

Building a virtual world in ROS based on the robot's  
perception.

Aiwu Sun  
*S1840266*

Saturday 10<sup>th</sup> August, 2019

# Contents

<b>1 Introduction</b>	<b>4</b>
1.1 Context	4
1.2 Problem definition	5
1.3 Approach	5
1.4 Research questions	5
1.5 Related Work	6
1.6 Report Organization	7
<b>2 Analysis</b>	<b>8</b>
2.1 System overview	8
2.2 Robotic platform	9
2.3 ROS and ROS packages	9
2.4 Kinect	10
2.5 SLAM algorithms	10
2.6 Store outputs from SLAM	13
2.7 Point cloud visualisation tools	14
2.8 VR plugins	15
2.9 VR headset	15
2.10 Approach overview	17
<b>3 Design and implementation</b>	<b>18</b>
3.1 Master side	20
3.1.1 Kinect package	20
3.1.2 RTAB-Map SLAM package	20
3.2 User side	21
3.2.1 Virtual Reality packages and Plugin build	22
3.2.2 VR headset implementation	22
<b>4 Evaluation</b>	<b>24</b>
4.1 Test Results	26
4.1.1 Image quality of point cloud	26
4.1.2 Image quality in Virtual Reality	27
4.1.3 System performance	28
4.2 Discussion	29
<b>5 Conclusion and recommendations</b>	<b>30</b>
5.1 Conclusion	30
5.2 Recommendations	31
<b>Bibliography</b>	<b>32</b>



# 1 Introduction

## 1.1 Context

A joint innovation centre called “i-Botics” has been founded by TNO and University of Twente. The community aims at creating Robotic solutions to face different needs. One of their research lines is telerobotics which is concerned with remote controlled robots in which telepresence and teleoperations is combined. By developing telerobotics, some complex tasks like the discovery and exploration of harsh environments for human operator in celestial and underseas structures can be performed. In the designs for teleoperated robots, multiple sensors are used to monitor the outer environment then process the information and reflect to the operator so the human intelligence still affects the activities while human access is difficult. This communication between operator and robot always causes a time-delay which will possibly affect human’s judging ability and being a serious problem in some operations like rescue activities and urban searching (Casper & Murphy, 2003). This “situation awareness” problem can be avoided or at least reduced to a certain extent by using Virtual Reality technology (Bejczy, Antal, 1996; Fong, Grange, Conti & Baur, 2001). The other important aspect in teleoperations like searching and exploring is that the system should provide the opportunities to store some important data such as images of observed objects and geographic information, so the human operators can promote training experience in the virtual world based on the robot's perception.

This project focuses on represent the sensory information well and allow users to explore into the virtual environment. System should also work as simulators that help human operators to practice and improve skills. So this work will be designed into a combined system of two parts, one of them will be made to locate robot itself and collect geographic information of the platform and surroundings by designing a framework which uses SLAM (Simultaneous Localization and Mapping) algorithms. And the other part should sufficiently support VR headsets to work in this system such as HTC Vive. And this real-time system is expected to have little delay during the transmitting from sensing phase to visualizing phase.

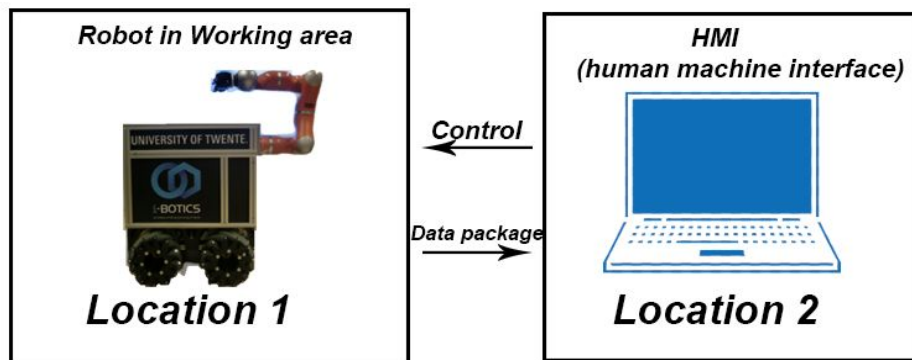


Figure 1.1: Elements in the user interaction

## 1.2 Problem definition

A number of challenges should be taken into consideration while implementing this project. First of all, considering the different compatibilities of all softwares and headset, very much of the choice can not be available such as the visual platforms supporting Virtual Reality can barely run independently in Linux system. So the compatibility of SLAM algorithms and Virtual Reality headsets have to be tested alternatively before the prototyping and also during the evaluation.

Once the data is obtained by Kinect, the system needs to not only show the image but also record the maps as point clouds. This process requires powerful GPU and CPU, as well as a proper way on memory management. Instead of building system in both Linux and Windows system, this work will only need the support from Ubuntu 18.04 and using several plugins to make VR connection possible. What's more, running VR and SLAM on one computer bring more pressure on hardware components, so changes on the hardware part should also be concerned. Once the sensory information is gathered, users need to view the scene with Virtual Reality headset, but there is not yet any Virtual Reality components supported by Linux officially which is the main difficulty.

## 1.3 Approach

The hardware part consists of a RGB-D camera and depth sensor on the robot side, a computer for processing and data transferring and, VR headset on human operator side for viewing the scenes. To communicate with sensors, ROS (Robot Operating System) need to be used as middleware and a rich open sources of libraries and tools that allow using existing applications or building own plugins.

Point cloud will be generated by SLAM algorithm based on the data from RGB-D camera which allows system to record the 3D information and show the images as well. Then the main focus will be running and interfacing fluent Virtual Reality in Linux which is also a key aspect of this project. For this assignment, transmitting delay and system pressure can be reduced and system can work in an easier way representing the image, modeling the shapes, storing the simulation and accessing Virtual Reality by having all tasks done inside one operating system.

## 1.4 Research questions

Since the main focus of the project is to gain a clear recognition from the master side then visualize these 3D maps in a VR environment, my literature study focused on 3D recognition part and looking into the opportunities to run the system with VR in Linux. Based on the

challenges described previously, the following research questions were formulated and coming with reliable literature:

- How to do 3D object recognition and visualization with point cloud?

Answering a set of sub-questions can be helpful dealing with other potential challenge. These sub- questions are:

- What aspects need to be cared about during 3D recognition and visualization?
- What are the difficulties or challenges while creating virtual environment based on point cloud ?

## 1.5 Related Work

Many teams put efforts to develop 3D recognition methods and SLAM project based on robot perceptions. Literature works had been done to figure out the common concerns while using SLAM to process point cloud data and also the find similar systems which aim to use VR in robot project. Those project cooperating with Virtual Reality in ROS environment is called “ROS reality” sometimes.

After obtaining point cloud from RGB-D sensors, the second stage of 3D recognition emphasizes on representing sensor data into interactive visualizations, this could be 3D modeling, 2D mapping and also VR environment building. But there are still some difficulties while visualizing point cloud such as data processing and memory management. Bakkay, Arafa, Zagrouba(2015) and later one other similar project made by Yan, Ye and Ren(2017) have created dense Visual SLAM based on RGB-D measurements using Kinect. Reconstructed scenes from point-cloud data set were clear and their map representation system provides globally consistent map of the environment for visual SLAM though they found that their approach suffered from some limitations which was common to the existing method. Those limitations bring difficulty in building 3D environment based on point cloud especially for building VR visuals.

One main limitation is the extremely high memory usage while rendering 3D objects in VR environment and at the same time process the data from RGB-D sensors. To solve this problem, the common used solution is to divide the tasks to two different computers, one with Linux system to serve ROS side and the other one using Windows system to interface the virtual reality. Windows have a great advantage in providing mature tools to support VR. Hofer, Seitner and Gelautz (2018) have proposed an end-to-end system for dynamic point cloud visualization from RGB-D input data that took advantage of the Unity3D game engine for efficient state-of-the-art rendering and platform-independence.

## 1.6 Report Organization

The outline is as follows:

- *Chapter 2*  
Chapter 2 is analysis that was made for the project. Descriptions on the system and overall approach are included. Considering the results of the literature study and the challenges, the analysis aims at drawing a conceptual solution.
- *Chapter 3*  
Chapter 3 discusses the design and implementation of the system. The software and all the tools used in the project are described and documented.
- *Chapter 4*  
Chapter 3 includes the evaluation of the final system and discusses the results of all sub-tests.
- *Chapter 5*  
Chapter 5 draws the conclusion on the whole system and gives recommendations on the future work.

