Title of project placed here

Chongbin Ren

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

A dissertation presented in part fulfilment of the requirements of the Degree of Master of Science at The University of Glasgow

Date of submiss

Abstract

Autonomous robots are getting more and more popular in recent days. This makes many types of robot middleware software framework created and in use currently. ROS is a kind of very popular robotic operating middleware, with an active community of users. ROS can run on different multiple ROS masters. It means that users can use many independent robots, each with its operating system, and all robots can communicate with each other. Some kinds of hardware such as Raspberry Pi is becoming more and more prevalent in robotic research. ROS2 which is under heavy development is a new generation of ROS. Investigating ROS1 and ROS2 performance can be interesting in the same condition.

To investigate this research, I installed ROS1 and ROS2 on Raspberri Pi 3B+ and run it on Ubuntu Bionic (64-bit) operating system. Sending different frequency message in different machines by using WIFI network through a router.

A sequence of experiments was conducted investigating ROS1 and ROS2 communication performance by scaling from low message frequency to high frequency. Frequencies are ranging from 1Hz to 20000Hz sending simple string message. Key ndings are that ROS1 message latency can vary to higher latency than ROS2 in the same message frequency. That means ROS2 can perform better than ROS1 in transferring a simple string message with the same condition. These findings can be used on the Sunfounder Raspberry Pi Video car, indicating the results can use on real robot platforms.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: _____ Signature: _____

Acknowledgements

I would like to thank the following people for their help and support throughout the whole project.

My supervisor, Professor Phil Trinder and Natalia Chechina, for their invaluable support and advice at all stages of investigating my experiment and this dissertation. Of course, every experiment may encounter problems and failures even many upsets. However, they guided me and encouraged me in different ways and gave me suggestions.

Furthermore, thanks to many students who I met here and work together for the whole day and night. We study together for the examination and help each other to pass the exam. We share our experience in the experiment and give hands to each other.

Finally, I am extremely indebted to my family for supporting me and making it possible for me to complete this Masters degree. I thank my parents who support me to have a different trip and experience in a foreign country. Then, I will thank my dear girlfriend who accompanies me and encourages me at all time.

Contents

Chapter 1

Introduction

1.1 Context

New industrial, personal, enterprise and toy robots are being announced pretty much daily. There are indications that a rapid increase in the numbers of robots employed in the industry is already taking place. For example, a drone is a flying robot that can be remotely controlled or fly autonomously through software-controlled flight plans in their embedded systems, working in conjunction with onboard sensors and GPS[26]. The SunFounder Smart Video Car Kit for Raspberry Pi is composed of Raspberry Pi, DC-DC Step-down Voltage Module, USB camera, and Servo Controller[27]. It can be controlled by an Android phone to move or take a picture. The smart car is developed based on the open-source hardware Raspberry Pi and integrates the knowledge of mechanics, electronics, and computer, thus having profound educational significance.

ROS is a software framework which can allow users to write applications and operate robotic hardware (hence Robot Operating System). ROS as it has existed for several years and has many versions. Some are older releases with long term support, making them more stable. ROS2 is a new version robot operating system but it is under heavy development. The target of the ROS 2 project is to adjust to some advantages and improve disadvantages from ROS1[28].

Publishers are classes provided by the ROS library which are used to publish messages to a topic. Subscribers are classes used to receive messages from topics. Once a subscriber subscribes to a particular topic, the Master manages a connection between the publishing node and the subscribing node, so that the information is passed directly between the two processes.

Messages: ROS data type used when subscribing or publishing to a topic. Topics: Nodes can publish messages to a topic as well as subscribe to a topic to receive messages. Master: Name service for ROS (i.e. helps nodes find each other)

The aim of the research is to measure the communication performance between ROS1 and ROS2 two different robot system generation. Through sending and receiving message between multiple machines and test message latency and throughput to compare them, so that future researchers can have an understanding of two different generation ROS system communication performance.

1.2 Contribution

This project makes several key contributions:

In this paper, I ~~resolve some problems which may occur in~~ ROS installation. ~~Trying to install ROS in the different operating system on raspberry pi~~. Due to more support on Ubuntu, ROS can be more convenient installed and run on Ubuntu operating system than others. This is covered in Chapter 3 and also including more detail steps.

I investigated the different ROS1 version in different Ubuntu operating system. Founding is that communication in ROS1 Melodic on Ubuntu Bionic(64-bit) performs much better than ROS1 Kinetic on Ubuntu Xenial(32-bit). This is covered in Experiment 1.

Furthermore, I also researched the performance between ROS1 and ROS2 with the same hardware and operating system. Running the message sending experiments and analyse data by using python. This is covered in Experiment 2.

Chapter 2

Background

2.1 Robotics Overview

Robotics is a branch of engineering that involves the conception, design, manufacture, and operation of robots[1]. Robots are machines that can be used to do jobs. Some robots can do work by themselves. Other robots must always have a person telling them what to do. Todays robotic systems generally formed by many small autonomous systems which work together[2].

This chapter provides the background knowledge required to get more understanding on the results in this paper. Section 2.1 discusses the robotics overview and hardware in this experiment. Section 2.2 describes the operating system and robotic middleware when constructing a robotic system. Next, section 2.3 introduces network topology and communication protocol used by the different robotic operating system.

2.1.1 Multi-Robot Systems

Given the rate of technological advancements over the past decade, the adoption of robotics has become increasingly widespread. In most domains, the performance of multiple robots outperforms a single robot, both concerning cost, efcacy and domain potential [3]. This has led to multi-robot systems becoming a key area of research within the eld of robotics in general [4].

Many intelligent agents can work together to solve a complete task that anyone system can not finish it alone. They form one multi-robot system. The multi-robot is distinct from a multi-agent system, as each robot can move in a multi-robot system[5]. In recent years, it can be more possible and necessary to make mobile robotics by using advanced battery and wireless communication technologies[6]

2.1.2 Raspberry Pi

The Raspberry Pi is a small single-board computer which is developed by the Raspberry Pi Foundation in the UK. The aim for it is to improve the teaching or testing at computer area in schools

and organisations [7]

The Raspberry Pi is a small-sized like a credit card, single-board computer. Its capable of doing everything that a desktop computer can do, from browsing the internet and installing applications, even playing games. It runs Linux and other available operating systems from a micro SD card. With the addition of a keyboard, mouse and micro USB power supply it can be connected to a screen monitor and used as a fully functioning desktop computer. In short, the Raspberry Pi is a small computer with relatively limited memory and performance, but it contains all needed functions.

The first Raspberry Pi was released in 2012, since then it has been used in a vast range of projects due to its low cost, portability, programmability and both wired and wireless connectivity[8]. Several generations of Raspberry Pis have been released.

