Abstract

This thesis describes development of an android application to do health monitoring of the elderly person and remotely controls a remote robot “TurtleBot” in order to allow caregivers to interact with elderly person in more convenient way.

A hybrid android application has been developed. The main services of the application are divided into two: i) Health Monitoring System which provides caregivers with elderly person's physiological data collected by various sensors installed in or around the house, and ii) Robot Telepresence System which controls the robot by using android application and provides video chat with elderly person.

The application is connected with a database server and is able to receive patient's physiological data by using HTTP Request. In order to remotely control the "TurtleBot", ROS (Robot Operating System) has been used. The video chat service is implemented by using WebRTC technology, which enables peer to peer connection between the application and the TurtleBot laptop.

The developed application is tested by using various performance testing tools. The results show that the application has relatively high performance. In this thesis, we have reached our goal to develop an android application with essential services we have decided, but the application still needs some improvements to be used in the real world.

Contents

1 Introduction **1**

2 Background **3**
2.1 Elderly Care Robot 3
2.1.1 Health Monitoring System 4
2.1.2 Robot Telepresence System 5
2.2 GiraffPlus Project 6
2.2.1 Giraff Robot 7
2.2.2 DVPIS 8
2.2.3 Pilot 12
2.2.4 Alternative for Interacting with Giraff Robot 13
2.3 TurtleBot & ROS 14
2.4 Mobile Application 15
2.4.1 Native Application 16
2.4.2 Web Application 17
2.4.3 Hybrid Application 18
2.4.4 Differences between Mobile and Desktop Application 18

Development of Android Application

3 Design of Android Application **22**
3.1 Main Services 22
3.2 Sensor Network for Android Application 24
3.3 Basic Structure of Application 25
3.4 Overview of the Developed Android Application 27

4 Health Monitoring System **29**
4.1 Technologies 30
4.1.1 Web Application Server (WAS) 30
4.1.2 Apache Tomcat 31
4.1.3 Spring Framework 32
4.1.4 REST API 34
4.1.5 MongoDB 34
4.2 Development of Database Server 36
4.2.1 Creating Demo Database 37

4.2.2 Communication Between Server and Database 39
4.3 Development of Android Part 43
4.3.1 UI of Android Application 43
4.3.2 REST API in Server Part 44
4.3.3 REST API in Android Part 46
4.4 General Workflow of Monitoring Service 47
4.5 Implementation of Reminder Service 48
4.6 Hosting the Database Server 49

5 Robot Telepresence System 51
5.1 Technologies 51
5.1.1 WebRTC 51
5.2 Development of Web Application for Pilot System 54
5.2.1 UI of Web Application 54
5.2.2 ROS Communication of TurtleBot 55
5.2.3 Communication between Web Application and ROS 57
5.2.4 General Workflow of Pilot Web Application 61
5.2.5 Local Costmap 62
5.2.6 Autonomous Navigation of the Robot 62
5.2.7 Hosting the Rosbridge Server 63
5.3 Web Application for Video Chat System 65
5.3.1 Development of Video Chat Web Application 66
5.4 Converting Web Applications to Android Application 68
5.4.1 Testing in Real World 70

6 Performance Testing 71
6.1 Methodology 71
6.1.1 Performance Indicators 71
6.1.2 Performance Testing Tools 71
6.2 Testing on Monitoring System 72
6.2.1 Load Testing on Database Server 72
6.2.2 Monitoring system in Android Application 75
6.3 Testing on Telepresence System 77
6.3.1 Performance of Pilot Web Application 77
6.3.2 Performance of Video Chat Web Application 78
6.3.3 Telepresence System in Android Application 79
6.4 Battery Usage of the Application 80

Conclusions and Future Works

7 Conclusions 82

8 **Future Works** **83**
8.1 Usability Testing 83
8.2 Improvement of the Telepresence System 83
8.3 Android Application for Elderly User 84
8.3.1 Collecting Physiological Data by Using Android Device 85

References **87**

Appendix A: Android Part **93**
Appendix B: Server Part and Database Part **103**
Appendix C: Pilot System **114**

List of Figures

1.1	The basic concept of the application	2
2.1	Giraff robot and Giraff robot in a real setting	7
2.2	General Layout of DVPIS Alpha Version	9
2.3	Physiological Data Panel	10
2.4	Activity Panel	10
2.5	The Structure of DVPIS	11
2.6	Pilot Software	12
2.7	Turtlebot	14
3.1	The general structure of the application	26
3.2	Login page and Monitoring page of the HMT	27
3.3	Video chat, Pilot with camera view, and map view	28
4.1	General structure of the Health Monitoring System	29
4.1	Workflow of a Web Application Server	31
4.2	High-level architecture diagram for Spring MVC	33
4.3	Basic Structure of the Demo Database	37
4.4	Starting MongoDB Server at Port 27017	39
4.5	Basic UI of Android Application	43
4.6	General workflow of the Monitoring System	47
4.7	Navigation drawer, Set reminder window, Get reminder window	49
5.1	The WebRTC signaling system using STUN, TURN and signaling server	53
5.2	Main UI of Web Application	54
5.3	Gazebo World with TurtleBot robot	56
5.4	Rviz tool for visualizing map and robot model	56
5.5	The General Workflow of Pilot System	61
5.6	Web application with map + local costmap view	62
5.7	Autonomous navigation with goal marker	63
5.9	Starting Ngrok server	64
5.10	QuickBlox video chat sample	66

5.11 Developed video chat web application 67
5.12 Video chat, ROS control with camera view, and local cost map view 69
5.13 Interacting with Real TurtleBot 70
5.14 Video chat, Pilot with camera view, and map view in real world 70
6.1 Configuration of the load test 73
6.2 Summary of the samples 73
6.3 Detail information about each samples 74
6.4 Response time graph of load test 75
6.5 Performance for monitoring system analyzed by Android Device Monitor 75
6.6 Information of transferred data while starting Pilot Web Application 77
6.7 Transferred camera view data from TurtleBot 78
6.8 Transferred map data from TurtleBot 78
6.9 Information of transferred data while starting Video Chat Web Application 78
6.10 Information of transferred data while establishing p2p connection 79
6.11 Performance for Pilot System analyzed by Android Device Monitor 79
6.12 Performance for Video Chat System analyzed by Android Device Monitor 80
6.13 Battery usage for using the HMT application 81
6.14 Battery usage for using "Facebook" application 81
8.1 Percentage of U.S. adults ages 65 and older who own smart phone 84
8.2 Examples of health-measuring android application iCare Health Monitor Pro, Instant Heart Rate and Runatic Heart Rate 86

List of Examples

4.1 The example of the "measurement" data in the database 38
4.2 MongoDB configurations in MongoController class 39
4.3 Mongo query operation "findOne" 40
4.4 Mongo query operation "updateFirst" 40
4.5 The process of the Aggregations operations 42
4.6 The example of using "RestTemplate" class 46
4.7 Data model of "reminders" field in the database 48
5.1 Establishing connection with rosbridge server 57
5.2 Creating and publishing ROS topic object 58
5.3 Subscribing ROS topic object 59
5.4 Creating map viewer in the web application 60
5.5 Loading html file in the android application 69

1 Introduction

In the last few years, there are a growing number of robots being designed and built to interact with people in a specific setting. Above all, recent studies on interaction with robots focus on the importance of social intelligence even more so in a healthcare/eldercare sector. A number of researchers have been inspired by the expected growth in the elderly population and the labor shortages in the healthcare sector to explore the applicability of intelligent systems in general and robotic products in particular to be used in assisted-living environments. For robots, the functionalities are related to supporting independent living by supporting basic activities: eating, bathing, toileting, getting dressed and mobility, and providing household maintenance, monitoring of those who need continuous attention and maintaining safety [1].

Meanwhile, the mobile app business is growing rapidly, and many studies about apps development have been conducted. Now the mobile applications have become the one of thing we can't leave behind in our daily life. Mobile apps were originally offered for general productivity and information retrieval, including email, calendar, contacts, stock market and weather information [2]. However, public demand and the availability of developer tools drove rapid expansion into other categories, such as those handled by desktop application software packages [3].

In this thesis, these two main services are combined by making an android application to remotely control a remote “TurtleBot 2” robot [4] in order to allow caregivers to interact with their patient in a more convenient way than using laptop. The fundamental idea is from GiraffPlus project which allows the caregivers to remotely control a existing elderly care robot "Giraff" by using their laptop [5]. The more detailed description about the GiraffPlus project will be described in section 2.2.

In fact, almost all types and versions of smart phones have already included videoconference capabilities and technologies needed for controlling a robot, such as standard internet connection, Wi-Fi, Bluetooth and accelerometer etc. That helps us develop the application. Figure 1.1 shows us the basic concept of the application we are going to deal with in this

thesis.

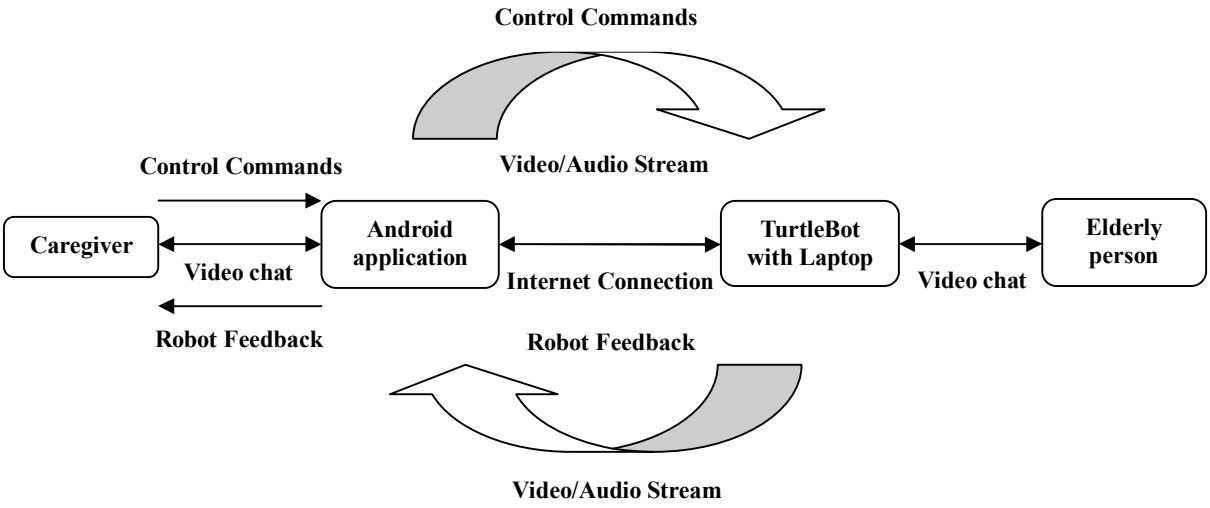


Figure 1.1 The basic concept of the application

This thesis describes the research about implementing two main services for assisting caregivers, e.g. Health Monitoring System and Robot Telepresence System. The systems are developed as combinations of several parts that work outside of the android application such as database server. The developed application will be a demo version aiming at investigating how to develop a complex system using different technologies and combine different parts into the android application in effective way.

The thesis consists of 8 chapters. The first chapter we introduce background of the thesis including the "GiraffPlus" project which is deeply related with our work. The second chapter describes about implementation of the project, from design of the application to actual development. In the last chapter the thesis describes the results of our work including performance testing of the application and our future work.

2 Background

2.1 Elderly Care Robot

In recent years, a number of robots have been developed for assisting elderly in their daily living. The idea behind elderly care robots has been around for years as the gap between the growing proportion of elderly people in the world and number of available caregivers. Furthermore, elderly people prefer more and more to live in their own homes as long as possible instead of being institutionalized in nursing homes [6].

Researchers all around the world are trying to solve this problem and are independently working to develop robots that have the potential to perform tasks as caregivers. Generally, studies on elderly care robot are related with two types of robots. First, there are robots that are used as assistive devices. Assistive robots aim at helping the elderly maintain independence by providing physical assistance and supporting basic activities, e.g. eating, bathing, toileting and getting dressed [6][7]. Such robots include Bestic feeding robot [8], and Sanyo electric bathtub robot that automatically washes and rinses [9]. In addition, certain robots provide monitoring of those who need continuous attention and maintaining safety, and mobility including navigation and mapping. For example nursebot Pearl developed by Carnegie Mellon University, is a mobile robot that can help elderly to navigate through the nursing facility [10]. It does have a user-friendly interface with a face, and can also provide advice and cognitive support for elderly. The Giraff mobile robot [11][18], which is deeply related with this thesis and will be described in more detailed later, is also a kind of assistive robot which provides to caregivers both monitoring and mobility for elderly health care.

Second, there are studies which focus on the pet-like companionship a robot might provide. The main function of these robots is to enhance health and psychological well-being of elderly users by providing companionship [12]. Such companion type robots include the seal shaped robot Paro [13] developed in Japan, and is produced by Intelligent System Co., Ltd.. It is developed to study the effects of Animal Assistive Therapy with companion robots [14]. The therapeutic robot “the Huggable” offers the perception of touch for physical ease of use and comfort on the part of the elderly and disabled [15]. A dog type robot "Aibo" by Sony is mobile and autonomous and is programmed to play and interact with humans [16]. It has

been used extensively in studies with elderly in order to try to assess the effects on the quality of life and symptoms of stress.

A recent trend has been to move away from costly, task-specific platforms supporting well-defined medical, commonly surgical tasks, towards cheaper more generic platforms. These robots can commonly support a number of different human-robot interaction (HRI) tasks and have been used to improve the medical conditions of patients. However, they have also been successfully applied to a range of tasks that do not address medical conditions directly. Instead, these latter applications have focused on generic health and quality of life issues, as well as issues related to social care [19].

In this thesis we consider mainly two systems related with recent studies of elderly care robot, e.g. health monitoring system and robot telepresence system.

2.1.1 Health Monitoring System

The health monitoring system is one of the fastest growing areas in eldery care field for supporting elderly people who are living alone. The researches mainly address three issues, e.g. i) early detection of possible deterioration of health, ii) providing adaptive support which can offer services to assist in coping with age-related impairments, and iii) ways of supporting preventive medicine [92].

To address these issues, researchers are developing sensor network infrastructure that allows long-term monitoring of physiological data e.g. blood pressure or temperature, and daily activities for detecting e.g. whether somebody occupies a chair, falls down or moves inside a room [92]. The system allows the elderly people to live longer in their home without compromising safety and ensuring the detection of health problems [17]. In addition, it reduces caregivers' necessary time and healthcare costs in general.

The sensor network is one of the core components for the proper monitoring service. Atallah, L., et al., classified sensors under three groups [93]:

- The Body Sensor Network (BSN): A network of wireless sensors worn by, or

implanted in a person, which collects biomedical, physiological and activity data [94].

- The Ambient Sensor Network (ASN): A network of wireless ambient sensors installed in or around the house, appliances and furniture, which are designed to collect information about the resident's activities and environment.
- The Personal Mobile Hub (PMH): This could be a mobile phone or a PDA (Personal Digital Assistant) and acts as the HUB of a Body Sensor Network. It can also serve as an alarm trigger with a emergency press button [95].

The data from sensors are usually collected by an abstract gateway and are sent to a server with a secured database through the internet, for offering data to caregivers.

2.1.2 Robot Telepresence System

Telepresence robot is an extension of Robotic Technology which allows a person to be virtually present and to interact with an environment from a remote place. The study of using robots to represent a person has started In 1980s, and in early 1990s similar kind of Telepresence robots were developed. Telepresence robot is becoming more and more common in the field of social interactions, where the primary aim of the system is to foster a social interaction between individuals [75][76].

In the last years, telepresence robots have researched in special direction of applicability, healthcare and social interaction for the elderly. Robotic telepresence is a promising tool that strives for motivating social communication with relatives and friends, helping in mitigating the loneliness of the elderly population. Telepresence robots can be a suitable and an affordable mean for caregivers to monitor elderly patients at home, and to be used as a moving phone to communicate with the elder patients. Telepresence robots, such as Giraff, Telerobot, TeCaRob, TRIC (Telepresence Robot for Interpersonal Communication), and Care-O-bot, were designed for home care assistance so that healthcare professionals, caregivers, and family members could check seniors and people with disabilities when necessary [23][76][79].

The current robotic telepresence systems are characterized by a video conferencing system mounted on a mobile robotic base. Telepresence robots provide interactive two-way audio

and video communication. Additionally, these telepresence robots can be controlled independently by an operator, which means that the person driving can explore and look around as he/she desires [77].

The robotic platform is accessed and controlled via a standard desktop or laptop using a software application. From a remote location a caregiver with limited prior computer training teleoperates the robotic platform while elderly person living in their own home, where the robot is placed, can receive their visit through the robot. The remote user can charge the robot batteries by driving it onto a docking station.

2.2 GiraffPlus Project

GiraffPlus is an FP7-funded project, which was coordinated by Örebro University, and included 12 collaboration partners in six European countries. The project aims at developing a complex system which collects elderly people's physiological parameters or daily behavior using various kinds of sensors, both in and around the home as well as on the body. The sensors can measure e.g. blood pressure and body temperature or detect e.g. whether somebody falls down [17][20].

At the heart of the system is a unique telepresence robot, Giraff, which lends its name to the project. With its help, the system provides e.g. caregivers and relatives to virtually visit an elderly person in the home. The caregivers can discuss and plan care measures based on the data that has been collected by the system. The project also provides services such as alarms via the Giraff robot which allow caregivers and relatives to enable timely involvement. The Giraff robot will be described more deeply in the next section [20][21].

The GiraffPlus system is installed and evaluated in a demo apartment in Örebro, and at least 15 homes of elderly people distributed in Sweden, Italy and Spain. These evaluations will drive the development of the system [21].