## 2 Analysis

The analysis part aims at getting deep into the details of this topic and using the existing knowledge to find possible solutions. This chapter consists of two parts, the first section discussed the tools and packages used in system and the second half compares the different SLAM projects and alternative VR platforms.

### 2.1 System overview

This section gives brief information about the teleoperation system including master side and user side and then provides the details of the tools used during the project. The initial system plan has two platforms which is illustrated in Figure 2.1.

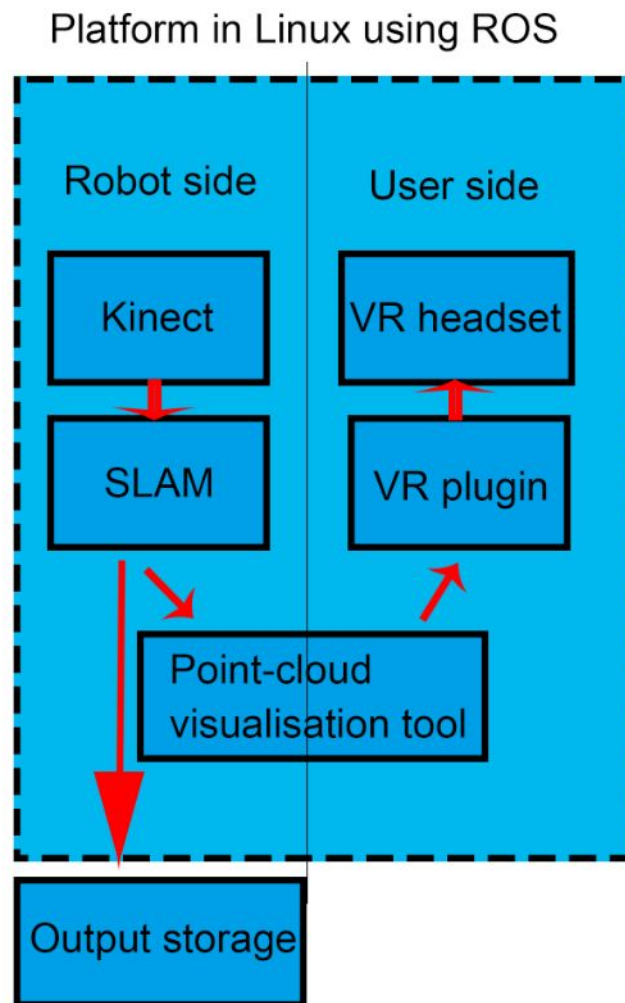


Figure 2.1: Overview System



## 2.2 Robotic platform

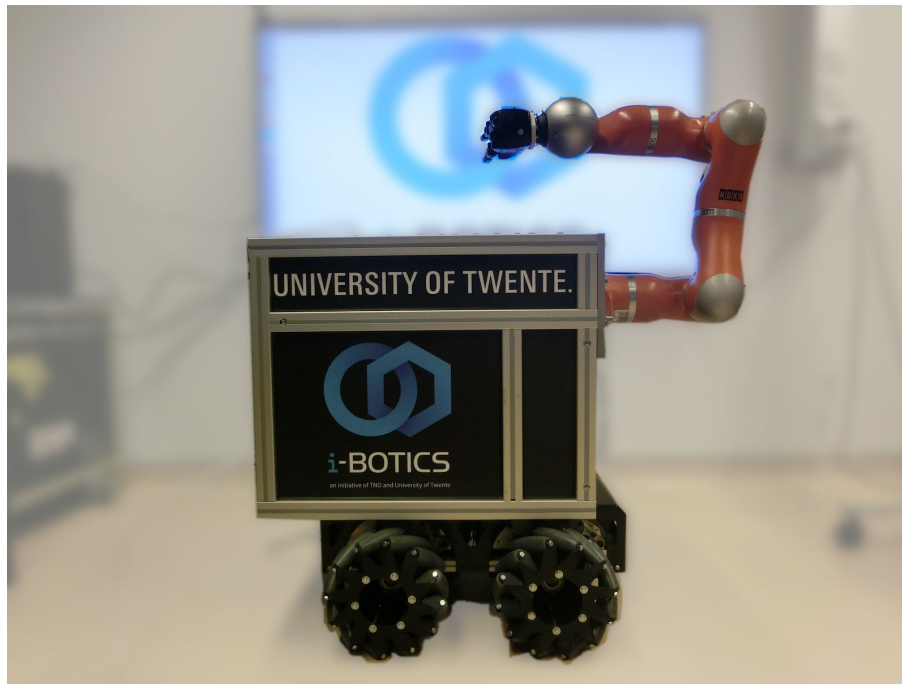


Figure 2.2: The i-Botics robotic platform

The teleoperation system will be developed in the i-Botics robotic platform. Figure 2.2 has shown side view of the platform which basically consists of three parts: the base, robotic arm and processing units inside the box. With the wheels on the base, the robot is able to move under the control of a human operator. It can be controlled remotely through joy stickers or the command sent to the control units. In this project, Kinect sensor can also be placed on the robot platform and connected with the processing units. The platform has an Intel NUC as processing unit and the Ubuntu as operating system. The system can be connected with other PC through wifi. And it can be connected with other sensors that supports Linux.

## 2.3 ROS and ROS packages

Robot Operating System, is described as "an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system". ROS is widely used in robot research and development to manage pack-age, transfer data and control devices and sensors with a rich collection of libraries and tools. However ROS currently only support Ubuntu and MACas the operating systems so a bridge is required to transfer data from ROS to a Windows software. For ROS, it allows a wide choice of packages and tools covers from telecommunications to 3D modeling. With ROS fully installed with all official packages, it contains most of the tools needed in this project such as RViz and drives for Microsoft Kinect

sensor. What's more, to make VR running on Linux, some packages such as OpenVR, OpenSDK will be required. Most of the packages can be installed manually through Github or using SSH in the terminal. The distribution of ROS used in this project is in Ubuntu 18.04 with ROS Melodic. All the libraries and packages are updated to the latest version. All the components should be installed to fit this version and so avoid systematic error.

## 2.4 Kinect

The Microsoft Kinect sensor features an RGB camera, depth sensor and microphone array in side one. It has very friendly price among all cameras which can provide 3D motion capture and facial recognition. An open-source driver was offered to developers in 2010 which allows Linux to use it as RGB camera and depth sensor. To use launch Kinect in Ubuntu requires an additional package names OpenNI which is an open-source software framework. The output of Kinect in RViz can be set to point cloud, raw image, depth image, etc. While using Kinect in SLAM algorithm, the map is built as point clouds.

## 2.5 SLAM algorithms

Simultaneous localization and mapping(SLAM) is "the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it ". SLAM use several different types of sensors to map the environment. In non static environments, such as those containing other vehicles or pedestrians, continue to present research challenges.

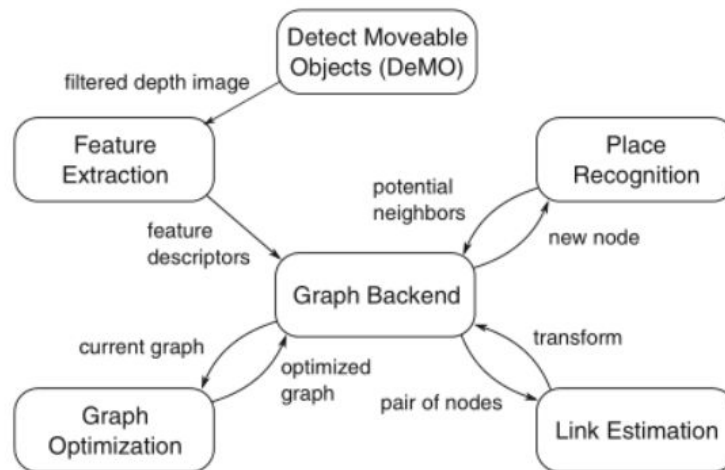


Figure 2.3: Components of Visual SLAM framework(Klüssendorff,Ehlers & Maehle. 2016)

In this section, an overview of the SLAM algorithm are included in the table 1. The overview of the alternatives is shown together with its main advances or focuses compared to the others.

