Fall 2016

An Investigation on a Mobile Robot in a ROS Enabled Cloud Robotics Environment

Theodore C. Smith

AN INVESTIGATION ON A MOBILE ROBOT IN A ROS ENABLED CLOUD ROBOTICS ENVIRONMENT

by

THEODORE CORNELIUS SMITH

(Under the Direction of Biswanath Samanta)

ABSTRACT

In modern day robotic applications, the use of cloud computing is being considered as a viable option giving rise to development of cloud robotics environment. Robots are also being developed to operate under an organized framework of robot operating system (ROS) for flexibility and better integration with robots of different types. In this work, an investigation is presented on the development of a mobile robotic platform and its integration in a ROS enabled cloud robotics environment. A mobile robotic platform was built with different sensors including a depth camera, integrated with Arduino and raspberry pi for interfacing the sensors, the drive system and on-board local processing of signals and with wireless communication capability for transmission and receiving data. The robot was operated using ROS framework within a cloud robotics network. Two issues of robot operation in ROS enabled cloud environment, namely, latency and data integrity were investigated for the developed robot under different operating conditions. The system was tested for baseline connectivity and under low bandwidth environment and performance was found to be satisfactory in the areas of latency and data integrity. Ongoing and future extensions are proposed to integrate this current robot with other existing robots within the ROS enabled cloud robotics environment.

INDEX WORDS: Cloud robotics, Network, Robot Operating System

AN INVESTIGATION ON A MOBILE ROBOT IN A ROS ENABLED CLOUD ROBOTICS ENVIRONMENT

by

THEODORE CORNELIUS SMITH

B.S.,Bethune-Cookman University, 2014

A Dissertation Submitted to the Graduate Faculty of Georgia Southern University in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

STATESBORO, GEORGIA

AN INVESTIGATION ON A MOBILE ROBOT IN A ROS ENABLED CLOUD ROBOTICS ENVIRONMENT

by

THEODORE CORNELIUS SMITH

Major Professor: Biswanath Samanta

Committee: Minchul Shin
Junghun Choi

Electronic Version Approved:

December 2016

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Biswanath Samanta for his encouragement and support throughout the course of this research. I thank Chris Reid who was my predecessor in the lab for all his help. I would like to thank the team that assisted in this work with me Emerald Simons, Gyunay Keten, Hunter Taylor, and Sylvia Bhattacharya- thanks for all your dedication. My mother, Jaratta Lamin, who taught me to reach for the skies and my friends who told me to see the goodness of the Lord

Especially my dearest friend Arianna Gray, whose unconditional love and support has motivated me through tough times.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

Arlobot – Arlobotic Platform System

IoT - Internet of Things

ROS- Robot Operating System

VM - Virtual Machine

Raspberry Pi- Rasp Pi

CHAPTER 1: INTRODUCION

1.1 The Growth of Robotic Systems

The study of Robotics is an amalgamation of studies between mechanical engineering, electrical engineering and computer science. Robotics concerns itself with the design, construction, operation, and application of robots as well as the computer systems for their control, sensory feedback, and information processing. Modern Robotics gives us the unprecedented ability to maneuver during various situations from manufacturing processes, assistive living, to many other scenarios including disaster management. Many robots do jobs that are hazardous to people such as defusing bombs, mines and exploring shipwrecks. Robotics is a rapidly growing field, as technological advances continue. Researching, designing, and building new robots serve various practical purposes, whether domestically, commercially, or militarily.

To meet these challenges, many robotics researchers have previously created a wide variety of frameworks to manage complexity and facilitate rapid prototyping of software for experiments, resulting in the many robotic software systems currently used in academia and industry (Kramer, 2007). Robots and manufacturing systems are becoming more flexible with progress because of computer technology and its many programming techniques.

1.2 The Present Work

The main hypothesis for this work is that with the use of a given network and infrastructure, a standardized robot can communicate and perform its tasks with the use of the cloud resources, if properly integrated, without a deterioration in network

performance and data integrity.

To support this hypothesis, the following objectives were established:

- Implementation of a mobile robotic platform (named in this work as Arlo Robot or Arlobot, in short) integrated with various sensors, including a depth camera, on-board processors with local processing and wireless communication capability.
- Integration of Arlobot within ROS framework.
- Operation of the Arlobot in cloud robotics environment.
- Investigations on the network latency and data integrity while operating the robot in the ROS-enabled network.

To achieve these objectives, a robotic platform was assembled using a robot base, sensors, drive system, an Arduino Mega and a raspberry pi as local processors and for communication wirelessly with the cloud environment. Software base was developed to adapt the robot within the ROS framework and integrate it with the cloud robotics environment already existing in the lab.

1.3 Organization of Thesis

The thesis is organized as follows:

Chapter 2 is the literature review that focuses on key points. First, features of Arlobot are described, along with its sensors and processors, and the network communication capability. Next cloud robotics and its challenges are reviewed. Next features of raspberry pi implementation on Arlobot are discussed. Ii is followed with brief discussions on Robot Operating System (ROS). Finally features of a depth camera used in this work are discussed.

Chapter 3 takes a look at the research methodology used to conduct this research. First the overall device configuration is presented. It is followed with the implementation of ROS within the cloud. Next details of the necessary configuration of the Arlobot to make it compatible with the cloud robotics environment is discussed. Finally, the design of the experiment to test the robot effect on network latency and data integrity, under different operating conditions, is presented.

Chapter 4 presents experimental results of network latency and data integrity. The results under different operating conditions are analyzed next.

Chapter 5 summarizes an overview of the results and recommendations for future work.

Additional documentation related to the research is provided in Appendices. The appendices include a list of Arduino and Python codes and scripts.

CHAPTER 2: LITERATURE REVIEW

2.1 Standardized System

Standardized System are functions that use more than one type of processor or core that helps in the performance of its tasks (Rogers, 2013).

Standardized systems have three guidelines to follow in order for the system to operate correctly:

1. The data traveling through the system must be processed by the machine, and thus exists a data chokepoint at the machine, reducing the reliability and increasing the response time of the system.
2. The system resources, both hardware and software, would exist at a single location, making any downtime the machine experiences catastrophic to the system.
3. Even with enormous complexity a system of multiple, corroborative robots performing tasks simultaneously would outperform a single machine as the task complexity increases (Cao, Fukunaga, and Kahng, 1997).

Standardized systems come in different shapes, sizes, and capabilities with that they complement each other and increase the capability of the system as a whole. Heterogeneous systems are more capable, efficient, fault tolerant, and expendable monolithic counterpart (Parker and Tang, 2006).

2.2.1 Arlobot Platform System

The robot, that has been used in this work, named Arlobot, is comprised of two 6 inch driven wheels, two electrical motors to drive the wheels, two 3 inch caster wheels,

an easy-to-machine plastic platform, a 12V rechargeable battery, a camera, and an LCD screen. The driven wheels are positioned at the sides of the robot and the caster wheels are positioned at the forward and rear positions of the robot. The robot is able to turn by adjusting the speed of the electrical motors independently. A Parallax HB-25 fan cooled motor controller is mounted beneath the platform for each of the electrical motors. The motor controller uses a single pulse to a set output. The specifications of the motor controller can be found in Appendix A. The motor controllers are linked to a Pololu Micro Maestro servo controller. The servo controller enabled individual speed and acceleration control for each channel. Its specifications can be found in Appendix B. The left and right Parallax motor controller channels W, R, B were connected to Pololu channels 1 and 2. The Pololu RX channel was connected to the Arduino microcontroller TX channel. The RX line is to receive serial commands. The Pololu TX channel was connected to the Arduino RX channel. The TX line is for responses to the serial commands. Figure 1 displays the motor and servo controller diagrams.

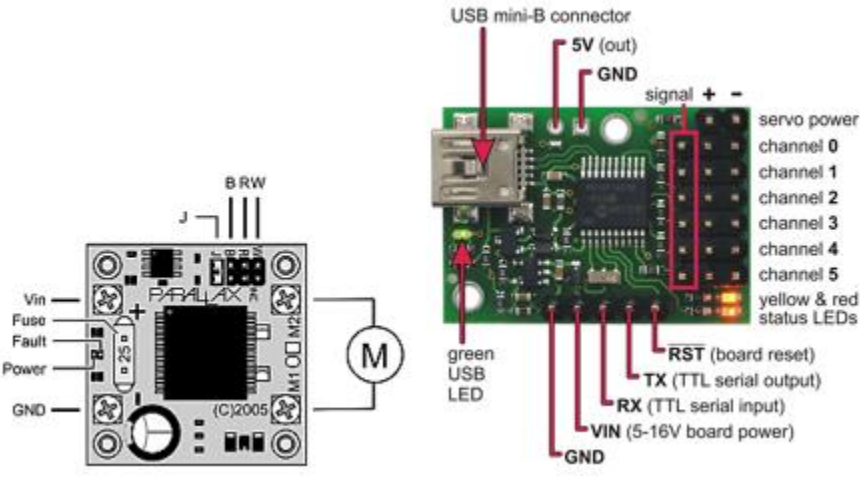

Figure 1: Parallax HB-25 (left) and Pololu Micro Maestro (right)

The various components used in the construction of the robot are:

This kit consists of a circular high-density polyethylene robotics platform having 220 square inches of usable surface area. It consists of hardware for both single and double battery mounting and a battery shelf designed to carry one or two 12 V lead acid batteries. A single caster wheel was used as the front wheel to provide direction and two motor mounted rear wheels provided the moving power.

Robot Base Front Wheels Rear Wheels

Figure 2: Parts used for the Arlobot

2.2.2 Arduino Microcontroller

The Arduino Mega 2560, the successor to the Arduino Mega, is a microcontroller board based on a ATmega2560 AVR microcontroller. It has 70 digital input/output pins (of which 15 can be used as PWM outputs and 16 can be used as analog inputs), a 16 MHz resonator, a USB connection, a power jack, an in-circuit system programming (ICSP) header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-

to-DC adapter or battery to get started.

The Mega 2560 differs from the preceding Mega in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega8U2 programmed as a USB-to-serial converter. This auxiliary microcontroller has its own USB bootloader, which allows advanced users to reprogram it.

2.2.3 Ping Ultrasonic Sensor

The PING Ultrasonic Distance Sensor (shown in Fig 3) is perfect for any number of applications that require distance between moving or stationary objects. The sensor measures distance using sonar; an ultrasonic (well above human hearing) pulse is transmitted from the unit and distance-to-target is determined by measuring the time required for the echo return. Output from the PING sensor is a variable-width pulse that corresponds to the distance to the target. 10 PING sensors were used for this project.

Figure 3: PING Ultrasonic Sensor

The robot has been designed to perform various tasks such as avoiding physical contact with objects, following an object, and moving in a formation with other robots. The

tasks were accomplished by incorporating PING sensors to detect presence of objects in its path. Five PING sensors at the forward and rear positions of the robot are placed 30° apart, as indicated in Figure 4.

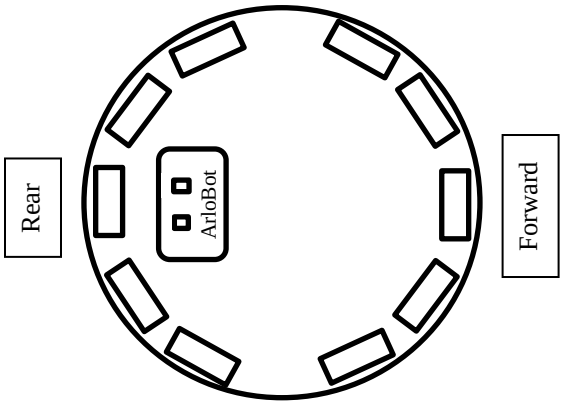

Figure 4: PING Sensor Layout

The PING sensor utilizes a very high frequency sound (40 kHz, it is beyond human audibility) and its echo to determine the distance of an object. A brief ultrasonic chirp produced by the speaker and the delay of the echo exerted on the microphone is measured. The distance is calculated by multiplying the time by the speed of sound in the air (340m/s). Figure 5 represents the principle behind the sensor.

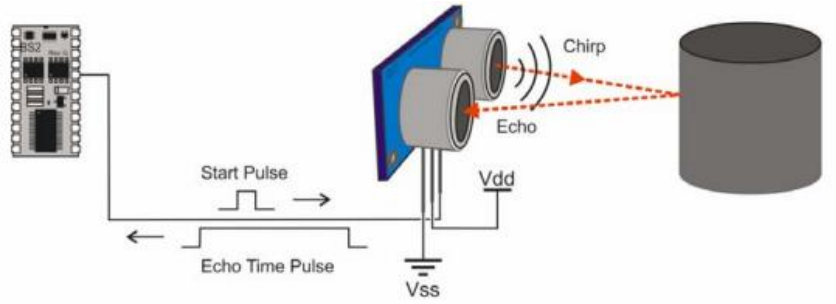

Figure 5: PING Sensor Operation

A platform tier was added to the structure to serve as a base for the pole supporting the camera and LCD Screen. A 5 inch space between the platforms ensured that all the components had enough clearance. The support pole was positioned at the center of the platform. The Arlobot power distribution board was placed on the platform at the forward position to allow easy on/off access to the users. The power distribution board uses the 12V battery as the power source and distributes 12V to each of the Parallax motor drivers, and 6.5V to the Arduino microprocessor board. The power distribution board wiring is displayed in Fig. 6.

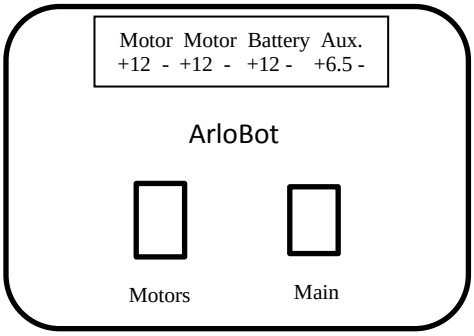

Figure 6: Switch case for the Arlobot

2.2.4 Raspberry Pi

Raspberry Pi (shown in Fig. 7) is a smaller than a phone computer that can be utilized as a desktop PC. It is little in any case, sufficiently capable to do consistent word

processing, spread sheet investigation and other general exercises. It can likewise do nonstandard task like being a media server or go about as a smart TV. The Raspberry Pi was developed in the United Kingdom by the Raspberry Pi Foundation with the intention of promoting the teaching of basic computer science in schools and developing countries. The original Raspberry Pi and Raspberry Pi 2 are manufactured in several board configurations through licensed manufacturing agreements with Newark element14 (Premier Farnell), RS Components and Egoman. The hardware is the same across all manufacturers.