2.2.1 Giraff Robot

The Giraff robot is a remotely controlled mobile, human-height robot integrated with a videoconferencing system, including a camera, display, speaker and microphone. The Giraff technology was developed by "Giraff" company in Sweden 2010 with CEO Stephen Von Rump. The fundamental idea was to keep elderly connected to their world and helps provide the confidence that allows them to continue living at home safely, independently and with a good quality of life. Most of these contacts came from countries in Europe with organized social care systems and a strong national sense of responsibility for elderly care [21][22].

Figure 2.1 Giraff robot (left) and Giraff robot in a real setting (right)

The Giraff robot is built upon a motorized wheeled platform endowed with a videoconferencing set, including camera, microphone, speaker and screen. The robot is placed in the home of patients and will be used to connect to caregivers as well as family members. The robot is powered by motors that can move and turn in any direction, even backwards and is accessed over a standard internet connection via desktop or laptop with main software DVPIS and Pilot, which will be described more deeply in the next section. The robot allows caregivers to visit their patients at home via a telepresence system using a Skype-like interface, which can look around via a pan/tilt/zoom camera and real time videoconference service via a life-size portrait image from their webcam. It enables

caregivers to communicate and move around in the home environment at the same time. It can help reducing elderly people's feeling of isolation and loneliness as well [17][23][24]. The GiraffPlus project provides caregivers manly two softwares, DVPIS and Pilot, which provides the caregivers with health monitoring of elderly people and remotely control the robot respectely. Since these softwares will be a basis of our android application they will be described in next two sections.

2.2.2 DVPIS

The overall software service that caregivers use in their desktop/laptop is called DVPIS, which means Data Visualization, Personalization and Interaction Service. The DVPIS is basically expected two main services [25]:

The Interaction and Visualization Service (IVS) is dedicated to front-end service, to produce a well-organized set of functionalities for allowing different users to connect with the GiraffPlus environment. Relevant components of IVS include videoconferencing system, robot controlling system and visualizing personal data gathered from sensor network in a home environment. The personal data is divided into biomedical parameters and physical activities. While personal biomedical parameters show blood pressure, blood glucose, weight, oxygen saturation and temperature, physical activities show the residence's activity during the day and night, and the collected date are plotted over time to give graphical views to how change happens over time [24].

The Personalization Service (PerS) is more dedicated to back-end service, to provide fine-grained personalization to the users. The personalization is generally a continuous loop around two main data structure, a User Knowledge Base and a Person Dynamic Model. While the User Knowledge Base contains data about user stereotypes of different classes of potential users, the Person Dynamic Model contains specific user models represented based on timelines. A user model is a set of relevant personal features that may influence human behaviors and can be used to perform personalization. For instance, variables like personality traits, perceived stress and anxiety are part of the user model. User models should be updated and maintained over time to enable more complex personalized services and to provide increasingly accurate model of the target [25].

In addition, reminding service and reporting service are also a part of PerS. The main task of the reminding service is delivering messages to both elderly and caregivers in order to remind them of important tasks, for example, taking medicines or doing some physical activity for rehabilitation. The reporting service is to analyze long term data in order to observe different users' trends. It is also used to detect emergence situations from deviations from elderly person's common behaviors.

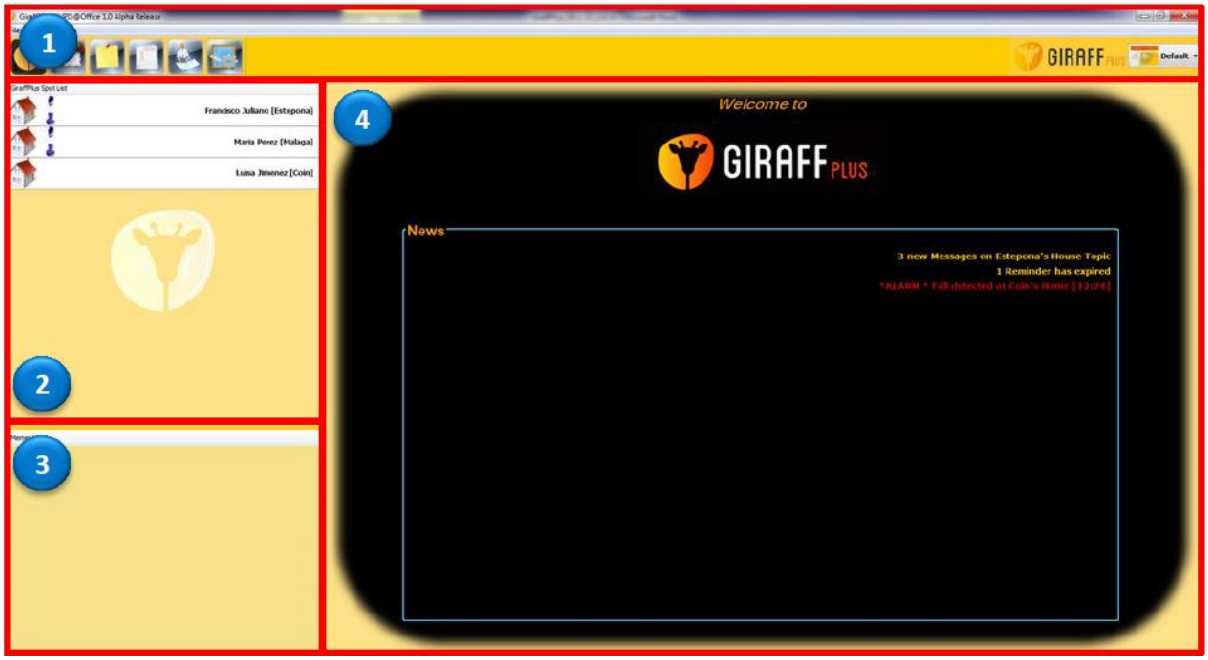


Figure 2.2 General Layout of DVPIS Alpha Version

The UI of the DVPIS software is composed of four basic main elements numbered in the figure above [25]:

1. Menu bar with general commands from the left side - Home panel, Private Messages Area, Reminder Service Control Panel, Report Generation Service Control Panel, Application Settings.
2. List of Elderly people currently under authorized access by the logged user. By clicking on an item of the list the Main control Area will be updated and populated with the related data.
3. Container that lists all incoming real time messages.
4. Main Control Area that displays all detailed information. The sub-components of this

area will be filled from data of the chosen elderly person. (See figure 2.3 and 2.4)

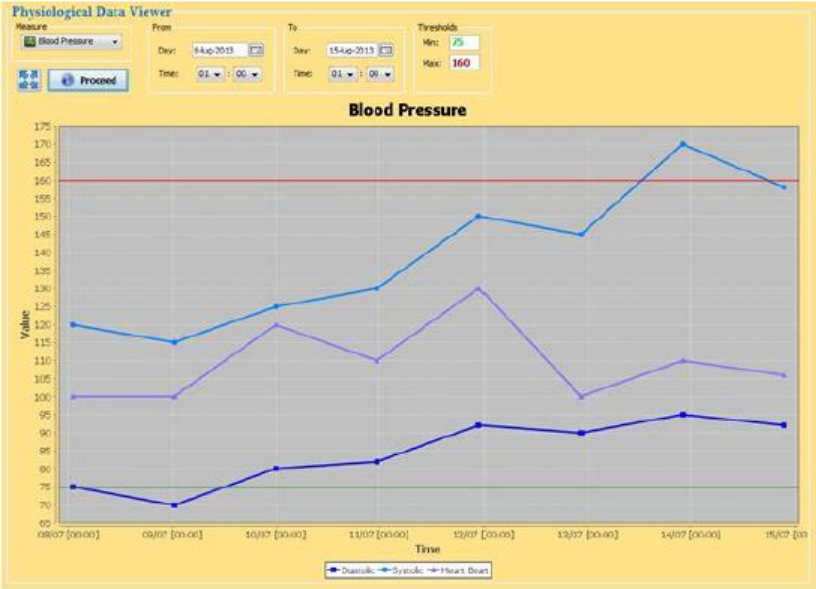


Figure 2.3 Physiological Data Panel

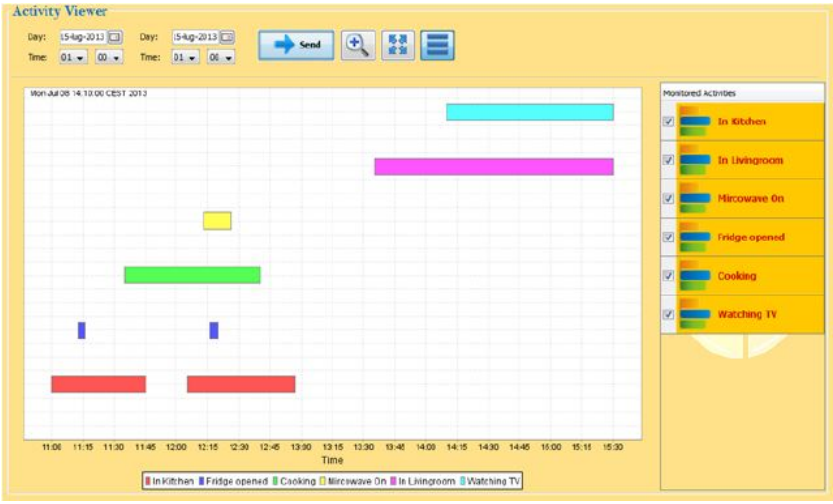


Figure 2.4 Activity Panel

The software in the GiraffPlus basically based on Java, which implies great flexibility regarding what platform it can run. At the core of the GiraffPlus cloud infrastructure, Spring framework is used [26]. The Spring framework is an open source application framework of the enterprise Java platform, which will be responsible to provide fast and secure access to the central data storage. It provides complete support for modern web technologies including REST, HTML 5 and AJAX, and also provides web support for the most popular mobile client platforms, such as Android and iPhone [27]. In order to store large amount of gathered data the GiraffPlus system uses MongoDB database technology, which can handle large volumes

of data, and has been proved its security and reliability [28]. On the server side of GiraffPlus system, the RESTful Web services is exposed to provide the ability for the clients to securely connect to the GiraffPlus infrastructure and gain access to data stored in database service as well as to send commands back to the infrastructure.

Regarding storing and using data, system named “Sentry” is used as a part of the DVPIS. The Sentry is a kind of management system that is used to manage robot configuration, users, groups, visit records, etc. It is accessed through a standard browser with a user ID and password, but only caregivers authorized by the resident can connect to the Sentry [17].

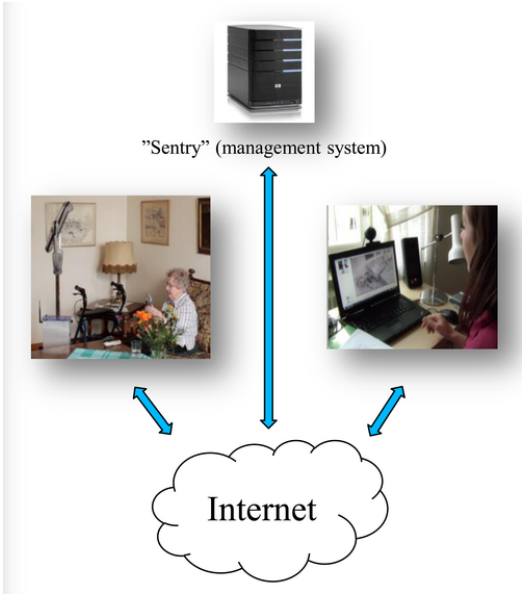


Figure 2.5 The Structure of DVPIS

In this thesis, we develop an android application, which is based on the GiraffPlus project, and it means that we consider how we can implement services of the DVPIS module into the android application and replace "Pilot" and "Sentry" system with other technologies that we can use in our circumstance.

2.2.3 Pilot

In order to remotely control the Giraff robot and control it, software named “Pilot” is used as a part of the DVPIS. Pilot is the client service that allows connecting to the Giraff robot and controlling the robot remotely. The Pilot application is integrated by using Java .jar plug-in that contains Java classes, resources and metadata describing the dependencies with other bundles. The Pilot is running in parallel with the DVPIS.

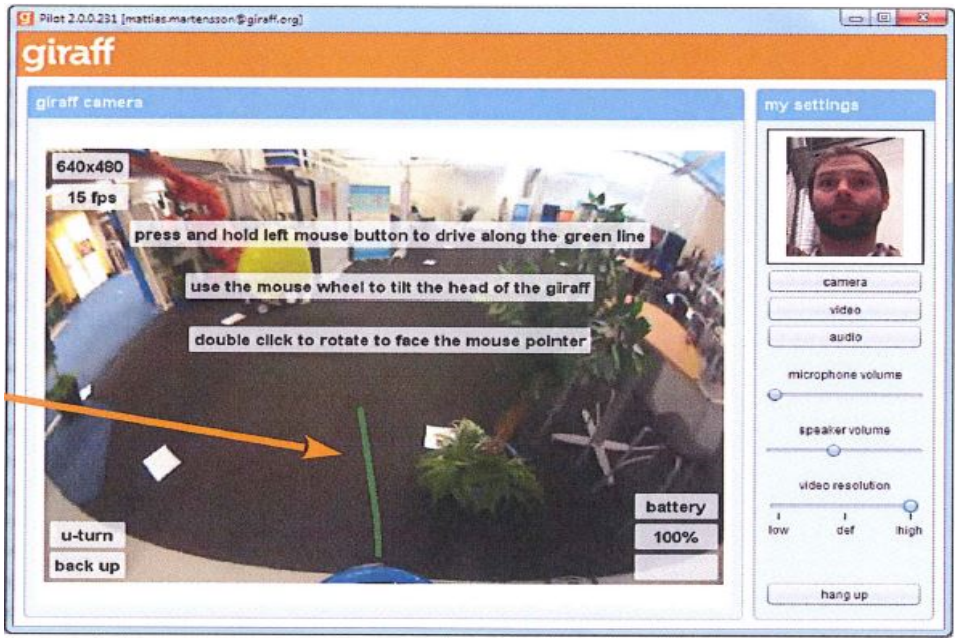


Figure 2.6 Pilot Software

In the Pilot software, the Giraff robot is controlled by clicking mouse on the video frame coming from the robot camera. When the user hover the mouse pointer over the video frame, a dark green line is drawn from the base of the robot to the mouse pointer. It is the default state that shows an approximate trajectory of the robot. When users press and hold the left mouse button, the line turns bright green and the robot starts driving. The robot's speed and direction are determined by the length and orientation of the line. When the user release the mouse button, the robot stops and the line turns back to dark green again. There are also "u-turn" button which makes the robot turn 180 degrees and "back up" button which makes the robot move slightly backwards on the lower left corner of the video frame. To make the robot turn toward an object or person while the robot is stopped, double click on the video frame and the robot will turn toward it. The user can also look up and down using the scroll wheel on the mouse [29].

2.2.4 Alternative for Interacting with Giraff Robot

The GiraffPlus project has already considered about using remote device such as, the Nintendo Wii remote controller, the Microsoft Kinect and smart phones equipped with accelerometers, with the purpose of enhancing the interaction capabilities of the Giraff robot to allow an increased engagement for remote operators during the interactions through the robot.

The use of computers and mouse devices for controlling the robot may result in a sort of "barrier" in having a thorough immersion in the remote environment. Therefore, the use of alternative tangible interfaces to control the robotic platform has been investigated.

Using smart phones with accelerometers introduced with both positive and negative aspects. Potentially, such devices allow to implement tangible interfaces that may allow the user to control the robot performing some gestures instead of using a mouse and a keyboards, and, then, having a more natural and immersive interaction. Moreover, holding the smart phone with the both hands and moving it in several directions can constitute a sort of virtual window that provides an access to the remote environment. In addition, smart phones are endowed with videoconference capabilities and, thus, they allow having all the functionalities required to perform a call through the Giraff robot in a pretty small object. But as such devices rely on mobile Internet connection, the available bandwidth may represent a critical issue. In fact, controlling the Giraff requires a real time control of the robot as well as a high quality video call service [24].

2.3 TurtleBot & ROS

TurtleBot is a low-cost, personal robot kit with open-source software. TurtleBot was created in November 2010, at Willow Garage, a robotics research institute, by Melonee Wise and Tully Foote. The TurtleBot is designed for being easy to buy, build, and assemble, using off the shelf consumer products and parts that easily can be created from standard materials. By default, the TurtleBot consists of a mobile base, a 3D Depth Camera(Kinect), a laptop computer, and the TurtleBot mounting hardware kit [4].

Figure 2.7 Turtlebot [78]

TurtleBot is built in collaboration with the original makers of ROS(Robot Operating System). The ROS is a collection of open source software libraries and tools that help us build robot applications. ROS was originally developed by the Stanford Artificial Intelligence Laboratory, and major contributions were provided by Willow Garage in 2007. ROS is a flexible framework for writing robot software with intent of simplifying the task of creating complex and robust robot behaviour across many different robotic platforms. ROS runs on Linux, is basically written in C++ and Python, and provides operating system-like functionality. The TurtleBot is powered by ROS software and handle vision, localization, communication and mobility. The ROS Wiki homepage provides many tutorials and open source codes for TurtleBot, including autonomous control and avoiding obstacles along the way [80][81].

A communication system of ROS is message passing system between nodes. ROS's built-in and well-tested messaging system saves us time by managing the details of communication between distributed nodes via the anonymous publish/subscribe mechanism. Nodes

communicate with each other by publishing messages to topics. A message is a simple data structure, comprising typed fields. By a set of standard message formats, there are message definitions for geometric concepts like poses, transforms, and vectors; for sensors like cameras, IMUs and lasers; and for navigation data like odometry, paths, and maps; among many others. The more detailed about ROS communication used in this thesis will be described in section... [81][82]

2.4 Mobile Application

Before we research about mobile application, we have to define the term "application". Thus, this section describes what "application" is and its varied types, and both positive and negative aspects of using mobile application.

The definition of "application" is defined as 'An application program is a program designed to perform a specific function directly for the user or, in some cases, for another application program' [30]. Examples of application include word processors, database programs, Web browsers, development tools, drawing, paint, image editing programs, and communication programs. Applications use the services of the computer's operating system and other supporting applications. The formal requests and means of communicating with other programs that an application program uses is called the application program interface (API).

