Development and Analysis of a Distributed VSLAM System for Low-Cost Robotics Using ROS 2 and Docker

Filipi Enzo Siqueira Kikuchi

October 10, 2025

Abstract

This work aims to analyze and compare different feature extraction algorithms applied to RGB-D images captured by a Kinect V1 sensor, in order to identify the most suitable methods for low-cost mobile autonomous robot applications. By evaluating techniques such as ORB, SURF, and BRISK, we aim to understand each method's performance in terms of robustness, repeatability, and computational efficiency. This investigation supports the selection of viable algorithms for visual SLAM systems operating under budget and processing constraints.

1 Introduction

Autonomous navigation in mobile robots requires the real-time construction and interpretation of maps of the surrounding environment. Among the most common approaches to achieve this goal are Simultaneous Localization and Mapping (SLAM) systems, especially vision-based SLAM (Visual SLAM), which rely on image sensors to estimate the robot's trajectory while building a representation of the environment.

With advances in computer vision, the detection and tracking of keypoints (features) have become essential components in the functioning of Visual SLAM systems. These features are used to estimate the camera's motion and reconstruct the 3D space around the robot. The choice of the feature extraction algorithm directly affects the system's accuracy, robustness, and computational cost.

This work investigates different feature extraction methods applied to RGB-D data captured by a Kinect V1 sensor, aiming to identify which performs best for low-cost mobile robotics scenarios. Choosing efficient algorithms can benefit academic projects, educational platforms, or commercial low-investment solutions, broadening access to autonomous navigation technologies.

2 System Components and Justification

The selection of each component in this project was guided by the objectives of low cost, modularity, and use of modern robotics software practices.

The **Kinect V1** sensor is widely adopted in robotics projects for offering both RGB images and depth data, making it a low-cost alternative to professional 3D mapping sensors such as LiDARs. Although originally developed for entertainment applications, the Kinect's real-time depth capabilities make it well-suited for navigation experiments and small-scale 3D reconstructions.

The goal of this project is to use the Kinect as an experimental platform to compare different feature extraction algorithms — such as ORB, SURF, and BRISK — applied to RGB and RGB-D images, evaluating their efficiency and robustness. These algorithms are extensively used in SLAM systems and computer vision applications, each with distinct trade-offs in execution time, invariance properties, and noise tolerance.

By understanding the behavior of these algorithms in a controlled environment with a low-cost sensor, we aim to contribute to the identification of a general-purpose approach for feature extraction in robotic platforms constrained by budget and processing power. This contribution is particularly relevant to educational contexts, emerging research labs, and accessible automation initiatives.

The **Robot Operating System 2 (ROS 2)** was selected as the middleware framework. Its architecture, based on the Data Distribution Service (DDS) standard, is inherently designed for distributed systems, enabling seamless communication between the Raspberry Pi and the desktop computer over a standard network.

Docker was employed to create a containerized development environment. This approach ensures that all dependencies—from system libraries and 'libfreenect' to specific Python packages and the ROS 2 distribution itself—are encapsulated into a single, portable image. This guarantees environment reproducibility and simplifies deployment on different machines.

Finally, **RTAB-Map (Real-Time Appearance-Based Mapping)** was chosen as the core SLAM algorithm due to its excellent performance with

RGB-D data, its robust loop closure detection capabilities, and its seamless integration as a suite of packages within the ROS ecosystem.

3 Literature Review

We present the following comparative analysis of key research papers that support the understanding of modern Visual SLAM approaches and their applications in autonomous robotics. This review provides a foundation to support the methodological decisions taken in this project.