Raspberry Pi 3B+ is used in this paper. The real structure picture Figure 2.1.

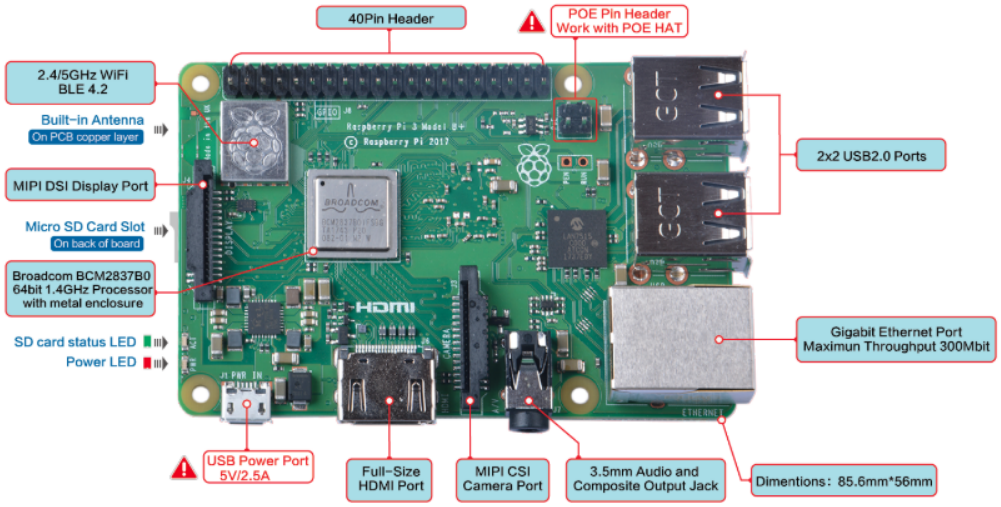

Figure 2.1: The structure of a Raspberry Pi 3B+

2.1.3 SunFounder Raspberry Pi Smart Video Car

SunFounder is a technology company focused on Raspberry Pi and Arduino open-source community development. The Pi Car-S Robot is a smart car which works based on a Raspberry Pi as shown in Figure 2.2.

The raspberry pi car comes with three sensor modules including ultrasonic obstacle avoidance, light follower, and line follower. The car robot has a durable and shatterproof plate and new style Servo, the MAX torque of the clutch gear digital servo is up to 1.6KG[9]. It rotates from 0-180 degrees.

Python code is provided for the car, and users can also program and debug it with Dragit, a Snap-based graphical interface, by just simply dragging and dropping the code blocks for more complex functions. A user can learn to programme conveniently on how to control the car.

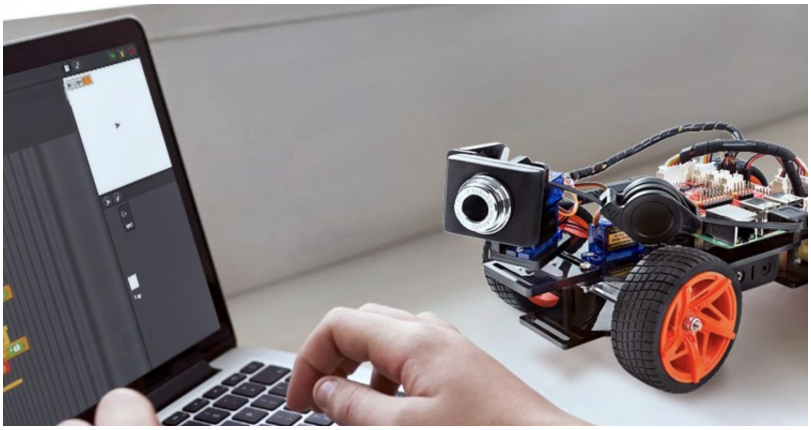

Figure 2.2: SunFounder Raspberry Pi Smart Video Car

2.2 Robot Software and Operating System

The following sections introduce the Robot software and operating system used in this project. The version of ROS and ROS2 is under development and changes frequently these years.

2.2.1 ROS

Robot Operating System (ROS) is robotics middleware which is a collection of software frameworks for robot software development.ROS is a set of software libraries which is used for people to build some robotic applications rather than a kind of operating system. It has drivers with powerful developer tools and its all open source[11]. ROS is a software framework meant to allow people to write applications which operate robotic hardware. As its most fundamental level, it is an abstraction layer - offering hardware abstraction. Generally, a robotics middleware would provide a common interface, in this way, hardware can produce different data but the results can be distributed in a consistent manner[10].

Robot Operating System is mainly composed of 2 things[12]:
1. A core (middleware) with communication tools.
2. A set of plug and play libraries.

2.2.2 ROS 2

ROS2 is a new version of ROS and that is under heavy development now. ROS2 is the next generation of robot operating system and under development which will fully replace ROS in the near future. The aim for ROS2 project is to extend the advantages and improve the disadvantages from the ROS1 generation[13].

A further reason to build ROS2 is to take advantage of the chance to improve users APIs. Thats great from the view of stable performance, but it also implies that we still live with API decisions that were made in the previous, which is not the best[14].

There some obvious differences between ROS and ROS2:

1. ROS1 is using Python 2, but ROS2 requires at least Python version 3

2. Catkin make is gone: Catkin has been replaced by 'colcon'. This new building system is, in essence, a universal building system for the ROS ecosystem.

3. ROS2 uses 'rclpy' as Python client library which replaces 'roscpp' in ROS.

2.2.3 Ubuntu

Ubuntu is a complete Linux operating system, which is an open-source Debian-based Linux distribution. Ubuntu is now the most used Linux operating system, both for desktops and servers.

Ubuntu is built on Debian's architecture and infrastructure, and comprises Linux server, desktop and discontinued phone and tablet operating system versions.[15] Ubuntu releases updated versions predictably every six months, and each release receives free support for nine months[16]. Downloading, installing, and using Ubuntu Linux doesnt cost a penny. Installing Ubuntu does not need high-end system requirements. So it is an ideal operating system for Raspberry Pi, as the Raspberry Pi has limited memory CPU performance.

Ubuntu has many kinds of the desktop version. Ubuntu MATE extends the Ubuntu base OS and adds his own desktop. The MATE Desktop is a desktop environment and includes some basic softwares like a text editor, calculator, browser, image viewer, and terminal[17]. And Ubuntu Mate offers a specific downloading installing package for Raspberry Pi. This can be downloaded from the official site.