The concept of the application is so wide and there are many different ways to classify different types of application software. But in this thesis, the two types of applications will be considered, "mobile application" and "web application".

A mobile application, most commonly referred to as an app, is a type of application software designed to run on a mobile device, such as a smart phone or tablet computer [31]. Apps are generally small, individual software units with limited function. Apps are typically written in Java for Android devices or Objective C or Swift for iOS devices. As researched by IDC Quarterly Mobile Phone Tracker, in the first quarter of 2017, most mobile devices use one of the two dominant operating systems: Google-developed Android (85.0%) and the Apple-developed iOS (14.7%) [32].

In this thesis, the application will be developed for Android platform. Android is a comprehensive software stack of mobile devices that includes an operating system, middleware and key application. This rich source of software bunch is used in Mobile Technology through its innovation module of The Android Software Development Kit (SDK) [34]. Java Programming Language is used as a basic building block and back bone for Android Application Development that allows developers to program comprehensive application on Java that runs on Android Mobile.

In regard to integrated development environment(IDE) for Android platform development, there are two popular choices, Android Studio and Eclipse Android Development Tools(ADT). Until the Android Studio was announced on 2013 by Google, the Eclipse ADT was actually the only solution to develop Android application, but it was replaced by Android Studio.

Mobile apps are sometimes categorized according to whether they are web-based or native apps, which are created specifically for a given platform. A third category, hybrid apps, combines elements of both native and Web apps [34][35].

2.4.1 Native Application

Native apps are what we would normally think about when we think about apps. They live on the device and are accessed through icons on the device home screen. Native apps are installed through an application store, such as Google Play or Apple's App Store. They are developed specifically for one platform, and can take full advantage of all the device features, such as the camera, the microphone, the GPS, the accelerometer, the compass, the list of contacts, and so on. And native apps can use the device's notification system and can work offline [35][36].

What distinguishes native apps from the alternatives mentioned is that they are designed and coded for a specific kind of device. For instance, Android apps are written in Java, and iPhone apps are written in Objective-C , etc. So they are faster and more refined than their hybrid counterparts, making them a perfect match for high-performance applications. Each

mobile platform offers developers their own development tools, interface elements and standardised SDK. This should enable any professional developer to develop a native app relatively easily [37].

2.4.2 Web Application

A Web application or Web app is generally a client–server software application program that is stored on a remote server and the client or user interface runs in a web browser. Common web applications include webmail, online retail sales, online auctions, wikis, instant messaging services and many other functions.

Actually, the general distinction between a dynamic web page of any kind and a "web application" is unclear. Web sites are most likely referred to as "web applications" which have similar functionalities to other kinds of applications that allow the user to perform actions. Someone says, a website can be defined by its content, while a web application can be defined by its interaction with the user. It means, a website can plausibly consist of a static content repository that's dealt out to all visitors, while a web application depends on interaction and requires programmatic user input and data processing [38].

Most Web applications are based on the client-server architecture where the website visitors submits and retrieves data to/from a database over the Internet using their preferred web browser. The data is then presented to the user within their browser as information is generated dynamically in a specific format, e.g. in HTML using CSS by the web application through a web server. Therefore web applications commonly use a combination of server-side script, using ASP, PHP, etc, and client-side script, using HTML, JavaScript, etc, to develop the application. The client-side script deals with the presentation of the information while the server-side script deals with all the hard stuff like storing and retrieving the information [39].

One of the advantages of using web application is that web application can be used in smart phone as well. There are numerous applications which are developed for being used in smart phone. They look and feel like native applications, but they are actually mobile-optimized web pages that look like apps. Users first access them as they would access any web page, and they navigate to a special URL and then have the option of “installing” them on their

home screen by creating a bookmark to that page. However, there are native features that remain inaccessible in the browser, f.eks the notifications, running in the background, accelerometer information, complex gestures. Developing a web app can be simple and quick, however, their simplicity is also their downside [40].

2.4.3 Hybrid Application

Hybrid apps are part native apps, part web apps. Hybrid apps embed a mobile web site inside a native app, possibly using a hybrid framework like Apache Cordova and Ionic or Appcelerator Titanium. With these tools we create HTML, CSS and JavaScript local files, design and build the app as if it was a website, then use Cordova to wrap them into a mobile app. This allows development using web technologies while also have access to the native device APIs and hardware, e.g. direct access to device hardware, offline operation, push notifications [35][40].

Often, companies build hybrid apps as wrappers for an existing web page. In that way, they hope to get a presence in the app store, without spending significant effort for developing a different app. Hybrid apps are also popular because they allow cross platform development and thus significantly reduce development costs, because they can use the same HTML code components on different mobile operating systems. But the app itself will not be as fast as a native app as it still depends on the browser speed, so running high performance apps and games can be a frustrating experience [35][36].

2.4.4 Differences between Mobile and Desktop Application

As described above mobile applications and desktop (web) applications have different features, and before we start to develop mobile application we need to consider those differences as in both positive and negative aspects to use mobile application.

- **Wireless network (4G)**

The biggest advantage of using mobile application is mobile phone's handiness. The most of mobile phones has capability to use wireless network whenever and wherever where there is internet connection even without Wi-Fi connection. Moreover the mobile phones are small

and can easily be put in our pocket. The only thing we need to do for using the application is take out the mobile phone from the pocket and turn on the phone.

On the other hand, the limitation of laptop with its time to put out from bag, start up and connect to Wi-Fi for using the software may result as a sort of "barrier" in having a thorough immersion in the remote environment. The most fundamental services of DVPIS such as monitoring physiological data and daily behavior or video chatting have great potential to be used in remote environment that allows caregivers to care elderly people more comfortably and with fewer burdens.

The wireless network allows caregivers to use application in situations like:

- Sitting on the couch at home
- Walking around, inside or outside
- Queuing for something
- Waiting for a bus, train, or plane, or travelling

On the other side, wireless networks tend to be unstable or even not available when the user is travelling. Therefore, when we are designing for the mobile application, we may need to take the network connectivity into consideration. We can develop the mobile application with offline capabilities so that it can run against the local database if no network connection and then synchronize data with the server-side consolidated database when network is available. What's more, since most mobile plans aggressively limit the amount of data that can be downloaded each month, we have to make sure that the application doesn't contain large graphics or embedded videos that take up a large amount of data [44].

- **Touch input**

One important aspect of many mobile devices is touch-based input, while desktop computers mainly use a mouse for input. Designing a user interface for interaction with the finger is very different than designing for mouse, such as [41][43]:

- Less precision:

  Tapping the small icon or text link with our big fingers is not an easy task. A mouse can

actually select a single pixel, although one-pixel buttons are not recommended. Apple recommends a minimum button size of 44px x 44px. Users can compensate for this by zooming in, but it's still awkward. It's best to replace those fiddly text links with nice, large, touch-friendly buttons and other controls.

- No hover events:

  When we design web application or web site with JavaScript and CSS, we usually use many hover events for tool tips and other roll-over effects. But the mobile device has no mouse point, so there is no concept of "hovering" over a element. Moreover, there is no right mouse click popup menu as well. Some mobile browsers use various tricks to compensate for this, such as firing a hover event when the user taps the element once, and a click event if they tap again. However, it's best not to rely on hover events for our mobile site's functionality.

- Gestures:

  Most modern touch devices allow the user to perform gestures using one or more fingers, such as swiping, pinching, stretching, and so on. Many JavaScript frameworks can generate events for various touch gestures, making it easier to add gesture support to web applications.

- Direct touch

  Instead of reaching for the mouse when we want to click the button, we use finger for the button directly. That might sound small, but it makes using device much easier for our brain. In addition, our fingers have nerves in them, which tell our brain when they touch something. When our finger is touching the screen, we don’t need visual feedback to confirm it [42].

- **Smaller screens**

One of the most obvious differences between mobile device and desktop is that mobile devices have smaller screens than desktop. A smaller display means that the user can see a lot less information at once, and the display also pushes more of the page content "below the fold". This means that only the primary functionalities can be displayed at the screen at a time, the secondary functionalities should be hidden [43][44].

So, for design mobile application, it needs to i) present important information near the top of the page, ii) use an easy-to-read font, and iii) not overwhelm the user with too much content on the page. In addition, the layout needs to be usable and it needs to look good in a small window too.

- **Portrait screens**

While most desktop displays have a landscape (horizontal) orientation, the mobile device can change the device orientation and most users hold their device in a portrait(vertical) orientation. People usually expect to use their mobile application in any orientation, therefore, the mobile application have to be well-suited to both orientation. For design an application with portrait orientation, we have to consider [43][44]:

- Fewer columns of content (a single column is ideal)
- No overly-wide elements: This includes large multi-column tables, as well as extra-wide images, slideshows, Flash movies, and iframes
- Navigation along the top rather than down the side

- **Performance**

Although they are catching up, mobile devices generally have much less performance than desktop computers. This is for various reasons, including less processing power, insufficient of multitasking and constrained memory.

Therefore, it is very important to build the application as memory efficient as possible. For example, the mobile device cannot have multiple windows open. The more windows open the more memory is consumed. If necessary, use the API to check the memory in use, and prompt or close some windows to release memory. Minimize use of global things, for example global functions, global objects and global structures is also important, except when truly necessary. Global things are always resident in memory so it consumes memory [44].

Development of Android Application

3 Design for Android Application

Now we are going to start making design for android application we are going to develop. The design includes: i) which functions or services the application should have, ii) basic structure of the application and iii) general workflow of the application.

3.1 Main Services

At first, we start with choosing what functions or services the application should have. When we consider about the main user of the application, it will be basically caregivers and relatives of elderly people. The main services of the application are to allow caregivers to monitor elderly person's physiological measurements, and interact with the elderly person via Internet. The services will be based on DVPIS and Pilot software that are described in chapter 2.2.2 and 2.2.3, and three essential services are chosen such as:

i. Access to the central database and visualize elderly person's physiological parameters.
ii. Remotely control the remote robot and navigate the robot via application
iii. Initiate call, perform face-to-face video chat

i) Monitoring of elderly person's physiological measurements. The measurements are collected by various kinds of sensors placed in elderly person's home environment and stored in central database. The application sends request for specific data the user wants to the database via main server and the retrieved data is visualized as a graph in the android application.

In this thesis, we don't use real sensors, but we research how the GiraffPlus uses sensors and makes sensor network in the project, and we assume that the developed application use same sensor network as the GiraffPlus project.

Technologies used in the Health Monitoring System are almost same as the technologies used in the GiraffPlus project. The main server is developed as a web application server by using

Spring Framework [45] and Apache Tomcat [47]. MongoDB [28] is used in order to make a demo database. More detailed information about the technologies used in the Health Monitoring System will be described in chapter 4.1.

ii) Remote robot control service, including mapping and navigation. Since we use a remote robot "TurtleBot" in this thesis, the service is using ROS technology [50] in order to remotely control the robot. The "TurtleBot" using ROS technology has researched by many developers and there are a lot of open source codes we can use for our developing. In the rest of the thesis, this service will be called Pilot System.

While the Pilot software of the GiraffPlus project used mouse click to navigate robot, our application will use virtual controller with arrows to navigate the robot, because of less precision of mobile device's touch based input.

The application displays basically camera view like the Pilot software of the GiraffPlus project does, but as an optional function, a map view will also be implemented with a auto navigation function, which can navigate the robot by clicking or touching a point of the map.

Since our goal is to make an application, which can remotely control the robot via standard Internet, we are going to consider how we can communicate with TurtleBot laptop to control the robot without Wi-Fi connection.

iii) In order to implement the video chat system we use WebRTC (real time connection) technology [47], which makes peer to peer connection between two clients. The client program is developed as a web application, and is integrated into the android application by using Apache Cordova software [48]. So, the android application will be used by caregivers, while the web application will be used by elderly people. When the caregiver initiates call by using android application, the video chat system using WebRTC makes a peer to peer connection with elderly person's web application and exchange audio and video stream [49].

The video chat system relies heavily on the device’s network performance. The mobile devices have much less performance in both network and processing aspects, so quality of the

video chat may be unstable when the mobile device is moving around. The ideal condition is that the mobile device has Wi-Fi connection while using video chat system, but the application should provide stable video chat service without Wi-Fi connection as well.

3.2 Sensor Network of GiraffPlus project

The application in this thesis is developed under the assumption that using an existing sensor network developed as a part of GiraffPlus project.

The GiraffPlus system uses a number of sensors placed in patients' house and collects both physiological and environmental data. The collected data are intelligently visualized to for example diagram and graphs to provide useful indications to caregivers. Caregivers can see those indications through their laptop whenever they want and it gives caregivers ability to care their patients more convenient. The system can also rise alarms and send warnings for instance in case of falls or in case of abnormal physiological parameters [71].

The monitoring of physiological measurement are using sensors from IntelliCare Company. In particular Intellicare has developed the Look4MyHealth kit. Look4MyHealth is a system for remote monitoring of several biomedical parameters for example blood pressure, blood glucose, weight, oxygen saturation and temperature, and well-being monitoring for example daily routine, aid requests, and fall detection, in the home environment [72].

On the other hand, the GiraffPlus system is using sensors from Tunstall Company in order to collect environment measurement. Tunstall has also developed AD-life - activities of daily life monitoring solution. As the name represent, AD-life provides caregivers context recognition function, which monitors the residence's activity during the day and night, and the collected date are plotted over time to give graphical views to how change happens over time. The sensors used by AD-life includes door opening and closing sensors, which shows how many times are the doors used, bed/chair sensors for monitoring resident's location and activities, and electric usage sensors, which alert if an electrical device as a toaster is forgotten. There are also environmental sensors for fire detection, fall sensors and gas detection sensors [71][72].

The sensors of Giraffplus system are using different wireless communication technologies as Bluetooth, wi-fi and ZigBee to collect data. The collected data sent by those sensors are acknowledged by the gateway that also supports Bluetooth, wi-fi and ZigBee. The Gateway can use GPRS, 3G and Ethernet to send further the data. It works by connecting an Ethernet wire to the gateway or simply by configuring the Wi-Fi option in the gateway. The data sent from gateway are then divided into two types, long-term data and real-time data. Some real-time data are sent directly through the middleware for alarms and long-term data are sent to be stored in the server [71][72].

3.3 Basic Structure of Android Application

The android application has a basic structure, which consists of 5 components, Android part, Server part, Database part, Video Chat part and Robot Controlling part. The reason we divide the system into these parts is that each part is developed in different environment and use different technologies, for example the server part is developed by using Java and Spring Framework, while the Video Chat part is developed as a hybrid application by using HTML and JavaScript.

The general structure of the application is shown in figure 3.1. Server part and database part are connected to each other and work together as a database server in order to transfer the personal data stored in the database. The main purpose of the server part is transferring of data between android part and database part. In addition, since the application requires user id and password to login, the security system is implemented for protecting user's privacy.

Video chat part and robot controlling part are developed as a web application first, and then integrated in android part by using Cordova technology. Video Chat part allows user to connect with a chosen elderly person and enables video chat. Robot controlling part makes connection with TurtleBot via workstation and use ROS technology to interact and control the robot.

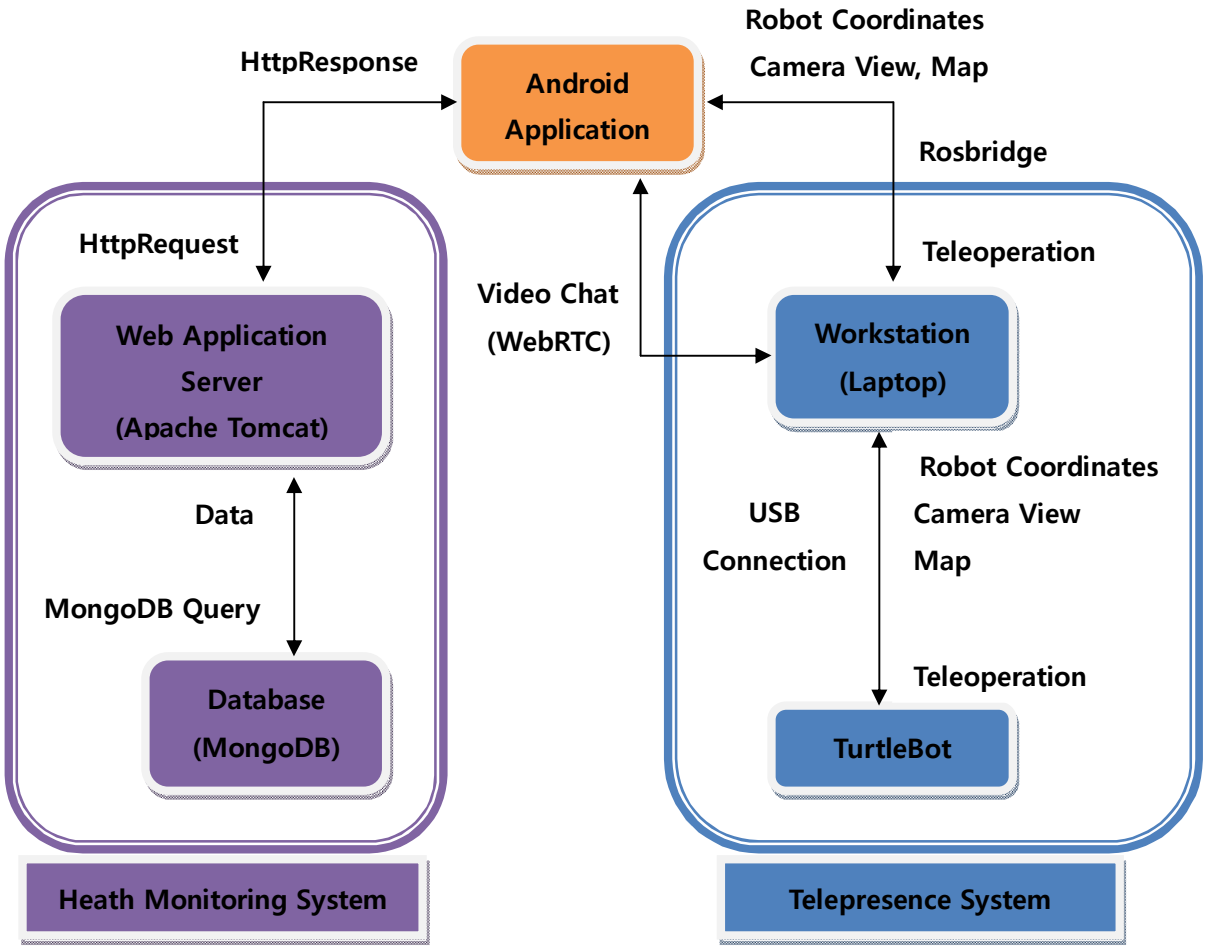


Figure 3.1 The general structure of the application

3.4 Overview of the Developed Android Application

This section introduces a simple overview of the developed android application before describing deeply about development of the application

+

Since the android application is designed to have mainly two systems: Health Monitoring System and Telepresence System. The name of the application is decided as, “HMT”, an acronym of the two systems. The figure 3.2 shows the log in page and main page of the HMT.

