CLOUD BASED ROS IMPLEMENTATION FOR INDOOR ROBOT

LIM JIA ZHI

A project report submitted in partial fulfilment of the requirements for the award of Bachelor of Engineering (Honours) Electrical and Electronic Engineering

Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

April 2019

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : _____

Name : _____

ID No. : _____

Date : _____

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"CLOUD BASED ROS IMPLEMENTATION FOR INDOOR ROBOT"** was prepared by **LIM JIA ZHI** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Honours) Electrical and Electronic Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature : _____

Supervisor : _____

Date : _____

Signature : _____

Co-Supervisor : _____

Date : _____

ACKNOWLEDGEMENTS

I would like to express my gratitude to my research supervisor, Mr Danny Ng Wee Kiat for his insightful advice, guidance and his enormous patience throughout the development of the research.

In addition, I would like to express my gratitude to my fellow friends from UTAR, Khor Jun Bin, Lim Wen Qing and Yong Cherng Liin for their invaluable help and support throughout the research.

ABSTRACT

Mobile robots nowadays are working in both the industry and non-industry environment. The advancement of mobile robotics allows for multiple use case of mobile robots such as warehouse management and personal assistant. All these robots have one commonality, which is the capability to work autonomously with minimum or no human intervention. For these robots to work autonomously, environment sensors such as laser scanning rangefinders and depth cameras, simultaneous localisation and mapping (SLAM) algorithm, path planners are implemented together in the robot. Multiple algorithms need to communicate with each other to ensure smooth operation of an autonomous mobile robot. Robot Operating System (ROS) is one of the widely used libraries to facilitate communication between processes in research. Offloading computation intensive tasks into cloud computing infrastructure will reduce the onboard processing resource required in a mobile robot, leading to an intelligent robot with lower weight and power consumption. In this study, we designed a simulation using Robot Operating System (ROS) to offload one resource intense process – simultaneous localisation and mapping (SLAM). A local client located in Malaysia fed laser sensor information to the cloud server hosting the SLAM process located in Singapore. The client and server were placed in the same network using VPN for ROS to operate normally. The average round trip time measured is 18.6540ms with a standard deviation of 23.7870ms. It is feasible to offload SLAM to a cloud server based on the result that we obtained. Extending on the result of our simulation, we conducted a study using a real robot with a similar implementation. Based on our result, it is feasible to implement ROS on cloud infrastructure for SLAM.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

AMI	Amazon Machine Image
API	application program interface
AWS	Amazon Web Service
CPU	central processing unit
EBS	Elastic Block Store
EC2	Elastic Compute Cloud
IaaS	Infrastructure as a Service
IP	Internet Protocol
LiDAR	light detection and ranging
LTS	long-term support
PaaS	Platform as a Service
PID	Proportional, integral, derivative
RAM	random-access memory
ROM	read-only memory
ROS	Robot Operating System
RTT	round-trip time
SaaS	Software as a Service
SLAM	simultaneous localisation and mapping
SSH	Secure Shell
TCP	Transmission Control Protocol
VPN	virtual private network

CHAPTER 1

INTRODUCTION

1.1 General Introduction

Robots have been applied in large scale both in fixed or mobile form. However, the robots are mostly constrained to operate at a fixed or known environment, dealing with well-defined, highly specialized or repetitive tasks. For tasks in an unknown environment such as search and rescue robots or household service robots, human intervention or remote control is usually required.

Recently, to support Internet of Things (IoT) and mobile applications, wireless technology and cloud computing infrastructure have gone through rapid development. Now is a suitable time for cloud robotics as the underlying technologies are getting matured. The prospects of robots are becoming smarter by having a “brain” in the cloud to tap into capabilities like big data and collective learning (International Federation of Robotics, 2017).

1.2 Problem Statement

Mobile robots are limited to low intelligence tasks due to limitation of onboard computation power, data storage, and battery which addition of any of them would require increase in robot size, weight and cost. Connecting robots to cloud computing infrastructure for computation offloading will allow robots to carry out more complex tasks, but current cloud computing platforms are built for web applications, there is no platform built purposely for robot applications. Hence, cloud implementation of robotic application is challenging and not straightforward.

1.3 Importance of Study

This study may offer insight into current status of cloud robotics, a different implementation of ROS on cloud computing infrastructure and suitable methods to offload computation of common robot tasks like SLAM, navigation, and recognition.

1.4 Aims and Objectives

This study aims to study the feasibility of implementing ROS on cloud computing infrastructure specifically for indoor robots. With access to cloud processing power, indoor robots can target higher complexity tasks while keeping the onboard hardware small and affordable. The detailed objectives of this research are:

- Investigate the latency of cloud-based implementation
- Study the feasibility of robot teleoperation using cloud infrastructure as the medium
- Study the feasibility of offloading computation onto cloud server or SaaS
- Develop an indoor robot with the capabilities to do SLAM

1.5 Scope and Limitation of the Study

Limited by cost, the cloud computation was run on free tier Amazon EC2 t2.micro instance, which has very limited computation resources and network capability, and lower performance compared to the local machine. Hence, only RTT of ROS message packets was used for performance evaluation, rather than actual performance improvement of the ROS system.

And due to time constraint, only SLAM algorithm was run on the setup to study its feasibility. But there are many more algorithms, such as path planner, object recognition and sound source localization, that needed to be integrated to build a fully autonomous indoor robot.

1.6 Contribution of the Study

This research study the feasibility of direct implementation of ROS system on cloud via VPN, by treating cloud computing infrastructure as one of the ROS computers. The research goal is designed to find an easier way to integrate existing ROS system with cloud computing infrastructure to improve the performance while lowering hardware requirement and enable more capable and cost-effective robot system.

1.7 Outline of the Report

Chapter 1 provides an overview of the current robot system requirements and the problems this research are trying to solve.

Literature review about ROS system concepts, different methods of implementation of cloud robotic with ROS and SLAM algorithm were highlighted in Chapter 2. Chapter 3 explain the setup of the computing infrastructure and building process of a mobile robot for computation offloading experimenting.

Chapter 4 shows the result of cloud implementation setup and mobile robot fabrication. Then, measurement results from computation offloading are presented and analysed.

Finally, in Chapter 5, the feasibility of cloud implementation of ROS is concluded, and suggestions about methodology improvement and future research directions are provided.

CHAPTER 2

LITERATURE REVIEW

2.1 Robot Operating System (ROS)

ROS is a middleware that provides an interface for data passing and communication between computing processes and robots. It is not a complete operating system but an abstraction layer or meta-operating system that run on top of Ubuntu to abstract hardware and low-level control from software application.

ROS packages are modular to promote code reusability and simplify application development. Moreover, active community from around the world have contributed packages on top of the core system, substantially extended the out-of-the-box capabilities of ROS. More than 3000 packages are contributed and accessible by the public, the packages range from low-level drivers, development tools to complex algorithms with industrial-level reliability (Open Source Robotics Foundation, 2018).

ROS is very popular, almost being the de facto standard middleware, in robot development in both academic and industry sector. ROS have allowed researchers and developers to create robot systems efficiently using well developed and defined packages, and focus on core application rather than standard functionalities like drivers of robots.

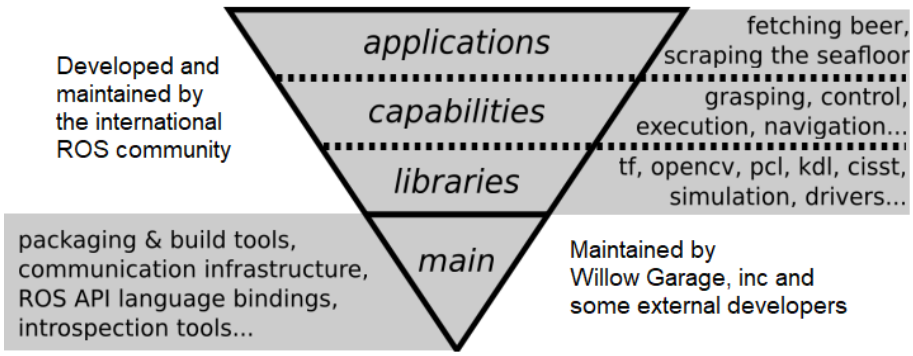

Figure 2.1: ROS development levels (Mösenlechner, 2012)

ROS system consists of a Master node and other multiple nodes. Master is the main control node that manages communication between nodes by tracking every node and the data they output or requested. However, Master does not relay messages between nodes but connect nodes for peer-to-peer data exchange via TCP protocol.

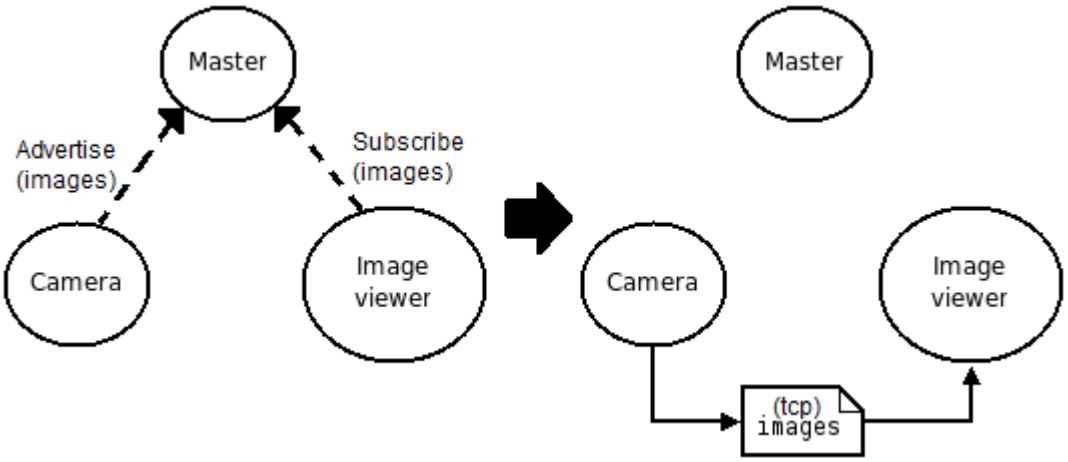

Figure 2.2: ROS messaging mechanism (Wu, 2018)

There are three methods of communication between nodes in ROS, the first and simplest one is topic which is based on publish-subscribe mechanism, providing asynchronous many-to-many communication. Then, service, a request-reply mechanism for synchronous communication normally used in one-time exchange of data or command. Finally, action allows sustained communication for long-running goal-based task server that provides feedback on task progress and preemptable by client (Marguedas, 2018).

In addition to providing an interface for different communication requirements, ROS also comes with built-in development tools such as command-line tool like roslaunch to launch multiple ROS nodes and set parameters value at the same time. Rqt to analyze nodes, topics and services as well as rviz for 3D visualization of sensors data and robot conditions (Mösenlechner, 2012).

Moreover, ROS has good integration with other open source libraries related to robot applications. Example of libraries available is Gazebo, a 3D robot simulator in virtual world to simulate written ROS nodes functionality directly or with some minor changes on the code. ROS also provides integration with OpenCV, a widely used computer vision library, and MoveIt!, a library for motion planning on robots with different driving mechanisms (Open Source Robotics Foundation, 2018).

2.2 Different Implementation of ROS on cloud infrastructure

Commercial cloud service provider like Amazon Web Services, Google Compute Engine and Microsoft Azure have been widely used to carry out highly intensive computation on demand. However, most cloud platforms are accessible via API only which is well suited for web applications but not for robot applications. ROS requires bi-directional connectivity between all machines and ports (Bhadani, 2018), most cloud services do not support such configuration. Hence, cloud implementation of ROS has to be on IaaS that provide flexible control on configuration of the computing machine.

2.2.1 RoboEarth

RoboEarth is intended to create "World Wide Web for robots" that allows sharing of knowledge critical to robot systems such as models of object and environments, and tasks like grasping. To simplify the creation and sharing of the knowledge to be as easy as sharing content on the internet, RoboEarth created a standardised language for knowledge encoding, and method to determine feasibility of specific knowledge on the robot. With RoboEarth, robots can access the database consisting of well-defined knowledge about objects, environments and actions, and then make decision automatically about whether the robot has the requiring capabilities to utilize the knowledge.

