3D Indoor Mapping Using ROS and Microsoft Kinect Sensor

Chirag Shah and Srijal Poojari
Project Guide: Kumar Khandagle

***Abstract*—The objective of this project is to create a 3D map of a given environment using a depth camera (Microsoft Kinect) and send this information over a Wi-Fi network to a laptop. This will enable mounting of the camera on a quadcopter and create 3D map of an indoor environment.**

We used the ROS framework running on a laptop and a Raspberry Pi, a Microsoft Kinect, RTAB-Map package for ROS for constructing the 3D map and freenect package for interfacing Kinect with ROS.

***Index Terms*—ROS, Robot Operating System, 3D-Mapping, Microsoft Kinect Sensor, Wireless Mapping, RTAB-Map, Raspberry Pi (RPi)**

I. Introduction

AUTONOMOUS robots operating in real life settings must be able to navigate in large, unstructured, dynamic and unknown spaces. To do so, they must build a map of their operating environment in order to localize itself in it [1]. We intend to mount the kinect sensor on a quadcopter and create a 3D map of an indoor environment.

We connected the Microsoft Kinect sensor to the RPi and interfaced it using the freenect package; Using the RGB-D image from the kinect sensor, RPi genrates a 3D map using RTAB-Map. RPi then transmits the 3D map to the laptop over a Wi-Fi network. RViz is used to visualize the 3D map on the laptop.

The paper is organized as follows. Section II describes the major components of our project. Section III describes the design of our wireless mapping setup. Section IV shows the resulting the map that is generated. Section V concludes the paper.

II. Major Components of Our System

A. Robot Operating System

The Robot Operating System (ROS) is a framework for writing robot software. It is a collection of tools, libraries and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms [2, p. 3]. It is not a programming language [3].

A ROS system is comprised of a number of independent nodes, each of which communicates with the other nodes using a publish/subscribe messaging model. For example, a particular sensors driver might be implemented as a node, which publishes sensor data in a stream of messages. These messages could be consumed by any number of other nodes. Note that nodes in ROS do not have to be on the same system or even systems of the same architecture. This makes ROS really flexible and adaptable to the needs of the user. The ROS framework is open source. ROS starts with the ROS Master. The Master allows all other nodes to find and talk to each other [4].

1) roscore: roscore is a service that provides connection information to nodes so that they can transmit messages to one another. Every node connects to roscore at startup to register details of the message streams it publishes and the streams to which it wishes to subscribe. When a new node appears, roscore provides it with the information that it needs to form a direct peer-to-peer connection with other nodes publishing and subscribing to the same message topics. Every ROS system needs a running roscore, since without it, nodes cannot find other nodes. With knowledge of the location of roscore on the network, nodes register themselves at startup with roscore and then query roscore to find other nodes and data streams by name. Each ROS node tells roscore which messages it provides and which it would like to subscribe to. roscore then provides the addresses of the relevant message producers and consumers [2, pp. 11-12].

2) Nodes: A node is a process that performs computation. Nodes are combined together into a graph and communicate with one another using streaming topics, RPC services, and the Parameter Server. These nodes are meant to operate at a fine-grained scale; a robot control system will usually comprise many nodes. For example, one node controls a laser range-finder, one Node controls the robot's wheel motors, one node performs localization, one node performs path planning, one node provide a graphical view of the system, and so on.

The use of nodes in ROS provides several benefits to the overall system. There is additional fault tolerance as crashes are isolated to individual nodes. Code complexity is reduced in comparison to monolithic systems. Implementation details are also well hidden as the nodes expose a minimal API to the rest of the graph and alternate implementations, even in other programming languages, can easily be substituted. A ROS node is written with the use of a ROS client library, such as roscpp or rospy [5].

