learning-collaboration from human demonstrations," *IEEE Transactions on Automation Science and Engineering*, vol. 16, no. 2, pp. 640–653, 2019.
- [12] T. Zhou and J. P. Wachs, "Spiking neural networks for early prediction in human–robot collaboration," *The International Journal of Robotics Research*, vol. 38, no. 14, pp. 1619–1643, 2019. [Online]. Available: <https://doi.org/10.1177/0278364919872252>
- [13] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China: IEEE, May 9–13 2011.
- [14] A. Hermann, F. Drews, J. Bauer, S. Klemm, A. Roennau, and R. Dillmann, "Unified gpu voxel collision detection for mobile manipulation planning," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 4154–4160.
- [15] A. Simion, V. Asavei, S. A. Pistricea, and O. Poncea, "Practical gpu and voxel-based indirect illumination for real time computer games," in *2015 20th International Conference on Control Systems and Computer Science*, 2015, pp. 379–384.
- [16] M. Sagardia, T. Stouraitis, and J. L. e Silva, "A new fast and robust collision detection and force computation algorithm applied to the physics engine bullet: Method, integration, and evaluation," in *Conference and Exhibition of the European Association of Virtual and Augmented Reality (2014)*, December 2014. [Online]. Available: <https://elib.dlr.de/102551/>
- [17] M. Muglikar, Z. Zhang, and D. Scaramuzza, "Voxel map for visual slam," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 4181–4187.
- [18] B. Meyer, *Design by contract*. Prentice Hall Upper Saddle River, 2002.