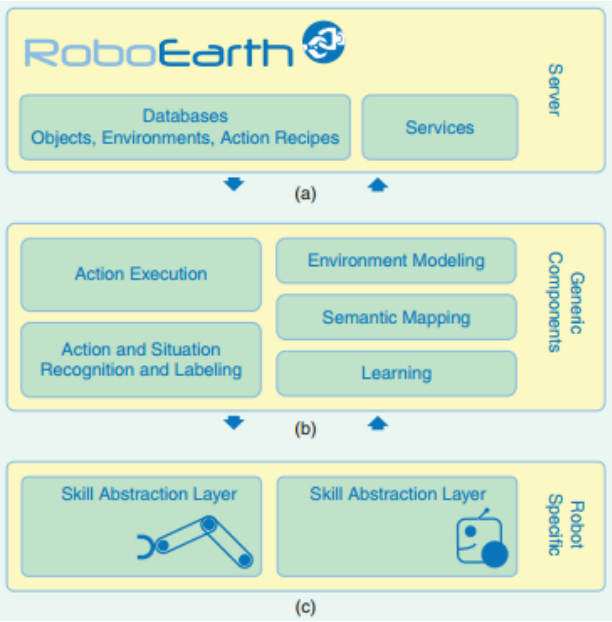

Figure 2.3: RoboEarth three-layered architecture (Waibel, et al., 2011)

2.2.2 Rapyuta

Cloud robotics platform that allows robots to offload intensive processing onto the cloud by cloning robots onto cloud and also provide access to RoboEarth repository. It is an open source project and also known as RoboEarth Cloud Engine. Rapyuta aims to provide an end-to-end cloud robotics platform that runs on elastic cloud computing infrastructure. Every robot in the system will be cloned to the cloud to outsource the processing (Mohanarajah, 2015), and communication between robots can be carried out via their clone in the cloud, providing high speed and reliable inter-robot communication.

Communication between Rapyuta and robots is based on WebSocket, allowing bidirectional and full duplex data exchange. Core component for processing of Rapyuta is computing environment running on cloud which is based on Linux Containers. Linux containers provide security, isolation of process and system resources, easy configuration for scalability and portability, and processing at native speed. It can be used in large scale applications such as visual processing or at a scale as small as just relaying control signals. The computing environment is a complete ROS environment running roscore and ROS parameter server. It can be used to run any number of ROS nodes and support ROS inter-node communication just as running ROS on a local computer, hence, it is compatible with most existing ROS packages.

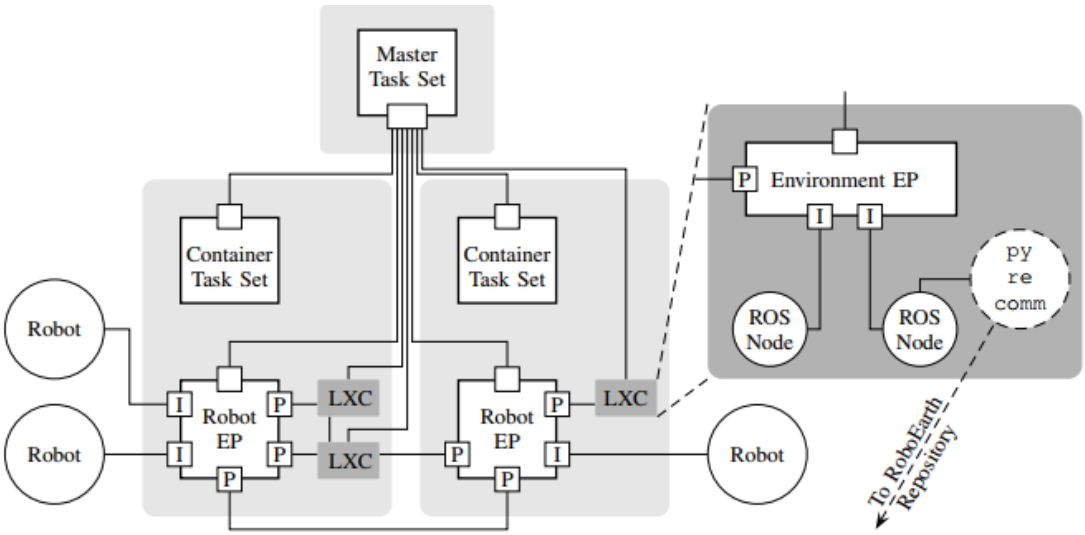

Figure 2.4: Example configuration of Rapyuta (Mohanarajah, et al., 2015)

Each light grey box represents a computing machine and there are three machines in the example in Figure 2.4. The Master Task Set is the main core that

controls every task sets and manages communications between Rapyuta and robots, only one Master Task is required for every Rapyuta platform. The EP represents endpoint process which is controlled by Master for communication between and within internal and external processes. EP also provide conversion of data format for communication between internal and external processes to ensure data format compatibility. LXC represent Linux containers which are where ROS nodes are run on. The example showed Rapyuta access to RoboEarth repository.

Rapyuta is available on Github as an open source project written in Python and C++, however, it was deprecated since 2015 due to change of robot requirements and dependent technologies. The core developers, Rapyuta-Robotics company is rebuilding a better version of the platform (rapyuta-robotics, 2015).

2.2.3 DAvinCi

DAvinCi stands for Distributed Agents with Collective Intelligence. The agents are the robots and collective intelligence is on the cloud. The framework aims augment robotics system in large environment by connecting ROS and Hadoop cluster.

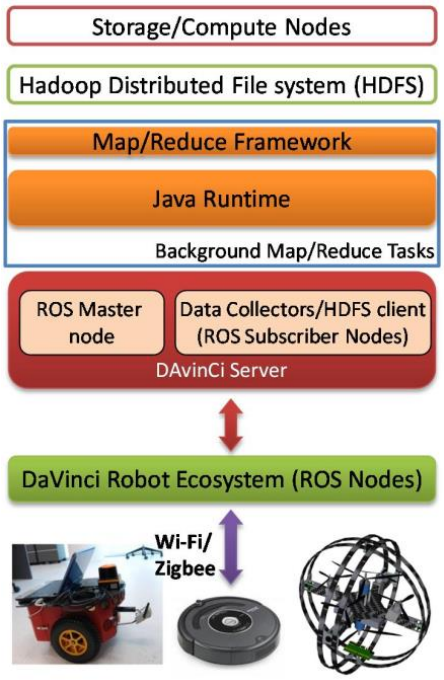

Figure 2.5: DAvinCi Architechture (Arumugam, et al., 2010)

In the architecture, a team of robots, shown at the bottom of Figure 2.5, all carrying essential sensors such as gyroscope and encoders for individual odometry

tracking and WiFi chip, can carry different sensors such as camera and laser rangefinder. The robot team communicate internally and with the DAvinCi server via ROS communication protocol. Sensors data will be sent to the server and further relayed into Hadoop for storage and processing. DAvinCi server connects robot team and users on ROS platform to Hadoop cluster for computation offloading. DAvinCi is implemented on FastSLAM algorithm to observe the improvement of performance, Figure 2.6 showed the reduction of time taken on execution with different amount of particles as the number of computing nodes increased. Execution time of the same algorithm on local computer can be assumed to be same as the time taken on running on single node, hence, the research showed that great speedup can be achieved by offloading computation onto the cloud.

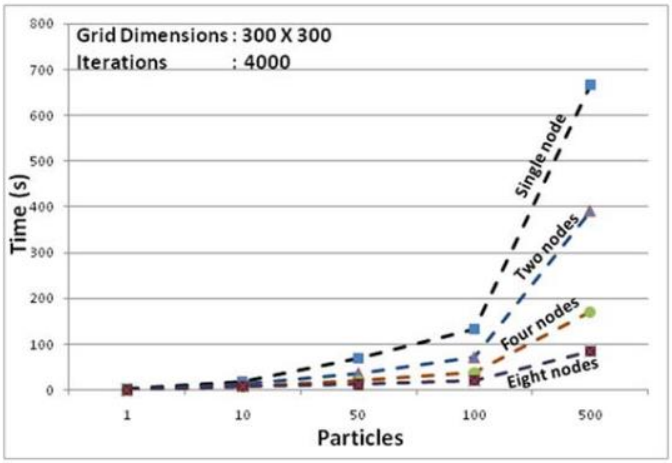

Figure 2.6: Execution time of FastSLAM in Hadoop vs. number of nodes (Arumugam, et al., 2010)

2.2.4 Robot Web Tool (RWT)

An open-source project that aims to provide interoperability and portability of robot applications to different systems. Based on ROS and its package, Rosbridge which allows interfacing of ROS functionality with programs outside of ROS using JSON based commands. It helps to overcome ROS disadvantages in platform dependencies as ROS only support Ubuntu and Debian officially which is not widely used compared to Windows and MacOS. RWT combine ROS with web technologies to make it more accessible to more developers who are expert in web application development.

RWT communicate over WebSockets and carry client-server architecture. WebSockets protocol provides operability on web browsers which run on

heterogeneous platforms, from different operating systems on computer to smartphones. RWT provides web client library named roslibjs for access of ROS features such as transform and URDF, and allow secured and efficient communications between robots and users.

Due to the limitation of WebSocket, sending of raw or binary data in high bandwidth is impractical. RWT has its own method to stream high-bandwidth messages such as transform, image and point cloud. Bandwidth requirement for transform is reduced by having a layer of ROS package that precompute transforms on demand and publish only when a change of transform over a specific limit occurred.

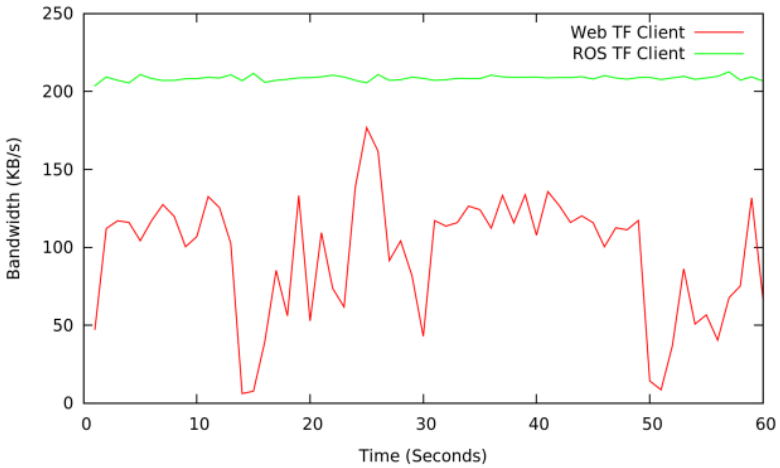

Figure 2.7: Average TF Bandwidth for ROS and Web (Toris, et al., 2015)

As shown in Figure 2.7, for transform of a simple robotic arm with 6-DOF, the extra layer reduced the bandwidth required from an average of 208.5 KB/s to 96 KB/s. Bandwidth for streaming of image, point cloud and generic message is reduced by utilizing embedded compression codecs in HTML such as MJPEG and VP8 codec. The web functions required by RWT are supported on modern browsers like Firefox and Chrome.

2.3 Simultaneous Localisation and Mapping (SLAM) algorithm

SLAM is a crucial feature of autonomous mobile robot. The process SLAM is designed to build a map, 2D or 3D, of an unknown environment while navigating through the environment (Nguyen Hoang Thuy & Shydlouski, 2018). The navigation process is run at the same time based on the map generated, hence the algorithm is named simultaneous localisation and mapping. There are multiple SLAM algorithms

available to be integrated into ROS system, namely gmapping, cargtographer, hector_slam and slam_karto (ROBOTIS, 2019).