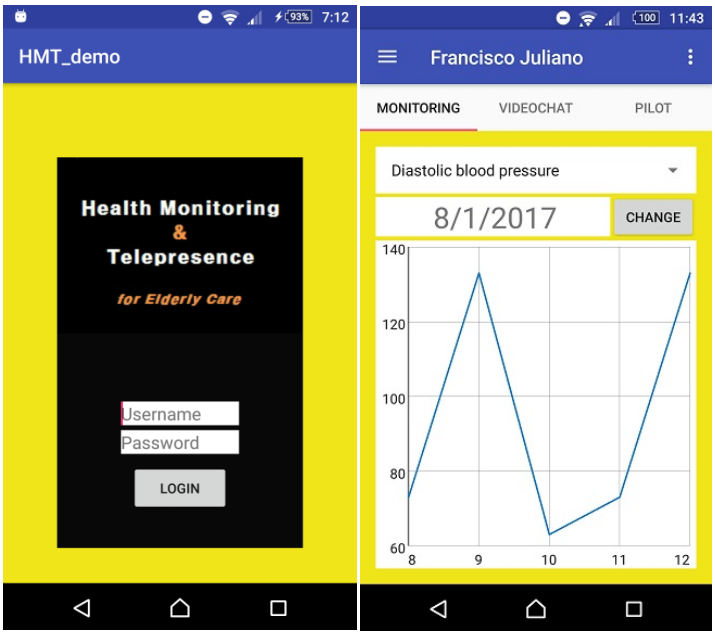


Figure 3.2 Login page (left) and Monitoring page (right) of the HMT

The application starts with login page (or activity in the android application) that requires valid id and password for using other functions. If the user logs in to the application the user can receive the personal data of elderly people who the user has responsibility to take care of.

In the main page of the application, a navigation bar is implemented with three elements: Monitoring, VideoChat and Pilot. Each element is connected to different services we have developed.

In the monitoring page the user can decide measurement type and period for requesting specific measurement data to the server, and received data is visualized in the application as a

graph. As we mentioned in chapter 2.3.5, the mobile phones has much smaller screen than laptop, so the graph should have a size that fits with the mobile screen.

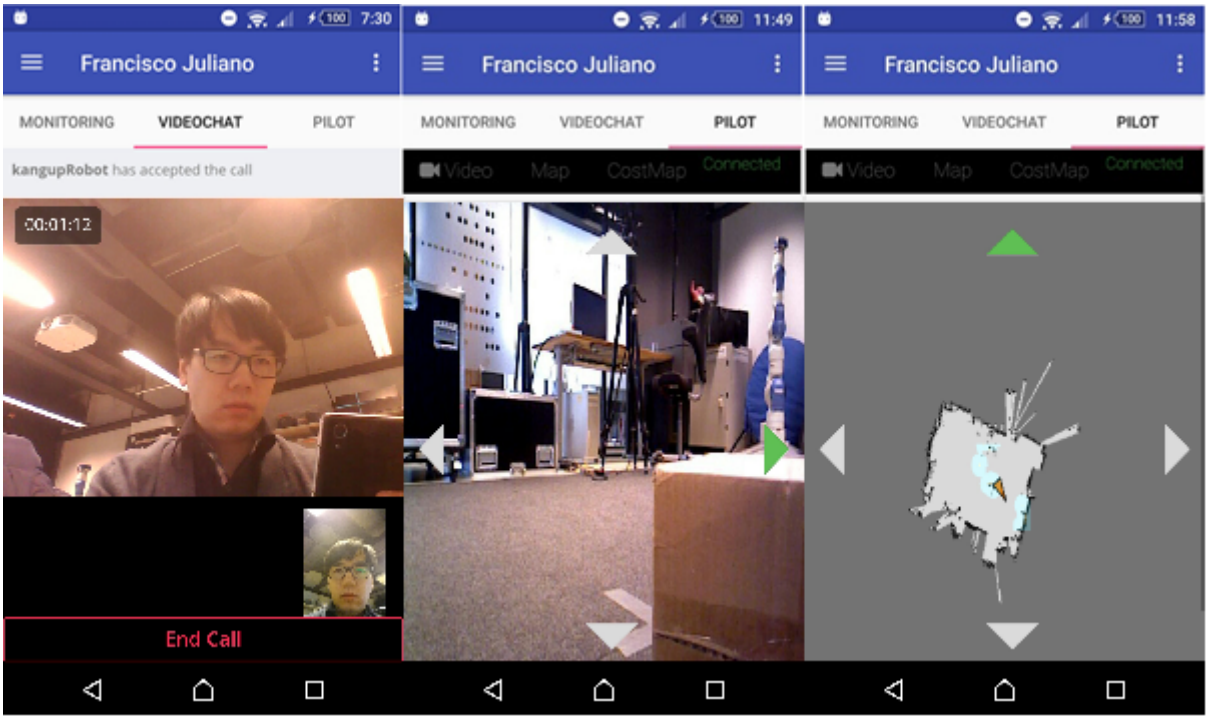


Figure 3.3 Video chat (left), Pilot with camera view (middle), and map view (right)

In the video chat page the user can initiate call to chosen elderly person and if the elderly person receives the call by using video chat web application, then video chat between two clients starts by using WebRTC technology with Skype-like interface.

Pilot system can be started from "Pilot" tap in the navigation bar. When the service starts, the application try to connect with TurtleBot using ROS. When the connection established, the application screen displays a front camera view of the TurtleBot. The display can be switched between the camera view and a map view. The robot can be controlled by using a virtual controller in the screen, and if the user touches a point on the map, the robot is navigated autonomously to the point.

4 Health Monitoring System

The health monitoring service is one of the core services of the application. The fundamental idea comes from the GiraffPlus project, as mentioned above. The system is designed as combination of 3 parts: server part, database part and android part. The figure shows the general structure of the Health Monitoring System.

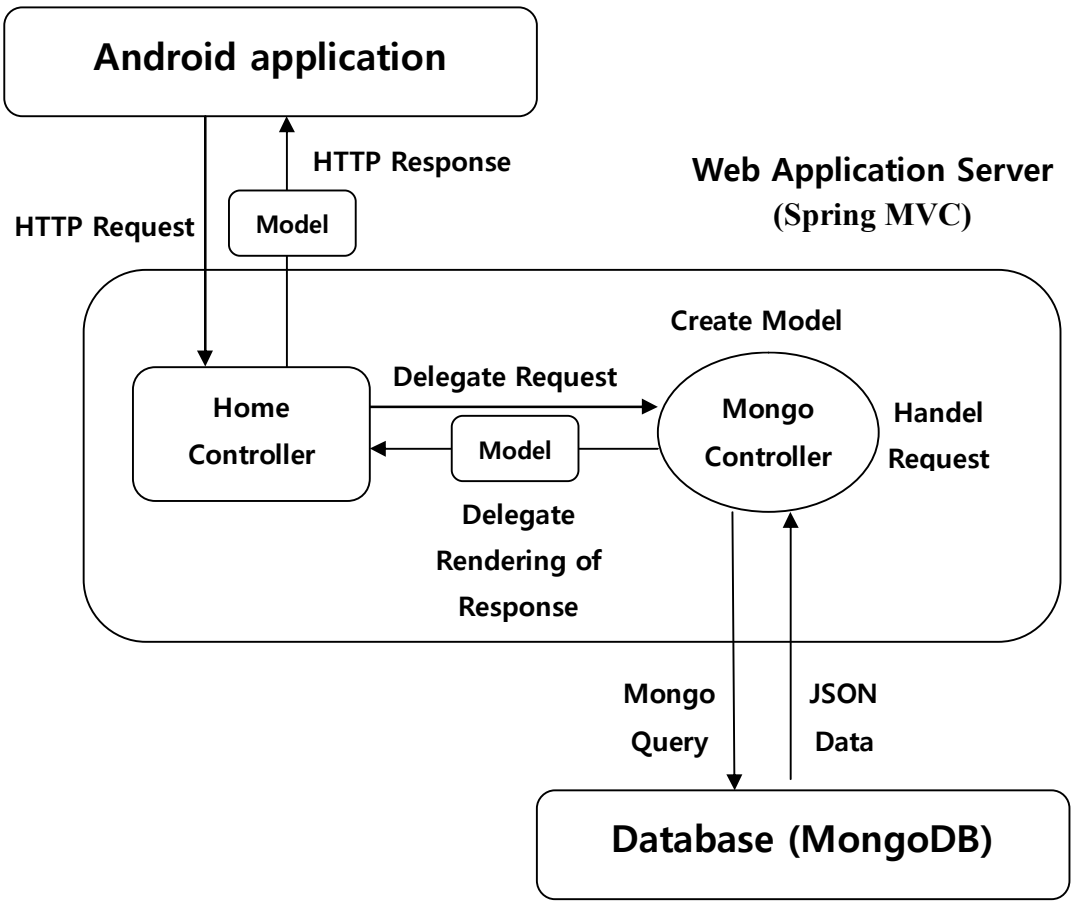


Figure 4.1 General structure of the Health Monitoring System

Since each part is developed by using different technologies, we need to look through which technologies are used before describing real development of the system.

4.1 Technologies

4.1.1 Web Application Server(WAS)

Before we even start to talk about Web Application Servers, let's look at term "web server" in general. A web server is generally a program that get request from clients and respond their requests using the Hypertext Transfer Protocol (HTTP). A protocol is a standard set of rules, which allows a server and clients to communicate that they must use the same protocol to communicate [55].

An Application Server is a term that sometimes is mixed with a web server. While a Web server mainly deals with sending HTML for display in a Web browser using mainly HTTP protocols, an application server provides access to business logic for use by client application programs through various protocols, including, but not limited to HTTP. The client application program can use this logic just as it would call a method on an object (or a function in the procedural world) [56].

Nowadays the application servers usually work together with web servers. It allows them to do more than just respond to simple requests for static documents, and many provide easy-to-use graphical user interfaces for administration and customization.

If we look at the request-response flow between client, web server and application server (see figure 4.1) when a client has requested for some other resources than normal web pages, such as database, the client's request first goes to the web server, which sends the required information to the application server. The web application accesses the databases servers to perform the requested task updating and retrieving the information lying within the database. The web application then presents the information to the user through the browser [57].

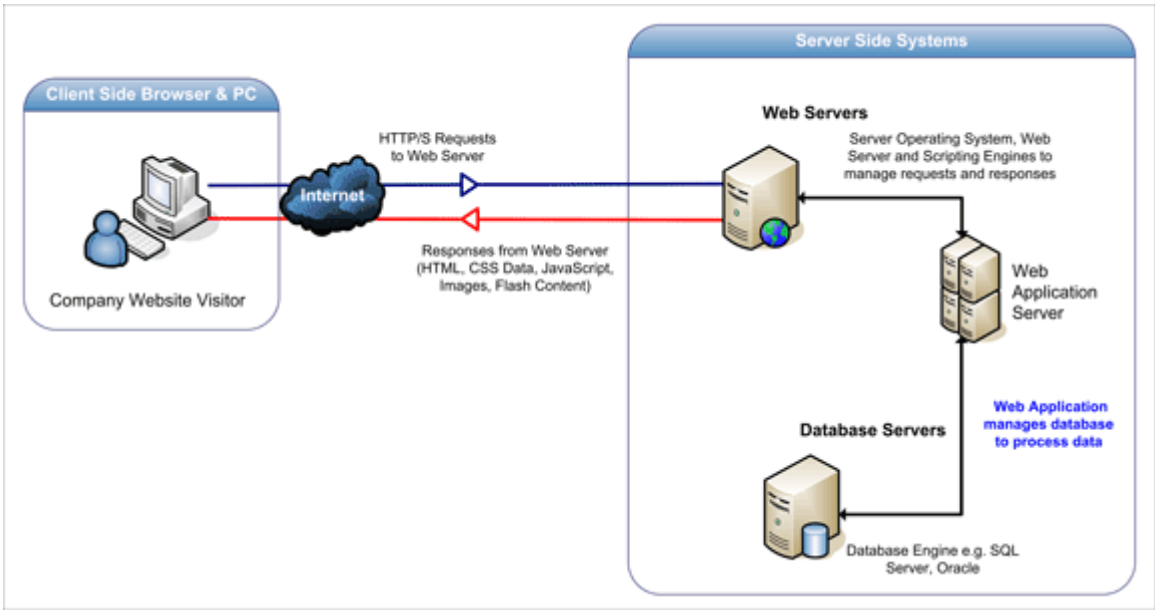


Figure 4.1 Workflow of a Web Application Server

The world's most used web application server software is Apache Tomcat, which is combination of web server Apache and Java application server Tomcat. As of July 2016, Apache Tomcat was estimated to serve 46.41% of all active websites [58], and Tomcat was estimated 63.9% in "Java application server market 2017" [59].

4.1.2 Apache Tomcat

As mentioned previously, in this thesis, world's most used web application server software Apache Tomcat is used. The Apache Tomcat server is an open source, Java-based web application container that was created to run servlet and JavaServer Pages (JSP) web applications. It is developed by the Apache Software Foundation and can be installed on almost all operating systems including Linux, UNIX, Windows, FreeBSD, Mac OS X and more [60].

As the web application server, the Apache Tomcat is combination of web server software Apache and application server Tomcat. The Apache Web server is cross-platform, lightweight, robust, and used in small companies as well as large corporations. The Apache, compared with other web server, has almost endless possibilities, due to its ability to pick and

choose various modules, which allows it to be integrated with numerous other applications. Moreover majority market share of the Apache has contributed strong user-community support. Meanwhile, Tomcat, as a web application server, provides additional functionality that makes it a great choice for developing a complete web application solution [61].

4.1.3 Spring Framework

The Spring Framework is a popular open source application framework that provides comprehensive infrastructure support for developing Java applications. It consists of a container, a framework for managing components, and a set of snap-in services for web user interfaces, transactions, and persistence. For developing the application, we can benefit from the Spring Framework with powerful and flexible collection of technologies such as :

- **Data access:** Support for traditional RDBMS as well as new NoSQL solutions (MongoDB), map-reduce frameworks and cloud based data services.
- **Security:** Authorization control for all tiers and authentication integration to dozens of providers.
- **Frontends:** Complete support for modern web technologies including REST, HTML 5, conversations and AJAX.

The Spring Framework is responsible to provide fast and secure access to the central data storage. It will expose RESTful web services and also host the CRUD web interface (CRUD - create, read, update, and delete).

The Spring Framework consists of features organized into about 20 modules to achieve different services and functionality for development of application, some popular modules are Spring Core(Base module), Spring JDBC(for persistence logic), Spring AOP(for cross context logic), Spring Transaction(For transaction support), Spring ORM and Spring MVC(Web aspect) [53][62]. However, the users do not have to use all of these modules. They are designed in such a way that no module is dependent to other module, except Spring core module. So based on their needs, Spring allows users to use only those parts that they need, without having to bring in the rest. In terms of web application, the Spring MVC

provides a full-featured MVC(Model View Controller) framework for creating web applications [53].

Spring MVC is a complete HTTP oriented MVC framework managed by the Spring Framework and based in Servlets. The most popular elements in it are classes annotated with @Controller and @RequestMapping annotations, where we implement methods we can access using different HTTP requests. With the introduction of Spring 3.0, the @Controller mechanism also allows us to create RESTful Web sites and applications, through the @PathVariable annotation and other features. Following figure shows the architecture diagram for Spring MVC [53][54],

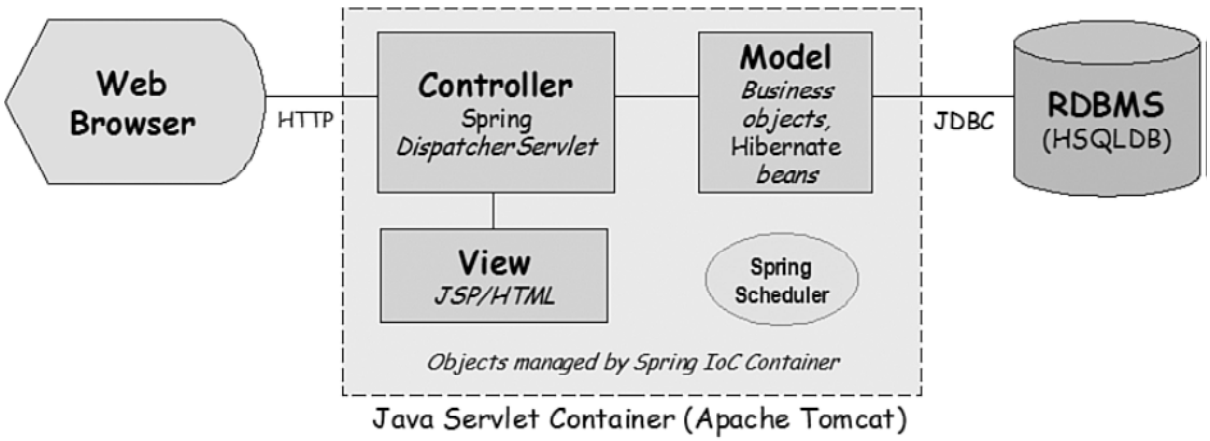


Figure 4.2 High-level architecture diagram for Spring MVC [54]

As we can see, all incoming HTTP requests from a web browser are handled by Controllers. A controller, as the name indicates, controls the view and model by facilitating data exchange between them. The key benefit of this approach is that the model can worry only about the data and has no knowledge of the view. The view, on the other hand, has no knowledge of the model and business logic and simply renders the data passed to it (as a web page, in our case). The MVC pattern also allows us to change the view without having to change the model.