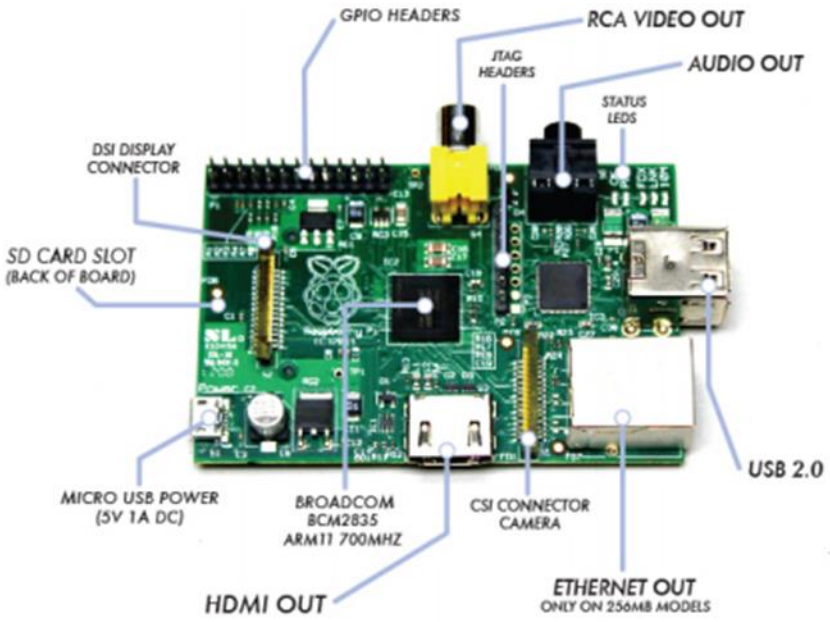

Figure 7: Raspberry Pi

2.3 Cloud Robotics

2.3.1 Overview of Cloud Robotics

NIST defines cloud computing as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider

interaction." (Mell, 2011)

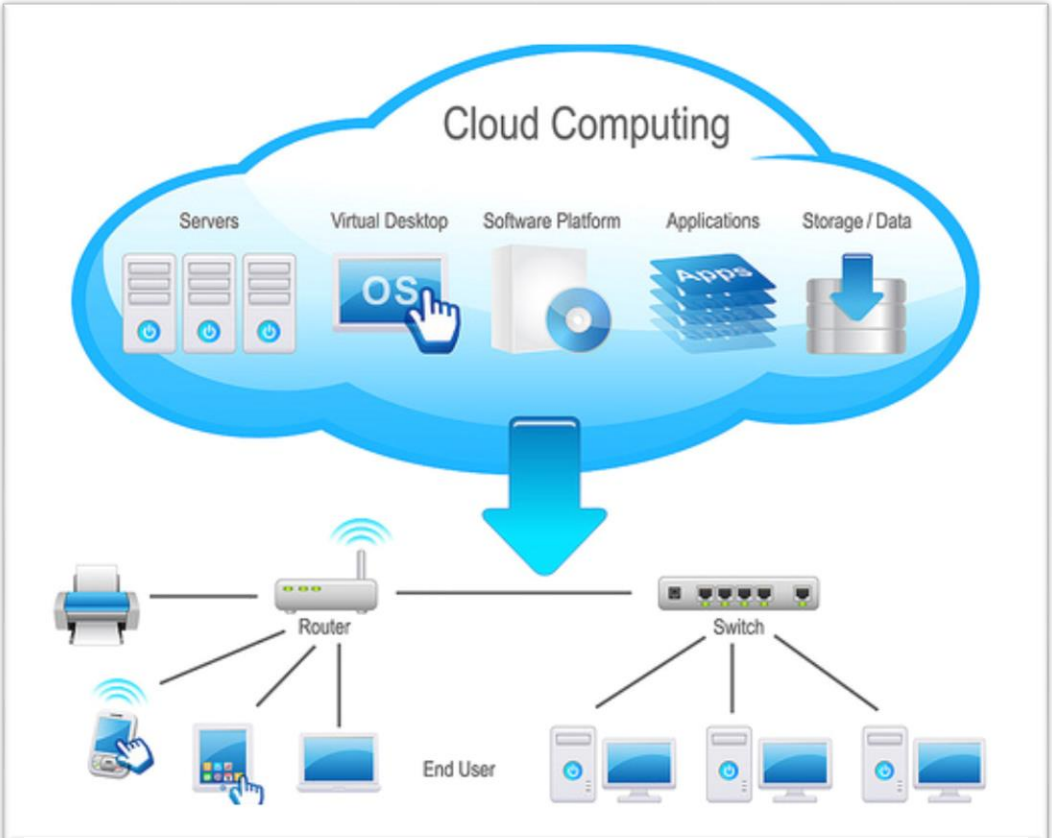

Figure 8: A brief graphic explaining cloud storage and cloud computing

Cloud robotics and cloud computing in general, offers the concept of utility computing; instead of buying computational resources to own, these resources can be provided for use on the short-term, as-needed basis (Armbrust, et al. 2010). With the use of cloud robotics, robots are able to store, interact, and solve more complex information in a datacenter. This way there is a decreased dependence needed in the middleware of a system.

2.3.2 Advantages of Cloud Robotics

With the growth of cloud robotics it should be assumed that it is doing very well.

Advantages of cloud robotics are:

1. Big Data: access to updated libraries of images, maps, and object/product data,
2. Cloud Computing: access to parallel grid computing on demand for statistical analysis, learning, and motion planning,
3. Collective Learning: robots and systems sharing trajectories, control policies, and outcomes, and
4. Human Computation: use of crowdsourcing to tap human skills for analyzing images and video, classification, learning, and error recovery.

2.3.3 Disadvantages of Cloud Robotics

Although cloud robotics is the change that society is vastly picking up, it is still taking time to get its proper share, because it has a few disadvantages such as:.

- Cloud-based applications can get slow or simply become unavailable leaving the robot "brainless".
- Tasks that involve real-time execution require onboard processing.
- Security issues
- Computation challenges:
    - Offload decision
    - Offload strategy

2.4 Challenges in Cloud Computing

2.4.1 Network Latency

Dealing with output to input network transportation paying attention to latency is key to successfully transmitting information. Network latency is an expression of how much time it takes for a packet of data to get from one designated point to another

(Rocus, 2014). In order for cloud computing to work effectively the latency would have to be managed proficiently.

2.4.2 Data Integrity

Data integrity is of prime importance for proper transfer of data over the network as it pertains to the proper working of the devices connected.. Data integrity has been used as one of the criteria in this work.

2.5 Virtualization

Virtualization is another element that adds on to latency in a cloud. The reason this needs proper consideration is because there is a change from virtual machine running task to a more complex web of hypervisor running several virtual machines. There have been tasks to make virtual networks, but it gave rise to many problems with several packet delay of data making too many decisions without properly going through correct process. Virtualization is the concept of abstracting hardware resources such that multiple paths to hardware can be provided, allowing numerous users' access to the same hardware. The use of virtualization minimizes the number of server machines which can save energy that is continuously used in the data center.

2.6 Robot Operating System

Robot Operating Systems (ROS), which was formulated by Quigley et. al, is not an operating system in the traditional sense of process management and scheduling. Rather, ROS provides a structured communications layer above the host operating

systems of a heterogeneous compute cluster. (Quigley, 2009) In this work the focus is to test the current condition of a wireless connectivity between a device and a network. ROS is a framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms (Quigley, 2007). When dealing with ROS there are a few basic definitions below, which will be referred to often in this work.

- Nodes- Nodes are processes that perform computation. ROS is designed to be modular at a fine-grained scale; a robot control system usually comprises many nodes. A ROS node is written with the use of a ROS client library, such as roscpp or rospy. Nodes communicate with each other by passing messages.
- Messages- Nodes communicate with each other by passing messages. A message is simply a data structure, comprising typed fields. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types. Messages can include arbitrarily nested structures and arrays (much like C structs). A node sends a message by publishing it to a given topic.
- Topics- Messages are routed via a transport system with publish/subscribe semantics. A node sends out a message by publishing it to a given topic. The topic is a name that is used to identify the content of the message. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics. In general, publishers and subscribers are not aware of each other's' existence. Although the topic-based publish-subscribe model is a flexible communications paradigm, its "broadcast"

routing scheme is not appropriate for synchronous transactions, which can simplify the design of some nodes.

- Services- The publish/subscribe model is a very flexible communication paradigm, but its many-to-many, one-way transport is not appropriate for request / reply interactions, which are often required in a distributed system. Request / reply is done via services, which are defined by a pair of message structures: one for the request and one for the reply. A providing node offers a service under a name and a client uses the service by sending the request message and awaiting the reply. ROS client libraries generally present this interaction to the programmer as if it were a remote procedure call.
- Packages – ROS packages are the base level bundles of code that are designed to be uploaded for sharing and downloaded for use. ROS packages can contain zero or more of nodes, messages, libraries, configuration files, build instructions, documentation, and more. Because of ROS's open source nature, packages are often free to download and use, and are developed by programmers across the world for a multitude of purposes that, due to the modular, standard-focused nature of ROS, can be applied to many different types of robots.
- Master- The ROS Master provides name registration and lookup to the rest of the Computation Graph. Without the Master, nodes would not be able to find each other, exchange messages, or invoke services.

2.7 ASUS Xtion PRO LIVE DEPTH CAMERA

The ASUS XTION PRO LIVE is based on the projective stereo technology

developed by PrimeSense. It provides RGB and depth images with VGA resolution (640 × 480 pixels) at a rate of 30 frames per second. A higher frame rate of 60 frames per second can be obtained by reducing the resolution to QVGA (320 × 240 pixels). Compared to the Microsoft Kinect the ASUS XTION PRO LIVE has the following advantages. The RGB and depth images are time synchronized and can be registered to each other on-board the camera. Furthermore, the camera has only a weight of ~150 g (Microsoft Kinect ~440 g) and only needs the USB connection as power supply.

Figure 9: ASUS Xtion PRO LIVE camera: 1 IR projector 2 RGB camera 3 IR camera

The depth values are encoded as 16 bit unsigned integer values representing the depth in millimeters. According to the specification, the depth values range from 0.8 m to 3.5 m. In experiments measurements in the range of 0.7 m up to 9.5 m have been obtained. Depth registration causes invalid pixels at the border of the depth image, because the depth image has to be transformed into the viewpoint of the RGB camera to associate every color pixel with a depth pixel.

CHAPTER 3: METHODOLOGY

3.1 System Configurations

In order to run these experiments on the topics examined in this study, mainly network latency and data integrity, the Arlobot was integrated with a cloud robotic environment developed in the bIRIS (Bio-inspired Robotics and Intelligent Systems) lab. Initially, virtual datacenter was used to give the establishment on which the cloud service, e.g., remote operation, and image processing were supported. ROS was integrated to the cloud computing environment to facilitate the operation of Arlobot.

The Arlobot was configured to connect to the cloud network through a wireless adapter with the use of a passcode. There was an extensive amount of information added to the ROS framework such as ROS drivers, directing robot movement, acquiring/storing robot ping sensor data, and attempting to integrate robots into a single control system.

For the robot test, the latency and data integrity of the connection between the robot and the cloud was tested by streaming data packets to the desired robot and monitoring what was returned from the packet. The information was used to see how well the network was performing over a long period of time and also viewed how it reacted with a lot of activities going on in the network. In addition to that, the Arduino was strictly used to navigate the Arlobot and follow given directions. The use of image detection was also viewed in this paper to see how it operated with Raspberry Pi and Arduino.

3.2 Combination of Instruments

A Raspberry Pi and an Arduino code were written to communicate with one another

over a serial line in order to establish the Pi as the controller and the Arduino as a controlled tool. The Raspberry Pi receives a string from the Arduino indicating the distances and the corresponding sensors. The string is parsed so that the information becomes useful through the “mystring.split()” command. This makes it so that the variable (a) now stores the sensor that has the lowest reading. The ard.inWaiting() command recognizes the amount of input from the Arduino and the ard.flushInput() command flushes the data if it is taking up space in the buffer. This does two things. The first thing is that it keeps the buffer low so that there is no latency issues with the processor and provides the newest commands from the Arduino. A string is created with name “left” and another with the name “right”. These strings have the characters corresponding to the sensor numbers in our robot and are used as references for turning the robot right or left (+ or -, respectively). The first character of the string sent from the Raspberry Pi initializes the response actuated from the Arduino. The characters are a guidepost for how the Arduino should treat the incoming string. If a (+) sign is received, then the robot will turn left, if an (=) sign is received, the robot will come to a full stop. The “F” and “R” represent forward and reverse. Figures 10 and 11 show the overview of processing in the Arlobot.