Each type of algorithm will be only collected once with its latest version. Furthermore, if our requirements have been met, it will be indicated how well this requirement is met with certain algorithm. The information is collected from Open SLAM<sup>1</sup> and Wikipedia page<sup>2</sup>.

Table 2.1: Overview of SLAM algorithms

Name	Advantages	Kinect supported?	3D ?	Type of Map
DP-SLAM	prevents errors in the map from accumulating over time	NO	NO	Grid maps
OpenCV RGB-Odometry	Built on OpenCV	YES	YES	Graphs
RGBD-SLAM	allows to quickly acquire colored 3D models of objects and indoor scenes	YES	YES	Pose graph with colored point clouds
ORB-SLAM	able to close large loops and perform global relocalisation in real-time and from wide baselines	YES	YES	Sparse 3D points
InfiniTAM	Real time sensing, high frame rate and high portability	YES	YES	camera poses
ElasticFusion & Kintinuous	Real time large scale dense RGB-D SLAM (BUT doesn't support ROS)	YES	YES	Models with point-cloud, normal images and camera poses
Co-Fusion	enable the robot to maintain 3D models for each of the segmented objects and to improve them over time through fusion	YES	YES	Models with point-cloud, normal images and camera poses
RTAB-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	YES	YES	Models with point-cloud, normal images and camera poses

<sup>1</sup> <https://openslam-org.github.io/>

<sup>2</sup> [https://en.wikipedia.org/wiki/List\\_of\\_SLAM\\_Methods](https://en.wikipedia.org/wiki/List_of_SLAM_Methods)

From the file type those methods support, it can be seen that some projects focuses on navigation while some others focuses on real time simulations and feature recognitions. Co-Fusion (Rünz & Agapito, 2017) is really good example which represented highly acceptable images with functions to save the file as point cloud and other 3D model format such as ply.. The one latest SLAM algorithm is RTAB-map, which is available in ROS system as a plugin for RViz and can be installed by calling commands in terminal. Among those SLAM algorithms, RTAB-map seems to be the most fittest one as it has more promised updating and maintenance. What's more, considering the system is built on one computer, the memory management is really important. RTAB-map has been designed with less CPU and GPU consuming which is very helpful in this project.

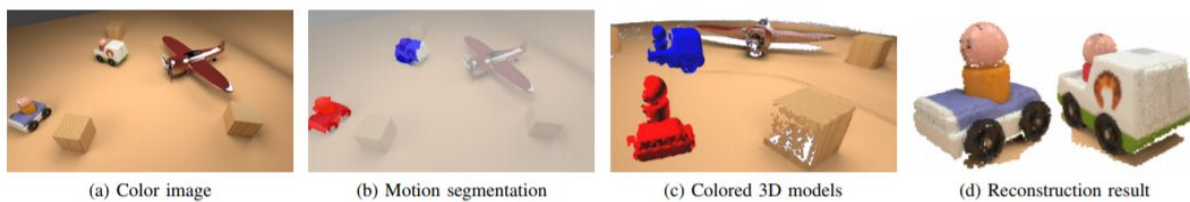


Figure 2.4: Example of the map produced by Co-Fusion

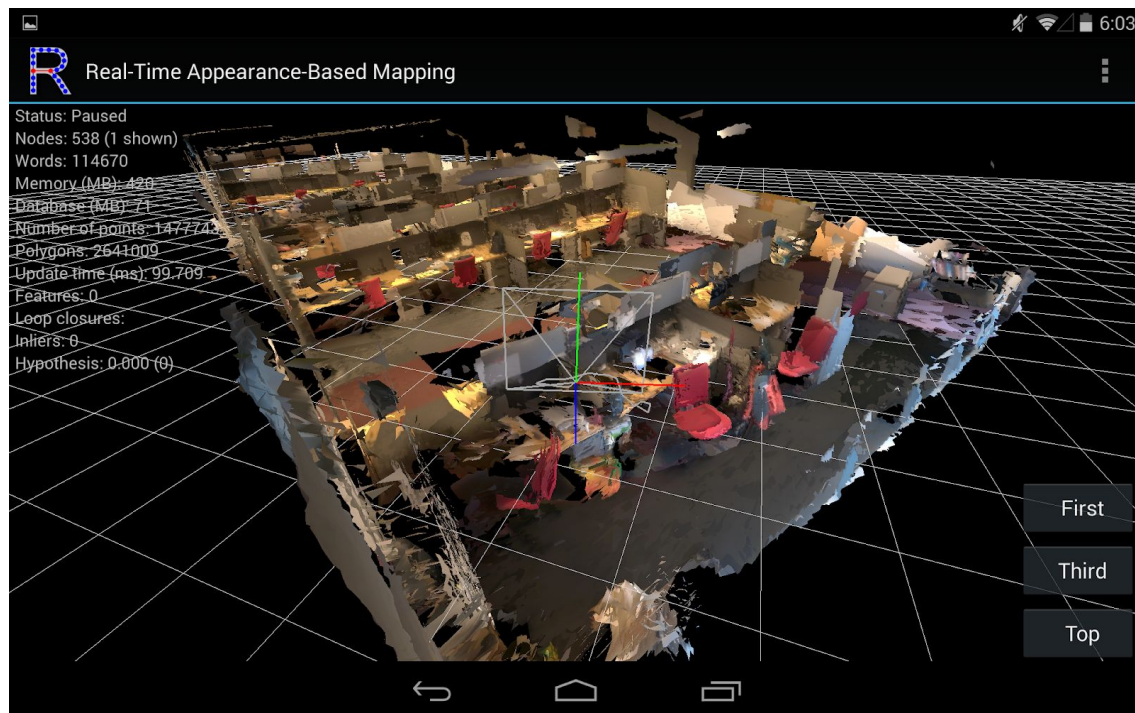


Figure 2.5: Example of the map produced by RTAB-map.

## 2.6 Store outputs from SLAM

As mentioned previously, one of the requirements of the system is having the functionality of saving data such as images and geographic information, so the important data can be saved and also being used for later access like training and learning. So the point cloud outputs of SLAM should be easily accessed and used in other applications. To have better accessibility to other platforms or software, the output file of SLAM should be stored with proper file format. Instead of saving the point cloud as bag file, the file which can include 3D information such as XYZ file, OBJ file and PLY file are preferred. Point cloud file types basically use two different methods, one is ASCII and the other is binary. ASCII consists OBJ, PLY, XYZ, PTX and ASC. Binary systems includes FLS and PCD.

**XYZ:** is a non-standardised set of files based on coordinates ('x' 'y' and 'z'). XYZ is an archetypal ASCII file type, conveying data in lines of text. There are no unit standardisations for XYZ files. Although there is wide compatibility across programs for this type of file, the lack of standardisation surrounding units and specifications makes it a fundamentally faulty method of data transfer unless additional information is supplied.

**OBJ:** first developed by Wavefront technologies, the format has been adopted by a wide range of 3D graphics applications. It is a simple data format that only represents 3D geometry, normals, colour and texture.

**PLY:** known as the polygon file format or Stanford triangle format, PLY was inspired by OBJ and purpose-built to store 3D data. PLY uses lists of nominally flat polygons to represent objects. It's a file format capable of representing colour, transparency, surface normals, texture, coordinates and data confidence values. There are two versions of this file, one in ASCII and the other binary.

Several other regularly used file types are capable of both ASCII and binary formats. These include PLY, FBX store data in both binary and ASCII, According to the comparison, PLY file seems like the best choice among those point cloud file types since it serves the robotic area and have better accessibility which makes both Windows based software and Linux based software can represent point clouds.

## 2.7 Point cloud visualisation tools

VR as a teleoperation interface, there are multiple ways of displaying the robot's state to the user and mapping the input to the robots. There are plenty of choice on the engines that can be used for Virtual Reality applications. Those engines are made with different addons to face different user needs. In order to narrow down the choice and save time for future production and experiment phases, several popular engines which supports VR and can be connected to ROS systems will be discussed.

**Gazebo:** Gazebo is an open-source 3D robotics simulator which can model sensors such as Kinect ,laser sensors and with opportunities to support VR. The advantage of Gazebo is that the engine is free access, specifically designed for ROS users and allow users to develop the addons on their own. However, Gazebo is not a visualisation tool but a dynamics simulator.

**RViz VR:** RViz is a visualization tool for ROS, which can display the sensory information from the Kinect and Laser scanner. Those representations can be displayed lively while the camera is working and also doing measurements and other processing at the same time. Different from Gazebo, Rviz can use VR plug in and communicate with VR headset and engines.