2.3.1 Google Cartographer

The algorithm used in this project is Cartographer, an open-sourced project developed by Google which provides real-time SLAM capability. Cartographer is able to compute loop closure constraints with lower computational power while mapping large environment in real-time (Hess, et al., 2016).

Google Cartographer support four input, laser scan, odometry pose, IMU data and fixed frame pose. Laser scan data will first be downsampled by voxel filter which put raw points in constant-sized cubes and output only the centroid of each cube. Size of voxel filter can be set through ROS parameter, lower cube size leads to lesser data point and hence lower computation. In 2D SLAM, IMU input is optional while in 3D SLAM, IMU input is a must for initial orientation prediction to reduce complexity of scan matching.

The system is separated into local SLAM and global SLAM. Pose extrapolator uses odometry pose and IMU data for initial pose guessing. With the initial pose guess, local SLAM inserts new scan into submap. The scan matching process is based on Ceres Solver, an open-sourced C++ library for non-linear least squares minimization, resulting in scan pose relative to current submap (Agarwal & Mierle, 2019). A motion filter drops matched scan that do not have sufficient motion to reduce number of scans in a submap. Result of local SLAM, the submap will drift over time, so submaps cannot be too big to contain the drift below resolution. Submaps are normally stored as probability grids but can be configured to store as truncated signed distance fields (TSDF).

Global SLAM runs in the background to compensate the drift. It is a pose graph optimization process that create constraints between submaps that are close to each other and are good match. With global SLAM, the submaps are aligned and linked to form a global map (Cartographer ROS, 2019).

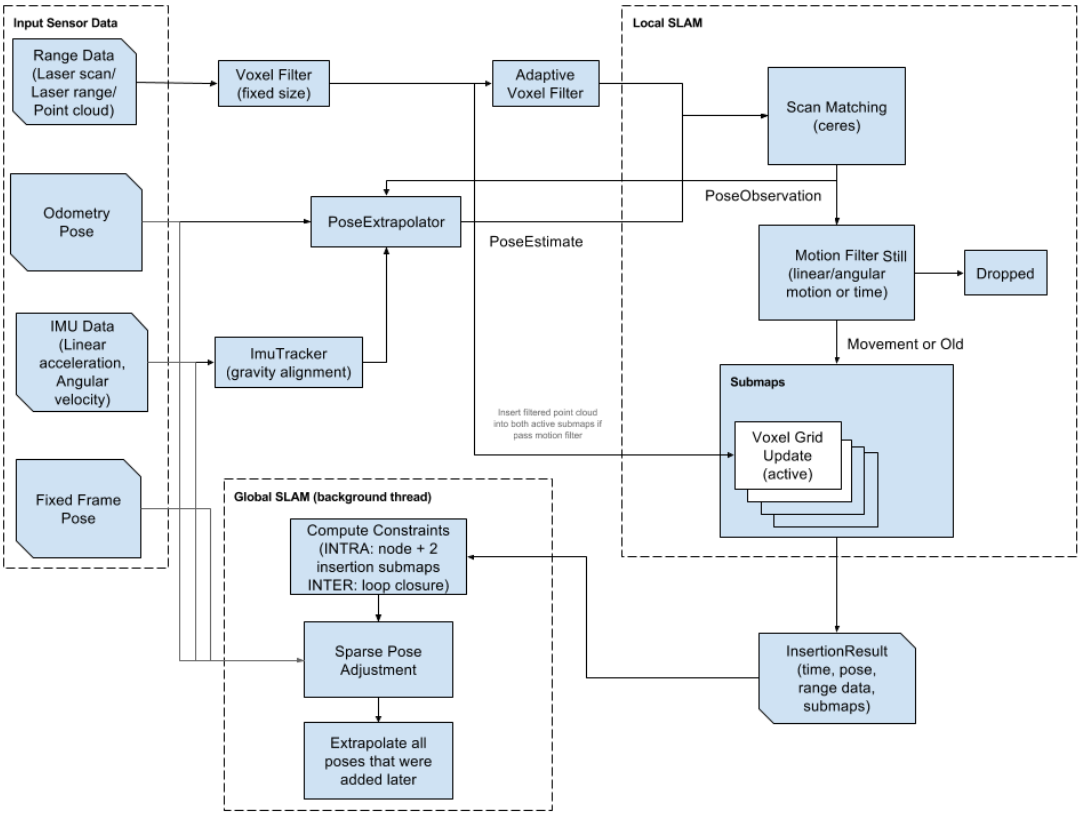

Figure 2.8: Google Cartographer high level system overview (Cartographer ROS, 2019)

In addition, cartographer stands out by supporting IMU compensation for tilting if the robot runs on uneven ground as well as 3D SLAM using 3D laser scanner. As compared to other algorithms available, Cartographer has lower CPU usage while producing accurate map with higher rate of successful loop closing (Coroiu & Hinton, 2017).

2.4 Summary

Cloud robotics has been a popular topic in research and multiple implementations of ROS were studied. From the literature review, Rapyuta platform is the most suitable architecture for this project as it provides efficient utilization of computing power of cloud computing infrastructure while keeping real-time response needed for robot system. However, all implementations studied require some extra configuration or overhead to connect robots to cloud infrastructure, increasing the complexity to develop a ROS system.

Google Cartographer was chosen to be the process for computation offloading because SLAM is one of the key capabilities for an autonomous mobile robot. Among SLAM algorithms, Cartographer has better performance while supporting IMU compensation and 3D SLAM, providing the option to scale up for more complex applications.

CHAPTER 3

METHODOLOGY AND WORK PLAN

3.1 Introduction

To study the implementation of ROS on cloud, the infrastructures had to be set up before any test can be carried out. This methodology included methods to launch cloud computing instance on Amazon Elastic Compute Cloud (EC2), then to configure ROS and OpenVPN on both cloud server and local machine client. A mobile robot equipped with ROS installed computer was created to study feasibility of the implementation in real world scenario by running SLAM in university hallway.

3.2 Initialisation of Amazon Elastic Compute Cloud (EC2)

To implement ROS on cloud infrastructure, Amazon EC2 was chosen as the infrastructure provider. AWS Free Tier for new user is available for one year after registration, instance named t2.micro is free for 750 hours a month. The free tier instance was chosen to test the ROS implementation. The early testing was on messages passing latency which do not require high computation resources, hence, the free tier instance was sufficient.

To create an EC2 instance or cloud computer, AWS provided a 10-Minute Tutorial for beginner, the tutorial “Launch a Linux Virtual Machine” provided a step-by-step tutorial on setting up a Linux machine on Amazon EC2 (AWS, 2018). Amazon Machine Image (AMI) is a template that comes with OS configured for EC2 to start the computing instance easily, Ubuntu 18.04 LTS AMI was provided in AWS Marketplace for free and the AMI was used in this project. Amazon cloud computing resources are available globally in regions such as United States, China, Europe and Asia Pacific. Singapore server was selected to host the EC2 instance due to its geographical proximity to Malaysia.

After choosing the AMI, instance type which consists of different CPU, RAM, ROM and network speed, was to be chosen to suit different applications and requirements. In this project, free t2.micro with 1 vCPU, 1 GiB RAM, Amazon Elastic Block Store (EBS) storage and low to moderate network performance was chosen. The EC2 instance was ready to be launched after AMI and instance type were chosen.

Public and private key pair for remote SSH access was required, new key pair was created by AWS and downloaded to local computer that will be used to SSH access the EC2 instance. When the instance was successfully launched, on instances viewing console of AWS, status and description of the instance could be monitored. In order to connect to the instance via SSH, IPv4 Public IP of the instance was copied down as the access point. With IPv4 Public IP, key pair, and SSH client software on local computer, access into the instance can be initiated.

After the EC2 was up and running Ubuntu 18.04 LTS, ROS melodic was installed into the Ubuntu with the same procedures like installing on local computer, just a barebone version of ROS was installed rather than a desktop full version of ROS because GUI tools were unnecessary on the server (ROS Wiki, 2018).

3.3 Configuration of OpenVPN on AWS EC2

While EC2 public IPv4 allows SSH access, the IP could not be used on ROS communication. Hence, to connect ROS running on local computer, robots and Amazon EC2 in a same network with addressable IP, a VPN was required to be set up on the EC2. The VPN server running on the cloud facilitated connection of ROS system by placing local clients and cloud server into same network. OpenVPN is an open source VPN software available for different operating system including Ubuntu used in this project.

Different methods were available to setup OpenVPN, but to simplify the setup, a script by Angristan was used to automate the setup and configuration of OpenVPN on EC2 (Angristan, 2018). The script prompted for user input on configuration of the OpenVPN and had a recommended choice on every configuration throughout the installation. After the script completed the configuration, it started OpenVPN server on the computer and generated a client configuration file to be used on client side for authentication.

The file was sent to local computer via SCP which securely transfer file using SSH between remote computers. Then, OpenVPN client package was installed in local computer via package manager APT. Finally, OpenVPN client was started using the configuration file generated from the script. A new network tunnel interface that connects the server on cloud and client on local machine in a VPN was created.

3.4 Configuration of ROS, EC2 and OpenVPN for Cloud communication

After both local and cloud machine were installed with ROS and OpenVPN, further configuration was required to allow ROS Master to run on EC2 while allowing ROS on local machine to connect to the ROS Master. Cloud machine and local machine had to have same ROS_MASTER_URI which was VPN IP of cloud machine.

Then, on AWS EC2 console security group inbound setting, port 1194 and 11311 were opened for UDP and TCP protocol for usage of OpenVPN and ROS. After this, the configuration was completed, the setup was tested by running roscore on cloud machine and listing ROS topic on local machine to test the connectivity, the setup was successful as topic list is shown on the command prompt.

3.5 Design of Mobile robot

To try the implementation of ROS on real-life scenario, a physical robot was required. TurtleBot 3 Burger from Robotis is a affordable mobile robot designed to explore ROS and design robot applications. Robotis opensourced software and hardware of the robot, allowing easy modification or customization of the robot.

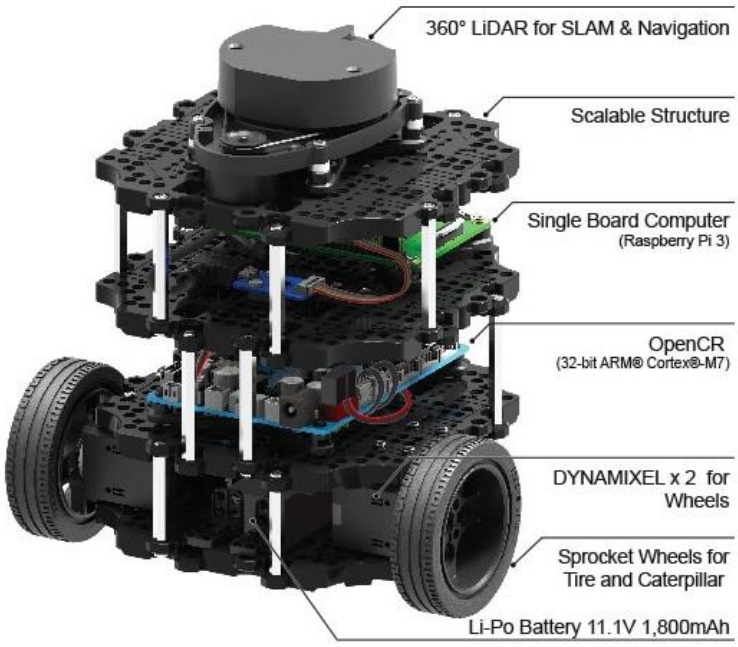

Figure 3.1: Robotis TurtleBot 3 Burger running ROS

The robot contains a LiDAR sensor, a main controller board OpenCR, a Raspberry Pi 3, 9-DOF inertial measuring units and two 360 degrees servo motor. With the components, TurtleBot 3 Burger is suitable to be used for studying ROS. Moreover, with the open-sourced software and hardware, a similar robot could be self-constructed with extra features such as adding depth sensor, microphone array and larger size.