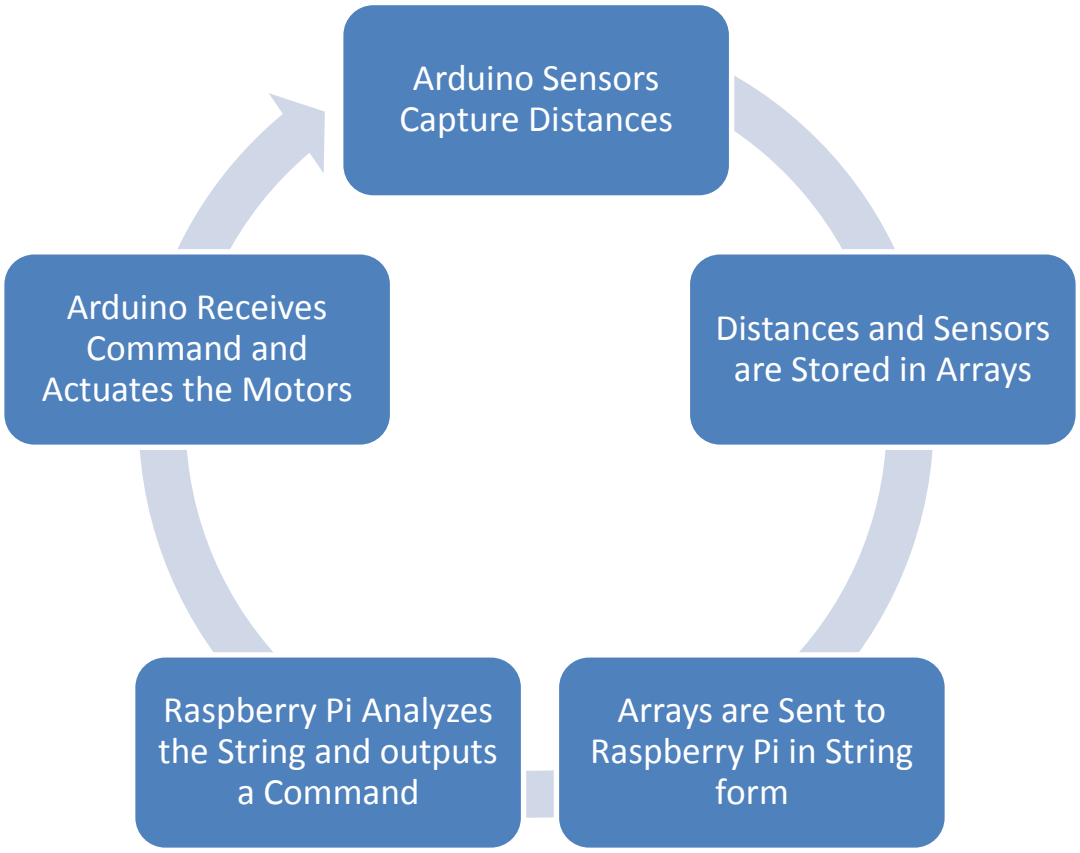

Figure 10: This is the overview of the overall process taking place

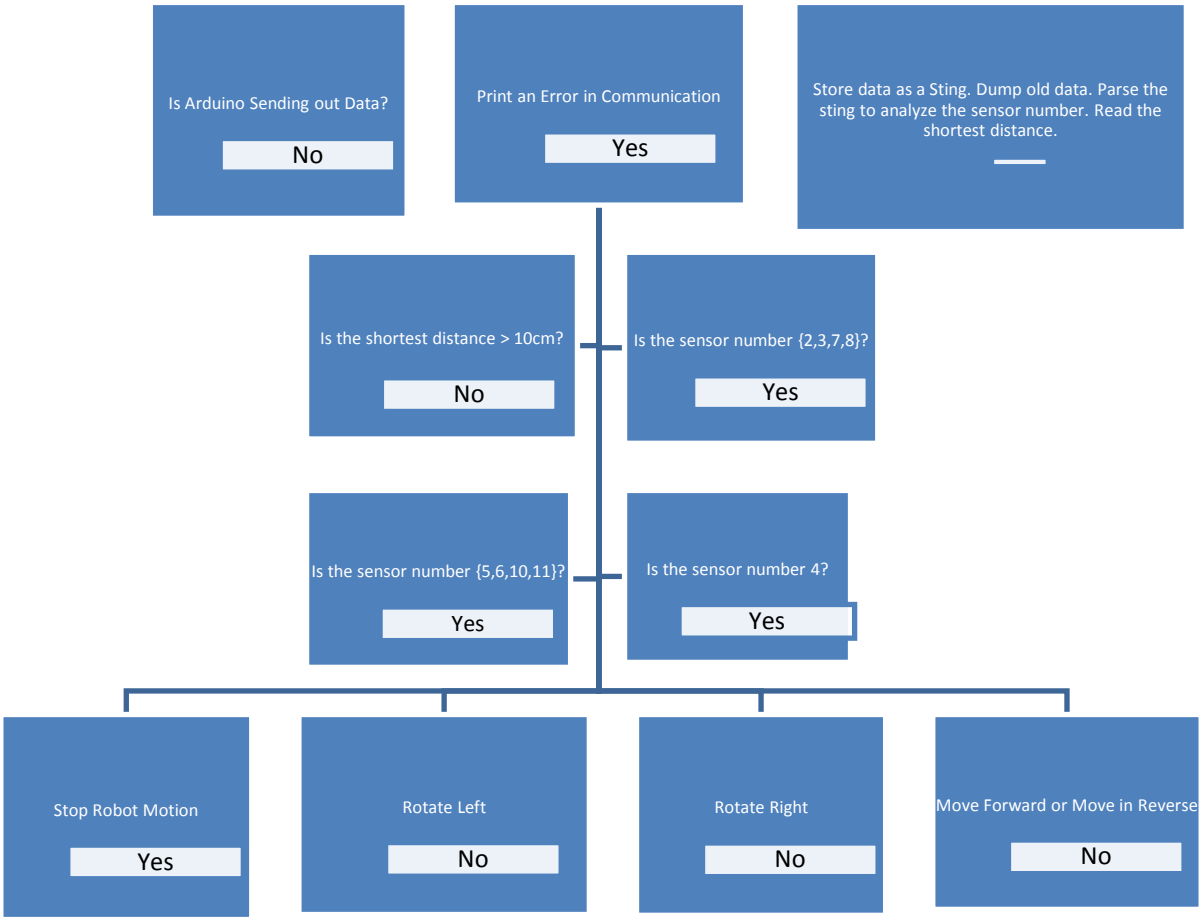

Figure 11: Process flow chart in Raspberry Pi

The Arduino code consists of a part that sorts the sensors and their distances into arrays and then sends those arrays out through the Serial.begin (9600) serial communication. The number of characters is designated to be nine as fits the number of bytes that are sent from the Raspberry Pi to the Arduino within the output. The code organizes these strings by recognizing the first character in the string to be "<" sign and the last character in the string to be ">". The void loop RecvWithStartEndMarkers is responsible for performing this action and storing the data in an index. The

VoidParseData void loop takes the index and parses it into usable data. For example, the messageFromPC and the IntegerFromPC are named from the string stored in the Index coming from the Raspberry Pi. This is stored as an integer or a character and used to actuate the motors through the Polulu controller.

3.3 bIRIS Network

3.3.1 Virtual Datacenter

The bIRIS virtual data center was chosen to make efficient use of server hardware while providing a dynamic, reconfigurable environment in which services and control systems could be built, altered, and shut down as needed by the client robot or users. Furthermore, the virtualized datacenter provides a high degree of fault tolerance for the virtual machines that isn't found in physical datacenters; virtual machines can be immediately rebooted upon crashing, or migrated to another host machine without an interruption in service should the host machine fail.

Six physical servers were built in total to provide ample resources for the virtual datacenter and future expansion. Five of the machines functioned as host servers for virtual machines, while the sixth provided network-attached storage (NAS) for the storage of the virtual machine files.

Through software-defined networking (SDN), the virtual machines and host machines did not have to use the same network interfaces for communicating with other devices, allowing for the segregation of network traffic based on the nature or purpose of the data being transferred. Three networks were defined to segment network traffic into three categories: management, storage, and testbed. Management traffic was placed on its

own network to prevent the disruption of the virtual datacenter operation by high network usage by client or storage devices. Finally, this leaves the testbed network, which contains all robots, virtual machines hosting control software, cloud resources outside of the virtual datacenter, and client PCs through which users interact with the cloud.

Physically, the devices were connected to a Dell PowerConnect 6248 Ethernet switch via category 6 twisted pair cables. The PowerConnect switch was segmented into three virtual local access networks (VLANs) to create the three aforementioned networks (shown in Figure 12) and prevent traffic from crossing over the networks. For wireless connectivity on the Testbed network, a Linksys E2700 wireless router was used to provide a Wi-Fi access point for remote systems.

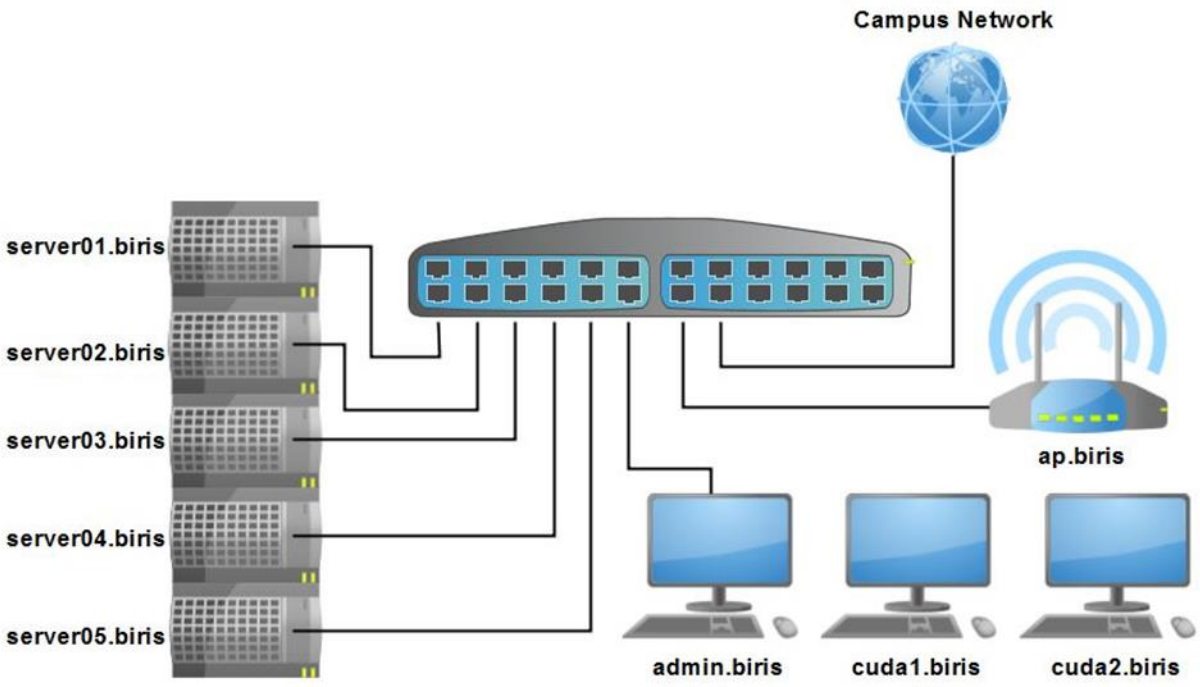

Figure 12: Testbed Network Physical Topology

Six computers were built to function as the host servers of the datacenter, though

one was later designated to provide NAS for the datacenter. The six machines were built nearly identically, with the sole difference being the NAS machine contained larger hard drives for increased storage capacity.

Five machines were provided with USB flash drives imaged with VMware's ESXi 5.5 Hypervisor operating system. The ESXi configuration screen is shown in Figure 3.4. USB flash drives were used as opposed to the local hard drives so that there would be no need to create a small partition on a hard drive in each machine for operating system storage; this way, the hard drives remain fully free for cloud storage use. Each machine was connected to an Avocent Autoview Digital KVM Switch, to which a console was attached that would provide a direct console user interface (DCUI) to each machine that was configured to be used in the datacenter.

Next, VMware's vSphere Client was installed on a computer temporarily located on the management network. Through the vSphere client, a local datastore was configured on which to store the datacenter management VM. Server01’s networking was further configured to designate the network interface connected to the management network to be used for management traffic.

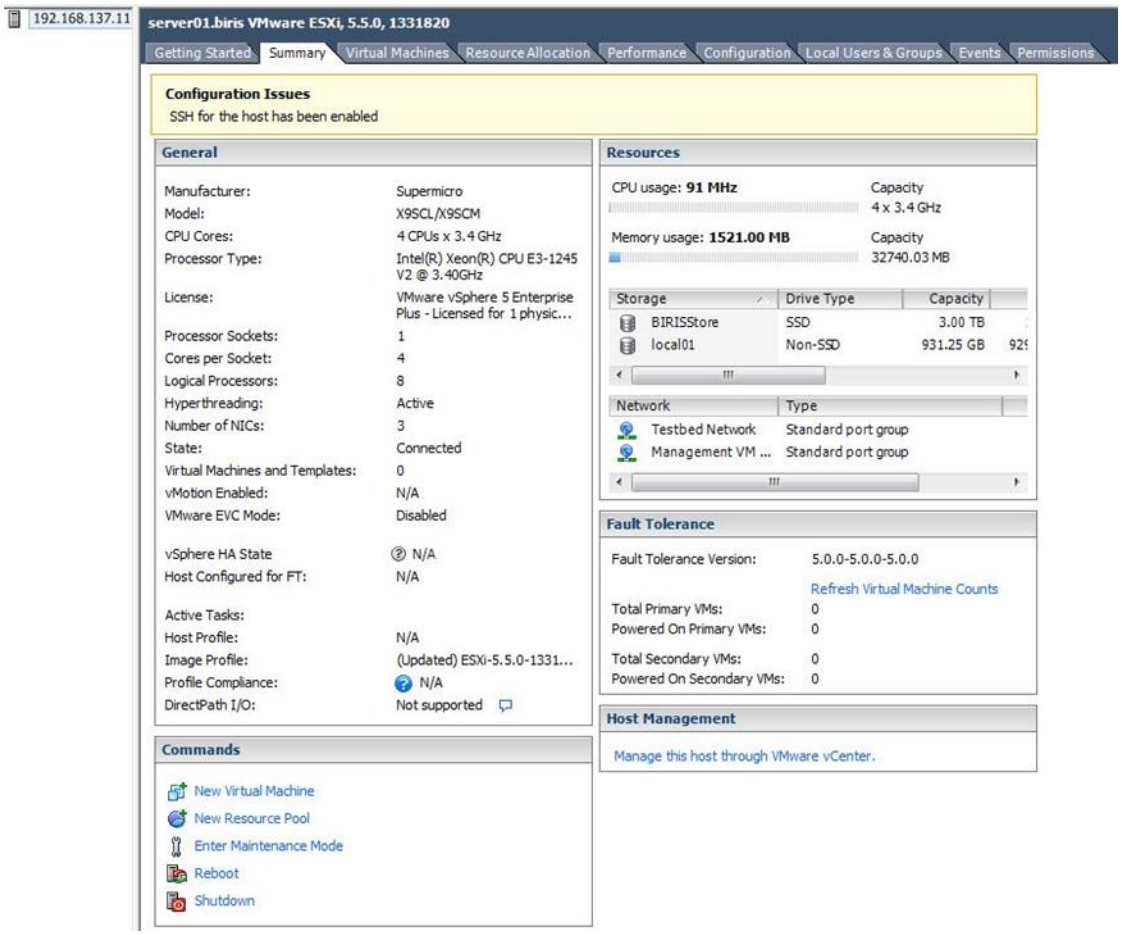

Figure 13: vSphere Web Client Interface to ESXi Machine

Through the vSphere Client interface, a virtual appliance (vApp) was deployed on server01. vApps are simply VMs with a preconfigured operating system and settings. This particular Linux-based vApp was loaded with the vCenter Server management software, which provides a management interface at the datacenter level, as opposed to the host level the ESXi operating system provides.

Through the DHCP server, the IP address of every Testbed network interface (whether a robot, PC, or virtual machine) was defined in a single location, as opposed to each robot, PC, and virtual machine deciding its own IP address.