**Unity 3d:** Unity is game engine used to create high qualified visual scenes but not a simulation tool. It's now the most popular game engine and widely used for VR tasks because it supports multi-platforms and have powerful physics engine. It also has the possibility to work with ROS instead of RVIZ. To some extent human-robot interaction is supported within Unity 3d. It uses C# or JavaScript, which is more preferred than C++.

**Equivalents:** other game engines such as Unreal, DirectX and CryEngine also include similar features as Unity. Unreal Engine focuses on graphis which give the gaming experience a realistic touch with features like advanced dynamic lightings. "Its new particle system that has the ability to handle as many as million particles in a single scene". But in this project, the scene we captured is from Kinect and doesn't need such a powerful graphic processing. CryEngine is very much similar to Unity but has better graphical capabilities.

In conclusion, RViz and Unity will be good try to start with. And both of them can be installed in Ubuntu. For the system side, however, Linux is more friendly to RViz. It's hard to test the performance of Unity in Linux because the SLAM in Unity is built for Augmented Reality. What's more, to implement RTAB-map in Unity requires more work and additional tools than RViz such as ROSbridge which will certainly increase memory usage and lower down the system performance. Lastly, considering the user experience, user interface of RViz is more simple and easier to learn how to use which focuses on visualizing robots and sensor data while Unity is more professional on modeling and designing.

## 2.8 VR plugins

VR plugins in Linux system consists of two different kinds, One like OpenVR and OpenHMD is used to allow VR components being recognized and accessed in Linux. These SDK (software development kit) have default application programming interface (API) and runtime which are mandatory to start VR. The second VR plugins are necessary for connecting SDK and local software such as Gazebo and RViz because SDK only consists of a source of codes for setting up VR environment but can't directly building VR functionality in software especially under Linux. Those plugins are not offered officially by Linux or VR vendors. However, some projects have shown strong possibilities to build VR plugins in Linux system such as "oculus\_rviz\_plugins"<sup>3</sup> which is a VR plugins in RViz for Oculus Rift built in 2014. Due to the updating of the headset and software, many VR plugins are no longer working and need to adjust to a specific system environment.

## 2.9 VR headset

Though selecting VR headset is not the main focus in this project, it would be nice to build a more accessible system not just support and serve one kind of hardware. In order to bring more comfortable user experience, some aspects of the headsets were compared. Among those aspects, the field of view, resolution and tracking are considered to be most important for this project.

However, it's hard to test all the VR headsets on the market. Comparison can only be made by pairing online data such as comments and evaluations from users. In this section, two tables from the internet have been collected to talk about the difference of VR headsets.

**Field of View:** The higher the better. This refers to how much of your eyesight will be covered when wearing the VR headset. The human eye has a field of view of around 180 degrees, so a field of view of 110 will pretty much cover your whole vision

**Resolution:** Again, the higher the screen resolution of your headset the better. Higher resolution screens are able to produce a clearer image and more immersive experience.

**Tracking:** This is a slightly more complicated subject, as some headsets are not designed to track your whole body and some can track you around an entire room. The PSVR uses a camera to track hand movements, however you'll be limited to the wire attaching you to the system. The same is true for more advanced HTC VIVE and Oculus Rift, however these utilize









---

<sup>3</sup> [https://github.com/ros-visualization/oculus\\_rviz\\_plugins](https://github.com/ros-visualization/oculus_rviz_plugins)



more accurate body tracking. Some headsets are stand alone and require no wires or cameras, however these usually don't allow for advanced body tracking.

Table 2.3: VR Headset Comparison Table<sup>4</sup>

Specs	HTC Vive	Oculus Rift	PSVR	Samsung Odyssey	Oculus Go	HTC Vive Pro	Lenovo Explorer	Samsung Gear VR
Image								
Max Field of View	110°	110°	110°	110°	100°	110°	110°	96°
Max Resolution	2160×1200 (1080×1200 per eye)	2160×1200 (1080×1200 per eye)	1920×1080 (960 × 1080 per eye)	2880 x 1600 (1440 x 1600 per eye)	2560 x 1440	2880 x 1600 (1440 x 1600 per eye)	2880 x 1440 (1440 x 1600 per eye)	2560 x 1440
Screen Type	Duel AMOLED	Duel AMOLED	AMOLED	AMOLED	Fast-Switch WQHD LCD	Duel AMOLED	LCD	AMOLED
Pixel Density	461ppi	456ppi	386ppi	615ppi	Unknown	615ppi	706ppi	–
Sensors	Accelerometer, gyroscope	Accelerometer, gyroscope, magnetometer	Accelerometer, gyroscope	6-Axis ACC & Gyro, 3-Axis Compass, Proximity sensor, IPD Sensor	Gyroscope	SteamVR Tracking, G-sensor, gyroscope, proximity, IPD sensor	Proximity, Gyroscope, Accelerometer, Magnetometer	Accelerator, gyrometer, geomagnetic, proximity
Max Refresh Rate	90Hz	90Hz	90Hz, 120Hz	90Hz	Unknown	90Hz	90 Hz	60Hz
Tracking	6 DOF IR Laser-based 360-degree tracking using "Lighthouse" Base Stations	6 DOF Constellation camera optical 360-degree IR LED tracking	6 DOF PlayStation Camera optical 360-degree LED tracking	2 x 6 DOF camera	Orientalional Tracking	–	2 x Inside-out motion tracking cameras	N/A
Weight	563g	470g	610g	644g	Unknown	–	380g	310g

<sup>4</sup> <http://www.threesixtycameras.com>



In this project, whether the headset can support Linux system is the most important and basic requirement, but it's hard to try the compatibility of each. The one developers commonly used in Linux is HTC Vive which uses SteamVR as platform and can be installed simply. So HTC Vive would be the best choice and using Oculus Rift as an alternative.

## **2.10 Approach overview**

The final approach has been chosen for this project after selecting alternatives and gathering enough sources. RTAM-map SLAM will be used in this project as a plugin in RViz. By connecting the Kinect sensor to ROS system using OpenNI, depth camera can be activated inside RViz and start to produce 3D point clouds. A Catkin workspace will be created and the package of VR plugins can be launched. The VR plugin uses a MakeList file to have it work with OpenVR and then allows headsets to be recognized by SteamVR. MakeList file can be gathered as open source from similar works from Github. Once the measurement is finished, the outputs will be stored as bag. file in Linux and PLY format point clouds. So others can run the simulation locally in RViz with bag. File or viewing the models of point clouds on other software.

### 3 Design and implementation

In the last section of Chapter 2, a brief approach was proposed which using RTAB-map as SLAM algorithm and building a plugin for VR headset in Linux. The RTAB-map package can be installed in the list of packages prepared for ROS system which names `rtabmap_ros`. Therefore, RViz with RTAB-map can achieve the goal: to represent the sensory information obtained by robot. Then the next stage is to decide on the VR components including both software and hardware. The final setup for the project is shown in Figure 3.1. Having both master side and user side in one environment, Robot works as sensor and human can process all the operations on one computer.