Furthermore, Ubuntu is also the recommended operating system in ROS. ROS offers more support for a few Ubuntu version. There is another operating system called Raspbian which is widely installed on Raspberry Pi. However, Raspbian is not well supported by ROS and it may occur many problems when installing ROS in Raspbian. Therefore, ubuntu is used in this experiment for Raspberry Pi as an operating system.

2.3 Communication on ROS vs ROS2

2.3.1 Star Network Topology

A star topology is a topology for a Local Area Network (LAN) in which all nodes are individually connected to a central connection point, like a hub or a switch. A star takes more cable than e.g. a bus, but the benefit is that if a cable fails, only one node will be brought down[18].

All traffic emanates from the hub of the star. The central site is in control of all the nodes attached to it. The central hub is usually a fast, self contained computer and is responsible for routing all traffic to other nodes. The main advantages of a star network is that one malfunctioning node does not affect the rest of the network[19]. However this type of network can be prone to bottleneck and failure problems at the central site.

It is the most common network topology in real life, as it is easy to install, scalable and easy to troubleshoot. At the centre of the network sits the multifunctional wireless router providing the functionality of a wireless access point (WAP), a modem, a switch and a router.

Star network topology is also used in this experiment for testing message latency communicating in different devices with a center router.

The star network topology graph is shown in Figure 2.3

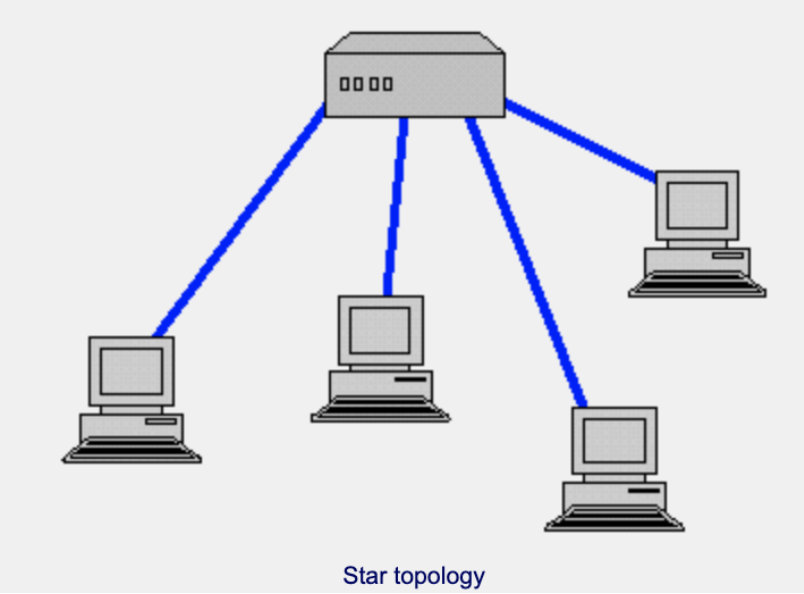

Figure 2.3: Star topology graph

2.3.2 TCP and UDP in ROS

ROS currently usees TCP and UDP as its transport protocol. The TCP/IP-based transport is known as TCPROS and streams message data through persistent TCP/IP connections. TCPROS is the default transport which is used in ROS. The UDP-based transport, which is known as UDPROS and is currently only supported in roscpp, which is C plus plus used in ROS. UDPROS is a low-latency, but lossy transport, so is best suited for tasks like remote communication. ROS nodes negotiate transport at runtime. For example, if a node prefers UDPROS transport but the other Node does not support it, it can fallback to use TCPROS[20].

TCPROS is a transport layer for ROS Messages and Services. It uses standard TCP/IP sockets for transporting message data. Inbound connections are received via a TCP Server Socket with a header containing message data type and routing information[21].

Therefore, in the experiment, ROS use TCPROS to communicate between different machines by default. The TCPROS communication structure is Figure 2.4.

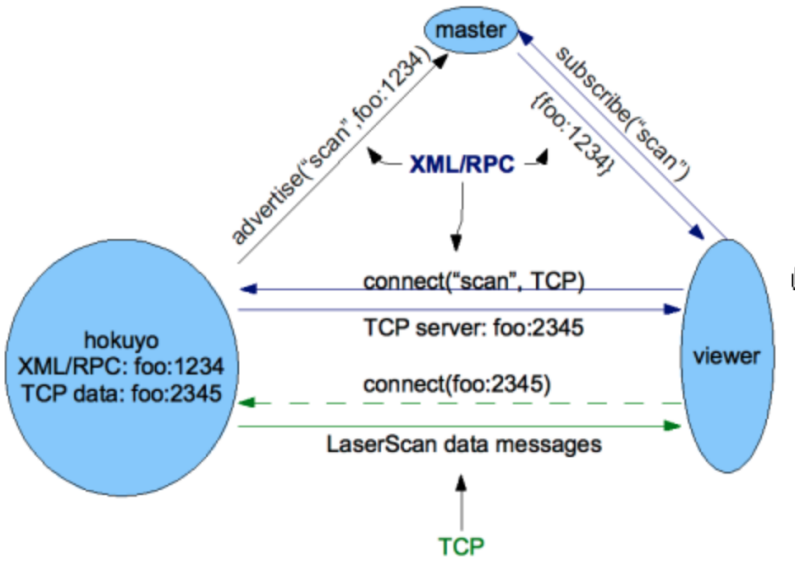

Figure 2.4: TCPROS communication structure

2.3.3 Data Distribution Service (DDS) on ROS2

The Data Distribution Service (DDS) is a middleware protocol and API standard for data connectivity from the Object Management Group. It integrates the components of a system together, providing low-latency data connectivity, extreme reliability[22].

The advantage of using DDS is that the code is easy to maintain and the behavior of the middleware have been extract to document. In addition to system-level documentation, DDS also has recommended use cases and a software API.[23].

ROS2 uses DDS as its message transport service. DDS also provides a publish-subscribe transport which is very similar to ROS transport. It allows two DDS programs to send message without the need for starting a tool like ROS master node[24]. This means the system can be more flexible and convenient.

The default implementation of DDS is over UDP, DDS implementations use (UDP) broadcast traffic for peer discovery. By default, when sending message, the ROS2 communication will broadcast to the local subnet (ie: all IPs in the same range).

Chapter 3

Installation and configuration

3.1 Ubuntu and ROS system installation

3.1.1 Ubuntu installation

1. Download the latest version image of Ubuntu Bionic (64-bits) for Raspberry Pi from official website.

2. Write the image to Raspberry Pi 3B+ by using SD card.

3. Run it with screen and configure all necessary dependency and install ”openssh” server for remote control.

Issues and Solution

1. Other operating system like Raspbian 10(buster) is not well supported by ROS system. There are some outdated dependencies which is needed in installing ROS. Therefore, there are many bugs and manually installation steps to solve when user want to install ROS in Raspbian operating system. Then the Ubuntu is recommend operating system as official website said.