The bIRIS Testbed system comprised of a wide range of systems and robots, including robots, cloud assets not oversaw by the datacenter, control systems and the VMs they were facilitated on, and the customer PCs through which clients could get to the Testbed system. The physical assets for the datacenter and in addition an example of the robots utilized for this study are shown in Figure 14.

Figure 14: The bIRIS Datacenter and Robots

3.3.2 Virtual Machines

A virtual machine (VM) is an operating system or application environment that is

installed on software which imitates dedicated hardware. The end user has the same experience on a virtual machine as they would have on dedicated hardware (Rouse, 2013). Numerous VMs were made at he datacenter to host software devoted to interfacing the robots with the ROS system or giving cloud administrations to the robots. These VMs had a system interface on the Testbed Network and were statically given IPs through the DHCP server, and also area names through the DNS server. Besides, a remote desktop convention (RDP) application was introduced on these VMs to give clients – through the physical PCs on the Testbed Network – access to the VMs to create and test programming.

3.3.3 Client Computer

A client computer is an individual PC that gets to the data and programs stored on a server as part of a network environment. Several physical PCs were put on the Testbed Network to permit users' entrance into the system to design robots and run programming situated on VMs. RDP programming was introduced on these PCs to be utilized to get to the VMs. These client customer PCs were furnished with a NIC such that the PCs would keep up their association with the campus network while likewise keeping up an association with the BIRIS Testbed Network.

3.4 Robots

Figure 14 shows the Arlobot robots used in this work.

Figure 15: Arlobot and Arlobot with Raspberry Pi

3.5 ROS in the Cloud Network

ROS was the key component in the cloud network for integrating the robot with the network. There were a few different applications added to the ROS, the first helped run the core ROS functionality which was a Linux Ubuntu virtual machine. Linux Ubuntu uses a unit as the main user interface for different software. This is important because Linux is really used as a stoplight, meaning it's only reason for being used in this scope is to direct ROS traffic throughout the network, this virtual machine was not used for any software development. The Linux VM was used in a minimal version of ROS that had only the base communication libraries which didn't need GUI tools because it wasn't needed in the scope of this work.

Once the first virtual machine was built, another virtual machine was made and this one used all of ROS VMs applications that included many ROS packages for simulators, navigation, and visualization tools. The purpose for this ROS virtual machine was for a template from which figure ROS VMs would be cloned. The workspace was made to keep track of everything in the software. There was source control software that was used to easily track change in code and distribute code. The VM was converted to a template once the RDP software was added to the VM to give client computers access to the network.

The vCenter Server software permits the use to make utilization of VM template. These templates are VMs that can't be turned on, however can rapidly be cloned into operational VMs. VM Templates can have extra programming introduced and documents put away on the PC this also have a custom equipment design. The beforehand made template was utilized as the establishment of all VMs that gave cloud robotics autonomy administrations through ROS.

3.6 Setup of Arlobot

3.6.1 Arlobot Installation

The Arlobot was loaded with Linux through a raspberry pi which had its own hostname. This Pi has it’s own firmware and a Micro SD card that could support a list of data. The Arlobot is shown in Figure 17.

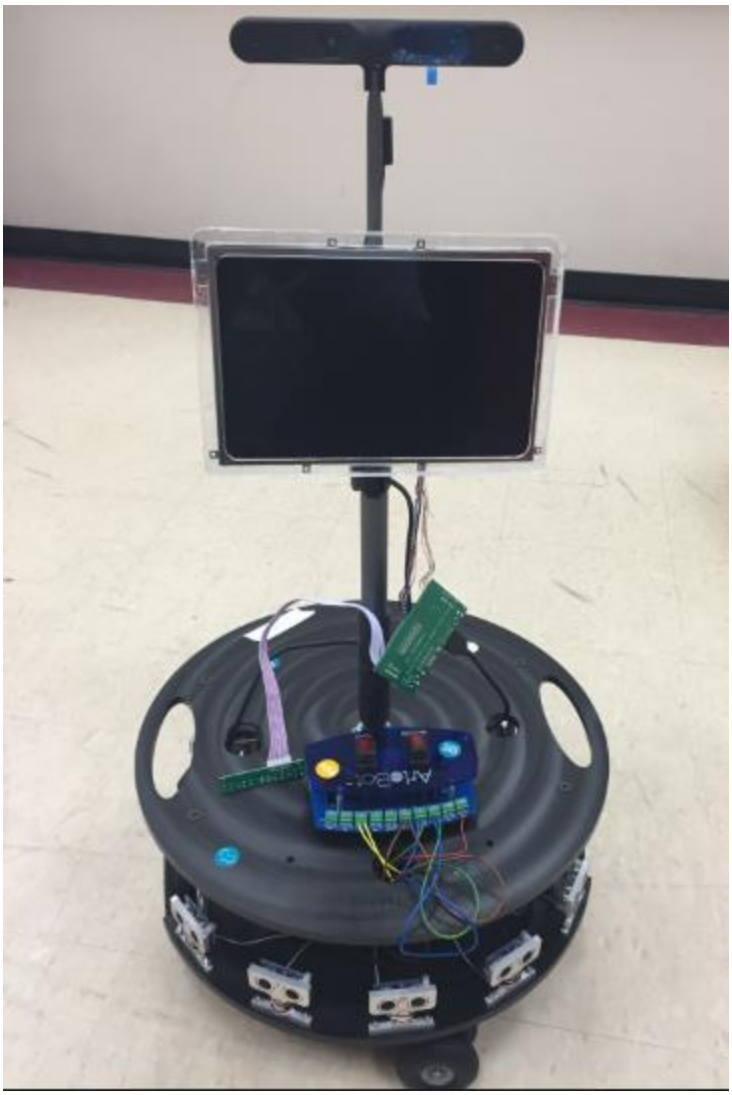

Figure 17: Arlobot with Raspbeery Pi and ASUS Xtion PRO LIVE

3.6.2 Arlobot Wireless Configuration

The Arlobot is one of the simpler devices to integrate because unlike other devices the raspberry pi helped in all the network system necessities. The Arlobot's native wireless interface was connected to the bIRIS Testbed Network using the network's SSID and the robot password.

3.6.3 Arlobot ROS Driver

The full ROS software was introduced on the Arlobot, alongside the Arlobot-

particular ROS bundles fundamental for interfacing with the automated base. Also, ROS-OpenNI programming was introduced for interfacing with the Arlobot's ASUS camera.

3.6.4 Raspberry Pi Specifications

All Raspberry Pi's include the same VideoCore IV GPU , and either a single-core ARMv6-compatible CPU or a newer ARMv7-compatible quad-core one (in Pi 2); and 1 GB of RAM (in Pi 2), 512 MB, or 256 MB (in older models A and A+). They have Secure Digital (SD) (models A and B) or MicroSD (models A+ and B+) sockets for boot media and persistent storage. In 2014, the Raspberry Pi Foundation launched the Compute Module, for use as a part of embedded systems for the same compute power as the original Pi. In early February 2015, the next-generation Raspberry Pi, Raspberry Pi 2, was released. The system does not come with a built in real time clock. Operating systems include any unix-like compiled for ARM architectures including linux and freeBSD.

The detail specification of this board is given in the Fig 16.

Component	Type/Count
SoC	Broadcom BCM2835 (CPU, GPU, DSP, SDRAM, USBport)
CPU	700 MHz ARM1176JZF-S core (ARM11 family)
GPU	Broadcom VideoCore IV @ 250 MHz
Memory	512 MB (shared with GPU)
USB 2.0 ports	2
Video input	A CSI input connector
Video outputs	HDMI
Audio outputs	3.5mm jack, HDMI
Onboard storage	SD card
Onboard Network	10/100Mbps Ethernet (8P8C)
Low-level peripherals	8 × GPIO,UART, I²C bus, SPI bus with two chip selects, I²Saudio +3.3 V, +5 V, ground
Power ratings	700 mA (3.5 W)
Power source	5 volt via MicroUSB or GPIO header
Size	85.60 mm × 53.98 mm (3.370 in × 2.125 in)
Weight	45 g (1.6 oz)
Supported OS	Arch Linux ARM, Debian GNU/Linux, Gentoo, Fedora, FreeBSD,NetBSD, Raspbian OS, RISC OS, Slackware Linux

Figure 16: Specification for Raspberry Pi

An LCD screen was added as an interfacing unit. The monitor would allow Python scripts to be adjusted on the fly. It would display images dependent on the environment that the robot is responding to. A frame was designed and laser cut from a 0.118” thick acrylic sheet. The frame comprised of four layers. The back layer consisted of a sheet with a cutout for the power connection port. Two shimming layers with a cutout size of the LCD screen was added to provide a gap between the back layer and the screen. The gap was to ensure that the LCD screen circuit di not overheat from constriction. A front layer completed the assembly of the frame by sandwiching the screen between the front and back layers. The layers were welded together with an acetone solution. The screen was to tilt about the horizontal axis 50° so that the orientation of the screen was able to fit the purpose of operation. Two brackets with slots

to allow the tilting of the screen were attached on the back of the frame. A mount was 3D printed with ABS plastic to serve as a pivoting point for the frame. The piece also mounts the frame onto the robot's camera supporting stand. The schematics of the frame assembly and mount are provided in Appendix A.

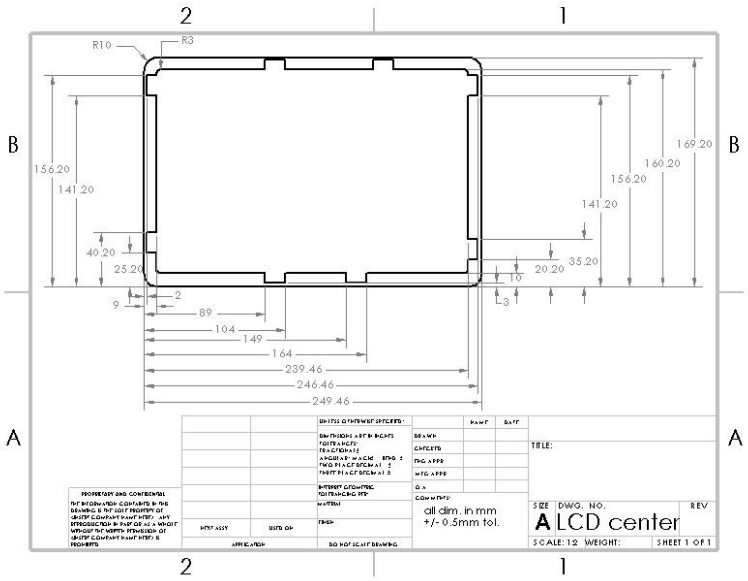

Figure 17: Layout of the Raspberry Pi Case

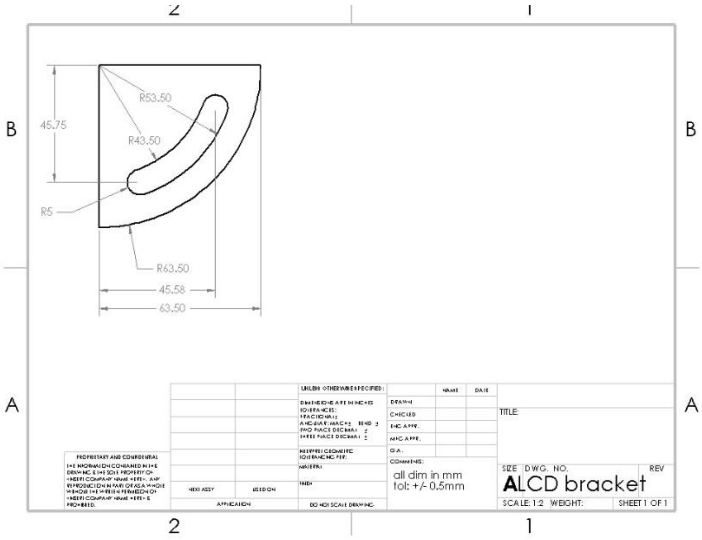

Figure 18: Arch holder for Raspberry Pi

3.6.5 Arduino Mega Board

This microcontroller was the brain of the robot. It processed the data from the sensors to tell the robot where to go. The Arduino Mega 2560 microcontroller was used to program the robot. The specification for the microcontroller can be found in Appendix C. 5 V supply from the Arduino is provided to the Pololu and ten PING sensors. Each PING sensor signals were connected to signal input channels 2 to 12.

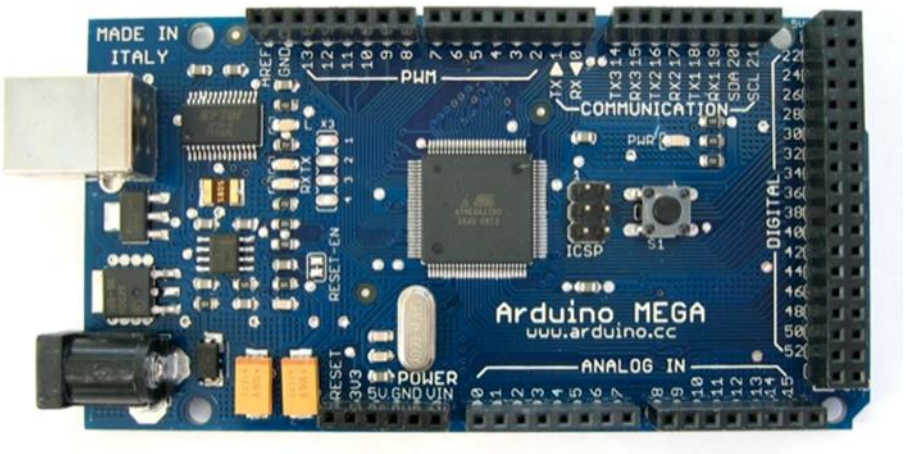

Figure 19: Arduino MEGA Board

The details of the Arduino Mega components are below.