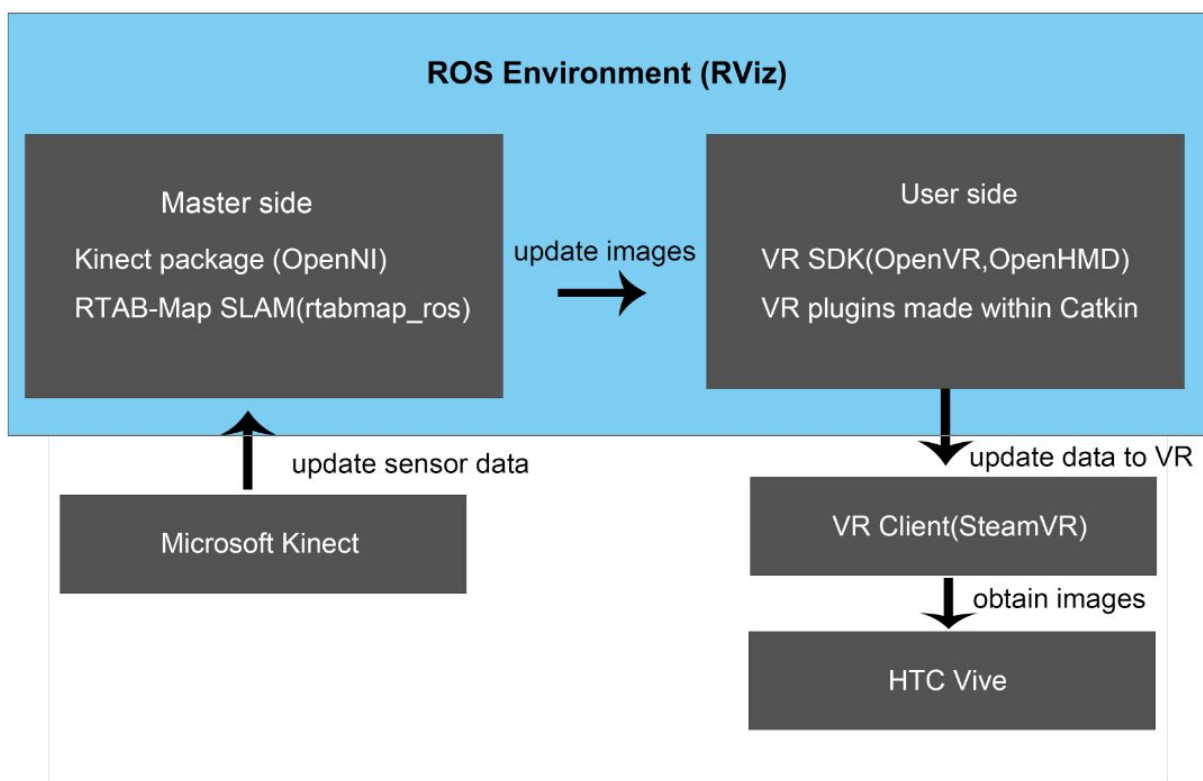


Figure 3.1: Final approach

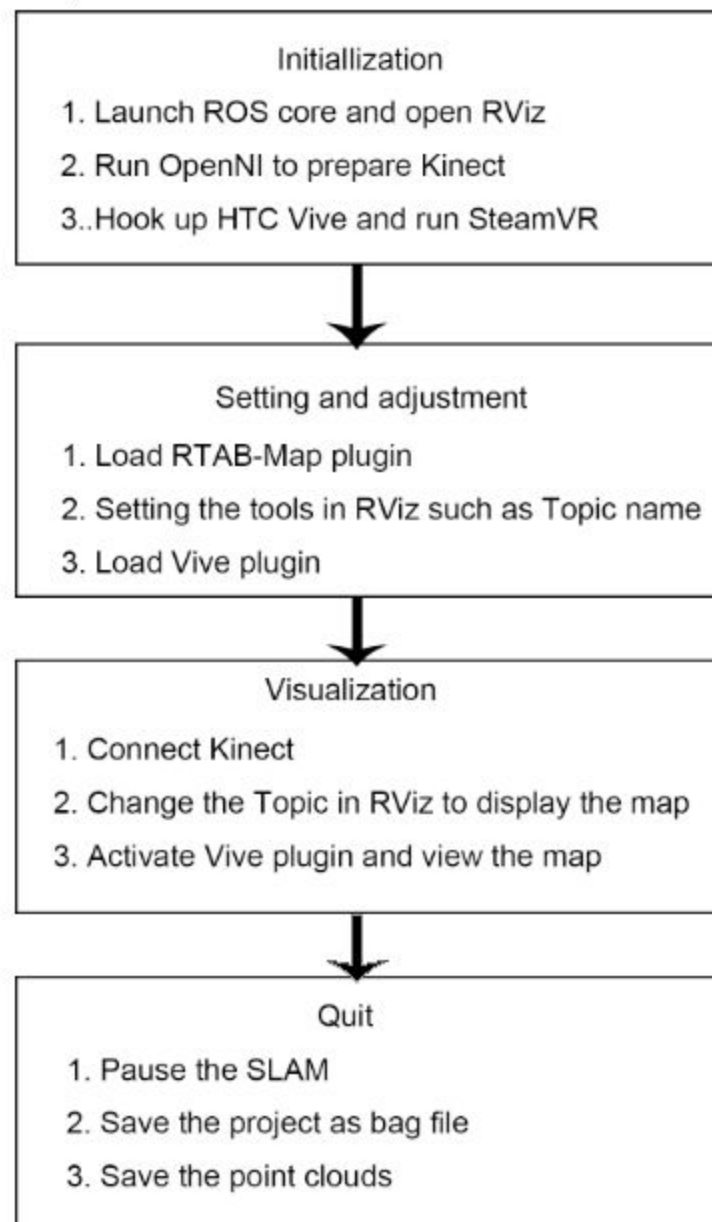


Figure 3.2: Order of workflow

The original setup has two parts environment, the ROS environment was built in one Linux computer and then sent the simulations to one Windows computer by using Rosbridge<sup>5</sup>. After obtaining the information from Robot side, visualizations were made in VR environment such as Unity which can perfectly support most of the headsets on the market. However, in the final

<sup>5</sup> [http://wiki.ros.org/rosbridge\\_suite](http://wiki.ros.org/rosbridge_suite)

configuration, Linux appears to be able to run the VR headset as well which makes both sides in one environment possible: The visual data from Kinect goes into RViz and Rviz offers VR scopes to view those data. Though there is only one platform in the final approach, it still consists master side and user side.

Figure 3.2 shows the order of execution for final system. The final system also has advantage on controlling part. A two platform system requires a lot of packages and proper settings. It only allows VR environment to show one type of image, point cloud or raw image. In order to switch among different images type, both of the platforms need new settings and this would take some time and additional works. Therefore, a system in one platform can be easier to control and require less packages and tools.. All the packages in this system consist: rtabmap\_ros, OpenNI, OpenVR, OpenSDK, RViz and SteamVR.

## 3.1 Master side

This section contains two parts, it describes in detail how the sensory information will be gathered from camera and how the SLAM algorithm can be implemented and used in RViz.

### 3.1.1 Kinect package

To make Kinect work properly in Ubuntu, a package is required. OpenNI(Open Natural Interaction)<sup>6</sup> is one commonly used software development kit. This API supports the interaction devices with recognition functionalities such as hand gestures and body motion tracking. Kinect is one of these devices and middlewares. This package contains launch files for Kinect and creates nodelet graph to transfer raw data into point clouds and types of images. After installation, it needs to be launched every time using Kinect. In order to display Kinect output in Rviz, the Fixed Frame and also settings on Topic needs to be changed into /camera\_link. Then the Kinect information can be obtained and displayed in Rviz.

### 3.1.2 RTAB-Map SLAM package

As mentioned previously, rtabmap\_ros<sup>7</sup> is package used for SLAM algorithms. It has been chosen because it can both generate images paired with locations then save the point-cloud maps. It is a RGB-D SLAM approach with real-time constraints which can be installed within ROS system through “apt-get install ros-melodic-rtabmap-ros”. RTAB-Map plugin in RViz consists of three different functions: MapCloud, Info and MapGraph as shown in Figure 3.3.

MapGraph display subscribes to /mapGraph topic. This section decides the colors of RTAB-map graph based on the link type. MapCloud is the main and also the most essential feature which is used to generate a 3D map cloud incrementally in RVIZ. When the robot is moving, graph is changed based on the Kinect, all point clouds added in RVIZ will be transformed to new poses.

---

<sup>6</sup> <https://github.com/OpenNI/OpenNI>

<sup>7</sup> [http://wiki.ros.org/rtabmap\\_ros](http://wiki.ros.org/rtabmap_ros)

MapCloud is using /mapData as topic and offers different options to visualize the data. Some of the settings are shown in Figure 3.3. All the parameters can be changed including shapes, depth and also how input depth are decimated. Default settings work well in this project so nothing needs to be adjusted while running system. Map and graph are downloaded after finishing mapping, ready to be used later as simulations. Not like Kinect which needs running OpenNI first, RTAB-Map once has been installed inside RViz , it can be added and run without using additional commands. After RTAB-map is done with saving the final map as database, 3D map can be viewed and edited using format converting software.

