Modular ROS 2 and YARP End-to-End Pipeline for 3D Scene Reconstruction Using Manipulator Arms

Bartek Łukawski*[1], Irene Calvo-Corredera[2], José Ignacio Ros-Pérez[2], Celia Martín-Parra[2],
Edwin Daniel Oña[1], Juan G. Victores[1], Carlos Balaguer[1], Alberto Jardón[1]
Departamento de Ingeniería de Sistemas y Automática, RoboticsLab
Universidad Carlos III de Madrid, ROR: 03ths8210
Avenida de la Universidad, 30 (edificio Sabatini), 28911, Leganés (Madrid), España
[1]{blukawsk*, eona, jcgvicto, balaguer, ajardon}@ing.uc3m.es
[2]{irene.c.corredera, joseignacio.ros, celia.m.parra}@alumnos.uc3m.es

***Abstract*—We present an end-to-end pipeline for live 3D scene reconstruction, from raw point cloud acquisition to final mesh generation, ready for 3D printing. The hardware setup relies on a consumer-grade RGB-D camera (e.g., the Intel RealSense), an optional turntable controlled via a low-cost microcontroller (Arduino board), and a robotic arm (we used the ABB CRB 15000 "GoFa" collaborative robot). The underlying software architecture relies on fully open-source tools, with the OpenCV implementation of the KinectFusion algorithm at its backbone. This approach leverages the popular ROS 2 and YARP robotic middlewares, enabling a distributed network of reusable, modular components. We seek to provide a flexible framework that can be easily extended to a wider range of tools and robots. The system has been validated across three real scenarios; the 3D scanned meshes of three different objects (rubber duck, toy robot, human head) of varying sizes and features have been obtained, successfully generating 1:1 scale 3D-printed models. The source code of all developed components is openly available.**

***Index Terms*—software architecture, collaborative robotics, 3D scene reconstruction, RGB-D sensor, middleware, ROS 2, YARP**

I. Introduction

3D scene reconstruction creates digital representations of real scenes from input data. In this work, we focus on generating mesh-based models of static indoor scenes, specifically real-world, isolated static objects, from point clouds acquired in real-time with consumer-grade RGB-D cameras in eye-in-hand setups. The underlying methodology has been extensively studied in recent years, stemming from the seminal KinectFusion system introduced by Newcombe et al. in 2011 [1], [2]. Years later, contributors of the open source OpenCV computer vision library reimplemented the method as "KinFu".

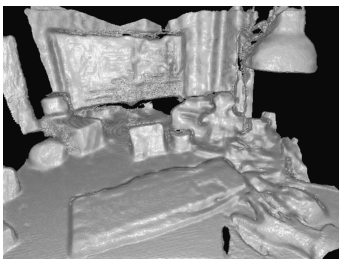
This project is framed in a broader effort within the RoboticsLab group to foster robotics education while enriching our software ecosystem. Introducing a robot arm into the scene reconstruction pipeline serves as an effective demonstration tool for showcasing robot capabilities to students in graduate and undergraduate courses. Additionally, we seek to expand the applications of industrial ABB robots in educational and research contexts. In practice, our system provides an end-to-end 3D scanning tool based on available building blocks to create physical models of arbitrary objects or scenes, rendering it useful for rapid prototyping, for instance.

II. Background

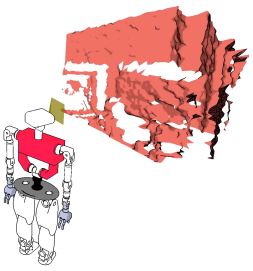
This work iterates over previous developments by the same authors; some are depicted in Figure 1.

Most notably, a library was introduced to expose Point Cloud Library (PCL) cloud and mesh processing algorithms through YARP interfaces, easing the development of applications and adding the necessary utilities for producing a mesh from the point cloud generated by KinectFusion (1a, [3]). An experiment with the OpenRAVE model of our humanoid robot TEO demonstrated its versatility by constructing meshes from real-time point cloud data acquired from an RGB-D sensor installed on its head (1b).