- Microcontroller: ATmega2560
- Operating voltage: 5 V
- Input voltage (recommended): 7-12 V
- Digital I/O pins: 70 (of which 15 provide PWM output)
- Analog input pins: 16*
- DC current per I/O pin: 40 mA
- DC current for 3.3V pin: 50 mA
- Flash memory: 256 KB of which 8 KB used by bootloader
- SRAM: 8 KB
- EEPROM: 4 KB
- Clock speed: 16 MHz

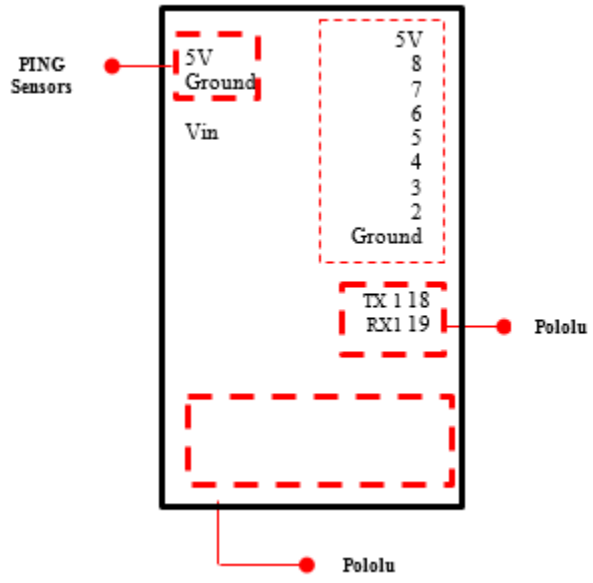

Figure 20: Arduino microcontroller wiring diagram

3.7 Arlobot ROS

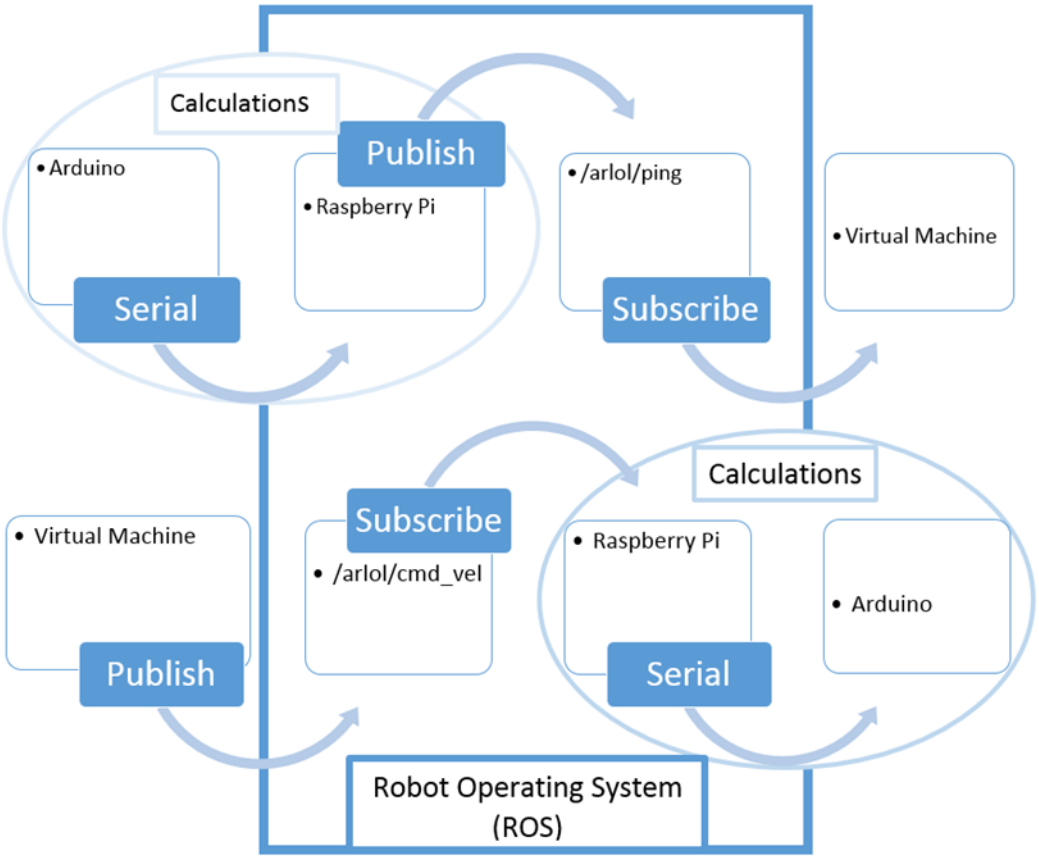

Figure 21: Complete Process for ROS

In the complete process above was launched in the console with the command line arguments to specify IP address of the targeted Arlobot and to give all of the ROS computations into a specified location. The horizontal latter step is necessary to properly execute all the needed operations for the network and for the data to transfer properly. The process begins with the Arduino and goes through publishers and subscribers and end in the Virtual Machine. Once in the virtual machine the information is processed the ROS network monitor and returns all the information back to the Arduino to execute.

The Arlobot was used in many different steps dealing with the time the ROS network execute the time. Most of the times were at random, but two were done at the same time in the day time and night with seconds between them to give a clear understanding of the time differences. The time analysis is to see how the latency is executed and helps in understand when best to run these experiments in in later works.

3.8 Image Detection

One of the primary goals of the project was to integrate a camera into the operation of the robot in order to perform long-range distance evaluation and decision-making. This would be accomplished by using an Asus Xtion Pro Live camera that included a laser rangefinder in its hardware. Using the laser rangefinder, the camera would be able to evaluate the distances to various objects in the two-dimensional image that it captures, allowing for more accurate distance evaluation over a significantly increased range. This data would then need to be communicated to the Raspberry Pi used to control the robot, which would use the data in conjunction with the data received from the array of Ping sensors to make decisions with regard to motor and motion control.

Building OpenNI2 from source is the main part of this operating well, but this would take a long time. To save time and work efficiently the information for the Raspberry Pi was packaged from Hirotaka's website, this saved a lot of time and worked exceptionally well for the main objective. The data is simply depth information for each pixel position. When the SimpleRead program was running understanding this will make a lot more sense when an object is moving back and forth in front of the camera.

The area scanned by ASUS XTION Pro Live below shoots a person, the manufacturer advices to keep the device at a distance of 1 and 8 meters from the

designated object because less is needed for the busts.

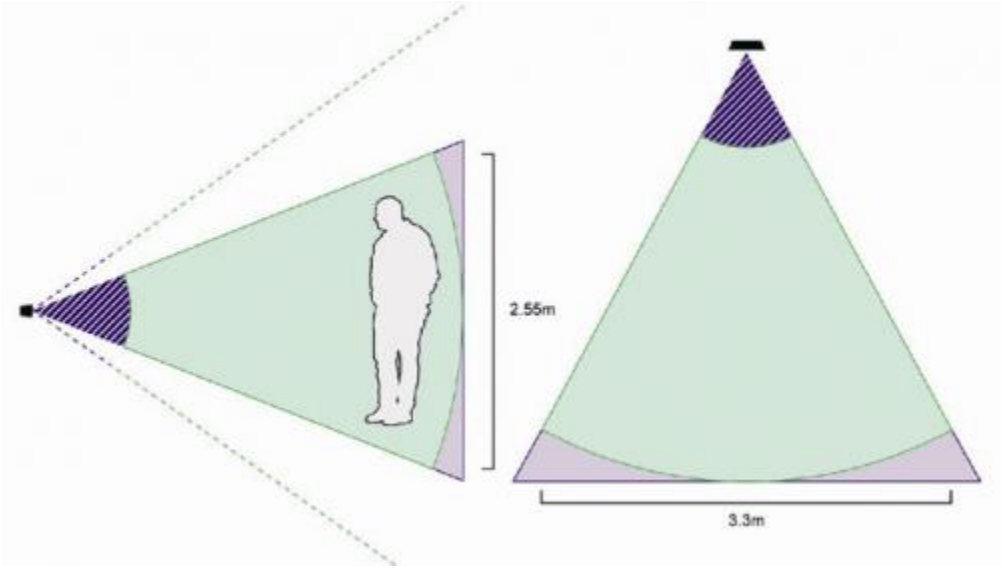

Figure 22: ASUS Xtion Shot of a Person

The most important point of view in this phase is to find a position in which the sensor operates at its best, and manages to produce reasonable details of what has been framed. The usage of the GPUs of graphics cards implies some compatibility limits, and in fact along with the software a compatibility list is supplied with the various graphics cards. The software has two operating modes: a normal one (Standard, with a 1x1x1m volume, subdivided in 256 elements for each dimension) or a high resolution one (Hi-res, with the same volume, but with 512 elements). (Landoni, 2015)

Of the potential methods assessed for obtaining and communicating range data from the camera to the Raspberry Pi, the most successful was the use of a pre-prepared archive file that operated on open Frameworks, an open source C++ toolkit that is designed to be easily used across multiple operating systems and IDEs. As the given archive could not operate on the base software of the Raspberry Pi, the open Frameworks software had to be installed before downloading the camera control archive. Following

the initial setup, the archive was installed, which allowed for range data to be recorded and sent to the Raspberry Pi.

3.8 Experimental Design

There were plenty of control and data acquisition used with the Arlobot and were implemented in the testing of the environment. Taking a look at how the ASUS camera will also be added to this the plan is to successfully have the camera pick an object up at a certain distance and have to Ping sensors react to that distance and send it to the Raspberry Pi. The ASUS will react when given objects are added around it and it should send a response to the Raspberry Pi to activate and the Arlobot will follow its main instructions.

To investigate the performance of Arlobot in the network, a network performance baseline had to be established, highlighting the least amount of measures of the robot needed to characterize the environment Network performance was determined based on three points:

- Network throughput, the speed at which data is transferred between devices,
- Network latency, the amount of time a data packet is in transition, and
- Dropped packets, the percentage of data packets that do not arrive at their destination and must be retransmitted.
- How the network reacts with various times in the network.

The normal idleness experienced by Arlobot is plotted for every experiment and the measurable parameters of the analysis are accounted for in a table. For the computation of the factual parameters, times of strangely high inertness were disregarded as they were not illustrative of the robot's impact on the system.

CHAPTER 4: EXPERIMENTAL RESULTS AND DISCUSSIONS

4.1 Baseline Testing

Testing the bandwidth of the network was necessary to determine the limits of the network. From the observation it was clear that the wired network performed exceptionally well compared to the wireless network, but the wireless network was a better fit. The figure below shows the maximum bandwidth data for the network (Reid, 2015).

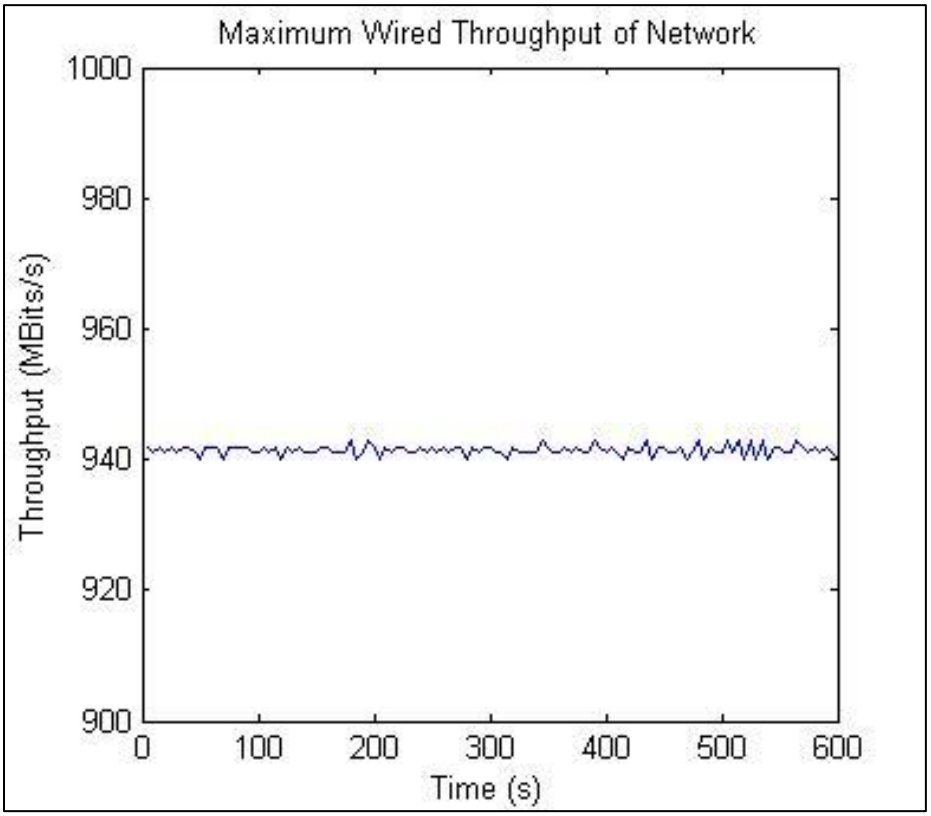

Figure 23: Maximum Wired Throughput of Network

The reason for the wireless network being a better fit for the experiment is because the wired network exceeded the specifications needed by 5.86% for the 1Gbit per second (Reid, 2015). Therefore the focus on the baseline wireless test was done for the Arlobot for the given time in the day and also key times in the day that would give a good

understanding of the experiment.

4.2.1 Daytime Test

The first baseline test (shown in Figure 24) was conducted by testing the latency between the network and the Arlobot connected to the network at a random time in the day and random time at night with the time length of 2000 seconds which is 33.33 minutes. The results are shown below.

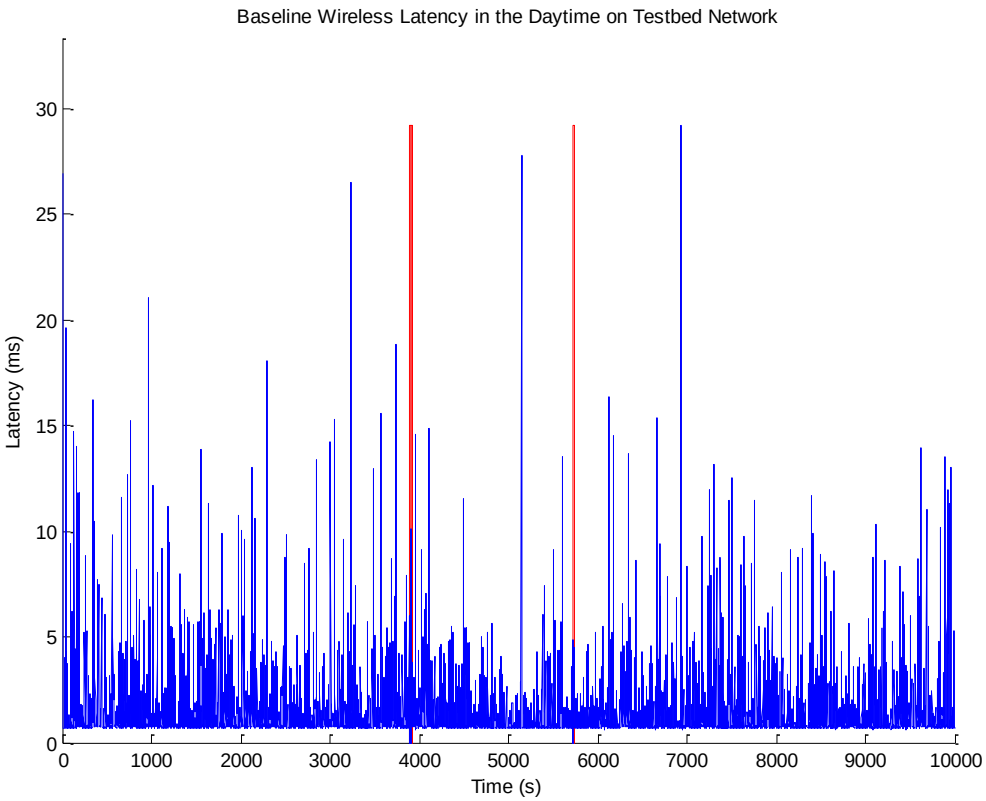

Figure 21: Baseline of Arlobot-to-Cloud Latency in the Daytime

The baseline wireless latency test gave an average of 2.557 ms, and a standard deviation of 2.969 ms for the time of 2000 seconds which is 33.33 minutes test. The packet loss was at zero, but there were a few that were not received, but this didn't affect the data transmitted. These losses are the red lines in the graph which could happen for

the same reasons as the graph changes because of different interfaces going on in the daytime.

