DEGREE PROJECT IN COMMUNICATION SYSTEMS, SECOND LEVEL
STOCKHOLM, SWEDEN 2015

Enabling Network-Aware Cloud Networked Robots with Robot Operating System

A machine learning-based approach

FREDRIK HANS NORDLUND

KTH ROYAL INSTITUTE OF TECHNOLOGY
INFORMATION AND COMMUNICATION TECHNOLOGY

Enabling Network-Aware Cloud Networked Robots with Robot Operating System

A machine learning-based approach

Fredrik Hans Nordlund

2015-03-02

Master's Thesis

Examiner and Academic adviser
Gerald Q. Maguire Jr.

Academic adviser in Japan
Shinji Shimojo

KTH Royal Institute of Technology
School of Information and Communication Technology (ICT)
Department of Communication Systems
SE-100 44 Stockholm, Sweden

Abstract

During the recent years, a new area called Cloud Networked Robotics (CNR) has evolved from conventional robotics, thanks to the increasing availability of cheap robot systems and steady improvements in the area of cloud computing. Cloud networked robots refers to robots with the ability to offload computation heavy modules to a cloud, in order to make use of storage, scalable computation power, and other functionalities enabled by a cloud such as shared knowledge between robots on a global level. However, these cloud robots face a problem with reachability and QoS of crucial modules that are offloaded to the cloud, when operating in unstable network environments. Under such conditions, the robots might lose the connection to the cloud at any moment; in worst case, leaving the robots “brain-dead”.

This thesis project proposes a machine learning-based network aware framework for a cloud robot, that can choose the most efficient module placement based on location, task, and the network condition. The proposed solution was implemented upon a cloud robot prototype based on the TurtleBot 2 robot development kit, running Robot Operating System (ROS). A continuous experiment was conducted where the cloud robot was ordered to execute a simple task in the laboratory corridor under various network conditions. The proposed solution was evaluated by comparing the results from the continuous experiment with measurements taken from the same robot, with all modules placed locally, doing the same task.

The results show that the proposed framework can potentially decrease the battery consumption by 10% while improving the efficiency of the task by 2.4 seconds (2.8%). However, there is an inherent bottleneck in the proposed solution where each new robot would need 2 months to accumulate enough data for the training set, in order to show good performance. The proposed solution can potentially benefit the area of CNR if connected and integrated with a shared-knowledge platform which can enable new robots to skip the training phase, by downloading the existing knowledge from the cloud.

Keywords

CNR, ROS, network awareness, cloud, open-source, TurtleBot.

Sammanfattning

Under de senaste åren har ett nytt forskningsområde kallat Cloud Networked Robotics (CNR) växt fram inom den konventionella robottekniken, tack vare den ökade tillgången på billiga robotsystem och stadiga framsteg inom området cloud computing. Molnrobotar syftar på robotar med förmågan att flytta resurstunga moduler till ett moln för att ta del av lagringskapaciteten, den skalbara processorkraften och andra tjänster som ett moln kan tillhandahålla, t.ex. en kunskapsdatabas för robotar över hela världen. Det finns dock ett problem med dessa sorters robotar gällande nåbarhet och QoS för kritiska moduler placerade på ett moln, när dessa robotar verkar i instabila nätverksmiljöer. I ett sådant scenario kan robotarna när som helst förlora anslutningen till molnet, vilket i värsta fall lämnar robotarna hjärndöda.

Den här rapporten föreslår en maskininlärningsbaserad nätverksmedveten ramverkslösning för en molnrobot, som kan välja de mest effektiva modulplaceringarna baserat på robotens position, den givna uppgiften och de rådande nätverksförhållanderna. Ramverkslösningen implementerades på en molnrobotsprototyp, baserad på ett robot development kit kallat TurtleBot 2, som använder sig av ett middleware som heter Robot Operating System (ROS). Ett fortskridande experiment utfördes där molnroboten fick i uppgift att utföra ett enkelt uppdrag i laboratoriets korridor, under varierande nätverksförhållanden. Ramverkslösningen utvärderades genom att jämföra resultaten från det fortskridrande experimentet med mätningar som gjordes med samma robot som utförde samma uppgift, fast med alla moduler placerade lokalt på roboten.

Resultaten visar att den föreslagna ramverkslösningen kan potentiellt minska batterikonsumptionen med 10%, samtidigt som tiden för att utföra en uppgift kan minskas med 2.4 sekunder (2.8%). Däremot uppstår en flaskhals i framtagna lösningen där varje ny robot kräver 2 månader för att samla ihop nog med data för att maskinilärningsalgoritmen ska visa bra prestanda. Den förlsagna lösningen kan dock vara fördelaktig för CNR om man integrerar den med en kunskapsdatabas för robotar, som kan möjliggöra för varje ny robot att kringå den 2 månader långa träningsperioden, genom att ladda ner existerande kunskap från molnet.

Nyckelord

CNR, ROS, nätverksmedvetenhet, moln, öppen källkod, TurtleBot.

Acknowledgments

This report is a result of a Master's thesis project at the Applied Information Systems Research Division (Shimojo Laboratory) at the Cybermedia Center, Osaka University, Japan, during the period of November 2013 to the end of August 2014.

I would like to thank the following persons for helping me to make this thesis project a success:

- Professor Gerald Q. Maguire Jr., for supporting me by providing an excellent template for a thesis report, as well as having the patience to sit through several meetings to discuss the thesis project and for several times taking the time to write feedback on the drafts of the report. I would not have been able to finish this report without your help.
- Professor Shinji Shimojo, for believing in me and for the many hours of support and engaging me in discussions about the project, as well as giving me the inspiration to strive to always do better during my thesis project.
- Professor Yuichi Teranishi, for spending many hours in meetings about the thesis project as well as helping with the writing of two conference papers that led to this thesis project. Without the many last-minute edits, this thesis would surely not turn out the way it has.
- Professor Manabu Higashida, for always taking the time to engage in a discussion or answer questions whenever I had something on my mind. The many advices I received during the year made me not only a better researcher, but a better person as well.

I would also like to give a special thanks to my girlfriend at the time, Satoko Miki, and her family for showing me endless support throughout the year. Your hospitality and kindness gave me the power to always do my best during the year.

Another special thanks to Anton Friberg and Erik Ledin for helping me with revising the report. The many comments and insights greatly helped in shaping the report.

Stockholm, February 2015
Fredrik Hans Nordlund

Table of contents

List of Figures

List of Tables

List of acronyms and abbreviations

AIBO	Artificial Intelligence roBOt
AMCL	Adaptive Monte Carlo Localization
API	Application Programming Interface
A_{bw}	Available bandwidth
CB_{task}	Control value of battery change
C_{bw}	Current bandwidth
C_l	Current latency
CNR	Cloud Networked Robotics
CPU	Central Processing Unit
CSAIL	Computer Science and Artificial Intelligence Laboratory
CT_{task}	Control value of task completion time
C-Ma	Client-Master
C-Mo	Client-Monitor
DARPA	(United States of America) Defense Advanced Research Project Agency
DC	Direct current
EC2	Amazon's Elastic Compute Cloud
GB	Gigabyte
GHz	Gigahertz
GUI	Graphical User Interface
Hz	Hertz
IC	Investigation cell
KB	Kilobyte
KLD-sampling	Kullback-Leibler divergence-sampling
LEGO®	A trademark of Lego A/S
LTS	Long Term Support
M2C	Machine-To-Cloud
M2M	Machine-To-Machine
MD(x,y,z)	Migration Decision tripple
MFG	Measurement Feedback Goal
MIT	Massachusetts Institute of Technology
ML	Machine Learning
MM	Multi-master
M-M	Master-Monitor
OS	Operating System
QoS	Quality of Service
P(x,y)	Point at coordinates x and y
$R(x_m,y_m,z_m,w_m)$	Quadruple of row m
RAM	Random Access Memory
ROS	Robot Operating System
RPC	Remote procedure call
RT-middleware	Real-time middleware
RTT	Round-trip time
R_{bw}	Bandwidth ratio
SLAM	Simultaneous Localization and Mapping
SM	Single-master
STAIR	STanford AI Robot
Std.Dev.	Standard Deviation
Std.Error	Standard Error

RPC	Remote Procedure Call
TCP	Transport Control Protocol
T_{task}	Task execution time
VM	Virtual Machine
V_r	Validity ratio
YAML	YAML Ain't Markup Language

1 Introduction

Until the end of the 1980's robots were mainly thought of as being big, expensive, and not accessible to the average person. This started to change with LEGO® introducing educational products [1] and when researchers started to focus on how to build lots of smaller inexpensive robots, rather than one large expensive robot [2]. Subsequently companies have developed robots for use by the general consumer, e.g. Sony's Artificial Intelligence robot pet (AIBO) that was envisioned as a form of robot-based entertainment [3] [4] and iRobot's robotic vacuum cleaner, *Roomba*, that was first put on the market in 2002 [5]. In more recent years, the accessibility of robots to the average person has greatly increased with the commercialization of robot development kits such as the TurtleBot™ 2 from *Clearpath Robotics* [6], the iRobot Create® 2 [7], and the Revolution JD [8]. However, a general problem with the development of robots is that robots are very complex systems. An average person or a software developer is unlikely to know all the details about how to build drivers for sensors and actuators and how to do low-level programming. There have been attempts to address this problem by providing high-level interfaces to sensors and actuators in the form of robotics middleware, e.g. real-time middleware (RT-middleware) [9] and the *Robot Operating System* (ROS) [10] [11]. In 2002, researchers started to try to overcome the limitations of stand-alone robots by connecting robots, sensors, and humans through networks, establishing the research area of *Networked Robotics* [12].

At the same time as robots were becoming increasingly accessible to general consumer, there were steady improvements in the area of Grid Computing, with the help of virtualization technology [13], to address the problem of reliability and Quality of Service (QoS) [14] [15]. This later resulted in Grid Computing evolving into what is now known as *Cloud Computing* [16] [17] [18]. In recent years, Cloud Computing has been brought to the public market in the form of popular services such as Amazon's Elastic Compute Cloud (Amazon EC2) [19], Apple's iCloud [20], and Microsoft Cloud [21]. By combining the increasingly accessible networked robot technology with the powerful technology of Cloud Computing, a new technology has been enabled which is called *Cloud Networked Robotics* (CNR) [12] [22].

1.1 Background

According to Hu, Tay and Wen, CNR improves upon the area of *Networked Robotics* in three major aspects: the "ability to offload computation-intensive tasks to the cloud", enabling "access to vast amounts of data", and by enabling "access to shared knowledge and new skills" [23]. Some of the proposed uses of CNR are improving robot grasping, navigation, daily services (e.g. wash your face, grocery shopping, medical treatment, housework, and much more), exploration of dangerous areas, intelligent transportation, smart home, education, defense, and much more [23] [12]. CNR can also provide robots that have limited resources with advanced services, e.g. tracking of dynamic objects [24], distributed sensing [25], localization services, scene recognition, robotic manipulation, and shared knowledge between robots [26].

However, CNR is not limited to services that are meant for cloud robots. A general overview of the different elements of CNR and their relationships can be seen in Figure 1-1. Some of the main concepts in CNR are sensors and actuators on the robots, *Machine-To-Machine* (M2M) communication between robots, *Machine-To-Cloud* (M2C) communication between robots and the cloud, and cloud computing. CNR also incorporates external services that can make use of the data that these cloud robots collect,

such as smartphone apps and web-services. Cloud robots can make use of data that comes from external sensor networks that are connected to the cloud. An example could be service cloud robots in a supermarket that acquire the traffic information. The cloud robots could notice that it is unlikely that there will be many customers on a certain day, hence some of the cloud robots would turn off in order to save energy.

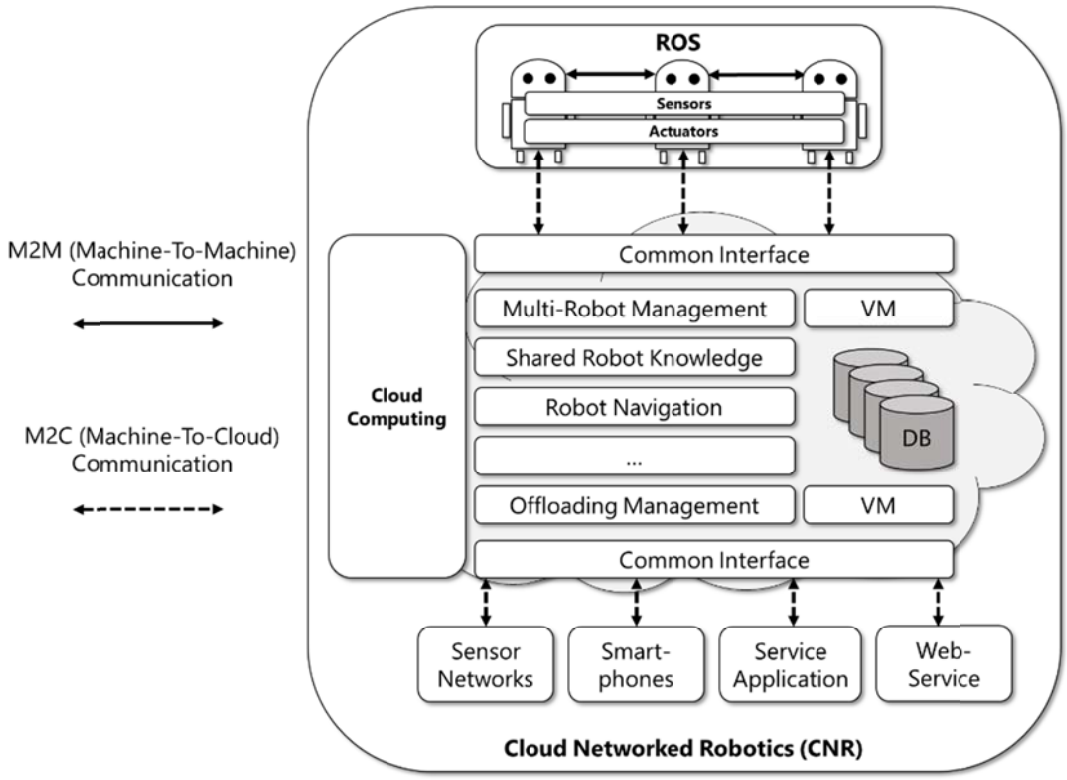

Figure 1-1: The different element of CNR and the relationships between the elements.

For the last few years, CNR has been a hot topic for researchers. However, most research has been about the different areas that could benefit from this new research area. For example, Turnbull and Samanta developed a small scale cloud infrastructure that could handle the formation control algorithms and compute loads of a vision acquisition system for their robots [27]. CNR has also been successfully applied to explore the possibilities of parallelizing robotic algorithms on the cloud [28], remote motor control of quadrotor teams constructing cubic structure [29], and much more [30] [31] [32]. Most, if not all, of this research builds upon the assumption of stable network environments. However, this assumption does not reflect real-life situations and places that are affected by unstable network environments, where the connection can instantly change from good to bad.

1.2 Problem definition

Much of the research concerning CNR assumes stable network environments where the connection between the robot and cloud is unlikely to be lost. In these scenarios, the cloud can be seen as an external brain for the robots [27]. However, many of the areas where CNR could be applied, such as exploration of dangerous areas, customer support in a super market, and data center monitoring (to name a few), may not have ideal network environments. Such unstable network environments might unplug the brain from the

robot, leaving it brain dead [27]. In the case of quadrotor teams [29], this would mean that the cloud would not be able to control the motors of the quadrotor teams, making them crash instead of executing their assigned tasks. Termination of a robotic service by accident, such as a robot losing connectivity to the cloud, must be prevented or addressed in some way since cloud networked robots are likely to become a critical part of our lives [12].

A serious problem for mobile devices is the energy problems of today's handheld devices. Even though new chemicals and alternative battery technology have been proposed, the energy concerns are unlikely to disappear [33]. This problem also occurs for CNR, since cloud robots are also likely to see an increase in energy-hungry applications and services as the area matures. Even though there have been many proposals for docking stations and battery exchange systems ([34], [35], to name a few), where cloud robots can go to recharge (or exchange) their batteries, it is still preferred that a robot will continue to provide a service or execute a task for as long as possible before having to recharge its batteries.

As to summarize, the problem with CNR is that when operated in an unstable network environment, the robot might lose connectivity to the cloud. How do you choose the best module placements so that (to the best effort) the robot will not be left brain dead, at the same time as striving to achieve long battery time by having modules on the cloud for as long as possible?

1.3 Purpose

The purpose of this thesis project is to gain a better understanding of the problems inherent in current cloud robot platforms when faced with unstable network environments. Furthermore, upon identifying the problems, some recommendations for future developers of CNR-technology are to be made in order to enable this technology to progress even further. These recommendations are given in Section 6.1.

1.4 Goals

As a consequence of the problem described in Section 1.2, the goal of this thesis project is to develop a network-aware framework that can be used in a CNR-system operating in an unstable network environment. The network-aware framework should satisfy two sub-goals in order to address the earlier mentioned problems:

1. Aim to increase the total battery time of the cloud robot.
2. Make efficient module placements choices depending on the task, place, and network condition.

A proposed machine learning-based network-aware framework solution, aiming to reach these two sub-goals, was developed upon a cloud robot prototype and is presented in Section 4.2. The results from an evaluation of the proposed solution, showing promising results, are presented in Section 5.1.

1.5 Research Methodology

This thesis project made use of an empirical quantitative research methodology to reach its goals. This approach was chosen since the author wanted to investigate the problem in a real environment rather than in a computer simulated environment. Initially an inductive

research approach was used to develop a cloud robot prototype that was observed when performing simple tasks. A theory of the current problems with CNR platforms was generated from these observations, and a deductive research approach was used to confirm these theories. A solution was implemented in order to tackle the problem specified in Section 1.2 and satisfy the goals described in Section 1.4. The solution was tested and evaluated to create some recommendations for future developers and researchers of the same area.

1.6 Delimitations

Since the whole area of CNR is too large to cover, this thesis project will focus on a specific part of CNR. An overview of the delimitations of this project can be seen in Figure 1-2. This project will focus on the *Machine-To-Cloud* (M2C) communications rather than *Machine-To-Machine* (M2M) communications (shown as a dashed line), i.e. the focus here will be on single-robot system *without* multi-robot coordination. Elements connected to the cloud such as external sensor networks, smartphones, external service applications, and web-services are also out of the scope of this project (this is indicated by their dashed outlines and gray shading).

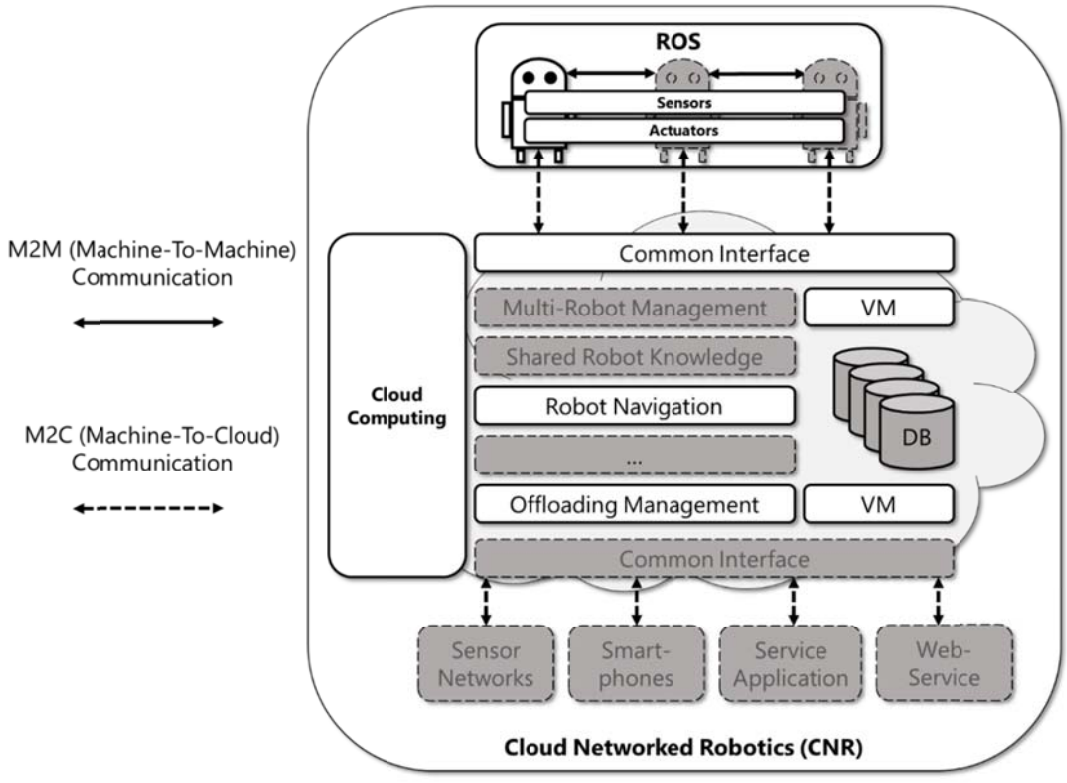

Figure 1-2: The delimitations of this project, excluded areas are drawn with dashed lines and filled with gray.

1.7 Structure of the thesis

Chapter 2 presents relevant background information about CNR, ROS, network-awareness, and related works. Chapter 3 presents the methodology and method used to develop and evaluate a solution to the problem. Chapter 4 presents the cloud robot prototype to be used in the specified use-case, as well as the design decisions made when

designing a network-awareness framework for this prototype. Chapter 5 presents the major results from the experiments and concludes with a discussion of these results. Lastly, Chapter 6 presents a conclusion, future work, and reflections on this thesis project.

2 Background

This chapter will provide the reader with sufficient background information on the different main concepts and elements that was used in this project. Section 2.1 will cover the background of CNR and give a deeper understanding of the concept and benefits of CNR. Section 2.2 will present the robotics middleware chosen for this project, ROS, and provide necessary background concerning the publish-subscribe messaging paradigm. Section 2.3 will present the TurtleBot 2 robot development kit that was used in this project to develop a cloud robot prototype (presented in Section 4.1), as well as to provide a deeper understanding of how the robot uses a *Simultaneous Localization and Mapping* (SLAM) [36] algorithm to navigate spaces. Section 2.4 will present several related works and provide an overview of what has already been done in this research area. Lastly, Section 2.5 will provide a summary of the whole chapter.

2.1 Cloud networked robotics

This section will give a detailed background on, as well as state the benefits of, the area of CNR. Different proposed cloud robotic computing architectures will be discussed and possible application areas will be presented.

2.1.1 Background of cloud networked robotics

In 2001, researchers started to do preliminary work on Internet-based tele-operated robots. They envisioned a system where robots could work in a cooperative way as well as interact with humans, all while being connected over a network. The name *Networked Robots* (or *Networked Robotics*) was coined in 2004 and refers to a system with five distinct elements being present: physical embodiment of a robot, autonomous capabilities, network-based cooperation, environment sensors and actuators, and human-robot interaction. [37]

The area of *Networked Robotics* is said to transcend stand-alone robots in conventional robotics by having an interrelation between robots, environmental sensors, and humans [37]. It is very difficult to create a stand-alone robot that can understand context and have the ability to communicate with people. The reason for this is that such a robot would need to understand the current situation, relate that to a relevant past, at the same time while performing necessary actions [38]. Even though *Networked Robotics* has seen huge success in several areas, an inherent problem in the architecture, which was not solved when evolving from stand-alone robotics, is the limitation of non-elastic resources, since all computations are conducted onboard the robots. This limitation makes for several constraints in resources, information and learning, and communication. *Cloud Networked Robotics* (CNR), first coined by James Kuffner in 2010 [39], is proposed to overcome these limitations by making use of the elastic-resource capabilities of cloud technology [12] [23]. The main benefits of CNR are mainly three aspects:

Offloading of computation heavy tasks to a cloud [27] [40]: Since the robots no longer require computational heavy tasks to be performed by the onboard hardware, the robots can be equipped with lighter hardware which results in lower power consumption and decreases the cost of the robots. Offloading tasks to a cloud may also improve performance when executing a task.

Access to a vast amount of data which does not need to created nor maintained by the individual robots [23] [31]: Cloud robots can receive information such as traffic, weather, and personal information (to name but a few), that may or may not be originally intended for the cloud robots. This enables the creation of extremely powerful robotic systems where robots could potentially predict behavior, future events, and understand context, thanks to the application of Big Data Analysis.

The possibility of having access to shared knowledge and learning new skills [26] [32]: Robots can be made with minimum software and download the necessary application and skills from shared robot databases. If a robot learns something new, this can be uploaded to the shared "brain" in order to make a world-wide community of cloud robots even smarter.

2.1.2 Machine-to-machine and machine-to-cloud communication

When talking about CNR, it is often important to distinguish between *machine-to-machine* (M2M) and *machine-to-cloud* (M2C) communications. The difference between M2M and M2C communications can be seen in Figure 2-1. M2M communications refers to robots talking to other robots over a wireless- or wired medium where these robots can be connected to each other in a structured or *ad hoc* fashion. The benefits are that when a robot that cannot establish a direct connection to the cloud (this can happen because of several reasons, such as the robot being too far away from a wireless access point) it may be able to connect to other nearby robots, then in an *ad hoc* fashion utilize these other robots to forward its messages to a robot that can communicate with a wireless access point. Robots can also share information in order to make use of collaborative decision making, which can be useful in various robot-related communications [23].

M2C communications refers to communication between robots and the cloud. This includes robots making use of all the benefits specified above in Section 2.1.1, such as computation offloading, vast amount of data, and possibilities of shared knowledge. The cloud also provides a large volume of storage, orders of magnitude larger than possible at individual robots.

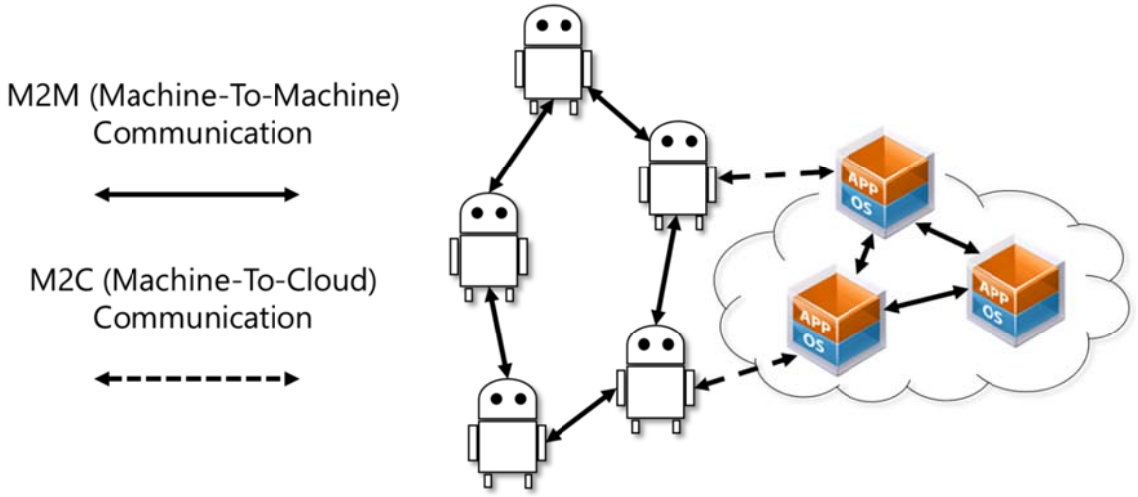

Figure 2-1: The difference between machine-to-machine communication and machine-to-cloud communication.

2.1.3 Cloud robotic computing architectures

Hu, Tay, and Wen [23], recognizes that there are mainly three system architectures that can make use of cloud computing: a *peer-based model*, a *proxy-based model*, and a *clone-based model*. An overview of the three elastic computing models can be seen in Figure 2-2.

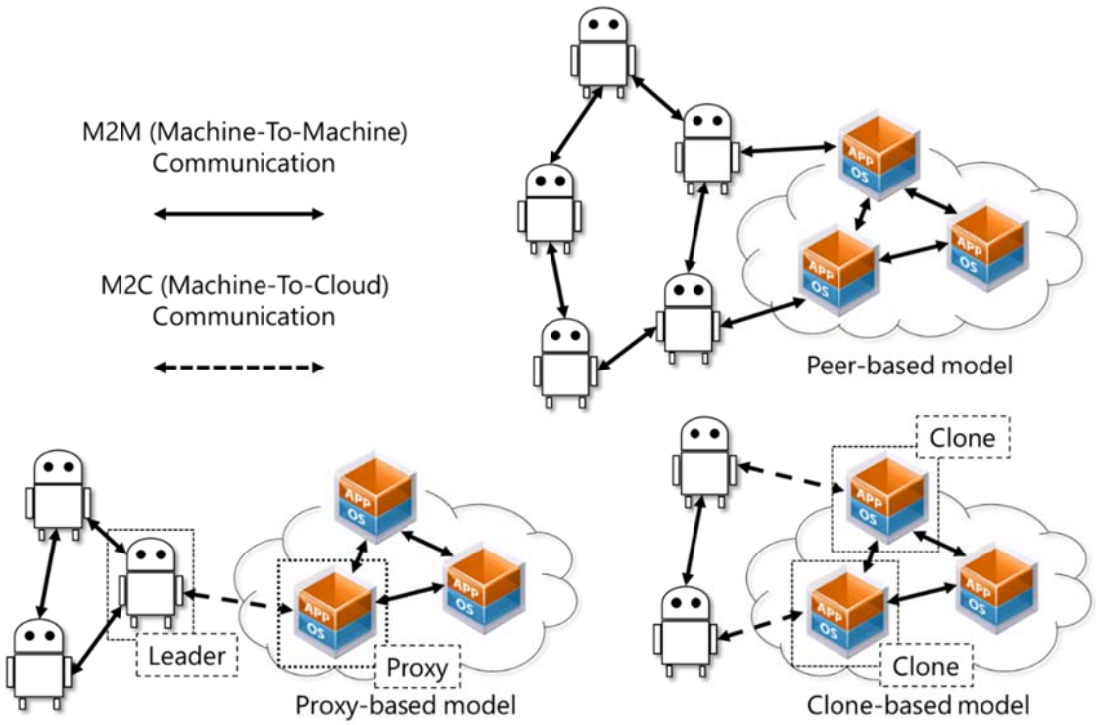

Figure 2-2: Elastic computing models for cloud robotics (Adapted from [23]).

Peer-based model In a peer-based model, each robots is considered as an individual computing unit creating a fully distributed computing mesh. There is no M2C communications present in this model since every robot works as an extension to the cloud architecture. This makes this model very well suited for systems that contain nodes with high mobility, since a Virtual Machine (VM) can be instantiated anywhere in the infrastructure.

Proxy-based model In a proxy-based model, there exists one robot leader that is assigned to act as a gateway to the cloud. This leader communicates with a specific VM proxy located in the cloud. The proxy can delegate computation tasks and storage between the nodes on the cloud. Since the robots' network is limited to a single point of connectivity to the cloud, this architecture is not very robust from a robot-cloud connectivity point of view. The leader-proxy connection might also be a bottleneck if the robot network runs applications that transfer a lot of data between the cloud and the robots. The robot leader will also most likely need to be equipped with more powerful hardware and larger battery in order to provide the robot network with connectivity to the cloud.

Clone-based model In a clone-based model, every robot has its own connection to a clone of itself running on a VM on the cloud. If the robot needs to offload some modules in order to, for example, save battery, it can order the clone to take care of the computation load. This model is very robust against connectivity issues since there are several connections to the cloud. The model works best for systems where the robots are likely to work in a certain area, since too high mobility might take the robots physically too far away from the VM. This might result in the need to migrate the VM to another data center in order to satisfy the QoS of the applications.

2.1.4 Possible application areas

In addition to the possible application areas presented in Section 1.1, CNR has also been proposed to improve traditional robot systems, such as Google's autonomous driving project [41] and a new approach to logistics and warehouse automation by Kiwa Systems [42]. Other interesting application areas of CNR are improving surgical robots [43], elderly-care [12], a museum guide [37], transportation of people [37], and agriculture [44].

2.2 Robot operating system

This section describes the robot middleware that was chosen for this thesis project. ROS is a meta-operating system for robot platforms. It is open-source* and provides hardware abstractions, inter-process message-passing, package management, low-level device control, and much more [11]. ROS started as a collaboration project between the Personal Robots Program at Willow Garage† and the STanford AI Robot (STAIR) project at Stanford University‡. Researchers at both projects wanted to develop a robot framework with emphasis on large-scale integrative robotics research in order to tackle the problem of developing applications for ever increasingly complex robot systems. They believed that ROS should be developed with five philosophical goals in mind, in order to tackle the problems: *peer-to-peer*, *tools-based*, *multi-lingual*, *thin*, and *free and open-source* [10].

ROS has been released in a number of different distributions since its first release in March 2010. The latest distribution of ROS is *ROS Indigo Igloo*, which was released in July 2014 and is the 8th generation of ROS. A new distribution of ROS, the 9th generation *ROS Jade Turtle*, is expected to be released in May 2015 [45].

2.2.1 ROS file system

The ROS file system was developed in order to enhance collaboration between researchers. The file system consists of four main elements: *packages*, *manifests*, *message types*, and *service types*. File system management in ROS is done by using a tool called *catkin*, which is a low-level build system. The four elements of the ROS file system are:

Packages Packages are a group of folders that contain a ROS project or library. A package may contain datasets, libraries, and ROS runtime processes. A package is an abstraction for organizing

* The source code is available at http://wiki.ros.org/ROS/.
† http://pr.willowgarage.com/
‡ http://stair.stanford.edu/

software in ROS.

Manifests Manifests are xml files that provide metadata about a package. A manifest also contains license information and dependencies.

Message types Message types define the data structures for messages in ROS. They are stored in "package/msg/SpecificMessageType.msg".

Service types Service types define the data structures for services in ROS. They are stored in "package/srv/SpecificServiceType.srv".

An overview of a typical workspace created using *catkin* is shown in Figure 2-3. A *catkin workspace* usually contains a top-level *CMakeLists.txt* file, which is used to link together the file system for indexing. As defined in [46], a *catkin workspace* can contain up to four different spaces: a *source-space*, a *build-space*, a *development- space*, and an *install-space*, each with their own purpose. In the example, the *catkin workspace* contains several different packages. A package contains a *CMakeList.txt* (in order to index the package, specify build locations, and add dependencies), a *package.xml* (specifies dependencies and package licenses), and four subfolders called *include*, *src*, *srv*, and *msg*. The "include" folder contains include libraries that are needed during runtime. The "src" folder contains the original source files of the application contained in the package, in the case of this thesis project the file extension was often "*.cpp*" since most of the applications were written in the C++ programming language. The "srv" folder contains the data structures for any service messages used in the application. Lastly, the "msg" folder contains the data structure for any topic message used in the application.

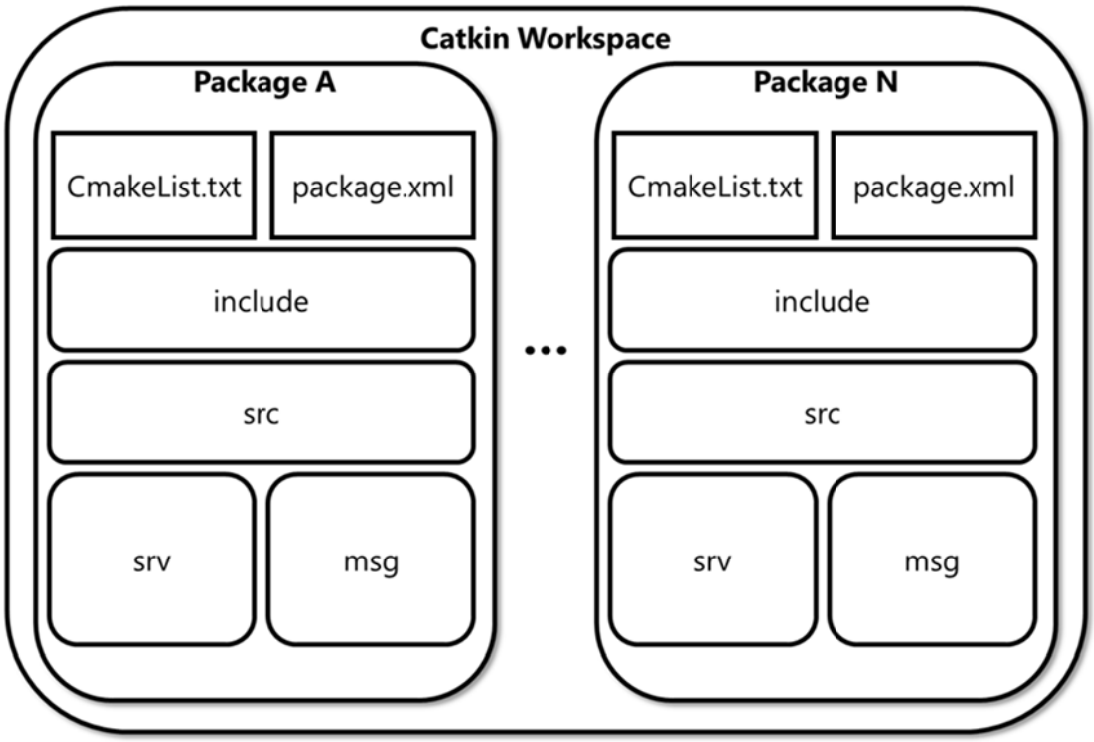

Figure 2-3: A typical catkin workspace containing catkin projects and necessary project files used when compiling and during runtime.