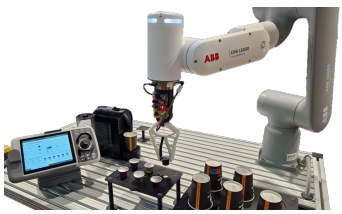
On the other hand, practical demonstrators have been developed on the ABB GoFa platform, for which a 3D-scanning application poses an interesting addition towards conducting practical sessions with students: a coffee-making station driven by a custom graphical interface (1c, [4]), and a drawing task programmed via kinesthetic teaching (1d, [5]).



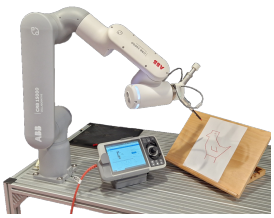
(a) Sample 3D scan using OpenCV's KinFu implementation [3].



(b) Mesh generation from live RGB-D data in OpenRAVE [3].



(c) Coffee-making demonstrator [4].



(d) Drawing application [5].

Fig. 1: Relevant previous works by RoboticsLab UC3M.

Scene reconstruction is a mature research topic; however, the algorithmic and theoretical aspects of this subject are outside the scope of the present work, which focuses on the integration strategies and software architectures involved in the data acquisition, processing and execution as a whole.

Several studies have tackled the conjunction of a vision system and a robotic platform to automate the reconstruction process. Integrating the latter usually plays a secondary role in the overall work in favor of other facets such as motion planning. For instance, Krainin et al. combined a mobile manipulator and an eye-to-hand RGB-D camera setup to propose an autonomous system that actively investigates the environment while enforcing a next-best-view (NBV) policy [6].

The remaining cited works adhere to an eye-in-hand setup whenever a robot is involved. Similarly to Krainin, Torabi et al. built an autonomous NBV-compliant system that combines a sensor-based path planner to account for collision-free paths, modeling static objects with a laser-range scanner [7]. On the other hand, Naazare et al. adopted NBV in an exploration and inspection context with unmanned ground vehicles (UGV) and RGB-D sensors for disaster response. They relied on popular open-source tools: Gazebo for simulation and the MoveIt stack of the ROS ecosystem for arm motion planning [8]. Vazquez-Gomez et al. approached the problem with an evaluation strategy that allowed for iteratively planning a set of views for scanning static objects with a redundant arm and a Kinect-like sensor [9]. Lastly, Madhusudanan et al. minimized data stitching errors caused by drifting Denavit-Hartenberg parameter values through periodic camera recalibration, achieving comparable quality and lower system downtime in contrast to industrial-grade marker-based methods [10].

In other works, additional mechanisms such as turntables are used instead of or to complement the motion of a robot arm. In order to build an image-model dataset of real-life objects for benchmarking, Calli et al. adopted two frame acquisition setups involving a turntable and a set of color and depth sensors placed along a circular arc. The turntable automatically turns and stops at specific angles. The entire acquisition and reconstruction process was partially automated to a varying extent depending on the method, leveraging open-source tools such as OpenCV for camera calibration, and ROS for managing point clouds and the generation of URDF descriptor files towards an easy integration in the Gazebo simulator. Similarly to our experimental setup, the Poisson reconstruction algorithm was adopted in their case [11]. On a final note, Azhar et al. assessed the outcome of a scanning sequence performed using a robotic arm, compared against a turntable-based object reconstruction. To implement their system, proprietary MATLAB software was used [12].

In this work, it is not our aim to assess the quality of the reconstruction nor to propose a planning algorithm to perform motion. We introduce a fully open-source framework for the pipeline, from data acquisition to motion command, encompassed in a modular and configurable end-to-end process. Beyond the use-case described later, others could be easily covered, e.g. eye-to-hand vision and NBV-based planning.