Title	What It Is	Goal	How the Goal Is Achieved
3D Local Map Construction Using Monocular Vision (2010)	Study using monocular vision with delayed initialization, SURF, and Kalman filter for localization.	Develop an efficient monocular SLAM technique to generate local 3D maps.	Uses SURF feature detection, 3D line-based depth estimation, and Extended Kalman Filter, avoiding complex parameter tuning.
ORB-SLAM (2015)	A real-time monocular SLAM system based on ORB features, robust to viewpoint changes.	Propose a robust and accurate monocular SLAM system for varied environments.	Uses ORB features throughout the pipeline (tracking, mapping, loop closure), with graph-based structure and automatic initialization.
Direct Sparse Odometry (2016)	Monocular odometry system based on direct photometric error and sparse optimization.	Develop an accurate and real-time visual odometry method.	Joint optimization of camera parameters, depth, and pose using gradient-rich sparse points.
ExplORB-SLAM (2022)	Active extension of ORB-SLAM2 using pose graph structure for exploration.	Optimize SLAM performance using utility-based navigation (D-optimality).	Detects frontiers, predicts graph expansion through "hallucination", and selects motion based on expected utility.
Tightly-Coupled LiDAR-Visual SLAM (2023)	SLAM system integrating LiDAR and monocular camera data using geometric features.	Create a robust and accurate SLAM system with low-cost sensors.	Fuses data in spherical coordinates; visual subsystem refines depth, LiDAR adjusts feature direction; fallback mechanism ensures robustness.

The progression of these works clearly highlights the trend:

- From purely geometric and monocular solutions to multimodal sensor fusion approaches;

- From point-only features to the inclusion of line and semantic features;
- From passive observation to active utility-guided navigation;
- And from high-cost sensors to accessible alternatives — such as Kinect-based RGB-D cameras.

From this table, we can observe how Visual SLAM has evolved over time, with the emergence of new techniques and approaches aimed at improving the accuracy, robustness, and efficiency of these systems.

Due to the continuous and long-term operation of agents that rely on SLAM-based navigation in various environments, concerns about redundancy in sensor readings become a key focus in papers proposing robust solutions. In this context, more recent works tend to adopt a hybrid approach, combining geometric analysis, active perception strategies, and semantic understanding, in addition to the fusion of data from multiple sensors such as LiDAR and monocular cameras. This fusion aims to enhance robustness by achieving higher reliability in perception, relocalization, and loop closure.

Among the evaluation metrics used in SLAM system analysis, techniques involving photometric and geometric error optimization—such as Root Mean Square Error (RMSE)—are the most common. These are often accompanied by real-time performance evaluations and assessments of system resilience in dynamic environments.

4 Methodology and System Development

The construction of the VSLAM system followed a structured methodology, encompassing system architecture design, environment containerization, custom node development, sensor calibration, and final integration.

- **Sensor Node (Raspberry Pi):** Its sole responsibility is to interface with the Kinect V1 sensor, capture RGB and depth images, and publish them as ROS 2 messages onto the network. This minimizes its processing load.
- **Processing Node (Desktop):** This machine subscribes to the image topics from the network. It runs the computationally intensive nodes: the calibration manager, the RTAB-Map SLAM and odometry algorithms, and the visualization tools (RViz).

4.1 Containerized Environment Setup

A Dockerfile was created to automate the entire environment setup, ensuring reproducibility. The image is based on ‘ros:jazzy-ros-base‘ and performs the following key steps:

1. Installs all system dependencies via ‘apt-get‘, including ‘libfreenect-dev‘, ‘python3-yaml‘, and the ‘ros-jazzy-rtabmap-ros‘ and ‘ros-jazzy-camera-calibration‘ packages.
2. Creates a ROS 2 workspace at ‘/root/ros2_ws‘.
3. Copies the custom ‘kinect_camera‘ package from the host machine into the image’s workspace.
4. Compiles the workspace using ‘colcon build‘.
5. Sets an ‘ENTRYPOINT‘ to automatically source the ROS 2 and workspace setup files, providing a ready-to-use bash terminal.

The container is launched with specific arguments to grant it access to the host’s USB devices (‘–privileged‘, ‘–device=/dev/bus/usb‘) and X11 server for displaying graphical interfaces like RViz.