2. Find the 64-bits version operating system because ROS2 only can run on 64-bits operating system.

3.1.2 ROS1 Installation

1. Follow the step details at first appendix A1 ROS1.

2. Configure Ubuntu repositories to allow ”restricted,” ”universe,” and ”multiverse.”

3. After installation, use *roscore* to test whether ROS system exist.

3.1.3 ROS2 Installation

1. Installing ROS2 via Debian Packages.

2. Follow the step details at first appendix A2 ROS2.

3. Using *ros2 run XXpackage XXfile* to test whether ROS2 can run successfully.

3.2 Router network configuration

1. Each Raspberry Pi should connect to the same LAN.

2. Assign static IP address to each Raspberry Pi for ssh remote control and management.

3. Using one computer to join this LAN for controlling the all Raspberry Pi.

4. Remote control other Raspberry Pi by ssh using *ssh hostname@ip address* with password.

3.3 Communication between different ROS hosts

Change the host file

Each host configuration file can be modify by *sudo vim /etc/hosts*. Add ip address and hostname in */etc/hosts* file for Raspberry Pi. Each ip address respond to one hostname which you give. Example hosts file is in the Appendix.

Running Talker and Listener

Enter ROS system in every Ubuntu system and ssh to com1. Then ping com1 and com2. If ping successfully meaning the network is fine, otherwise check the router and network problem.

The host name and ip address is respond to each Raspberry Pi, but the Master URI should be identical. Create a *.sh* file to export environment paths. See the appendix

Configure the file and execute this file in every Raspberry Pi host. Then run the talker and listener python file seperately in different hosts to check if listener can receive messages from talker.

Issues and Solution

Make sure setting the ROS hostname and IP address correctly and must explicitly set it. Otherwise the listener cannot receive talker because the host cannot find it in the network. The Talker will send messages continuously but listener can not receive these messages.

Check the hostname and ip address setting can use:

```
echo $ROS_HOSTNAME
echo $ROS_IP
```

Chapter 4

Experiment and Evaluation on ROS1 and ROS2

4.1 Experiment 1: Investigating different ROS1 version message latency with previous experiment

Objective
This experiment is aiming to investigate the message sending performance in different ROS version with different OS platform by using Wi-Fi network connection through a router. This experiment was executed by Issac in the previous and I execute this experiment in different software and hardware and compare the results to the previous one[25].

Rationale
In order to figure out and better understand the performance characteristics of ROS communication channels, this experiment use two ROS hosts to commnunicate with each other and get the message latency results in different message frequency. I reproduced Issac's experiment to see what is the difference between my experiment and the previous one then analyze the reason to cause these differences.

Procedure
1. Using 'roscore' to start the ROS master node.

2. Start one publisher node on one host and send different frequency message to a echoer node on another host by using Wifi network.

3. Run the python code which send timestamped string messages from the publisher node to the echoer node. Then the echoer node receive message and send it back to publisher node.

4. The master node will receive the message and then record message id and time in sending and receving into a file.

5. Message frequency is from 200 to 2000Hz (200,400,600...2000)which is same with the previous experiment.

6. Every frequency will be recorded 3 times with a range of message frequencies to obtain averaged results for each message frequency.

7. Running this experiment in different ROS1 version and operating system.

Hardware Configuration
4 Raspberry Pi 3B+ (1.4GHz 64-bit quad-core ARM Cortex-A53 CPU , 1GB RAM LPDDR2)
TP-Link 150 Mbps Router

Software Configuration
Ubuntu 18.04 Bionic (64-bit)
ROS Melodic (Latest, Released May, 2018)

Ubuntu 16.04 Xenial (32-bit)
ROS Kinetic Kame (Released May, 2016)

ROS Melodic is installed on Ubuntu 18 Bionic; ROS Kinetic is installed on Ubuntu 16 Xenial.

Hypothesis
1. The message latency should be like the previous experiment results. Although I use different hardware and software, but the message latency should not have very obvious difference with the previous experiment.

2. Different message frequency may perform different latency. For example, lower message frequency can have lower message latency.

Results
The results reported by Jordan[25] for Experiment 1 are reproduced as Figure 4.1. This experiment plots six different frequency instead of Jordan 10 frequency because it may make the graph more clear to check out. Figure 4.2 is message latency from ROS kinetic on Ubuntu Xenial and Figure 4.3 is ROS Melodic on Ubuntu Bionic.

1. The message latency from the previous experiment by Jordan[25] is all almost under 20ms lower latency like 200Hz and 400Hz are very low latency and perform better than higher frequency like 1800Hz and 2000Hz, as Figure 4.1.

2. In this experiment, message latency is high and vary from a large scope in ROS kinetic on Ubuntu Xenial 32bits, as Figure 4.2. Message latency can be up to hundreds of millisecond which is ten times than previous experiment message latency.

3. Then in ROS Melodic on Ubuntu Bionic 64bits, the message latency is low and the result is very like the previous Issac's one. In lower frequency 200Hz and 400Hz show under 10ms latency and higher message frequency, latency can be varied at 15-20Hz as Figure 4.3.

Conclusion
Firstly, the ROS version and operating system version can have an influence on the communication performance between different hosts. In my experiment, ROS kinetic on Ubuntu 16 message latency can vary at a large scope that means it is unstable when changing frequency. It performs worse than using ROS Melodic on Ubuntu 18. There is a 400 percent message latency increase in 1200Hz comparing with 400Hz latency.

Secondly, in my experiment ROS melodic Ubuntu 18 performs better when sending a message than