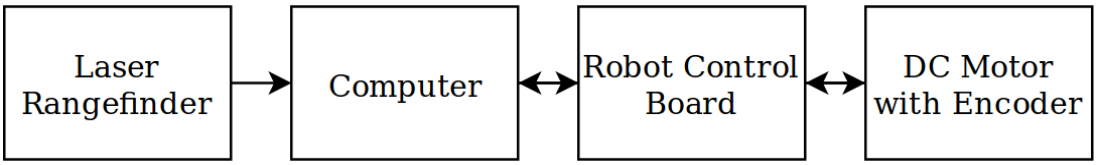

Figure 3.2: Block Diagram of Proposed Robots

The robot required a computer with resources enough to run Ubuntu, USB ports to connect to peripherals, and WiFi or cellular hardware for internet connection. The computer was connected to a main control board that handles the low-level hardware control like motor for wheels or arms and IMU sensor data reading. LiDAR was required for 2D SLAM as well as navigation.

3.6 Analysis of Messaging Latency

To study the feasibility of cloud implementation of ROS, latency is the deciding factor as robot applications require real-time performance and minor delay in message passing will have severe effect on the application. Two messaging routes were tested.

- Local computer and cloud computer
- Two local computers via cloud computer

The message latency was evaluated based on RTT. The RTT contain propagation and overhead time taken to send and acknowledge reception of the message. RTT was captured using Wireshark. Since ROS data was exchanged within the VPN with TCP, Wireshark filters were applied to filter out the TCP originate and end at cloud and local machine exposed IP address. After filtering, RTT was obtained from the packet list pane.

Wireshark is an open-source packet analyser released under GNU General Public License. It is the most widely-used packet analyser, the de facto standard for

many commercial and educational institutions. Wireshark support live capture of packets of hundreds of protocols and offline inspection of the captured packets. Moreover, community developed plugins are available if the protocol to be inspected is not supported out of the box. Fortunately, ROS topic packet dissector was available as built-in plugin under the name tcpros. New protocol dissector was applied from Analyze>Enabled Protocols and ticking the box beside the protocol as shown in Figure 3.3. Moreover, coloring rules was added from View>Coloring Rules as shown in Figure 3.4 to visualize the packets better.

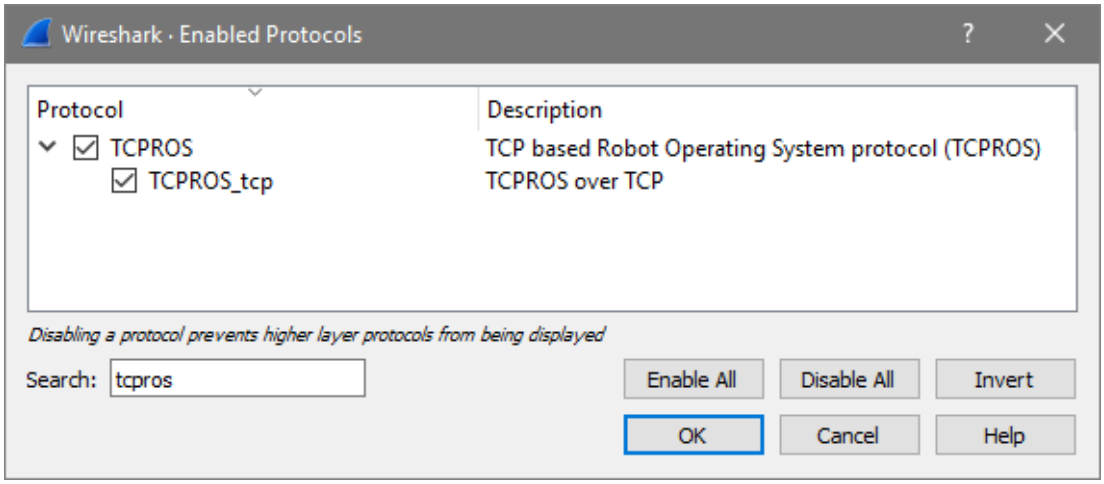

Figure 3.3: Wireshark enabled protocols pop up window

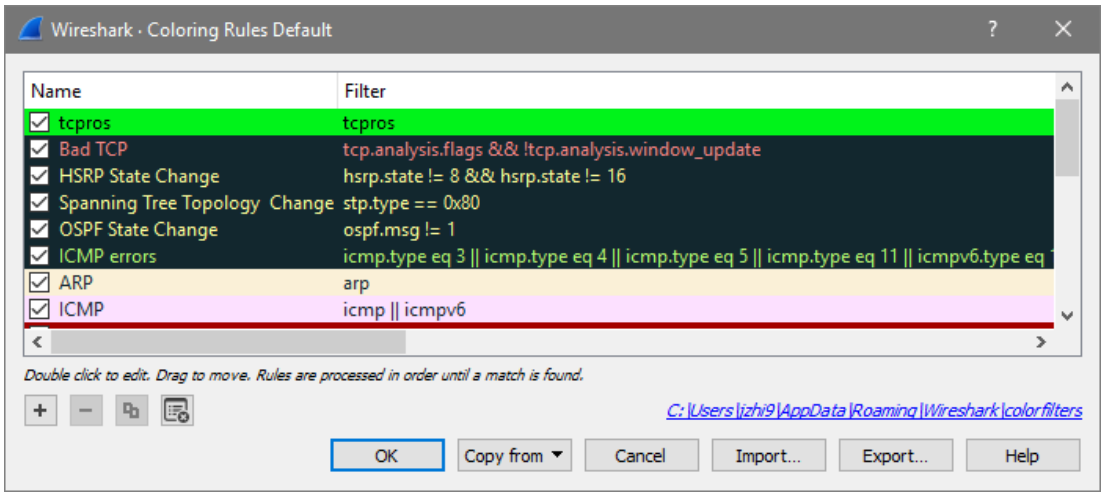

Figure 3.4: Wireshark coloring rules pop up window

After enabling the tcpros protocol and applied colouring rules onto the protocol, ROS topic packets captured were highlighted in bright green colour as shown in Figure

3.5. In addition, the tcpros protocol dissector also parsed contents of the packet and displayed it in packet details pane in the middle of the interface. The dissector simplified the packets inspection by categorizing packets and displaying their contents in human readable form.

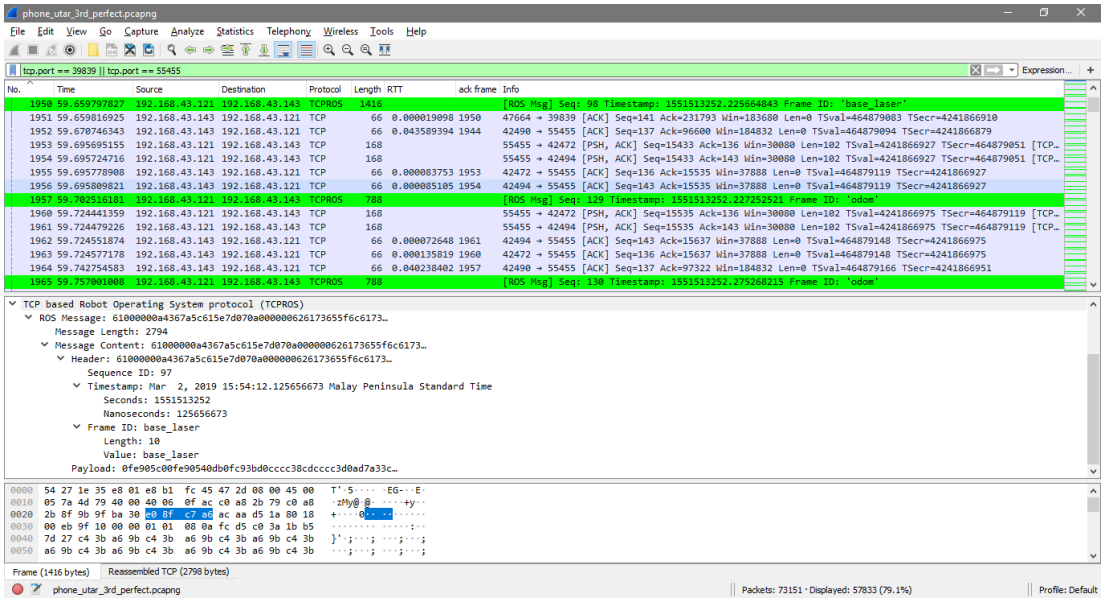

Figure 3.5: Wireshark panel after adding tcpros protocol and coloring rule

3.7 Computation Offloading

Cartographer SLAM was run on the cloud server while sensor data are streamed to the cloud running from a local client. Revo LDS laser data from the cartographer dataset (Cartographer, 2019) was used in this test. The construction of the map was view in real-time in the local client using rviz, a visualization software that come together with ROS. Cartographer demo roslaunch file which launch all ROS nodes in a single machine was separated into two file as shown below, where the first file launch cartographer_node and cartographer_occupancy_grid node on Amazon EC2 instance for SLAM computation while the second file launching rosbag and rviz were launched on local machine to playback laser data and visualize the SLAM process.

```
<launch>
  <param name="/use_sim_time" value="true" />

  <node name="cartographer_node" pkg="cartographer_ros"
      type="cartographer_node" args="
          -configuration_directory $(find
             cartographer_ros)/configuration_files
          -configuration_basename revo_lds.lua"
      output="screen">
    <remap from="scan" to="horizontal_laser_2d" />
  </node>

  <node name="cartographer_occupancy_grid_node"
       pkg="cartographer_ros"
      type="cartographer_occupancy_grid_node" args="-resolution
            0.05" />
</launch>
```

SLAM simulation launch file on Amazon EC2

```
<launch>
  <param name="/use_sim_time" value="true" />

  <node name="rviz" pkg="rviz" type="rviz" required="true"
      args="-d $(find
            cartographer_ros)/configuration_files/demo_2d.rviz" />
  <node name="playbag" pkg="rosbag" type="play"
      args="--clock $(arg bag_filename)" />
</launch>
```

SLAM simulation launch file on local computer

In the cloud-based implementation, Amazon EC2 instance, local computer and robot computer were connected within a VPN as shown in Figure 3.6. Both Amazon EC2 and robot computer were controlled by remote computer through SSH. The robot state node and laser node were running on robot computer. Local computer only acted as robot motion controller, cartographer node was running on Amazon EC2 to test computation offloading onto cloud, data were passed between computers and Amazon EC2 through VPN tunnel.

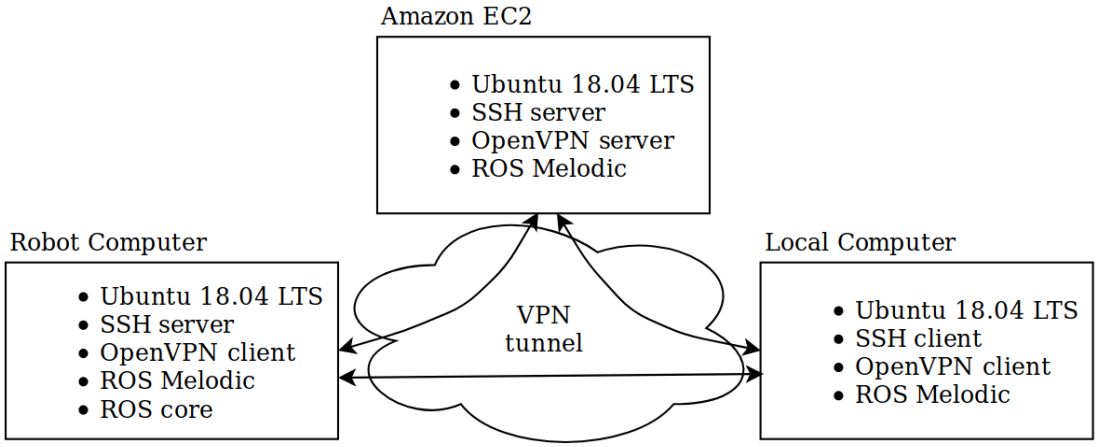

Figure 3.6: Setup of cloud-based ROS with mobile robot

3.8 Summary

The objective is to study the feasibility of cloud implementation of ROS, so most parts of the project were on experimenting and trying get ROS system running with cloud infrastructure. Then, ROS equipped mobile robot was built to experiment the implementation in real application. Finally, method to improve robot system performance utilizing cloud infrastructure was tested by moving SLAM computation directly onto EC2 in simulation and real robot.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 EC2 Configuration Result

Amazon EC2 instance was configured as stated in methodology. The instance details can be visualized from Amazon EC2 management console as shown in Figure 4.1. An Amazon EC2 t2.micro instance was launched with Ubuntu 18.04 LTS AMI. It was hosted from Singapore server with accessible public IPv4 address of 54.254.210.201, the IP address will be changed for every restart of the instance. Amazon EC2 instance contained only the computing resources, the CPU, RAM and networking capability. The configuration came with a non-volatile memory, Amazon EBS volume that stored system files and data for the EC2 instance. The volume had 8 GiB storage size and 100 input/output operations per second (IOPS). The storage had limited size and speed, but it was enough for our testing.