4.2 Custom ROS 2 Node Development

Since no standard ROS 2 driver was readily available that fit our exact needs, two custom Python nodes were developed.

kinect_publisher.py This node acts as the bridge between the ‘libfreenect‘ library and ROS 2. Its main loop captures synchronized video and depth frames, converts them to ‘sensor_msgs/Image‘ messages, and publishes them on ‘/kinect/rgb/image_raw‘ and ‘/kinect/depth/image_raw‘ topics. Crucially, it sets the ‘header.frame_id‘ to ‘”camera_link”‘ and uses the ‘qos_profile_sensor_data‘ QoS profile for compatibility with real-time subscribers.

calibration_manager.py This node handles all aspects of camera calibration. It publishes the ‘sensor_msgs/CameraInfo‘ message required by RTAB-Map by loading parameters from a ‘.yaml‘ file. It also provides the ‘/kinect/rgb/set_camera_info‘ service, allowing the ‘camera_calibration‘ tool to update the parameters on-the-fly. This modular design separates the hardware-driving logic from the calibration data management.

4.3 Camera Calibration and SLAM Execution

The final pipeline requires a precise sequence of operations.

1. **Static Transform Publication:** A ‘static_transform_publisher‘ from the ‘tf2_ros‘ package is launched to continuously publish a transform from a ‘base_link‘ frame to the ‘camera_link‘ frame. This establishes the geometric foundation of our robot’s coordinate system.

2. **Node Execution:** The custom ‘kinect_publisher‘ and ‘calibration_manager‘ nodes are executed.

3. **VSLAM Launch:** Finally, the main RTAB-Map launch file is executed using ‘ros2 launch‘. The command includes critical parameters and remappings determined through our development process:

    - ‘approx_sync:=true‘: To handle potential minor timestamp misalignments from the Kinect V1.
    - ‘rgb_topic‘, ‘depth_topic‘, ‘camera_info_topic‘: Remapped to the topics published by our custom nodes.
    - ‘frame_id:=”camera_link”‘: To inform RTAB-Map of the primary reference frame for incoming data.

This sequence brings the entire system online, allowing RTAB-Map to receive all necessary data streams (RGB, Depth, CameraInfo, and TF) to begin the mapping and localization process, visualized through RViz and the RTAB-Map GUI.

4.4 Robot Testing

After all the calibration and mapping, the primary tests occurred using equipment from Inteli’s Maker Laboratory, which is filled with a range of equipment, ranging from cheap sensors and microcontrollers to fully commercial robots.

For the first implementations, the team focused on mapping tests throughout the lab using local computers for this purpose. The focus was to move the Kinect around on a chair to test the processing capacity of the devices and the quality of the map built using the computer vision algorithm. We first looked to generate an accurate mapping of the environment.

For a rapid comparison, we tested the accuracy of generating a map using a Dell G3 15 and the lab’s high-performance computer, which has extremely high specifications compared to the first PC. The computer specifications can be checked in Table 1.

Table 1: Hardware Comparison

Hardware Comparison		
Component	**Dell G3**	**Lab Computer**
CPU	Intel Core i5 9th Gen	–
GPU	NVidia GeForce 1050	Nvidia A10

The results from the first mapping tests are detailed in Table 2.

Table 2: Results for the First Tests

Results		
Item	**Dell G3**	**Lab Computer**
Frame rate processing	2 frames/s	15 frames/s
Time for building the map	3 minutes	1 minute
Error from position A to B (meters)	3 meters (±50 cm)	5 cm (±2 cm)

First Conclusions

The map can be built on a considerably outdated personal computer with somewhat limited resources available, but it takes more time and is more error-susceptible. On the other hand, using a powerful computer driven by heavy data processing, it was seemingly easy to build such a map. The error reduced drastically, and the overall quality of the generated map for navigation was perceptibly better.

Recommendations

Using something in between the two tested machines may provide the best cost-benefit for generating high-quality maps for a low-budget purpose.

References