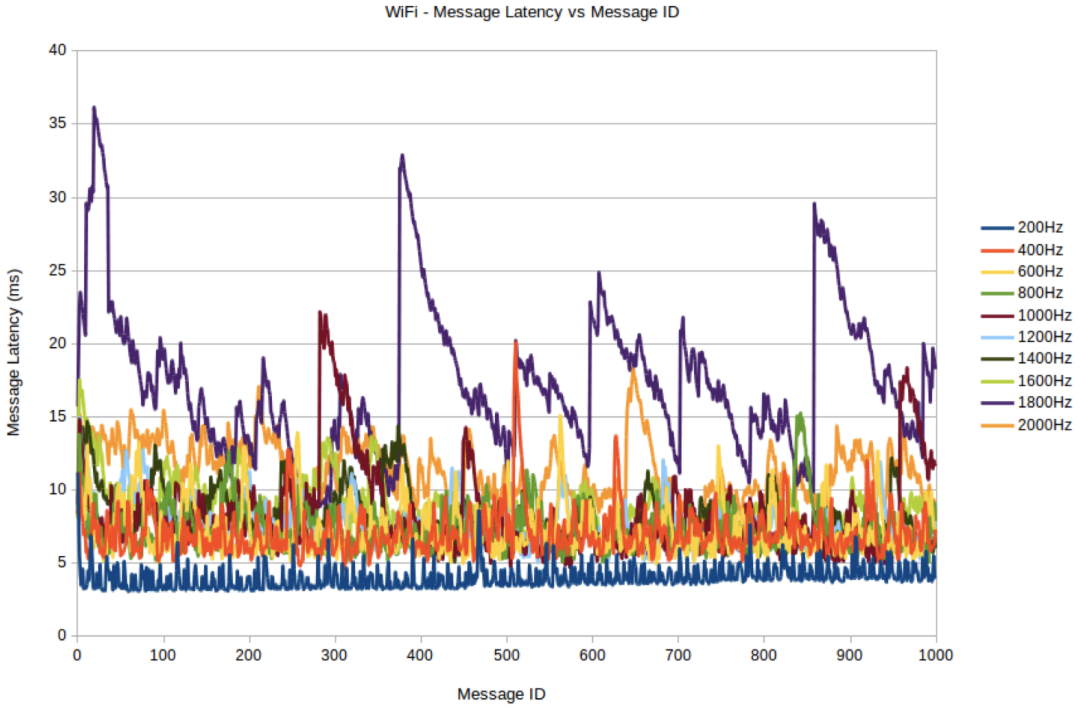

Figure 4.1: Message latency for all frequencies from Jordan[25]

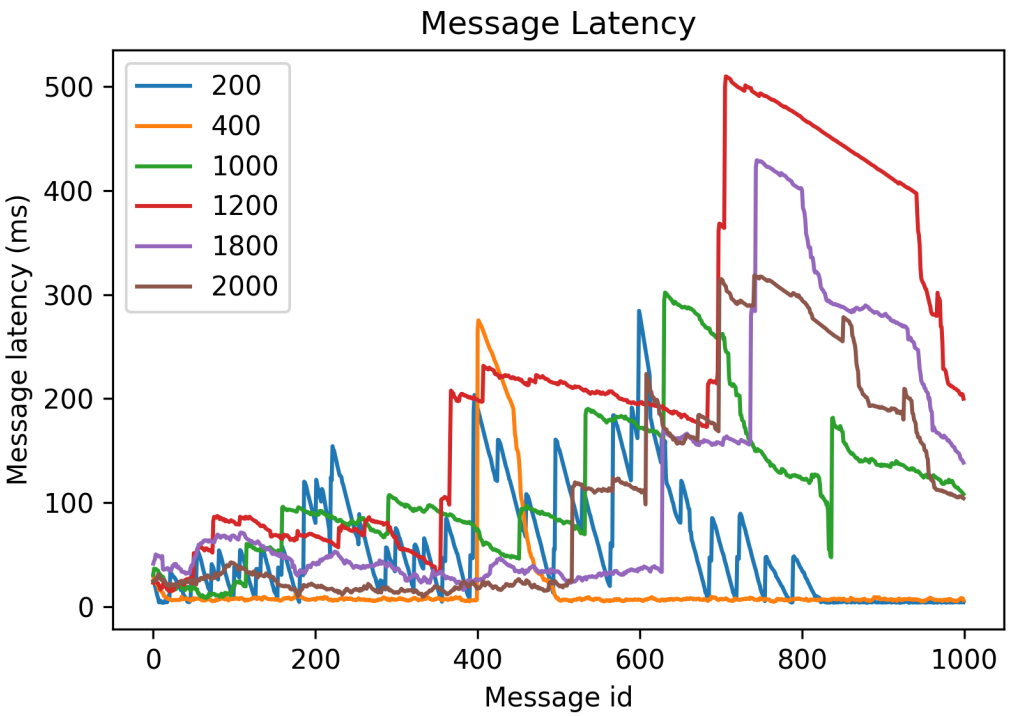

Figure 4.2: Message latency for ROS kinetic on Ubuntu Xenial 32-bit

previous experiment ROS Kinetic on Rasbpian. Because there is no obvious spike on ROS melodic

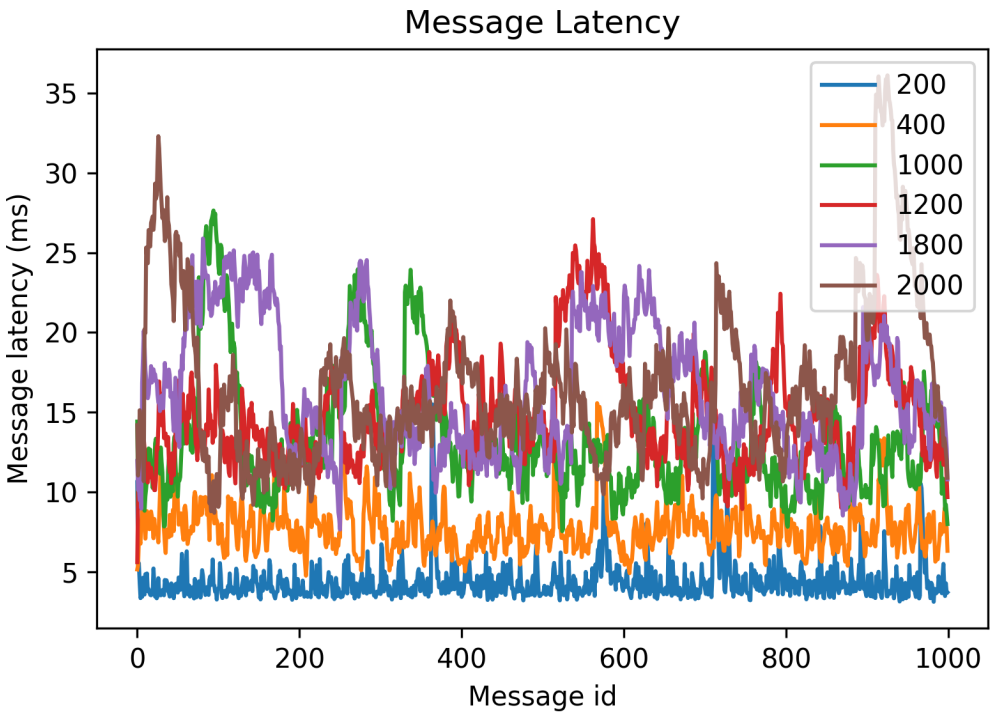

Figure 4.3: Message latency for ROS Melodic on Ubuntu Bionic 64-bit

Ubuntu 18. However, the router could also be a factor that has slightly impact on message latency.

