3D Reconstruction Using Kinect and RGB-D SLAM

Shengdong Liu, Pulak Sarangi, Quentin Gautier

June 9, 2016

Abstract

Visualization is a powerful technique to reinforce human cognition, and archaeologists uses it extensively to study and showcase artifacts and architectures. Currently, archaeologists create visualization using drawings, which is inefficient especially visualizing big artifacts and architectures. This project aims to provide archaeologists with a mobile system that can reconstruct an artifact with real-time feedback. Our system consists of a Microsoft Kinect sensor to scan the artifact and a laptop to run RGB-D SLAM, which generate a 3D point cloud of artifact. After some field testing in Guatemala, we concluded that while the portability of the setup needs to be improved, the system is stable and it can generate a 3D representation of up to 25 meters of a tunnel in 20 minutes.

1 Introduction

As technology advances, we have access to newer and faster ways to complete a task. This project aims to apply the technology we have in the field of archeology, specifically in visualizing large artifacts and architectures. Visualization helps archaeologists to document their findings to be studied off-site, shared with other archaeologists, and presented to interested audience, thus it is important for archaeologists to have a reliable way to create accurate visualization of their finding. Currently, the most common way for archaeologists to create visualization is to draw the artifact by hand. This approach, although effective, takes a long time. For big artifacts and architectures, a painstaking effort is needed to create a detailed and accurate 3D representation by hand. The 3D reconstruction project seeks to tackle this problem using a mobile system that can reconstruct an object in 3D with real-time feedback. This system will help archaeologists create 3D visualizations of the artifacts or architectures at a much faster pace.

The 3D reconstruction project is supervised by Quentin Gautier, a member of Engineer for Exploration group at University of California San Diego. The group strive for engineering solutions that extend beyond technology itself to drive the future of exploration. They have projects in aquatic, terrestrial and aerial environments, as well as international collaborators in ecology, conservation and archeology[2]. Through Engineer for Exploration, we collaborate with archaeologists in an effort to 3D reconstructs an underground environment of Mayan architectures in Guatemala.

The main components of our system are a Microsoft Kinect Senor and a laptop. The Kinect sensor is a web-cam style camera with added infrared and other peripherals. The Kinect services support depth image, RGB-image, tilt, microphone array, and skeleton tracking[3]. For our project, we use the RGB-image and depth image information to create the 3D point cloud to visualize an artifact or architecture. We also use a laptop equipped with i7 processor and graphics processing unit (GPU), which enables smooth rendering of the visual feedback.

For creating the 3D representation of the object, we use RGB-D SLAM, which is a simultaneous localization and mapping algorithm that uses both the RGB image and the depth sensor to generate an accurate point cloud[4]. The advantage of using a SLAM algorithm is that as the mapping takes place, the algorithm also calculates the position of the camera relative to the map generated. This feature allows the algorithm to dynamically correct drift and fix inaccuracies in the point cloud.

This paper broken down into two major sections. The Technical Material section aim to walk the reader through of the require equipments, the setup steps and the usage of our system. At the end of the section, we also discuss some of the results from our own test and field test in Guatemala, as well as some of the issues our system face. The milestone section shows our project timeline. This include milestones we had set for this project on a weekly basis and what milestones we were able to complete.

2 Technical Material

2.1 3D Reconstruction System

This section focus on how to put the 3D reconstruction system together. We will list the components needed for the system and walk through the setup step go get RGB-D SLAM up and running.

2.1.1 System Requirements

For this system, we have the following components:

- Microsoft Kinect sensor
- Laptop (prefer GNU on board but it is not necessary)
- External battery and power converter to power the Kinect.

2.1.2 Set Up Instructions

The work environment for this project is Ubuntu[5] with C++. We also need the following open source projects, ROS[6], OpenCV[7], OpenGL[8], PCL[?], OctoMap[9], SiftGPU[10], and g2o[11] since they are dependencies of RGB-D SLAM[4]. We will list the resources we use to set up our system in this section.