III. Materials and Tools

A. Hardware

Figure 2 depicts the hardware tools involved in the final application, along with their control or data acquisition units: an RGB-D sensor (2a), a 1-DoF actuated rotating platform or turntable (2b), and a robotic manipulator arm (2c). The proposed architecture is flexible enough to accommodate a wide variety of products and vendors. In this work, an Intel RealSense D435i depth camera is used to register point clouds of the scanned object. It is attached through a custom 3D-printed adapter to an ABB CRB 15000-5 "GoFa" manipulator; this 6-DoF collaborative robot (cobot) incorporates intrinsic safety mechanisms, thus enabling shared environments in which the cobot can collaborate with human operators. Lastly, a 20 cm diameter turntable has been designed, resin-printed, and actuated with a NEMA 17 bipolar stepper motor, controlled by an A4988 driver and an Arduino board.



(a) RGB-D camera (b) Turntable (c) Manipulator arm

Fig. 2: Hardware tools.

B. Software

Two popular open-source robotics middleware frameworks, YARP [13] and ROS 2 [14], have been adopted in order to take a modular approach that leverages distributed computing and empowers reuse across the software components involved in the final application. YARP relies on a lightweight stack, a rich library of hardware drivers (or "devices"), and C++/Python programming interfaces. It facilitates the development of applications in the existing RoboticsLab ecosystem, although it also provides enough flexibility for other environments while not impeding a future migration to ROS 2. This latter middleware, on the other hand, was devised to handle robot motion control due to its widespread adoption in the robotics community, thus ensuring its compatibility with diverse robot platforms. Particularly, a bridge between ROS 2 and the Externally Guided Motion (EGM) high-frequency (250 Hz) command feature of ABB controllers has been introduced to enable communication with this brand of robots [15]. On a final note, the OpenCV library is leveraged in the cloud processing step due to the implementation of KinectFusion algorithms.

IV. Proposed Architecture

Figure 3 illustrates the software architecture proposed in this work. It leverages YARP and ROS 2 to introduce a distributed network of single-purpose components implementing inter-process communication (IPC) techniques such as the Remote Procedure Call (RPC) protocol and data broadcasting. This strategy provides advantages over a monolithic solution such as flexibility and modularity. For instance, developers aren't constrained to tie their applications to a single operating system, programming language, computing unit or framework. Each building block introduced in this architecture can run on Linux, Windows (YARP and ROS 2) or Mac (YARP) systems, be written in C++ or Python, execute on different machines over a local network, or rely on diverse third-party libraries (e.g. instead of the OpenCV implementation presented here).

For the sake of simplicity, a single "control PC" is depicted in the diagram, even though its components could be shared across a constellation of machines with varying computing capabilities, such as a dedicated GPU for the more hardware-intensive KinectFusion reconstruction process. This PC orchestrates depth frame acquisition from the RGB-D camera attached to the robot, it sends motor commands to the Arduino controller (or equivalent) governing the turntable, and interfaces with the ABB robot through its proprietary controller via external EGM commands and state messages.

Within the "control PC" block, a scene reconstruction service is executed in the background, orchestrated by YARP and communicated with other components through YARP ports following two models: depth frames arrive from an external RGB-D frame acquisition service that internally handles low-level communication with the depth sensor (sender-receiver model); execution flow through "pause" and "resume" commands can be issued externally via RPC (client-server model). Internally, this service implements the KinectFusion algorithm as detailed in the following section. On each iteration, once the RPC client issues a start command, incoming point clouds are used to construct the 3D surface of the scanned object. Its live Phong-rendered results are output to a viewer tool, enabling the user to supervise the progress of the reconstruction.

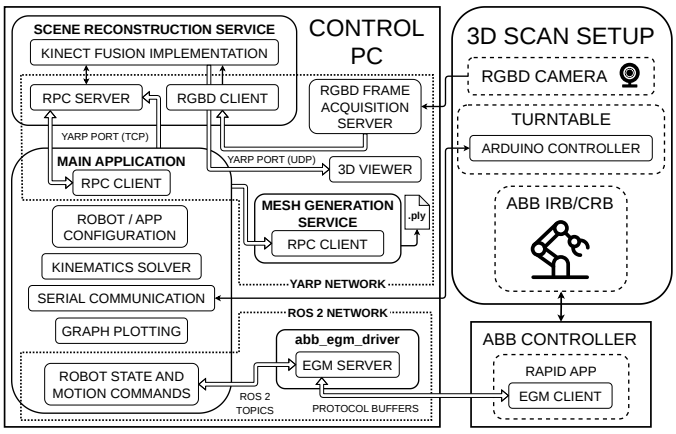


Fig. 3: Proposed software architecture.