4.1.4 REST API

Representational state transfer (REST) or RESTful Web services are a way of providing communication between computer systems on the Internet. RESTful Web services allow requesting systems to access and manipulate Web resources using a uniform and predefined set of stateless operations. “Web resources” were first defined as documents of files identified by their URLs, but today they have a much more generic and abstract definition as everything or entity that can be identified on the Web.

In a RESTful Web service, requests made to a resource's URL will elicit a response that may be in XML, HTML, JSON or some other defined format. The response may confirm that some alteration has been made to the stored resource, and it may provide hypertext links to other related resources or collections of resources. Using HTTP, as is most common, the kind of operations available include those predefined by the HTTP verbs GET, POST, PUT, DELETE and so on. By making use of a stateless protocol and standard operations, REST systems aim for fast performance, reliability, and the ability to grow, by re-using components that can be managed and updated without affecting the system as a whole, even while it is running [63][64].

HTTP-based RESTful APIs are defined with the following aspects:

- base URL, such as http://api.example.com/resources/
- an internet media type that defines state transition data elements, for example Atom, microformats, application/json, etc. The current representation tells the client how to compose requests for transitions to all the next available application states. This could be as simple as a URL or as complex as a Java applet.
- standard HTTP methods (e.g., OPTIONS, GET, PUT, POST, and DELETE)

4.1.5 MongoDB

MongoDB is an open source cross-platform document-oriented NoSQL database system, which stores structured data as JSON-like documents with dynamic schemas, making the integration of data easier and faster. MongoDB was founded in 2007 and since its inception. MongoDB has been downloaded 15 million times and is supported by more than 1,000

partners [28].

As a document-oriented database, MongoDB stores semi-structured data and descriptions of that data in document format. It allows developers to create and update programs without needing to reference master schema. MongoDB documents are similar to JavaScript Object Notation (JSON) objects, a data interchange format that has gained wide currency among web application developers, although XML and other data formats can be used as well. Document databases are used for content management and mobile application data handling [69].

NoSQL databases are useful for large sets of distributed data that relational databases aren't built to solve. They are most effective when an organization must analyze large chunks of unstructured data or data that's stored across multiple virtual servers in the cloud.
Furthermore, MongoDB provides high availability with replica sets. All the stored data will be automatically replicated to two or more copies. It provides automatic failover and automatic recovery of the data.

4.2 Development of Database Server

The first attempt to develop the Health Monitoring System is developing Database Server by creating the Server part and the Database part in parallel. A database server is simply the server connected with a database. The server is developed as a web application server by using Apache Tomcat software. The web application server is written in java and developed in Spring Framework + Maven environment by using Eclipse IDE (integrated development environment). The combination of Eclipse and Spring Framework is very popular choice for developing a web application server which is also called Spring MVC (Model-View-Controller) application [54]. This combination provides various plug-ins or dependencies for web server developing. For instance it allows the user to easily integrate Apache Tomcat web server technology in the project and provides REST API which provides simple way for request-response transactions of data.

The main purpose of the web application server is transferring data between Database and Android application. In order to accomplish this purpose, two controllers are developed in the server part: Home Controller and Mongo Controller. The Home Controller is responsible for communication with the android application, while the Mongo Controller is responsible for communication with the MongoDB database. The source codes of Home Controller and Mongo Controller can be found in Appendix B.

In order to guarantee the data security, the server is protected by combination of basic Spring Security Configuration [65] and JSON Web Token (JWT) [67] technologies that provide the user specific authorization token for each user when the user log in, and the user has to verify this token every time when the user wants to access to other services in the application.

In addition the server includes model structure that is used when the server receives data from the database and transfer data to the android application. The model structure is based on structure of the database, which will be described detailed in the next section. The structure of the project for server part and important source codes are introduced in Appendix A

4.2.1 Creating Demo Database

All the data including caregivers' username and password, and elderly people's physiological data are stored in the database. In the real world, the database requires sensors which collect various data from elderly people's home environment and send them to database. But in this thesis we assume that we use same sensor network as the GiraffPlus project (see chapter 3), so the monitoring system uses a demo database that is created for testing the application. The demo database is based on the database structure described in the document provided by GiraffPlus project [102]. The following UML shows basic structure of our demo database.

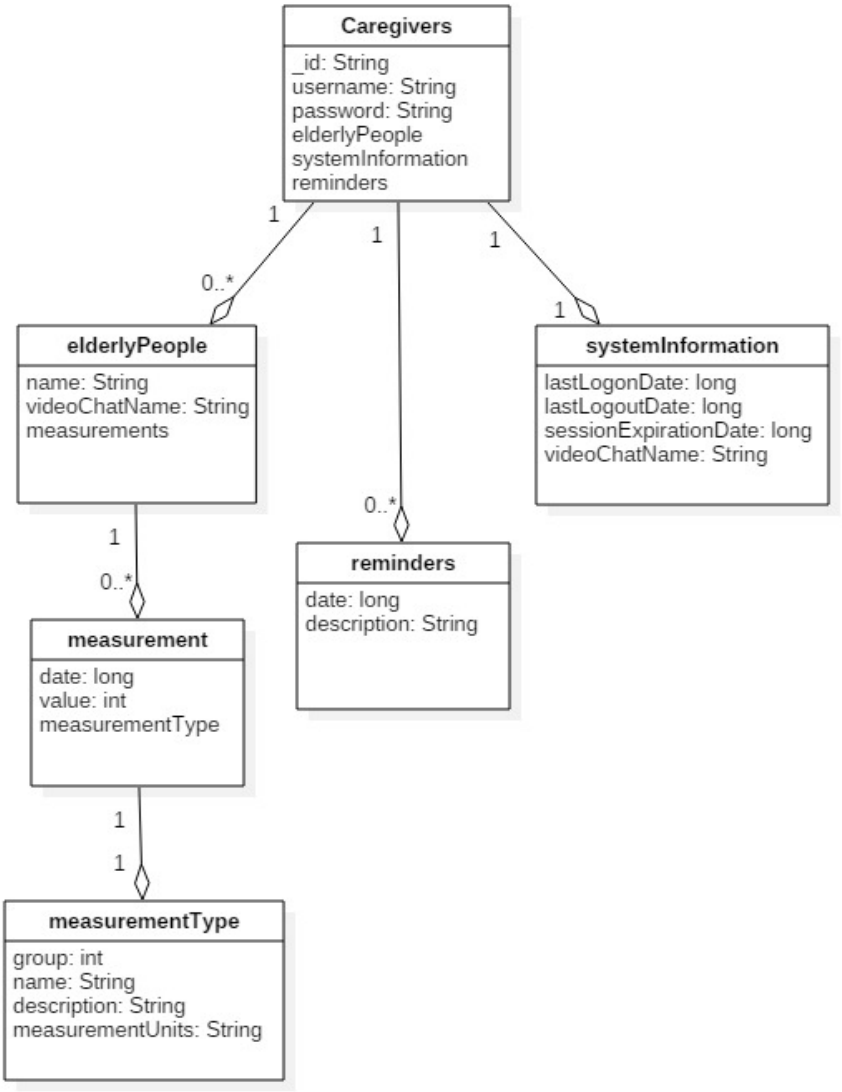


Figure 4.3 Basic Structure of the Demo Database

The demo database contains collections with JSON documents that contain user data for caregivers. Caregivers have their unique username and password that they use to log in to the application. Each caregiver has link to elderly people who the caregiver has responsibility to take care of. The most important data for our system is “measurement” since it contains physiological data for elderly people. The example of the "measurement" data is shown in the following example:

```
{
        "date" : NumberLong(1483574400000),
        "value" : 79.0,
        "measurementType" : {
                "group" : 3,
                "name" : "HR",
                "description" : "Heart rate",
                "measurementUnits" : "bpm"
        }
},
```

Example 4.1 The example of the "measurement" data in the database

The date value is indicated in the numerical value corresponding to the number of milliseconds since 1 January of 1970 00:00 GMT. For example [102]:

- "2012-09-12T15:24:47.657+0100" -> 1347459887664

The “measurementType” represents literally which type of the measurement the value belongs to. In the demo database there are 3 measurement types: Heart rate, Diastolic blood pressure and Systolic blood pressure. Appendix B contains an example of the demo database.

4.2.2 Communication Between Server and Database

In order to use the MongoDB database, the MongoDB server has to be started first. Starting the MongoDB server is arranged by executing mongodb.exe in the command line, and by default, the server starts at port 27017 [101].

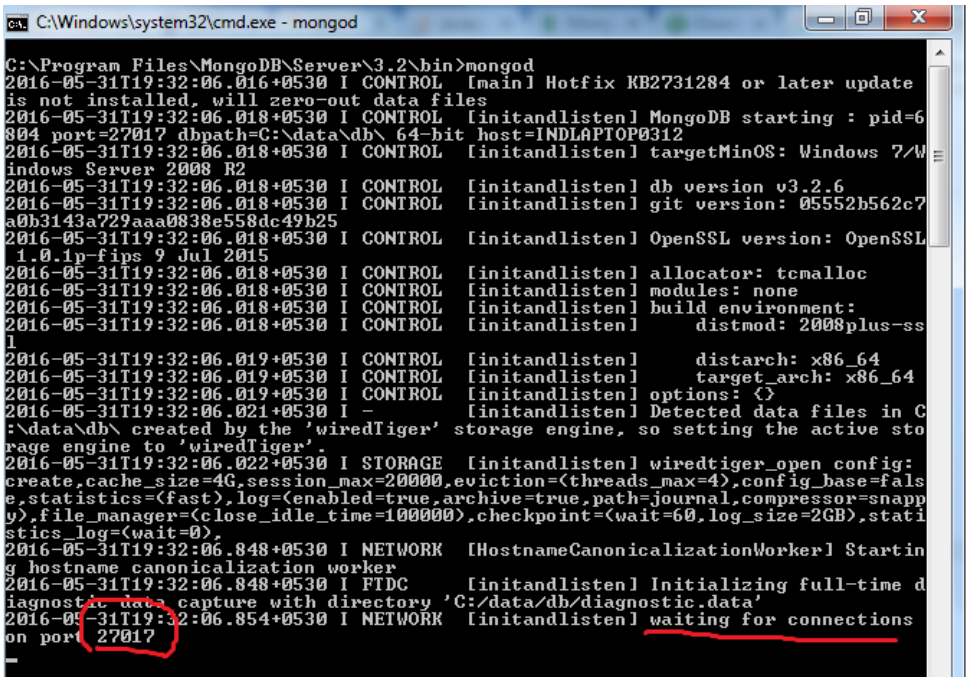


Figure 4.4 Starting MongoDB Server at Port 27017

In order that the server communicates with the database, “MongoTemplate” class is used in the server part, which is provided by Spring Framework. For using the "MongoTemplate", the class is declared in the "MongoController" with its configurations (see example 4.2). The "ServerAddress" contains host address and port of the MongoDB server. The name of the database, "hmt_database" is assigned in "Mongo DB Factory" [72][73].

```
ServerAddress serverAddress = new ServerAddress("localhost", 27017);

MongoClient mongoClient = new MongoClient(serverAddress,Arrays.asList(credential));

// Mongo DB Factory
SimpleMongoDbFactory simpleMongoDbFactory = new SimpleMongoDbFactory(
                                                    mongoClient, "hmt_database");

MongoTemplate mongoTemplate = new MongoTemplate(simpleMongoDbFactory);
```

Example 4.2 MongoDB configurations in MongoController class

By using the MongoTemplate simple Mongo query operations can be implemented like the following codes [74]:

```
Query query = new Query();
query.addCriteria(Criteria.where("username").is("testUser").and("password").is("testPass"));

Caregiver user = mongoTemplate.findOne(query, Caregiver.class, "Caregivers");
```

Example 4.3 Mongo query operation "findOne"

In example above, we create a query class that specifies certain document which has unique username (‘testUser”) and password (“testPass”). Then, the “findOne” method performs query on the “Caregivers” collection in the database, and create a “Caregiver” class (model) that contains results from the performed query.

The “MongoTemplate” provides not only “findOne” method, but also other methods that provide simple CRUD operations. For example we can find a certain caregiver and update his/her password to a new password (newPass) with following codes:

```
Query query = new Query();
query.addCriteria(Criteria.where("username").is("testUser").and("password").is("testPass"));

Update update = new Update();
update.set("password", "newPass");

mongoTemplate.updateFirst(query, update, "Caregivers");
```

Example 4.4 Mongo query operation "updateFirst"

By using these methods the server is able to implement almost all services that we need in this project, but there is still problem we have to consider that the methods described above are not able to perform more complicated query.

With the simple CRUD operations we described above, when we needs to find more specific data, such as “finding an array of measurement values of certain elderly person in certain period”, we have to receive all the data of the caregiver (document) from the database including all unnecessary data that we don't need to know, and then we have to find the data we want by using for example for-loop and if-statements in the server part. Actually this kind

of data is needed when we visualize a graph for providing certain measurement data to the caregiver. But if the caregiver has a large amount of data of for example 100 elderly people in the database, there may be some server lag every time the application receives measurement data from the server. So the server is supposed to be able to get only specific part of the data that the user actually want, not whole document.

To resolve this problem, the server part employs "Aggregations" operations provided by MongoDB [103]. The aggregation operations group elements from multiple documents together, and perform a various operations on the grouped data to return a single result. The aggregation operations are performed like a pipeline that consists of multiple stages. Each stage transforms the documents by using various operators as the data passes through the pipeline with filters. The operators used in the pipeline are called "Stage Operators". In this thesis we use mainly 4 stage operators:

- Match: Filters the documents to pass only the documents that match the specified condition(s) to the next pipeline stage.
- Unwind: Deconstructs an array field from the input documents to output a document for each element. Each output document is the input document with the value of the array field replaced by the element.
- Sort: Sorts all input documents and returns them to the pipeline in sorted order. Reorders the document stream by a specified sort key.
- Group: Groups documents by some specified expression and outputs to the next stage a document for each distinct grouping.

By using the aggregation operations, an example described above, "Finding an array of measurement values of certain elderly person in certain period", is performed with following codes:

```
AggregationOperation match1 =
                Aggregation.match(Criteria.where("username").is(secondaryName));
AggregationOperation unwind1 = Aggregation.unwind("primaryUsers");
AggregationOperation match2 =
                Aggregation.match(Criteria.where("primaryUsers.name").is(primaryName));
AggregationOperation unwind2 = Aggregation.unwind("primaryUsers.measurements");
AggregationOperation match =
                Aggregation.match(Criteria
                .where("primaryUsers.measurements.measurementType.group").is(group)
                .andOperator(Criteria.where("primaryUsers.measurements.date").gte(date),
                                Criteria.where("primaryUsers.measurements.date").lte(endDate)));

AggregationOperation sort =
                Aggregation.sort(Sort.Direction.ASC, "primaryUsers.measurements.date");
AggregationOperation group_query =
                Aggregation.group("null").push("$primaryUsers.measurements.value")
                .as("output_measurements");

Aggregation aggregation =
        Aggregation.newAggregation(match1,unwind1,match2,unwind2,match,sort,group_query);

AggregationResults<QueryOutput> result =
                mongoTemplate.aggregate(aggregation, "Secondary_Users", QueryOutput.class);
```

Example 4.5 The process of the Aggregations operations

In the example 7 operators are used in order of i) match, ii) unwind, iii) match, iv) unwind, v) match, vi) sort and vii) group. The detailed processes of the operations are:

i) First operation "match" filters the documents to find only the documents that have the username of the user. Since the username is a unique element in the database, the result will be a document with data of the user.

ii) Operation "unwind" deconstructs "primaryUsers" field and return several identical documents which have each elements of "primaryUsers" field.

iii) Operation "match" filters deconstructed documents and returns only a document which have specific primary user name.

iv) Operation "unwind" deconstructs "measurements" field in the "primaryUsers" field and returns several identical documents which have each "measurements" elements.

v) Operation "match" filters deconstructed documents and returns the documents which contain certain type of measurement values that are measured in certain period.

vi) Operation "sort" sorts out documents by "date" value in "measurements" elements.

vii) Operation "group" makes a new document that contains measurement values specified by operations.

The operations are performed by “aggregate” method and the result will be returned as an array of the measurements values.

4.3 Development of Android Part

In the monitoring system, the android part is responsible for handling general front-end services of the application, including design of the user interface. The main purposes of the android part in the monitoring system are: i) Requesting certain data the user wants to receive, and ii) Visualizing received data with a graph. The android part is written in native code by Android Studio, an official IDE for Android operating system announced on 2013 by Google.