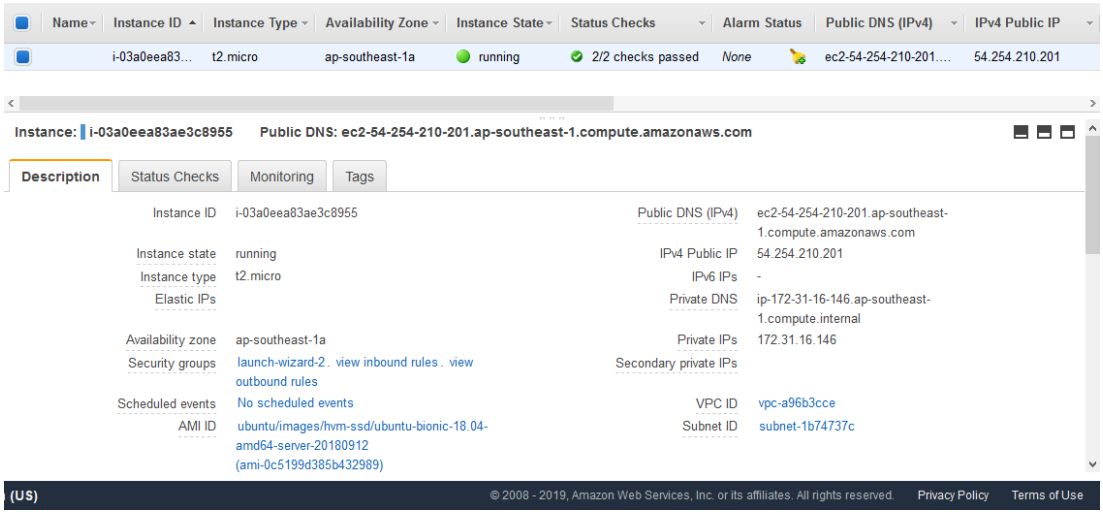

Figure 4.1: AWS management console EC2 instance

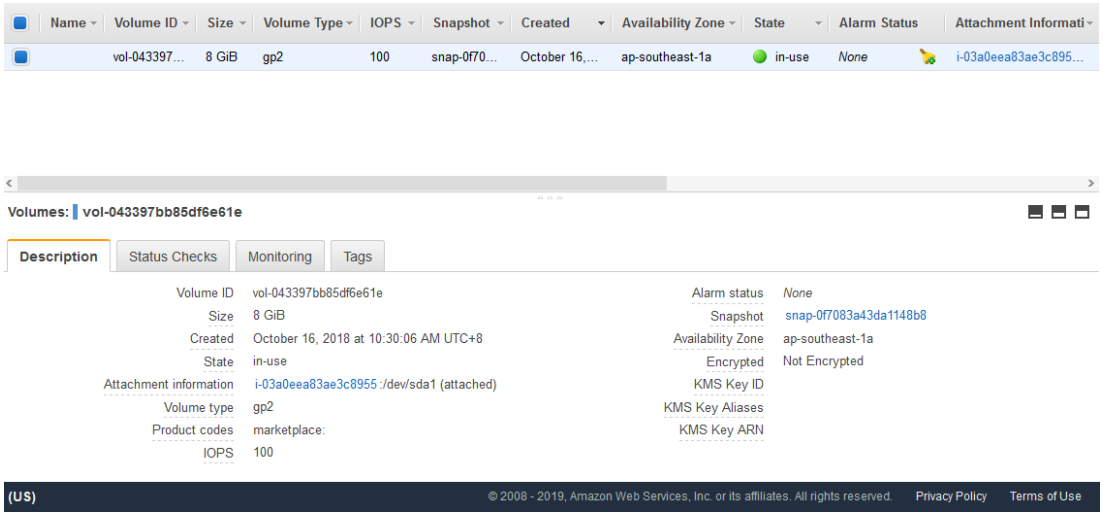

Figure 4.2: AWS management console volume

The setup of the cloud computing infrastructure system was as shown in Figure 4.3. Amazon EC2 and EBS were independent, hence, the Ubuntu 18.04 LTS image configured and stored in EBS could be launched with different instance type for different computation resources needed. Instance type was changed to t2.small with 1 vCPU and 2 GiB RAM when compiling Google Cartographer from source code because t2.micro instance would crash due to insufficient RAM. Besides, the EC2 instance was associated with security group launch-wizard-2 which the details were shown in Figure 4.4. AWS security group was like a firewall, having rules for inbound and outbound network traffic. Default setting of security group was outbound connection opened for all traffic and inbound connection only allowed TCP through port 22 for SSH. Extra rules were added for inbound connection, which port 11311 was opened to all IP for TCP and UDP connection for ROS master while port 1194 was opened to all IP for UDP connection for OpenVPN.

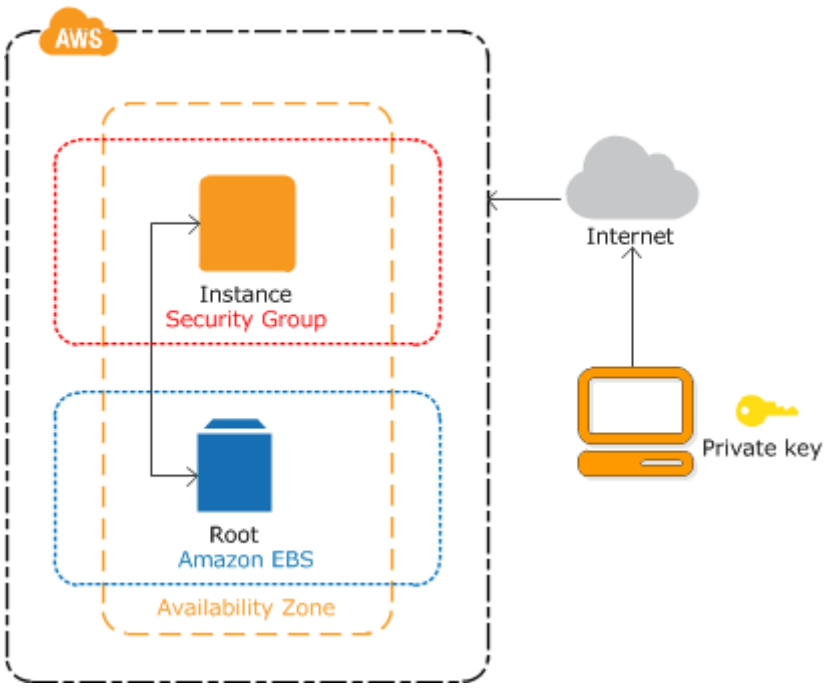

Figure 4.3: AWS cloud computing infrastructure system overview

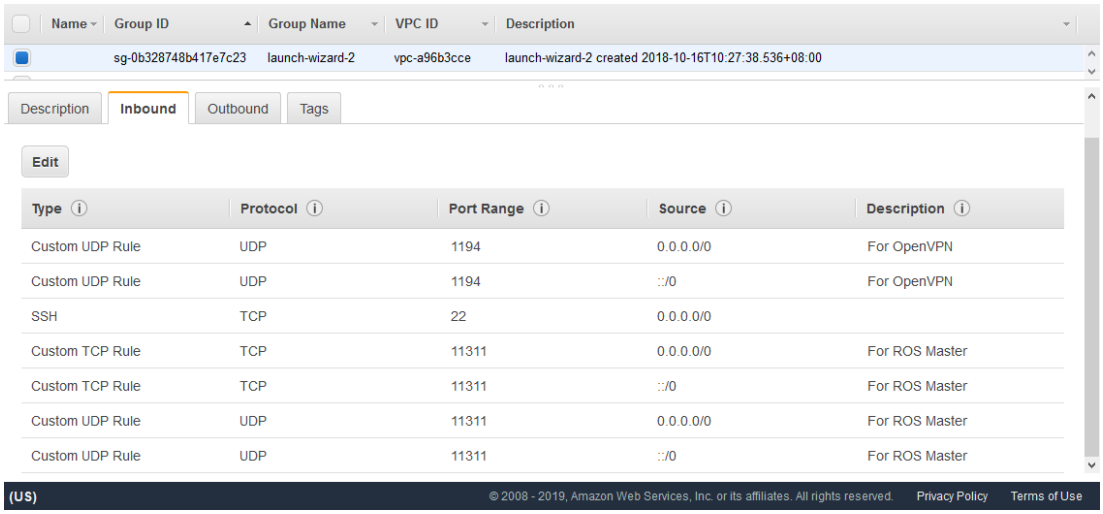

Figure 4.4: AWS management console EC2 security group

4.2 Cloud Implementation of ROS System

In the implementation shown in Figure 4.5, the local machine and Amazon EC2 instance were both running Ubuntu 18.04 LTS. Windows on the right are the SSH of Amazon EC2 instance that acted as ROS Master running roscore and also turtle_teleop_key ROS node which published control signal to turtlesim. The turtlesim node was running on local machine, receiving control signal from cloud and moved the turtle in TurtleSim simulation at the bottom right window. Figure 4.5 shows that

the system configuration is feasible and ROS system can run directly on VPN as expected.