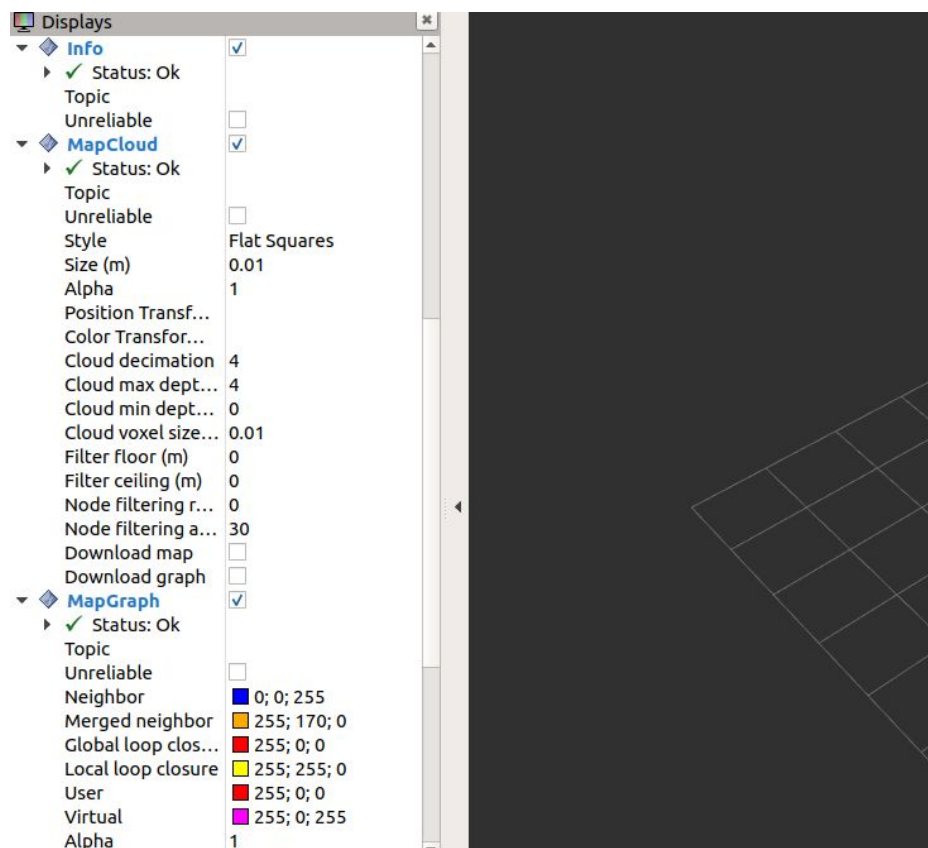


Figure 3.3: Interface of RTAB-Map SLAM in RViz

## 3.2 User side

The following section provides insights on how the visualization and Virtual Reality displays are happening. The section consists of three parts. The first part describes how to make proper settings of visualization components and let RViz present the information. The second part focuses on how virtual reality plugins will be built and implemented into RViz. The third part describes the way of running VR headset under Linux system.

### 3.2.1 Virtual Reality packages and Plugin build

Since RViz doesn't have VR functionalities officially, some similar projects as mentioned previously were using additional plugins and packages to run VR in RViz. In order to run VR, libraries consist of runtime files and running environment are required. Most of them can not be installed straightforward in ROS system. The system first needs an open SDK, which is used to recognize VR facilities such as OpenVR and OpenHMD. OpenVR supports HTC Vive and OpenHMD makes Oculus Rift works. Both OpenVR<sup>8</sup> and OpenHMD<sup>9</sup> are open source on Github. By following the instructions for compiling and installing, these two packages can cooperate with SteamVR. After installing these two repositories, plugins with VR functions shall work in Linux.

To build a plugin in RViz, Catkin has to be included in the ROS system which is most of the time a part of default installation of ROS. Then a catkin workspace will be created to place the empty or existing sources such as the Oculus plugin made by Willow Garage (2013). By default, the Oculus view will be rendered from the same position as the main RViz camera while following your head's orientation, similar to the Vive plugin made by André Gilerson(2016). In this project, the orientation and reflection part is not in the plan but maybe the future development, so a plugin which only has function of supporting VR headset is what we need in this system. Their ideas on making the plugins and Makelists will be followed and parts of the code will be changed or removed to make the program lighter.

### 3.2.2 VR headset implementation

Though there are alternatives VR headsets on the market, not all of them can be considered as open source in Linux system. A test has been made with Oculus Rift DK2, Windows Mixed Reality (LENOVO Explorer) and HTC Vive. As mentioned in previous section, OpenVR, OpenHMD and SteamVR packages can support VR environment in Ubuntu by simply hooking up the headset and call the runtime environment<sup>10</sup> of SteamVR. Runtime SDK for SteamVR allows the local plugins to activate VR headset and build connections. This SDK can be installed manually following the github page.

Among the three tested headsets, HTC Vive appeared to be the only one support Linux system. Window MR required Windows operating system while Oculus Rift's SDK on Linux is based on OpenHMD. In the official announcements from Oculus, "development for OS X and Linux has been paused in order to focus on delivering a high quality consumer-level VR experience at launch across hardware, software, and content on Windows" from 2015 which means as long as the Oculus SDK stopped updating, the system may be unstable. As the result of the test, ROS core can not recognize Oculus Rift using Nvidia latest graphics card driver 430. This

---

<sup>8</sup> <https://github.com/ValveSoftware/openvr>

<sup>9</sup> <https://github.com/OpenHMD/OpenHMD>

<sup>10</sup> <https://github.com/ValveSoftware/steam-runtime>

means HTC Vive will be the best option. However, by downgrade the driver version, Oculus Rift can be used as an alternative headset.

## 4 Evaluation

The previous sections have described the implementation of the system and the usage of packages. This is a single client system built in Linux which means point cloud data is directly obtained in ROS system without additional transferring tasks, so there is no lag or delay for data transferring in this system after Kinect being activated. Thus the evaluation consists of two parts, starting with testing visualizations quality of point clouds and virtual reality including color, resolution and scale rate in scopes. The other important aspect is to evaluate the memory management of the system which may interrupt the workflow. Since this project is designed for human operators to obtain visual information in teleoperation tasks, the fluency of the system remains priority and the limitation of the system should be tested in evaluation as well. As shown in the graph below, several tests are designed and inserted into different phases of working pipeline.

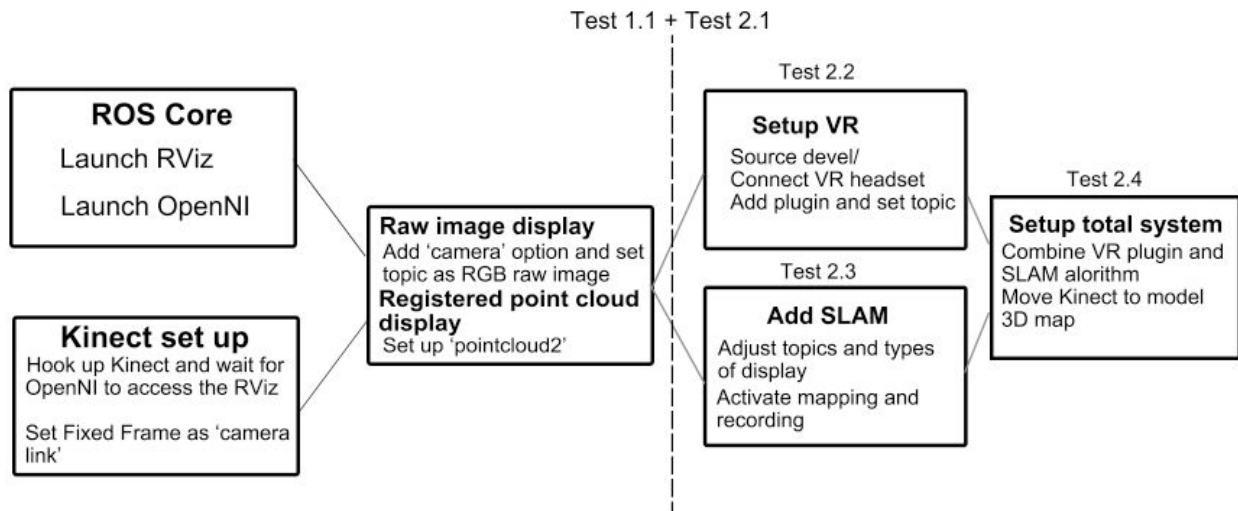


Figure 4.1: Location of tests in different phases of launching and running the system

Test 1 aims to evaluate the quality of visualization by answering the questions:

1. How is the image quality of point cloud image in RViz?
2. How is the image quality in Virtual Reality compared with raw image or point cloud image?