4.3.1 UI of Android Application

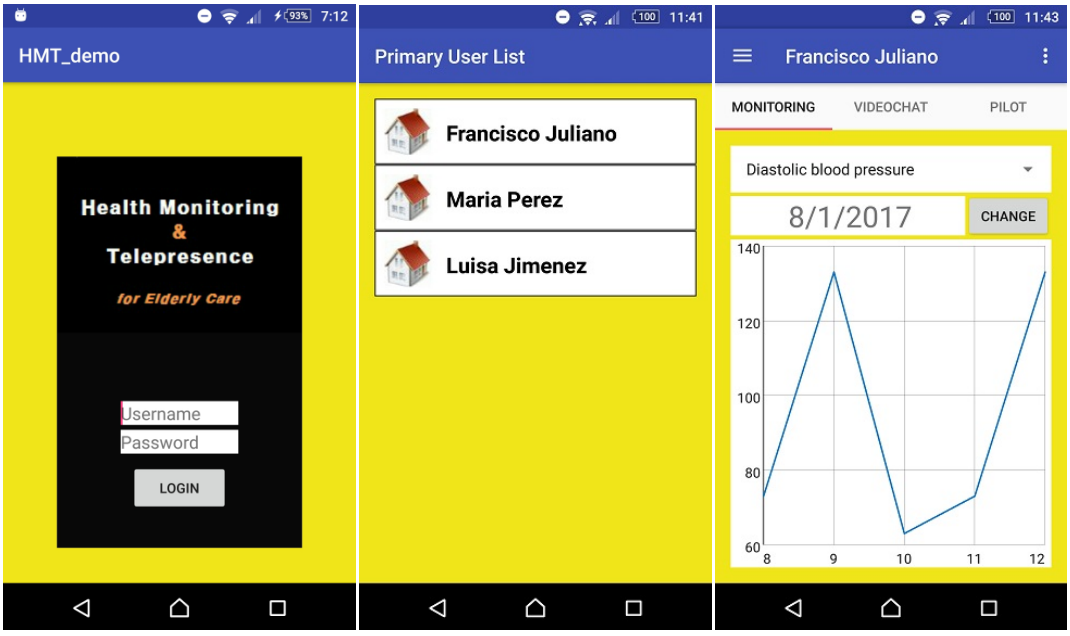


Figure 4.5 Basic UI of Android Application

Since the fundamental idea of the application comes from DVPIS software of GiraffPlus project (see figure 2.2, 2.3 and 2.4), the basic UI of the application is very similar with it. The monitoring service of the application includes the following components:

① Basic login system that requires username and password.

② List of elderly people currently under authorized access by the logged user. The list shows each elderly people's name.

③ Tap bar which has 3 taps for monitoring service, video chat service and pilot service. The tap can be changed by clicking or touching one of elements in the tap, or swiping the screen from side to side.

④ Graph that displays chosen elderly person's physiological data. X-axis presents date of the each point and y-axis presents the value of the data. Each axis displays most 5 points in order that user's readability.

⑤ Area where the user can change type of the physiological data with top-down menu, and date value from calendar.

⑥ Navigation bar which has three elements that lead to other functions.

⑦ Navigation drawer, a panel that displays several services on the left edge of the screen. It is appeared when the user touches the icon in the action bar (see the figure 4.7).

The design of the interface and the interaction with the application has the clean and simple feeling that was planned. The design lacks some testing from users outside of the development team and consequently some additional rework has to be accounted for in future iterations.

4.3.2 REST API in Server Part

The request-response transactions between android application and the server are performed in “HttpRequests” class of the android part and “HomeController” class of the server part. Access to the service will be allowed by using REST calls. Calls consist of a base URL of Apache web server, “http://localhost:8080/”, with addition of the resource identifier and the parameters used in the call. The data encapsulation supported is JSON. Upon receiving the request, the server sends back a status line, such as "HTTP/1.1 200 OK", and a message of its own. The body of the message is typically the requested resource, although an error message or other information may also be returned.[2]

The "HomeController" class in the server part contains several methods that respond to HTTP requests from the android part. When the server receives a HTTP request, the server matches the URL of incoming request with access URL of the methods, and assigns the request to the matched method. Some important methods created in “HomeController” class are following:

- Access URL: {baseURL}/authenticateUser/{username}/{password}
  Description :
  GET method responsible for authenticating the user. Two string parameters “username” and “password” are provided from login page in the application. If the authentication successes, the server creates a temporary token providing the user identification when executing the other service calls. The token will be included in the header of HttpResponse and sent back to the application. The method returns a string array with elderly people’s names as well.

- Access URL : {baseURL}/getMeasurementsSetting
  Description :
  GET method responsible for obtaining the various data that are necessary to make monitoring activity in application. The method returns “MeasurementSetting” class that contains variables that we need to set up the graph

- Access URL : {baseURL}/getMeasurementsByPeriod/{group}/{date}/{periodType}
  Description :
  GET method responsible for obtaining the specific type of measurements in the certain period. Parameter “group” is number of measurement type the user wants to find. Parameter “date” represents start date of time period. Parameter “periodType” represents how many values the user wants from the start date. In the demo database the “periodType” are fixed with 5. The method returns int array with the values of measurements.

4.3.3 REST API in Android Part

The HTTP requests from the android application are sent by using "RestTemplate" class provided by Spring Framework. The "exchange" method of the "RestTemplate" provides simple way to send HTTP request to the server and receives the HTTP response automatically. The example codes of using "RestTemplate" are:

```
RestTemplate restTemplate = new RestTemplate();
restTemplate.getMessageConverters().add(new MappingJackson2HttpMessageConverter());
HttpHeaders header = new HttpHeaders();
header.add("Authorization", token);
header.add("primaryUser", primaryName);
HttpEntity<String> httpEntity = new HttpEntity<String>(header);

ResponseEntity<int[]> response =
restTemplate.exchange(baseURL+"/getMeasurementsByPeriod/" + group + "/" + date + "/" +
periodType, HttpMethod.GET, httpEntity, int[].class);
int[] measurements = response.getBody();
```

Example 4.6 The example of using "RestTemplate" class

The example codes send HTTP request to the server for calling "getMeasurementsByPeriod" method in the "HomeController" class in the server (see the previous chatper). The "RestTemplate" class includes HTTP headers that contains authentication token and name of the elderly person the user wants to access. In the example, the response from the server contains an array of measurement values and we can get the array by using "getBody" method.

4.4 General Workflow of Monitoring System

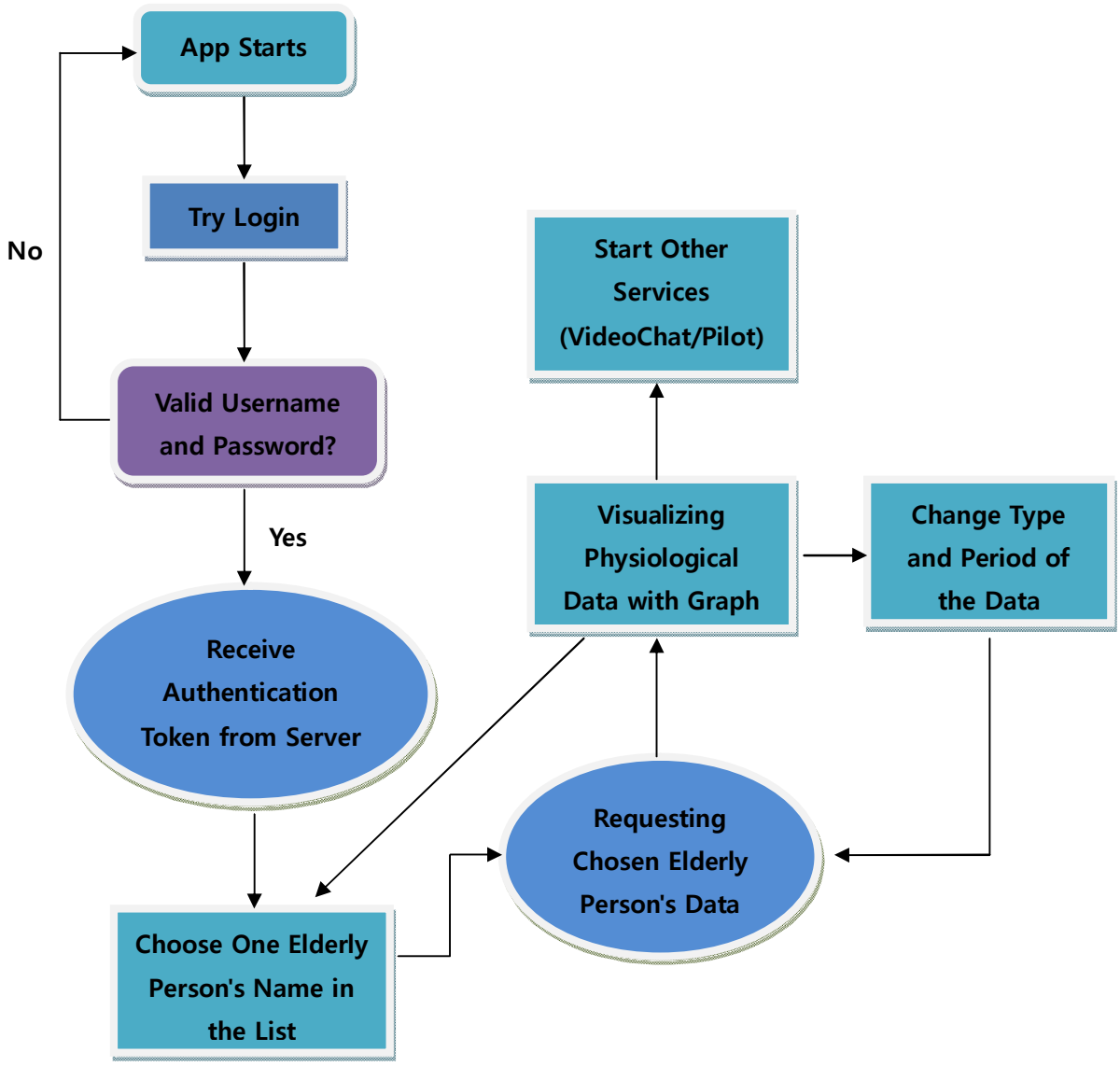


Figure 4.6 General workflow of the Monitoring System

1) Login system which requires valid user id and password

When we start the application the login page is displayed (figure 4.5). Before the user try to login, the web server and database naturally has to be turned on. The login system requires valid username and password, which are stored in demo database. When the user types in the username and password, and click the login button, the application sends request with the inputted username and password to the server. The server communicates with the database and check if the values are valid - in other words check if there are same username and

password in the database, and if the values are verified the server sends response including an array of elderly people's names and a token value as well. If the user types in invalid username or password, naturally an error message appears and the login fails. All other services than authentication service that are performed by server, require the token value the user received when the user login.

2) Choose one elderly person the user wants to access

When the user logs in successful, a list with names of elderly people is displayed in the application. By choosing one of those names, the application automatically requests the latest physiological data of the chosen elderly person to the server.

3) Visualizing physiological data

The application receives response from the server and the received data is visualized as a graph. For readability of the graph, only values of 5 days are plotted in the graph. Type and period of the values can be changed by multiple choices spinner and a calendar above the graph.

4.5 Implementation of Reminder Service

As a part of the Health Monitoring Service, a reminder service is implemented. The process of the service is very simple. When the user starts the service by touching "Reminder" in the navigation drawer (see the figure 4.7), reminder window is appeared (see the figure 4.7). The user can adjust the date and time, and the description of the reminder message. If the user touches "confirm" button, the application make a reminder model that contains "date" value with long number and String "description" value. The model is sent to the database via server and stored in the reminders field, as a following example:

```
"reminders" : [
      {
            "date": NumberLong(1510137900000),
            "description": "test description"
      }
]
```

Example 4.7 Data model of "reminders" field in the database

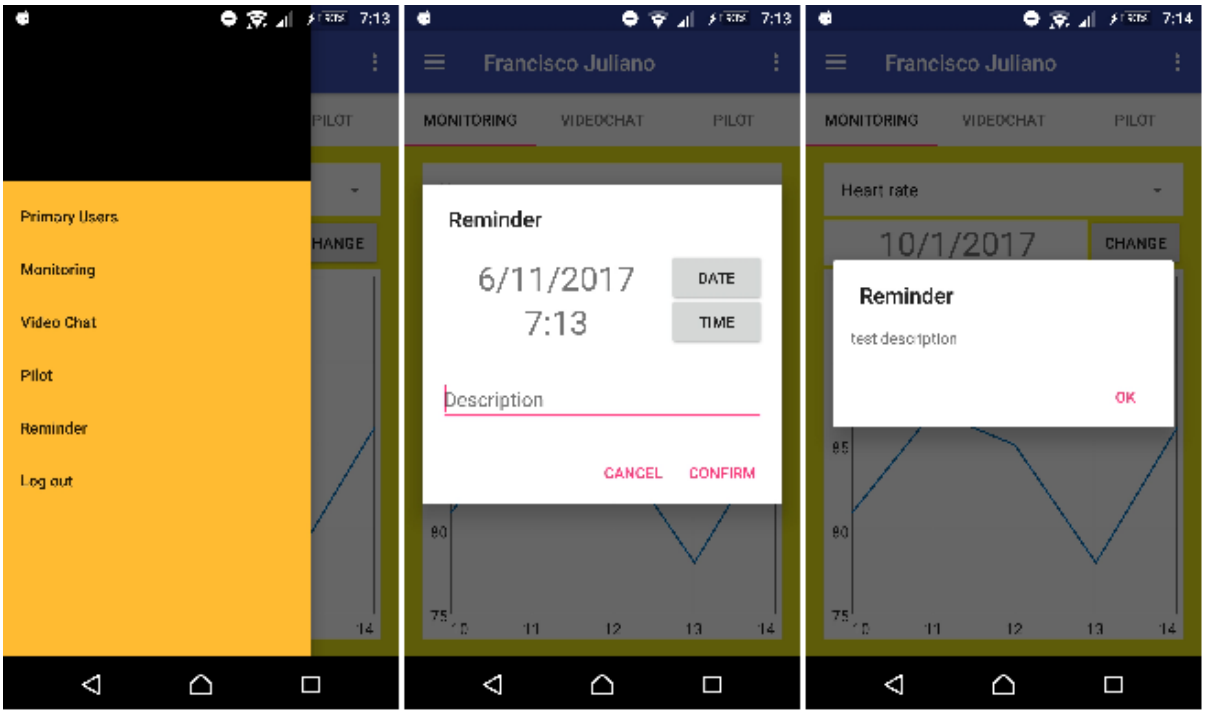


Figure 4.7 Navigation drawer (left), Set reminder window (middle), Get reminder window (right)

After the application sends the reminder data, the application checks continuously if there is any reminder data that has the less date value than current date value. If there is, the application displays reminder window with the description of the reminder data. The figure is showing the result of getting a reminder data.

4.6 Hosting the Database Server

As described above the database server is run on the localhost with certain ports. It means if the android application wants to communicate with the database server, the android device need to be connected with the same Wi-Fi network with the server. It's not that we want, and in order to reach one of our goals, "Allowing the caregivers to monitor physiological data of elderly people whenever and wherever they want", hosting the database server is necessary.

Web Hosting Service allows the users to make the server accessible via the standard Internet. In the database server, there are two servers we need to host, the web application server and the mongoDB server, and these two servers will be hosted in different ways.

The web application server is hosted by using Amazon Web Services (AWS) [104]. The AWS provides various services, each of which exposes an area of functionality. In this thesis we use "Elastic Beanstalk" service that allows us to upload our web application source codes, and provisions us a unique host URL for our server. Elastic Beanstalk supports applications developed in various languages such as Java, PHP, Node.js and Python. When we upload our web application source codes as a .zip file, the Elastic Beanstalk service automatically hosts our server with the assigned host URL. The hosted server URL needs to be declared as "baseURL" variable in the "HTTPRequests" class in the HTTPRequests class (see Appendix A).

For hosting the mongoDB server, "mLab" cloud database service [105] provides an easy way to host mongoDB server. The only thing we have to do is uploading the database to the "mLab" service, and the service provisions us a unique host URL for our mongoDB server. As we have seen in the example, the hosted server URL needs to be declared in the MongoController class as ServerAddress class (see Appendix B).

With the hosted database server, we can finally use the health monitoring system via Internet whenever we want.

5 Robot Telepresence System

The second goal of this thesis is developing a low cost telepresence system using an android application and existing mobile robot. Most of commercially available robots have specialized hardware for processing activities, and such robots increase the cost rapidly. Therefore, as the main cost reduction method, our design consists of a normal laptop mother board and a processor for the main processing hardware of the robot. In addition, the other peripherals of the laptop such as screen, microphone, speakers, webcam (if equipped), wireless network card are used with the streaming and networking activities at the robot.

The "TurtleBot" is chosen as the robot we use. The application will have services for interacting with the "TurtleBot" via standard internet. Our robot telepresence system requires two main components:

- Mobile robot control via android application (pilot system)
- Videoconferencing system between android application and mobile robot (video chat system)

The result is a hybrid application, which is written in combination of JavaScript and HTML, and integrated into android application by using Cordova software [48]. There are two reasons we use web based application, not using native application. The first one is for using web based open source codes and libraries available in internet and the second one is for allowing users to use the telepresence system both via android application and desktop with the same API. The second reason is also considering elderly people to use the telepresence system at home by using their desktop or laptop.

5.1 Technologies

5.1.1 WebRTC

WebRTC is a powerful tool, released by Google in May 2011 that can be used to infuse Real-Time Communications (RTC) capabilities into browsers and mobile applications. WebRTC is being standardized by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF). The WebRTC provides peer to peer connection between applications, and by using the webRTC modern web applications can easily stream audio and video content to millions of people. Recent years, many kinds of videochat program using webRTC are developed as web application. WebRTC provides 4 main JavaScript APIs [49][51]:

- getUserMedia(): capture audio and video.
- MediaRecorder: record audio and video.
- RTCPeerConnection: stream audio and video between users.
- RTCDataChannel: stream data between users.

getUserMedia() and MediaRecorder are used to capture and record audio and video respectively. RTCPeerConnection is used to establish peer to peer connection between clients, but WebRTC still needs a signaling server to establish connection.