2.2.2 Publish-subscribe messaging paradigm

The publish-subscribe messaging paradigm is a software architecture used to propagate information through a distributed system in a *many-to-many* communication fashion [47]. Entities that provide information are called *publishers* and entities that tap into this information are called *subscribers*. The key characteristics of this messaging paradigm is that a *publisher* does not need to know the address of a particular *subscriber* nor does it need to know the number of *subscribers* to the information. This introduces better scalability in the network as well enabling a more dynamic network topology. The publish-subscribe messaging paradigm can be divided into two main types: *content-based* and *topic-based*.

In a *content-based* publish-subscribe messaging paradigm, *subscribers* specify special attributes sought for in published content. As a consequence, a subscriber will only receive messages that contains these specific attributes. In a *topic-based* model, there exist predefined subjects (also known as *topics*) to which *publishers* may publish information and to which *subscribers* may subscribe to in order to receive information. Topics may be handled by some kind of message broker that controls what *publisher* can publish to which *topic* and that knows which *subscribers* are currently subscribed to each *topic*. Figure 2-4 shows an example of the *topic-based* publish-subscribe messaging paradigm with *publishers* publishing messages to three different topics handled by a message broker. The *subscribers* subscribe to their choice *topic* (a *subscriber* can subscribe to several *topics* at a time) and the information published by the *publishers* will reach the correct *subscriber* even though either one is directly aware of the existence of the other.

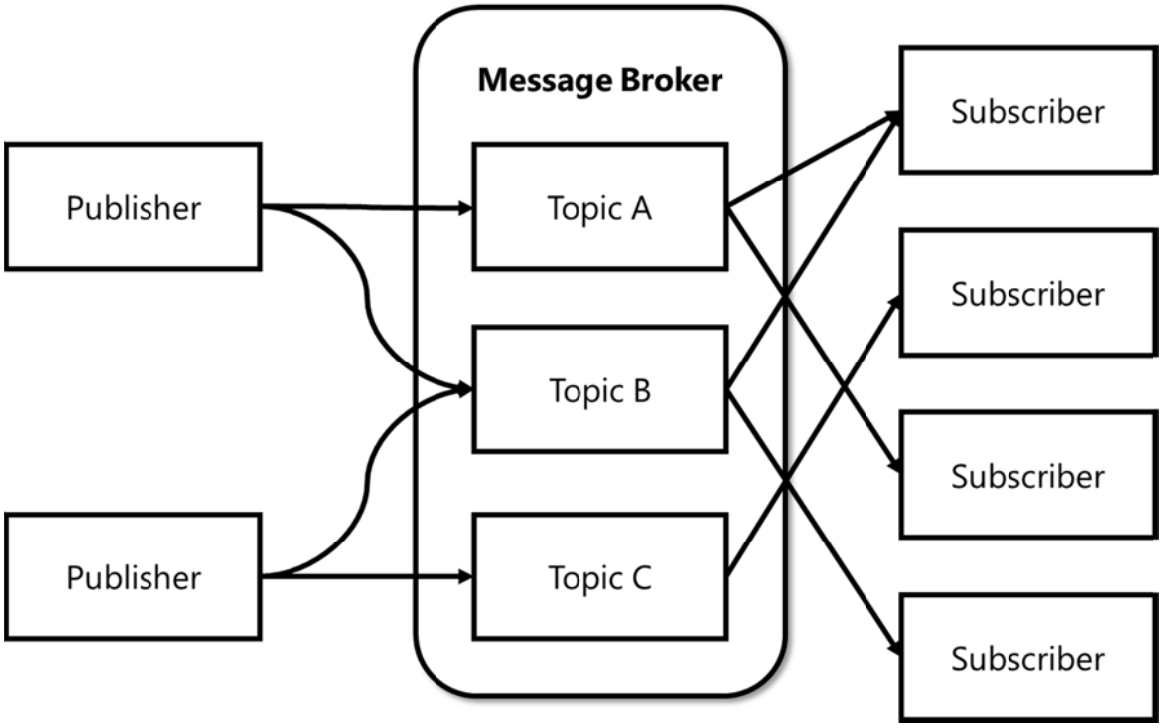

Figure 2-4: General idea of the topic-based publish-subscribe paradigm.

2.2.3 ROS computation graph

The ROS computation graph is based on a topic-based publish-subscribe model because of the flexibility it provides. However, this model is not very well suited for synchronous transactions, which are often useful in robot systems. As a consequence, the ROS computation graph also introduces the notion of *services* into the topic-based publish-subscribe model. Another feature that was deemed useful for debugging a robot system is storing sequence of data in order to enable the possibility to replay the data in the exact same sequence to a topic at a later date. Such as stored sequence of data is referred to as a *bag*. The ROS computation graphs consists of seven elements: the *master*, *nodes*, the *parameter server*, *topics*, *messages*, *services*, and *bags*. They are described as follows:

Master The master is a process node that acts as a name registration-server and lookup-server for the rest of the computation graph; just as the message broker in Figure 2-4. When a node wants to publish to a topic it first looks up the address of all other nodes through the master node. Without the master node, all of the other nodes in a ROS system would unable to find each other.

Nodes A *node* is a process that performs computation of some sort. Nodes can be used for many different things, e.g. one node could control an actuator while another node performs path finding for the robot. A node can be a publisher, subscriber, service provider, service requester, or all these at the same time.

Parameter server The *parameter server* is described by ROS Wikipedia[*] as a "server [which] allows data to be stored by key in a central location. It is currently part of the Master."

Messages A *message* is a data structure which is made up of specified fields. Nodes use messages to communicate with each other. The fields can be made up by standard primitive types such as: Integer, Point, and Boolean. The fields may also be made up by nested primitive types or other previously defined message types if they are included in the project. An example of a message-type definition in ROS can be seen in Table 2-1. This example data type consists of two primitive data types called *array_x_value* and *array_y_value*, and an included message type from the *geometry_msgs* library of type *Pose*, here called *pose*.

Topics Messages are sent and received between nodes on the basis of publishing and/or subscribing to a *topic*. The name of a topic is used to identify the content of a message and it also determines where a message should be routed. Instead of sending a message to a certain host, a node just publishes a message to a topic. If a node is interested in a certain type of messages, it can subscribe to a topic to receive all messages that are published for that certain topic. For a given topic, there may be multiple concurrent publishers and subscribers. However, there may only be a single message-type on a specific topic.

[*] Available at http://wiki.ros.org/.

Services Although the publish-subscribe model of topics is very flexible, its many-to-many, one-way transport is not particularly good for request-reply interactions commonly found in distributed systems. For this type of interaction between nodes, services are used instead. Services can be seen as a *remote procedure call* (RPC). A service is defined by a pair of request-reply message structures.

Bags Bags are the format for recording and saving message data, such as sensor data, in ROS. The recorded message data can also be played back into a topic at a later time. This is very useful for forwarding messages or developing and testing algorithms.

Table 2-1: ROS message type consisting of two primitive data types and an included message type.

int64 array_x_value
int64 array_y_value
geometry_msgs/Pose pose

Figure 2-5 shows a schematic summary of the ROS computation graph. As noted above, a node can be a subscriber or a publisher of a topic, or both. As can be seen in the figure, by subscribing to a topic, a node tells the master to add its address to the list of subscribers for this topic. When a node later publishes a message on this topic, the master will tell the node which nodes have subscribed to this topic; then the node will use this information to send copies of the message to all of these destination nodes. In the case of a service, when a node asks for a service, the master answers with the address of the node that offers the requested service. The node that wants this service will then initiate a one-to-one communication channel to the service node and gain access to the desired service. Topics and services in ROS uses TCP as the default transport protocol.

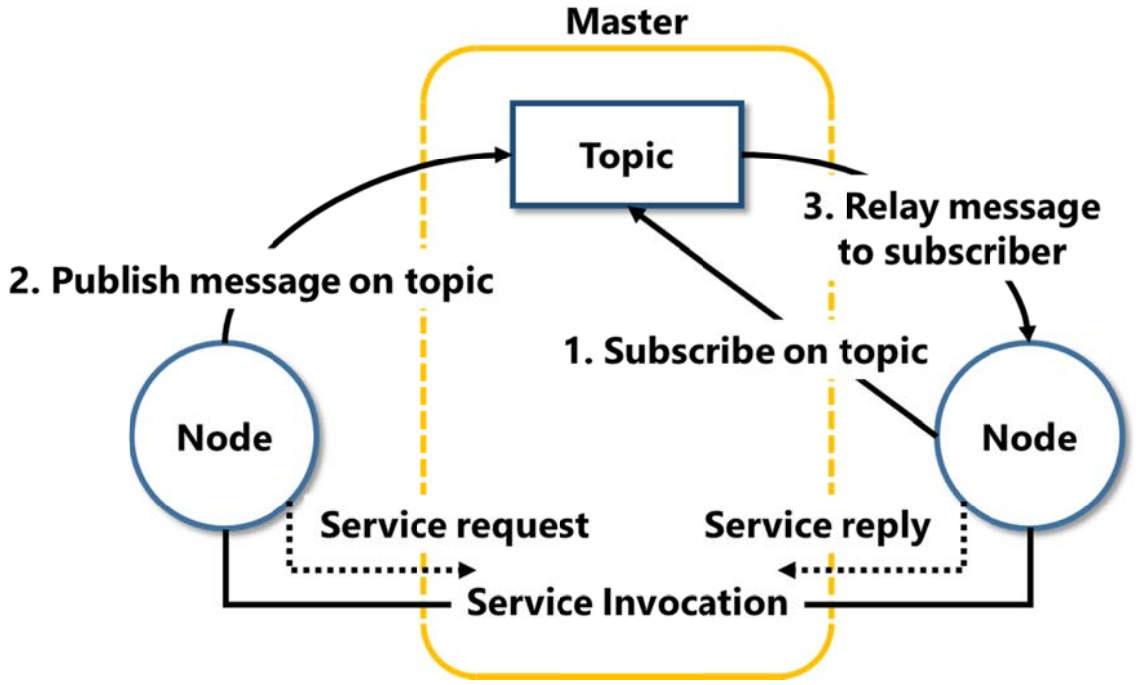

Figure 2-5: A schematic diagram of the ROS computation graph.

2.3 TurtleBot 2 robot development kit

TurtleBot 2 is the second generation of the TurtleBot development kit series originally developed by Willow Garage [48]. More recently, TurtleBot has become a licensed trademark of the Open Source Robotics Foundation*. The robot development kit consists of a hardware platform (shown in Figure 2-6) and an open-source software platform with many useful modules. A more detailed description of these modules is given in Section 2.3.1.

The TurtleBot 2 hardware platform (shown in Figure 2-6) consists of a Yujin Robot Kobuki base station, a frame with three levels, a Microsoft Kinect, and a ROS compatible laptop computer. The Kestrel weather sensor was connected to the Kobuki base station in order to satisfy the use-case described in Section 4.1. The Kobuki base station is powered by a 12V brushed DC motor and uses a factory calibrated 3-axis digital gyroscope for localization†. The laptop computer runs ROS on top of an Ubuntu 12.04 Long Term Support (LTS) distribution. It is used to control the data flows between hardware modules as well as doing the laser scan- and point-cloud computations used in navigation. The Microsoft Kinect is used to provide depth perception, imaging, and a point cloud. These operations will be further explained in Section 2.3.2 and Section 2.3.3.

* More info about retailers and the TurtleBot trademark can be found on http://www.turtlebot.com.
† A more detailed hardware description can be found at http://kobuki.yujinrobot.com/home-en/documentation/hardware.

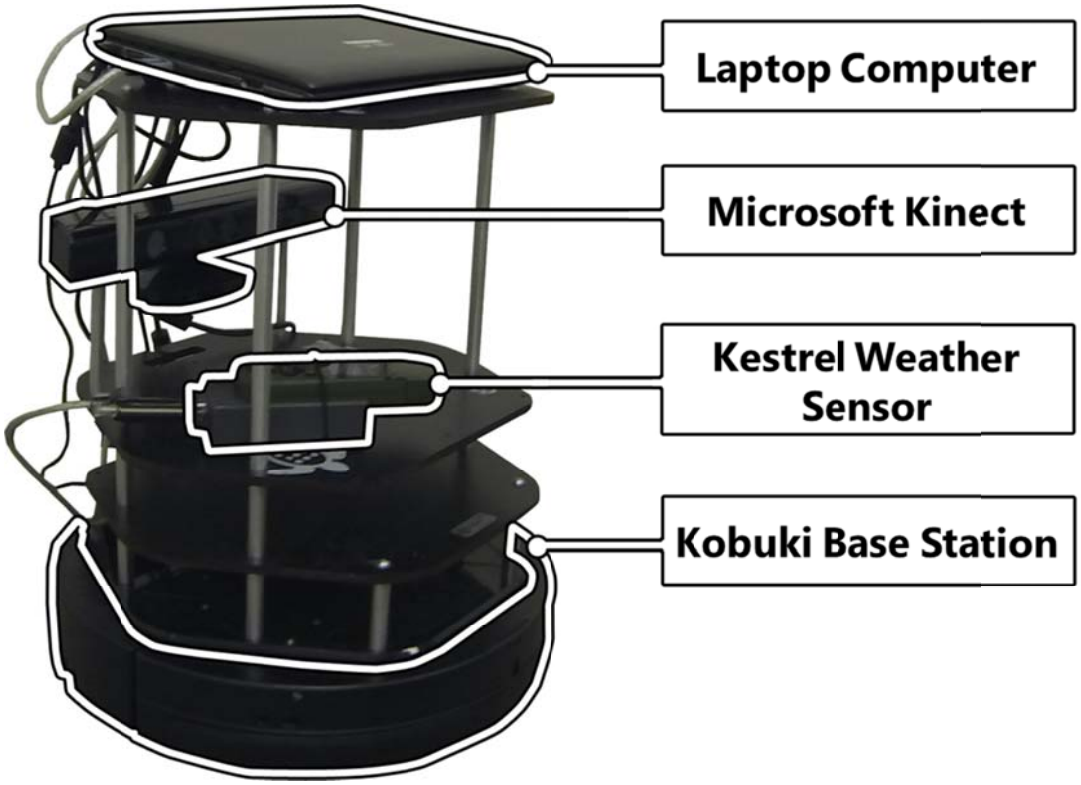

Figure 2-6: Hardware modules of the TurtleBot 2 robot development kit and a Kestrel Weather Sensor.

2.3.1 Turtlebot 2 software modules

The TurtleBot 2 robot development kit has an open-source software platform* with many useful modules that include basic drivers, a navigation stack, a person recognition and follower stack, and much more. Some of the key libraries and modules used in this thesis project are *turtlebot_bringup*, *turtlebot_navigation*, *gmapping*, *amcl*, *move_base*, *map_server*, and *rviz*. A more detailed description of each module and library are as follows:

turtlebot_bringup — This library contains all the scripts necessary for setting up variables and starting low-level drivers for all of the hardware components.

turtlebot_navigation — This library contains all modules necessary used for navigation in spaces.

gmapping — Gmapping is a third-party package that provides the TurtleBot 2 with laser-based SLAM functions. It can use laser data from the Kinect and pose data collected from the gyroscope to create a 2-D occupancy grid map of a space. This grid map can later be saved and reused to perform localization in a building.

* The source code can be found at https://github.com/turtlebot.

amcl	Amcl is a module that provides probabilistic localization of a robot moving in a 2D space. Adaptive Monte Carlo Localization (AMCL) [49] [50] uses adaptive particle-filters, such as KLD-sampling (Kullback-Leibler divergence) [51], to compute the pose of the robot against a known map.
move_base	The move_base package is a major component of the turtlebot_navigation stack. A user can send an action to the move_base in the form of a goal in the map. The module can use maps, laser scans, point clouds, transforms from the amcl module, and odometry information to create a global- and local plan to reach the goal in the map. Further details about this module are described in Section 2.3.3.
map_server	The map_server node provides other nodes in ROS with map information. It can also save a dynamically generated map, created with the gmapping tool, to a file for later use. Maps are created in the YAML file format. More about this server is given in Section 2.3.2.
rviz	Rviz is a 3D visualization tool which enables an operator to view the robot on the map and interact with move_base through a Graphical User Interface (GUI). Rviz can be used for many different things, but its details are outside the scope of this thesis project*.

2.3.2 Map creation using the gmapping tool

In order to understand how the TurtleBot navigates in a space, it is essential to know how maps are created and what algorithms used to solve the simultaneous localization and mapping (SLAM) problem. This subsection will not dive deeply into the area of robot localization techniques, but will cover the basic concepts behind the techniques used by the *gmapping* module.

The *gmapping* module is a ROS wrapper for the third-party *gmapping* tool written by Grisetti, Stachniss, and Burgard [52]. The tool uses laser scanner and odometry data to build an occupancy grid of a space by estimating the posterior probability over maps and trajectories with the help of a Rao-Blackwellized particle filter [53] [54]. The general idea of an occupancy grid is that a space is represented as an array of cells. Each cell in the occupancy grid contains a probabilistic value that the cell is occupied. An occupancy grid map will usually represent a cell as a pixel in an image, where a black pixel means that the cell is occupied (by a wall, human, or something else), whereas a white pixel means that the cell is free. A gray pixel means that the occupancy of the cell is unknown [55].

Before the TurtleBot can navigate using a map of a known space, it is necessary to create the map and save it to a file. This can be done by running the *gmapping_demo* application, containing the *slam_gmapping* node, in the *turtlebot_navigation* library. The *slam_gmapping* node takes laser scan data from the Kinect and odometry data while the robot is running around in a space. It uses these data to build a map which is published on

* Interesting guides and tutorials for the use of rviz can be found at http://wiki.ros.org/rviz.

a topic with the message type *nav_msgs/OccupancyGrid*. Such a map represents a 2D (grid) map of a space, where each cell in the grid represents the probability of occupancy. The *view_navigation* application GUI in the *rviz* library can be used to guide the robot around the room, as well as to view the map as it is being generated by the *slam_gmapping* node. Figure 2-7 is a map of the laboratory corridor generated by having the robot run around with the *gmapping_demo* application along the path shown as a red line. This map was saved to a file using the *map_saver* function in the *map_server* application.

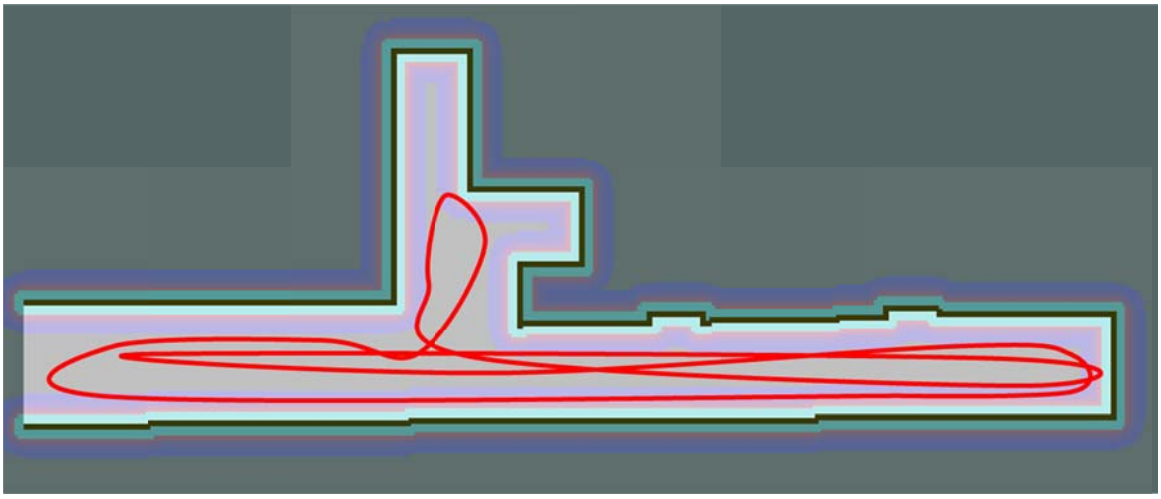

Figure 2-7: The resulting occupancy grid map of the laboratory corridor and the path of the robot when generating the map with the gmapping module.

The *map_server* application subscribes to a topic with the message type *nav_msgs/OccupancyGrid*. When the user is satisfied with the map and wants to save it, the *map_server* will retrieve the occupancy grid map from the topic and save the map as a pgm* image file and saves the metadata in a YAML† file. Table 2-2 shows the resulting YAML file of the generated map. The file consists of six different parameters: *image*, *resolution*, *origin*, *negate*, *occupied_thresh*, and *free_thresh*.

Table 2-2: The contents of the resulting occupancy map after saving it to a YAML file.

image: /home/turtlebot/Turtlebot_Maps/my_map.pgm
resolution: 0.050000
origin: [-29.800000, -34.600000, 0.000000]
negate: 0
occupied_thresh: 0.65
free_thresh: 0.196

* A grayscale image format. Further details are available at http://netpbm.sourceforge.net/doc/pgm.html.
† YAML (YAML Ain't Markup Language) is a data serialization standard. More details are at http://www.yaml.org/.

image	The file path to the pgm image file that represents the occupancy grid.
resolution	The size of a pixel relative to the real world in units of meters along the edge of the pixel.
origin	The original 2-D pose (also called origin of the map) of the robot in units of pixels from the lower-left pixel in the map.
negate	Specifies if the occupancy color scheme should be reversed or not.
occupied_thresh	Pixels with an occupancy probability greater than this threshold should be classified as occupied.
free_thresh	Pixels with an occupancy probability lower than this threshold should be classified as free.

2.3.3 Navigation using simultaneous localization and mapping

Now that the TurtleBot has a functioning occupancy grid map (which can be retrieved from the *map_server* module), the robot can use its navigation stack to navigate to positions (goals) set by the user. The way the TurtleBot uses sensor data, the grid map, and SLAM to navigate an area is well described in [56]. Figure 2-8 is an attempt to visualize the entirety of the navigation stack and the relationships between the relevant modules.

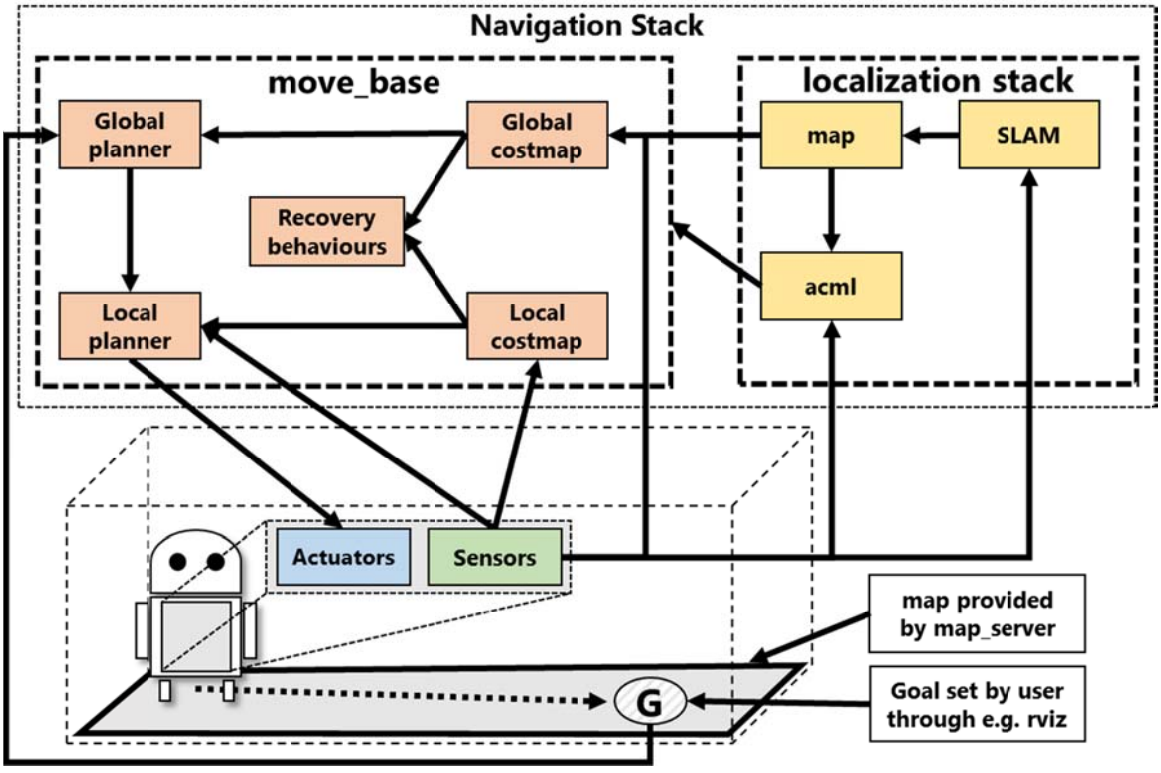

Figure 2-8: A summary of all the elements and modules of the ROS navigation stack and their relationships.

In this process we assume that a robot with sensors and actuators is placed in a room, an occupancy grid map of the room has already been created with the *gmapping* tool, and the *map_server* module provides the map to the modules that request it via topics and services. The first step is that a user (or program as in this thesis project) provides the robot with a goal to which the robot should move. A user can do this by simply publishing a *geometry_msgs/PoseStamped* goal to the global planner module on the topic *move_base_simple/goal*. If the user needs continuous updates about the status of the goal, it is also possible to make an interactive action call to a SimpleActionServer in the move_base module[*]. Once a goal is set in the global planner, the amcl module will make use of laser scan data (from the Kinect) and odometry data (derived from the gyroscope) in order to localize the robot on the map. The localization is done by computing the posterior robot pose estimates for a set number of particles and trying to fit them to the known map. The SLAM algorithm dynamically updates the map in order to accommodate moving objects.

When the robot has found its position and pose in the map, the global planner starts to calculate a rough potential path to the goal with the help of the global costmap. The global costmap incorporates data about where in the map it is possible for the robot to move without bumping into anything. However, since the global planner tends to create infeasible paths to the goal, a local planner uses a dynamic window approach to account for this shortcoming [56]. The local planner also sends commands to the actuator in order to move the robot along a path. All the modules described above are concurrently executing in order to localize the robot on the map, update the map, create costmaps used for path planning, and plan both a global and a local path to the goal.

2.4 Related work

The problems described in Section 1.2 are not exclusive to the area of CNR. The mobile industry has been trying to solve its energy problems by remote execution or migrating applications, or even entire VMs, to the cloud depending on network conditions and other parameters [33]. This has spawned a whole new area in the mobile industry called *mobile offloading*. There have already been several proposed architectures to solve this problem such as MAUI [33], CloneCloud [57], ThinkAir [58], and a proposed use of machine learning algorithms to create a dynamic offloading scheduler [59]. This section will introduce what have already been done in the area of CNR to make cloud robot architectures network aware. However, since the mobile industry is working on solving a similar problem, related work to mobile offloading will also be introduced.

2.4.1 Pioneering module offloading work in CNR

Already in the early stages of CNR, Hu, Tay, and Wen proposed that a CNR architecture should have offloading strategies that consider various factors, such as task deadline and amount of data to be exchanged, in order to minimize battery consumption while providing high QoS [23]. They identified the fundamental problem as the trade-off between the energy consumption from data transmissions, when transmitting the necessary data to the cloud, and the energy consumed when executing a task locally on the robot.

[*] This is done with the help of the ROS “actionlib” package, which provides an interface for interfacing with pre-emptible tasks. More info about this package can be found at http://wiki.ros.org/actionlib.

A dynamic voltage scaling energy consumption model for local execution was compared against a polynomial energy consumption model for the cloud execution, in order to find the optimal operational regions of both schemes. They show that for a given delay deadline and a preset number of bits to be transferred, that the minimum energy consumed can be compared between these two models. Although they take into consideration that the network conditions may change by looking at the delay, there are still questions as to how this model would work in an unstable network environment, since they have not consider changing network bandwidth in their work.

2.4.2 Team MIT's DARPA robotics challenge contribution

Fallon et al. describe their contribution to the (United States of America) Defense Advanced Research Project Agency (DARPA) Robotics Challenge Trial* in a Massachusetts Institute of Technology (MIT) Computer Science and Artificial Intelligence Laboratory (CSAIL) technical report [60]. They used an Atlas humanoid robot frame and prioritize teleoperation by a human over attempting to create an autonomous system. The robot system is divided into three main components: an operator control unit, a field computer, and the hardware components. The different planners (manipulation-, footstep-, and task planner) are deployed on the operator control unit, but a module may be offloaded to a cloud for better performance. In between the operation control unit and the field computer on the robot is a restricted link that is supposed to simulate different harsh network environments of a danger site.

A part of the technical report explains the different implementation choices of the network- and transmission layer modules used for communication between the robot and the operator side, as well as the tools and programs used to simulate different harsh network environments. The system focuses on compressing data in order to transfer data efficiently between the robot and the operator control unit. However, there is not much explanation of how the robot would fare if the connection were to be lost unexpectedly. The Robotics Challenge Trials let participants have pretty good control over their robots with teleoperation. However, the Robotics Challenge Finals promise to require the robots to function more autonomously, by inducing "long [network] blackouts of up to a minute" [61]. This sounds like a tough criteria to fulfill based on what is written in the technical report, as the robot would need some kind of migration module to make use of the planners even under total blackout of communications.

2.4.3 MAUI: code offloading in smartphones

Cuervo et al. [33] were ones of the earliest groups to propose an architecture that makes use of code offloading to solve the battery problem in the mobile industry. Their proposed architecture, called MAUI, uses code portability to create a clone of a smartphone in the cloud. They strived to maximize energy savings while minimizing the need to change anything in existing applications. The general idea is that a clone of the smartphone is running in a server (in the cloud) and every time a function is called, an optimization framework calculates whether the function should be run locally or be executed in a server.

* DARPA Robotics Challenge is a competition of teams that develop hardware and software robot systems designed to assist in man-made or natural disasters. Read more at http://www.theroboticschallenge.org/.

The optimization framework makes use of information, gathered from profiling the device, the program, and the network, as input to a global optimization problem. The profiling of the device looks at the energy consumption and Central Processing Unit (CPU) utilization of the function. This is similar to what a program profiler does, i.e., it analyzes the program and determines what state is required to be transferred to the cloud, as well as considering the program's runtime duration and required number of CPU cycles. For the profiling of the network, MAUI sends 10 KB of data over a Transport Control Protocol (TCP) connection to the server-side and calculates the average throughput by measuring the time it took to transfer all of the data. A possible downside of this solution is that it is hard to profile a dynamic program which has very different total runtime and whose bandwidth requirements depend upon the environment and task.

2.4.4 CloneCloud

CloneCloud, by Chun et al. [57], is a mobile offloading architecture that tries to solve the same problem as MAUI. CloneCloud also uses an optimization solver to optimize the energy consumption and execution time. Similarly to MAUI, the solver determines what functions should be executed in the cloud and which functions should be executed locally in the robot, by profiling the program. Although similar in concept, CloneCloud differs greatly from MAUI in implementation, MAUI is implemented on Microsoft's .NET framework while CloneCloud is implemented upon a Dalvik VM.

CloneCloud improves upon MAUI by not requiring source code availability and does not require that the programmer identify which functions would work better in the cloud. It also addresses the scalability problem of MAUI by using VMs in the cloud to simulate the whole smartphone. CloneCloud is very similar to MAUI, hence it also has the same potential downside in that it is hard to make a good estimate of the runtime in a dynamic program and what bandwidth is needed as both may differ greatly depending upon the environment and task.

2.4.5 ThinkAir

Kosta et al. [58] proposes a mobile offloading framework called ThinkAir that promises to solve MAUI's scalability problem, as well as to improve upon CloneCloud by using an online method-level offloading solution. ThinkAir improves upon the CloneCloud VM solution by enabling parallelization between multiple VMs and providing dynamic resource allocation depending on the task.

The ThinkAir architecture was designed with four main objectives in mind: the architecture should dynamically adapt to changing environments, it should be easy to implement for developers, performance should be improved by using cloud computing, and the architecture should support dynamic scaling in computational power. In order to dynamically adapt to a changing environment, ThinkAir implements three different profiling models: a hardware-, software-, and network profiler. The hardware profiler monitors the CPU, the screen, the Wi-Fi interface, and the 3G radio interface, then sends this state information to an energy estimation model. The network profiler measures both the round-trip time (RTT) between the smartphone and the server and the amount of data sent and received during a time interval, in order to calculate the effective network bandwidth.

2.4.6 Machine learning offloading scheduler

While other related works has focused on the core mechanics of offloading techniques, Eom et al. [59] propose to improve upon these systems by implementing an adaptive offloading scheduler. Their adaptive offloading scheduler is based on a machine learning algorithm that takes into consideration varying network conditions, as well as the workload and the load at the device. They consider several different machine learning algorithms and come to the conclusion that an Instance-Based Learning algorithm best fits the problem.

The machine learning algorithm takes four different variables as input in order to decide upon a module's placement: the local execution time, the size of the data to be transferred, the network bandwidth, and number of invocations needed for arguments setups. They show that with the help of a training set, the adaptive offloading scheduler can choose the "best" module placements on the fly 87.5% of the time in an environment when the network conditions are fluctuating. However, they do not report on how the system would fare with an application where it is hard to estimate how much bandwidth is needed at a certain time in the future.

2.4.7 Point cloud culling under communication restraints

Beksi and Papanikolopoulos [62] present another approach to successfully make use of module offloading to a cloud even under constrained network conditions. Their solution focuses on enabling real-time vision tasks on a robot with low on-board resources, by offloading the computationally heavy modules to the cloud. The transmitted data is analyzed on the server-side to discover if there is any congestion building up in the network. The server-side can throttle the connection using an adaptive threshold setting, in order to avoid causing congestion even in a network with constrained communication resources. The downside of this solution is that it is not robust against complete network failure, which means that the robot would not be able to recover if network connectivity were lost.

2.5 Summary

This chapter has presented the basic concepts of CNR as well as a general background on the ROS middleware and the TurtleBot robot system and its functionalities. Most of the related work in CNR has focused on enabling robots to function even with constrained communications, by using data compression or transmission throttling. However, there seems to still be a gap in what should be done when connectivity is temporarily lost. Although some of the related work on mobile offloading seems promising, it is hard to estimate how well the offloading solutions would work with a CNR system. The reason for this is that most of the mobile applications have predictable behavior in both CPU and bandwidth requirements; which we will see later in Section 4.2.2, is not always the case for a CNR system.

3 Methodology

The purpose of this chapter is to provide an overview of the research method used in this thesis. Section 3.1 describes the research process. Section 3.2 details the research paradigm. Section 3.3 focuses on the data collection techniques used for this research. Section 3.4 describes the experimental design. Section 0 explains the techniques used to evaluate the reliability and validity of the data collected. Section 3.6 describes the method used for the data analysis. Finally, Section 3.7 describes the framework selected to evaluate the cloud robot prototype and the network aware framework.

3.1 Research Process

Since the complexity of a CNR system is high, instead of looking at a simulated environment where the complexity might be simplified, it was decided that a real life cloud robot was to be tested and observed when placed in an unstable network environment. If any problems are found during the observations, the aim is to design and implement a solution to the problems. Figure 3-1 shows the steps conducted in order to carry out this research.

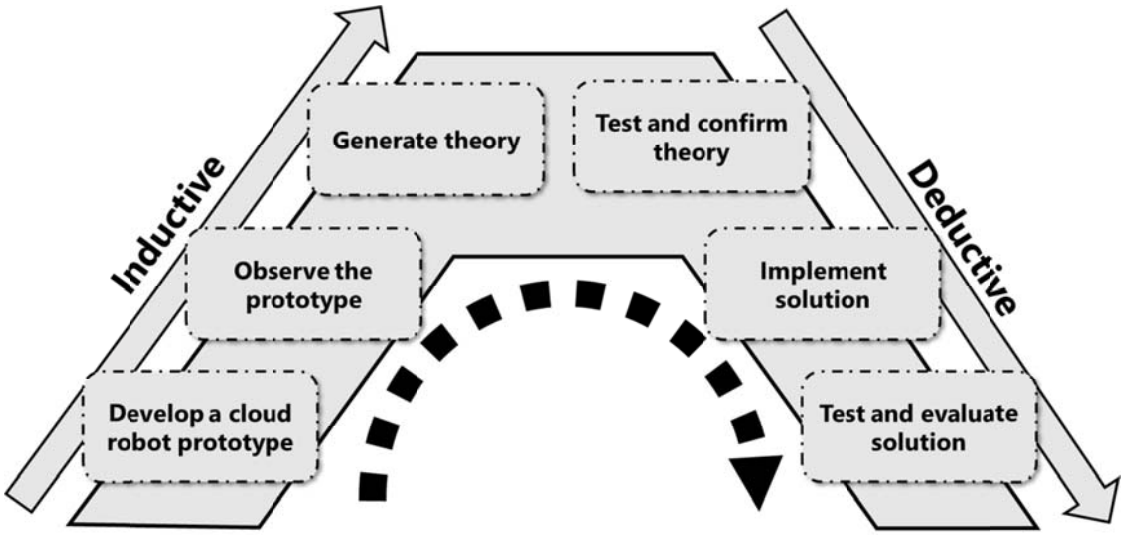

Figure 3-1: Overview of the research process used to carry out this research.