3) Topics: A topic is a name for a stream of messages with a defined type. Topics implement a publish/subscribe communication mechanism, one of the more common ways to exchange data in a distributed system. Before nodes start to transmit data over topics, they must first announce, or advertise, both the topic name and the types of messages that are going to be sent. Then they can start to send, or publish, the actual data on the topic. Nodes that want to receive messages on a topic can subscribe to that topic by making a

request to roscore. After subscribing, all messages on the topic are delivered to the node that made the request. In ROS, all messages on the same topic must be of the same data type [2, pp. 31].

B. Microsoft Kinect

Microsoft Kinect is a RGB-D camera. Kinect uses an RGB camera with depth sensor and infrared projector with a monochrome CMOS sensor. The Kinect functions by covering the room with a constant, predetermined pattern of infrared dots. The monochrome CMOS sensor is placed at an offset relative to the IR transmitter, and the difference between the observed and expected IR dot positions is used to calculate the depth at each pixel. This gives a depth image of the surroundings. This depth image is super imposed with the RGB image to obtain a RGB-D image.

C. RTAB-Map

RTAB-Map (Real-Time Appearance-Based Mapping) is a RGB-D Graph-Based SLAM approach based on an incremental appearance-based loop closure detector. The loop closure detector uses a bag-of-words approach to determinate how likely a new image comes from a previous location or a new location. When a loop closure hypothesis is accepted, a new constraint is added to the maps graph, then a graph optimizer minimizes the errors in the map. A memory management approach is used to limit the number of locations used for loop closure detection and graph optimization, so that real-time constraints on large-scale environments are always respected. RTAB-Map can be used alone with a hand-held Kinect or stereo camera for 6DoF RGB-D mapping, or on a robot equipped with a laser rangefinder for 3DoF mapping [6].

D. Raspberry Pi

We are using Raspberry Pi 2 Model B along with a WiFi adapter. Higher versions of Raspberry Pi would provide better performance.

III. System Design

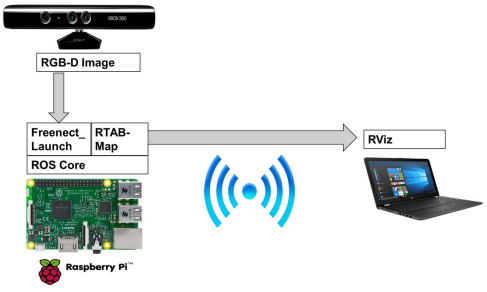

Fig. 1. System Design

The OS installed on RPi is Ubuntu Mate, V14.2. ROS Kinetic version 2.3 is installed on the RPi. The Kinect provides the RPi with a RGB-D image. Freenect_launch package is used for interfacing Kinect with ROS on the RPi (Openni_launch can also be used for interfacing the kinect). RTAB-Map package is used on the RPi for constructing a 3D map from the image provided by the Kinect.

The roscore runs on the Raspberry Pi. This roscore is common to the laptop which is on the same network. Thus all the map/sensor data is available to the laptop over the WiFi network. Rviz is used for visualizing the map on the laptop. Similarly rtabmapviz can be used for visualizing the map.

IV. Results

A frame rate of 20-25 fps is obtained when the Kinect is connected to the laptop, RTAB-Map does the mapping and RTAB-Mapviz/RViz visualizes the map.

3-4 fps is obtained when the kinect is connected to the Raspberry Pi and mapping is done by it and the map is visualized on the laptop. The data stream from the Kinect has to be divided by two to allow the RPi to process it fast enough. RPi requires 80-100kbps bandwidth for transmitting the map over the network to the laptop.