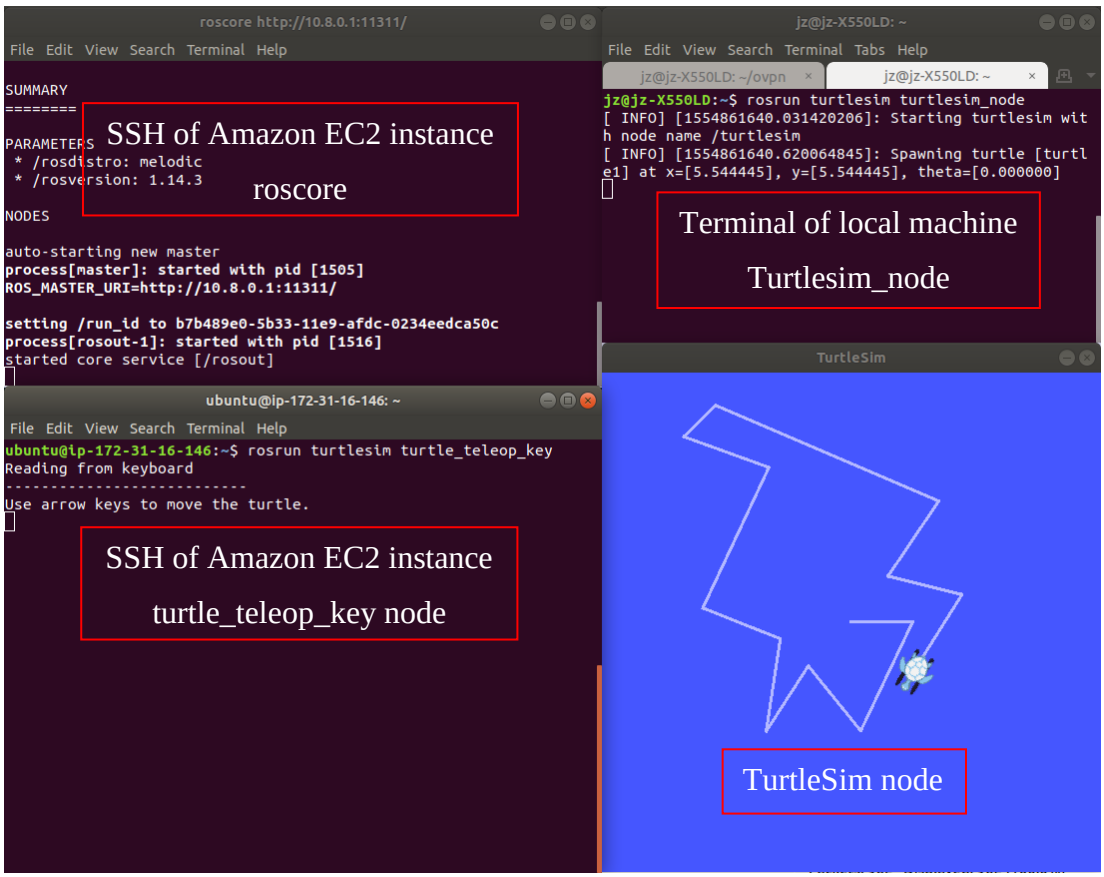

Figure 4.5: Screenshot of SSH of EC2 instance and turtlesim on local Ubuntu

4.3 Messaging Latency Measurement

Wireshark filter was applied to filter down to only data exchange between cloud server and local machine ROS tcp port. Packet from 10.8.0.1 to 10.8.0.2 was the control signal from server to local machine and from 10.8.0.2 to 10.8.0.1 was the acknowledgement from local machine. Figure 4.2 shows that RTT for turtle_teleop_key ROS node command signal was around in the range of 20.18 ms to 51.75 ms. The RTT was short enough to provide responsive remote control of robot. However, further experiments are required for different types and sizes of data.

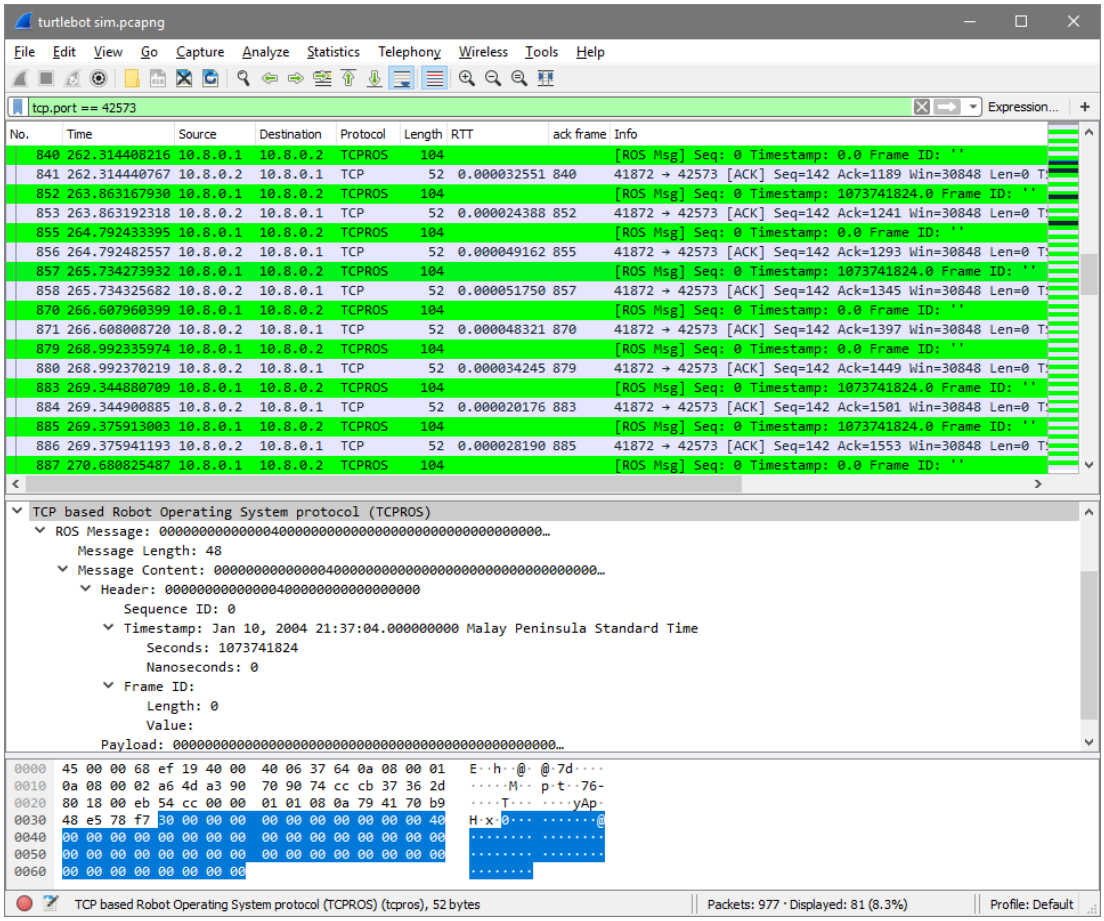

Figure 4.6: Wireshark capturing ROS messages

4.4 Mobile Robot

A mobile robot was built to test the feasibility on real world scenario. The components of the robot were as shown in Figure 4.7. The computer ran on Ubuntu 18.04 LTS with ROS Melodic, and were connected to laser rangefinder and robot control board via USB. Robot control board controlled DC motor velocity based on computer command and provided encoder feedback for odometry. Laser rangefinder provided 240-degree laser range scanning data for SLAM. The computer acted as the bridge between the robot with other ROS processing nodes, it collected and published sensors data and reacted to control command.

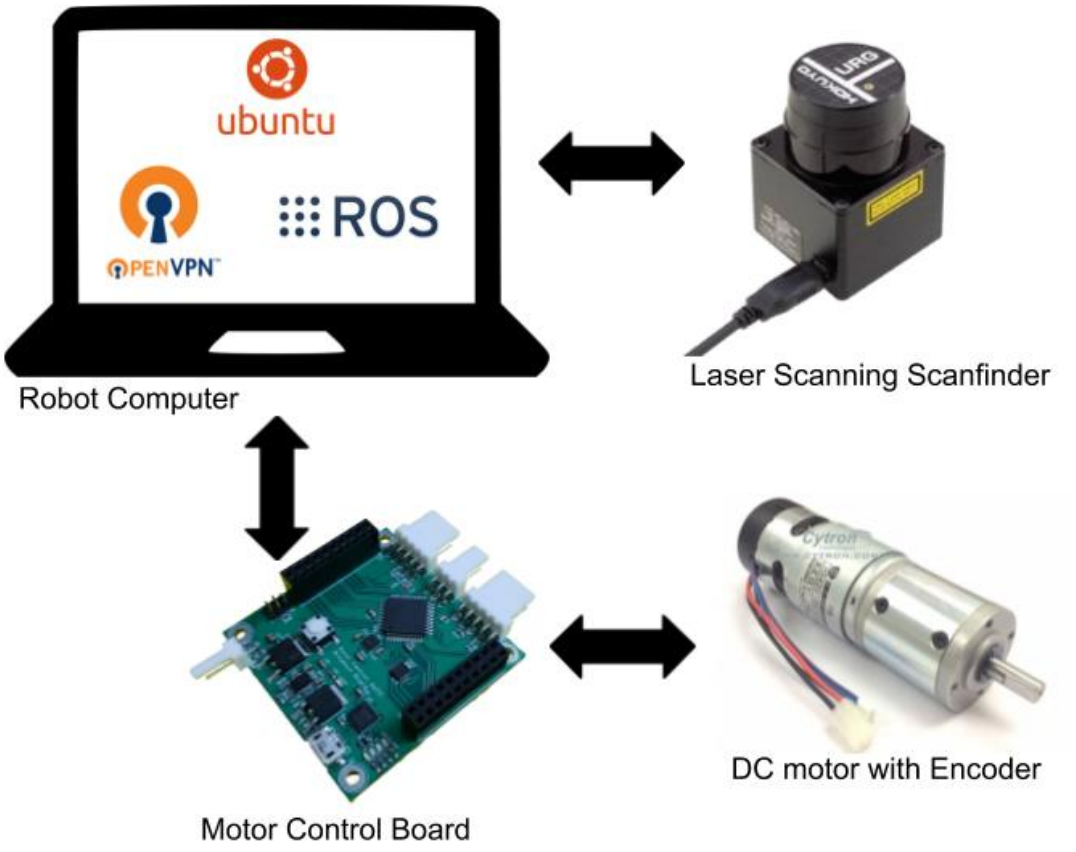

Figure 4.7: Components of mobile robot

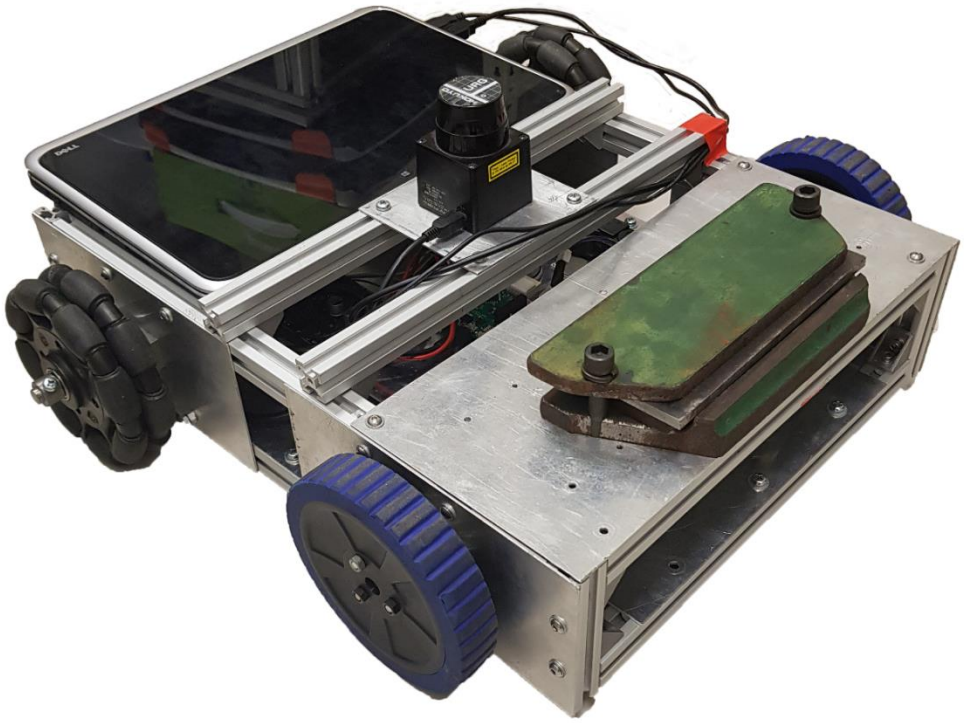

Figure 4.8: The resulting mobile robot

4.4.1 Robot Base

The robot base was built with aluminium profiles and plates as shown in Figure 4.9. Designing with aluminium profiles ease the binding process between bars and plates with the use of L-brackets and T-slot nuts, without the needs of drilling holes on the bars.

The robot driving method is differential drive, a two-wheeled drive system requiring independent motor for each wheel. In this driving method, the movement of robot is the result of difference in wheels motion. The design used two omnidirectional wheels as the free turning wheels to allow the back of mobile robot to move in all direction freely as driven by the front wheels.