4.2.2 Evening Test

Next, the Arlobot was used in the evening at a random time also and the baseline test was conducted (shown in Figure 25) by testing its latency between the network and the Arlobot connected to the network with the same time length of 2000 seconds which is 33.33 minutes. The Results are shown below.

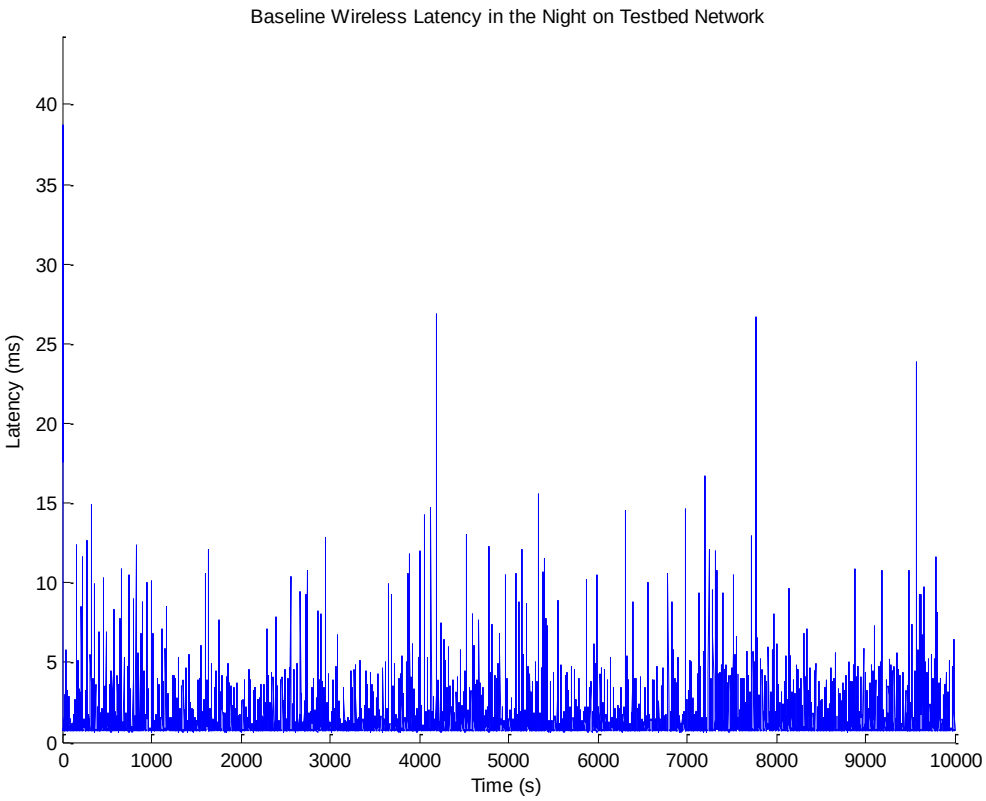

Figure 22: Baseline of Arlobot-to-Cloud Latency in the Evening

The baseline wireless latency test gave an average of 2.425 ms, a standard deviation of 3.077 ms, and zero loss in packets transmitted for the time of a 2000 seconds. There was a few jumps in the graph that were due to the many involvements of

the many wireless networks used on campus and clients devices in the building used for the experiment.

4.2.3 Comparisons of the Daytime and Evening

Table 1: ROS Day and Night Arlobot Latency Results

Arlo Robot Time Period	Average Roundtrip Time	Maximum Roundtrip Time	Minimum Roundtrip Time	Standard Deviation	Packets Dropped
	(ms)	(ms)	(ms)	(ms)	(%)
Daytime	2.557	58.474	1.236	2.969	0
Evening	2.425	77.419	1.257	3.077	0

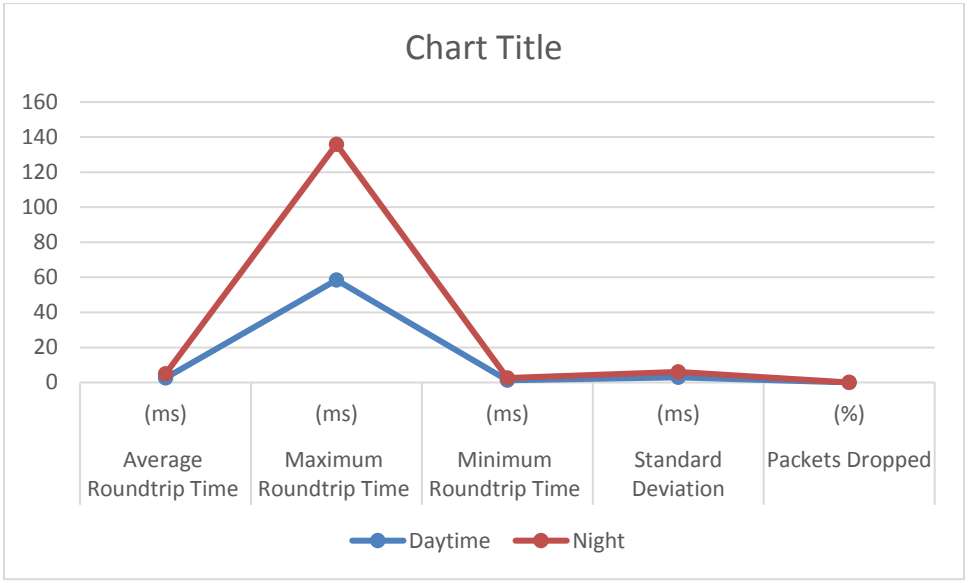

Figure 26: Graph of Day and Evening Latency Comparison

The table shown above shows that time of experimentation makes a difference for the robot, because more is needed from the network to operate a functional robot with no issues.

4.3 Daytime and Night Exact Time Observation

This test was done to show what the results could possibly change if the same time in the day and night was used for the network test. The daytime would be at 12:03pm and night 12:03am with seconds separating them. These test were done for 3 hours 06 minutes and 67 seconds which was fairly long but was chosen to get a full understanding of the two time spans.

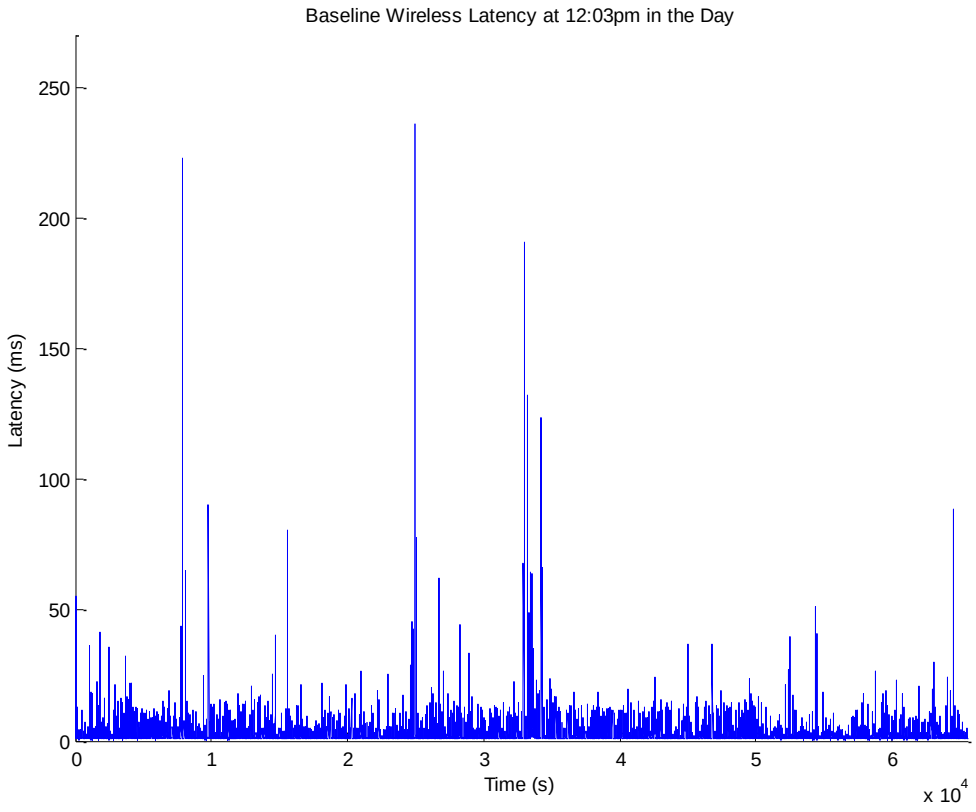

Figure 27: Baseline Wireless Latency at 12:03pm in the Day

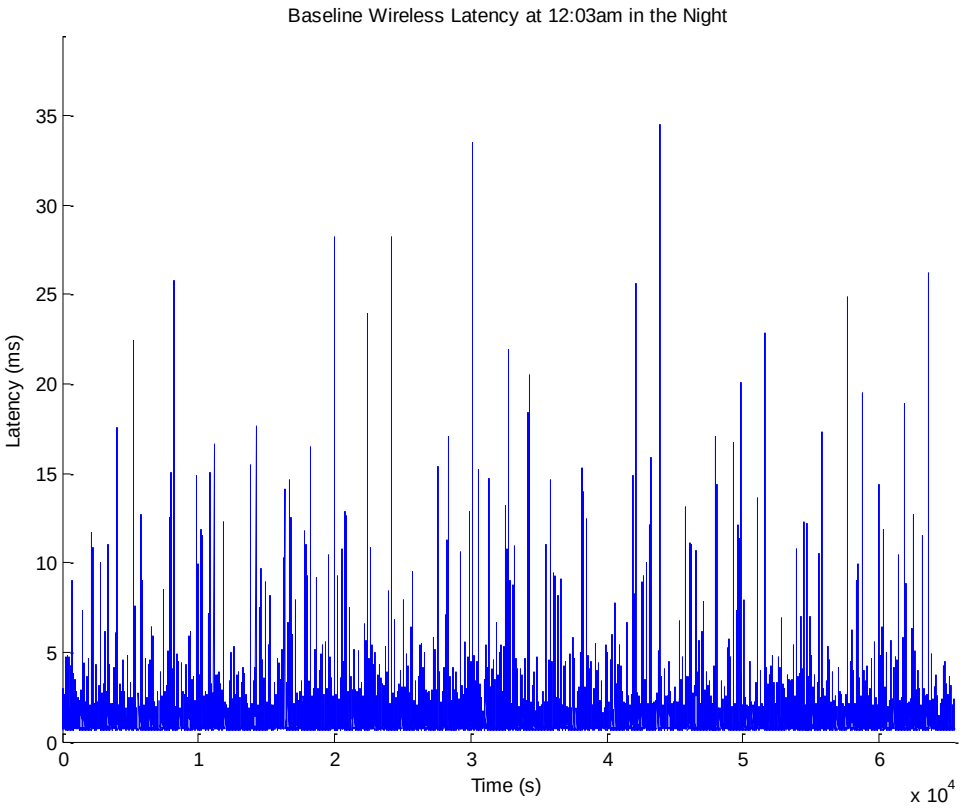

Figure 28: Baseline Wireless Latency at 12:03pm in the Day

4.4 Daytime and Night Longest Time Period Observation

For the next test done for the longest time it was done for 3 hours 33 minutes and 33 seconds. This time was done for an extremely large time period to test how well the network with the Arlobot operated. The result of this are shown below.