In order for a WebRTC application to set up a 'call', its clients need to exchange information by using signaling server:

- Network information such as IP addresses and ports
- Session control messages and error messages used to open or close communication
- Information about media and client capability, such as resolution and codecs

It is not generally possible to setup a WebRTC peer connection without a signaling server because most users do not have fixed IP addresses and are behind network address translation (NAT) systems and stateful firewalls. The signaling server relays messages between two users until they can successfully negotiate a peer connection using. As part of this process, the WebRTC APIs use STUN servers to get the IP address of our computer, and TURN servers to function as relay servers in case peer-to-peer communication fails, as the figure 5.1 shows. Actually the signaling server is not implemented by the WebRTC APIs, users need to

build it themselves [83].

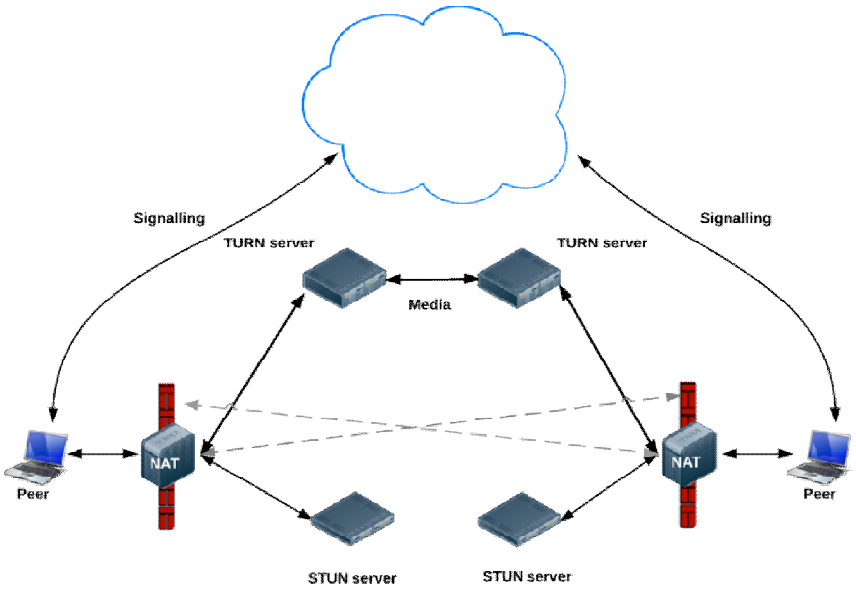


Figure 5.1 The WebRTC signaling system using STUN, TURN and signaling server [83]

5.2 Development of Web Application for Pilot System

As the first attempt to develop a telepresence system, a web application which interacts with "TurtleBot" using ROS is developed.

5.2.1 UI of Web Application

The developed web application has very simple UI that consists of mainly 3 components: Navigation bar, Main view and Virtual controller.

Figure 5.2 Main UI of Web Application

On the top of the application there is a navigation bar which has 2 elements, Video and Map. Each element is connecting to different html pages that display camera view and map respectively from "TurtleBot" in the main view.

The virtual controller is used to control the robot. All html pages have same virtual controller on the main view. The controller has common shape with arrows for each direction e.g. "up" arrow driving the robot forward and "down" arrow backward.

5.2.2 ROS Communication of TurtleBot

As mentioned above “TurtleBot” is operated by using ROS, and the ROS is running in Linux operating system. For using ROS a Linux pc has to be connected with “TurtleBot” robot. The connection can be both physically using USB port and using web server, but in this thesis the physically connection will be used.

As described in the section..., communication system of ROS is messages passing system between nodes, and nodes send and receive messages each other by publishing topics that contain messages. For using nodes in the ROS, we need to launch ROS packages first. A ROS package is kind of organized software in the ROS which contains ROS nodes, a ROS-independent library, a dataset, configuration files, a third-party piece of software, or anything else that logically constitutes a useful module [106]. The packages are launched on the Linux pc connected with "TurtleBot" by executing "roslaunch" scripts for each packages.

For developing the web application that using ROS communication protocol, we first use several simulators that create virtual world with TurtleBot model, before using the real TurtleBot robot. In addition, navigation tools that help the user building a new map or navigating the robot are also used. There are many useful open source tools we can use, and in this thesis we will use “rviz” 3D visualization tool [87], gmapping [88] and amcl [89]. The "roslaunch" scripts of the simulators and tools are:

- roslaunch turtlebot_gazebo turtlebot_world.launch
  - Launching gazebo simulator for creating virtual world with Turtlebot model (see figure 5.3).
- roslaunch turtlebot_ gazebo gmapping_demo.launch
  - Launching a 2D map yaml file. It can be a new file for building a new map or an existing map file for navigating the robot.
- roslaunch turtlebot_gazebo amcl_demo.launch
  - Launching a costmap which uses sensor data and information from the static map to store and update information about obstacles in the map. (see figure 5.4)
- roslaunch turtlebot_rviz_launchers view_navigation.launch
  - Launching rviz tool which visualizes map and costmap provided by gmapping and amcl, and robot model on the right position in the map. (see figure 5.4)

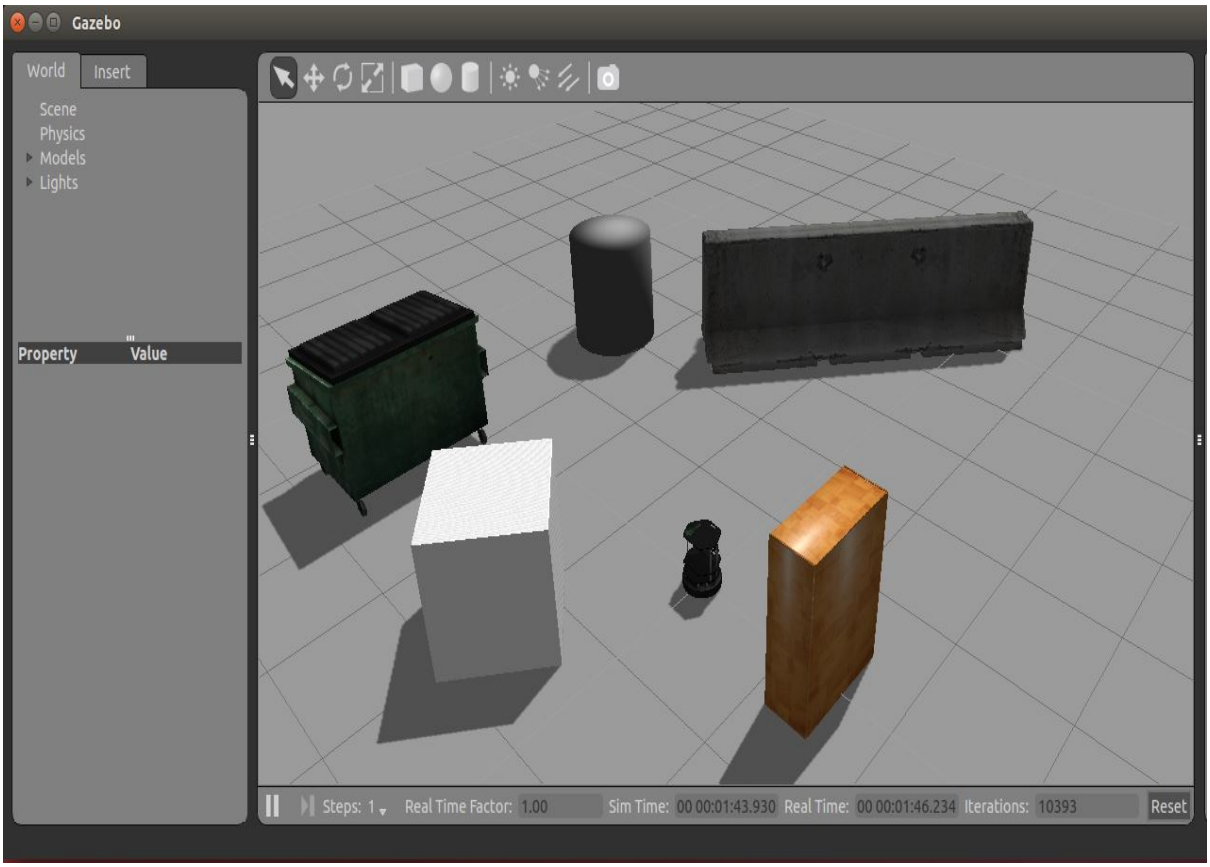


Figure 5.3 Gazebo World with TurtleBot robot

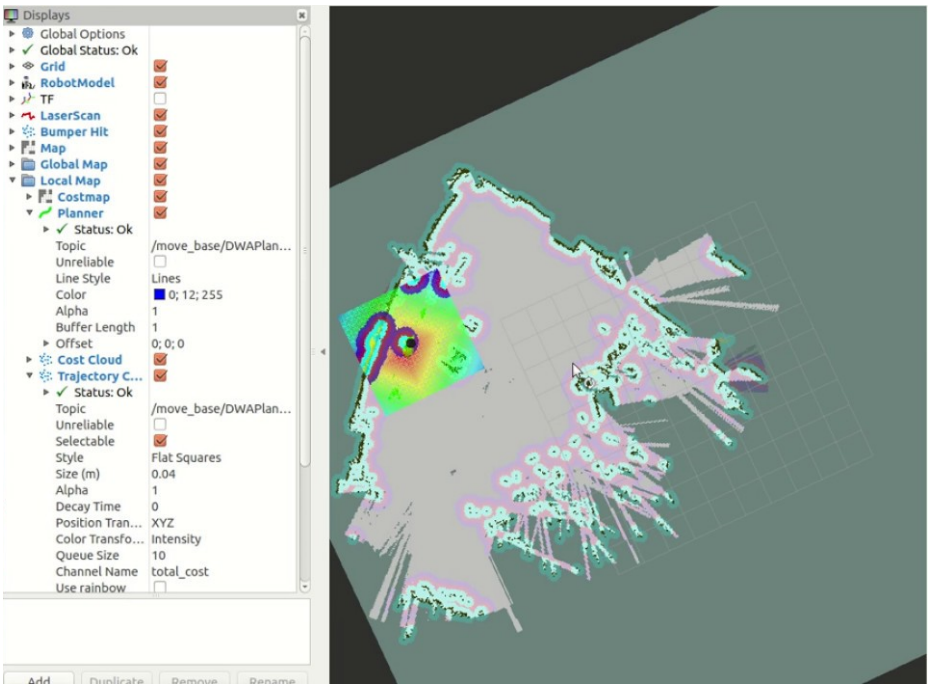


Figure 5.4 Rviz tool for visualizing map and robot model

5.2.3 Communication between Web Application and ROS

Now we are finished preparing to interact with TurtleBot in the virtual world, by using the web application. For developing the web application open source JavaScript Libraries "roslibjs"[84], "ros2djs"[85] and "nav2djs"[86] provided by RobotWebTools [109] are used. "roslibjs" is a standard ROS JavaScript library which provides easy way to establish connection with “rosbridge” and API for subscribing and publishing ROS topics. "ros2djs" is JavaScript libraries for 2d visualization of ROS topics, including visualizing RGB camera view and map topics from "TurtleBot".

The web application is based on open source example codes "simple.html" and "node_simple.js" provided by RobotWebTools [109] that shows simple way to use "roslibjs" library in the web application [84].

To be able to control and navigate the TurtleBot through the web application the "rosbridge" has to turn on. Rosbridge provides a JSON API to ROS functionality for non-ROS programs. With rosbridge non-ROS programs including web application which can interact with ROS via web socket. The rosbridge protocol allows users to send and receive JSON based commands from ROS that covers subscribing and publishing topics, service calls, getting and setting parameters, and even compressing messages and more. To launch the rosbridge and its packages a launch file has to be executed [52]:

- roslaunch rosbridge_server rosbridge_websocket.launch

This will run rosbridge and create a WebSocket on port 9090 by default. Then in the web application we need to create a ROSLIB.Ros node object to communicate with the rosbridge, as the example below:

```
var ros = new ROSLIB.Ros({
        url : 'ws://localhost:9090'
});
```

Example 5.1 Establishing connection with rosbridge server

The establishing connection is started automatically when the application is started and the

navigation bar shows whether the connection is successfully established or not (see the example)

In order to interact with the “TurtleBot”, the web application publishes and subscribes ROS topics by using “roslibjs” library. The “roslibjs” provides “ROSLIB.topic” object for creating ROS topic. The topic contains its name and message type. For an example of publishing ROS topic is:

```
var cmdVel = new ROSLIB.Topic({
   ros : ros,
   name : '/cmd_vel_mux/input/teleop',
   messageType : 'geometry_msgs/Twist'
});

var twist = new ROSLIB.Message({
   linear : {
      x : 0,
      y : 0,
      z : 0
   },
   angular : {
      x : 0,
      y : 0,
      z : 0
   }
});

cmdVel.publish(twist);
```

Example 5.2 Creating and publishing ROS topic object

In the example the application publishes a ROS topic “/cmd_vel_mux/input/teleop” with message type “geometry_msgs/Twist” for controlling the robot. By using the virtual controller the application changes these x, y and z values and publishes the topic to the “TurtleBot”. Values of "linear" makes the robot moves in one direction, while values of "angular" makes the robot turns. For example when “up” arrow button is pushed, x value of “linear” is changed to 0.25 and is published to ROS. It makes the robot "move straight forward" in speed 0.25 until the button becomes inactivated.

Other than "/cmd_vel_mux/input/teleop" the application subscribes 3 topics for visualizing camera view, map image and costmap image:

- Topic name: /camera/rgb/image_raw/compressed
  - Message type: sensor_msgs/CompressedImage
- Topic name: /map
  - Message type: nav_msgs/OccupancyGrid
- Topic name: /move_base/local_costmap/costmap
  - Message type: nav_msgs/OccupancyGrid

Subscribing ROS topic is also easy by using ROSLIB. The example for subscribing camera view from the ROS is:

```
var sub_image = new ROSLIB.Topic({
      ros : ros,
      name : '/camera/rgb/image_raw/compressed',
      messageType : 'sensor_msgs/CompressedImage'
});

   sub_image.subscribe(function(message){
      //code for displaying subscribed image in the main view of the web application
   }
})
```

Example 5.3 Subscribing ROS topic object

But in this thesis, the topic of map image is subscribed in different way. The ros2djs and nav2djs libraries that are provided by RobotWebTools [109], provide more convenient way to subscribe topics that contain 2d images and help to render the images to a HTML5 canvas element. In this thesis we use two methods from libraries, "Viewer" method from ros2djs library and "OccupancyGridClientNav" method from nav2djs library. The "Viewer" method creates the main viewer in the "canvas" element, and the "OccupancyGridClientNav" method subscribes the topics for map image and renders the images to the main viewer the "Viewer" method created. The only thing we have to do for using the methods is assigning option variables of the methods. The example 5.4 shows how the methods are used for subscribing map view topic. The source codes of the "index.js" of pilot system can be found in Appendix C.

```
// Create the main viewer.
var viewer = new ROS2D.Viewer({
      divID : 'map',
      width : window.innerWidth,
      height : window.innerHeight
});

// Setup the nav client.
var nav = NAV2D.OccupancyGridClientNav({
      ros : ros,
      topic : '/map',
      rootObject : viewer.scene,
      viewer : viewer,
      continuous : true,
      serverName : '/move_base',
      actionName : '/move_base_msgs/MoveBaseAction',
      withOrientation : true
});
```

Example 5.4　Creating map viewer in the web application

5.2.4 General Workflow of Pilot Web Application

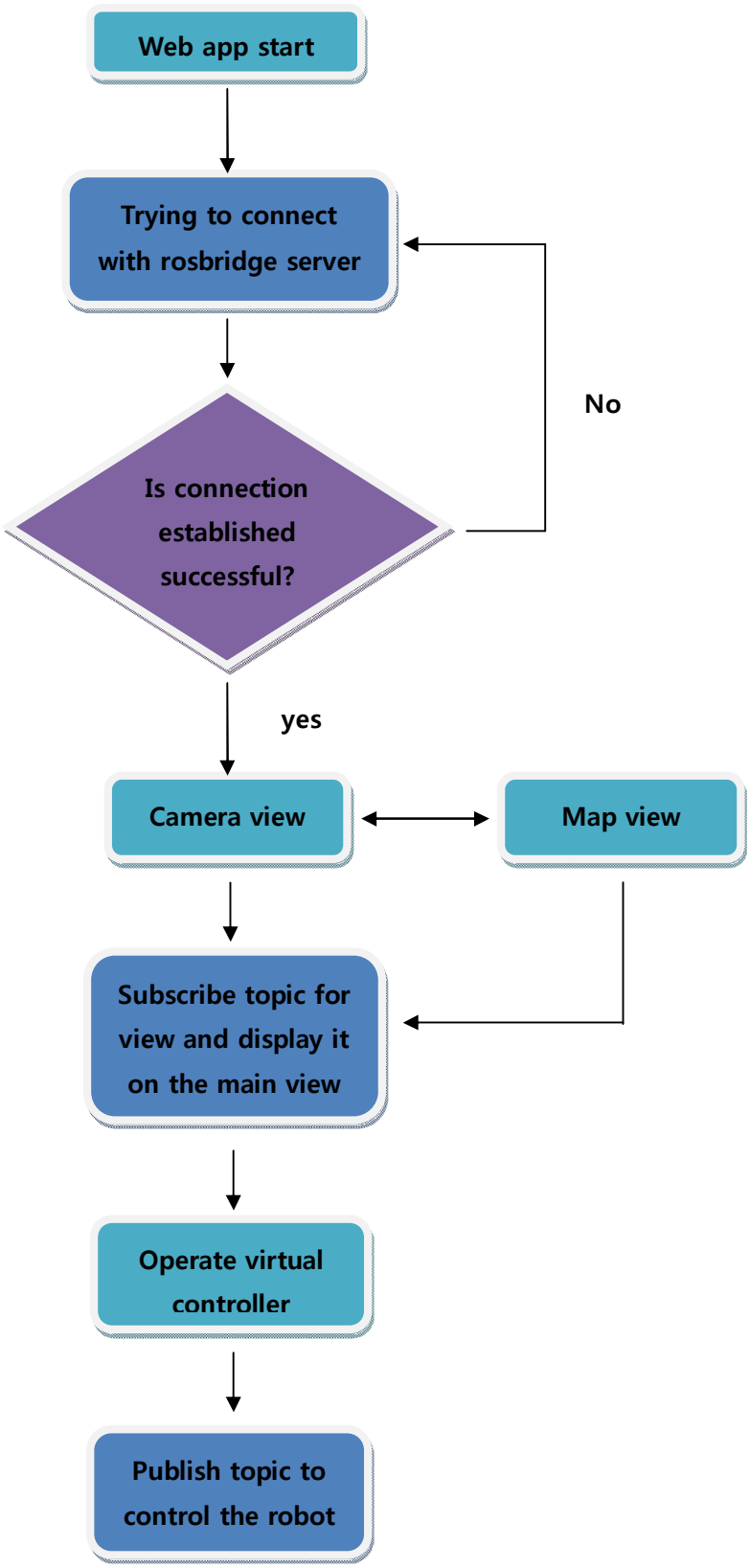


Figure 5.5. The General Workflow of Pilot System

5.2.5 Local Costmap

The main view of the map page is actually displaying map + costmap as we can see in figure 5.6. The costmap shows information about obstacles around the robot. It's very useful information to build a new map by moving the robot around the map. But only costmap without map we can't know where in the map the robot has gone through but there is only information around the robot. So in the map view the application subscribe both normal map topic and costmap topic by using "ros2djs" and "nav2djs" libraries (see chapter), and display both in the main view.