As the availability of commercial cloud robots is low, initially the focus will be put on developing a use-case where a cloud robot would be subject to an unstable network environment, and based on the use-case develop an actual cloud robot prototype. This step is described in Section 4.1. The next step will be to observe the prototype while it is executing a simple task (according to the use-case) in an unstable network environment. Based on the observations, any potential problems will be examined and theories to why these problem occur will be proposed. These theories will then be tested and confirmed in a series of experiments. These three steps are described in [63] (see Appendix VI) and Section 4.2.2. A network aware framework solution will then be designed and implemented to solve these potential problems. This step is described in Section 4.2. Lastly, the proposed solution will be tested and evaluated by a continuous running experiment which will result in recommendations for other researchers and developers in the area of CNR.

The results from these experiments and the recommendations and conclusions can be read in Chapter 5 and Chapter 6.

3.2 Research Paradigm

In conjunction with the research process, the research methodology of this thesis project will be qualitative. There have still not been any extensive research about network aware CNR systems, since the area of CNR is fairly new. Therefore a mixed research approach, containing both an inductive and a deductive touch, will be used in order to reach the goals of this project. Firstly, an inductive research approach is applied to create theories from observations of a cloud robot, then a deductive research is used to develop a network aware framework solution.

3.3 Data Collection

The data will be collected by a series of experiments where the proposed solution will execute a task in a corridor under various network conditions. The data will be collected by the program written by the author and saved to a tab separated value file (.tsv). Since more measurements are preferable, as many measurements as possible will be collected during a period of two months (July, August, 2014) when the robot will not be in the way of everyday activities at the laboratory.

3.4 Experimental design

This section will describe the test bed where the experiments were conducted as well as the hardware and software used in the experiments.

3.4.1 Experimental test bed

The experimental test bed was placed in a corridor in the Cyber Media Center Suita-branch laboratory. An overview of the experimental test bed can be seen in Figure 3-2. The physical layer of the experimental setup consists of three main components: a cloud robot (described in section 4.1), a wireless router, and a VM on a local cloud. The cloud robot was placed in the corridor and connected to a wireless router using Wi-Fi (802.11g). While the cloud robot was connected to a dedicated wireless router, there were other wireless networks in the vicinity of the experimental setup. The VM on the local cloud was connected to the router using a 100 Mbit/s Ethernet cable. The cloud robot and VM communicates using both ROS topics and services over TCP. The Linux network controlling tool *tc* (traffic control)[*] was used both in the cloud robot and the VM to simulate latency and bandwidth restrictions in the network. An example of how *tc* was used to model the traffic can be seen in Appendix I.

[*] Tc is used to show and/or manipulate traffic control settings in Linux kernel. More info can be found at http://lartc.org/manpages/tc.txt.

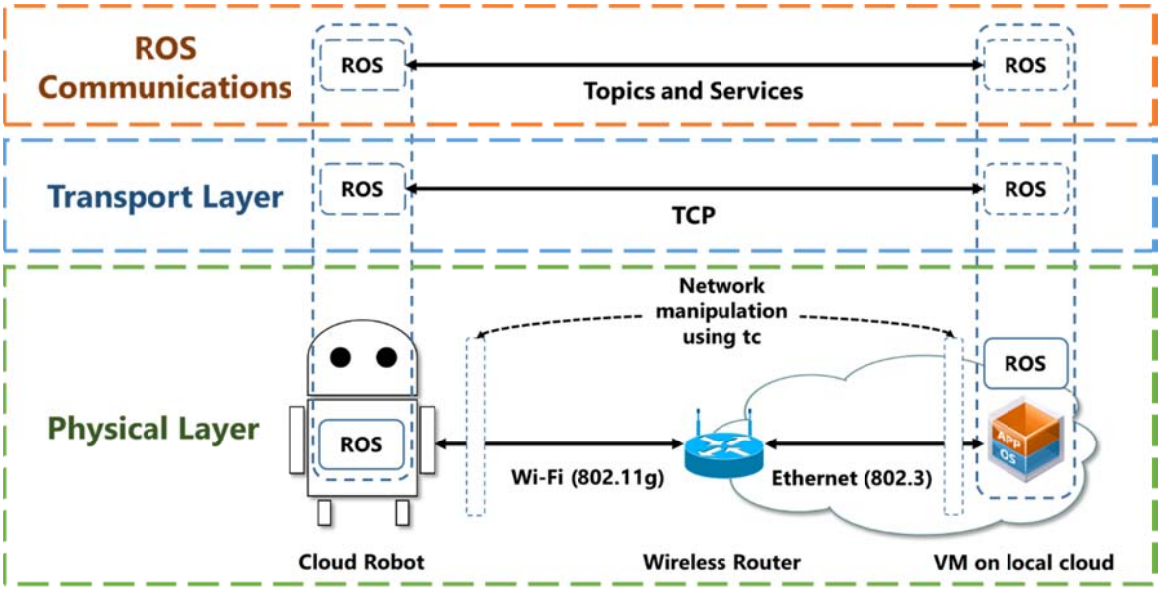

Figure 3-2: An overview of the experimental test bed.

In order to create a stable test bed with as few changing variables as possible, the Linux power management was turned off by calling *$sudo /etc/init.d/acpi-support stop* in the command line. Linux also has CPU frequency scaling functionality built into the system which enables the Operating System (OS) to automatically scale the CPU frequencies depending on the system load in order to save power. This functionality was disabled and the CPU frequency was set to always stay at 1.6 GHz by using the following command: *$sudo cpufreq-set –c [CPU_NUMBER] –f 1.6GHz*.

3.4.2 Hardware/Software to be used

The cloud robot prototype was implemented on top of a TurtleBot 2 robot development kit running ROS Hydro Medusa. The technical specifications of the laptop used by the TurtleBot 2 and the VM on the local cloud can be seen in Table 3-1. The cloud robot prototype also made use of a Kestrel Weather Sensor to measure the temperature in the corridor. The current latency was measured using an echo-response program written by the author and the network bandwidth was estimated using *netperf* [64]. The currently used bandwidth of the applications were measured using the *ifstat* [65] Linux module.

Table 3-1: Technical specifications of the devices used in the experiments.

Device	OS	CPU	RAM
TurtleBot 2 laptop	**Ubuntu 12.04 LTS 32-bit**	**Intel® Atom™ N2600 1.6GHz * 4 core**	**1 GB**
Local cloud VM	**Ubuntu 12.04 LTS 32-bit**	**Intel® Core™ i7-4900MQ 2.80GHz * 8 core**	**16 GB**

3.5 Assessing reliability and validity of the data collected

This section will briefly introduce the methodology behind how the reliability and validity of the collected data is assessed.

3.5.1 Reliability

The reliability of the collected data will be assessed by looking at what variables that might affect the results. Another variable that will be taken into account when assessing the reliability is the fact that the experiment is conducted with an actual robot instead of just a computer simulation.

3.5.2 Validity

The validity of the collected data is to be assessed by comparing the measurements obtained from the experiment in the laboratory corridor to that of the measurements that will be obtained from another experiment, which is to be performed in the actual data center.

3.6 Planned Data Analysis

This section describes the techniques used to analyze the obtained data as well as the software tools used to carry out the data analysis.

3.6.1 Data Analysis Technique

Before starting the data analysis, the data was put through data cleaning. Values that are corrupt or erroneous are not considered for the later data analysis. The data contains two classes of data (“onload” and “offload”), which will be plotted onto a graph for region distribution analysis. In order to find different relationships, the data will be put through a sorting process so that different variables in the data can be tested against each other. After making observations of different graphs, line fitting techniques will be applied to the data in order to find the point of trade-offs between onloading and offloading. Lastly, measurements from when modules were executed locally on the robot will be compared to measurements from when modules were executed remotely on the cloud.

3.6.2 Software Tools

As described in Section 3.3, the data will be collected by the program written by the author and stored in a tab separated value file (.tsv). Microsoft Excel will be used clean and sort the data imported from the tsv file. The sorted data will be analyzed using gnuplot[*] by looking at plotted graphs and applying line fitting techniques.

3.7 Evaluation framework

A normal CNR system would not be able to offload modules to the cloud safely when operating in an unstable network environment. Therefore, the proposed solution will be evaluated by comparing the potential energy savings and task efficiency of the proposed

[*] A command-line driven graphing utility for Linux. More info can be found on their homepage: http://www.gnuplot.info/.

solution, when operating under good network conditions, against a cloud robot where modules are only placed locally on the robot. Evaluating the proposed solution under random network conditions against a robot with locally placed modules is for future work.

4 Network-Aware Cloud Networked Robot

This thesis proposes to integrate functionality from the area of mobile offloading into the area of cloud networked robotics, in order to improve the total operation time of the robots and execution time of robot tasks when operating in unstable network environments. An overall summary of the proposed system can be seen in Figure 4-1.

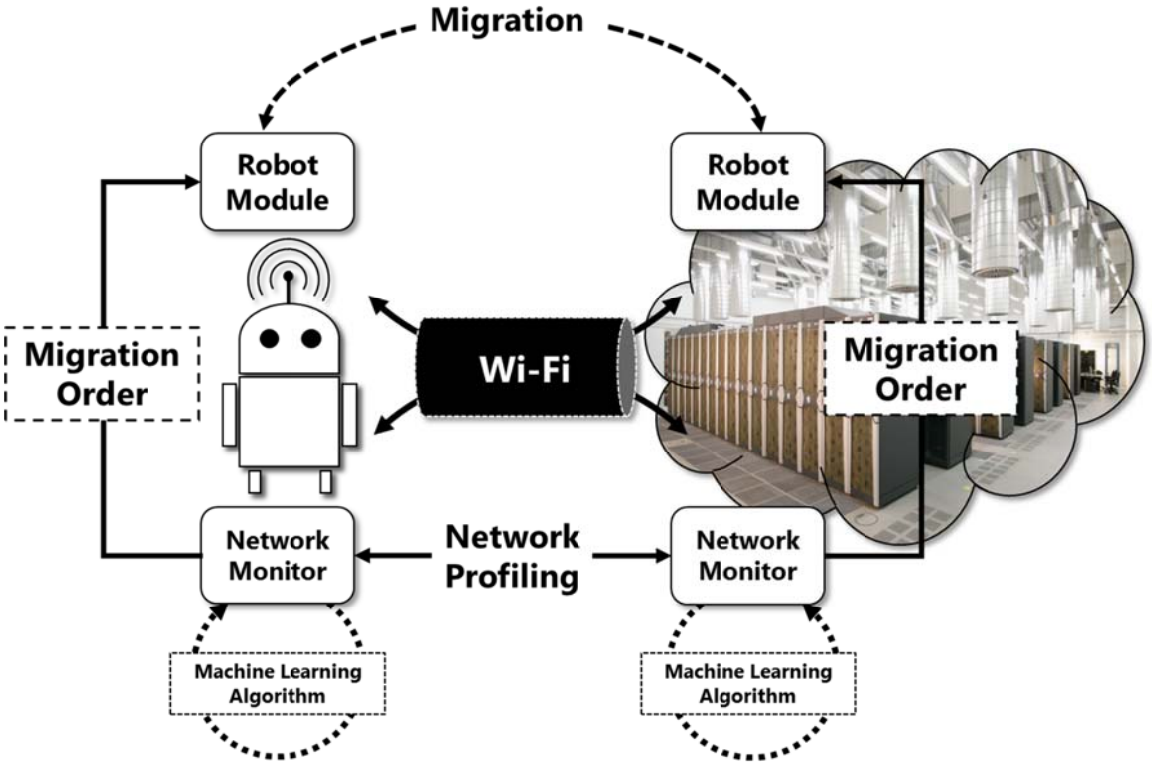

Figure 4-1: An overview of the proposal system design.

As can be seen in the overview, a robot is connected to a cloud, located in a datacenter, via a Wi-Fi connection. Inside the robot there are several robot modules that communicates with other modules inside the robot, as well as modules placed on the cloud. Network monitor modules are placed both in the robot and in the cloud. The network monitor modules exchange information about the network, through network profiling, and information about the status of the systems they are running on top of. This information is proposed to be run through a machine-learning algorithm which will decide if the network condition is good enough for the robot modules to be placed on the cloud, or if it is more efficient to place the robot modules inside the robot. When a decision has been made that the current network condition cannot provide sufficient resources, a migration order is sent from the network monitor to the destination robot module to trigger a migration.

This chapter is divided into two sections. To create a network-aware cloud networked robot system, a cloud networked robot prototype had to be developed to use as a reference point. Chapter 4.1 covers the major design decisions and implementation stages done in order to develop a cloud networked robot prototype. Chapter 4.2 covers the development process of the network-aware cloud networked robot system as well as introduces the notion of using tasks as evaluators instead of time.

4.1 Cloud Networked Robot Prototype

Before it was possible to decide on a design for the network-aware CNR system, an actual cloud networked robot prototype had to be developed to use as a reference point. The cloud networked robot prototype described in this chapter was developed for this thesis but was also partly presented in [63] (see Appendix VI).

The first thing that had to be done was to decide on a use-case for the cloud networked robot prototype. Several different use-cases were considered, but the use-case which was deemed most feasible to complete under the time-limit at the time was "a datacenter monitoring cloud networked robot". At the start of this project, a project to create an autonomous datacenter monitoring system was underway at Osaka University [66]. A simulation showed that there was a possibility of hotspots forming in the data center, which the temperature sensors on the hardware would not be able to sense right away. The report proposes that autonomous robots could move around in the data center and measure the temperature at these spots. The required information could then be transferred to a cloud to be processed by a map-reduce system, such as Hadoop, to uncover trends in the data. An explanation of the use-case can be seen in Figure 4-2.

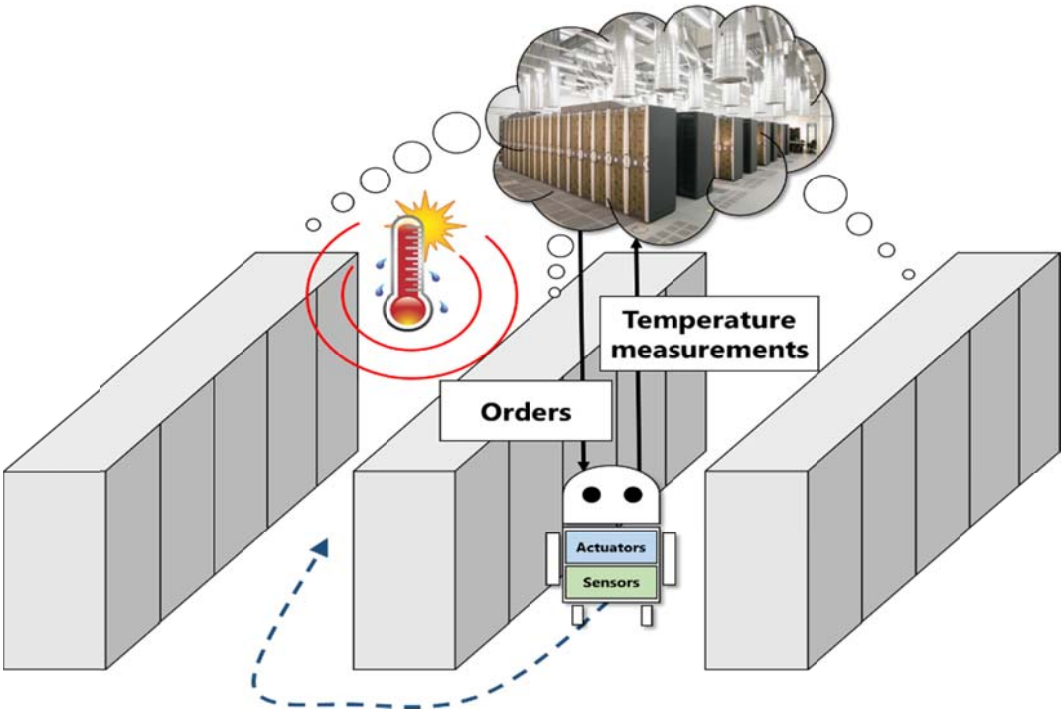

Figure 4-2: The use-case where a cloud robot monitors the corridors of a datacenter for temperature hotspots.

Instead of putting a lot of temperature sensors in the data center to cover all areas, a cloud robot can use its equipped temperature sensors to search the corridors of the datacenter for temperature hotspots. The robot will get orders from the cloud on where to search for these temperature hotspots and begin to move to the position of the order while continuously scanning the route for these hotspots. While moving around in the datacenter, the robot should also be able to detect and avoid both immovable and movable objects that are in the way, e.g. the datacenter service personnel. When an abnormal value has been found, the robot should try to find the origin, the temperature hotspot, and inform the cloud of the findings.

4.1.1 Datacenter monitoring cloud robot implementation

The TurtleBot 2 robot-suite [6], described in Section 2.3, was chosen as a base for the cloud robot and additional high-level modules such as Hadoop, multi-robot coordination modules, and machine-learning algorithms for determining new goals for the robot was placed on a local cloud.

As described in Section 2.3.1, there are several basic modules available on the Turtlebot 2 platform, such as move_base and turtlebot_bringup that interact with sensors and actuators. If we can compare these modules explained above in terms of a human, there is a *body* with sensors and actuators on the physical robot and *intelligence* on top of the cloud. However, there is no *brain* to connect these two elements. A proposal on how to connect the *body* and *intelligence* can be seen in Figure 4-3.

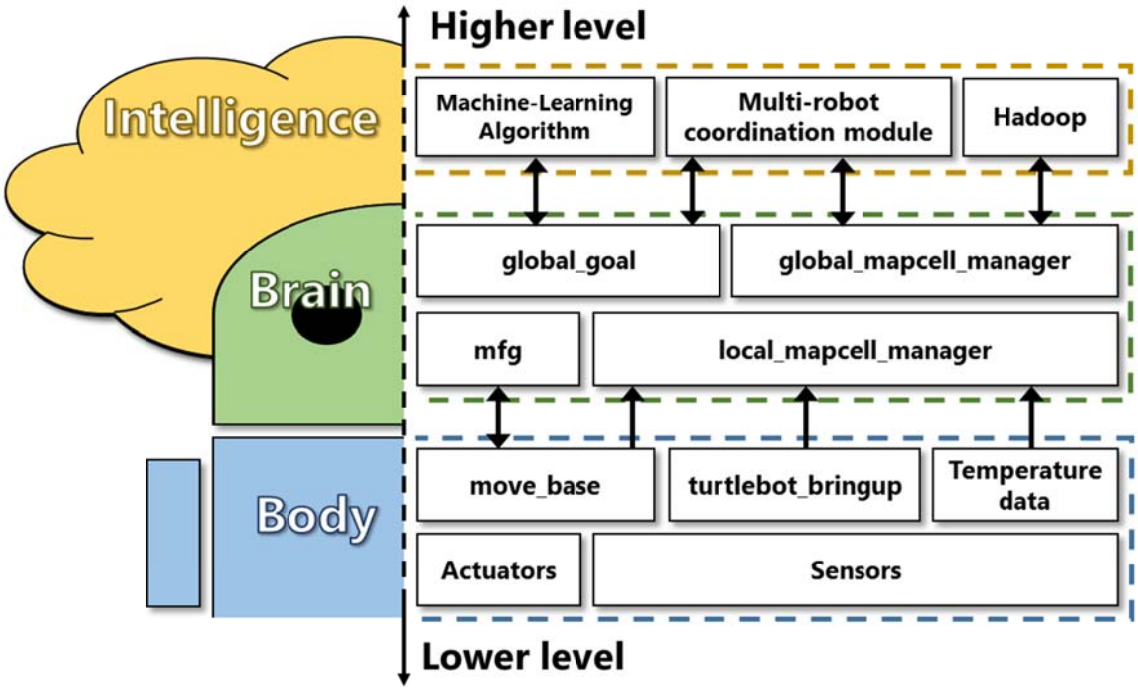

Figure 4-3: An overview of the proposal cloud network robot design.

The *brain* contains modules that can all interact with each other in order to get or set information. However, in order to tackle the problem where a cloud networked robot loses connectivity to a cloud and stops completely, a kind of hierarchy within the brain was thought up where modules can be seen as either *global* or *local*.

Modules that can be seen as mostly *local* should take commands, or have some of the knowledge from the *global* modules. At the same time, *local* modules should be able to be independent for some time from the *global* modules so that even if connection is lost, the robot will still continue to be useful. *Global* modules on the other hand should have a full overview of the state of the cloud robot, as well as the information gathered by the robot. By using this relationship between *local* and *global* modules, placing *local* modules on the robot and *global* modules on the cloud can increase the cloud robots robustness against the connection problems in unstable network environments.

As can be seen in Figure 4-3, move_base interacts with two *local* modules called *mfg* (measurement_feedback_goal) and *local_mapcell_manager*. The *mfg* module sends objectives, "goals", to the move_base with the help of SimpleActionServer. Move_base responds with updates to the goal, e.g. the message "SUCCEEDED" if the robot has successfully reached the goal. The *local_mapcell_manager* receives information from the move_base module of where the robots has recently been and stores this for future use.

4.1.2 Map and mapcells

As was described in Section 2.3, move_base takes goals with the help of a SimpleActionServer in the form of *geometry_msgs/PoseStamped* messages. Likewise the robot position in the provided map, which is acquired by listening to frame transformations from the "tf" module, is also in the form of the PoseStamped message type. The PoseStamped message type contains the x, y, and z-position in a map as well as relative rotation of the robot in the form of x, y, z, and w-values.

However when listening to frame transforms, the robot position will be very precise in the map. This makes it harder to develop simple algorithms for where the robot should move after having completed a task. Therefore, a concept of map and mapcells was thought up where the map, which was provided to the "map_server", is divided into several square areas, as can be seen in Figure 4-4 where the right side of the map is divided into blue squares. By doing this, a certain position in the map can be specified to be a part of a mapcell. Such a mapcell can contain all kinds of information such as the last time the robot was in the mapcell, as well as the time and position of every temperature measurement that was recorded in the mapcell. In short, a mapcell acts as a container of different information concerning a certain area of a room.

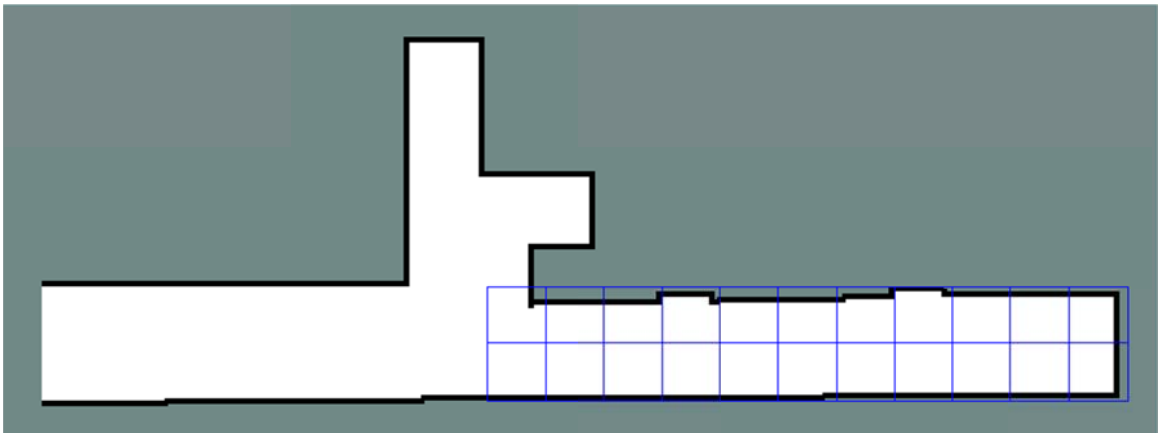

Figure 4-4: An area of the laboratory corridor is divided into two rows of mapcells.

As described in Section 2.3.2, when the map was originally created, the *gmapping* program remembers the original 2-D pose (also called origin of the map) of the robot until the whole map is created. It then calculates and sets the origin as pixels from the lower-left pixel in the map. The left part of the corridor was also mapped in order for the robot to be able to have fast localization even in the left part of the corridor. The origin in the map was automatically set to be [-29.8, -34.6, 0.0] in the pgm image when it was created with the *map_saver* program. As this only signifies the default position of a robot before calibrating the current position, this value was unchanged. The resolution of this map is 0.05 meters/pixel and each mapcell was set with a width and height of 1 meter. Since the corridor was 11 meters long and almost 2 meters wide, it was divided into 22 mapcells; 11

mapcells long and 2 mapcells wide. The dimensions of a mapcell is decided by the sextuplet: <*min_x, max_x, min_y, max_y, center_x, center_y*>. The center values in this sextuplet serves as a reference point in a mapcell, where the robot can move to if it gets an order to move to a certain mapcell. The algorithm to decide which mapcell a position belong to is described in Figure 4-5.

The algorithm described in Figure 4-5 is very complex for such a simple task and could have been replaced with a simple mapping transform algorithm, which could be based upon the transformation to a position given the size of a mapcell and an origin. However, there was not enough time to implement this and it was decided that the algorithm was good enough for this small scale thesis project.

```
INPUT value_x, value_y
FOR X = 0 to 10
   FOR Y = 0 to 1
      IF MapCell[X][Y].min_x < value_x &&
               MapCell[X][Y].max_x >= value_x &&
               MapCell[X][Y].min_y < value_y &&
               MapCell[X][Y].max_y >= value_y
         RETURN MapCell[X][Y]
      ELSE
         CONTINUE
      END IF
   END FOR
END FOR
RETURN FALSE
```

Figure 4-5: The pseudo code for the algorithm that decides which MapCell a coordinate resides in.

4.1.3 Global goals and measurement feedback goals

Current hardware on most cloud robot bases are not powerful enough to be able to analyze massive amounts of data, in order to decide on complex tasks or planning of paths, at the same time as being able to properly control local modules. This can be solved by moving such resource heavy modules to a cloud. However, if the robots are operating in environments with unstable network connections, losing connection to such important modules might mean that the robot will stop to function, since it will not get any new goals after completing a previous goal.

A goal or a task can be everything from a complex task such as “scanning a part of a datacenter and compare to monthly trends”, to more simple tasks such as “move the robot 20 cm forward”, “avoid the obstacle in front of the robot”, “calculate the current robot position”, and “measure the current temperature”. What can be noted is that there is a huge difference in how much time it takes to complete a complex task compared to a much

simpler one. While “scanning a part of a datacenter and compare to monthly trends” can take everything between 4-20 minutes (maybe even longer), a simple task such as “measure the current temperature” takes less than 10 milliseconds.

This difference can be exploited in order to make the cloud robot more robust against the problem explained above. Complex tasks, which we can call *global goals*, can be decided by a module that is placed on a cloud while less complex tasks, which we can call *measurement feedback goals* (MFG), can be placed locally on the robot. The relationship between *global goals* and MFGs can be seen in Figure 4-6. In this example a square room is divided into five times five mapcells. A mapcell in the grid at x, y, can be addressed as mapcell(x)(y). A “global goal” is assigned to mapcell(5)(0) by a module residing on a cloud. MFGs are tasks that can be seen as necessary in order to complete the *global goal* or in worst case a detour for the *global goal*.

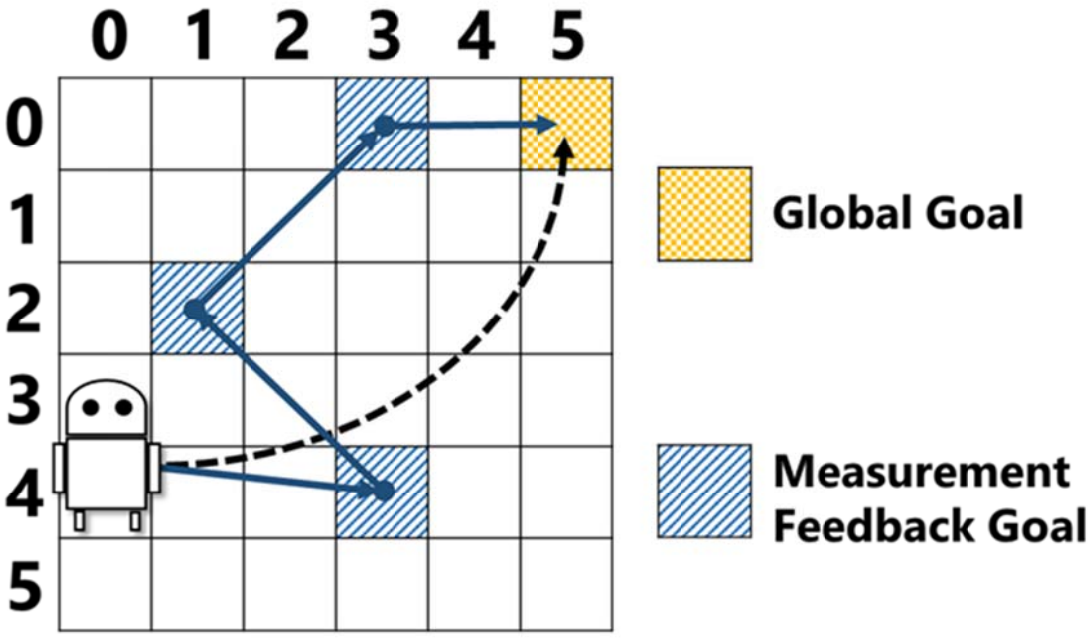

Figure 4-6: The relationship between measurement feedback goals and global goals.

4.1.4 Global goal example algorithm: least recently visited

For the prototype implementation of the cloud robot a general area coverage algorithm was decided upon to act as the global goal algorithm. This general coverage algorithm is called “Least recently visited” (inspired by [67]) and has its goal to find an area that the robot has visited least recently. An example of this algorithm can be seen in Figure 4-7.

In the figure, an area is divided into five times five mapcells, which all contains a visiting counter. The visiting counter in every mapcell is incremented every time the robot enters a new mapcell. This means that the higher the visiting counter, the longer time ago the mapcell was visited. In the first step, the robot’s current position is at mapcell(0)(4) and the mapcell that has been least recently visited is mapcell(4)(0), since it has the highest visiting counter.

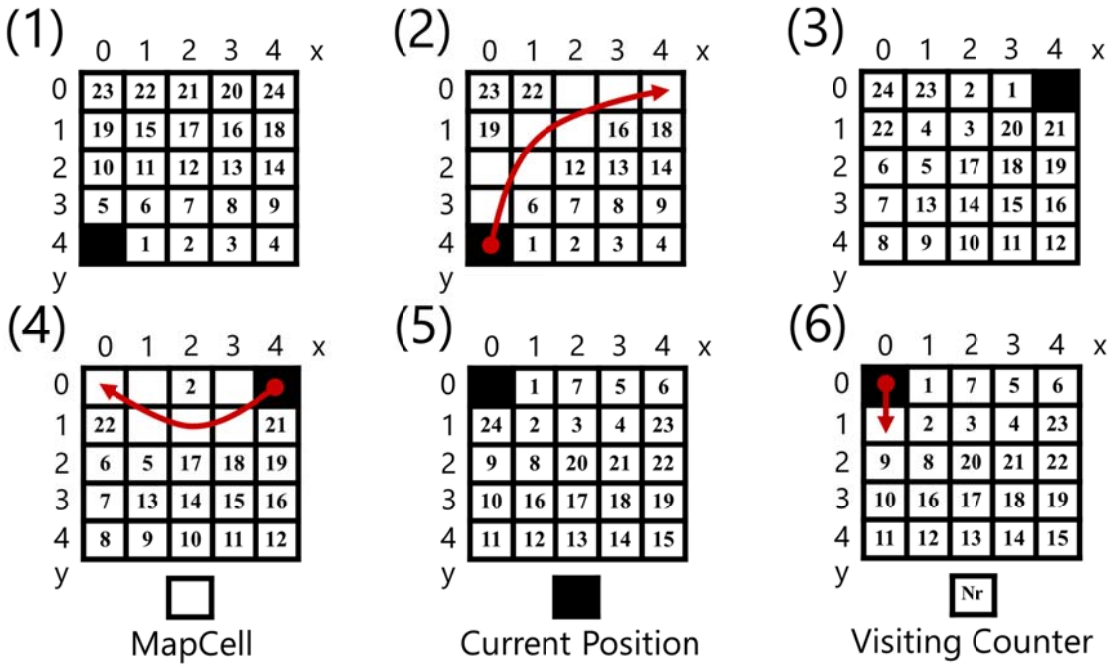

Figure 4-7: An example of a global goal algorithm to decide a major task.

In the second and third step the robot starts moving towards the *global goal* set at mapcell(4)(0) and continuously increment the visiting counters of every mapcell when entering a new mapcell. When the robot finally arrives at mapcell(4)(0), all the visiting counters are recalculated and a new *global goal* is selected. This time mapcell(0)(0) has the highest visiting counter and is therefore the mapcell that has been least recently visited. This continues until the global_goal_manager, briefly described in section 4.1.1, decides on another global goal policy. The robot may do local task while moving to another least recently visited mapcell.

Another more intuitive way of achieving the same behavior is by putting a timestamp in the previous visited mapcell when moving into another mapcell. The least recently visited mapcell will then be the mapcell with the earliest timestamp. The benefit of this is that you only need to update the timestamp of one mapcell when moving to another, as opposed to the proposed algorithm where all mapcells need to be updated every time the robot enters a new mapcell. Implementation of such an algorithm is left for future work.

4.1.5 Measurement feedback goal example algorithm

In order to create a cloud robot that could satisfy the use-case presented in the beginning of section 4.1, a measurement feedback algorithm was implemented that has as its goal to search for abnormal temperature values on the path to a *global goal*. If an abnormal temperature value is found, the algorithm aims to locate the temperature hotspot; the origin of the hotspot. An example of the algorithm can be seen in Figure 4-8.

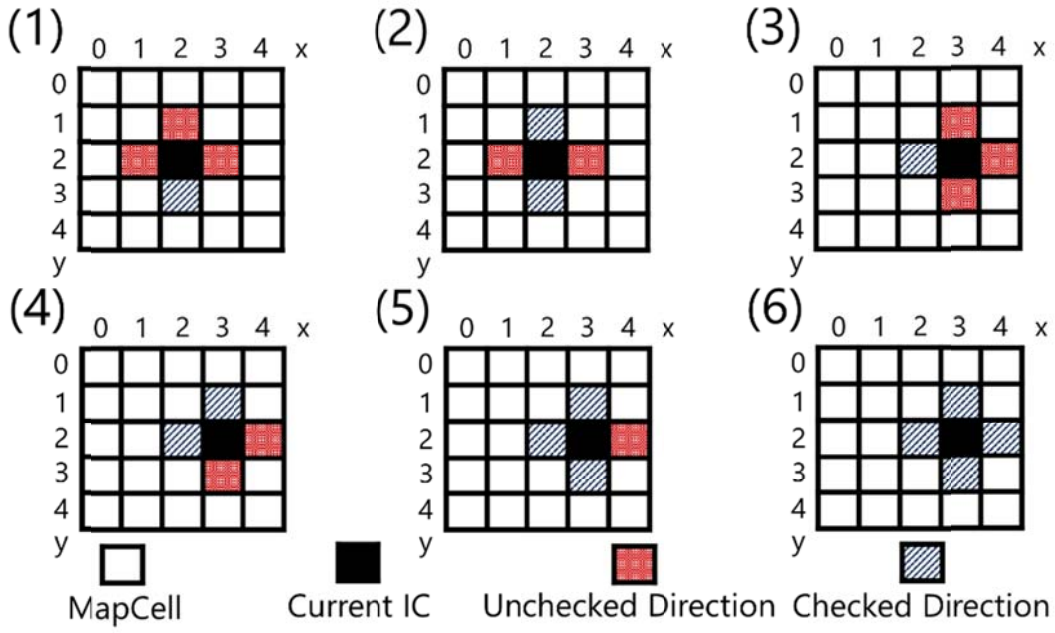

Figure 4-8: An example of an algorithm that decides a sub-task to the major task.