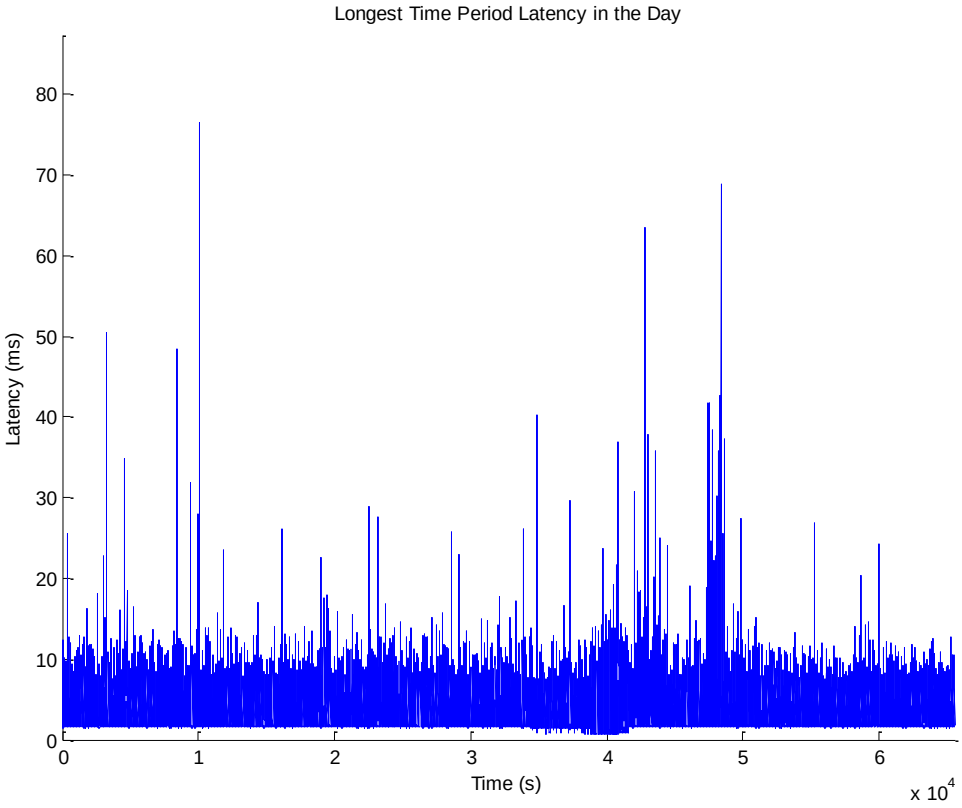

Figure 29: Longest Time Period Latency in the Day

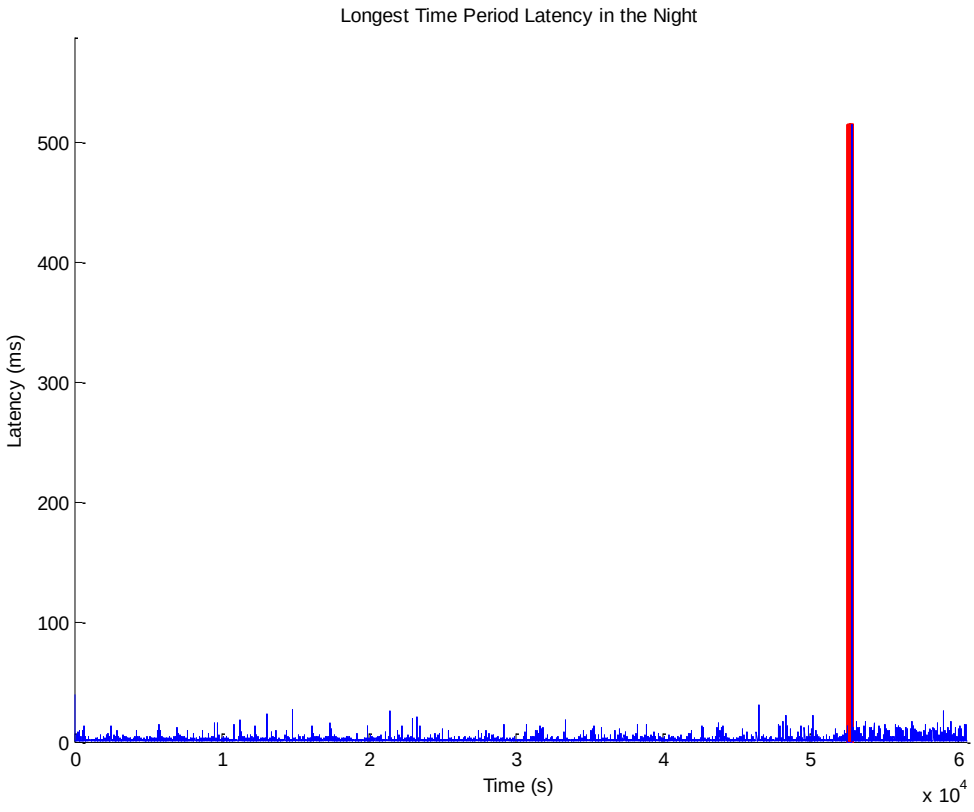

Figure 30: Longest Time Period Latency in the Night

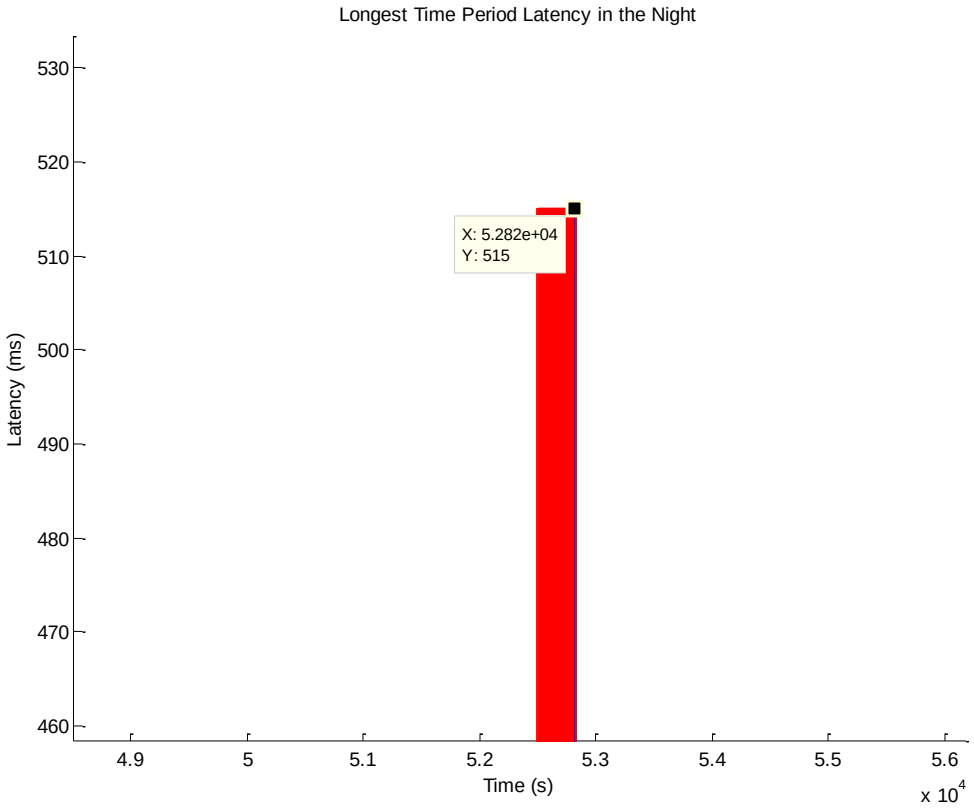

Figure 31: Longest Time Period Latency in the Night (Large packet loss)

4.5 Daytime and Night Final Time Test

As determined from the longest time periods test, a lot of data was not received so the test was done on minimal time scale. The first test was done for 2 hours, 23 minutes, and 33 seconds in the day and night. The last test was done for 1 hours, 6 minutes, and 67 seconds. Results are presented in Figures in 33-36.

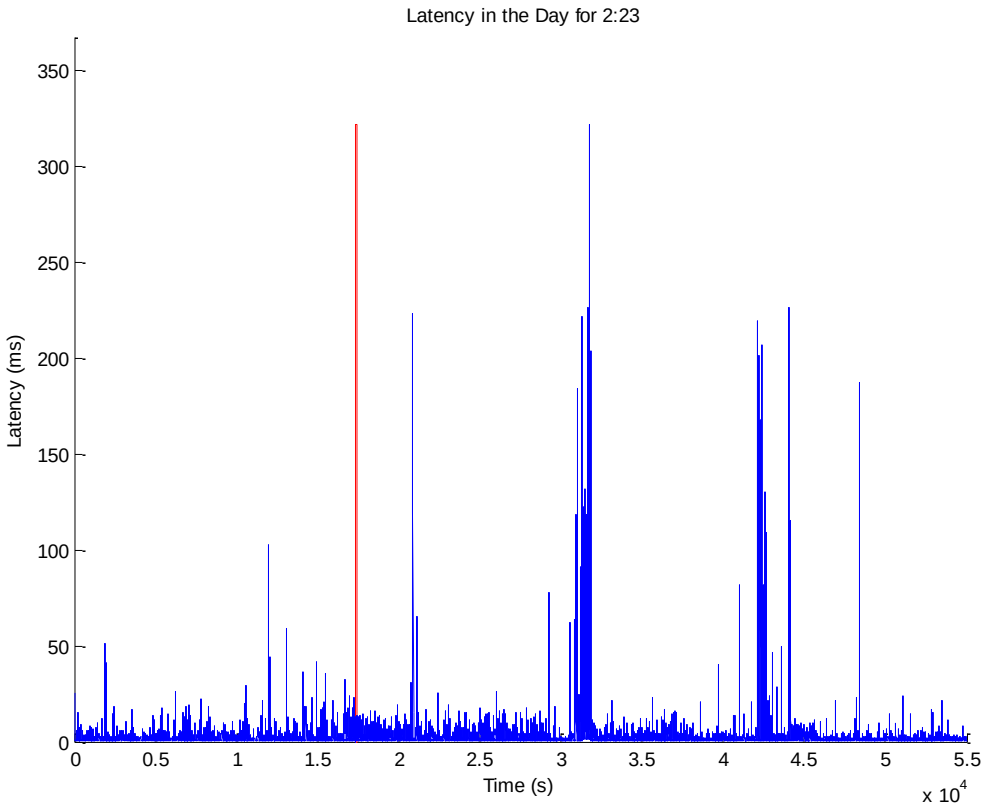

Figure 32: Latency in the Day for 2:23

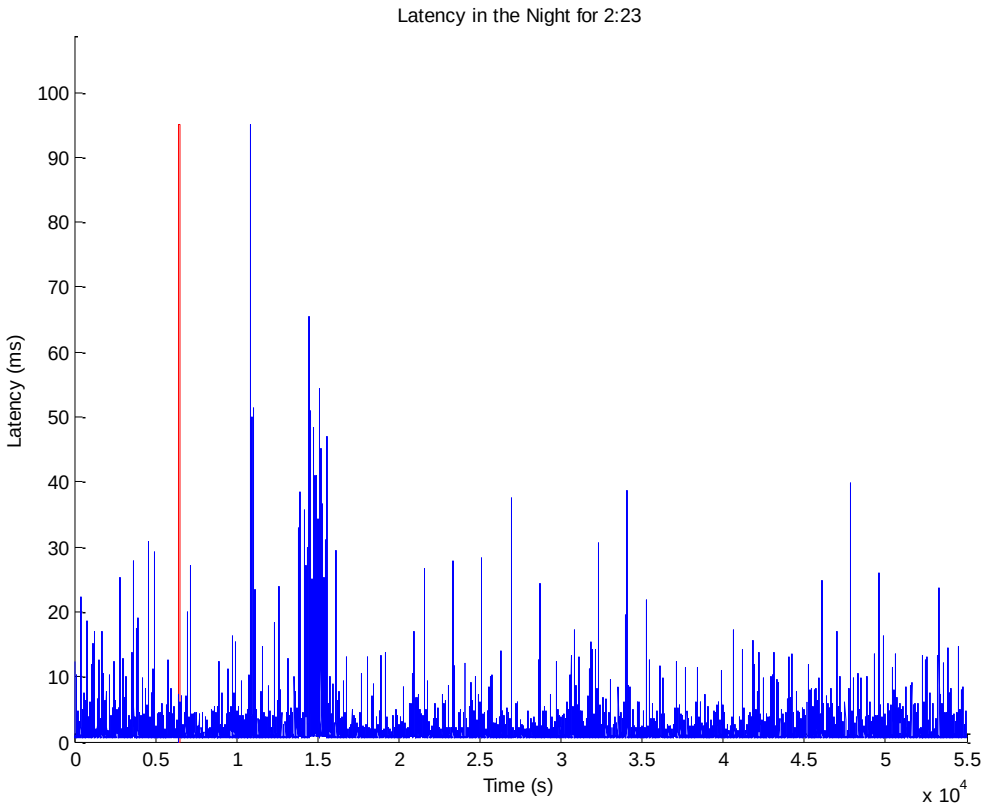

Figure 33: Latency in the Night for 2:23

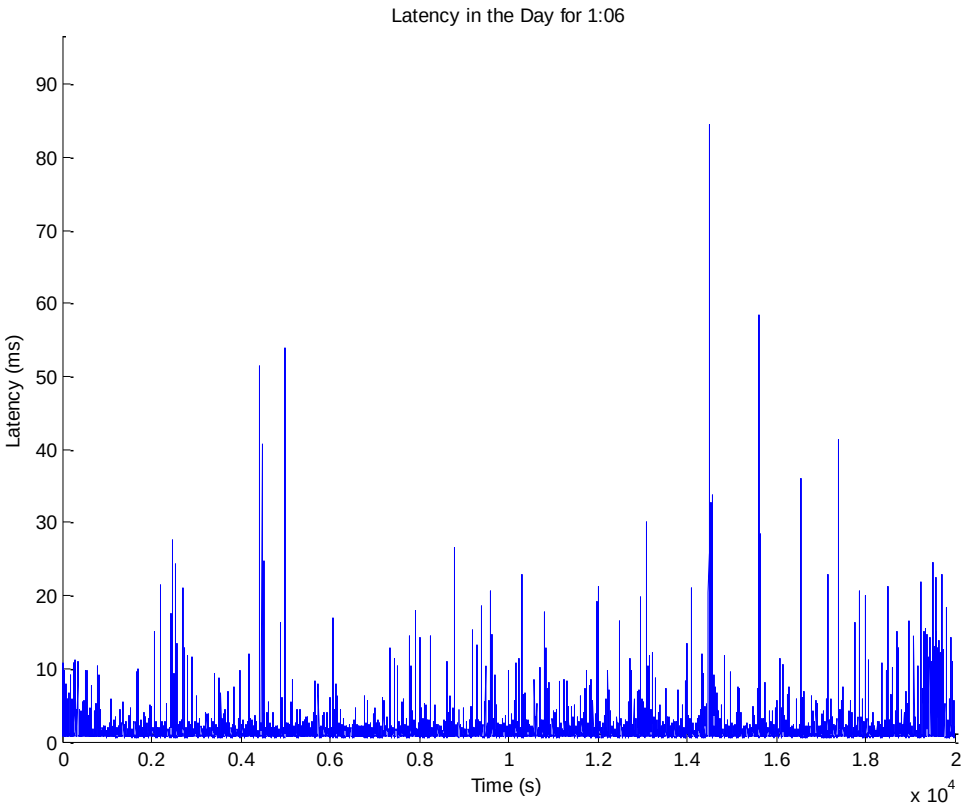

Figure 34: Latency in the Day for 1:06

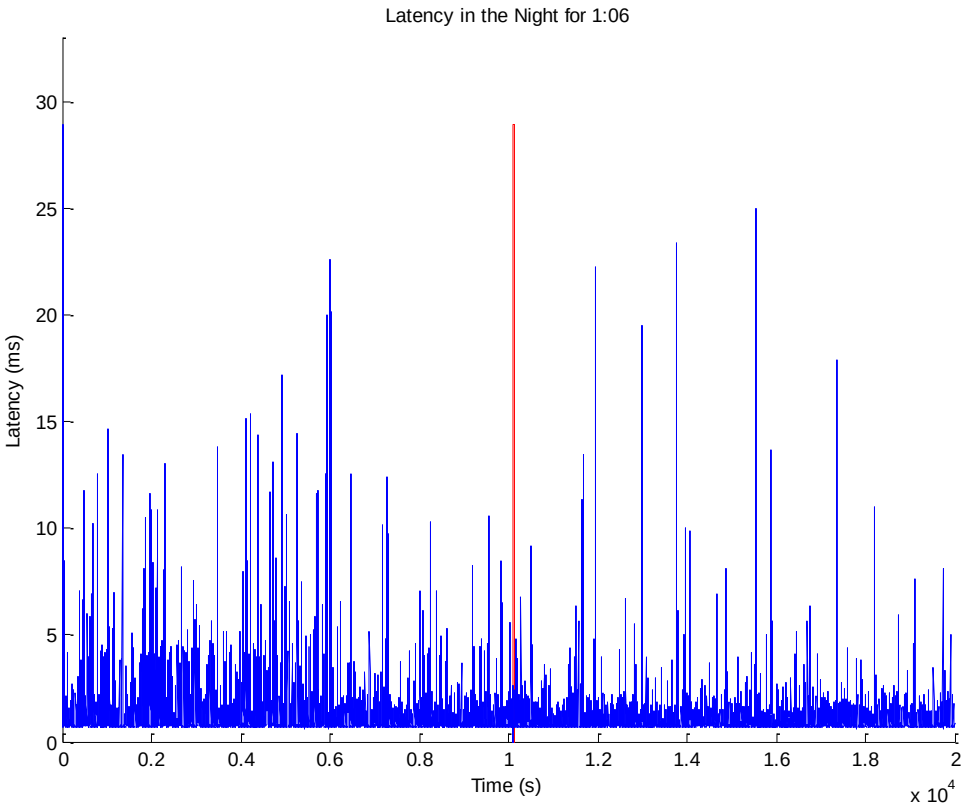

Figure 35: Latency in the Night for 1:06

4.6 Analysis of the Three Time Periods

In the table below, results and statistical data for the four times are summarized to show how each one operated in the network with its given time scale.