1. **Install Ubuntu 14.04 Trusty Tahr**. Ubuntu is an open source operating system popular among software developers, and it is the OS we used to set up this project. Here is a link to install Ubuntu 14.04: `http://howtoubuntu.org/how-to-install-ubuntu-14-04-trusty-tahr`
2. **Install ROS Indigo Igloo.** ROS is an open-source robotics framework that uses publisher-subscribers to communicate information. the RGB-D SLAM algorithm is created in this framework, thus having the ROS frame before installing the RGB-D SLAM is essential. We highly recommend ROS Indigo because it is the latest stable vision. You can find the installation instructions here: `http://wiki.ros.org/indigo/Installation/Ubuntu`
3. **Install OpenCV 2.4** OpenCV is an open-source computer vision library that RGBD-SLAM depends on. We recommend OpenCV 2.4 because ROS frames work set OpenCV 2.4 as the default version of OpenCV to look for. Thus installing OpenCV 2.4 will save a lot of trouble linking dependencies. You can find the source code on `http://opencv.org/downloads.html` and installation instruction on `http://docs.opencv.org/2.4/doc/tutorials/introduction/linux_install/linux_install.html`.
4. **Install RGB-D SLAM.**. Now you should be able to setup and compile RGB-D SLAM, the mapping algorithm we are using. You can find the source code and instruction for indigo version on `https://github.com/felixendres/rgbdslam_v2`.
5. **Install any missing library.** Most of the aforementioned dependencies comes with either ROS or OpenCV, but you might still need to install libraries manually. Check RGB-D SLAM compilation message to see if your system is missing any library.
6. **Install OpenNI.** This open-source project can be use to visualize the RGB-D SLAM during run time. To install from commandline: $ sudo apt-get install ros-indigo-openni-launch

2.1.3 RGB-D SLAM Parameters

This section, we will go over some of the RGB-D SLAM parameters. Tuning these parameter can optimize the algorithm for an specific setting. There are three columns in the table presented below, they are RGB-D parameter name, the value we use for our setup, and the description of the parameter.

Parameter	**Value**	**Description**
topic_image_mono	std::string("/camera/rgb/image_rect_color")	Color or grayscale image of the environment.
camera_info_topic	std::string("/camera/rgb/camera_info")	Required for backproject if no pointcloud topic.
topic_image_depth	std::string("/camera/depth_registered/sw _registered/image_rect_raw")	Depth image of the environment.
bagfile_name	std::string("")	Read data from a bagfile, make sure to enter the right topics above.
cloud_creation_skip_step	static_cast<int> (2)	Downsampling factor (rows and columns, so size reduction is quadratic) for the point cloud.
create_cloud_every_nth_node	static_cast<int> (10)	Create a point cloud only for every nth frame. Increase this parameter if you dont need a dense point cloud.
feature_detector_type	std::string("ORB")	SIFTGPU, SURF or ORB
feature_extractor_type	std::string("ORB")	SIFTGPU, SURF or ORB.
matcher_type	std::string("FLANN")	SIFTGPU or FLANN or BRUTEFORCE.
detector_grid_resolution	static_cast<int> (3)	detect on a 3x3 grid (to spread ORB keypoints and parallelize SIFT and SURF)
max_keypoints	static_cast<int> (600)	Extract no more that this many keypoints. Increase this to stitch the point cloud more accurately at the cost of slower run time.
max_matches	static_cast<int> (300)	Keep the best n matches. Increase this to stitch the point cloud accuracy at the cost of slower run time. Should be lower than max_keypoints.
min_sample_candidates	static_cast<int> (4)	Frame to Frame comparison to random frames for big loop closures.
predecessor_candidates	static_cast<int> (4)	Frame to frame comparisons to sequential frames.
neighbor_candiates	static_cast<int> (4)	Frame to frame comparisons to graph neighbor frames.
ransac_iterations	static_cast<int> (300)	Number of iterations for registration. Increase this to stitch the point cloud accuracy at the cost of slower run time.
cloud_display_type	static_cast<std::string>("POINTS")	POINTS, TRIANGLE_STRIPE. Drastically affect rendering time.
pose_relative_to	std::string("largest_loop")	optimize only a subset of the graph: "largest_loop" for everything from the earliest matched frame to the current one, "first" for full graph, "inaffected" for frames that were matched.
backend_solver	std::string("pcg")	"pcg" is faster and good for continuous online optimization, "cholmod" and "csparse" are better for offline optimization.
optimizer_skip_step	static_cast<int> (300)	Optimize only every n-th frame. Increase this value to reduce computation during run time. Also this process slows down over time, so use wisely.