4.2 Experiment 2: ROS1 and ROS2 message latency analysing

Objective
This experiment aims to analyse message latency performance in communication between ROS1 and ROS2 version with all other same condition. Then collect different message frequency data and analyse data to measure communication performance.

Rationale
ROS2 has more improvement comparing with ROS1 and it use different api and protocol when sending message. So in this experiment, I am going to test the message latency in different frequency and anaylysing what happend in different ROS version.

Procedure
1. There is no master node on ROS2 so you can directly start one node on one Raspberry Pi and send different frequency message to another host through wifi router.

2. Running python code which sends string messages from the sender host to echoer host with timestamps. Then the echoer host sends the message back to sender host.

3. The sender node will receive the message and then record message-id and time elapsed in communication into a file.

4. Message frequency is from low frequency (1Hz and 10Hz), middle frequency(100 and 200Hz) and high frequency (1000 and 2000Hz). Every frequency will be run 3 times and get a mean time result.

5. Using "ros2 run" in ROS2 run python3 code which also sends and receive the message and then record the results data.

6. Record all result data and plot by using matplotlib python in Jupyter notebook.

Hardware Configuration
4 Raspberry Pi 3B+ (1.4GHz 64-bit quad-core ARM Cortex-A53 CPU , 1GB RAM LPDDR2)
D-link Wireless AC1200 Dual Band Gigabit Router

Software Configuration
ROS1 Melodic (Latest, Released May, 2018) on Ubuntu 18.04 Bionic (64-bit)
ROS2 Crystal Clemmys (Released December 2018) on Ubuntu 18.04 Bionic (64-bit)

~~**Hypothesis**~~
~~The hypothesis in this experiment can be that ROS2 communication performs better than ROS1 because it is new ROS version and improvements in communication between different ROS version.~~

Results
1. From the experiment, overall, in Figure 4.4 and 4.5, ROS1 is less stable than ROS2 from low frequency to high frequency. All message latency is under 25ms in ROS1 comparing under 10ms in ROS2.

2. Figure 4.6 and 4.7 shows that at low frequencies like 1Hz and 10Hz. ROS1 latency can vary

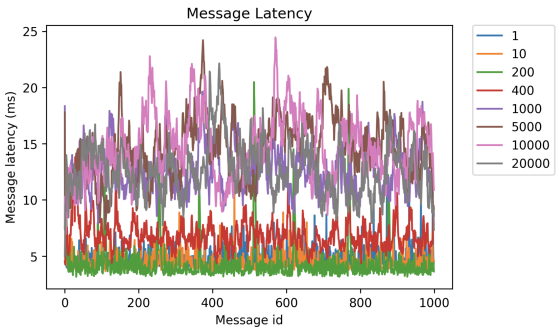

Figure 4.4: ROS1 all frequency

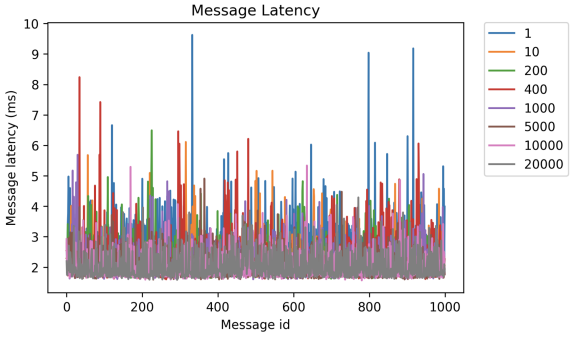

Figure 4.5: ROS2 all frequency

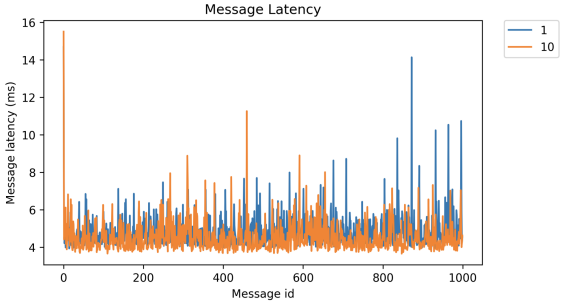

Figure 4.6: ROS1 1Hz and 10Hz

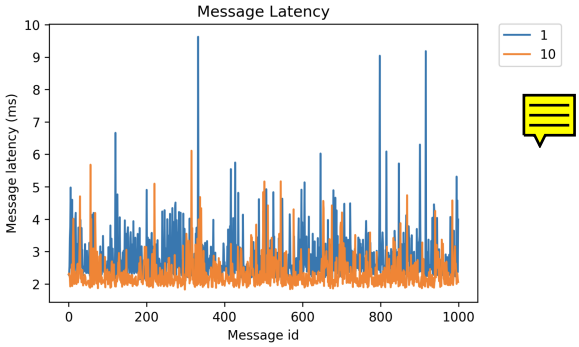

Figure 4.7: ROS2 1Hz and 10Hz

between 4ms to 15ms. However, in ROS2, message latency can vary from 2ms to 10ms.

3. Then in middle frequency, in Figure 4.8 and 4.9, from 100Hz to 1000Hz, ROS1 message is increasing with higher message frequency. However, ROS2 shows a stable and low message latency under 8ms.

4. In High frequency, in Figure 4.10 and 4.11, 1000Hz to 20000Hz, message latency can be unstable but looks the same vary from 5ms to 20ms in ROS1, however, the message latency can be low in ROS2 high-frequency which is under 5ms.

Conclusion

In order to acquire a detailed performance characteristic of ROS1 and ROS2, this experiment uses Raspberry Pi 3B+ which installed Ubuntu Bionic(64-bit) operating system. Then communicating by the same router with ROS1 and ROS2 separately. This can guarantee the same variable and validity of this experiment.

From the data analyse results, in low message frequency, for example, in 1Hz, ROS1 has around 66 per cent increase comparing to ROS2 in message latency. In middle frequency, for instance, in 500Hz message latency is around 8ms in ROS1 and 3ms in ROS2. which is around 250 per cent increase. Therefore, we can see there is a high per cent latency increase from ROS1 to ROS2.

To conclude, ROS2 can show lower message frequency than ROS1 with the same hardware and operating system. ROS2 show around the low latency in all message latency. However, ROS1 can vary from low frequency to high frequency. Due to using DDS in ROS2, a more advanced communication protocol so the message communication performance is better than ROS1.