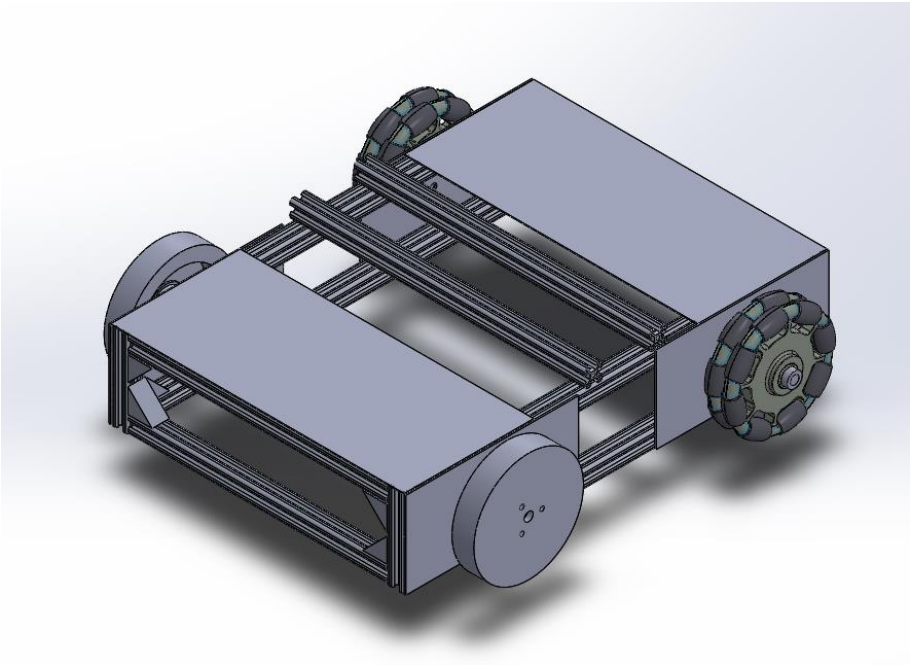

Figure 4.9: CAD drawing of mobile robot base

4.4.2 Motor Control Board

The motor control board has Microchip dsPIC33FJ128MC804 16-bit digital signal controller as its main controller for precision motor control. The controller consists of two quadrature encoder interface for incremental encoders to decode wheel position or rotation speed and 6-channel 16-bit motor control PWM for DC motor speed control (Microchip, 2012). There is an inertial measuring unit (IMU) at the centre of the control board providing angular velocity and acceleration measurement in different

direction for robot position. The control board acts as agent between robot hardware and computer running ROS, collecting and sending sensors and wheel encoder data to the computer via USB for further processing. In addition, the board converts velocity command from the computer into motor control signal.

4.4.3 Laser Scanning Rangefinder

Laser Scanning Rangefinder used in the mobile robot is Hokuyo URG-04LX-UG01 which has measuring range of 60 mm to 4095 mm with a tolerance of 1% when measuring between 1000 mm and 4095 mm and ±10mm tolerance when measuring between 60 mm and 1000 mm. The measuring angle is 240° with 0.36° angular resolution and it has 100 ms scanning time. It provides high accuracy and resolution laser scanning suited for autonomous robots. The compact form factor and light weight of the laser allowed easy integration into the robot base. Moreover, the laser uses near infrared light source at 785 nm rated at safety class 1 which is safe to be used in all conditions. The laser rangefinder requires input power of 5 V and 500 mA from USB, and it uses USB as its data interface as well (HOKUYO AUTOMATIC CO., LTD., 2014). In the mobile robot, Hokuyo URG-04LX-UG01 was integrated into the system by connecting to the robot computer via USB port.

4.4.4 Performance of mobile robot

Initially, the robot movement was not smooth due to overshooting of PID control signal and skidding of driving wheels. It was improved after PID tuning with Ziegler-Nicholas method and adding weight, the green metal plates as shown in Figure 4.8, onto the robot. The improved robot motion enabled accurate odometry which was able to drive the robot in a 1 meter square with drift contained under 5 cm. Accurate odometry is important to SLAM algorithm as it provide approximate robot position for localization.

4.5 SLAM simulation

For the SLAM simulation, internet service used by the local client was fiber optic Residential High-Speed Internet (UniFi) provided by Telekom Malaysia Berhad. The internet service had a maximum download rate of 23.42Mbps and maximum upload rate of 16.31Mbps. Laser data was published by rosbag, a tool to record and play back

pre-recorded data in ROS, after a connection was formed between Cartographer node and rosbag node through the laser scan data topic. Rviz was used to visualize the slam process in real time in the local machine. Figure 4.10 shows the resulting graph of the ROS setup used in the simulation.

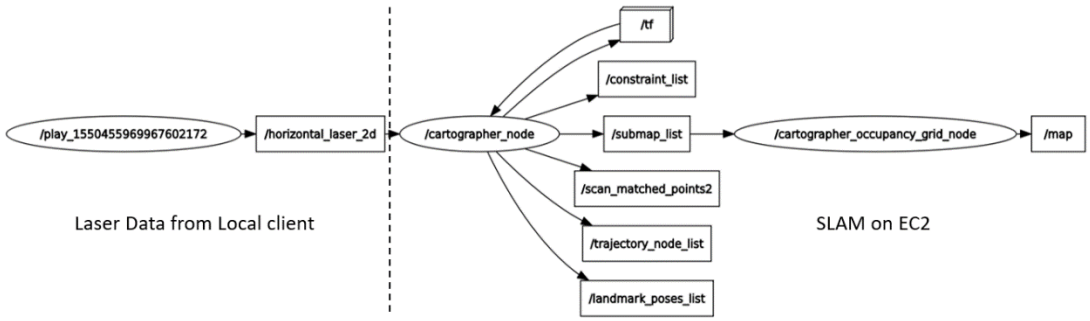

Figure 4.10: ROS computation graph formed in SLAM simulation

Table 4.1 shows the RTT obtained using the simulation setup described in the methodology. An Average RTT of 18.6540ms with standard deviation of 23.7870ms was obtained from the simulation. Figure 4.11 shows the RTT measured over a period of simulation. There are certain instances within the simulation where increase in RTT of packets can be observed. The highest delay recorded in the simulation is 666.5872ms. However, the sporadic increase in RTT throughout the simulation did not affect the capability of Cartographer in generating the map from recorded sensor data. Figure 4.12 shows the map generated by Cartographer at the end of the simulation.

Table 4.1: Round trip time for all packets travelling between local client and cloud server

	Round Trip Time (ms)
Average	18.6540
Median	14.8116
Standard Deviation	23.7870
Maximum	666.5872

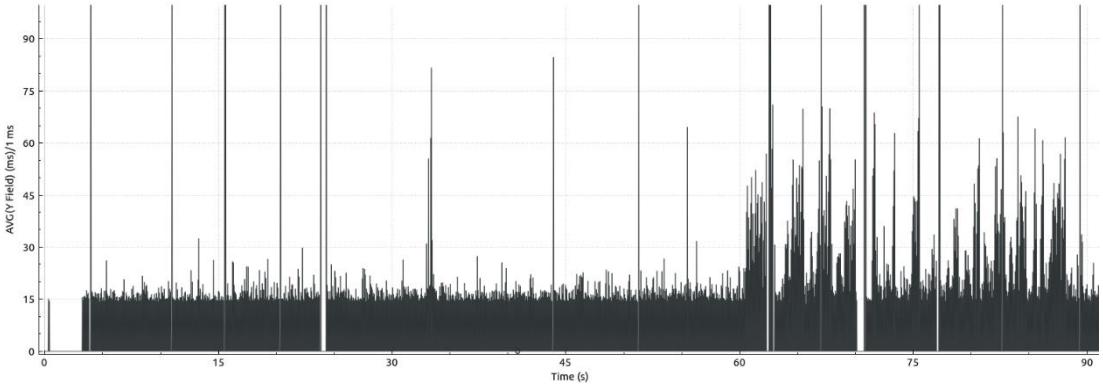

Figure 4.11: Round trip time of packets between local client in Malaysia and EC2 server running Cartographer in Singapore

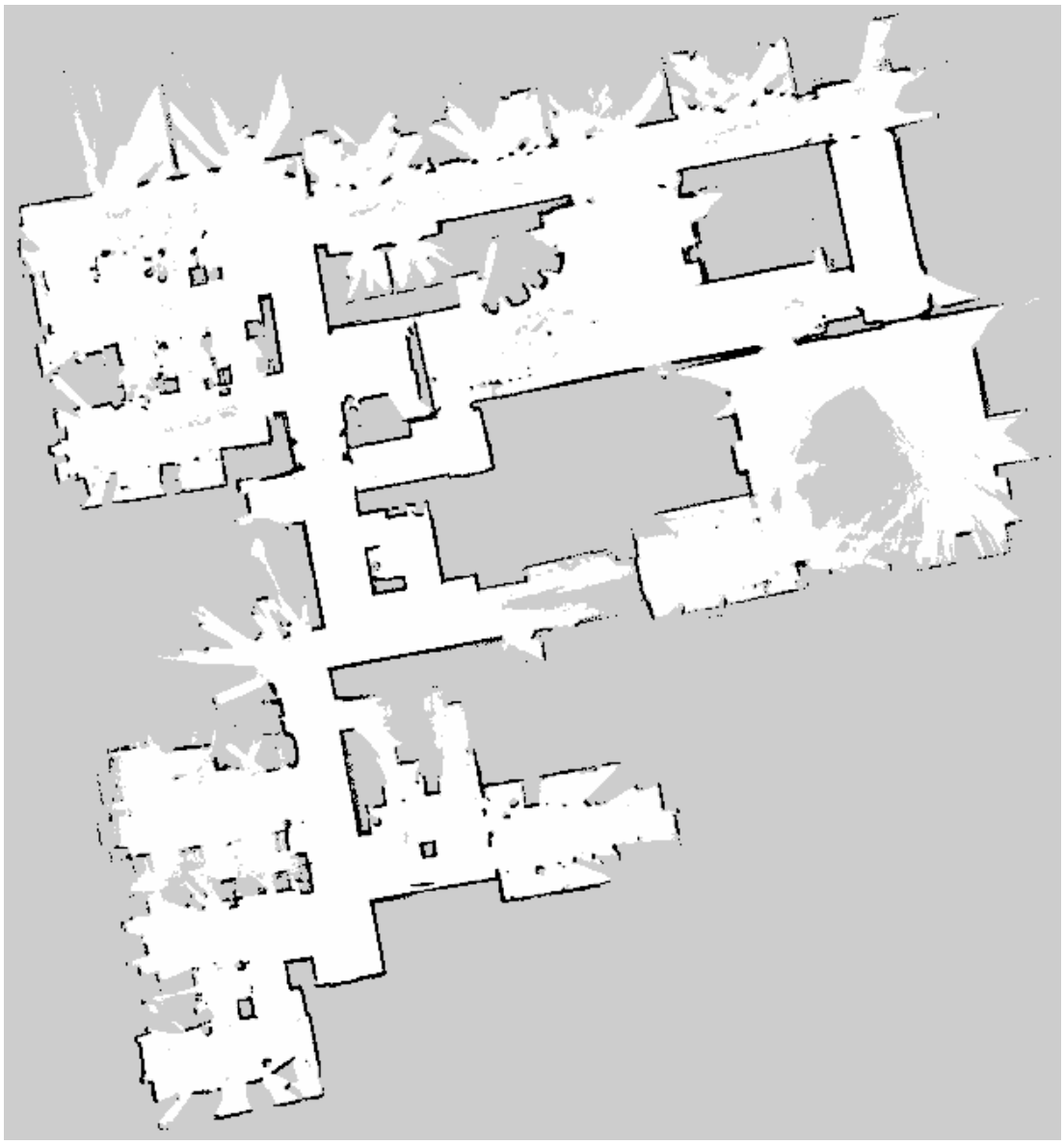

Figure 4.12: SLAM generated map from simulation with Cartographer dataset

4.6 SLAM on mobile robot

The mobile robot SLAM operation offloading was run on 4G network. The network provider was Umobile 4G LTE service with a maximum download rate of 4.23 Mbps and maximum upload rate of 15.9 Mbps with 25 ms latency during the period of testing. Table 4.2 shows the RTT of different data topic for testing on 4G network and Figure 4.13 shows the resulting computation graph.