An area is once again divided into a grid of five times five mapcells. The first stage of the algorithm is to scan for abnormal temperature values, which if found would indicate that a hotspot has formed somewhere in the datacenter. As seen in (1) in Figure 4-8, when traveling from mapcell(2)(3) to mapcell(2)(2) the robot finds an abnormal temperature value. By finding the abnormal temperature value, the algorithm enters its second stage which is to locate the origin of the hotspot. The second stage starts by assigning a "current investigation cell" (Current IC) to the mapcell where the abnormal value was found; mapcell(2)(2). The algorithm then continues to mark surrounding mapcells as "unchecked directions" and the mapcell visited before finding the abnormal value (mapcell(2)(3)) as "checked direction"; since it had a lower average temperature.

As can be seen in (2) in Figure 4-8, the robot will then start to check surrounding mapcells for their average temperatures. If a surrounding mapcell has a lower average temperature than the "current IC", that mapcell is marked as a "checked direction". However, in (3) the robot finds that the average temperature of mapcell(3)(2) has a higher average temperature then the "current IC". The algorithm will now assign mapcell(3)(2) to be the new "current IC" as well as mark the previous IC as a "checked direction". This continues until all surrounding mapcells of a "current IC" has been checked as "checked directions"; meaning that the hotspot has been found since all surrounding mapcells have lower average temperature.

4.1.6 Merger module

When implementing the cloud robot, a problem was discovered when trying to synchronize the robot position to a temperature measurement. The problem can be best described by Figure 4-9. It turns out that the temperature sensor used, *Kestrel Meter 4000 Weather Meter*[*], can only publish temperature measurements at a rate of 3Hz while the position module can publish position measurements at a much higher rate.

[*] http://kestrelmeters.com/products/kestrel-4000-weather-meter

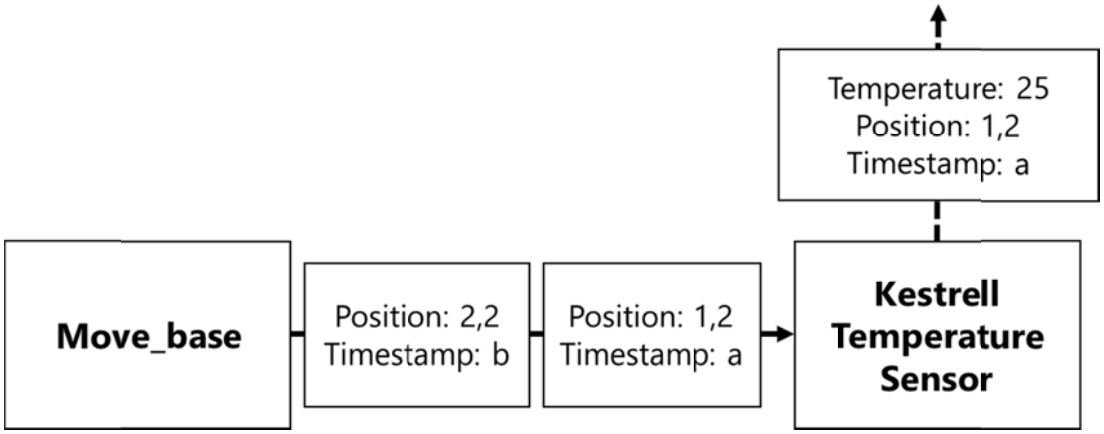

Figure 4-9: Merging position and temperature data based on the newest position update.

Illustrated in Figure 4-9, the move_base module publishes position measurements at a higher rate than the Kestrell Temperature Sensor. If the position measurement and temperature measurement is merged in the Kestrell temperature Sensor module before being sent to the mapcell_manager, it becomes hard to know which temperature measurement that corresponds to what position. Even though the temperature measurement might have been taken at position (1,2), the newest position value to arrive at the Kestrell module will be (2,2). This makes it so that the temperature value is merged together with a position where it was not taken.

What can be done to prevent this, and how it was implemented in the prototype, can be seen in Figure 4-10. In this case another module acts as a merger of the two types of measurements. Both the move_base and the Kestrell Temperature Sensor will publish their measurements with a timestamp. The merger module will then merge the position and temperature measurements that have the closest timestamps and forward this to the mapcell_manager. In the case seen in Figure 4-10, timestamp “c” of the temperature measurement is closer to timestamp “b” than timestamp “a”. The merger module therefore merges temperature 25 together with position (2,2) and forwards it in newly crafted message.

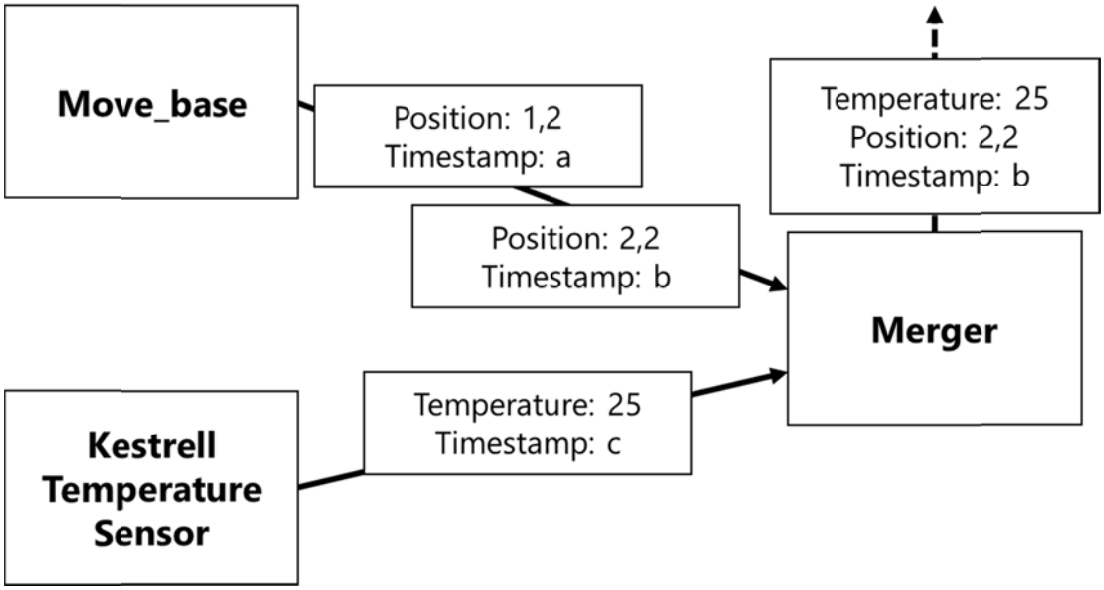

Figure 4-10: Merging position and temperature data based on time stamps.

4.2 Network-Aware Cloud Networked Robot

The cloud robot prototype presented in the chapter 4.1 was used as a base in the development of a network-aware cloud robot. Building upon the architecture seen in Figure 4-3, this chapter discusses the major reasons behind design choices and the implementation of network-aware functionality onto the existing architecture in order to achieve the goals specified in Section 1.4. An overview of the proposal network-aware cloud robot system can be seen in Figure 4-11.

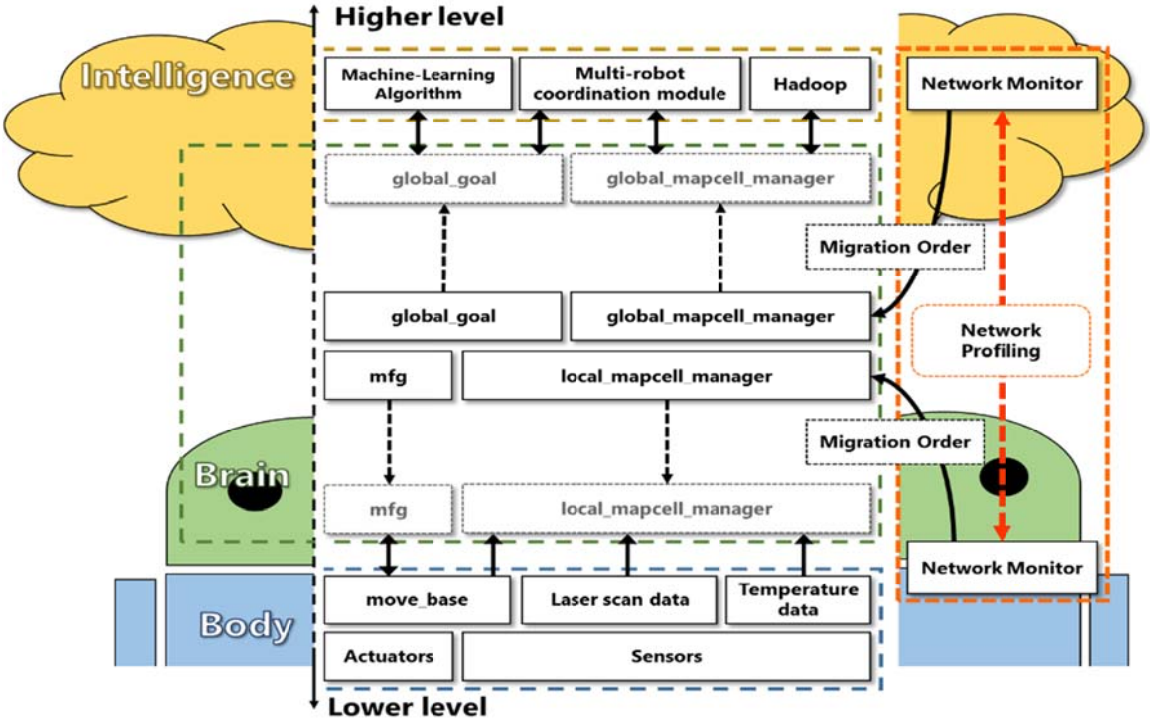

Figure 4-11: An overview of the proposal network-aware system design.

As was presented in chapter 4.1.1, the "brain"-part of the system has modules that can be migrated between the cloud and the robot. Working as an extension of the existing cloud robot prototype, network monitors are placed on both the robot and the cloud. These network monitor modules perform continuous profiling of the network condition. These modules can also access information about the required quality of service for each module in the "brain"-part of the system, and if needed, issue a migration order to the affected module. Such a migration order will result in the affected module having to migrate to either the cloud or the robot depending on its current position.

4.2.1 Migrating modules in ROS

The idea with ROS, and other component oriented frameworks, is to start all nodes and define all node relationships at the beginning and then refrain from making any major changes until the shutdown of the system. However, when moving this kind of system into the realm of unstable network environments, a need for the migration of modules becomes apparent. This thesis proposes two possible methods to do module migration in ROS, seen in Figure 4-12 below.

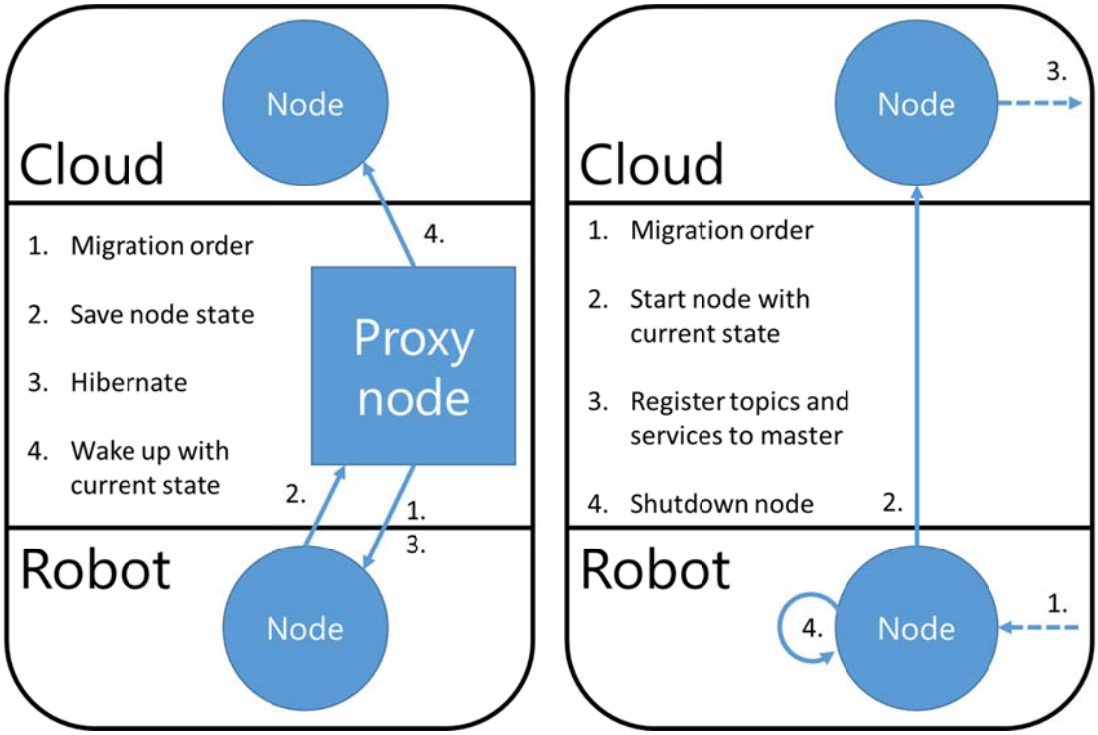

Figure 4-12: Two methods of migrating a module in ROS.

The figure depicts two examples where a network monitor issues a migration order to a node, currently placed locally on the robot, to migrate to the cloud. The first method, seen to the right in the figure, is the easiest to implement. The method consists of four steps to complete the migration. Firstly, a migration order is sent from the local network monitor to the node on the robot, asking the node to migrate to the cloud because of a recent change in the network condition. The node on the robot will then save its current state and start a clone of itself remotely on the cloud with its current state. Launching a node from another node can be achieved by using a process management API of your choosing or using rosspawn*, which can provide that kind of service from within ROS. The newly created node on the cloud then needs to register all the topics and services that it intends to use to the master. Lastly, the node on the robot will shut down so to not disturb the node on the cloud.

The second method, seen to the left in the figure, is probably the more efficient but more resource consuming method, since it requires the state of the entire node to be left in memory. Here, two nodes of the same kind will run simultaneously, although one of the nodes will be hibernating. A network monitor will send a migration order to a proxy node that will forward the migration order to the node on the robot. The node on the robot will save its state and send it to the proxy, upon it will receive an order to hibernate. The proxy node will then order the hibernating node on the cloud to wake up with the saved state.

It was decided that the method of using a proxy node would take too long to implement. Therefore, the method where you startup and shutdown nodes was selected for the implementation of network-aware functionality onto the existing cloud network robot architecture. These two ways of migrating modules are parts of the clone-based model of CNR presented in Section 2.1.3. For this to work, all the message types, manifests, and

* More info about this library can be found at http://wiki.ros.org/rosspawn.

other ROS elements described in Section 2.2.1 that are needed by the programs, has to be read and put on the cloud clone before starting the modules.

4.2.2 Limitations of the cloud robot prototype

After working with (and observing) the cloud robot prototype in order to develop network-aware functionality, it became apparent that the cloud robot prototype and ROS posed some important limitations that had to be taken into consideration.

The first important limitation is depicted in Figure 4-13 and has to do with the inconsistency in path selection of the cloud robot. The cloud robot prototype uses the move_base module (see Section 2.3 for further details) in ROS to create paths, do localization, and mapping of the area. Even though the robot is to start from one position in the corridor and move to another, the move_base module will choose a path from many different variables, which are almost impossible to predict going by experience. That is not a problem, however, when the robot moves closer to objects or walls it tends to use more bandwidth, since it needs to send more data in order to calculate a local path to avoid collision. Illustrated in Figure 4-13, the robot chooses two different paths to complete the same task, moving from the black dot to the white dot. The problem as stated above is that the path of the dotted line will use more bandwidth. Therefore, it is hard to predict how much bandwidth the robot will use, even for the simplest tasks.

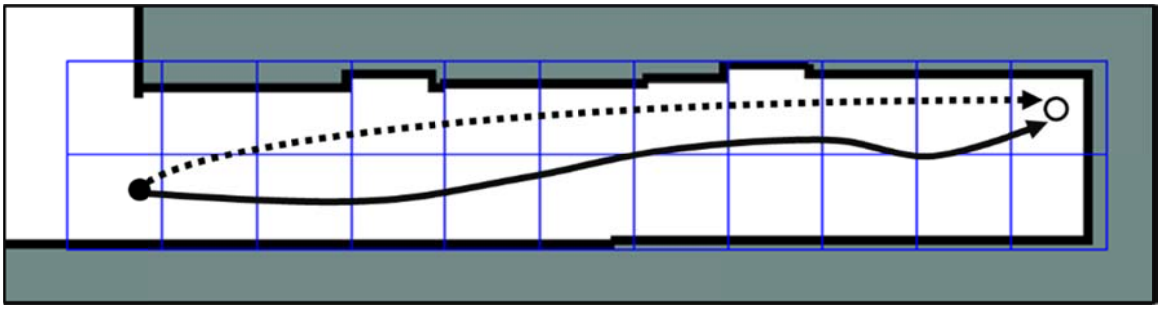

Figure 4-13: The robot taking two different paths, from the same origin to the same goal, in the corridor of the laboratory.

Another limitation of ROS is that topics and services (see section 2.2.3) behave quite differently under different network conditions. Table 4-1 shows the summary of results from an experiment to measure the round-trip delay time (RTT) between a client node and a monitor node (server), when using topics and services with different delays and node placements. The four different node placements scenarios can be seen in Figure 4-14. Seen in the upper left of the figure is a scenario where the client node (“C”) is in the robot and the master (“Ma”) and monitor (“M”) nodes are on the cloud. Next to the upper right is a scenario when the client and master nodes are on the robot and the monitor is on the cloud. To the lower left is the scenario where the monitor and client nodes are on the robot and the master is on the cloud. Lastly, to the lower right, is a scenario where there are master nodes in both the cloud and the robot and the client and monitor nodes are in different places. The full table of results can be seen in Appendix II.

The summary of results from this experiment, seen in Table 4-1, shows that services introduces a substantial amount of extra delay in RTT for every scenario compared to topics. The reason for this is that when publishing a message on a topic, the publishing node will retrieve the list of subscribers of a topic at the master node only once and then

buffer this information. However, a node that tries to utilize a service repeatedly needs to lookup the list of service providers at the master node every time service call is executed.

Table 4-1: Summary of results when measuring the RTT between two nodes in different settings and using either topics or services.

Services (ms)	0ms M-M	50ms M-M	0ms C-Ma	50ms C-Ma	0ms C-Mo	50ms C-Mo	0ms Multi	50ms Multi
Average	22.45	291.70	14.40	183.50	12.45	118.80	14.05	163.90
Std.Dev	4.73	28.59	1.64	35.22	3.36	8.46	3.32	2.85
Std.Err	1.06	6.39	0.37	7.88	0.75	1.89	0.74	0.64
Topics (ms)	**0ms M-M**	**50ms M-M**	**0ms C-Ma**	**50ms C-Ma**	**0ms C-Mo**	**50ms C-Mo**	**0ms Multi**	**50ms Multi**
Average	2.70	57.95	3.15	52.35	0.00	0.00	2.25	53.45
Std.Dev	1.56	20.23	2.08	0.75	0.00	0.00	0.44	2.58
Std.Err	0.35	4.52	0.47	0.17	0.00	0.00	0.10	0.58

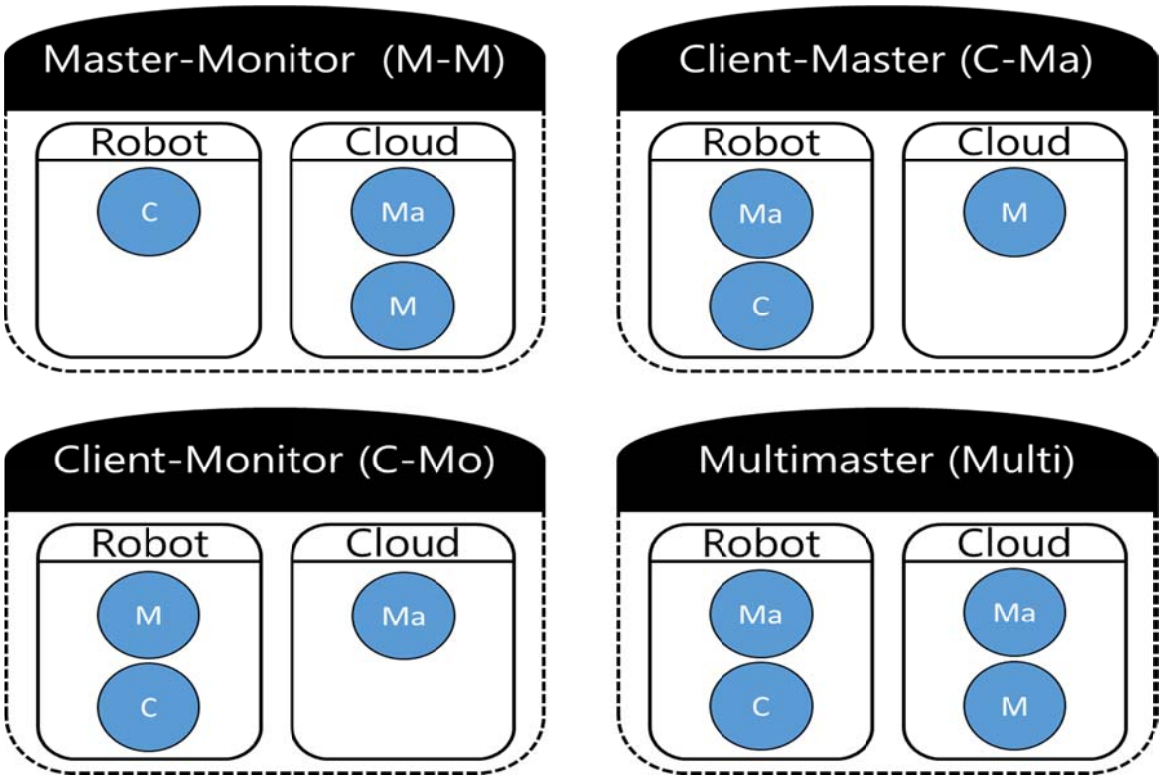

Figure 4-14: The four different node placements in the experiment to measure RTT in different scenarios.

Based on the information about services and topics that was gained from the experiment described above, another minor experiment was done in order to see how delay and the two scenarios, single-master and multimaster, affects the startup time of a resource consuming module. The module that was chosen for this experiment was

move_base and the full table of results can be seen in Appendix III. What can be concluded from the summary of results, seen in Table 4-2, is that the move_base module has a minimum startup time of 5.3 seconds. Moreover, using only a single master results in an exponential growth in startup time when the delay of the network is increased. The startup time of a minimum of 5.3 seconds has to be taken into account when evaluating the network-aware cloud robot with the chosen migration technique that uses startup and shutdown of nodes. What can also be noted is that the move_base module would take several minutes to start when using a single master while only a few seconds to start when using a multimaster, when the network is subject to high round-trip latency. Due to the nature of unstable network environments, where the robot can be connected to the cloud at one time only to lose the connection the next moment, the use of only a single ROS master would make the system stop every time the robot loses connection to the cloud. Therefore, in order to develop network-aware functionality upon the cloud robot prototype, the system must use several ROS masters. This can be done for example by using the multimaster library[*]. Lastly, a small experiment was also conducted to stress test the move_base module when it is placed on a cloud. What was found in that experiment is that the move_base module will cease to function, stop giving move commands to the actuators, when the round-trip latency of the network raises to around 160 milliseconds.

Table 4-2: Summary of results measuring the startup time of move_base under different conditions where "SM" stands for "single-master" and "MM" for "multi-master".

Delay (ms)	0ms (SM)	50ms (SM)	140ms (SM)	0ms (MM)	50ms (MM)	140ms (MM)
Average	13414.00	95688.00	230206.00	5350.00	5350.00	6112.00
Std.Dev.	854.01	1555.95	1556.01	236.11	182.21	246.82
Std.Err.	381.92	695.84	695.87	105.59	81.49	110.38

The reason for choosing the move_base module is that it is by far the most resource consuming module in the cloud robot prototype. This was discovered during an experiment to map the CPU usage of all modules. In that experiment, the robot was set to move to the end of the corridor and back. The battery consumption, CPU execution time, and different modules' CPU usage was measured when the modules was placed locally on the robot and remotely on the cloud. The full table of results from this experiment can be seen in Appendix IV. The average bandwidth usage of move_base, when placed on the cloud, is 3.4 Mbit/s. The reason why the amount of bandwidth used by the move_base module is this high, is because the module is transmitting a lot of raw data from the kinetic sensor. This could be improved by both improving the pathfinding, by pre-computing all paths to a goal, and by optimizing the module to only transmit the most necessary data.

Another limitation of the cloud robot prototype is that the temperature sensor is set to measure the temperature three times every seconds. The problem is that the program does not take into consideration that the rate of change of temperature is proportional to the difference between the temperature on the sensor and the temperature of its surroundings. This results in a heat source looking skewed or larger than it might be in reality. This could be corrected by applying Newton's law of cooling in order to calculate the "real" temperature at a certain position. However, there was not enough time to do this and this was left out for future work.

[*] http://wiki.ros.org/multimaster_fkie.

4.2.3 Network-aware machine learning algorithm

The network-aware machine learning algorithm was developed to take the limitation of the system described in 4.2.2 into account, while aiming to satisfy the two sub-goals presented in Section 1.4. The algorithm described in this and following sections of the chapter works to find the most efficient module placement for modules depending on the task, the location where the task is to be executed, and the initial network condition at the location; satisfying sub-goal number 2. A task can, as previously described in chapter 4.1.3, be all from scanning a part of a room for abnormal temperature values to just moving the robot from one point to another.

The benefit of having the network-aware machine learning algorithm try to find the best placement policy, instead of having only a continuous evaluation of the network condition and migrating modules on the fly, is that you can achieve an overall more efficient and better migration policy. If modules are migrated every time on the fly when the network condition get better or worse than a certain limit, you might end up with a lot of unnecessary migration. Since migrating modules is expensive, you might end up with a less efficient system than if you do not migrate any modules at all. By trying to find the most efficient module placement while keeping the number of migrations during a task relatively low, the algorithm satisfies sub-goal number 1 as specified in Section 1.4.

An overview of the proposed network-aware machine learning algorithm can be seen in Figure 4-15. Before the start of a task the network is being constantly monitored by a network profiler. The network profiler estimates the condition of the network and parses this into two variables, A_{bw} and C_l, which are forwarded to the machine-learning algorithm. The machine-learning algorithm lookups the training set for the specific task at the specific location. It then executes a classification algorithm to come to the conclusion if the modules need to be migrated, or not, before the start of the task. This means that the module placements will be static during the execution of the task. The migration decision could include the options to say “migrations are only allowed x amount of times”, in order to enable modules to migrate on the fly if the network goes down when a crucial module is on the cloud. This however would take too much time to implements and was left out for future work.

After the end of a task comes a phase in the algorithm which evaluates if the migration decision was good or not. After an evaluation of the migration decision has been done, the decision evaluation module inserts the newly gained data into the training set for the specific task at the specific location, which makes the algorithm come full circle from making a decision and then as time goes learns by classification and newly gained data.

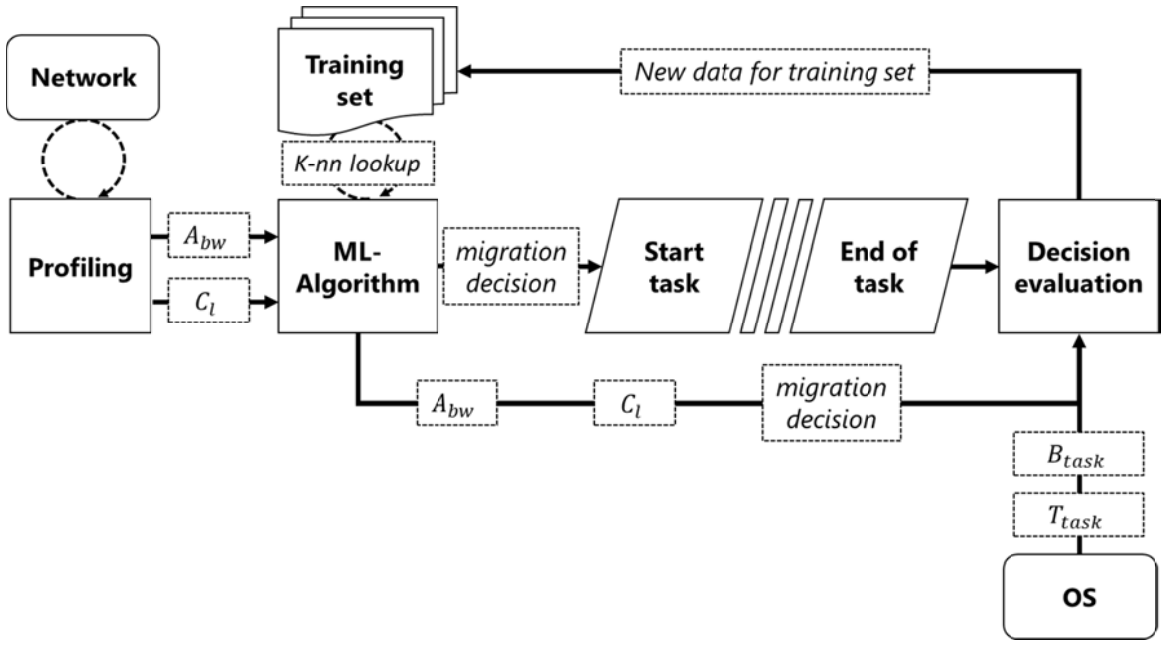

Figure 4-15: The machine learning algorithm used to enable network-aware functionality in the cloud robot prototype.

4.2.4 Network profiling

The first phase in the machine-learning algorithm is acquiring information about the current state of the network. This is done by doing continuous profiling of the network. Usually this can be done by measuring the latency and available bandwidth. However, as discussed in section 4.2.2, the cloud robot requires different amount of bandwidth to operate depending on the route taken to the goal. Therefore, instead of just looking at the currently available bandwidth of the network, or the currently used bandwidth of the robot, a new ratio called "bandwidth ratio" (R_{bw}) is introduced. This new ratio gives a better overview of how much more bandwidth the robot can use before filling up the network. An explanation of this ratio can be seen in Equation 4-1.

$$R_{bw} = \frac{A_{bw}}{C_{bw}}$$

Equation 4-1: The relationship between the bandwidth ratio and available bandwidth and currently used bandwidth.

In this equation the available bandwidth (A_{bw}) is the currently maximum available bandwidth of the network, and the current bandwidth (C_{bw}) is the bandwidth currently used by the robot. A way to picture this ratio can be seen in Figure 4-16. The reason the ratio is derived this way, and not for example in the inversed way, is that the algorithm described in Section 4.2.5 works better when the ratio and latencies are on a similar scale. Choosing the ratio to be expressed in this way makes it get closer in scale to the latency axis after put through a normalization process. The bandwidth ratio is expressed in percent and indicates how many more percent of C_{bw} is left from the available bandwidth. So the information that is needed by the machine-learning algorithm from the network profiling phase is: the current latency (C_l), the currently used bandwidth (C_{bw}), and the currently

available bandwidth of the network (A_{bw}). An overview of the whole network profiling phase and the programs used to acquire the required information can be seen in Figure 4-17.

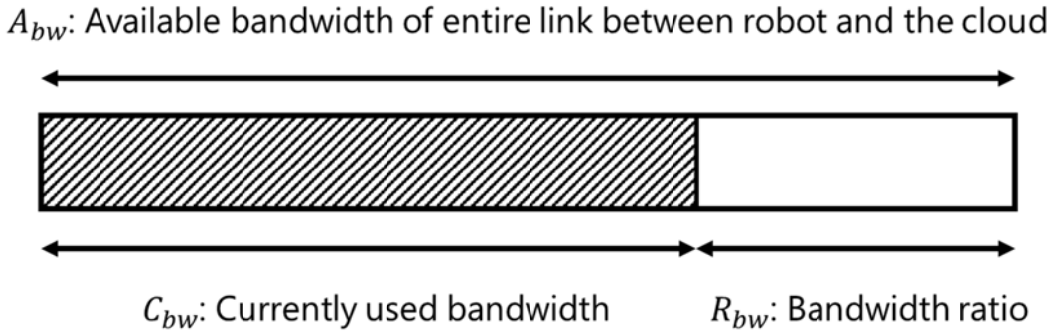

Figure 4-16: A descriptive schematic of the bandwidth ratio in relation to the currently used bandwidth and the currently available bandwidth.

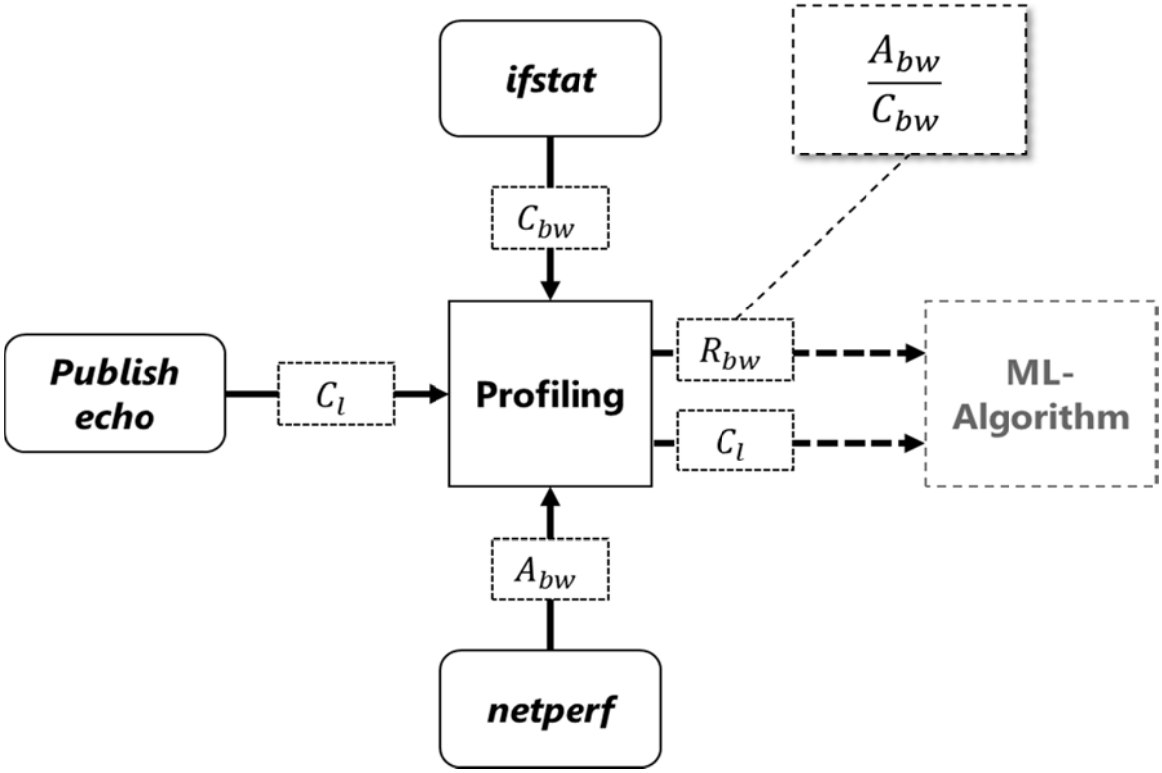

Figure 4-17: Overview of which programs that are used to acquire information about the network condition and how this information is parsed for the machine learning algorithm.

The current latency (C_l) is acquired by measuring the round-trip delay time (RTT) between the robot and the cloud. The RTT was measured by a publish echo program that was created by the author. This publish echo program runs on top of the publish-subscribe messaging paradigm, and echoes messages containing timestamps. An overview of the procedure of the program can be seen in Figure 4-18. Here, the program will receive a request for the current latency by the network profiling module. It will then continue by taking a timestamp and include this in a message which is published on a topic to a node

on the cloud. When the message arrives to the module on the cloud, that module will proceed to echo the message back by publishing it on a topic to the node on the robot. When the message arrives to the node on the robot, the node takes another timestamp and subtracts the time of the timestamp in the message. The calculated current latency is then sent back to the network profiling module.

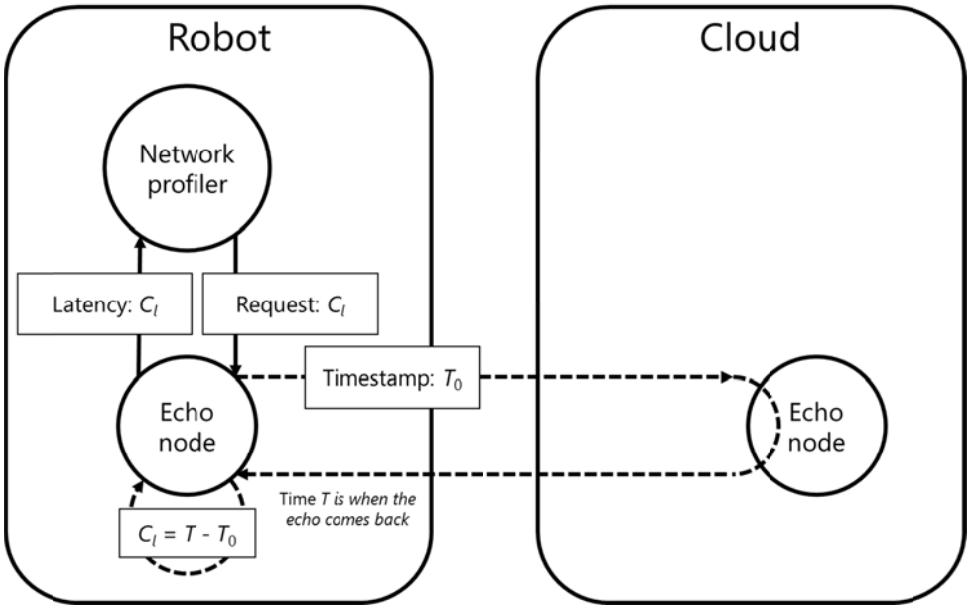

Figure 4-18: Simple overview of how the publish echo program calculates the current latency.