Test 2 will focus on evaluating the memory management and user experiences:

1. What's the frame rate while system working on different tasks?
2. What does the system performance response to each different task?
3. Does the system performance show any difference while the robot's moving?

Test 1 consists of two sub-tests, each one will answer one of the questions. The first sub-test will compare the quality between two versions of the images which are raw RGB image and a point cloud image registered with colors. The quality will be evaluated by comparing resolutions, colors and range of views. The second sub-test will be held after launch VR plugins and access the view in VR headset, then the captured images in VR lens will be compared with the normal point cloud images in Fixed Frame.

Test 2 consists of four sub-tests. Each sub-test includes one task for RViz and for each test, frame rate and system performance will be recorded. Since the default setting of maximum frame rate is 30 fps in RViz which can't show significant differences, it should be set to 60 fps to evaluate performance easier. The first test will measure the original system performance after accessing the image of Kinect. The second test will be held after VR plugin being activated while SLAM is not launched yet. Then the third system will run the second test on the other way, with SLAM working instead of VR. Finally, the last test will measure the performance of system with both VR and SLAM running.

Serial Number	00329-10544-20263-AA087
Number of cores	6
Number of threads	12
RAM	8.00 GB
CPU	Core(TM)i7-8750H@ 2.20GHz
GPU	Nvidia GeForce GTX 1060 (6GB)
Hard Disk	256GB SSD

Table 4.1: Specifications of the computer used in the evaluation

## 4.1 Test Results

After running the tests, images and data had been recorded in order to evaluate the system. The result of each test will be discussed in separate sections below. The evaluation chapter will be ended with a discussion section.

### 4.1.1 Image quality of point cloud

In the Test 1.1, the point cloud images have been captured to be compared with raw image from Kinect camera. The image has a resolution of 640\*480 pixels provided by Kinect. During the evaluation, the point cloud in the fixed frame displayed the same scenes but some parts of the image lost its shapes and colors. These loss of images were expected as the characteristic of 3D point clouds. The point cloud maps could be scaled, rotated in RViz. In Figure 4.3, it shows that the point cloud image displays the centre part better than the outline part. There is one red wire and green wire and a power switch in the middle part of the image in Figure 4.2 which is also displayed clearly in 4.3 as point clouds.



Figure 4.2: Raw RGB image from Kinect



Figure 4.3: Point cloud image from Kinect

The images further from the mid zone have lost more details on colors and shapes such as the bottle and table in the figures. With the help of SLAM algorithm, more information of the images have been captured and displayed after moving or rotating Kinect for a little degree.

In conclusion, Kinect has very acceptable resolution as depth camera and point cloud images from Kinect show high clarity on colors as original RGB images. However, the points of the maps are not integrated enough. The middle part of map has more pixels while the corners show less pixels which means the loss of some image data. This problem doesn't bring much

trouble to the system because images will be more complete while moving Kinect and updating scenes. Briefly, point clouds in RViz has acceptable image quality.

#### 4.1.2 Image quality in Virtual Reality

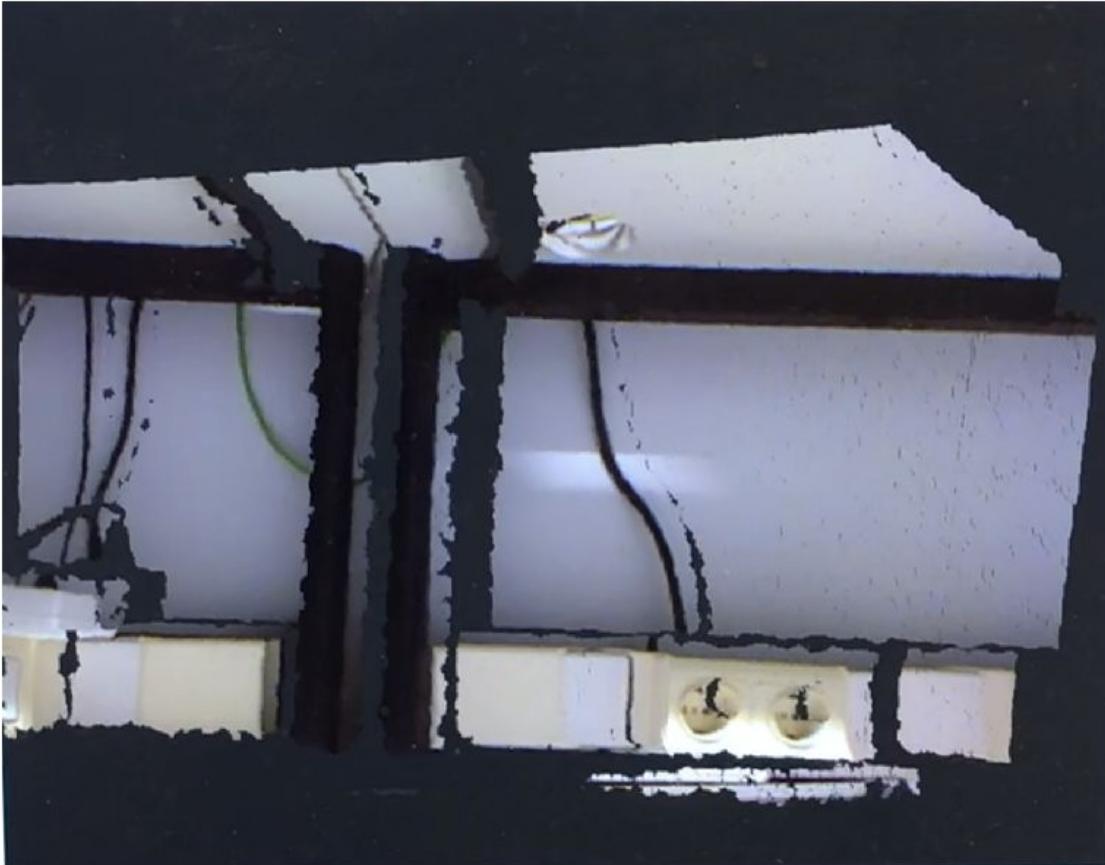


Figure 4.4: Point cloud from Kinect

While activating Virtual Reality plugin in RViz, one additional screen appeared which showed the views in VR lens. The results showed that Virtual Reality plugin gave clear images as the normal point cloud. But the VR view displayed different background color and different brightness felt by users. The size of the image didn't perfectly fit the size of lens which required additional adjustment in RViz.