Execution flow is orchestrated by the main application. On startup, a set of parameters is loaded to define the behavior of the application and the kinematics of the robot. The latter are used to build a feasible trajectory for the depth camera acting as the robot's end-effector. Plotting tools can be integrated to provide a graphical representation of said trajectory, allowing the user to make any adjustments needed in case joint limits or singularities are reached, for instance. A serial communication channel is opened to command the rotary motion of the turntable, while a ROS 2 publisher is created to communicate with the EGM-based node we developed (abb_egm_driver) to send Cartesian and joint commands to the ABB robot.[1]

After initial configuration and trajectory validation by the user, the application issues motion commands to the robot and turntable, and initiates the scene reconstruction process through an RPC request. Another request is sent at the very end to pause the reconstruction; at that point, an external mesh generation service can be run to fetch the resulting point cloud from the scene reconstruction service via RPC request, in order to generate a mesh and store it on disk in the PLY format.

V. Implementation

A. Scene Reconstruction Service

Our scene reconstruction service is built around the *KinectFusion* C++ interface and its concrete instantiations, proxying OpenCV's classes. A Unified Modeling Language (UML) class diagram representing their relationships is depicted in Figure 4. This interface leverages YARP utilities for storing point clouds and images to operate upon the color and depth channels received from the RGB-D camera. The contract is fulfilled by the class template *KinectFusionImpl*, which privately stores a handle to a generic instance of the underlying OpenCV implementation. Instantiations are provided for each supported KinectFusion algorithm variation.

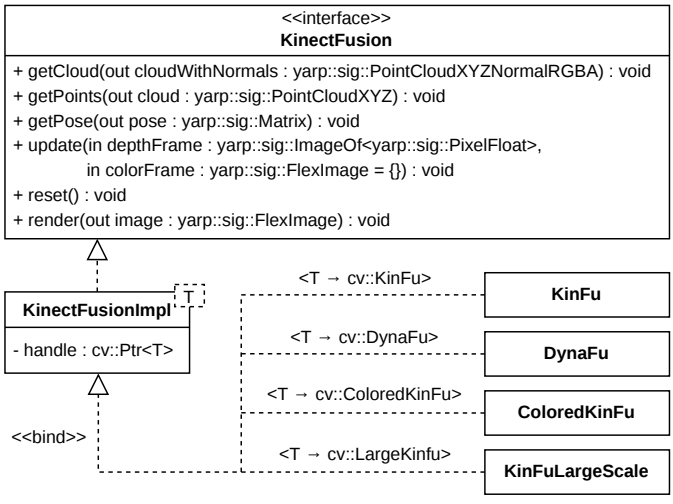


Fig. 4: UML class diagram of the KinectFusion specification (C++ interface) and its realization (concrete classes).

[1] https://github.com/roboticslab-uc3m/abb_egm_driver

The base class methods are listed next:

- `getCloud`: retrieves the resulting point cloud with normals and color data (if available).
- `getPoints`: retrieves the resulting point cloud.
- `getPose`: gets the current camera pose as a homogeneous transformation matrix.
- `update`: accepts a depth and color frame (if available) for the next reconstruction step.
- `reset`: discards and starts a new reconstruction.
- `render`: prepares a Phong-rendered image of the current reconstruction step.

In order to expose these methods and their results to clients, the service adheres to the Thrift Interface Description Language (IDL) to implement them along with additional `pause` and `resume` commands. A YARP RPC server port is used as communication channel for that purpose.