2.2 System Usage

In this section, we will cover how to use the system.

To launch RGB-D SLAM with default parameters, you can ros-launch script at the commandline by typing in:

```
$ roslaunch rgbdslam rgbdslam.launch
```

To improve tweak experience, we created an custom launch file with fine tuned parameter as well as an shell script to run RGB-D SLAM and record the camera data at the same time. This way, the archaeologist could launch the system with a simple command:

```
$ ./launch rgbdslam [bagfile_name]
```

and they will get real time feedback of their reconstruction progress as well as saving the camera data for post processing.

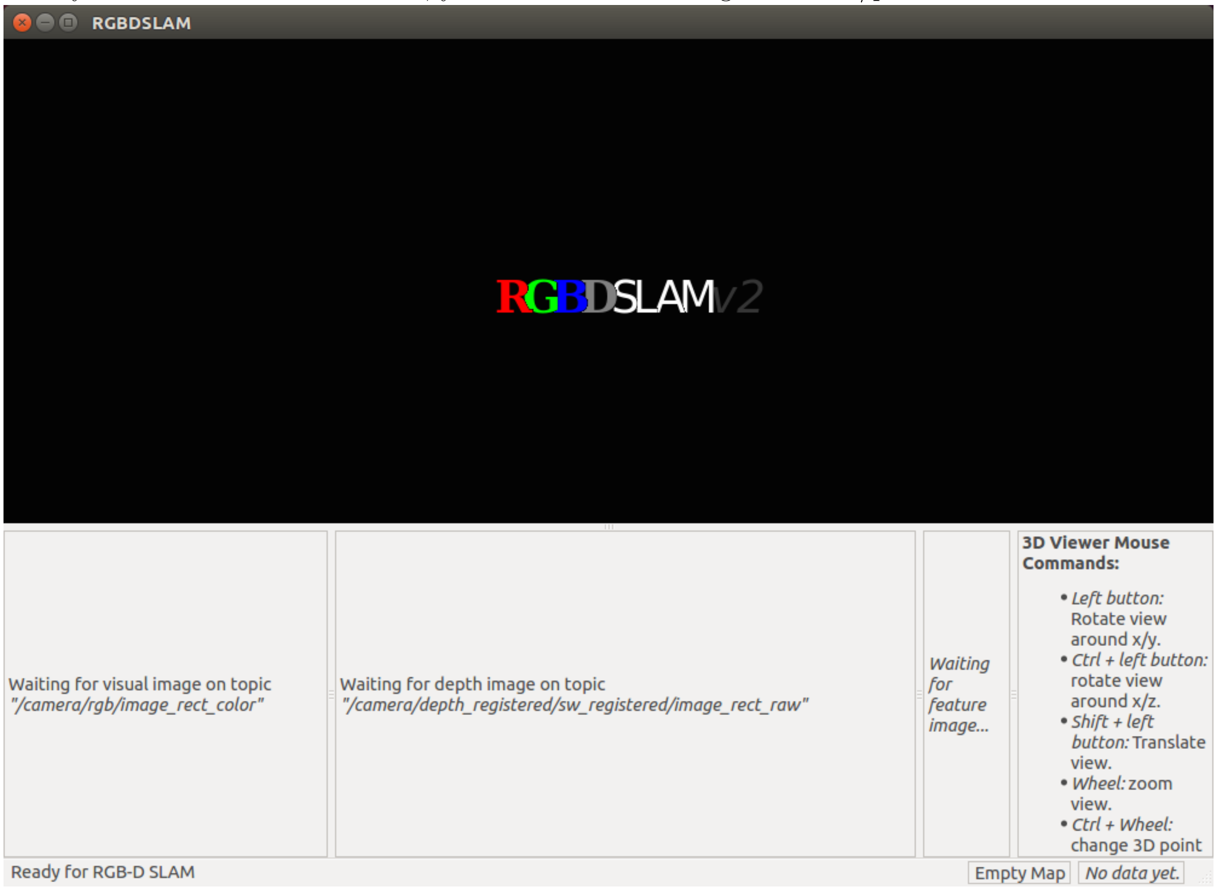
You can find the source code for our launch file at:

`https://bitbucket.org/cse145237d3dreconstruction/3d-reconstruction/src/2041ced88c51465708d8e32cb05b581b57dfba23/rgbdslam_v2/launch/rgbdslam+record.launch?at=kinect&fileviewer=file-view-default`

and the shell script at:

`https://bitbucket.org/cse145237d3dreconstruction/3d-reconstruction/src/2041ced88c51465708d8e32cb05b581b57dfba23/launch_rgbdslam.sh?at=kinect&fileviewer=file-view-default`

Once you launch the RGB-D SLAM, you will see the following interface: /par



Waiting for visual image on topic *"/camera/rgb/image_rect_color"*

Waiting for depth image on topic *"/camera/depth_registered/sw_registered/image_rect_raw"*

Waiting for feature image...

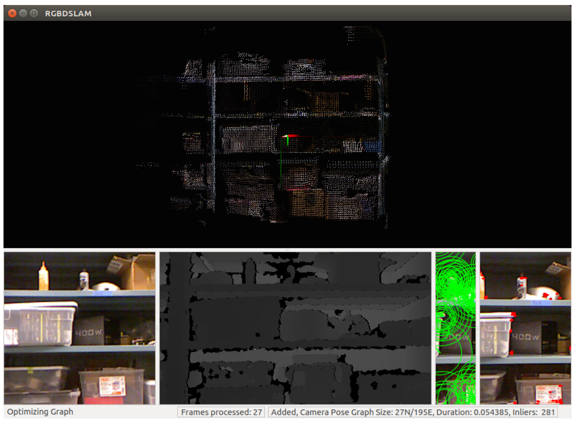
3D Viewer Mouse Commands:

- *Left button:* Rotate view around x/y.
- *Ctrl + left button:* rotate view around x/z.
- *Shift + left button:* Translate view.
- *Wheel:* zoom view.
- *Ctrl + Wheel:* change 3D point

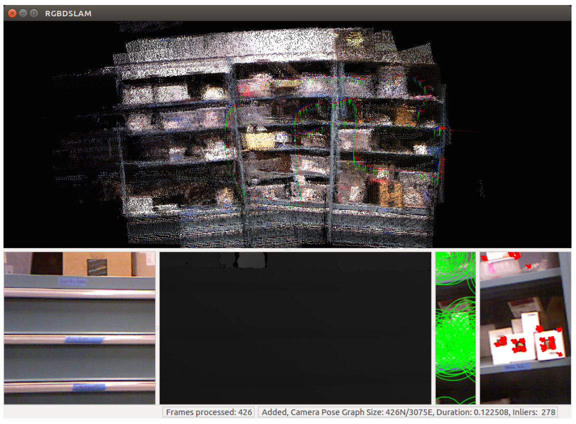
Ready for RGB-D SLAM

Empty Map *No data yet.*

The top half of the screen is where the point cloud will be displayed. The four panels on the bottom will display color image, depth image, feature matching, and corner detection in that order. As the images below shows.