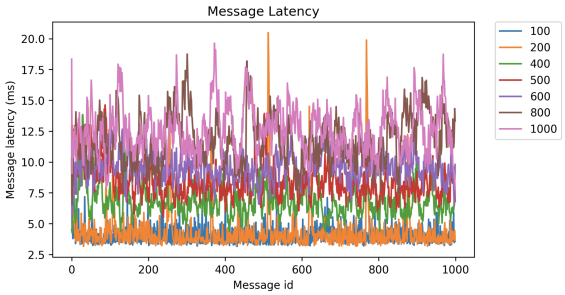

Figure 4.8: ROS1 100Hz and 200Hz

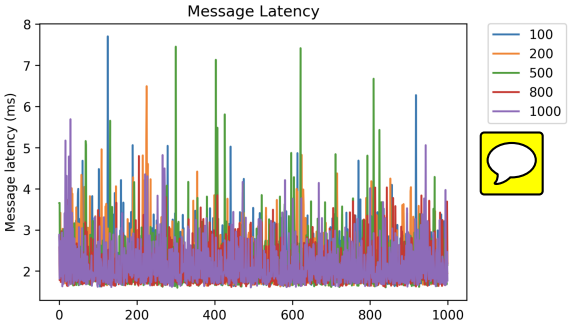

Figure 4.9: ROS2 100Hz and 200Hz

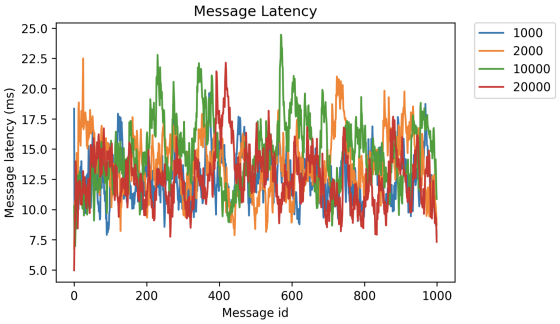

Figure 4.10: ROS1 1000Hz and 2000Hz

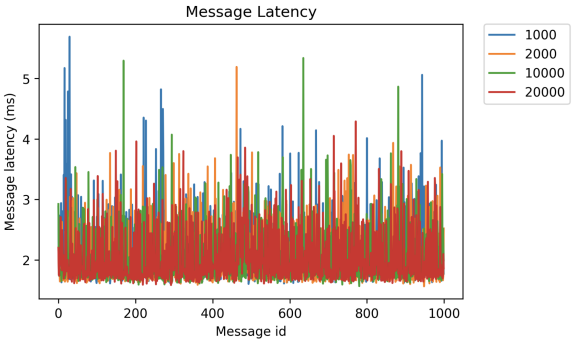

Figure 4.11: ROS2 1000Hz and 2000Hz

Chapter 5

Conclusion

5.1 Summary

This project began with solving a set of installation problems on installing ROS1 and ROS2 on the different operating system. Then got the easy and successful way to install ROS1 and ROS2 on Raspberry Pi 3 B+. The installation and configuration steps are in Chapter 3.

The next phase of the project (Experiment 1) investigated the message latency on ROS1 in different operating system platform. The first one is previous Jordan[25] experiment which used ROS Kinetic on Rapbian and my experiment includes ROS Kinetic on Ubuntu 16 and ROS Melodic on Ubuntu 18. By the result, it shows ROS Melodic on Ubuntu 18 performs better than ROS Kinetic on Raspbian and Ubuntu 16.

Finally, research was conducted in Experiment 2 into the performance on ROS1 and ROS2, which is a more interesting experiment. Through running the message latency experiment in two different ROS1 and ROS2 version, the result shows that ROS2 has lower message latency comparing to ROS1. ROS2 uses more advanced technology and protocol to communicate between hosts' nodes. The throughput can also show the performance of network communication, however, the throughput between two hosts was not conducted due to time constraints in the project. Above contributions are valuable for other people who will investigate more things on ROS.

5.2 Future work

There is some more interesting area that not investigate in my experiment due to the limited time:

1. Investigating sensor data communication between ROS1 and ROS2. In my experiment, I send a simple string message in a different frequency. However, the real sensor data message sending experiment is not conducted in this project due to time.

2. Investigating the effect of nodes throughput in different ROS version. When discussing network throughput, the measurement is typically taken per unit time between two machines, and repre-

sented as Bits per second (bps), (Kbps), (Mbps), and so on. Throughput is an important index for network performance. There will be a test of throughput performance in future work.

3. Conducting CPU clock speed impact in ROS1 and ROS2. There is a hypothesis that different ROS version can use a different percentage of CPU time. Therefore, an experiment can be conducted to investigate the CPU clock speed on different ROS version.

Appendix A

ROS installation steps

A.1 ROS1 Melodic installation commond steps on Ubuntu

```
# Install system
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc)
main" > /etc/apt/sources.list.d/ros-latest.list'
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key
C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
sudo apt update
sudo apt install ros-melodic-ros-base
# Install tools and dependencies
sudo rosdep init
rosdep update
echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
source ~/.bashrc
sudo apt install python-rosinstall python-rosinstall-generator python-wstool
build-essential
# Start ROS service
roscore
# create the catkin folder
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/
catkin_make
source devel/setup.bash
```

Figure A.1: ROS1 installation steps

A.2 ROS2 installation commond steps on Ubuntu

```
# Install
sudo locale-gen en_US en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LANG=en_US.UTF-8
sudo apt update && sudo apt install curl gnupg2 lsb-release
curl -s
https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
install
export CHOOSE_ROS_DISTRO=crystal
sudo apt update
sudo apt install ros-$CHOOSE_ROS_DISTRO-ros-base

# Environment setup
sudo apt install python3-argcomplete
source /opt/ros/$CHOOSE_ROS_DISTRO/setup.bash
echo "source /opt/ros/$CHOOSE_ROS_DISTRO/setup.bash" >> ~/.bashrc
sudo apt install python3-colcon-common-extensions

# create workspace
mkdir -p ~/ros2_example_ws/src
cd ~/ros2_example_ws


# /etc/hosts file

ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

192.168.1.11 com1
192.168.1.12 com2

# config.sh file

# /bin/sh
export ROS_HOSTNAME=com1
export ROS_IP=192.168.1.11
export ROS_MASTER_URI=http://com1:11311/
```