B. Main Application: Trajectory Generation and Execution

The main application orchestrates the execution flow across a sequence of steps, guiding the user through the process:

1) Load configuration via YAML files. This contains trajectory parameters and the Denavit-Hartenberg description of the robot.
2) Generate the Cartesian trajectory and a plot thereof for user inspection.
3) Perform inverse kinematics to obtain the separate trajectories of each robot joint. Generate a plot for inspection, flag unreachable points, calculate trajectory parameters (e.g. total execution time and joint ranges).
4) Command the robot to the initial pose with a point-to-point command via ROS 2 topic.
5) Issue an RPC command to the scene reconstruction service to begin processing frames.
6) Issue motion commands to the turntable (via serial communication) and the robot (via ROS 2 topic). The scanning process happens through this stage.
7) Once motion is completed, issue an RPC command to the scene reconstruction service to finish processing.

The application aims to prevent interrupting the later scanning process by performing prior kinematics checks. This allows users to design and preview a trajectory without full prior knowledge about the final outcome. The user is therefore prompted to validate the parameters or make any posterior adjustments if necessary.

C. Mesh Generation Service

Implemented as a stand-alone process, the mesh generation service is launched on-demand once scene processing is finished in order to issue an RPC command to its integrated YARP server port. The returned point cloud resulting from the 3D scan is retrieved along with the associated normal vectors. A helper utility developed by the authors in previous works [3] bridges YARP types with the open-source Point Cloud Library (PCL) to construct a mesh from said cloud. It is stored on disk in a PLY file for later use, e.g. to build a 3D-printed model of the scanned object.

VI. Experiments and Results

The algorithm and architecture have been validated using the materials described in Section III. Table I groups the most relevant configuration parameters pertaining to the scene reconstruction service, the RGB-D frame acquisition server, and the trajectory generator embedded in the main application, respectively. The trajectory radius accounts for the maximum reach of the CRB 15000-5, while the work envelope of $40 \times 25 \times 40$ centimeters is expressed in XYZ camera coordinates, where Y translates to the object's height.

TABLE I: Selection of relevant parameters.

Scene reconstruction parameters (KinectFusion)	
depth sigma (bilateral smoothing)	0.01 [m]
pre-scale factor (per 1 meter)	1000
ICP distance threshhold	0.01 [m]
threshold for depth truncation	2.5 [m]
TSDF truncation distance	0.01 [m]
volume pose (translation XYZ)	(-0.2, -0.15, 0.2) [m]
voxel size	0.001 [m]
resolution of voxel space	(400, 250, 400)
RGB-D camera parameters	
width	640 [px]
height	480 [px]
frames per second	30
Trajectory parameters (main application)	
centroid (XYZ)	(600, 0, 80) [mm]
radius	350 [mm]
height	105 [mm]
inclination	0 [deg]
resolution (points)	200
speed	40.0 [mm/s]
start angle	-179 [deg]
end angle	179 [deg]

After initial configuration, the Cartesian trajectory is constructed using the aforementioned parameters. The kinematics solver computes joint values, represented as a time series in Figure 5. Joint limits and the actual maximum and minimum values are displayed on the plot and via terminal.

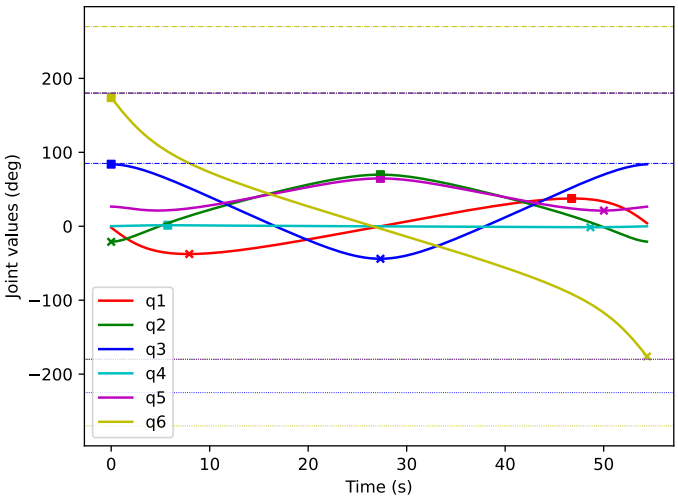


Fig. 5: Time series joint trajectory plot for all six robot axes.