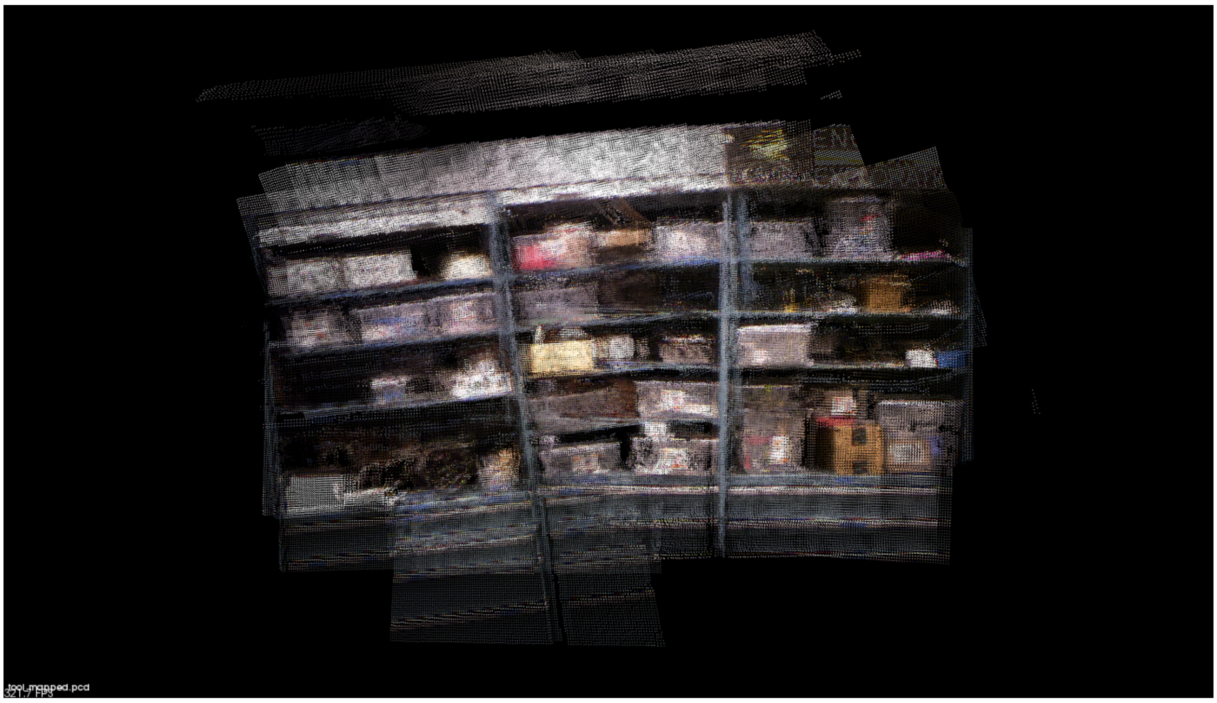


```
Frames processed: 27 | Added, Camera Pose Graph Size: 27N/195E, Duration: 0.054385, Inliers:  281
```



```
Frames processed: 426 | Added, Camera Pose Graph Size: 426N/3075E, Duration: 0.122508, Inliers:  278
```

In these example images we mapped a shelf of tool boxes. The image on the left is what the camera sees initially, and the image on the right is the result of mapping the shelf. The point cloud can then be saved and visualized again in an 3D point cloud viewer, such us pcl_viewer.



2.3 Test Result

We Tested our devices against the tool shelf, as the shelf is in an plan yet has a lot of texture because of all the different boxes it holds. We believe this set has the most similar to that of an tunnel with Mayan architecture we can find within our reach. As the images in the previous section shows, our system can accurate create a 3D representation of the shelf while provide feedback in real time.

Quentin, who took our 3D reconstruction system using Kinect and RGB-D SLAM to field test in Guatemala, told us that the Kinect“turn out to be pretty reliable”. It did not lose track of the environment often, and when it did, it was able to recover nicely. However, Quentin does advise us to avoid sudden movements to minimize the chance the algorithm gets lost. The algorithm slows down over time because it keeps track of a growing point cloud. Even then, the algorithm is decently fast for a mapping algorithm, and definitely faster than drawing an archaeological site by hand. Quentin was able to map out a roughly 25-meter tunnel in about 20 minutes.

One issue we encountered was portability. We initially envisioned our system to have only a Kinect sensor plugged into a laptop. However, Kinect is quite power hungry and needs its own energy source independent of the laptop. This means we have to add external battery to power the Kinect as well as power converter cables. These added components make the mapping process difficult to perform with a single person. Therefore, we would like to improve on portability of our system in the future.