The currently available bandwidth of the network (A_{bw}) and the currently used bandwidth of the robot (C_{bw}), are acquired by using two external programs called *netperf* [64] and *ifstat* [65]. *Ifstat* is a program that reports interface statistics back to a user. It does this by using operating systems drivers to gather statistics. The driver that is used in the experimental test bed is the "*/proc/net/dev*" file. *Netperf* is a network performance benchmarking program developed by the HP information networks division. An example of the bash commands that can be used to acquire the information from these two programs can be seen in Table 4-3. The actual code used for acquiring this information from the two programs can be seen in Appendix V.

Table 4-3: Two examples of how to acquire the currently used bandwidth (C_{bw}) from ifstat, and the currently available bandwidth (A_{bw}) from netperf, using the command line.

bash$ ifstat –i wlan0 0.1 1
bash$ netperf –H 192.168.0.2

4.2.5 Migration decision using k-nearest neighbor algorithm

It was decided that a migration decision will be selected with the help of a classification algorithm named *k-nearest neighbor* (*k-nn*) because it is easy to implement, compatible to adding of new data to the training set, and because of the results presented in [59]. However, before explaining the details of the classification algorithm, it is important to specify the structure of the training set.

The idea is, as specified in chapter 4.2.3, that there exists a tuple $\langle task, location \rangle$ for every pair of specific task and specific location, and that each such tuple has a training set. Such a training set is built up like a database of rows, where each row represent the prerequisite information, migration decision, and evaluation parameters of a run of the specific task at the specific location. The specification of a row can be seen in Table 4-4. The first two parameters of a row are the parameters received during the network profiling phase. Then comes the migration decision of that specific run, and lastly, three parameters that are used during the evaluation phase as well as for data analysis. The benefit of these training sets are that they can be uploaded to a world-wide knowledge database (such as described in [26]) of training sets, which can be downloaded by robot that wants to perform such a task at a similar location. By doing this, new robots does not need to rediscover this knowledge since another robots may have already done the same task at a similar location.

Table 4-4: The structure of a row in a training set.

$\boldsymbol{<R_{bw}>}$ $\boldsymbol{<C_l>}$ ***<Migration Decision>*** ***<Mission Status>*** $\boldsymbol{<B_{task}>}$ $\boldsymbol{<T_{task}>}$

The parameters that are used for making a migration decision are the first three: R_{bw}, C_l, and *Migration Decision* (MD). In Figure 4-19 is an example of what the *k-nearest neighbor* classification algorithm tries to do with a training set. In this example, which is not actually accurate but rather tries to make it easier to explain, the whole graph is the training set and each point is a row in the set. Each dot has a color, black or blue, which represents if a module should be offloaded or not. If a migration decision needs to be made at a place in the graph where there is no dot, selecting a row in the training set would not solve the problem. What is done instead is that the *k-nearest neighbor* algorithm classifies an area in the graph to either output the migration decision onload (when in the white area) or offload (when in the blue shaded area). As more data is added to the training set (the graph), the classification becomes more accurate.

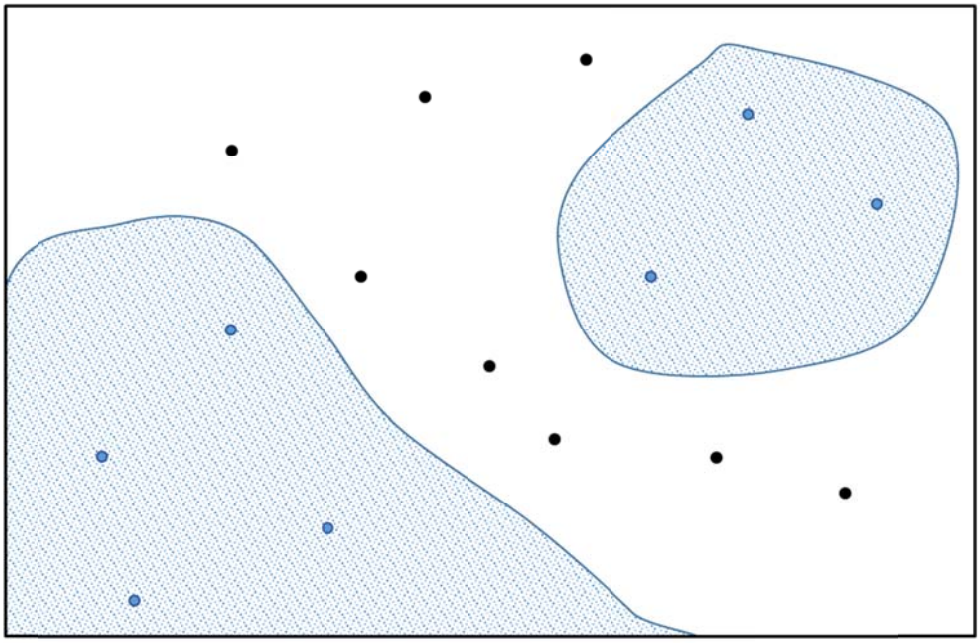

Figure 4-19: An example of what the k-nn classification algorithm tries to do.

Calculating the migration decision (*MD*) can be seen as an optimization problem of the triple $MD(x, y, z)$, where x is C_l, y is R_{bw}, and z is the migration decision onload when equal to zero and offload when equal to one. Each row in the training set can be described as seen in Equation 4-2. A row is here specified as a quadruple of variables where x_n is the C_l value of row n, y_n is the R_{bw} of row n, z_n is the MD of row n, and w_n is the Euclidean distance between the two points $P(x,y)$ and $P(x_n,y_n)$.

$$R(x_n, y_n, z_n, w_n), \qquad D(n) = [0, N_r - 1], \qquad N_r: Number\ of\ rows\ in\ the\ training\ set$$

Equation 4-2: The definition of a row in the training set.

The implementation of the *k-nn* algorithm in this project aims to find the answer to z by finding the Euclidean distance between the two parameters, R_{bw} and C_l, and every row in the training set. The Euclidean distance between the migration decision and row n in the training set (w_n) can be seen in Equation 4-3.

$$w_n = \sqrt{(x - x_n)^2 + (y - y_n)^2}$$

Equation 4-3: The Euclidean distance between the current run and row n in the training set.

The specific implementation of *k-nn* in this project uses $k = 5$ since this was a good fit for the data after some trial and error. This means that the five rows with the smallest Euclidean distance from the current run are selected from the training set and put into another quadruple set, $R(x_m, y_m, z_m, w_m)$, where $D(m) = [0, k - 1]$. A majority vote is then held between these five rows with the five smallest values to w_n, to decide if the migration decision of the current run should be offloaded (z equals one) or onloaded (z equals zero).

The equation of the majority vote can be seen in Equation 4-4. This equation in conjunction with the specifications and equations above states that if there are more rows with a migration decision of offload, than rows with the migration decision onload, in the vicinity of the current migration decision (*w* in *MD(x,y,z)*), then the current migration decision should be offload. The other way around means that the current migration decision (*w* in *MD(x,y,z)*) should be onload.

$$z = \begin{cases} 1, & \sum_{m=0}^{k} z_m > \frac{k}{2} \\ 0, & \sum_{m=0}^{k} z_m < \frac{k}{2} \end{cases}$$

Equation 4-4: Calculating the migration decision by majority vote.

4.2.6 Decision evaluation and evaluation parameters

After a task has ended, it is essential to evaluate if the decision that was made before the start of the task was right or wrong, or in some cases less wrong than right. However, to do an evaluation, there is a need to specify what information to use in an evaluation. It was decided that an evaluation is to be based on the time it took to execute the task (T_{task}), and how much battery the task consumed (B_{task}).

As the OS of choice in this project was an Ubuntu Linux distribution, these parameters can be calculated by reading specific files in system and process directories. More specifically, the time it takes for a task to execute (T_{task}) can be calculated by reading the */proc/stat* file [68] before the task started and once again after the task has ended. The total time in jiffies, or "USER_HZ", can be obtained by adding together all numbers after the *cpu* line in the file, in order to create an accurate timestamp. The total execution time of the task is obtained by subtracting the timestamp from before the start of the task with the timestamp after the end of the task. It is possible to calculate the battery consumption of the task by reading the two files *charge_full* and *charge_now* under the */sys/class/power_supply/BAT0/* directory, before the start and after the end of a task. By dividing the number in *charge_now* with the number in *charge_full* you can get a more accurate reading of the battery level in percent, before and after the task. Subtracting the battery level after the task with the battery level before the task gives the change in battery charge, hence the battery consumption in percent.

The next problem is what to compare these two parameters against to make an evaluation. Because of the lack of time, this project focuses on evaluating decisions to offload modules before a task and leaves out the evaluation of migration decisions to put modules locally on the robot. In the case where you want to evaluate a decision to offload modules, the answer to what to compare the two parameters against would be the average of the corresponding parameters when having the module on the robot. In order to get this information, a small experiment was conducted to create two control values; one control value for the battery consumption (CB_{task}) and one control value for the task completion time (CT_{task}). The experiment was run on top of the test bed specified in chapter 3.4 while having all the modules locally on the robot. The task presented to the robot was to run to the end of the corridor and back to the beginning. The experiment consisted of twenty runs

and the summary of results of this experiment, when the CPU frequency was set to 1.6GHz and power management was disabled, can be seen in Table 4-5.

Table 4-5: Summary of results from having the robot run to the end of the corridor and back with all modules locally when having the CPU frequency set to 1.6GHz.

	CB_{task} – (%)	CT_{task} – (s)
Average	0.004586	87.9286
Std.Dev.	0.000702	14.3815

We now have values that represents the performance of a run and two control values to compare to in order to evaluate if the migration decision was good or bad. The way to compare the acquired values to the control values is still up to discussion, but an example could be to introduce a weight. Since migration in ROS in its current state is quite resource expensive, a weight could be put on the acquired values to say that if the decision performed only 5% worse than the control values it would still be an acceptable decision. Such a weight could be defined as in Equation 4-5 where V_r stands for "validity ratio", which would be 1.05 if a decision is allowed to perform 5% worse than the control values. A decisions that equals one in this figure would mean an acceptable decision, where a decision that equals zero means another decision would have been better.

$$Decision = \begin{cases} 1, & \dfrac{\left(\dfrac{T_{task}}{CT_{task}}\right) + \left(\dfrac{B_{task}}{CB_{task}}\right)}{2} \leq V_r \\ 0, & \dfrac{\left(\dfrac{T_{task}}{CT_{task}}\right) + \left(\dfrac{B_{task}}{CB_{task}}\right)}{2} > V_r \end{cases}$$

Equation 4-5: A decision is allowed to perform worse than the control values in proportion to the validity ratio.

When a decision has been evaluated, the information acquired during the network profiling phase, the migration decision, and the evaluation values are added to the training set of the specific task at the specific location as a new row (see the definition of a row in Table 4-4). This makes the machine-learning algorithm comes full circle since as more data is added to the training set, more accurate migration decisions can be made by the machine-learning algorithm.

5 Analysis

In this chapter the major results from the experiments described in chapter 3 is presented and analyzed. The experiment used the network-aware cloud robot presented in chapter 4. Furthermore, a reliability analysis is performed on the acquired results in section 5.2 and the major results are discussed in section 5.4.

5.1 Major results

In this section the major results of the continuous experiment is presented. The cloud robot with the network-aware functionalities presented in chapter 4 was set running, with the specific task described in chapter 3.4.1, in the laboratory corridor over the span of two months while running the machine-learning algorithm described in 4.2. This resulted in a training set with 572 valid measurement after the data was cleaned. A subset of these 572 entries in the training set was plotted in a graph. Following this, a classification algorithm was run as an overlay, in order to decide if an area should be considered best for offloading modules to a cloud or placing modules locally on a robot. The resulting scatter graph can be seen in Figure 5-1.

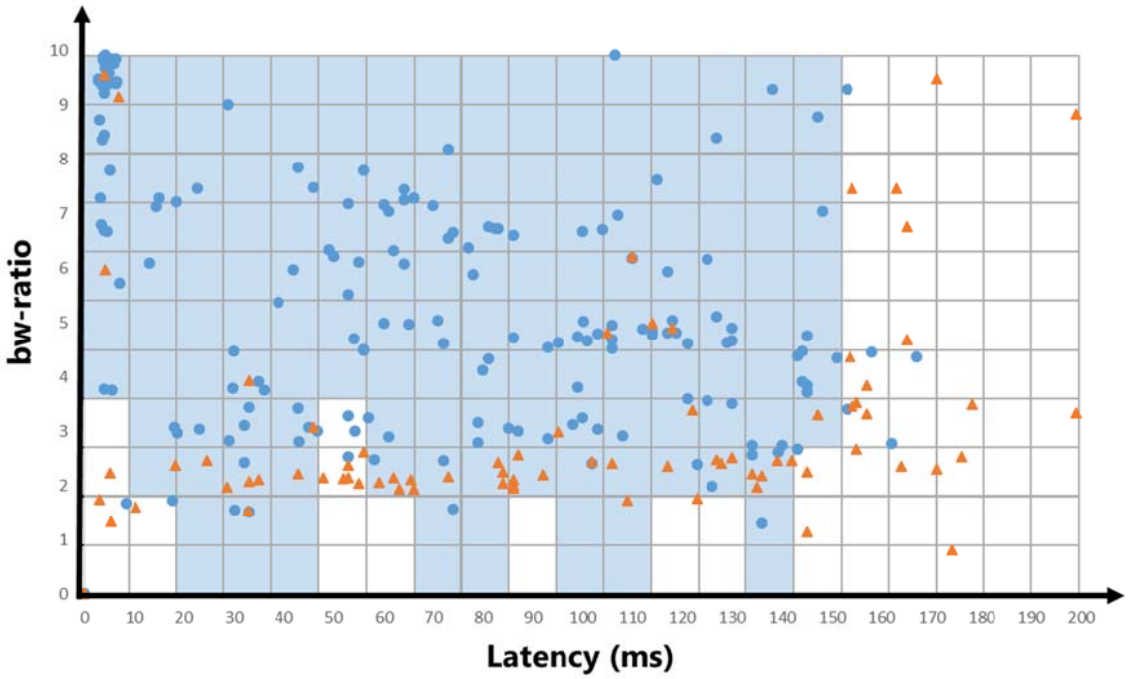

Figure 5-1: Subset of the training set plotted by latency and bandwidth ratio and with an added overlay that displays what module placements is the most efficient.

Here, the x-axis represents the RTT latency of the network and the y-axis represents the bandwidth ratio (C_{bw}). A blue dot signifies a migration decision where offloading was the best decision and an orange triangle represents a migration decision where onloading would have been the best decision. The white areas are where the classification algorithm calculates that onloading modules would be the best decision and the blue area is where the classification algorithm calculates that offloading modules would be the best decision.

The graph seen in Figure 5-1 represents the areas of offloading modules or placing modules locally on a robot based on the validity ratio (V_r) presented in section 4.2.6. This graph only shows a very general picture of the most efficient module placement and it also presents data that is derived from a variable defined by the author. Furthermore, it also does not take the migration time into consideration. Therefore, there was a need to further analyze the data and find the tradeoff between offloading modules and placing modules locally on the robot, when looking specifically on latency and bandwidth ratio. Since most values above 160 milliseconds tend to point to local module placements being the most efficient, all values where the RTT latency was lower than 160 milliseconds were selected for a new subset. The aim of this new subset is to see how the bandwidth affected the battery consumption and task execution time. The graph where these values were plotted to show the relation between bandwidth ratio and the task execution time can be seen in Figure 5-2.

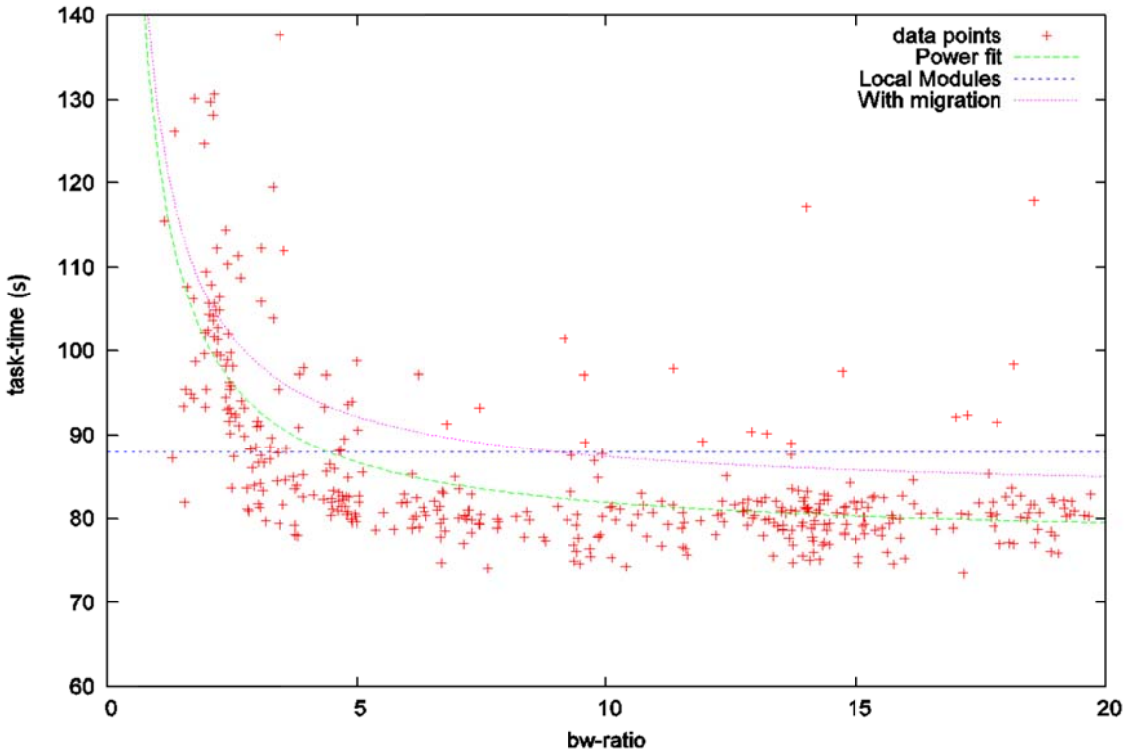

Figure 5-2: Plotting a subset of the training sets task execution time against the bandwidth ratio including trend lines.

The graph shows all values where the latency was lower than 160 milliseconds. The x- and y-axis represents the bandwidth ratio and the task execution time. All values are represented as red crosses and the task execution time control value (CT_{task}), presented in section 4.2.6, is represented by a blue dotted line called *Local Modules*. The values were run through a power fit (because it seemed to fit the data the best) to create a trend line, which is represented as a green dotted line called *Power fit* in the graph. Lastly, a simulation was done on the subset of values to show how the trend line looks when you take the five second migration time (see section 4.2.2, Table 4-2) into account. This trend line is represented as a violet small dotted line called *With migration*.

The subset of values where the RTT latency was lower than 160 milliseconds were also plotted to see how the bandwidth ratio affects the total operating time of the system. This can be seen in Figure 5-3. Here, the values are once again represented as red crosses and the total operation time, derived from the control values presented in 4.2.6, is represented as a blue-dotted line called *Local Modules*. This control line was derived from the two control values by Equation 5-1. The values was also analyzed and run through a line fitting program to create a trend line which is represented as a green dotted line called *Linear fit*. This time the values seem to fit a constant growth line better.

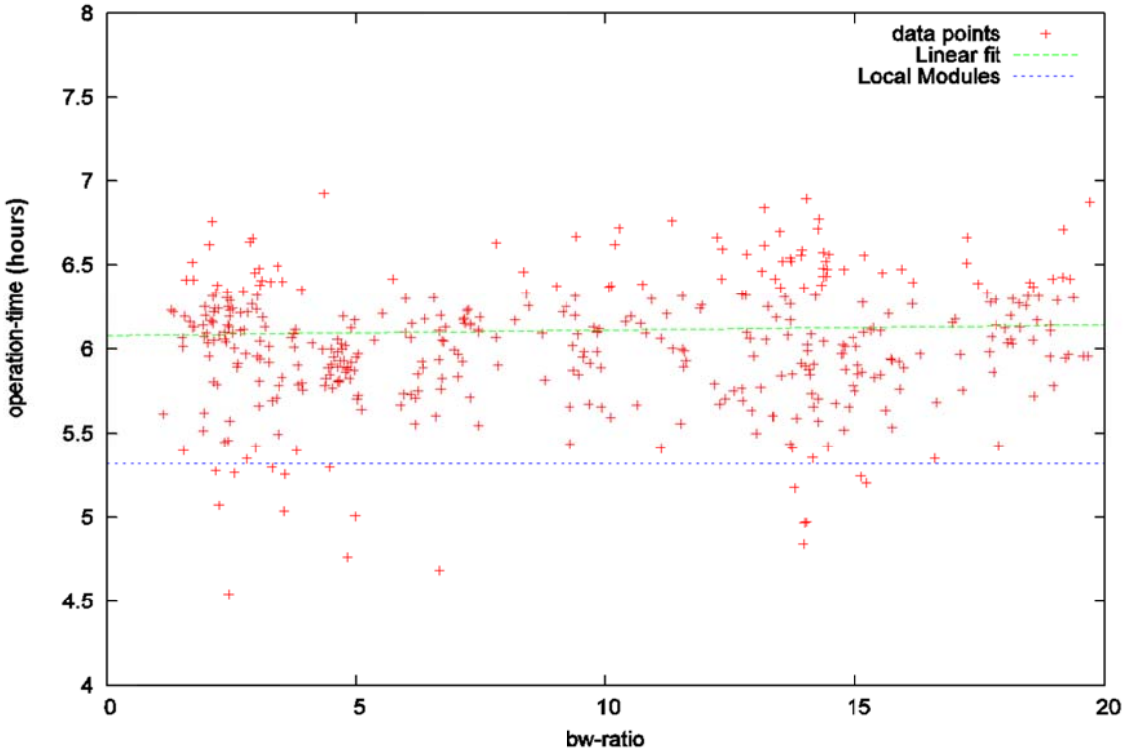

Figure 5-3: Plotting of a subset of the training sets robot total operation time against bandwidth ratio including trend lines.

$$Operation\ Time\ (hours) = \frac{CB_{task}}{CT_{task} \times 3600}$$

Equation 5-1: Deriving the total operation time in hours from the control values.

What can be seen especially in Figure 5-2, is that when the bandwidth ratio gets closer to 10 units, the bandwidth ratio's effect on the task execution time start to pan out. Further analysis was therefore carried out to see how the latency affect the task time execution. The resulting graph when plotting a new subset of the training set, with all values with a bandwidth ratio over 10 units, can be seen in Figure 5-4. The data points are once again represented as red crosses and the task execution time control value (CT_{task}) is represented as a blue dotted line called *Local Modules*. The data seem to have a linear trend and was therefore run through a linear fit program to produce a linear fit trend line, which is represented in the graph as a green dotted line called *Linear Fit*. A simulation was done on the trend line to see how the extra five seconds introduced from module migration

would affect the task execution time. This simulation trend line is represented as a violet dotted line called *With migration*.

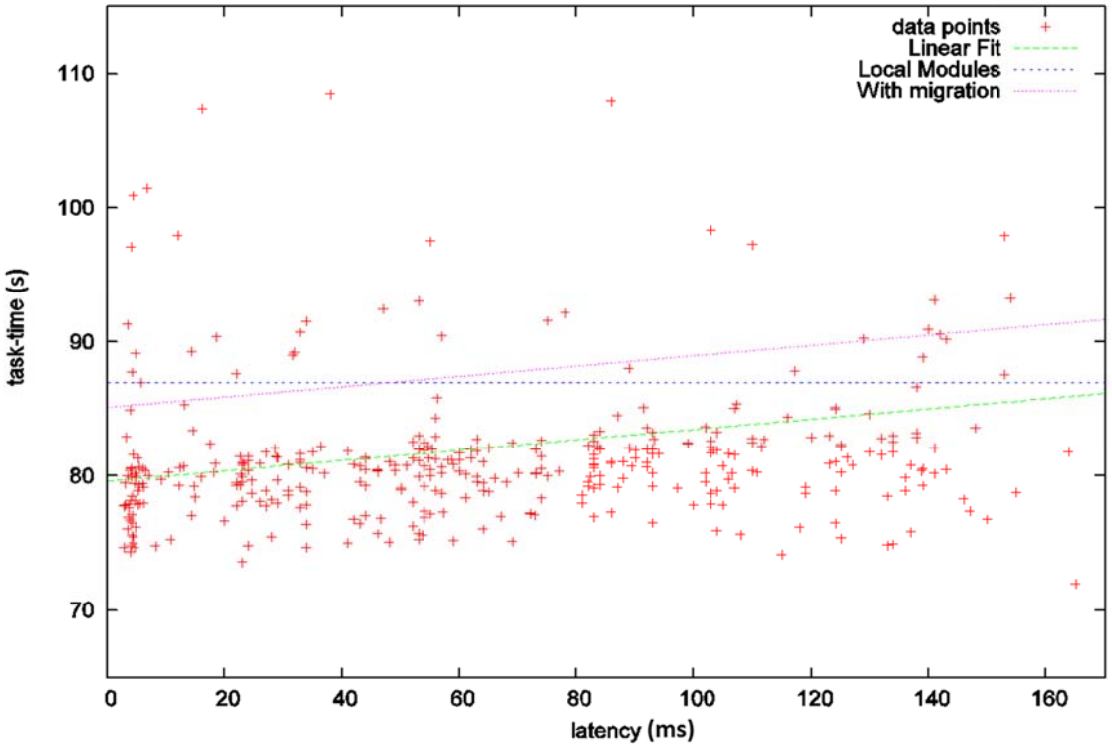

Figure 5-4: Plotting a subset of the training sets task execution time against latency with trend lines.

The previous graphs presents how the performance differs when faced with different network conditions. However, these graphs does not show how much time and battery can potentially be saved by running the modules remotely on a cloud, when the network is working without any set restrictions to latency or bandwidth. The robot was set running the same task another twenty times without imposing any restrictions in latency or bandwidth on the network. The results of these twenty runs when compared to the control values can be seen in Figure 5-5 and Figure 5-6.

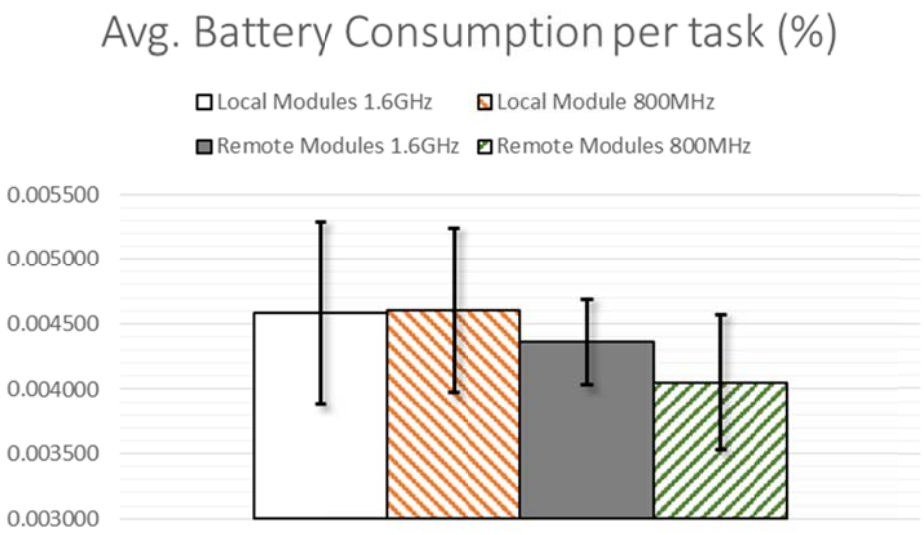

Figure 5-5: Average battery consumption per task comparison between local and remote module placement with different local CPU frequency.

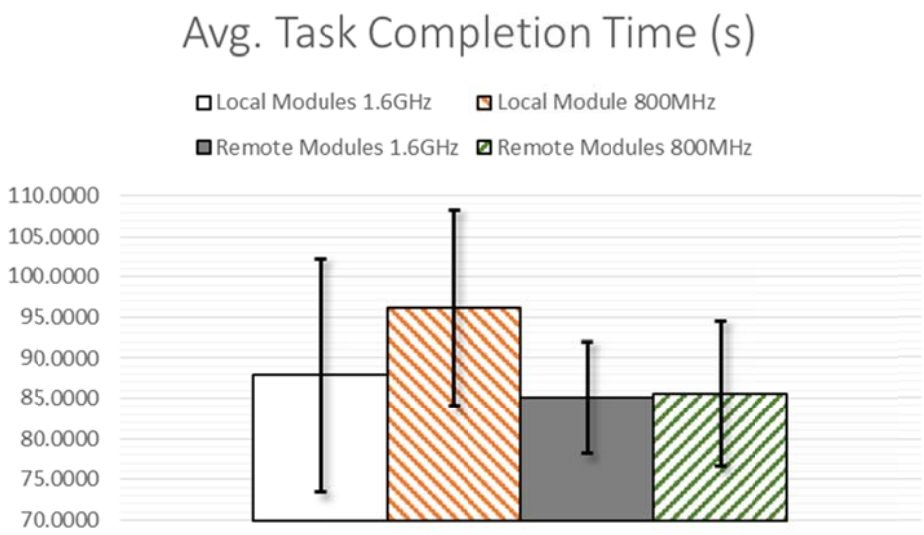

Figure 5-6: Average task completion time comparison between local and remote module placement with different local CPU frequency.

5.2 Reliability Analysis

The results are assumed to be reliable since the experimental setup was created to be as sterile as possible, so that no external variables would influence the results. In order to produce reliable results, the robot was assigned to complete a simple task in order to achieve as many consistent runs as possible. Furthermore, the power management on the laptop was disabled and the CPU frequency was configured to 1.6GHz, in order to eliminate unpredictable behavior when either the battery level gets lower or the load on the CPU changes.

Two variables that need to be taken into consideration, when analyzing the reliability of the results, is the unpredictable pathfinding by the move_base module and the natural behavior of batteries. The charge in batteries sometimes unpredictably drops more than usual and sometimes even recharge. Some test runs with the cloud robot would even show that the robot would recharge during the run because of this. Even though these two variables are unpredictable, they are affecting the system whether the modules are placed locally on the robot or offloaded to the cloud. Therefore, they can be seen as natural elements in the system and the results would reflect the real world much better than a computer simulation.

5.3 Validity Analysis

The validity of the results could not be checked since there was no time to redo the experiments in a real environment of a datacenter, and compare the results from such an experiment to the results from the experiment in the corridor. The reason being that the laboratory was scheduled for a renovation and the data center moved to another building.

5.4 Discussion

By looking at the graph produced by the machine learning algorithm, seen in Figure 5-1, it looks like the proposed solution is actually able to map the most efficient module placement to the network condition. Without specifying an actual limit in either bandwidth ratio or latency, the proposed solution is able to find a border for when placing modules on the cloud will be inefficient, by comparing results to a control value and classifying new values on the fly. However, as can be seen in the graph, there is not nearly enough data points to make a really accurate classification. For example, between 20 and 40 milliseconds RTT latency the graph implies that the bandwidth ratio does not matter and that it will always be beneficial to place the modules on the cloud. This is clearly not true as the move_base module needs a certain amount of bandwidth to function. This error originates from not having enough data points in the vicinity of those areas, which resulted in the classification algorithm being unable to correct the error that is easily found by the human eye.

A question that arises from this problem is that acquiring these 572 measurements took over two months, but how long will it actually take for the algorithm to make plausible decisions? This may seem like a trivial matter, but it is not practical for a robot to take several months to learn even such a simple task. However, this can be integrated with a shared-knowledge platform, as presented in [26], so that another robot in a similar setting can download this data and learn from it; making it so that new robots does not have to take several months to function efficiently.

When looking more closely at the graph in Figure 5-2 it seems like as the bandwidth ratio grows larger, the trend line moves closer to what looks to be an asymptote around 78 seconds task execution time. On the left part of the graph, at around a bandwidth ratio of 4.9 units, the trend line (labeled “Power fit” in the graph) crosses the control line (labeled “Local modules” in the graph). The point where the two lines cross acts as the border between when modules should be placed on the cloud and when modules should be placed on the robot, when running a certain task at that specific location with the specific module configuration. However, this trend line does not take the migration time into account. Instead, when looking at the simulated trend line (labeled

“With migration” in the graph), where the 5 seconds migration time is taken into account, the graph shows a not so promising result. The simulated trend line and the control value line crosses at around a bandwidth ratio of 10 units; more than double the amount of where the trend line without migration overhead crosses the control value. This means that even though the migration overhead of 5 seconds may seem small, it has a negative effect on the task execution time in this certain scenario. In many of those runs, the execution time becomes the same (or in some cases longer) as when having modules locally on a robot.

Even though Figure 5-2 shows that the task execution time can be higher or lower than the control value depending on the bandwidth ratio, Figure 5-3 shows that the total operation time, derived from the battery consumption and task time execution, does not change a lot depending on the bandwidth ratio. The data points are quite spread out, but a major part of the data points seem to indicate that it is possible to achieve a longer total operation if modules are offloaded to a cloud. Furthermore, the trend line indicates that, depending on the network condition, a robot can have up to an hour longer total operation time by offloading modules. Although that is to be expected, since lowering the load on the CPU should also lower the battery consumption. However, there is still a tradeoff where offloading modules to a cloud increases the required bandwidth, which means that the network card will see an increased load. In this experiment it seems that even though there is an increased amount of data sent over the network, the robot can still save battery time by offloading modules to a cloud.

Figure 5-4 shows how the task execution time changes depending solely on the RTT latency when the bandwidth ratio is over 10 units; which means that the robot is not even using one tenth of the available bandwidth. What can be seen in Figure 5-4 is that the data points are once again spread out. Even though the data is spread out, there is a notable linear trend that goes upwards; as the RTT latency increases, the task execution time increases. However, even when the RTT latency is over 160 milliseconds, the trend line still indicates that it is possible to have a task execution time that is lower than the control value. On the other side, one thing that is worrisome is that the simulated trend line for migration crosses the control line at 55 milliseconds RTT latency. This means that by using the proposed method of migrating modules, it would only be beneficial to place modules on a cloud if the system is run over a network that has a RTT latency lower than 55 milliseconds. The RTT latency when using a commercial cloud, e.g. Amazon Web Services, is usually over 100 milliseconds, which means that the only feasible solution would be to install a local cloud solution for offloading purposes.

Lastly, what can be gathered from Figure 5-5 and Figure 5-6, is that under stable network conditions there is clearly a lot to be gained from offloading modules to a cloud; 10% decrease in battery consumption and 2.4 seconds (2.8%) faster task completion. What is also interesting is that the results show that it is beneficial to change the local CPU frequency, depending on whether the modules are offloaded to a cloud or placed locally on the robot. When the modules are placed locally on the robot, it is better to set the CPU to a high frequency. Even though the CPU will consume a higher amount of battery per second, it will enable the robot to finish a task faster, which in turn lowers the energy consumption per task. When modules are offloaded to a cloud, it is more beneficial to lower the frequency of the CPU on the robot, in order to save battery consumption when possible. The only thing that a higher frequency on the local CPU seem to provide, when the modules are offloaded to a cloud, is the variance in measurements go down. The most probable reason for this is that a low CPU frequency means that the program will compete more with other programs running in the system. Sometimes there will be no backend tasks that will compete with the program, while sometimes there might be a lot of

competition for execution time in the CPU. When the CPU is configured to operate with a lower frequency it probably cannot handle this change. This means that you can have a more predictable system if you configure the CPU to run at a high frequency.

6 Conclusions and Future work

This chapter will conclude the thesis report with a conclusion of the project in Section 6.1. Section 6.2 discusses the limitations of the projects. This is followed by a description in Section 6.3 of the future work that remains. The chapter concludes in Section 6.4 with some reflections regarding the project.

6.1 Conclusions

This thesis project proposed a machine learning-based network aware framework solution to solve the problems with having cloud robots in unstable network environments. The goals of this thesis project were met since the proposed solution is able to save battery on the cloud robots as well as choose the most appropriate module placements depending on task, place, and network condition.