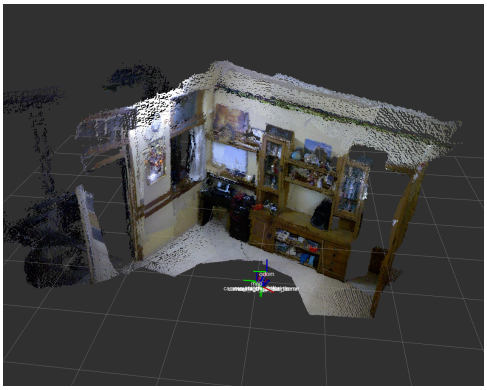

Fig. 2. Mapping with kinect connected to a laptop

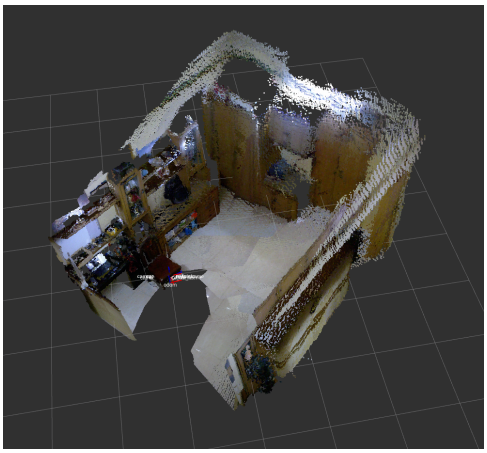

Fig. 3. Mapping with kinect connected to a laptop

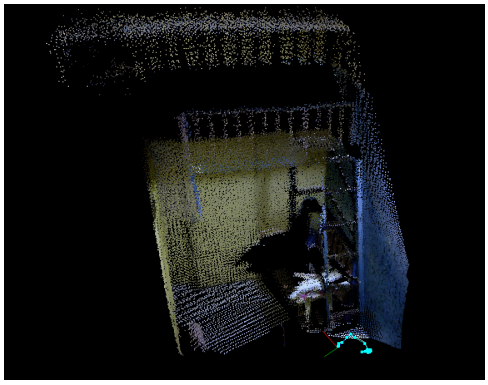

Fig. 4. Wireless Mapping with kinect connected to a Raspberry Pi 2

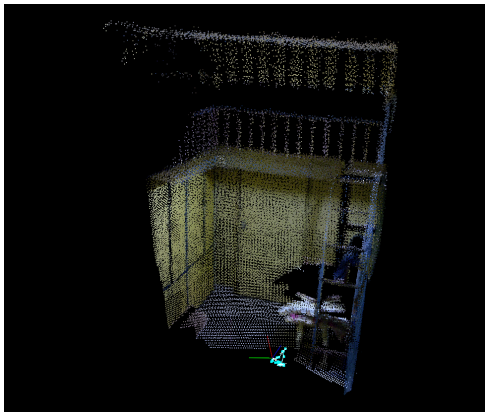

Fig. 5. Wireless Mapping with kinect connected to a Raspberry Pi 2

V. Conclusion

We were able to create a mobile system to generate a 3D map using the ROS framework on RPi and a Kinect sensor. RPi provides a sufficient throughput for computing a 3D Map at 4-5 fps and transmit the same over a Wi-Fi.

This project enabled us to understand how the ROS framework works and how different packages integrate with each other.

Acknowledgment

We would like to thank out mentor Kumar Khandagle for his support and guidance.

We would also like to thank our college Sardar Patel Institute of Technology for providing us with a platform for us to showcase our project.

References

[1] M. Labb and F. Michaud, "Memory management for real-time appearance-based loop closure detection," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 12711276.

[2] Morgan Quigley, Brian Gerkey, William D. Smart, *Programming Robots with ROS*. 2016, p. 3.

[3] Po Jen Lai, "What is ROS? [Online]. Available: https://www.quora.com/What-is-ROS [Accessed: 31- Oct- 2017].

[4] Zeeshan Khan S, "What is ROS?" [Online]. Available: https://www.quora.com/What-is-ROS [Accessed: 31- Oct- 2017].

[5] ROS.org, "Nodes." ROS wiki [Online]. Available: http://wiki.ros.org/Nodes

[6] Mathieu Labb, "Overview on RTAB-Map." [Online]. Available: http://introlab.github.io/rtabmap/ [Accessed: 31- Oct- 2017].