Table 2: ROS Night Time Latency Results

Arlo Robot Time Scale in the Night	Average Roundtrip Time	Maximum Roundtrip time	Minimum Roundtrip time	Standard Deviation	Packets Dropped
	(ms)	(ms)	(ms)	(ms)	(%)
0:33	2.425	77.419	1.257	3.077	0
1:06	1.989	57.979	1.248	1.892	0
2:23	2.196	190.542	1.206	3.386	0
3:06	1.922	117.578	1.214	1.997	0
3:33	2.053	1030.082	1.241	6.642	39

also

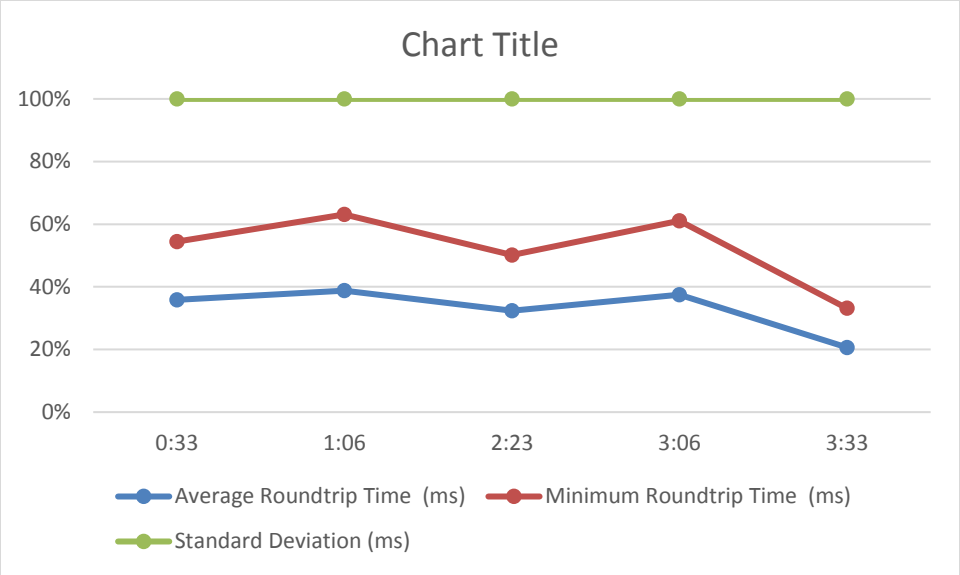

Figure 36: Minimum, Standard Deviation, and Average Graph

One clear conclusion is that these tests are better done in the night time with a short time scale. The best results came from the 1:06 time scale which wasn't too long and the results were great.

CHAPTER 5: CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusion

A mobile robotic system, Arlobot, worked well without significant issues of network latency and data integrity. The robot was integrated with a ROS enabled cloud robotic environment. The Arlobot had local processors of Arduino Mega and Raspberry Pi. The Robot Operating System was used as the main source of interaction for the robot-to-cloud in the cloud network. The Raspberry Pi provided the interface between the cloud and the robot in the network.

The network performance and data integrity were investigated when the Arlobot communicated with the cloud network. Baseline performance was established with the robot in the network. Experiments were conducted to test the network performance under different conditions.

This analysis confirmed with the thesis hypothesis that the standardized robot could work well without significant issues of network latency and data integrity in the cloud network.

5.2 Recommendations for Future Work

Due to the time constraints, the investigations were limited to the low bandwidth data. In the next phase, investigations with high-bandwidth data would be considered.

The Arlobot had a lot of room for improvement although everything was running correctly, another part that should have been viewed is possibly using another Arduino for the simple fact that the Arduino board kept burning out. In the next phase, the camera

would be utilized with more image processing tasks. The inclusion of heterogeneous group of robots in the network would also be the topic of next phase investigations.

References

Jones, R. and S. Smith. 2008. Title of An Excellent Article. Journal of Educational

Kerl, Christian. 2012. Odometer from RGB-D Cameras for Autonomous Quadcopters.

ROSWIKI, 2012 http://wiki.ros.org/ (accessed September 14, 2012).

Li, Xinrong. A low-cost wireless sensor network system using Raspberry Pi and Arduino for environmental monitoring applications. Diss. UNIVERSITY OF NORTH TEXAS, 2014.

Arlobotic Kit, 2014: https://www.parallax.com/product/arlo-robotic-platform-system (accessed, 2016).

Rogers, Phil, and A. C. Fellow. "Heterogeneous system architecture overview." Hot Chips. Vol. 25. 2013.

Arduino MEGA, 2013: http://www.arduino.cc/en/Main/arduinoBoardMega2560 (accessed November 14, 2013).

Mell, Peter, and Tim Grace. "The NIST definition of cloud computing." Communications of the ACM 53.6 (2010): 50.

Kerl, Christian. Odometer from rib-d cameras for autonomous quadcopters. Diss. Master's thesis, Technical University Munich, Germany, 2012.

Arduino. 2013. http://arduino.cc/ (accessed November 18, 2014).

ASUS Xtion Pro Live. 2015: http://www.open-electronics.org/3d-scanning-with-microsoft-kinect/ (accessed May 6, 2015)

Ferdoush, Sheikh, and Xinrong Li. "Wireless sensor network system design using Raspberry Pi and

Raspberry Pi package from Hirotaka's website. 2013: http://www.hirotakaster.com/archives/2013/01/raspberry-pi-and-openni2.php (accessed February 28, 2013)

Quigley, Morgan, et al. "ROS: an open-source Robot Operating System." ICRA workshop on open source software. Vol. 3. No. 3.2. 2009.

Quigley, Morgan, Eric Berger, and Andrew Y. Ng. "Stair: Hardware and software architecture." AAAI 2007 Robotics Workshop, Vancouver, BC. 2007.

Koren, Yoram, and Yoram Koren. Robotics for engineers. Vol. 168. New York et al: McGraw-Hill, 1985.

Markus Waibel, Michael Beetz, Javier Civera, Raffaello d'Andrea, Jos Elfring, Dorian Galvez-Lopez, Kai HÃ¤ussermann, Rob Janssen, J.M.M. Montiel, Alexander Perzylo, Bjoern Schiessle, Moritz Tenorth, Oliver Zweigle and M.J.G. (RenÃ©) Van de Molengraft. RoboEarth â€“A World Wide Web for Robots. In Robotics & Automation Magazine, IEEE, vol 18, no 2, pp 69-82, June 2011.

Zwet, J, and Strom, D. Cloud Latency Issues? Dedicated Network Connections Will Help. , Feb 2013

Cloud Robotics with ROS Titto Thomas Roll No.47 S7 A College of Engineering, Chengannur

J. Kramer and M. Scheutz, "Development environments for autonomous mobile robots: A survey," Autonomous Robots, vol. 22,no. 2, pp. 101–132, 2007.

APPENDICIES

Appendix A: Arduino code for Arlobot.

```
// Receive with start- and end-markers

const int PING_MAP[] = {
  2, 3, 4, 5, 6, 7, 8, 9, 10, 11};

const byte numChars = 11;
char receivedChars[numChars];

boolean newData = false;

void setup() {
    Serial.begin(9600);
    Serial.println("<Arduino is ready>");
}

void loop() {
    recvWithStartEndMarkers();
    if (newData == true)
      process_serial_in();

    // Process local data to send to RPi
    String serial_output = process_serial_out();

    // Send serial data to RPi
    send_serial_out(serial_output);

}

void recvWithStartEndMarkers() {
    static boolean recvInProgress = false;
    static byte ndx = 0;
    char startMarker = '<';
    char endMarker = '>';
    char rc;

    while (Serial.available() > 0 && newData == false) {
        rc = Serial.read();

        if (recvInProgress == true) {
            if (rc != endMarker) {
                receivedChars[ndx] = rc;
```

```
            ndx++;
            if (ndx >= numChars) {
               ndx = numChars - 1;
            }
         }
         else {
            receivedChars[ndx] = '\0'; // terminate the string
            recvInProgress = false;
            ndx = 0;
            newData = true;
         }
      }

      else if (rc == startMarker) {
         recvInProgress = true;
      }
   }
}

void process_serial_in()
{

  String serial_input = receivedChars;
  // serial_input is of the form "+###,+###"

  int sign_x = (serial_input[0] == '+') ? 1 : -1;
  int sign_z = (serial_input[serial_input.indexOf(',')+1] == '+') ? 1 : -1;

  String x_str = serial_input.substring(1,serial_input.indexOf(',')-1);
  String z_str = serial_input.substring(serial_input.indexOf(',')+2);

  int x = sign_x * x_str.toInt();
  int z = sign_z * z_str.toInt();

  write_motor_commands( x, z);
  //Serial.print("X value is ");

  //Serial.print(x);
  //Serial.print("| Z value is ");
  //Serial.println(z);
}

void write_motor_commands(int x, int rotz)

{
  int left_int = (x + rotz)/100*128 + 127;
```

```
  int right_int = (x - rotz)/100*128 + 127;

  byte left_byte = (byte)left_int;
  byte right_byte = (byte)right_int;

  write_left_motor(left_byte);
  write_right_motor(right_byte);
}

void write_left_motor(byte value)

{
  Serial1.write(0xFF);
  Serial1.write(1);
  Serial1.write(value);
}

void write_right_motor(byte value)
{
  Serial1.write(0xFF);
  Serial1.write(2);
  Serial1.write(value);

}

String process_serial_out()

{
  String serial_out = "<";

  for ( int i = 0; i < sizeof(PING_MAP)/sizeof(int); i++)
  {
    serial_out += String(read_ping(PING_MAP[i]));
    serial_out += '_';
  }
  serial_out [serial_out.length()-1] = '>';

  return serial_out;
}


void send_serial_out(String serial_output)
{
  Serial.println(serial_output);
}
long read_ping(int pin_number)
```

```
{
  pinMode(pin_number, OUTPUT);
  digitalWrite(pin_number, LOW);
  delayMicroseconds(3);
  digitalWrite(pin_number, HIGH);
  delayMicroseconds(10);
  digitalWrite(pin_number, LOW);

  pinMode(pin_number, INPUT);
  long result = pulseIn(pin_number, HIGH);

  delay(25);

  return result;
}
```

Appendix B: Processing code for Python.

```
#!/usr/bin/python
#=======================================
#
# EV3 ROS Driver edited by Theodore Cornelius Smith
# Filename: serial_proj.py
#
#=======================================
# Import the serial
import serial

# Import the rospy module
import rospy

# Import ROS Msgs

from std_msgs.msg import Int8

from geometry_msgs.msg import Twist

# Import ev3 driver messages
# Might use generic messages, but for now we need the header data
# to determine network integrity
# Dict for EV3 peripherals (sensors + motors)

p = {}

cs_pub = None

def init_ros_publishers():

        global cs_pub

        cs_pub =rospy.Publisher('arlobot/ping',UInt8MultiArray,queue_size=10)

def publish_ping_sensor():

        global cs_pub

        global startMarker, endMarker
        startMarker = '<'
        endMarker = '>'

        ck = ""
        x = "z" # any value that is not an end- or startMarker
```

```
        byteCount = -1 # to allow for the fact that the last increment will be one too
many

        # wait for the start character
        while  ord(x) != startMarker:
         x = ser.read()

        # save data until the end marker is found
         while ord(x) != endMarker:
           if ord(x) != startMarker:
             ck = ck + x
             byteCount += 1
           x = ser.read()


# Add sensors from Guynays code here

        cs_pub.publish(ck.split('_'))
# print.cs_pub



def subscribe_cmd_vel(data):

        global p

        potential_left = (data.linear.x - data.angular.z)* 255

        if potential_left > 255:

                potential_left = 255

        elif potential_left < -255:

                potential_left = -255

        potential_right = (data.linear.x + data.angular.z) * 255
        if potential_right > 255:

                potential_right = 255

        elif potential_right < -255:
                potential_right = -255

        str = '<' + potential_Left + ', ' + potential_Left + '>'
# print.str
```

```
        ser.write(str)

def on_shutdown():
        global cs_pub
        cs_pub.unregister()

serPort = "/dev/ttyACM0"
baudRate = 9600
ser = serial.Serial(serPort, baudRate)
print "Serial port " + serPort + " opened  Baudrate " + str(baudRate)

# Main function

def arlo_driver():

        rospy.init_node('arlo_driver',anonymous=False)
        init_ros_publishers()
        s =
rospy.Subscriber("arlobot/cmd_vel",Twist,subscribe_cmd_vel,queue_size=10)

        rate = rospy.Rate(100) # 25Hz

        while not rospy.is_shutdown():
                if s.get_num_connections() == 0:
                        publish_ping_sensor()

                rate.sleep()

if __name__ == '__main__':
        try:
                arlo_driver()
        except rospy.ROSInterruptException:
                pass
```