Looking at the results and taking into account the many hours of observing a cloud robot working in an unstable network environment, the area of CNR would ***not*** benefit from the proposed solution, since it has an inherent bottleneck in the training phase of the machine learning algorithm. Currently all cloud robots that wants to use the proposed solution would need to train for over 2 months to work efficiently. However, if connected and integrated with a shared-knowledge platform on a global level, the proposed solution could be very powerful, since all new robots would be able to skip the training phase by downloading the knowledge.

A recommendation for the ROS community is to develop better support for migrating modules within ROS. Even though this goes against the basic principles of ROS, where all modules should be initiated at the start of the system and remain static, module migration functionality would enable many powerful applications. Another recommendation is to rework the service implementation so that it can handle buffering of addresses (just as it can for topics), in order to reduce the time it takes to repeatedly make a service call when the network is subject to high latency. Furthermore, future applications should be developed with network awareness in mind, by decoupling functionality that *must* be placed locally on the robot from functionality that can make use of the power of the cloud.

I have learned a lot from this thesis project, since I had to learn and use a wide variety of programs as well as develop drivers, program functionality on different layers, and test the whole system myself. If I had to do it again I would limit the scope of the thesis project even further and put more time into defining the goals, the purpose, and the research methodology (specifically I would write chapters 1, 2, and 3 *before* starting to develop the solution). This would have helped tremendously during the early- and middle-stages of the project.

6.2 Limitations

The largest limitation of this thesis project was the limited amount of time to develop and test the proposed solution before the renovation of the laboratory facilities in the middle of August 2014. Because of this there were insufficient time to implement module migration functionality into the solution. Furthermore, the limited amount of time also meant that it was only possible to take 572 valid measurements during the experiment; this can be seen in the results and Figure 5-1. During the middle-stages of the project, a lot of time was dedicated to writing research papers that were published in various settings. Although the

time was well spent and I gained a lot of experience, this time could have been put into development and testing of the proposed solution.

Another large limitation was the fact that the TurtleBot navigation stack was a bit inconsistent and would at times get stuck, hence all the applications had to be restarted. This would happen randomly and the position of the modules did not matter. Although the consistency of the current navigation stack is much better than that of the first generation TurtleBot, there is still a need for further improvements.

The results were limited by the fact that the network profiling is done only once for every task. Since there is no continuous profiling of the network, the proposed solution cannot *adapt* to unpredictable network changes. This could be solved by implementing continuous network profiling, however this was out of the scope of this thesis project due to the limited amount of time. Furthermore, the experiment was only performed using a single type of cloud robot, hence it is impossible tell from these results if another cloud robot system would be able to improve the performance of a task and reduce its battery consumption, by implementing the proposed solution.

6.3 Future work

As this thesis project was done by a single person, there was not enough time to develop and implement a lot of the proposed functionality into a running solution; hence quite a lot has been left for future work. Some of this future work includes implementing the proxy-based module migration functionality (shown to the left in Figure 4-12) into the proposed solution. It would also be interesting to expand the single-robot system into a multi-robot system, where the knowledge gained from the machine learning-based network aware framework could be shared between robots. Another important future work is to implement continuous network profiling into the network aware framework, in order to enable the robot to adapt to unpredictable network changes. Furthermore, the proposed machine learning-based solution can currently only evaluate if an offloading migration decision was right or wrong. Future work will also need to address how to evaluate an onloading migration decision, in order to prevent the training set from becoming biased in the favor of onloading.

Other future work includes improving the temperature sensing module by applying Newton's law of cooling, in order to increase the precision of the temperature measurements compared to the "real" temperature at that location. It would also be interesting to explore the possibilities to connect the proposed solution to an external service, such as a Hadoop cluster where the data could be analyzed in order to find trends that are invisible to the human eye. In order to improve the performance of the proposed solution, the algorithms introduced in Section 4.1.2 and Section 4.1.4 could be improved as stated in those sections.

6.4 Reflections

As this technology develops further, robots will be able to autonomously move around in environments currently inhabited only by humans. This will surely be beneficial on many levels; however, this technology will surely encounter some ethical and security issues. These robots may go around and collect (on purpose or not) private information which can be shared via the cloud. This may lead to many serious problems concerning misuse or leakage of this information [12]. Furthermore, there will surely also be some ethical issues regarding where these cloud robots are allowed to move around. Legislators are recently

starting to acknowledge these issues around privacy and security [69], which is good since cloud networked robots are surely here to stay.

Bibliography

[1] M. Resnick, S. Ocko and S. Papert, "LEGO, Logo, and Design,," *Children's Environments Quarterly*, vol. 5, no. 4, pp. 14-18, 1988.

[2] R. A. Brooks and A. M. Flynn, "Fast, Cheap and Out of Control: A Robot Invasion of the Solar System," *Journal of The British Interplanetary Society*, vol. 42, pp. 478-485, 1989.

[3] M. Fujita and H. Kitano, "Development of an Autonomous Quadruped Robot for Robot Entertainment," *Autonomous Robots*, vol. 5, no. 1, pp. 7-18, 01 03 1998. DOI: 10.1023/A:1008856824126.

[4] Sony Corporation, "Sony Launches Four-Legged Entertainment Robot," Sony Corporation, 11 05 1999. [Online]. Available: http://www.sony.net/SonyInfo/News/Press_Archive/199905/99-046/index.html. [Accessed 27 12 2014].

[5] iRobot Corporation, "iRobot: Our History," iRobot, [Online]. Available: http://www.irobot.com/About-iRobot/Company-Information/History.aspx. [Accessed 27 12 2014].

[6] Clearpath Robotics Inc., "TurtleBot™ - Clearpath Robotics. TurtleBot is an open source ROS robot," Clearpath Robotics Inc., [Online]. Available: http://www.clearpathrobotics.com/turtlebot_2/. [Accessed 27 12 2014].

[7] iRobot Corporation, "iRobot-Create 2," iRobot Corporation, [Online]. Available: http://www.irobot.com/About-iRobot/STEM/Create-2.aspx. [Accessed 27 12 2014].

[8] EZ-Robot Inc., "Revolution JD," EZ-Robot Inc., [Online]. Available: http://www.ez-robot.com/Shop/AccessoriesDetails.aspx?productNumber=31. [Accessed 27 12 2014].

[9] N. Ando, T. Suehiro, K. Kitagaki, T. Kotoku and W.-K. Yoon, "RT-Component Object Model in RT-Middleware - Distributed Component Middleware for RT (Robot Technology)," in *2005 IEEE International Symposium on Computational Intelligence in Robotics and Automation, 2005. CIRA 2005. Proceedings*, Espoo, 2005. DOI: 10.1109/CIRA.2005.1554319.

[10] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler and A. Ng, "ROS: an open-source Robot Operating System," in *Proc. Open-Source Software workshop of the International Conference on Robotics and Automation (ICRA)*, Kobe, 2009.

[11] Open Source Robotics Foundation, "ROS.org | Powering the world's robots," Open Source Robotics Foundation, [Online]. Available: http://www.ros.org/. [Accessed 30 12 2014].

[12] K. Kamei, S. Nishio, N. Hagita and M. Sato, "Cloud Networked Robotics," *IEEE Network*, vol. 26, no. 3, pp.28-34, 17 05 2012. DOI:10.1109/MNET.2012.6201213.

[13] W. Voorsluys, J. Broberg and R. Buyya, "Introduction to Cloud Computing," in *Cloud Computing: Principles and Paradigms*, R. Buyya, J. Broberg and A. Goscinski, Eds., Hoboken, NJ: John Wiley & Sons, Inc., 2011, pp. 1-41. DOI: 10.1002/9780470940105.ch1.

[14] R. Buyya and S. Venugopal, "Market-Oriented Computing and Global Grids: An Introduction," in *Market-Oriented Grid and Utility Computing*, R. Buyya and K.

Bubendorfer , Eds., Hoboken, NJ: John Wiley & Sons, Inc., 2009, pp. 24-44. DOI: 10.1002/9780470455432.ch1.

[15] M. Hamdaqa and L. Tahvildari, "Cloud Computing Uncovered: A Research Landscape," in *Advances in Computers*, vol. 86, A. Hurson and A. Memon, Eds., Elsevier, 2012, pp. 41-85. DOI: 10.1016/B978-0-12-396535-6.00002-8.

[16] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," National Institute of Standards and Technology, Gaithersburg, 2011.

[17] I. Foster, Y. Zhao, I. Raicu and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," in *Grid Computing Environments Workshop, 2008. GCE '08*, Austin, TX, 2008. DOI: 10.1109/GCE.2008.4738445.

[18] G. B. Kandiraju, H. Franke, M. D. Williams, M. Steinder and S. M. Black, "Software defined infrastructures," *IBM Journal of Research and Development*, vol. 58, no. 2/3, pp. 2:1-2:13, 15 04 2014. DOI: 10.1147/JRD.2014.2298133.

[19] "AWS | Amazon Elastic Compute Cloud (EC2) - Scalable Cloud Hosting," Amazon Web Services, Inc., [Online]. Available: https://aws.amazon.com/ec2/. [Accessed 30 12 2014].

[20] "iCloud," Apple Inc., [Online]. Available: https://www.icloud.com/. [Accessed 30 12 2014].

[21] "Microsoft Cloud," Microsoft, [Online]. Available: http://www.microsoft.com/enterprise/microsoftcloud/default.aspx. [Accessed 30 12 2014].

[22] D. Lorencik and P. Sincak, "Cloud robotics: Current trends and possible use as a service," in *2013 IEEE 11th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, Herl'any, 2013. DOI: 10.1109/SAMI.2013.6480950.

[23] G. Hu, W. P. Tay and Y. Wen, "Cloud Robotics: Architecture, Challanges and Applications," *IEEE Network*, vol. 26, no. 3, pp. 21-28, 17 05 2012. DOI: 10.1109/MNET.2012.6201212.

[24] M. Roth, D. Vail and M. Veloso , "A Real-time World Model for Multi-Robot Teams with High-Latency Communication," in *Proceedings. 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003. (IROS 2003)*, Las Vegas, Nevada, 2003. DOI: 10.1109/IROS.2003.1249244.

[25] L. E. Parker, K. Fregene, Y. Guo and R. Madhavan, "Distributed Heterogeneous Sensing for Outdoor Multi-Robot Localization, Mapping, and Path Planning," in *2002 NRL Workshop Multi-Robot Systems: From Swarms to Intelligent Automata*, Washington, DC, 2002.

[26] M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Gálvez-López, K. Häussermann, R. Janssen, J. Montiel, A. Perzylo, B. Schießle, M. Tenorth, O. Zweigle and R. v. d. Molengraft, "RoboEarth: A World Wide Web for Robots," *IEEE Robotics & Automation Magazine*, vol. 18, no. 2, pp. 69-82, 16 06 2011. DOI: 10.1109/MRA.2011.941632.

[27] L. Turnbull and B. Samanta, "Cloud Robotics: Formation Control of a Multi Robot System Utilizing Cloud Infrastructure," in *2013 Proceedings of IEEE Southeastcon*, Jacksonville, FL, 2013. DOI: 10.1109/SECON.2013.6567422.

[28] R. Arumugam, V. R. Enti, L. Bingbing, W. Xiaojun, K. Baskaran, F. F. Kong, A. Kumar, K. D. Meng and G. W. Kit, "DAvinCi: A Cloud Computing Framework for Service Robots," in *2010 IEEE International Conference on Robotics and Automation (ICRA)*, Anchorage, AK, 2010. DOI: 10.1109/ROBOT.2010.5509469.

[29] Q. Lindsey, D. Mellinger and V. Kumar, "Construction with quadrotor teams,"

Autonomous Robots, vol. 33, no. 3, pp. 323-336, 01 10 2012. DOI: 10.1007/s10514-012-9305-0.

[30] Y.-Y. Chen, J.-F. Wang, P.-C. Lin, P.-Y. Shih, H.-C. Tsai and D.-Y. Kwan, "Human-Robot Interaction Based on Cloud Computing Infrastructure for Senior Companion," in *2011 IEEE Region 10 Conference TENCON 2011*, Bali, 2011. DOI: 10.1109/TENCON.2011.6129046.

[31] B. Kehoe, A. Matsukawa, S. Candido, J. Kuffner and K. Goldberg, "Cloud-Based Robot Grasping with the Google Object Recognition Engine," in *2013 IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, 2013. DOI: 10.1109/ICRA.2013.6631180.

[32] M. Tenorth, K. Kamei, S. Satake, T. Miyashita and N. Hagita, "Building Knowledge-enabled Cloud Robotics Applications using the Ubiquitous Network Robot Platform," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Tokyo, Japan, 2013. DOI: 10.1109/IROS.2013.6697184.

[33] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra and P. Bahl, "MAUI: Making Smartphones Last Longer with Code Offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services (MobiSys '10)*, New York, NY, USA, 2010. DOI: 10.1145/1814433.1814441.

[34] Y.-C. Wu, M.-C. Teng and Y.-J. Tsai, "Robot Docking Station for Automatic Battery Exchanging and Charging," in *Proceedings of the 2008 IEEE International Conference on Robotics and Biomimetics*, Bangkok, Thailand, 2009. DOI: 10.1109/ROBIO.2009.4913144.

[35] J. Zhang, G. Song, Y. Li, G. Qiao and Z. Li, "Battery Swapping and Wireless Charging for a Home Robot System with Remote Human Assistance," *IEEE Transactions on Consumer Electronics*, vol. 59, no. 4, pp. 747-755, 23 12 2013. DOI: 10.1109/TCE.2013.6689685.

[36] J. J. Leonard and H. F. Durrant-Whyte, "Simultaneous Map Building and Localization for an Autonomous Mobile Robot," in *IEEE/RSJ International Workshop on Intelligent Robots and Systems '91. 'Intelligence for Mechanical Systems, Proceedings IROS '91*, Osaka, 1991. DOI: 10.1109/IROS.1991.174711.

[37] A. Sanfeliu, N. Hagita and A. Saffiotti, "Network Robot Systems," *Robotics and Autonomous Systems*, vol. 56, no. 10, pp. 793-797, 2008. DOI: 10.1016/j.robot.2008.06.007.

[38] T. Akimoto and N. Hagita, "Introduction to a Network Robot System," in *International Symposium on Intelligent Signal Processing and Communications, 2006. ISPACS '06*, Yonago, 2006. DOI: 10.1109/ISPACS.2006.364842.

[39] James Kuffner (Google), "Cloud-Enabled Humanoid Robots," in *Presentation at 2010 10th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2010 Workshop)*, Nashville, 2010.

[40] G. Mohanarajah, D. Hunziker, R. D'Andrea and M. Waibel, "Rapyuta: A Cloud Robotics Platform," *IEEE Transactions on Automation Science and Engineering*, vol. PP, no. 99, pp. 1-13, 11 07 2014. DOI: 10.1109/TASE.2014.2329556.

[41] E. Guizzo , "How Google's Self-Driving Car Works," IEEE Spectrum, 18 10 2011. [Online]. Available: http://spectrum.ieee.org/automaton/robotics/artificial-intelligence/how-google-self-driving-car-works. [Accessed 15 01 2015].

[42] Kiwa Systems, "Kiva's warehouse automation system is the most powerful and flexible Automated Storage and Retrieval System (AS/RS) available for

automating DCs," Kiwa Systems, [Online]. Available: http://www.kivasystems.com/solutions/. [Accessed 15 01 2015].

[43] J. Markoff, "New Research Center Aims to Develop Second Generation of Surgical Robots," New York Times, 23 10 2014. [Online]. Available: http://www.nytimes.com/2014/10/23/science/new-research-center-aims-to-develop-second-generation-of-surgical-robots.html. [Accessed 15 01 2015].

[44] "VINBOT | VINBOT Official Website," VINBOT, [Online]. Available: http://vinbot.eu/. [Accessed 15 01 2015].

[45] T. Foote, "Distributions - ROS Wiki," Open Source Robotics Foundation, 22 07 2014. [Online]. Available: http://wiki.ros.org/Distributions. [Accessed 15 01 2015].

[46] T. Foote and D. Thomas, "Naming Conventions for Catkin Based Workspaces," 03 07 2014. [Online]. Available: https://github.com/ros-infrastructure/rep/blob/master/rep-0128.rst. [Accessed 15 01 2015].

[47] R. Baldoni, M. Contenti and A. Virgillito, "The Evolution of Publish/Subscribe Communication Systems," in *Future directions in distributed computing*, Heidelberg, Springer-Verlag, 2003, pp. 137-141.

[48] E. Ackerman, "Interview: TurtleBot Inventors Tell Us Everything About the Robot," 26 03 2013. [Online]. Available: http://spectrum.ieee.org/automaton/robotics/diy/interview-turtlebot-inventors-tell-us-everything-about-the-robot. [Accessed 17 01 2015].

[49] F. Dellaert, D. Fox, W. Burgard and S. Thrun, "Monte Carlo Localization for Mobile Robots," in *Proceedings. 1999 IEEE International Conference on Robotics and Automation, 1999*, Detroit, MI, 1999. DOI: 10.1109/ROBOT.1999.772544.

[50] D. Fox, "Adapting the Sample Size in Particle Filters Through KLD-Sampling," *International Journal on Robotics Research (IJRR)*, vol. 22, no. 12, pp. 985-1003, 2003.

[51] D. Fox, "KLD-Sampling: Adaptive Particle Filters," in *Advances in Neural Information Processing Systems 14 (NIPS-01)*, MIT Press, 2001.

[52] "OpenSLAM.org," [Online]. Available: http://openslam.org/gmapping.html. [Accessed 19 01 2015].

[53] G. Grisetti, C. Stachniss and W. Burgard, "Improving Grid-based SLAM with Rao-Blackwellized Particle Filter by Adaptive Proposasls and Selective Resampling," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation, 2005. ICRA 2005*, 2005. DOI: 10.1109/ROBOT.2005.1570477.

[54] G. Grisetti, C. Stachniss and W. Burgard, "Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34-46, 05 02 2007. DOI: 10.1109/TRO.2006.889486.

[55] H. P. Moravec, "Sensor Fusion in Certainty Grids for Mobile Robots," *AI Magazine*, vol. 9, no. 2, pp. 61-74, 07/08 1988. ISSN: 0738-4602.

[56] H. W. Keat and L. S. Min, "An Investigation of the Use of Kinect Sensor for Indoor Navigation," in *TENCON 2012 - 2012 IEEE Region 10 Conference*, Cebu, 2012. DOI: 10.1109/TENCON.2012.6412246.

[57] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik and A. Patti, "CloneCloud: Elastic Execution between Mobile Device and Cloud," in *Proceedings of the Sixth Conference on Computer Systems (EuroSys '11)*, Salzburg, Austria, 2011. DOI: 10.1145/1966445.1966473.

[58] S. Kosta, A. Aucinas, P. Hui, R. Mortier and X. Zhang, "ThinkAir: Dynamic

resource allocation and parallel execution in the cloud for mobile code offloading," in *2012 Proceedings IEEE INFOCOM*, Orlando, FL, 2012. DOI: 10.1109/INFCOM.2012.6195845.

[59] H. Eom, P. St Juste, R. Figueiredo, O. Tickoo, R. Illikkal and R. Iyer, "Machine Learning-Based Runtime Scheduler for Mobile Offloading Framework," in *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing (UCC)*, Dresden, 2013. DOI: 10.1109/UCC.2013.21.

[60] M. Fallon, S. Kuindersma, S. Karumanchi, M. Antone, T. Schneider, H. Dai, C. P. D'Arpino, R. Deits, M. DiCicco, D. Fourie, T. Koolen, P. Marion, M. Posa, A. Valenzuela, K.-T. Yu, J. Shah, K. Iagnemma, R. Tedrake and S. Teller, "An Architecture for Online Affordance-based Perception and Whole-body Planning," Cambridge, 2014.

[61] DARPA, "Upgraded Atlas Robot to Go Wireless as the Stakes Are Raised for the DARPA Robotics Challenge Finals," 20 01 2015. [Online]. Available: http://www.theroboticschallenge.org/upgraded-atlas-press-release. [Accessed 21 01 2015].

[62] W. J. Beksi and N. Papanikolopoulos, "Point Cloud Culling for Robot Vision Tasks Under Communication Constraints," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014)*, Chicago, IL, USA, 2014. DOI: 10.1109/IROS.2014.6943088.

[63] F. Nordlund, M. Higashida, Y. Teranishi, S. Shimojo, M. Yokoyama and M. Shimomura, "Designing of a Network-Aware Cloud Robotic Sensor Observation Framework," in *2014 IEEE 38th International Computer Software and Applications Conference Workshops (COMPSACW)*, Västerås, 2014. DOI: 10.1109/COMPSACW.2014.51.

[64] Jones, Rick;, "The Netperf Homepage," Hewlett-Packard Company, [Online]. Available: http://www.netperf.org/netperf/. [Accessed 06 01 2015].

[65] G. Roualland, "Ifstat," 01 02 2004. [Online]. Available: http://gael.roualland.free.fr/ifstat/. [Accessed 06 01 2015].

[66] M. Higashida, "Autonomic Datacenter Operation under the concept of Cloud Network Robotics," in *IEICE Technical Report*, Osaka, 2013.

[67] M. A. Batalin, "Cooperative Algorithms for Mobile Robots and a Sensor Network," 2004.

[68] M. Kerrisk, "proc(5) - Linux manual page," 31 12 2014. [Online]. Available: http://man7.org/linux/man-pages/man5/proc.5.html. [Accessed 06 01 2015].

[69] A. A. Proia, D. Simshaw and K. Hauser, "Consumer Cloud Robotics and the Fair Information Practice Principles: Recognizing the Challenges and Opportunities Ahead," *Minnesota Journal of Law, Science & Technology, Forthcoming,* 2014. DOI: 10.2139/ssrn.2466723.

Appendix I: TC network controlling example

This appendix will describe an actual example of the settings used in *tc* to simulate various network conditions. The example will make use of queuing disciplines (qdisc) and hierarchical token buckets (htb) in order to induce the network with latency and bandwidth. The example will set the bandwidth of the wireless network link to 1Mbit/s and 135ms latency. The settings can be seen in the table below.

```
# tc qdisc add dev wlan0 root handle 1: htb default 10
# tc class add dev wlan0 parent 1: classid 1:1 htb rate 1000kbit burst 500k
# tc qdisc add wlan0 parent 1:1 handle 10: netem delay 135ms
# tc filter add dev eth0 parrent 1: protocol ip u32 match ip dst 10.0.0.2 flowid 1:1
```

Appendix II: Detailed results of RTT experiment

The RTT when using ROS services under various latency and module placements. See Section 4.2.2 for further details about the experiment.

ms (Master-Monitor)	0	20	50	100
1	19	126	278	539
2	24	136	282	532
3	19	126	283	526
4	22	134	284	528
5	35	131	278	556
6	31	139	277	541
7	19	142	280	532
8	23	158	283	535
9	27	131	281	529
10	20	171	280	530
11	30	128	276	526
12	22	132	285	532
13	19	143	288	535
14	18	124	279	551
15	21	128	385	531
16	20	124	281	536
17	22	131	301	528
18	20	138	361	535
19	18	128	287	532
20	20	131	285	534
Average	22.45	135.05	291.70	534.40
Standard Deviation	4.73	11.67	28.59	7.67
Sample Size	20.00	20.00	20.00	20.00
Confidence Coff.	1.96	1.96	1.96	1.96
Margin of Error	2.07	5.12	12.53	3.36
Upper Bound	24.52	140.17	304.23	537.76
Lower Bound	20.38	129.93	279.17	531.04
Max	35.00	171.00	385.00	556.00
Min	18.00	124.00	276.00	526.00
Range	17.00	47.00	109.00	30.00
Variance	22.37	136.26	817.17	58.78
Standard Error	1.06	2.61	6.39	1.71
Median	20.50	131.00	282.50	532.00

ms (Client-Master)	0	20	50	100
1	14	75	168	335
2	13	79	176	319
3	14	76	181	321
4	20	78	170	316
5	13	78	168	323
6	13	88	172	323
7	15	83	170	323
8	14	79	173	321
9	14	83	172	322
10	14	74	275	337
11	17	77	171	322
12	15	77	170	396
13	15	88	169	319
14	14	78	170	324
15	14	78	169	319
16	14	105	170	316
17	15	83	176	322
18	13	82	177	326
19	14	82	296	322
20	13	99	177	322
Average	14.40	82.10	183.50	326.40
Standard Deviation	1.64	7.86	35.22	17.18
Sample Size	20.00	20.00	20.00	20.00
Confidence Coff.	1.96	1.96	1.96	1.96
Margin of Error	0.72	3.44	15.44	7.53
Upper Bound	15.12	85.54	198.94	333.93
Lower Bound	13.68	78.66	168.06	318.87
Max	20.00	105.00	296.00	396.00
Min	13.00	74.00	168.00	316.00
Range	7.00	31.00	128.00	80.00
Variance	2.67	61.78	1240.79	295.09
Standard Error	0.37	1.76	7.88	3.84
Median	14.00	79.00	171.50	322.00

ms (Client-Monitor)	0	20	50	100
1	9	58	115	221
2	10	87	115	220
3	14	54	115	219
4	13	58	151	214
5	14	55	119	219
6	12	51	116	217
7	12	57	118	232
8	14	58	123	217
9	12	53	115	214
10	11	99	113	223
11	10	56	114	216
12	12	53	120	220
13	8	62	123	212
14	10	56	112	219
15	10	57	117	218
16	13	52	117	216
17	21	55	117	216
18	21	62	116	295
19	11	54	113	214
20	12	57	127	218
Average	12.45	59.70	118.80	222.00
Standard Deviation	3.36	11.90	8.46	17.69
Sample Size	20.00	20.00	20.00	20.00
Confidence Coff.	1.96	1.96	1.96	1.96
Margin of Error	1.47	5.22	3.71	7.75
Upper Bound	13.92	64.92	122.51	229.75
Lower Bound	10.98	54.48	115.09	214.25
Max	21.00	99.00	151.00	295.00
Min	8.00	51.00	112.00	212.00
Range	13.00	48.00	39.00	83.00
Variance	11.31	141.69	71.64	313.05
Standard Error	0.75	2.66	1.89	3.96
Median	12.00	56.50	116.50	218.00

ms (Multimaster)	0	20	50	100
1	14	77	162	312
2	13	73	162	312
3	19	77	162	311
4	10	76	163	314
5	20	78	165	319
6	14	71	163	313
7	12	71	165	315
8	15	73	166	312
9	12	74	163	310
10	13	68	160	311
11	11	78	161	313
12	23	88	164	312
13	17	72	165	312
14	12	91	171	311
15	14	75	163	318
16	14	77	167	313
17	11	72	170	312
18	12	70	162	311
19	13	77	162	322
20	12	111	162	313
Average	14.05	77.45	163.90	313.30
Standard Deviation	3.32	9.65	2.85	3.05
Sample Size	20.00	20.00	20.00	20.00
Confidence Coff.	1.96	1.96	1.96	1.96
Margin of Error	1.45	4.23	1.25	1.33
Upper Bound	15.50	81.68	165.15	314.63
Lower Bound	12.60	73.22	162.65	311.97
Max	23.00	111.00	171.00	322.00
Min	10.00	68.00	160.00	310.00
Range	13.00	43.00	11.00	12.00
Variance	11.00	93.10	8.09	9.27
Standard Error	0.74	2.16	0.64	0.68
Median	13.00	75.50	163.00	312.00

The RTT when using ROS topics under various latency and module placements. See Section 4.2.2 for further details about the experiment.

ms (Master-Monitor)	0	20	50	100
1	2	22	52	108
2	2	22	52	102
3	5	22	52	102
4	2	22	52	102
5	2	22	52	102
6	2	22	59	102
7	2	22	52	103
8	3	22	143	104
9	2	22	52	102
10	2	22	58	102
11	2	22	52	103
12	2	22	52	102
13	2	29	53	229
14	2	22	56	102
15	8	22	62	102
16	2	22	52	104
17	2	24	52	103
18	3	22	52	102
19	2	22	52	107
20	5	25	52	228
Average	2.70	22.60	57.95	115.55
Standard Deviation	1.56	1.70	20.23	38.67
Sample Size	20.00	20.00	20.00	20.00
Confidence Coff.	1.96	1.96	1.96	1.96
Margin of Error	0.68	0.74	8.87	16.95
Upper Bound	3.38	23.34	66.82	132.50
Lower Bound	2.02	21.86	49.08	98.60
Max	8.00	29.00	143.00	229.00
Min	2.00	22.00	52.00	102.00
Range	6.00	7.00	91.00	127.00
Variance	2.43	2.88	409.21	1495.00
Standard Error	0.35	0.38	4.52	8.65
Median	2.00	22.00	52.00	102.00

ms (Client-Master)	0	20	50	100
1	2	22	52	102
2	5	25	52	107
3	5	22	52	102
4	10	23	52	102
5	2	25	52	106
6	2	23	52	103
7	3	22	52	103
8	2	23	52	106
9	2	25	52	103
10	2	22	52	102
11	3	22	52	102
12	2	25	52	102
13	5	23	53	102
14	2	24	55	102
15	6	22	53	102
16	2	27	52	102
17	2	24	52	102
18	2	22	53	102
19	2	22	53	102
20	2	22	52	102
Average	3.15	23.25	52.35	102.80
Standard Deviation	2.08	1.48	0.75	1.58
Sample Size	20.00	20.00	20.00	20.00
Confidence Coff.	1.96	1.96	1.96	1.96
Margin of Error	0.91	0.65	0.33	0.69
Upper Bound	4.06	23.90	52.68	103.49
Lower Bound	2.24	22.60	52.02	102.11
Max	10.00	27.00	55.00	107.00
Min	2.00	22.00	52.00	102.00
Range	8.00	5.00	3.00	5.00
Variance	4.34	2.20	0.56	2.48
Standard Error	0.47	0.33	0.17	0.35
Median	2.00	23.00	52.00	102.00

ms (Master-Monitor)	0	20	50	100
1	2	22	52	108
2	2	22	52	102
3	5	22	52	102
4	2	22	52	102
5	2	22	52	102
6	2	22	59	102
7	2	22	52	103
8	3	22	143	104
9	2	22	52	102
10	2	22	58	102
11	2	22	52	103
12	2	22	52	102
13	2	29	53	229
14	2	22	56	102
15	8	22	62	102
16	2	22	52	104
17	2	24	52	103
18	3	22	52	102
19	2	22	52	107
20	5	25	52	228
Average	2.70	22.60	57.95	115.55
Standard Deviation	1.56	1.70	20.23	38.67
Sample Size	20.00	20.00	20.00	20.00
Confidence Coff.	1.96	1.96	1.96	1.96
Margin of Error	0.68	0.74	8.87	16.95
Upper Bound	3.38	23.34	66.82	132.50
Lower Bound	2.02	21.86	49.08	98.60
Max	8.00	29.00	143.00	229.00
Min	2.00	22.00	52.00	102.00
Range	6.00	7.00	91.00	127.00
Variance	2.43	2.88	409.21	1495.00
Standard Error	0.35	0.38	4.52	8.65
Median	2.00	22.00	52.00	102.00

ms (Client-Master)	0	20	50	100
1	2	22	52	102
2	5	25	52	107
3	5	22	52	102
4	10	23	52	102
5	2	25	52	106
6	2	23	52	103
7	3	22	52	103
8	2	23	52	106
9	2	25	52	103
10	2	22	52	102
11	3	22	52	102
12	2	25	52	102
13	5	23	53	102
14	2	24	55	102
15	6	22	53	102
16	2	27	52	102
17	2	24	52	102
18	2	22	53	102
19	2	22	53	102
20	2	22	52	102
Average	3.15	23.25	52.35	102.80
Standard Deviation	2.08	1.48	0.75	1.58
Sample Size	20.00	20.00	20.00	20.00
Confidence Coff.	1.96	1.96	1.96	1.96
Margin of Error	0.91	0.65	0.33	0.69
Upper Bound	4.06	23.90	52.68	103.49
Lower Bound	2.24	22.60	52.02	102.11
Max	10.00	27.00	55.00	107.00
Min	2.00	22.00	52.00	102.00
Range	8.00	5.00	3.00	5.00
Variance	4.34	2.20	0.56	2.48
Standard Error	0.47	0.33	0.17	0.35
Median	2.00	23.00	52.00	102.00

Appendix III: Measuring startup time of move_base

This appendix presents the full table of results from the experiment where the startup time of the move_base module was measured, when using single-master and multimaster.

Using single-master:

move_base startup time under delay (RTT ms)	0ms	20ms	50ms	100ms	120ms	140ms
1	14520	44380	93320	180020	200900	229240
2	12150	46040	95730	173150	199720	229760
3	13230	46280	97320	175460	200580	231140
4	13630	44670	96800	171630	201740	228500
5	13540	45290	95270	172000	199840	232390
Average	13414.00	45332.00	95688.00	174452.00	200556.00	230206.00
Standard Deviation	854.01	828.60	1555.95	3452.89	826.49	1556.01
Sample Size	5.00	5.00	5.00	5.00	5.00	5.00
Confidence Coff.	1.96	1.96	1.96	1.96	1.96	1.96
Margin of Error	748.57	726.30	1363.85	3026.59	724.45	1363.91
Upper Bound	14162.57	46058.30	97051.85	177478.59	201280.45	231569.91
Lower Bound	12665.43	44605.70	94324.15	171425.41	199831.55	228842.09
Max	14520.00	46280.00	97320.00	180020.00	201740.00	232390.00
Min	12150.00	44380.00	93320.00	171630.00	199720.00	228500.00
Range	2370.00	1900.00	4000.00	8390.00	2020.00	3890.00
Variance	729330.00	686570.00	2420970.00	11922470.00	683080.00	2421180.00
Standard Error	381.92	370.56	695.84	1544.18	369.62	695.87
Median	13540.00	45290.00	95730.00	173150.00	200580.00	229760.00

Using multimaster:

move_base startup time under delay (RTT ms)	0ms	20ms	50ms	100ms	120ms	140ms
1	5530	5280	5450	5450	6020	5900
2	5330	5160	5450	5500	6060	5980
3	5330	5490	5030	5760	5680	6360
4	5580	5720	5370	5410	6150	6400
5	4980	5460	5450	6050	6020	5920
Average	5350.00	5422.00	5350.00	5634.00	5986.00	6112.00
Standard Deviation	236.11	214.29	182.21	269.69	179.11	246.82
Sample Size	5.00	5.00	5.00	5.00	5.00	5.00
Confidence Coff.	1.96	1.96	1.96	1.96	1.96	1.96
Margin of Error	206.96	187.83	159.71	236.39	157.00	216.35
Upper Bound	5556.96	5609.83	5509.71	5870.39	6143.00	6328.35
Lower Bound	5143.04	5234.17	5190.29	5397.61	5829.00	5895.65
Max	5580.00	5720.00	5450.00	6050.00	6150.00	6400.00
Min	4980.00	5160.00	5030.00	5410.00	5680.00	5900.00
Range	600.00	560.00	420.00	640.00	470.00	500.00
Variance	55750.00	45920.00	33200.00	72730.00	32080.00	60920.00
Standard Error	105.59	95.83	81.49	120.61	80.10	110.38
Median	5330.00	5460.00	5450.00	5500.00	6020.00	5980.00

Appendix IV: Finding a resource heavy module

This appendix presents the detailed results from the experiment to determine which modules that are resource heavy in the TurtleBot. The experiment were divided into two settings: local modules with power management disabled and CPU frequency set to user space mode 1.6GHz, and remotely placed modules with power management disabled and local CPU frequency set to user space mode 1.6GHz.