Figure A.2: ROS2 installation steps

Appendix B

Screenshot displaying

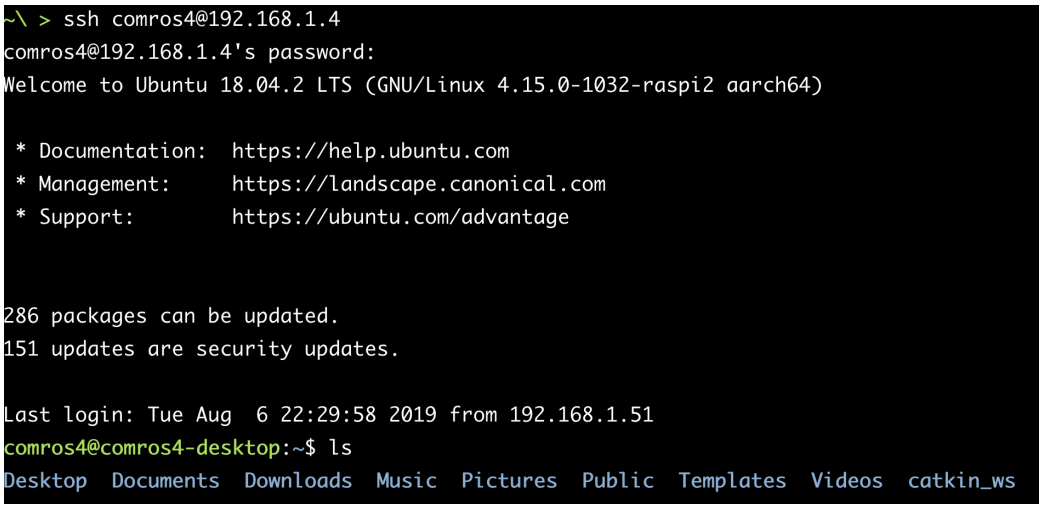

Figure B.1: ROS installation successfully screenshot on Ubuntu 18

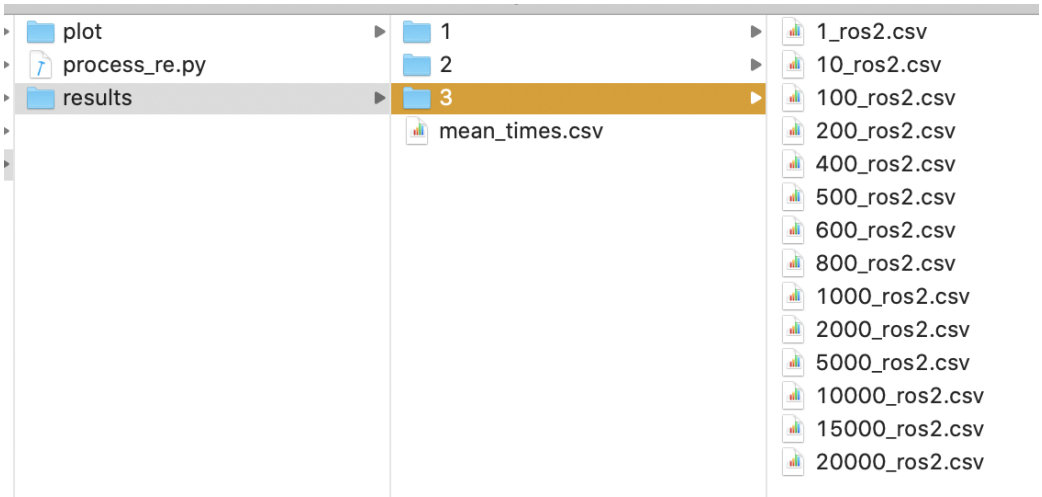

Figure B.2: Running experiment 3 times data collecting

Bibliography

[1] Sandra May. What Is Robotics? https://www.nasa.gov/audience/forstudents/k-4/stories/nasa-knows/what_is_robotics_k4.html. Nov. 9, 2009

[2] B. Bauml and G. Hirzinger. Agile Robot Development (aRD): A Pragmatic Approach to Robotic Software. Oct 2006.

[3] Avinash Gautam and Sudeept Mohan. A review of research in multi-robot systems. Aug 2012.

[4] Pedro U. Lima and Luis M. Custodio. Multi-Robot Systems. In Innovations in Robot Mobility and Control, pages 164. Springer, Berlin, Heidelberg, Aug 2005.

[5] Zhi Yan, Nicolas Jouandeau, and Arab Ali Cherif. International Journal of Advanced Robotic Systems, 10(12):399, 2013.

[6] Deborah Estrin, David Culler, Kris Pister, and Gaurav Sukhatme. Connecting the physical world with pervasive networks. 2002.

[7] Cellan-Jones. A15 computer to inspire young programmers. May 2011

[8] Gibbs, Samuel. Raspberry Pi becomes best selling British computer. The Guardian. December 2016.

[9] SunFounder PiCar-S Kit V2.0 page. https://www.sunfounder.com/picar-s-kit.html.

[10] Isaac Jordan. Robotic Middleware. https://github.com/Sheepzez/ros-multirobot-evaluation/tree/master/experiments/message_latency/no_echo_delay_no_disk_write. 2017.

[11] ROS Melodic Morenia. wiki.ros.org. Retrieved 10 June 2018.

[12] The Robotics Back-End Home page. https://roboticsbackend.com/what-is-ros. 2019

[13] ROS2 Overview Home page. https://index.ros.org/doc/ros2/

[14] Brian Gerkey. Why ROS 2?. http://design.ros2.org/articles/why_ros2.html, 2017

[15]"Ubuntu and Debian". Ubuntu.com. Canonical Ltd. Retrieved 14 December 2013.

[16]"Ubuntu and Debian". Ubuntu.com. Canonical Ltd. Retrieved 14 December 2013.

[17] What is Ubuntu MATE?. Ubuntu MATE Team. https://ubuntu-mate.org/what-is-ubuntu-mate/. 2019

[18] Star topology. http://www.telecomabc.com/s/star.html. 2015

[19] Star topology. Cisco Networking Academy 2014.

[20] Topics. http://wiki.ros.org/Topics. 2019-02-20

[21] TCPROS. http://wiki.ros.org/ROS/TCPROS. 2013-04-15

[22] What is DDS. https://www.dds-foundation.org/what-is-dds-3/

[23] Why Consider DDS. ROS on DSS. https://design.ros2.org/articles/ros_on_dds.html

[24] Where did DDS come from. ROS on DSS. https://design.ros2.org/articles/ros_on_dds.html

[25] Issac Jordan. Experiment https://github.com/Sheepzez/ros-multirobot-evaluation/tree/master/experiments/message_latency/bag-sender-receiver.2017.

[26] Drone (UAV) https://internetofthingsagenda.techtarget.com/definition/drone. July 2019

[27] Sunfounder Smart Video Car Kit for Raspberry Pi. https://github.com/sunfounder/Sunfounder_Smart_Video_Car_Kit_for_RaspberryPi.

[28] ROS 2 Overview. https://index.ros.org/doc/ros2/.