The generated trajectory was validated using the ABB RobotStudio simulator. The resulting sequence of points, starting from an initial position above the center of the circular path, is represented in Figure 6. According to the configuration parameters, the robot brings down its end effector until it reaches the first point of the circle, using an end-to-end motion command. Afterwards, a high-frequency stream of commands is sent in regular intervals of 4 ms (250 Hz) to traverse the discrete sequence of points that constitute the circle. The camera is always kept facing the target object to be placed at its center. At the end, the robot returns to the initial position.

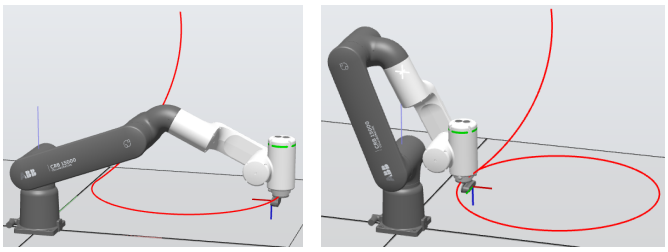


Fig. 6: Simulated trajectory using ABB RobotStudio.

Since the virtual controller is based on the same underlying kinematics as its real counterpart, the simulated experiment enabled validation of the generated trajectory prior to execution on the actual robot. After this initial offline validation, the real setup was prepared as depicted in Figure 7.

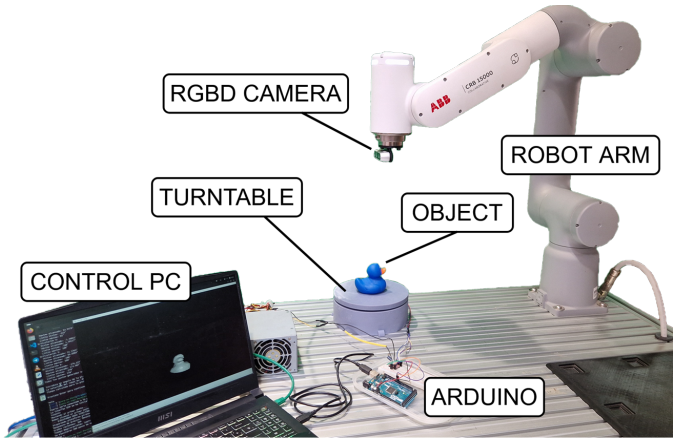


Fig. 7: Experimental setup and its main hardware components.

Figure 8 depicts the execution of the process. Once the first point of the circular path is reached, the KinectFusion algorithm begins real-time processing (on an RTX 4070 GPU) and a Phong-rendered model is displayed in the viewer tool, as seen on the computer screen. Simultaneously, the turntable begins to spin clockwise, while the robot draws a circle in mid-air in the opposite direction, always keeping the scanned target at the center, on the rotating platform. Since one revolution is performed by either mechanism, the target is scanned fully twice, from all its sides. At the selected tangential speed of the robot of 40 mm/s, the duration of the trajectory is 54 seconds.

The results of three successful experiments are shown in Figure 9. These objects have been selected according to the work envelope specified in the configuration parameters, which in turn depends on the robot's reachability: a rubber duck, a toy robot, and a human head. The duck and the robot were placed on the turntable and subject to two revolutions as perceived by the depth sensor, whereas the human head was kept off the turntable for practical reasons and received a single scanning pass (revolution). The resulting watertight mesh was obtained after applying the Poisson surface reconstruction algorithm [16], as implemented in the PCL library wrapped by our YARP-PCL bridge library [3], to the point cloud retrieved from the KinectFusion service, and stored to disk afterwards. Manual post-processing was required to produce a valid 3D-printing input mesh since there may be present some openings (i.e. open meshes) and floating elements. The open-source MeshLab tool served for this purpose. Finally, the resulting file was transferred to a BambuLab H2S 3D printer in order to prototype a 1:1 model with PLA filament.