Local modules with power management disabled and CPU frequency set to user space mode 1.6GHz

	battery percentage change	CPU working time (jiffies)	CPU working time (s)	CPU UTIME	CPU NICE	CPU STIME	CPU IDLE
1	0.00507617	40486	101.215	26668	0	5950	7566
2	0.00429523	32612	81.53	21098	0	4712	6550
3	0.00644279	49765	124.412	31433	0	7339	10621
4	0.00429517	33767	84.4175	21914	0	5005	6586
5	0.00429517	33399	83.4975	21768	0	4801	6552
6	0.00546658	42164	105.41	26898	0	6146	8787
7	0.00429523	32769	81.9225	21173	0	4765	6570
8	0.00449044	33114	82.785	21363	0	4806	6693
9	0.00468564	34128	85.32	22052	0	4905	6915
10	0.00449044	33227	83.0675	21677	0	4941	6367
11	0.00448608	35101	87.7525	22441	1	5432	6926
12	0.004291	33130	82.825	21146	0	5008	6728
13	0.00448608	33321	83.3025	21263	0	5035	6762
14	0.00468111	37488	93.72	23856	0	5644	7679
15	0.004291	32957	82.3925	21026	0	5075	6609
16	0.004291	33199	82.9975	21224	0	4930	6770
17	0.00429106	33062	82.655	21181	0	4801	6846
18	0.004291	33284	83.21	21199	1	4908	6927
19	0.00448602	33446	83.615	21638	0	4790	6762
20	0.004291	33010	82.525	20991	0	4814	6951
Average	0.00458591	35171.45000000	87.92860000	22600.45000000	0.10000000	5190.35000000	7108.35000000
Standard Deviation	0.00070242	5752.65199616	14.38150230	3513.35038579	0.00000000	867.88273657	1368.63858146
Sample Size	10	10	10	10	10	10	10
Confidence Coff.	1.96	1.96	1.96	1.96	1.96	1.96	1.96
Margin of Error	0.00043537	3565.53064726	8.91374748	2177.59712972	0.00000000	537.91929315	848.29098136
Upper Bound	0.00502128	38736.98064726	96.84234748	24778.04712972	0.10000000	5728.26929315	7956.64098136
Lower Bound	0.00415054	31605.91935274	79.01485252	20422.85287028	0.10000000	4652.43070685	6260.05901864
Max	0.00644279	49765.00000000	124.41200000	31433.00000000	0.00000000	7339.00000000	10621.00000000
Min	0.00429517	32612.00000000	81.53000000	21098.00000000	0.00000000	4712.00000000	6367.00000000
Range	0.00214762	17153.00000000	42.88200000	10335.00000000	0.00000000	2627.00000000	4254.00000000
Variance	0.00000049	33093004.98888880	206.82760846	12343630.93333330	0.00000000	753220.44444444	1873171.56666667
Standard Error	0.00022213	1819.14828942	4.54783034	1111.01894373	0.00000000	274.44861895	432.80152110
Median	0.00449044	33583.00000000	83.95750000	21841.00000000	0.00000000	4923.00000000	6639.50000000

CPU IO WAIT	Between first and last packet (s)	Number of sent packets	Avg. packets/sec	Avg. packet size (bytes)	Avg. bytes/sec	RVIZ (utime) (%)	RVIZ (stime) (%)
35	95.757	16	0.167	348.75	58.273	31.4899	1.11644
20	7.6	8	1.053	348.75	367.106	30.9671	1.10389
24	58.886	16	0.272	348.75	94.759	30.7827	1.03888
14	34.718	30	0.864	370.333	320.003	31.4508	1.11647
37	86.716	12	0.138	348.75	48.261	31.9531	1.21261
22	87.596	12	0.137	348.75	47.776	31.4581	1.10758
17	48.926	12	0.245	348.75	85.537	31.9113	1.10165
22	7.494	8	1.067	348.75	273.277	31.6452	1.10225
10	71.813	12	0.167	348.75	58.277	31.669	1.02555
24	7.827	8	1.022	348.75	356.465	31.9018	1.1918
54	7.55	8	1.06	348.75	369.543	30.2214	1.07689
23	61.389	30	0.489	370.333	180.978	30.2415	0.971929
25	82.743	12	0.145	348.75	50.578	30.5693	1.04739
31	77.344	18	0.233	325.222	75.688	30.6685	1.00299
24	7.816	8	1.024	348.75	356.956	30.8645	1.01344
30	7.79	8	1.027	348.75	358.167	31.1425	0.960872
17	82.78	13	0.157	350.846	55.098	31.1264	1.0314
20	33.549	30	0.894	370.333	331.153	30.5192	1.04555
21	73.561	16	0.218	348.75	75.855	31.37	1.0943
19	7.745	8	1.033	348.75	360.243	30.3272	0.0969403
24.45000000	47.48000000	14.25000000	0.57060000	350.91585000	196.19965000	31.11397500	1.02294107
8.40965054	34.99586096	6.53537383	0.42585021	6.82514387	139.16288743	0.39201622	0.05738458
10	10	10	10	10	10	10	10
1.96	1.96	1.96	1.96	1.96	1.96	1.96	1.96
5.21235541	21.69065935	4.05066667	0.26394469	4.23026800	86.25405125	0.24297417	0.03556733
29.66235541	69.17065935	18.30066667	0.83454469	355.14611800	282.45370125	31.35694917	1.05850839
19.23764459	25.78934065	10.19933333	0.30665531	346.68558200	109.94559875	30.87100083	0.98737374
37.00000000	95.75700000	30.00000000	1.06700000	370.33300000	367.10600000	31.95310000	1.21261000
10.00000000	7.49400000	8.00000000	0.13700000	348.75000000	47.77600000	30.78270000	1.02555000
27.00000000	88.26300000	22.00000000	0.93000000	21.58300000	319.33000000	1.17040000	0.18706000
70.72222222	1224.71028468	42.71111111	0.18134840	46.58258890	19366.30923649	0.15367672	0.00329299
2.65936500	11.06666293	2.06666667	0.13466566	2.15830000	44.00716900	0.12396641	0.01814660
22.00000000	53.90600000	12.00000000	0.25850000	348.75000000	90.14800000	31.56755000	1.10573500

camera/rectify_ir (utime) (%)	camera/rectify_ir (stime) (%)	camera/register_depth_rgb (utime) (%)	camera/register_depth_rgb (stime) (%)	camera/points_xyzrgb_sw_registered (utime) (%)
0.0246999	0.0296399	0.0222299	0.0296399	0.0271699
0.0306636	0.0245308	0.0245308	0.0306636	0.0275972
0.0200944	0.0321511	0.0221039	0.0301417	0.0221039
0.0148074	0.0384991	0.0236918	0.0296147	0.0266532
0.0329351	0.0239528	0.0209587	0.0329351	0.029941
0.0237169	0.030832	0.0260886	0.0260886	0.0237169
0.027465	0.027465	0.0305166	0.0244133	0.027465
0.0271788	0.024159	0.0332186	0.0181192	0.0181192
0.0263713	0.0293015	0.0263713	0.0234412	0.0175809
0.0240768	0.030096	0.0210672	0.0331056	0.0210672
0.0199425	0.0313381	0.0227914	0.0313381	0.0227914
0.0211289	0.0332025	0.0211289	0.0301841	0.0211289
0.02701	0.0240089	0.0180067	0.0330122	0.02701
0.0293427	0.0266752	0.0186726	0.0346778	0.0266752
0.0212398	0.0333768	0.0212398	0.0303426	0.0212398
0.0240971	0.0271093	0.021085	0.0331335	0.021085
0.024197	0.024197	0.0181477	0.0362954	0.0302462
0.0300445	0.0240356	0.0180267	0.0360534	0.02704
0.0209293	0.0328888	0.0239192	0.026909	0.0298989
0.0242351	0.0302939	0.0181763	0.0333232	0.0242351
0.02470881	0.02888767	0.02259859	0.03017161	0.02463825
0.00514117	0.00442744	0.00409301	0.00473748	0.00428427
10	10	10	10	10
1.96	1.96	1.96	1.96	1.96
0.00318653	0.00274416	0.00253688	0.00293632	0.00265542
0.02789534	0.03163182	0.02513546	0.03310793	0.02729366
0.02152227	0.02614351	0.02006171	0.02723529	0.02198283
0.03293510	0.03849910	0.03321860	0.03310560	0.02994100
0.01480740	0.02395280	0.02095870	0.01811920	0.01758090
0.01812770	0.01454630	0.01225990	0.01498640	0.01236010
0.00002643	0.00001960	0.00001675	0.00002244	0.00001835
0.00162578	0.00140008	0.00129432	0.00149812	0.00135480
0.02553560	0.02947070	0.02411130	0.02962730	0.02518505

camera/points_xyzrgb_sw_registered (stime) (%)	camera/depth_registered_rectify_depth (utime) (%)	camera/depth_registered_rectify_depth (stime) (%)	camera/points_xyzrgb_hw_registered (utime) (%)
0.0271699	0.0197599	0.0321099	0.0271699
0.0245308	0.0275972	0.0245308	0.0153318
0.0321511	0.0221039	0.0301417	0.0221039
0.0296147	0.0236918	0.0266532	0.0266532
0.0269469	0.029941	0.0239528	0.0179646
0.0332037	0.0237169	0.030832	0.0213452
0.027465	0.0152583	0.03662	0.0305166
0.0362385	0.0271788	0.0271788	0.0301987
0.0322316	0.0293015	0.0263713	0.0234412
0.030096	0.0240768	0.030096	0.0240768
0.0284892	0.0227914	0.0284892	0.0199425
0.0332025	0.0271657	0.0271657	0.0211289
0.02701	0.0180067	0.0360133	0.02701
0.0266752	0.0186726	0.0320102	0.0186726
0.0303426	0.0212398	0.0333768	0.0212398
0.0301214	0.0240971	0.0271093	0.0301214
0.0211723	0.024197	0.0302462	0.024197
0.02704	0.0210311	0.0330489	0.0240356
0.0239192	0.0179394	0.0358787	0.0209293
0.0302939	0.0272645	0.0242351	0.0212057
0.02889573	0.02325157	0.03009604	0.02336424
0.00353717	0.00452067	0.00386652	0.00494817
10	10	10	10
1.96	1.96	1.96	1.96
0.00219236	0.00280194	0.00239650	0.00306691
0.03108809	0.02605351	0.03249254	0.02643114
0.02670336	0.02044963	0.02769955	0.02029733
0.03623850	0.02994100	0.03662000	0.03051660
0.02453080	0.01525830	0.02395280	0.01533180
0.01170770	0.01468270	0.01266720	0.01518480
0.00001251	0.00002044	0.00001495	0.00002448
0.00111855	0.00142956	0.00122270	0.00156475
0.02985535	0.02389685	0.02863740	0.02375900

camera/points_xyzrgb_hw_registered (stime) (%)	camera/disparity_depth (utime) (%)	camera/disparity_depth (stime) (%)	camera/disparity_registered_sw (utime) (%)	camera/disparity_registered_sw (stime) (%)
0.0271699	0.0271699	0.0246999	0.0222299	0.0321099
0.0367963	0.0345308	0.0275972	0.0245308	0.0275972
0.0321511	0.0261228	0.0261228	0.0241133	0.0301417
0.0266532	0.0207303	0.0355377	0.0207303	0.0355377
0.0359292	0.0239528	0.0329351	0.0239528	0.029941
0.0332037	0.0260886	0.0284603	0.0237169	0.030832
0.0244133	0.027465	0.027465	0.0213617	0.0335683
0.0211391	0.0120795	0.0422782	0.0271788	0.0271788
0.0293015	0.0146507	0.041022	0.0175809	0.0322316
0.030096	0.030096	0.0270864	0.0180576	0.0331056
0.0431871	0.0227914	0.0313381	0.0199425	0.0313381
0.0301841	0.0211289	0.0301841	0.0181105	0.0362209
0.0300111	0.0240089	0.0300111	0.0210078	0.0300111
0.0346778	0.0266752	0.0240077	0.0266752	0.0266752
0.0333768	0.0151713	0.03644111	0.0242741	0.0333768
0.0240971	0.0271093	0.0271093	0.0180728	0.0331335
0.0332708	0.0211723	0.0332708	0.0302462	0.024197
0.0300445	0.0210311	0.0360534	0.02704	0.02704
0.0298989	0.026909	0.0239192	0.026909	0.0298989
0.0302939	0.0272645	0.0242351	0.0272645	0.0242351
0.03079477	0.02380742	0.03048873	0.02314978	0.03041852
0.00499645	0.00681787	0.00632690	0.00298893	0.00262324
10	10	10	10	10
1.96	1.96	1.96	1.96	1.96
0.00309683	0.00422576	0.00392145	0.00185256	0.00162590
0.03389160	0.02803317	0.03441018	0.02500234	0.03204442
0.02769794	0.01958166	0.02656727	0.02129722	0.02879262
0.03679630	0.03453080	0.04227820	0.02717880	0.03553770
0.02113910	0.01207950	0.02469990	0.01758090	0.02717880
0.01565720	0.02245130	0.01757830	0.00959790	0.00835890
0.00002496	0.00004648	0.00004003	0.00000893	0.00000688
0.00158002	0.00215600	0.00200074	0.00094518	0.00082954
0.02969875	0.02610570	0.02802875	0.02297340	0.03147095

camera/disparity_registered_hw (utime) (%)	camera/disparity_registered_hw (stime) (%)	map_server (utime) (%)	map_server (stime) (%)	amcl (utime) (%)	amcl (stime) (%)	move_base (utime) (%)	move_base (stime) (%)
0.0222299	0.0321099	0.0296399	0.0296399	2.44035	0.941066	17.8062	2.29215
0.0214645	0.0337299	0.0337299	0.0306636	2.25684	0.972035	17.6407	2.23537
0.0261228	0.0301417	0.0321511	0.0321511	2.61831	1.00472	16.5699	2.28474
0.0296147	0.0236918	0.0296147	0.0296147	2.30402	0.998016	17.5941	2.38695
0.029941	0.0239528	0.0269469	0.0359292	2.21863	1.05392	17.4436	2.25755
0.0213452	0.0332037	0.0284603	0.0332037	2.60412	0.95342	16.6113	2.41913
0.0305166	0.01831	0.0305166	0.0335683	2.36504	0.979584	17.6539	2.33452
0.0332186	0.024159	0.0271788	0.0332186	2.38268	0.972398	17.3673	2.39778
0.0293015	0.0263713	0.020511	0.0380919	2.27086	1.0695	17.156	2.52579
0.0240768	0.0331056	0.0180576	0.0391248	2.39564	0.954043	17.2631	2.46185
0.0170935	0.0398849	0.0227914	0.0341871	2.42728	0.999972	17.8001	2.59822
0.0211289	0.0332025	0.0301841	0.0301841	2.24268	1.0655	17.706	2.61696
0.0210078	0.0330122	0.0330122	0.0240089	2.22382	1.12542	17.7186	2.62897
0.0293427	0.0240077	0.0320102	0.0293427	2.39277	1.06167	17.3736	2.86758
0.0212398	0.0303426	0.0273083	0.0333768	2.21197	1.06199	16.8159	2.87648
0.0240971	0.0271093	0.0331335	0.0271093	2.20489	1.07232	16.9433	2.81334
0.0332708	0.0211723	0.0362954	0.024197	2.23519	1.06769	17.2131	2.64957
0.0330489	0.0210311	0.02704	0.0360534	2.20526	1.06958	16.7438	2.9053
0.0239192	0.026909	0.0298989	0.026909	2.28129	0.995635	17.1142	2.6909
0.0302939	0.0212057	0.0181763	0.039382	2.15692	1.06634	16.8919	2.85671
0.02611371	0.02783265	0.02833286	0.03199781	2.32192800	1.02424095	17.27133000	2.55499300
0.00429518	0.00530773	0.00491715	0.00331081	0.13732087	0.04275692	0.42655347	0.09454779
10	10	10	10	10	10	10	10
1.96	1.96	1.96	1.96	1.96	1.96	1.96	1.96
0.00266218	0.00328977	0.00304768	0.00205206	0.08511236	0.02650101	0.26438058	0.05860133
0.02877589	0.03112242	0.03138054	0.03404986	2.40704036	1.05074196	17.53571058	2.61359433
0.02345153	0.02454288	0.02528517	0.02994575	2.23681564	0.99773994	17.00694942	2.49639167
0.03321860	0.03372990	0.03372990	0.03912480	2.61831000	1.06950000	17.80620000	2.52579000
0.02134520	0.01831000	0.01805760	0.02961470	2.21863000	0.94106600	16.56990000	2.23537000
0.01187340	0.01541990	0.01567230	0.00951010	0.39968000	0.12843400	1.23630000	0.29042000
0.00001845	0.00002817	0.00002418	0.00001096	0.01885702	0.00182815	0.18194787	0.00893928
0.00135826	0.00167845	0.00155494	0.00104697	0.04342467	0.01352092	0.13488805	0.02989864
0.02771215	0.02825650	0.02903750	0.03321115	2.37386000	0.97599100	17.40545000	2.36073500

Remotely placed modules with power management disabled and local CPU frequency set to user space mode 1.6GHz

	battery percentage change	Local CPU UTIME	Local CPU NICE	Local CPU STIME	Local CPU IDLE	Local CPU IO WAIT
1	0.00427932	9551	0	3403	19744	54
2	0.00466835	11300	0	3838	20906	114
3	0.00447387	10538	0	3698	21052	123
4	0.00525194	12629	0	4280	22991	200
5	0.00447387	10856	0	3601	19509	50
6	0.00427932	9283	0	3126	18666	82
7	0.00447387	10284	0	3206	19017	51
8	0.00427938	9662	0	3025	18212	138
9	0.00427932	10357	0	3389	18555	48
10	0.00408483	10293	0	3542	18142	94
11	0.00487614	12130	0	4037	22164	73
12	0.00409597	10236	0	3476	18358	57
13	0.00409597	10444	0	3606	18287	47
14	0.00390095	8876	0	2806	18100	83
15	0.00409591	10223	0	3370	18067	90
16	0.004291	10145	0	3371	17977	47
17	0.00448602	10613	0	3423	18553	47
18	0.00429106	10024	0	3068	18273	64
19	0.004291	10527	0	3395	18382	94
20	0.004291	10464	0	3638	18194	52
Average	0.00436295	10421.75000000	0.00000000	3464.90000000	19157.45000000	80.40000000
Standard Deviation	0.00032355	971.17386588	0.00000000	372.12805442	1550.46130770	49.48445323
Sample Size	10	10	10	10	10	10
Confidence Coff.	1.96	1.96	1.96	1.96	1.96	1.96
Margin of Error	0.00020054	601.93979836	0.00000000	230.64735771	960.98587463	30.67078187
Upper Bound	0.00456349	11023.68979836	0.00000000	3695.54735771	20118.43587463	111.07078187
Lower Bound	0.00416242	9819.81020164	0.00000000	3234.25264229	18196.46412537	49.72921813
Max	0.00525194	12629.00000000	0.00000000	4280.00000000	22991.00000000	200.00000000
Min	0.00408483	9283.00000000	0.00000000	3025.00000000	18142.00000000	48.00000000
Range	0.00116711	3346.00000000	0.00000000	1255.00000000	4849.00000000	152.00000000
Variance	0.00000010	943178.67777778	0.00000000	138479.28888889	2403930.26666667	2448.71111111
Standard Error	0.00010232	307.11214202	0.00000000	117.67722332	490.29891563	15.64835810
Median	0.00437663	10325.00000000	0.00000000	3472.50000000	19263.00000000	88.00000000

Remote CPU working time (jiffies)	Remote CPU working time (s)	Between first and last packet (s)	Number of sent packets	Avg. packets/sec	Avg. packet size (bytes)
17065	85.325	90.841	79630	876.59	481.873
18278	91.39	97.709	86811	888.469	475.811
18018	90.09	94.992	73299	771.63	531.147
20628	103.14	103.367	92804	897.807	473.215
17088	85.44	90.301	79656	882.112	482.211
16371	81.855	86.091	76106	884.014	478.916
16899	84.495	86.598	78651	908.227	470.206
16328	81.64	84.416	76160	902.196	469.243
16509	82.545	86.434	77214	893.334	475.731
16001	80.005	85.15	76884	902.928	470.458
19815	99.075	104.812	95915	915.118	467.182
16398	81.99	86.709	79519	917.082	466.083
16377	81.885	86.286	79033	915.944	466.654
16270	81.35	84.687	77316	912.965	467.051
16199	80.995	85.208	77895	914.176	466.326
15942	79.71	84.084	77562	922.439	464.308
16726	83.63	88.856	81393	916.013	466.721
16383	81.915	86.405	79210	916.727	466.491
16528	82.64	88.522	80759	912.309	467.052
16348	81.74	85.178	78431	920.785	464.869
17008.55000000	85.04275000	89.33230000	80212.40000000	898.54325000	473.57740000
1371.37589061	6.85687945	6.27709498	5811.05307439	39.65496523	18.26446915
10	10	10	10	10	10
1.96	1.96	1.96	1.96	1.96	1.96
849.98758314	4.24993792	3.89058378	3601.72801056	24.57840209	11.32043526
17858.53758314	89.29268792	93.22288378	83814.12801056	923.12165209	484.89783526
16158.56241686	80.79281208	85.44171622	76610.67198944	873.96484791	462.25696474
20628.00000000	103.14000000	103.36700000	92804.00000000	908.22700000	531.14700000
16001.00000000	80.00500000	84.41600000	73299.00000000	771.63000000	469.24300000
4627.00000000	23.13500000	18.95100000	19505.00000000	136.59700000	61.90400000
1880671.83333333	47.01679583	39.40192143	33768337.83333330	1572.51626779	333.59083321
433.66713426	2.16833567	1.98499172	1837.61633192	12.54000107	5.77573228
16982.00000000	84.91000000	88.44950000	77932.50000000	890.90150000	475.77100000

Avg. bytes/sec	Avg. MBit/sec	RVIZ (utime) (%)	RVIZ (stime) (%)	camera/rectify_ir (utime) (%)	camera/rectify_ir (stime) (%)	camera/register_depth_rgb (utime) (%)
422404.959	3.379	4.95752	6.11778	0	0.0703194	0
422743.287	3.382	5.12638	6.34096	0	0.0601816	0
409848.993	3.279	4.1625	5.64991	0	0.0610501	0
424856.001	3.399	5.17258	6.12759	0	0.0630211	0
425364.147	3.403	5.14396	5.81695	0	0.0819288	0.00585206
423368.501	3.387	5.09437	5.95565	0	0.067192	0
427053.895	3.416	5.16599	6.06545	0	0.0650926	0
423349.778	3.387	5.29152	6.07545	0	0.0673689	0
424986.028	3.4	5.23351	5.89375	0.0060573	0.060573	0
424789.318	3.398	5.23092	6.26211	0	0.0624961	0
427526.567	3.42	5.81882	6.33863	0.00504668	0.0605602	0
427436.268	3.419	6.08611	6.24466	0	0.0731797	0
427428.642	3.419	5.59932	6.49692	0	0.0427429	0
426401.726	3.411	5.34726	6.72403	0	0.0676091	0
426303.562	3.41	5.71023	6.4325	0	0.055559	0
428295.738	3.426	6.15983	6.26019	0	0.0564546	0
427522.384	3.42	5.6738	6.3195	0	0.0717446	0
427645.185	3.421	5.51181	6.62882	0	0.054935	0
426095.88	3.409	5.83858	6.39521	0	0.054453	0
428044.148	3.424	5.65207	6.50844	0	0.0672865	0
425073.25035000	3.40045000	5.39885400	6.23272500	0.00060573	0.06318741	0.00029260
4785.26110024	0.03818668	0.32752120	0.20684677	0.00191549	0.00654387	0.00185058
10	10	10	10	10	10	10
1.96	1.96	1.96	1.96	1.96	1.96	1.96
2965.93555797	0.02366835	0.20299974	0.12820496	0.00118723	0.00405593	0.00114700
428039.18590797	3.42411835	5.60185374	6.36092996	0.00179296	0.06724334	0.00143961
422107.31479203	3.37678165	5.19585426	6.10452004	-0.00058150	0.05913148	-0.00085440
427053.89500000	3.41600000	5.29152000	6.34096000	0.00605730	0.08192880	0.00585206
409848.99300000	3.27900000	4.16250000	5.64991000	0.00000000	0.06018160	0.00000000
17204.90200000	0.13700000	1.12902000	0.69105000	0.00605730	0.02174720	0.00585206
22898723.79745130	0.00145822	0.10727014	0.04278559	0.00000367	0.00004282	0.00000342
1513.23242754	0.01207569	0.10357130	0.06541069	0.00060573	0.00206935	0.00058521
424078.90950000	3.39250000	5.15497500	6.07045000	0.00000000	0.06405685	0.00000000

camera/register_depth_rgb (stime) (%)	camera/points_xyzrgb_sw_registered (utime) (%)	camera/points_xyzrgb_sw_registered (stime) (%)	camera/depth_registered_rectify_depth (utime) (%)
0.0703194	0	0.0703194	0.00585995
0.0492395	0	0.0711238	0
0.0555001	0	0.0555001	0
0.0630211	0	0.0581734	0
0.0643727	0	0.0643727	0
0.067192	0	0.0610836	0
0.0650926	0	0.0650926	0
0.0612445	0	0.0489956	0
0.0726876	0	0.060573	0
0.0499969	0	0.0499969	0
0.0555135	0	0.0555135	0
0.079278	0	0.079278	0
0.0427429	0	0.0427429	0
0.0614628	0	0.0491703	0
0.0617322	0	0.055559	0
0.0690001	0	0.0564546	0
0.0597872	0	0.0597872	0
0.0427272	0.00610389	0.0610389	0
0.0484027	0	0.0484027	0.00605034
0.0795204	0	0.0672865	0.00611696
0.06094167	0.00030519	0.06006769	0.00090136
0.00800123	0.00000000	0.00759235	0.00185308
10	10	10	10
1.96	1.96	1.96	1.96
0.00495921	0.00000000	0.00470579	0.00114855
0.06590088	0.00030519	0.06477348	0.00204991
0.05598246	0.00030519	0.05536190	-0.00024719
0.07268760	0.00000000	0.07112380	0.00585995
0.04923950	0.00000000	0.04899560	0.00000000
0.02344810	0.00000000	0.02212820	0.00585995
0.00006402	0.00000000	0.00005764	0.00000343
0.00253021	0.00000000	0.00240091	0.00058600
0.06369690	0.00000000	0.06082830	0.00000000

camera/depth_registered_rectify_depth (stime) (%)	camera/points_xyzrgb_hw_registered (utime) (%)	camera/points_xyzrgb_hw_registered (stime) (%)	camera/disparity_depth (utime) (%)
0.0585995	0	0.0703194	0
0.0656527	0	0.0547106	0
0.0666001	0	0.0666001	0
0.0727167	0	0.0678689	0
0.0643727	0	0.0702247	0
0.0733003	0	0.067192	0
0.0710101	0	0.0650926	0
0.05512	0	0.0673689	0
0.0666303	0	0.0787449	0
0.0499969	0	0.0562465	0
0.0605602	0	0.0605602	0
0.0670814	0	0.079278	0
0.048849	0	0.048849	0
0.0553165	0	0.0614628	0.00614628
0.0617322	0.00617322	0.0617322	0
0.0627274	0	0.0564546	0
0.0478297	0	0.0657659	0
0.0610389	0	0.0671428	0
0.054453	0	0.0605034	0
0.0795204	0.00611696	0.0489356	0
0.06215540	0.00061451	0.06375266	0.00030731
0.00767879	0.00000000	0.00689268	0.00000000
10	10	10	10
1.96	1.96	1.96	1.96
0.00475936	0.00000000	0.00427213	0.00000000
0.06691476	0.00061451	0.06802478	0.00030731
0.05739604	0.00061451	0.05948053	0.00030731
0.07330030	0.00000000	0.07874490	0.00000000
0.04999690	0.00000000	0.05471060	0.00000000
0.02330340	0.00000000	0.02403430	0.00000000
0.00005896	0.00000000	0.00004751	0.00000000
0.00242825	0.00000000	0.00217966	0.00000000
0.06612640	0.00000000	0.06728045	0.00000000

camera/disparity_depth (stime) (%)	camera/disparity_registered_sw (utime) (%)	camera/disparity_registered_sw (stime) (%)	camera/disparity_registered_hw (utime) (%)
0.0644594	0	0.0644594	0
0.0711238	0	0.0547106	0
0.0666001	0	0.0555001	0
0.0727167	0	0.0630211	0
0.0643727	0	0.0702247	0
0.0733003	0	0.0794087	0
0.0650926	0	0.0710101	0
0.0673689	0	0.0612445	0
0.0726876	0	0.0666303	0
0.0624961	0	0.0499969	0
0.0555135	0.00504668	0.0605602	0.00504668
0.0670814	0	0.0670814	0
0.048849	0	0.048849	0
0.0676091	0.00614628	0.0491703	0
0.0493858	0	0.0617322	0
0.0627274	0	0.0501819	0
0.0777233	0.00597872	0.0538084	0
0.0671428	0	0.0671428	0
0.0605034	0	0.0484027	0
0.0734035	0	0.0672865	0
0.06550787	0.00085858	0.06052109	0.00025233
0.00406700	0.00000000	0.00878247	0.00000000
10	10	10	10
1.96	1.96	1.96	1.96
0.00252075	0.00000000	0.00544343	0.00000000
0.06802862	0.00085858	0.06596452	0.00025233
0.06298712	0.00085858	0.05507766	0.00025233
0.07330030	0.00000000	0.07940870	0.00000000
0.06249610	0.00000000	0.04999690	0.00000000
0.01080420	0.00000000	0.02941180	0.00000000
0.00001654	0.00000000	0.00007713	0.00000000
0.00128610	0.00000000	0.00277726	0.00000000
0.06698450	0.00000000	0.06374025	0.00000000

camera/disparity_registered_hw (stime) (%)	map_server (utime) (%)	map_server (stime) (%)	amcl (utime) (%)	amcl (stime) (%)	move_base (utime) (%)	move_base (stime) (%)
0.0585995	0	0.0644594	0.492236	2.092	3.28743	5.41459
0.0547106	0	0.0656527	0.514279	2.19389	3.2334	5.53124
0.0555001	0	0.0666001	0.49395	1.77045	3.441	4.6953
0.0678689	0	0.0678689	0.591429	2.12333	3.40314	4.73143
0.0760768	0	0.0819288	0.456461	2.08919	3.23034	5.00351
0.0610836	0	0.0794087	0.525319	2.04019	3.44512	4.85615
0.0650926	0	0.0591751	0.443813	2.12439	3.34931	4.94112
0.0612445	0	0.0734934	0.459334	2.11293	3.57055	4.85056
0.0726876	0	0.0787449	0.454298	2.08371	3.27094	5.00333
0.0499969	0	0.0687457	0.456221	2.16861	3.16855	5.30592
0.0555135	0.00504668	0.0555135	0.519808	2.31643	3.63361	5.14762
0.0731797	0	0.0731797	0.518356	2.30516	3.51872	5.64093
0.0549551	0	0.0549551	0.519021	2.26537	3.54155	5.581
0.0553165	0	0.0737554	0.473264	2.30486	3.49723	5.26122
0.055559	0	0.0740786	0.500031	2.27175	3.5496	5.32132
0.0627274	0	0.0564546	0.464183	2.32719	3.8891	5.76465
0.0717446	0	0.0717446	0.526127	2.24202	3.40189	5.46455
0.0488311	0	0.0610389	0.488311	2.25233	3.17402	5.59727
0.054453	0	0.0484027	0.0484027	2.31123	3.35189	6.06244
0.0795204	0	0.0795204	0.458772	2.28774	3.89038	5.03426
0.06173307	0.00025233	0.06773606	0.47018079	2.18413850	3.44238850	5.26042050
0.00821924	0.00000000	0.00745545	0.04563561	0.11709592	0.12418388	0.28830468
10	10	10	10	10	10	10
1.96	1.96	1.96	1.96	1.96	1.96	1.96
0.00509434	0.00000000	0.00462093	0.02828524	0.07257681	0.07696996	0.17869309
0.06682741	0.00025233	0.07235699	0.49846603	2.25671531	3.51935846	5.43911359
0.05663873	0.00025233	0.06311513	0.44189554	2.11156169	3.36541854	5.08172741
0.07607680	0.00000000	0.08192880	0.59142900	2.19389000	3.57055000	5.53124000
0.04999690	0.00000000	0.05917510	0.44381300	1.77045000	3.16855000	4.69530000
0.02607990	0.00000000	0.02275370	0.14761600	0.42344000	0.40200000	0.83594000
0.00006756	0.00000000	0.00005558	0.00208261	0.01371146	0.01542163	0.08311959
0.00259915	0.00000000	0.00235762	0.01443125	0.03702898	0.03927039	0.09116994
0.06116405	0.00000000	0.06830730	0.47578500	2.10246500	3.31837000	4.97222500

Appendix V: Network profiling code

```
/*************************************************************************************************/
/*************************************************************************************************/
/******************FUNCTIONS CONNECTED TO GATHERING*************************/
/******************INFORMATION FOR MACHINE LEARNING*************************/
/*************************************************************************************************/
/*************************************************************************************************/
vector<float> parseIfstatData(char* buf){
	vector<string>	strings;
	istringstream	iss(buf);
	string		s;
	char*		buff;

	while(getline(iss, s, '\n')){
		strings.push_back(s);
	}

	strings.clear();
	buff = (char*) strings[2].c_str();
	istringstream fs(buff);
	while(getline(fs, s, ' ')){
		strings.push_back(s);
	}
	vector<float> ret;
	for(int i=0; i<strings.size(); i++){
		if(strings[i].length() > 0){
			ret.push_back(atof(strings[i].c_str()));
		}
	}
	return ret;
}

void getIfstatData(void){
	int		fd[2];
	pid_t		pid;
	int		childStatus;

	if(pipe(fd) == -1){
		perror("pipe");
		exit(1);
	}

	pid = fork();
	if(pid == -1){
		perror("fork");
		exit(1);
	}else if(pid == 0){
		while((dup2(fd[1], STDOUT_FILENO) == -1) && (errno == EINTR)){}
		close(fd[0]);
		close(fd[1]);
		execl("/usr/bin/ifstat", "/usr/bin/ifstat", "-i", "wlan0", "0.1", "1", (char*) 0);
		perror("execl");
		_exit(1);
	}

	close(fd[1]);
	while(true){
		ssize_t count = read(fd[0], buffer, sizeof(buffer));
		if(count == -1){
```

```
                if(errno == EINTR){
                        continue;
                }else{
                        fprintf(stdout, "Couldn't read from child process\n");
                        exit(1);
                }
        }else if(count == 0){
                break;
        }else{
                break;
        }
    }
    close(fd[0]);
    waitpid(pid, &childStatus, WEXITED);
    return;
}

float parseNetperfData(char* buf){
    vector<string>      strings;
    istringstream       iss(buf);
    string              s;
    char*               buff;

    while(getline(iss, s, '\n')){
        strings.push_back(s);
    }

    strings.clear();
    buff = (char*) strings[6].c_str();
    istringstream fs(buff);
    while(getline(fs, s, ' ')){
        strings.push_back(s);
    }
    vector<float> values;
    for(int i=0; i<strings.size(); i++){
        if(strings[i].length() > 0){
            values.push_back(atof(strings[i].c_str()));
        }
    }
    return values[values.size()-1];
}

void getNetperfData(void){
    int         fd[2];
    pid_t       pid;
    int         childStatus;

    if(pipe(fd) == -1){
        perror("pipe");
        exit(1);
    }

    pid = fork();
    if(pid == -1){
        perror("fork");
        exit(1);
    }else if(pid == 0){
        while((dup2(fd[1], STDOUT_FILENO) == -1) && (errno == EINTR)){}
        close(fd[0]);
        close(fd[1]);
```

```
            execl("/usr/bin/netperf", "/usr/bin/netperf",  "-H", "133.1.134.124", (char*) 0);
            perror("execl");
            _exit(1);
        }

        close(fd[1]);
        while(true){
            ssize_t count = read(fd[0], buffer, sizeof(buffer));
            if(count == -1){
                if(errno == EINTR){
                    continue;
                }else{
                    fprintf(stdout, "Couldn't read from child process\n");
                    exit(1);
                }
            }else if(count == 0){
                break;
            }else{
                break;
            }
        }
        close(fd[0]);
        waitpid(pid, &childStatus, WEXITED);
        return;
}

std::string floatToString(float number){
        std::ostringstream  buf;
        buf << number;
        return buf.str();
}

std::string intToString(int number){
        std::ostringstream  buf;
        buf << number;
        return buf.str();
}

void getCpuStat(const char* proc_stat_file_path){
        procStatTokens.clear();
        ifstream    proc_stat_file(proc_stat_file_path);
        if(proc_stat_file.is_open()){
            for(int i=0; i<12; i++){
                getline(proc_stat_file, token, ' ');
                procStatTokens.push_back(token);
            }
        }else{
            fprintf(stderr, "No file path: %s\n",
                proc_stat_file_path);
            exit(1);
        }
        proc_stat_file.close();
        return;
}

unsigned long getCpuStat(void){
        unsigned long       ret = 0;
        for(int i=2; i<12; i++){
            ret = ret + strtoul(procStatTokens[i].c_str(), NULL, 0);
        }
```

```
        return ret;
}

float getCurrentBatteryCapacity(void){
        string info;
        ifstream file(CHARGE_FULL_PATH);
        getline(file, info);
        float charge_full = atof(info.c_str());
        ifstream file2(CHARGE_NOW_PATH);
        getline(file2, info);
        float charge_now = atof(info.c_str());
        file.close();
        file2.close();
        return (charge_now/charge_full);
}
/*************************************************************************************************/
/*************************************************************************************************/
/******************FUNCTIONS CONNECTED TO GATHERING************************/
/*******************INFORMATIO FOR MACHINE LEARNING*************************/
/*************************************(END)*****************************************************/
/*************************************************************************************************/
/*************************************************************************************************/
```

Appendix VI: COMPSACW2014 Paper

Some of the work in this thesis (most notably Section 4.1) were taken from one of the author's own papers (Nordlund et al., 2014). IEEE (publisher of the proceedings of the Computer Software and Applications Conference Workshops, COMPSACW) who accepted this article state that the use of the author's work in their own thesis is allowed.