3 Milestones

This project was completed in the course of ten weeks as an class project, thus well will include our timeline and the completion status of all the milestone we set. In the following table there are three columns: week number, milestone, and comments on the completion status. Notice there are no milestones for the first two weeks because we put together our initial milestones during week 3. (*Note these milestones are not necessarily completed in that order, but for simplicity, we ordered them according to our original milestones.)

Week	Objective	Completion Status
3	Complete Project Specifications	Completed:Project Spec
3	Set Up Kinect	Completed:Setup Instruction
4	Set Up Project Webpage	Completed:Project Webpage
4	Get Data From Kinect Sensor	Completed:RGB-D SLAM Video Demo
4	Choose the most fitting SLAM Algorithm	Completed:RGB-D SLAM
5	Oral Project Update	Completed:Update Presentation Slides
5	Test Kinect functionality and capabilities	Completed:RGB-D SLAM Video Demo
6	Launch RGBD-SLAM on a computer	Completed:Run Instruction
6	Test SLAM using bagfiles	Completed:Bagfile Video Demo
7	Milestone Report	Completed:Milestone Report
7	Test Mapping with SLAM in real time	Completed:RGB-D SLAM Video Demo
7	Formulate a controlled test for Kinect vs Tango results	Completed:SLAM Benchmarks
8	Optimize SLAM Parameters	Completed:Launch file with custom parameters
8	Controlled Kinect testing with SLAM.	Not completed: We put most of our time during week 8 and 9 to tune for the optimal parameters to achieve stability in the algorithm, getting ready for the field test in Guatemala on week 10
9	Compare result from Kinect and Tango	Not completed: We put most of our time during week 8 and 9 to tune for the optimal parameters to achieve stability in the algorithm, getting ready for the field test in Guatemala on week 10
9	Try Implementing RTAB	Not completed: This idea was original brought us as an mean to compare our system with the Tango system. However, we scrap this task in the end in favor of making our system more stable and more user friendly for field testing.
10	Final Presentation	Completed:Final Presentation Slides

Although we didn't get an chance to create a rigorous test comparing our system against an Tango, our field test result shows that Kinect is more stable. Quentin told us that the Tango devices is prone to crash when the architecture to map is large, whereas the Kinect system does not suffer from crashing. Thus, our project to reconstruction environment in 3D using a Kinect and RGB-D SLAM is recommend for mapping large architectures or when portability is not an issue.

4 Conclusion

Visualization help archaeologists bring their finding off-site to study and showcase and currently it takes a long time to create those visualization as they are usually hand drawn. The our project, 3D reconstruction using Kinect and RGB-D SLAM, presents a efficient and stable way of reconstructing an 3D environment for archaeologists. Through collaborating and with Engineer for Exploration, we are able to conduct so field test in tunnels in Guatemala. The preliminary result shows us that our system, although suffer a little from portability issue, can create 3D representation of a 20 25 meter tunnel in 20 minutes. This will significantly help archaeologists speed up their process of creating visualization of artifacts and architecture. Our immediate future goal is to improve portability and usability, so that every archaeologist can bring our system to create 3D reconstruction of their finding if they so desire.

References

[1] *Robot Operation System* http://www.ros.org/

[2] *Engineer for Exploration* http://ngs.ucsd.edu/

[3] *Microsoft Kinect Sensor* https://msdn.microsoft.com/en-us/library/hh438998.aspx

[4] *RGB-D SLAM* http://felixendres.github.io/rgbdslam_v2/

[5] *Ubuntu* http://www.ubuntu.com/

[6] *OpenCV* http://opencv.org/

[7] *OpenGL* https://www.opengl.org/

[8] *OctoMap* https://octomap.github.io/

[9] *SiftGPU* http://www.cs.unc.edu/~ccwu/siftgpu/

[10] *g2o: A General Framework for Graph Optimization* https://openslam.org/g2o.html