To sum up, the printed prototypes adequately resemble the original models. The main inaccuracies have been identified in the depth camera's resolution, and the application of the Poisson algorithm to convert the point cloud into a mesh. The former introduced artifacts on otherwise smooth surfaces in the model. On the other hand, the latter tends to close the gaps between nearby elements; this is especially noticeable in the robot model due to the cavities present, e.g., between the legs, and between the arms and its torso. We believe the effects of both sources could be amended by a careful selection of the hardware (e.g. a depth sensor with improved resolution) and software (e.g. a different mesh reconstruction algorithm) tooling and parameters. Thanks to the flexibility of our architecture, users can extend or replace its individual building blocks to conform with their task requirements.

VII. Conclusions

We have presented a flexible, modular architecture and framework to fulfill a 3D scanning task using a robot arm, a depth sensor, and an optional rotating platform (turntable). Although minimal restrictions are placed on the hardware and software, we provide the tools and interfaces to overcome most limitations: replacing any hardware component or adhering to a different programming interface or standard boils down to replacing one of the basic building blocks of our architecture, or developing a new one. This approach empowers code reuse and leverages YARP and ROS 2 to benefit from a distributed architecture. Moreover, relying on ROS 2 for motion control paves the way for interoperability with a wide range of robots and tools.

Future work includes enhancing trajectory generation, especially in an autonomous fashion. The source code of all implemented components, particularly the scene reconstruction and mesh generation services[2] and the main application,[3] is publicly available on GitHub.

[2]https://github.com/roboticslab-uc3m/vision ("sceneReconstruction")
[3]https://github.com/roboticslab-uc3m/gofa-kinfu

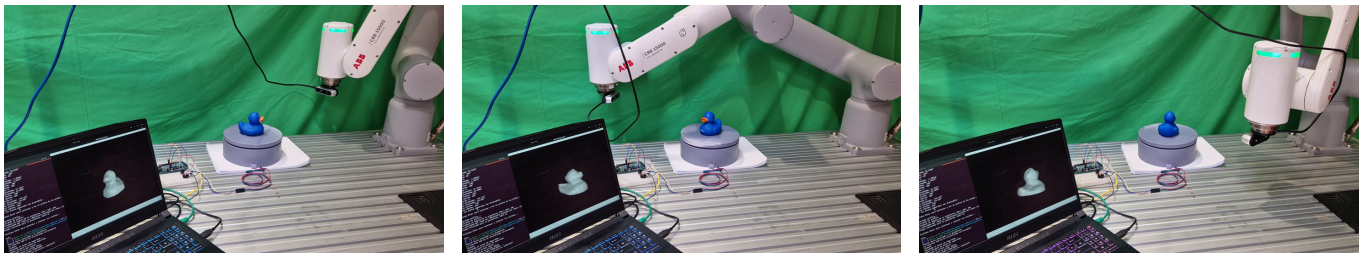


Fig. 8: Representation of an ongoing scene reconstruction process.



Fig. 9: Sample results of three scanned objects: a rubber duck (upper row), a toy robot (middle row), and a human head (bottom row). Each row represents the original object (left), the resulting 3D reconstruction as a mesh visualization (center), and its 1:1 3D-printed model (right).

ACKNOWLEDGMENTS

This research has been financed by "iRoboCity2030-CM, Robótica Inteligente para Ciudades Sostenibles" (TEC-2024/TEC-62) and "FotoArt5.0-CM, Laboratorios inteligentes para la ciencia del futuro" (TEC-2024/TEC-308), both funded by "Programas de Actividades I+D en Tecnologías de la Comunidad de Madrid"; "ROBOASSET: Intelligent robotic systems for assessment and rehabilitation in upper limb therapies" (PID2020-113508RB-I00, financed by AEI/10.13039/501100011033); "iREHAB: AI-powered Robotic Personalized Rehabilitation" (DTS22/00105, financed by Instituto de Salud Carlos III (ISCIII)); and EU structural funds.