Designing of a Network-Aware Cloud Robotic Sensor Observation Framework

Fredrik NORDLUND†, Manabu HIGASHIDA†, Yuuichi TERANISHI†‡,
Shinji SHIMOJO†‡, Masanori YOKOYAMA⋆, Michio SHIMOMURA⋆
† Osaka University
‡ National Institute of Information and Communications Technology
⋆ Nippon Telegraph and Telephone Corporation
fredrik.nordlund@ais.cmc.osaka-u.ac.jp, {manabu, teranisi, shimojo}@cmc.osaka-u.ac.jp,
{yokoyama.masanori, shimomura.michio}@lab.ntt.co.jp

***Abstract*—This paper proposes a network-aware cloud networked robotic sensor observation framework based on the topic-based publisher/subscriber messaging paradigm. Cloud network robotics systems have some advantages in making use of rich computation or storage resources of clouds, low maintenance cost, and easy development of applications. But a problem occurs when systems that need fast feedback from application modules are run over networks with intermittent connection. In this framework, the placement of the robot application modules is changed according to the QoS provided by the physical network. We also propose a mapcell-based robotic sensor observation algorithm that suits to the framework. A prototype system of a data center monitoring robot that is based on the proposed framework and algorithm is implemented and a preliminary experiment was carried out to demonstrate the benefit of the system. The experiment result shows that the performance of the data center monitoring could be improved and hot spots can be found faster by making the cloud networked robot network-aware.**

***Keywords*—cloud networked robotics; sensor networks; network qos; big data analysis; publish subscribe messaging paradigm;**

I. Introduction

Cloud Networked Robotics (CNR) has been paid great attention and many development efforts have been made for CNR systems, such as [1], [2], and [3]. In CNR, robot application modules can be more efficient if they are placed on a cloud since this saves computation power, memory, and battery consumption on the robots. Furthermore, application modules are easier to maintain since updates to the application module software does not have to be done for each and every robot, thus reducing maintenance cost. It is also beneficial to put some of the robot modules on a cloud, since the cloud can provide the modules with rich computation resources that can be used for complex computations, and storages for large amount of data [4]. Another advantage of CNR is that it makes it easier for programmers to write a program without having detailed knowledge about robots. Some applications in areas such as building management [5], agriculture [6], and the medical field [7], have been developed and demonstrate the effectiveness of CNR.

One of the most significant applications of CNR is to observe and analyze real-time situations of the real-world. Robots can move around uncertain remote areas and observe sensor data of a target region. The sensor data can then be analyzed and the results can be used as input to a robot's next movement plan. This can for example be used by a building-management system, where sensor data of temperature/humidity is gathered up in a cloud to be processed and then forwarded to a facility management system, in order to improve the efficiency of air conditioners. Based on this concept, we are developing an automatic data center monitoring system that uses cloud robotics and CFD-simulations (Fig. II) [8].

Existing CNR-based systems assume stable network environments in which the robots can always contact the processes on the cloud immediately. However, this assumption does not match the case with uncertain remote environments such as wide farms and disaster areas, where the condition of a wireless network may not always be stable. Such situations may easily occur even in an office or a data center where the target area may be separated by objects or walls, which results in large variations in network conditions. In order to apply CNR-based robots to this kind of environments, we must assume that network connections are intermittent or can only achieve narrow bandwidth transmissions. There has been a study to apply Delay/Disruption Tolerant Networks (DTN) to CNR in order to keep the data transfer stable in environments with intermittent network connections, but it causes high latency when the condition of the physical wireless network degrades [9]. Although the problem with intermittent connections is solved by using DTN, the requirements of the feedback speed or bandwidth for robots are not always satisfied.

To tackle this problem with potentially high latency and narrow bandwidth, modules that require fast feedback or high bandwidth could be placed on robots so that the system can always get a fast response. This solution is not optimal since it goes against the policy of CNR-based systems by increasing the computation load on robots and not making use of the rich computation power on the cloud. Since we must consider the scenarios concerning uncertain remote areas, where uniform wireless network quality cannot be assumed, we can say that if a network link has high bandwidth and low latency, the modules would most likely be better off placed on the cloud. On the other hand, if the network condition were bad with low bandwidth and high latency, these modules would be better off placed inside the robot. This means that the most suitable module placement may dynamically change, since in some places the network is stable, and in other places unstable.

In this paper we present a framework for making a cloud robotic sensor observation system "network-aware" in order for it to be able to change the placement of the modules to the most suitable place, following the QoS (Quality of Service) of the network. The contributions of this paper are as follows:

- Presents a network-aware cloud robotic sensor frame-

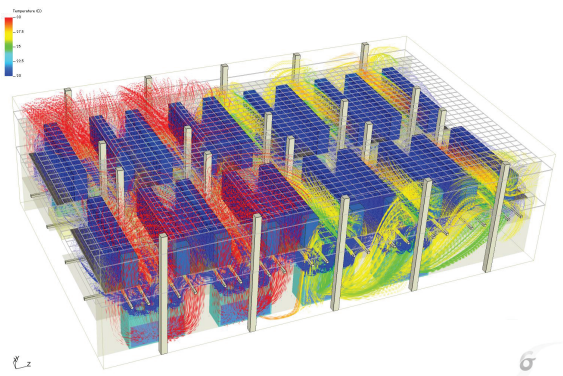

Fig. 1. A CFD-simulation of the data center [8].

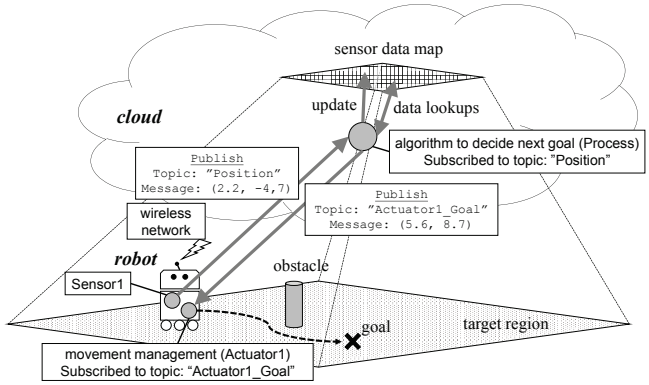

Fig. 2. The assumed environment where a cloud has a sensor data map of a target region and decides goals for a robot.

work that is based on the Topic-Based Publisher Subscriber (TBPS) messaging paradigm.

- Proposes a mapcell-based robotic sensor observation algorithm that suits to the network-aware cloud robotic sensor framework. The algorithm consists of two types of goal decision algorithms, global goal (GG) and measurement feedback goal (MFG).
- Shows the effectiveness of the framework and algorithm through an experiment using a prototype implementation of a data center monitoring robot on ROS [1].

II. Assumed Environment

This section presents an overview of the system structure and mission, as well as the messaging paradigm assumed in this study.

A. System Structure and Mission

Fig. II-B shows an overview of our assumed environment. The mission of the system is to create a sensor data map for the target region, which plots observed sensor value. In conjunction with the mission, we envision an automatic data center monitoring system where robots moves around in the data center and monitor the temperature. If any abnormal values are found, the robots should try to find, and if possible correct, a temperature hot spot as soon as possible. The data is also to be gathered up in the cloud be used as the input for the CFD-simulation module.

Each of the robots is assumed to be equipped with sensors, such as a temperature sensor, a humidity sensor, a camera sensor, and so on. According to the sensor data observed by the sensors, the next goal, which means the next sensing target position for the robot is decided. The next sensing target positions are decided by some algorithms to achieve fast construction of a sensor data map. The algorithm should be changed depending on the application or sensing target, such as an algorithm which selects a position where the oldest sensing value is mapped, an algorithm which selects a position where an abnormal sensor value is estimated by a calculation of trends and patterns of the sensor data, etc.

We also assume that a movement management process exists on the robots and that the robots operate in a wireless network environment with intermittent connections. The movement management process detects and avoids obstacles to reach to the specified goal. If the process modules in the cloud and the robots were to be connected together in a DTN-network to uphold connectivity, there would be very high latency and small bandwidth. But when the robot is located at the position where the wireless network is stable, the robot can have good bandwidth and low latency.

B. Messaging Paradigm (TBPS)

The topic-based publish/subscribe (TBPS) messaging paradigm is assumed to run on all robot modules because it enables the development of scalable and distributed information processing systems [10], which is essentially what a cloud network robots system is. An example of a CNR system that uses TBPS is ROS [1], which adopts TBPS as its base messaging mechanism for inter-module communications.

An example of the TBPS messaging paradigm can be seen in Fig. II-B. A process might need position data from a sensor, but instead of asking the sensor directly for the robot position, it can subscribe to a topic with the name `Position`. A topic works as a message filter that only handle messages that contain the same type of data that the topic is configured for. The sensor can publish its position data on this topic and all subscribers of the topic will receive the same data. After the sensor data has been processed, the processor can publish an actuation command on the topic `Actuator1_Goal`, so that "Actuator1" can move the robot to a goal in the target region.

III. Network-Aware Cloud Robotic Framework

In this section, we present a network-aware cloud robotic framework that is based on the TBPS messaging paradigm. In this framework, the modules change their placement to the most suitable place (i.e. on the cloud or on the robot) following the current QoS of the network.

We treat the following QoS for TBPS messaging:

- **Latency:** message propagation delay between the publisher and subscriber. If DTN is applied as a base network system, the latency can be very high, for example from 300sec - 1day. Normally, when Wi-Fi is applied as a base network system, the latency may be for example 1ms - 10ms.
- **Bandwidth:** amount of data sent within a certain period of time. In a Wi-Fi environment, the bandwidth

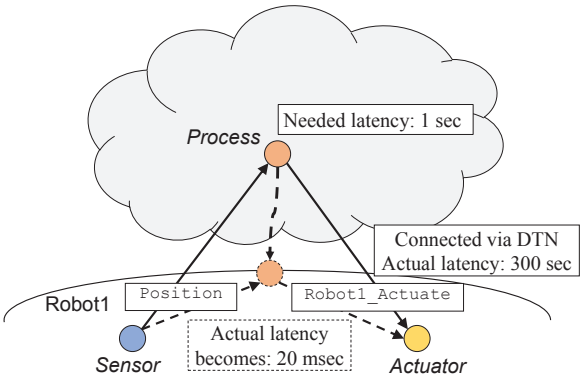

Fig. 3. A case where a process running on the cloud would be better placed in the robot because of intolerable latency.

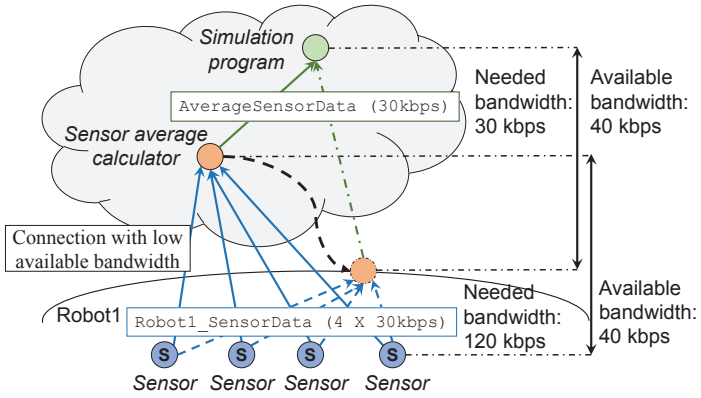

Fig. 4. A case where a module running on the cloud would be better placed in the robot because of low available bandwidth.

may be 10Mbps - 50Mbps, while in a Bluetooth SPP environment, the bandwidth may be 10kbps - 500kbps.

In our framework, the current latency (referred to as "actual latency") and available bandwidth are monitored. Every module that publishes or subscribes to a topic, specifies its required latency as "needed latency" and required bandwidth, for publishing or subscribing to a certain topic, as "needed bandwidth". When the monitored QoS does not satisfy the needed QoS in the current module placement, it considers moving the module to the robot from the cloud. Since the publisher module does not need to worry about the place where the subscriber module is running when using TBPS, there is no need to modify the program code on the modules.

The case seen in Fig. II-B identifies the problem with placing a crucial module on the cloud when the network is subject to high latency. In this example, the sensors on "Robot1" publish its position on the topic `Position`, and the actuator on Robot1 subscribes to the topic `Robot1_Actuate` to receive actuation commands. The process on the cloud subscribes to the topic `Position` and when it receives a position message from Robot1, it decides the next action and publishes the command on the topic `Robot1_Actuate`. At this time, the process has a "needed latency" of 1 second. But Robot1 is connected via DTN, so the actual delay at this moment is 300 seconds. In order to satisfy the QoS, the module is migrated to the robot where the actual delay is 20 msec.

Using the QoS with bandwidth may be more complex. Such a case can be seen in Fig. II-B, where a robot is connected to a cloud via a link with low available bandwidth. Several sensors in the robot need to send their sensor data to a module that will calculate the average of all the data and forward this average to a simulation program. Each sensor sends data at a rate of 30kbps and there are four sensors, which means that the sum of "needed bandwidths" is 120kbps. The available bandwidth is only 40kbps and will therefore not be enough. Moving the average calculating module to the robot solves this problem since the average calculator module has a needed bandwidth of 30kbps, which is less than the available bandwidth of 40kbs.

But by always placing the modules in the robot, there is a valuable trade-off from maintenance cost, energy consumption, and possibilities to make use of the vast amount of different information available on the cloud. Both ways have their benefits and downfalls, so we therefore propose a network-aware cloud networked robot that monitors the network to get network data about the actual delay and available bandwidth, upon it may restructure itself to migrate modules freely between the cloud and the robot. In terms of adapting to network changes, a network-aware cloud robot would act according to the following:

- **Required QoS can be satisfied on cloud (good bandwidth and/or low latency):** The system migrates the modules to the cloud in order to increase the efficiency and make better use of available data, as well as decrease the usage of resources in the robot.
- **Required QoS cannot be satisfied on cloud (low bandwidth and/or high latency) or no network connection:** The system migrates the modules to the robot in order to continue to provide the functions with required QoS.

IV. Design of a Network-aware Cloud Robotic Sensor Observation System

In this section we propose a design of a network-aware cloud robotic sensor observation system. Firstly we introduce a system design that realizes the network-aware cloud robotic framework described in the former section. We then introduce an abstraction of a robot's movement planning that can do sensor observation while having measurement feedback. Lastly, we propose a mapcell-based robotic sensor observation algorithm that suits to the network-aware cloud robotic sensor framework, which is described in the former section. The algorithm consists of the two types of goal decision algorithms, global goal (GG) and measurement feedback goal (MFG).

A. Design of a network-aware cloud robotic system

Fig. IV shows a system design of the framework. We assume publisher/subscriber modules use pub/sub network layer on top of the lower layer networks.

The publisher/subscriber modules may specify required QoS in form of needed latency and bandwidth. The required QoS is specified with the publish and subscribe command messages. If a module does not have any requirements for network QoS, the module does not have to specify any required QoS. In this case, the module does not run as network-aware.

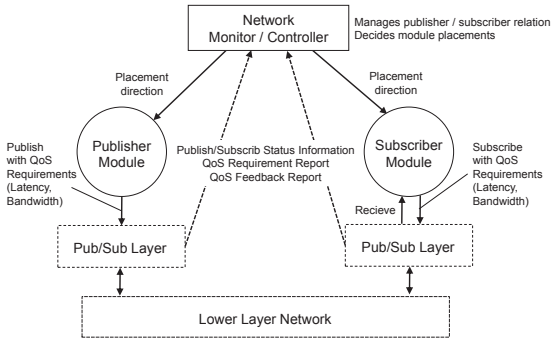

Fig. 5. The design of the network monitor.

The pub/sub network layer reports publish/subscribe status information and QoS requirements to a network monitor. The network monitor creates and keeps publisher and subscriber relations based on the received publish/subscribe status information. In addition, the network monitor maintains the QoS requirements for each publisher and subscriber relation.

The pub/sub network layer also measures the actual QoS, the actual latency and available bandwidth. The actual QoS is measured through passive monitoring of the network. This means that instead of actively sending messages to test the latency and bandwidth, actual application data is measured to obtain more accurate network situation. The actual QoS will be sent to the network monitor as a QoS feedback report. A usual problem with passive monitoring is that it relies on the data sent by applications, which means that the information acquired through this might be out-of-date. If the sensors always publish messages, the network monitor will be informed of any changes in the network. Otherwise, the publisher must send periodic test message to measure the actual latency and available bandwidth.

If any of the module's required QoS is not satisfied on network monitor, the network controller estimates the QoS in the alternative module placements keeping the publisher and subscriber relations. If QoS is considered to be satisfied in the other module placement, a placement direction command will be sent to a module to move itself to another position if needed.

B. Sensor Observation With Measurement Feedback

The problem faced in the assumed environment is that the network connection may suddenly go down or present very high latency. To be able to utilize the vast resources provided by the cloud to decide complex movement schemes, even when modules have to be migrated to the robot to satisfy the needed QoS, we propose to divide the robots movement planning into two different goal decision algorithms, global goal (GG) and measurement feedback goal (MFG).

A global goal (GG) is an abstraction of a complex objective, e.g. searching through a part of a room in order to find temperature hot spots. In order to complete such an objective the robot will need to perform many minor tasks, such as deciding where to continue to search after an abnormal value has been found, avoid running into objects, decide when a hot spot has been found, and so on. We call these many minor tasks that often involve reacting to the surrounding environment, measurement feedback goals (MFG). An important difference to take notice of between these two goal decisions algorithms are that they operate on different time scales. A GG might be completed every 3 to 15 minutes, while a MFG could be completed all between 50 milliseconds to 2 seconds.

If we apply these two goal decision algorithms to an example similar to the example presented in the previous section concerning high latency, we get a scenario that can be seen in Fig. IV-A. This time a robot with sensors and actuators are connected via DTN to a cloud with measurement feedback and global goal modules. Sensors on the robot send the robot's position to the measurement feedback, which decides a movement so that the robot gets closer to accomplishing the global goal. The robot expects a needed latency between itself and the measurement feedback to be less than 1 second, but the actual delay is over 300 seconds. By moving the measurement feedback module to the robot, the actual delay goes down to 20 ms, which is less than the needed latency of 1 second. The actual delay between the robot and the global goal module will still be up to 300 seconds, but is no problem since the global goal module expects that it will take over 360 seconds to complete a global goal. This enables the system to utilize the power of the cloud while having real-time feedback, even under unstable network conditions.

C. Global Goal

Before we can define a goal decision algorithm, we need to define an abstraction, which can hold information about a target region. We propose that a target region is divided into squares, which we call "MapCells". A MapCell contains information about the coordinates of a square area in a target region, temperature data, and the latest time an area was visited by a robot.

An example of an algorithm that can be used by the global goal module to decide which area the robot should scan next can be seen in Fig. IV-B. It is called the "least recently visited" algorithm and was used by [11]. Each MapCell contains the time when the MapCell was last visited by a robot and the algorithm uses this to decide which MapCell that was least recently visited. A room, presented as a black square, is divided into squares of MapCells, which can be placed on a x- and y-axis. Hereafter, (x, y) denotes a MapCell at the coordinates x and y. The six steps in the example that explains the algorithm are as follows:

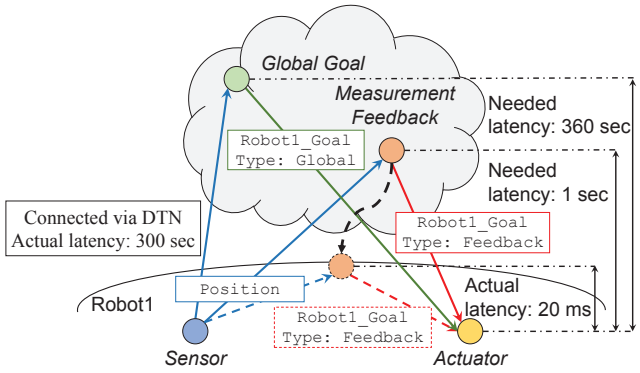

Fig. 6. An example that shows how the concept of global- and measurement feedback goals can use the vast resources of the cloud while providing real-time feedback to modules.

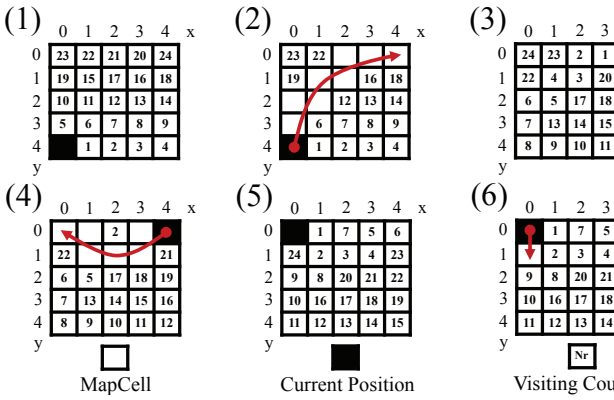

Fig. 7. The algorithm used by the global goal module to decide where the robot should scan for temperature hot spots.

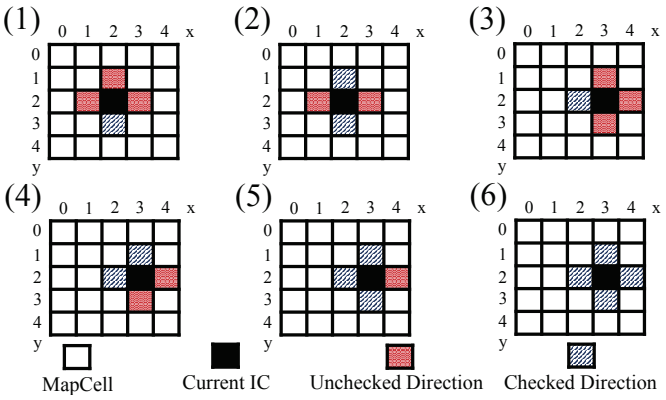

Fig. 8. The algorithm used to find a temperature hot spot when an abnormal value has been found and needs to be investigated.

- **Step 1):** The robot, placed at the black square, selects a MapCell to visit by finding the highest number; the higher the number, the longer time ago since a MapCell was visited. The MapCell that is least recently visited is (4, 0).
- **Step 2) and 3):** The robot moves to (4, 0) and recalculates all the values so the algorithm can find out which is the next least recently visited MapCell.
- **Step 4) and 5):** It is decided that (0, 0) is the next least recently visited MapCell and the robot moves there. When arriving at the MapCell, all the values are once again recalculated.
- **Step 6):** (0, 1) is found to be the next least recently visited MapCell. The robot moves there and this continues indefinitely or until another algorithm takes over. An example of this is the measurement feedback module that may start locating a temperature hot spot if an abnormal value has been found.

D. Measurement Feedback Goal

An example of an algorithm that can be used by the measurement feedback module to locate a temperature hot spot, when an abnormal value has been found, can be seen in Fig. IV-B. The algorithm can be explained by an example in six steps. The steps are as follows:

- **Step 1):** The algorithm starts when an abnormal value has been found. The first action is marking the area of the room ((2, 2)), where the abnormal value was found, as being the current investigation cell (IC). This is followed by marking the previously visited MapCell ((2, 3)) as being a checked direction, as well as marking all surrounding MapCells as unchecked directions that needs to be investigated.
- **Step 2):** The robot will start investigating the surrounding unchecked directions to see if their average temperature is larger or lower than the current IC. It chooses to start the investigation in (2, 1). The average temperature is found to be lower than that of the current IC, and (2, 1) is therefore marked as a checked direction.
- **Step 3:** The robot finds an area ((3, 2)) that has higher average temperature than that of the current IC, and therefore updates (3, 2) to be marked as the current IC. All surrounding MapCells are reset and once again marked. The previous IC is marked as a checked direction while the all the other surrounding MapCells are marked as unchecked directions.
- **Step 4) and 5):** The robot continues to investigate surrounding unchecked directions. It is found out that (3, 1) and (3, 3) has a lower average temperature than the current IC. Both MapCells are therefore marked as checked directions.
- **Step 6):** The last surrounding MapCell is found to also have a lower average temperature than that of the current IC, and the algorithm therefore decides that the hot spot has been found. The robot can now go back to completing the objective set by the global goal module.

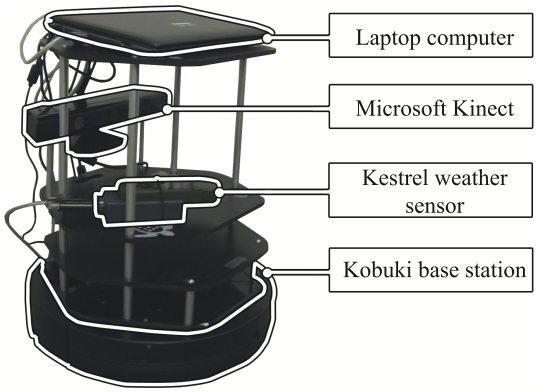

Fig. 9. The prototype implementation.

V. Prototype Implementation and Experiment

To show the feasibility and effectiveness of the framework presented in the former sections, we implemented a prototype system and conducted an experiment, based on the scenario with the data center presented in section II, to see how much time that could be saved from finding temperature hot spots when a measurement feedback module, used for finding hot spots, is placed in the robot instead of in the cloud.

A. Prototype Implementation

A prototype implementation of the network-aware cloud robotic sensor observation system was developed on top of ROS [1]. The prototype implementation uses the two algorithms described in section IV-C and IV-D, but does currently only support static module placements.

The Fig. IV-D shows the overview of the prototype implementation of the robot. The robot used was a "Turtlebot 2" [12] robot suite from Willow Garage, which includes a Microsoft Kinect, a Kobuki base station with factory calibrated gyro, a wood and steel frame for mounting modules, and a ROS compatible laptop computer (Netbook).

The Microsoft Kinect uses a camera, an infrared projector, and a specialized chip to recognize gestures, patterns, and depth. This can be used to create a 3D-map of a room, and with the help of the gyro of the Kobuki base station, the robot can locate itself in the map and navigate the room with simultaneous localization and mapping (SLAM) algorithms. The Kobuki base station is a movement base similar to the famous "Roomba" cleaning robot [13]. A laptop computer (Netbook), running Linux (Ubuntu), is used to run modules for robot controls such as a "move_base" module that takes goals in a map as input and forward actuation commands to the Kobuki base station, a program to fetch sensor data from the Kinect, and so on. At the cloud side, there is a normal PC-server with Linux (Ubuntu), and the laptop connects to the server on the cloud side via Wi-Fi connection (802.11n).

B. Experiment Setup

The experiment setup, seen in Fig. V-A, was designed to take the scenario presented in section II, where the server corridors could make network connection go bad, into account. The experiment setup was placed in a corridor, which is 2 meters wide and 12 meters long. A pair of electrical heaters was placed by the walls to simulate a temperature hot spot.

To simulate a network with intermittent connections, we statically configured an area in the corridor to be without network connection. As can be seen in Fig. V-A, two areas (each about 1.5 times 2 meters) at each end of the corridor are configured to have network connection, while all the area in between is configured to have no network connection.

The sensors were set to publish measurements at a rate of three times a second. The global goal module was configured to publish an actuation command on a topic called "Global_Goal", ordering the robot to move to the end of the corridor, back to near the beginning, and then to the end of the corridor once again, resulting in a movement similar to the letter "Z". The robot was instructed to run for an extra 50 measurements after a temperature hot spot was found, i.e. around an extra 16 seconds.

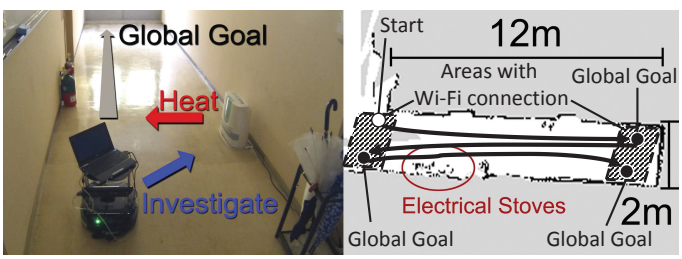

Fig. 10. The experiment setup in a corridor with heaters on the side to simulate a temperature hot spot.

The experiment was divided into two runs, one run where global goal and measurement feedback were placed on the cloud, and the other run where the global goal module is placed on the cloud while the measurement feedback module is placed in the robot. The second run can be seen as our "network-aware" system, having sensed or predicted the loss of network connection, and then migrated the MFG module to the robot in order to satisfy future QoS.

In the former experiment run (normal), sensor data (temperature and position) does not reach the measurement feedback module, since it does not meet the required "needed latency" set by the MFG module in this environment. That is, in the former experiment, only the global goal module runs to manage movement of the robot.

C. Experiment Results

The results from the experiment can be seen in Fig. V-C. The graph shows temperature on the y-axis and corresponding measurement number on the x-axis, essentially meaning that the two lines represents how temperature changes over time when the robot moves along the predetermined path. The dotted line, which is named "Normal" shows the experiment run where only the global goal module runs on the cloud. The straight line, called "Network-aware", represents the experiment run where the measurement feedback goal module is placed on the robot, while the global goal module is placed on the cloud.

The results show that while running down the corridor the prototype implementation with "network-aware" finds an abnormal value and start to investigate, while the other prototype implementation runs to the end of the corridor until it get network connection again and can go and investigate the area around the abnormal value. When the measurement feedback goal module is placed in the robot, the robot will investigate around the area of the heaters to finally find the hot spot. The difference in time between the two runs was roughly around 100 measurements; meaning around 33 seconds.

VI. Related Work

A QoS-driven framework for self-adaptive mobile applications have been proposed to support seamless configurations [14]. A middleware discovers alternative remote providers of functionalities required by an application to keep the SLA (Service-Level Agreement).

Some researches have been studied so far on 'mobile offloading', which aims to offload computing tasks to a cloud in order to save computational power and increase battery life on the local devices [15]–[17].

Our framework does not need specified remote-able methods since TBPS does not need to know where a module is, but rather what information that module subscribes to or publishes. Furthermore, our framework takes care of the movement planning module of the autonomous robots in the

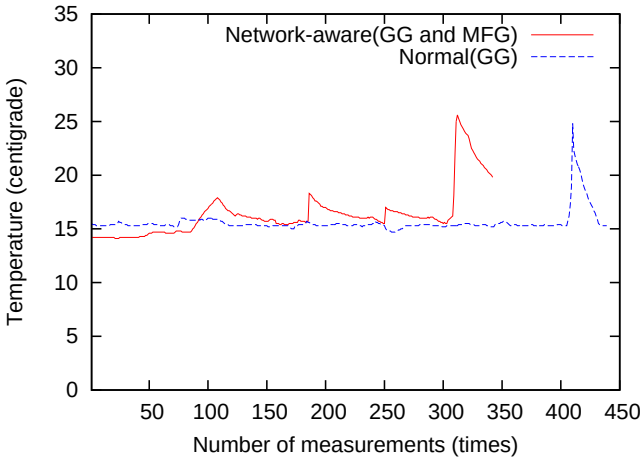

Fig. 11. The results from the experiment.

unstable wireless network, which is out of scope in the existing works.

VII. Conclusion and Future Work

In this paper we proposed a network-aware cloud networked robotic sensor observation framework based on the topic-based publisher/subscriber messaging paradigm. A real-life example scenario was also presented and an experiment was carried out to see how the example scenario could benefit from migrating modules from the cloud to the robot. Our preliminary results show that having modules in the robot could find hot spots faster than having the modules in the cloud, opening a possibility for making the data center more energy efficient and reduce running cost.

The next step is to fully implement network monitors into the system to be able to receive information about the network condition from the network. At this time, since the actual QoS is collected on the network monitor, the future QoS for each publisher/subscriber may be estimated. For this purpose, we will consider mechanisms to manage the actual QoS and the robot's geographical coordinates, as well as mechanisms to predict the future QoS.

Another aspect yet to be taken into consideration is the time it will take to move modules from one place to another. If degrading network conditions are predicted beforehand or the network condition is gradually getting worse, the modules can be moved without any problems. The problem is the cases where no changes in network condition is predicted and a sudden change in network condition occurs, leaving little to no time for the system to adapt. Furthermore, future work also includes investigating the trade-off between migrating modules to decrease CPU loads and additional networking required to migrate the modules.

Unlike the experiment in this paper, where modules are placed in a position known beforehand to be better or worse, future works will conduct experiments with dynamic network environments. The current system used in the experiment does not need the vast computing capacity or storage that a cloud provides, but the system we are aiming to create does. Network monitors will be implemented to monitor the network condition while evaluating if modules need to be moved to another position in order to satisfy QoS, at the same time as predicting future QoS and estimating the time it will take to move the modules to the new placement. This will be done in order to, as often as possible, prevent the scenarios where sudden changes occurs in the network condition.

References

[1] ROS development community, *Documentation ROS Wiki*, Available at: http://wiki.ros.org, Accessed Mar. 2014.

[2] N. Ando, T. Suehiro, K. Kitagaki, K. Kotoku, W. K.Yoon, *RT-Middleware: Distributed Component Middleware for RT (Robot Technology)*, 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2005), pp.3555-3560, Aug. 2005.

[3] Robot Service initiative, *Robot Service Network Protocol*, 5th Japan - Korea Service Robot Workshop, 26th Nov. 2009, Available at: http://robotservices.org, Accessed Mar. 2014.

[4] R. Arumugam, V. R. Enti, L. Bingbing, W. Xiaojun, K. Baskaran, F. F. Kong, A. S. Kumar, K. D. Meng, G. W. Kit, *DAvinCi: A Cloud Computing Framework for Service Robots*, in Proc. of 2010 IEEE International Conference on Robotics and Automation (ICRA 2010), pp.3084-3089, 3-7th May 2010.

[5] I. G. Alonso, P. Suarez, V. Curto, O. Alvarez Fres, *Work in progress: Smart home energy (SHE)*, 2011 2nd IEEE PES International Conference and Exhibition on Innovative Smart Grid Technologies (ISGT Europe), Manchester, pp.1-4, 5-7th Dec. 2011.

[6] Fujitsu Limited, *Fujitsu Expands Lineup of Akisai Food and Agriculture Cloud Services*, Tokyo, 24th Dec. 2013, Available at: http://www.fujitsu.com/global/news/pr/archives/month/2013/20131224-01.html, Accessed Mar. 2014.

[7] T. Yokoo, M. Yamada, S. Sakaino, S. Abe, *Development of a Physical Therapy Robot for Rehabilitation Databases*, 2012 12th IEEE International Workshop on Advanced Motion Control (AMC), Sarajevo, pp.1-6, Mar. 2012.

[8] M. Higashida, *Autonomic Datacenter Operation under the concept of Cloud Network Robotics*, in IEICE Technical Report, CNR2013-10-16, Vol.113, No.248, pp.33-38, Oct. 2013 (In Japanese).

[9] S. Shimojo, M. Higashida, Y. Teranishi, *Future Network Required for Cloud Robotics*, The Journal of the Institute of Electronics, Information and Communication Engineers (J. IEICE), Vol. 95, No. 12, pp.1057-1061, Dec. 2012 (In Japanese).

[10] J. Li, X. Ji, X. Liu, Y. Jianguo, S. Gopalakrishnan, F. Hu, *Topic-Based Resource Allocation for Real-Time Publish/Subscribe Communication Systems*, 2010 5th International ICST Conference on Communications and Networking in China (CHINACOM), Beijing, pp.1-9, 25-27th Aug. 2010.

[11] M. A. Batalin, *Symbiosis: Cooperative Algorithms for Mobile Robots and a Sensor Network*, Ph.D. Dissertation Proposal, University of Southern California, Jul. 2004.

[12] Open Source Robotics Foundation, *Turtlebot™*, Available at: http://turtlebot.com/, Accessed Mar. 2014.

[13] iRobot Corporation, *iRobot Roomba Vacuum Cleaning Robot*, Available at: http://www.irobot.com, Accessed Mar. 2014.

[14] R. Rouvoy, P. Barone, Y. Ding, F. Eliassen, S.O. Hallsteinsen, J. Lorenzo, A. Mamelli, U. Scholz *MUSIC: Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments*, Software Engineering for Self-Adaptive Systems, LNICS, Vol. 5525. Springer-Verlag, pp.164-182, 2009.

[15] S. Kosta, A. Aucinas, P. Hui, R. Mortier, X. Zhang, *ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading*, in Proc. of IEEE INFOCOM 2012 , pp.945-953, Mar. 2012.

[16] H.R.F. Macario, S. Srirama, *Mobile code offloading: should it be a local decision or global inference?*, in Proc. of ACM MobiSys 2013 pp.539-540, June 2013.

[17] L. Jiao, R. Friedman, X. Fu, S. Secci, Z. Smoreda, H. Tschofenig, *Cloud-based Computation Offloading for Mobile Devices: State of the Art, Challenges and Opportunities*, in Proc. of Future Network and Mobile Summit 2013, pp.1-11, July 2013.

TRITA-ICT-EX-2015:8