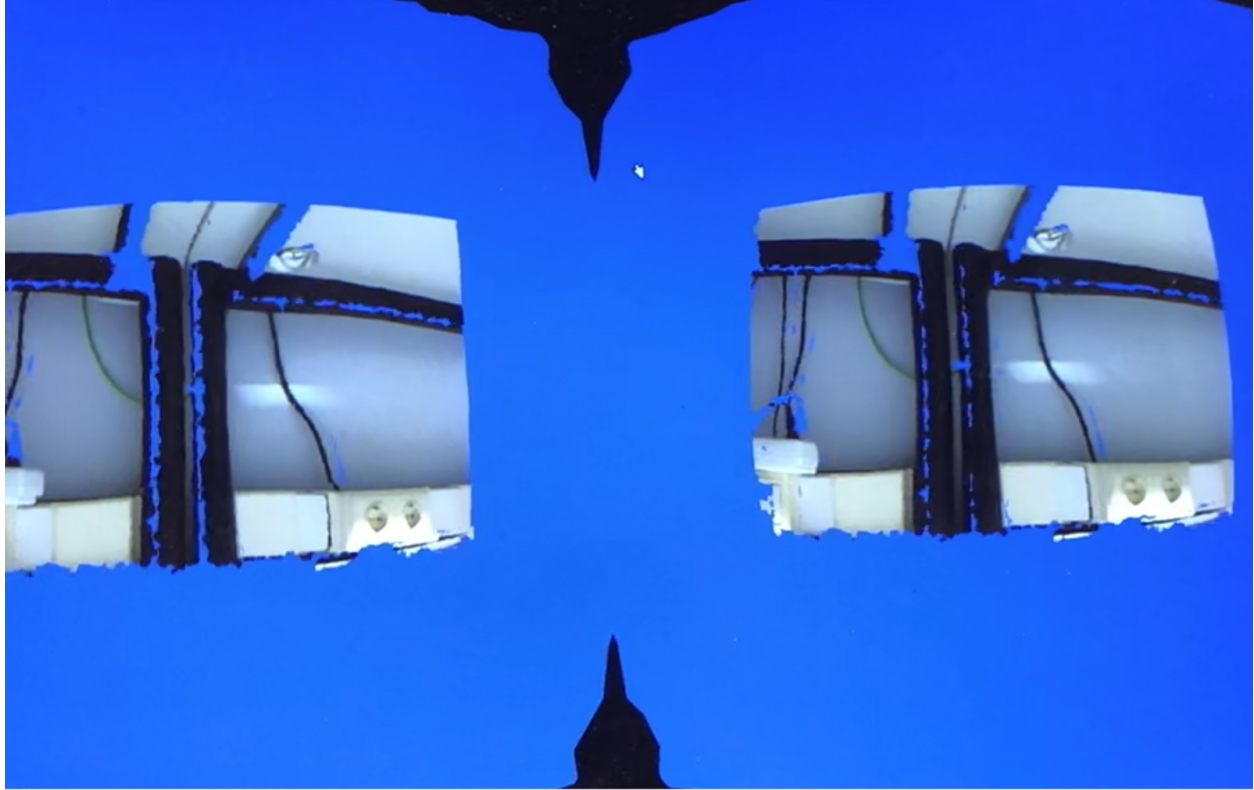


Figure 4.5: Scenes displayed in Virtual Reality lens.

#### 4.1.3 System performance

In Test 2, system has been tested with four different tasks on one computer which means the tests environment and variables had been controlled. The first task “Point Cloud” in the experiment tested the system performance while running Kinect and displaying point cloud images from Kinect. The second test “VR plugin” was designed for testing the performance of VR in RViz to display point clouds. The third one called “SLAM” which tested the performance while activating Kinect and only using RTAB-map in RViz. Last test was made to test the full functionality of the system with RTAB-map and VR plugins both activated.

The table below shows the test results. The first test had the best performance on frame rate which almost remained 60 frames per second and system worked very fluently while changing the position of the Kinect. The result of the second test also showed steady and well performance of the system. The third test and forth test showed slightly different results, so it can be concluded that SLAM algorithm has brought most of the system pressure. Especially each time Kinect’s location changed, system had performed FPS drop and higher usage in CPU, GPU and RAM. What’s more, activating SLAM and VR at same time caused system crash and showed segmentation error if Kinect was moving too fast.

Tasks	Frame Rate (fps)	FPS while Kinect moving (fps)	CPU (%)	GPU (%)	Memory (%)
Point Cloud	58~60	57~60	10~16	5~9	46
VR plugin	54~60	49~60	23~38	12~21	58~59
SLAM	18~26	3~22	36~89	39~52	73~82
VR+SLAM	12~23	~16 Crash!	63~	41~71	76~86

Table 4.2: Test results of system performance

## 4.2 Discussion

This section gives discussions on the results of tests and findings in previous sections. To begin with, RViz is a good choice of platform which can both handle with point cloud processing and setting up VR environment. It has a rich library to support sensors so Kinect performed very stable work in RViz. Secondly, RTAB-map analysed point cloud effectively and has well memory management. However, SLAM is the main reason for system's overload and this overload actually caused RViz crashed out. In case of such crash happen, the previous captured information will be lost and can not be recovered, which is the main limitation of this system.

Virtual Reality has very stable performance and doesn't cost much system usage, but the quality of the image in VR is still needs to be improved. The uncomfortable brightness and bug in color lead to worse user experience.

All that brings to the conclusion that the visualization system can handle with point cloud, provides VR views and work as simulator but all of those features still have flaws. Some of those flaws are caused by the nature of tools, and some can be fixed by improving the codes such as the Virtual Reality part. However, more research needs to be done to develop the accessibility to different VR headsets and quality of VR can be surely improved by developing the codes.

## 5 Conclusion and recommendations

### 5.1 Conclusion

The goal of this project is to build up a system which can view the sensory information as point cloud in Virtual Reality and have a way to record the point cloud map for later use. So the research question “How to do 3D object recognition and visualization with point cloud?” has been made to help build the system. And this main research question was answered by the design of the final system which performance all the requirements.

After following my sub-question “What aspects need to be cared about during 3D recognition and visualization?” which emphasized finding the important aspects for point cloud recognition and visualization. The system has been made to face those aspects such as the clarity of depth value in point cloud image, this has been solved by using RViz which has better nature on robot platform as visualization software instead of using Unity which is designed for modeling and gaming design in Windows. The fluency of the work has been ensured by using RTAB-Map as SLAM algorithm which performances well on memory management.

What's more, in order to answer the third sub-question “What are the difficulties or challenges while creating virtual environment based on point cloud?” many experiments have been made to first try the possibility of using Virtual Reality in Linux system. And this question has been answered by using libraries and open sources to build a VR plugins in RViz. VR plugin has stable work in RViz but it still consists of many bugs. And due to the different nature of VR headsets, this plugin can not be commonly used for all kinds of headset, different headset requires totally different SDK and libraries.

In conclusion, RViz shows its strong ability in handling with point cloud includes capturing sensory data and also recording 3D point cloud maps. What's more, the second reason to choose RViz is because Linux system gives more freedom on using open sources to build own applications and plugins for different needs. However, there are many uncertainties while using the sources in Linux. For example, in this project, system couldn't recognize the VR headset until the driver of graphic card had been downgrade to a very old version which means some of the libraries are no longer supporting the latest graphic driver. There would be more hidden aspects to cause the conflict in the system such as the failure on the background color in this project and the reason is very hard to be figured out. One other disadvantage for this system is that it takes many steps each time setting up the system such as launching all packages and ROS core, sourcing the dev. File to run the workplace of plugins, etc. So it is really hard to say if this one-platform-system in Linux is better than the regular ROS reality project which has Windows to work on the VR and Linux to focus on the sensors.

## 5.2 Recommendations

For further study and development, the following recommendations are given based on my personal experience in this project:

- Adding control part in the system  
Different from normal VR games or software, this project can not provide different views while changing the position of VR headsets. To bind the Kinect with a controlled robot, it will be able to control the robot's "head" by adding the orientation function in the VR.
- Using better hardware to improve the system performance  
Running SLAM and VR at the same time in one computer consumes a lot of system's power including CPU, GPU and RAM. So having better hardware will bring better results.

However, as mentioned previously, there is not much support from manufacturer of VR headset nowadays, the hardest part of this project and similar works in Linux is to have the access to VR headset and that is the main reason that users are more willing to use a second Windows computer even with the risk of data lag or delay because it's much more convenient to have VR work on Windows than Linux environment. So I recommend to continue this topic but have more focus on hardware side which has more space to be developed.

## Bibliography

Casper, J., & Murphy, R. R. (2003). Human-robot interactions during the robot-assisted urban search and rescue response at the world trade center. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 33(3), 367-385.

Bejczy, A. K. (1996, November). Virtual reality in robotics. In *Proceedings 1996 IEEE Conference on Emerging Technologies and Factory Automation. ETFA'96* (Vol. 1, pp. 7-15). IEEE.

Fong, T. W., Conti, F., Grange, S., & Baur, C. (2001, March). Novel interfaces for remote driving: gesture, haptic, and PDA. In *Mobile Robots XV and Telem manipulator and Telepresence Technologies VII* (Vol. 4195, pp. 300-311). International Society for Optics and Photonics.

Bakkay, M. C., Arafa, M., & Zagrouba, E. (2015, June). Dense 3D SLAM in dynamic scenes using Kinect. In *Iberian Conference on Pattern Recognition and Image Analysis* (pp. 121-129). Springer, Cham.

Yan, Z., Ye, M., & Ren, L. (2017). Dense visual SLAM with probabilistic surfel map. *IEEE transactions on visualization and computer graphics*, 23(11), 2389-2398.

Hofer, H., Seitner, F., & Gelautz, M. (2018, December). An End-to-End System for Real-Time Dynamic Point Cloud Visualization. In *2018 International Conference on 3D Immersion (IC3D)* (pp. 1-8). IEEE.

Klüssendorff, J. H., Ehlers, K., & Maehle, E. (2016). Visual Mapping in Light-Crowded Indoor Environments. In *Intelligent Autonomous Systems 13* (pp. 913-922). Springer, Cham.

Rünz, M., & Agapito, L. (2017, May). Co-fusion: Real-time segmentation, tracking and fusion of multiple objects. In *2017 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 4471-4478). IEEE.