REFERENCES

[1] J. Li, W. Gao, Y. Wu, Y. Liu, and Y. Shen, "High-quality indoor scene 3D reconstruction with RGB-D cameras: A brief review," Comp. Visual Media, 8(3), 2022, pp. 369–393. DOI: 10.1007/s41095-021-0250-8

[2] R. A. Newcombe et al, "KinectFusion: Real-time dense surface mapping and tracking," Int. Symposium on Mixed and Augmented Reality (ISMAR), IEEE, 2011, pp. 127–136. DOI: 10.1109/ISMAR.2011.6092378

[3] B. Łukawski, A. Rodríguez-Sanz, J. G. Victores, and C. Balaguer, "A user-friendly point cloud processing pipeline for interfacing PCL with YARP," Jornadas de Automática, CEA, 45, 2024. DOI: 10.17979/ja-cea.2024.45.10925

[4] B. Łukawski, M. Martín-Cervera, C. Menchén-Pascual, E. D. Oña, J. G. Victores, and A. Jardón, "Espresso macchiato, por favore: collaborative robotic coffee-making for education," Jornadas de Automática, CEA, 46, 2025. DOI: 10.17979/ja-cea.2025.46.12274

[5] B. Łukawski, E. D. Oña, J. G. Victores, C. Balaguer, and A. Jardón, "Assessing student satisfaction and system usability in industrial robotics education," Simposios del Comité Español de Automática, CEA, 2(1), 2026. DOI: 10.64117/simposioscea.v2i1.150

[6] M. Krainin, B. Curless, and D. Fox, "Autonomous generation of complete 3D object models using next best view manipulation planning," Int. Conf. on Robotics and Automation (ICRA), IEEE, 2011. DOI: 10.1109/ICRA.2011.5980429

[7] L. Torabi, and K. Gupta, "An autonomous six-DOF eye-in-hand system for in situ 3D object modeling," The Int. J. of Robotics Research, 31(1), 2012, pp. 82–100. DOI: 10.1177/0278364911425836

[8] M. Naazare, F. G. Rosas, and D. Schulz, "Online next-best-view planner for 3D-exploration and inspection with a mobile manipulator robot," IEEE Robotics and Automation Letters, 7(2), 2022, pp. 3779–3786. DOI: 10.1109/LRA.2022.3146558

[9] J. I. Vasquez-Gomez, L. E. Sucar, and R. Murrieta-Cid, "View planning for 3D object reconstruction with a mobile manipulator robot," Int. Conf. on Intelligent Robots and Systems (IROS), IEEE, 2014. DOI: 10.1109/IROS.2014.6943158

[10] H. Madhusudanan et al., "Automated eye-in-hand robot-3D scanner calibration for low stitching errors," Int. Conf. on Robotics and Automation (ICRA), IEEE, 2020. DOI: 10.1109/ICRA40945.2020.9196748

[11] B. Calli et al., "Yale-CMU-Berkeley dataset for robotic manipulation research," The Int. J. of Robotics Research, 36(3), 2017, pp. 261–268. DOI: 10.1177/0278364917700714

[12] N. Azhar, W. H. M. Saad, S. A. A. Karim, N. M. Saad, and N. Abd Manap, "3D model construction of turntable and robotic arm sequence," Journal of Telecommunication, Electronic and Computer Engineering (JTEC), 10(2-6), 2018, pp. 71–75. Corpus ID: 69638787

[13] G. Metta, P. Fitzpatrick, and L. Natale, "YARP: Yet Another Robot Platform," Int. Journal of Advanced Robotic Systems, 3(1), 8, 2006, pp. 43–48. DOI: 10.5772/5761

[14] M. Quigley et al., "ROS: an open-source Robot Operating System," ICRA workshop on open source software, 3, 2009. Corpus ID: 6324125

[15] B. Łukawski, E. D. Oña, A. Jardón, J. G. Victores, and C. Balaguer, "Development of educational applications with ABB GoFa collaborative robot using Externally Guided Motion," Int. Conf. on Control, Automation and Diagnosis (ICCAD), IEEE, 2025. DOI: 10.1109/ICCAD64771.2025.11099395

[16] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," Eurographics Symposium on Geometry Processing (SGP), 7(4), 2006, pp. 61–70. DOI: 10.2312/SGP/SGP06/061-070