Table 4.2: Round trip time for packets in mobile robot SLAM on 4G network

	Round trip time (ms)		
	Odometry	**Laser scan**	**Velocity control**
Average	63.0136	63.0371	107.6125
Median	54.2609	57.2985	101.8911
Standard Deviation	31.7453	29.2055	31.4774
Maximum	749.5426	671.9794	720.1007

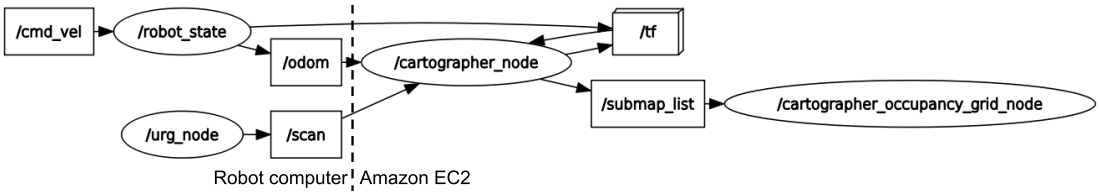

Figure 4.13: ROS computation graph formed in mobile robot SLAM

Mobile robot testing on 4G network had higher latencies, with average of 63.0136 ms, 63.0371 ms and 107.6125 ms for odometry, laser scan and velocity control data respectively. Velocity control packets had the highest latency, 71 % higher compared to odometry and laser scan because the packets travelled from remote computer to robot computer through cloud computer as VPN server while odometry and laser scan were just transmitted from robot computer to cloud computer.

Latency on velocity control data was slightly noticeable when controlling the robot motion, especially the spike of latency which could go up to 720.1007 ms due to unstable bandwidth of 4G network. However, the occurrence of high latency was not frequent, the overall response time was sufficient for SLAM operation. The average latencies were unnoticeable when controlling robot motion and had no impact on SLAM output quality, the resulting map is shown in Figure 4.14.

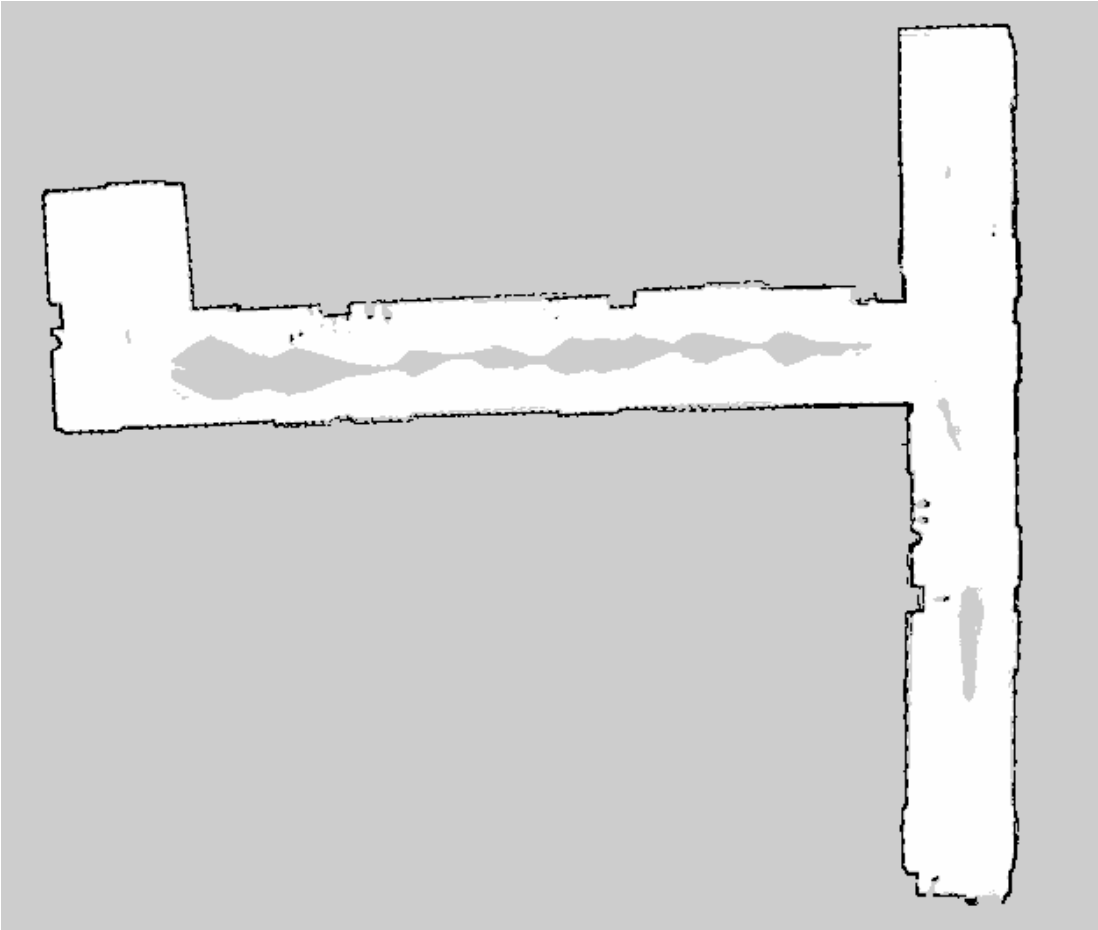

Figure 4.14: SLAM generated map from mobile robot (UTAR KB 3rd floor hallway)

4.7 Summary

Cloud implementation of ROS by putting all machines in a ROS system into a VPN is feasible, and RTT of ROS message packets between cloud computer and local computer can be captured using Wireshark for latency analysis. The ROS equipped mobile robot with wheels encoder odometry and laser range scanning performed as expected in methodology. Message latency results from SLAM algorithm offloading to Amazon EC2 instance ran in simulation and real world scenario through fiber optic internet and 4G network respectively shows that 2D SLAM offloading is feasible, with occasional minor delay in robot motion control signal when running in 4G network.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

Cloud implementation of ROS was configured by placing cloud computing infrastructure and local machines within a VPN. A differential-drive mobile robot with ROS equipped was built to be integrated into the setup. Then, ROS messaging latencies were captured and analysed using Wireshark to study the response time of the aforementioned setup.

In conclusion, it is feasible for a mobile robot to offload its computationally heavy process to a cloud server from the result obtained. A mobile robot that is capable of offloading part of its process to the cloud has numerous advantages. The robot would require less processing power, which can lead to a mobile robot with lower cost and higher battery efficiency. Algorithm with higher complexity can be implemented on the cloud easily when the need arises as cloud server can be scaled up accordingly.

5.2 Recommendations for future work

In this study, RTT was the only metric defined to investigate the feasibility of cloud implementation of ROS. To study the impact of cloud implementation of ROS in depth, more metrics should be considered, for example, data bandwidth, difference in computation time, and energy consumption of mobile robot.

Future study should look into the implementation of this setup for other important robot features such as 3D SLAM, object recognition, sound source localisation and voice recognition in mobile robot. In addition to direct offloading of ROS algorithms, the implementation can utilise optimised and well-developed algorithms in improving the robot capabilities, by using web services for speech and object recognition like Amazon Rekognition, Google Object Detection API and Amazon Transcribe.

REFERENCES

Agarwal, S. & Mierle, K., 2019. *Ceres Solver — A Large Scale Non-linear Optimization Library.* [Online] Available at: http://ceres-solver.org/ [Accessed 11 Jan 2019].

Angristan, S., 2018. *OpenVPN-install - GitHub.* [Online] Available at: https://github.com/Angristan/OpenVPN-install [Accessed 15 July 2018].

Arumugam, R. et al., 2010. DAvinCi: A cloud computing framework for service robots. *2010 IEEE International Conference on Robotics and Automation.*

AWS, 2018. *How to Launch a Linux Virtual Machine on the Cloud – AWS.* [Online] Available at: https://aws.amazon.com/getting-started/tutorials/launch-a-virtual-machine/?trk=gs_card [Accessed 22 June 2018].

Bhadani, R., 2018. *ROS/NetworkSetup - ROS Wiki.* [Online] Available at: http://wiki.ros.org/ROS/NetworkSetup [Accessed 15 July 2018].

Cartographer ROS, 2019. *Algorithm walkthrough for tuning — Cartographer ROS documentation.* [Online] Available at: https://google-cartographer-ros.readthedocs.io/en/latest/algo_walkthrough.html [Accessed 22 Jan 2019].

Cartographer, 2019. *Running Cartographer ROS on a demo bag.* [Online] Available at: https://google-cartographer-ros.readthedocs.io/en/latest/demos.html [Accessed 15 Jan 2019].

Coroiu, A. T. & Hinton, O., 2017. *A Platform for Indoor Localisation,Mapping, and Data Collection using anAutonomous Vehicle,* s.l.: Combain Mobile AB.

Foote, T., 2013. tf: The transform library. *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA),* pp. 1-6.

Hess, W., Kohler, D., Rapp, H. & Andor, D., 2016. Real-time loop closure in 2D LIDAR SLAM. *2016 IEEE International Conference on Robotics and Automation (ICRA).*

HOKUYO AUTOMATIC CO., LTD., 2014. *Scanning Rangefinder Distance Data Output/URG-04LX Product Details.* [Online] Available at: https://www.hokuyo-aut.jp/search/single.php?serial=165 [Accessed 23 January 2019].

International Federation of Robotics, 2017. *World Robotics 2017 edition,* s.l.: s.n.

Marguedas, 2018. *actionlib - ROS Wiki.* [Online] Available at: http://wiki.ros.org/actionlib [Accessed 30 August 2018].

Microchip, 2012. *16-bit Digital Signal Controllers (up to 128 KB Flash and 16K SRAM) with Motor Control PWM and Advanced Analog.* [Online] Available at: http://ww1.microchip.com/downloads/en/DeviceDoc/70291G.pdf [Accessed 12 Jan 2019].

Mohanarajah, G., 2015. *Rapyuta: A Cloud Robotics Framework.* [Online] Available at: http://rapyuta.org/welcome [Accessed 25 July 2018].

Mohanarajah, G., Hunziker, D., D'Andrea, R. & Waibel, M., 2015. Rapyuta: A Cloud Robotics Platform. *IEEE Transactions on Automation Science and Engineering,* 12(2), pp. 481-493.

Mösenlechner, L., 2012. *Introduction to ROS,* s.l.: Technische Universit¨at M¨unchen.

Nguyen Hoang Thuy, T. & Shydlouski, S., 2018. Situations in Construction of 3D Mapping for Slam. *MATEC Web of Conferences,* Volume 155.

Open Source Robotics Foundation, 2018. *Is ROS For Me?.* [Online] Available at: http://www.ros.org/is-ros-for-me/ [Accessed 30 August 2018].

Open Source Robotics Foundation, 2018. *ROS.org | Integration.* [Online] Available at: http://www.ros.org/integration/ [Accessed 30 August 2018].

rapyuta-robotics, 2015. *RCE.* [Online] Available at: https://github.com/rapyuta-robotics/rce [Accessed 15 July 2018].

ROBOTIS, 2019. *ROBOTIS e-Manual.* [Online] Available at: http://emanual.robotis.com/docs/en/platform/turtlebot3/slam/#run-slam-nodes [Accessed 15 Jan 2019].

ROS Wiki, 2018. *Ubuntu install of ROS Melodic.* [Online] Available at: http://wiki.ros.org/melodic/Installation/Ubuntu [Accessed 23 March 2018].

Toris, R. et al., 2015. Robot Web Tools: Efficient messaging for cloud robotics. *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).*

Waibel, M. et al., 2011. RoboEarth. *IEEE Robotics & Automation Magazine,* 18(2), pp. 69-82.

Wu, Y., 2018. *Master - ROS Wiki*. [Online] Available at: http://wiki.ros.org/Master [Accessed 30 August 2018].