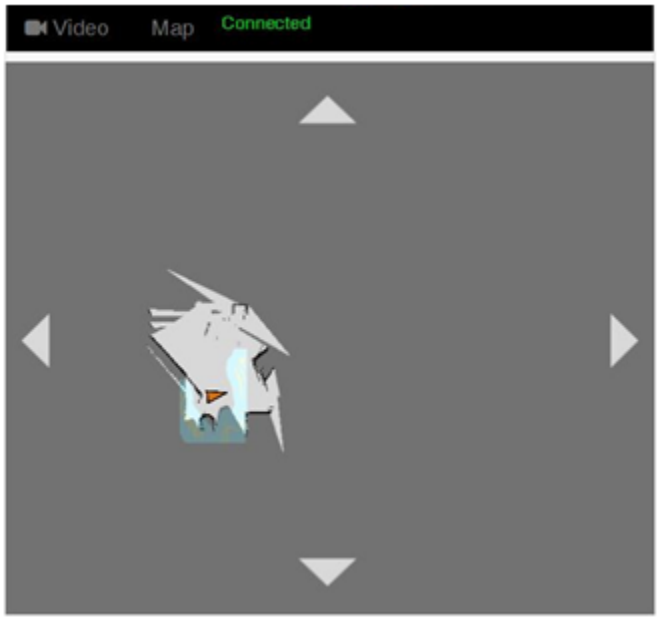


Figure 5.6 Web application with map + local costmap view

5.2.6 Autonomous Navigation of the Robot

As a part of the pilot system, autonomous navigation of the robot system is implemented in the map view of the web application. Actually the autonomous navigation system is implemented automatically when we implement subscribing topics for map image. The "OccupancyGridClientNav" method of the "nav2djs" library that is used for creating map view, calls another method "Navigator" that provides simple click-to-navigate function on the map view. When we touch or click a point in the map a goal marker is appeared on the clicked point with a pink triangle and the robot start to move to the goal, avoiding detected

obstacles as shown in the figure 5.7.

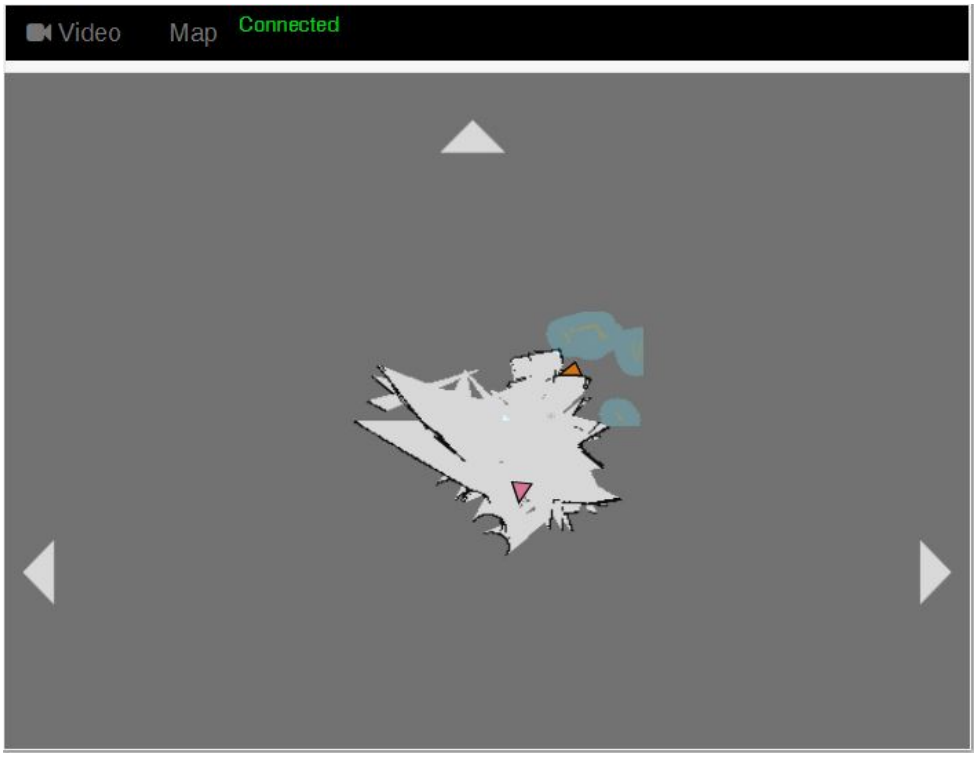


Figure 5.7 Autonomous navigation with goal marker

The autonomous navigation uses navigation stack which takes in information from odometry and sensors via tools such as map_server, gmapping and amcl, and makes a safe plan for navigation to the destination. The "Navigator" method of the "nav2djs" library performs the navigation by sending a message: "move_base_msgs/MoveBaseActionGoal" which contains coordinates of the goal marker.

5.2.7 Hosting the Rosbridge Server

As we hosted the database server, it is necessary that the application is able to connect to the rosbridge server via Internet. Unfortunately, I couldn't find a typical way to host the rosbridge server. The biggest problem is that the rosbridge server has to be run in the same Linux pc with other ROS packages. So in this case, we can't use typical web hosting service such as Amazon Web Service that we used in the chapter...

The techonology I chose for hosting the rosbridge server is "ngrok" that exposes local web

servers to the public internet over secure tunnels [108]. By using the ngrok we can decide what port of our local server is going to be exposed. The ngrok is started by "ngrok http" command with the port number. Since the rosbridge server is using port 9090, the command will be "ngrok http 9090". When we start ngrok server, the ngrok provides us public URL with random host name such as "http://92832de0.ngrok.io" that is connected to our local server on port 9090.

Ngrok provides also unique host name if the user pay the monthly payment. The server with unique host name is protected by basic auth credentials with username and password. The command has to include "-auth" option, for example:

- ngrok http -auth "user:password" 9090

The following figure shows the ngrok server started with my unique host name "kangup":

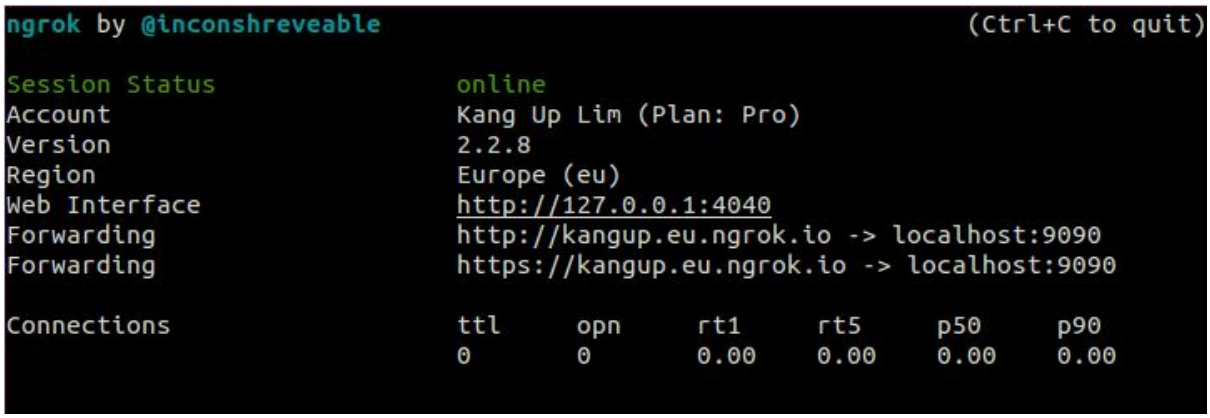


Figure 5.9 Starting Ngrok server

5.3 Web Application for Video Chat System

The developed robot controlling system provides real time video streaming, but not audio streaming. Without audio streaming the video chat is not available. Since our system assumes that the caregiver uses android device and the elderly person uses Linux laptop connected with the robot, Skype, a most popular video chat application which is available in both android device and Linux laptop, can be a good solution. But considering allowing caregivers to remotely control the robot while video chatting, it would be better we integrate video chat system in the developed application.

For integrating a video chat service in our previously developed web application above, we will first develop a simple WebRTC based video chat web application, and it will replace "camera view" html page of the previously developed web application. The application for video chat has very simple structure and minimal functions, e.g. calling and receiving a call and exchanging video and audio streaming between two clients. The WebRTC technology actually provides multiple video chat system but in our application we implement only one-on-one video chat between caregiver and an elderly person.

In this thesis we use one of these open source video chat program as basement and develop a simple video chat program which has only essential functions initiating and receiving calls.

The video chat system we develop is going to be used between a smart phone of caregiver and the robot in elderly person's home environment. The call is basically initiated by the caregiver and the elderly person accepts the call the video chat starts. Initiating call from the elderly person is not going to be implemented in this thesis.

In this thesis, in order to implement a signaling server for establishing peer to peer connection between two clients, an open source back-end API provided by Quickblox Enterprise is used. The web application is based on open source sample application that uses Quickblox JavaScript SDK [107]. The sample application provides simple WebRTC video chat service with following UI:

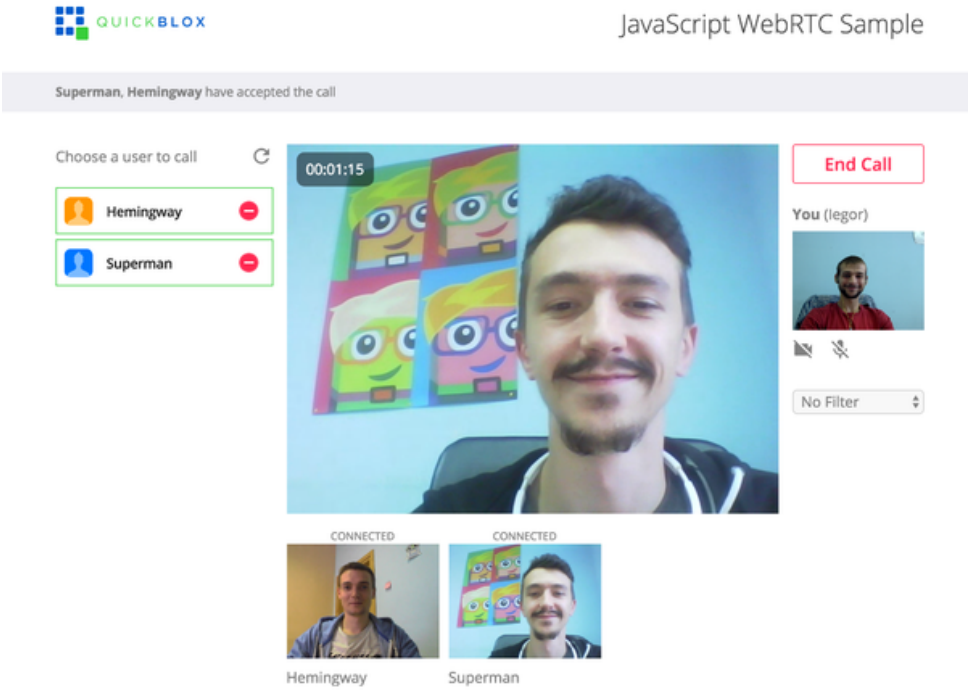


Figure 5.10 QuickBlox video chat sample

The sample application allows user to make a room with specific name. The user can make a new room or can join the room made by anyone else. When a participant enters a room and if there are other participants in the same room, they start to exchange signaling messages each other in order to begin a WebRTC session. The sample application exchanges signaling messages by using Quickblox signaling protocol with "QB.createSession" method [107].

5.3.1 Development of Video Chat Web Application

I'm not going to explain deeply about my codes of web application in this thesis because I didn't modify much from the sample codes. There are mainly two things I modified from the sample application:

- Deleted unnecessary UI and functions for making the application simple that has only minimum functions for performing one-on-one video chat.
- Assigned name of the room and user names for both caregivers and the elderly people.

The UI of the modified web application is shown in the following figure:

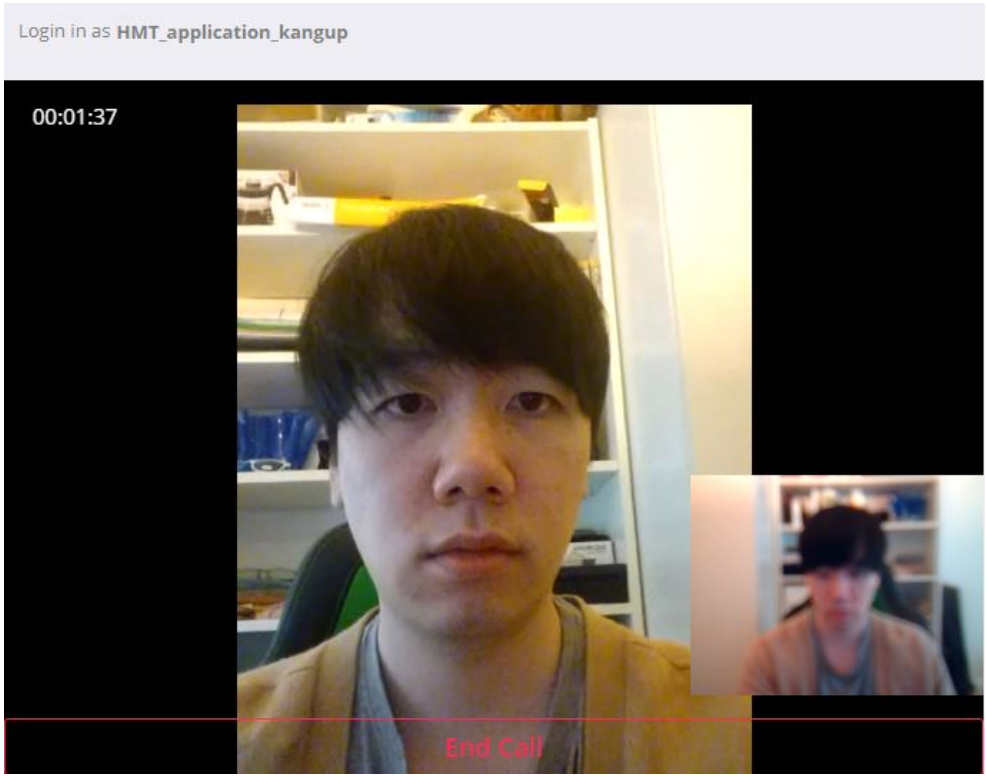


Figure 5.11 Developed video chat web application

Firstly, in the modified web application user don't need to type in both name of the room and user name when they enter the room. In this thesis the name of the room and usernames for caregivers and elderly people are assigned like [90][91]:

- Room name: HMT_room
- Caregiver's username: HMT_application_{user name of the application user}
- Elderly person 's username: HMT_robot_{last name of elderly person}
  (For example "HMT_robot_Francisco")

The assigned names for both caregivers and elderly people are stored in the database with name "videoChatName".

For evading confusion from calling to wrong person, the client doesn't have function to choose which person the user will initiate call to, but the client initiates call automatically to the elderly person, who is chosen from the list, when the user enters the room.

For performing the video chat with the developed application the two clients need to be

turned on. In this thesis one client is integrated in android application for the caregiver and the other one is web application which will be turned on in Internet browser such as Firefox or Chrome browser.

5.4 Converting Web Applications to Android Application

Now we have developed two web applications which are able to interact with and control the remote robot TurtleBot and perform video chat. The only thing remains is to convert these web applications into the android application we have developed in chapter 2. As we mentioned above we use PhoneGap software with Cordova API, which is the most well known among hybrid app platforms and the easiest way to convert web application to android application [35].

The converting is performed by using Cordova command "cordova platform add android" in the terminal, and then the converted application codes is created in the "/platforms/android" folder. For integrating the converted application into the android application we need to move the created folder in the "assets" directory of the android application directory, and the "index.html" file of the web application is loaded by "WebView" class in the android application with configurations for using native feature of the android device e.g. video camera and audio. The example 5.5 shows how the video chat web application is loaded by using "WebView" class.

```
webview = (WebView) view.findViewById(R.id.webView);
webview.setWebChromeClient(new WebChromeClient() {
      public void onPermissionRequest(final PermissionRequest request) {
            getActivity().runOnUiThread(new Runnable() {
                  public void run() {
                        request.grant(request.getResources());
                  }
            });
      }
});

WebSettings webSetting = webview.getSettings();
webSetting.setJavaScriptEnabled(true);
webSetting.setJavaScriptCanOpenWindowsAutomatically(true);
webSetting.setDomStorageEnabled(true);
webSetting.setAllowFileAccessFromFileURLs(true);
webSetting.setAllowUniversalAccessFromFileURLs(true);

webview.loadUrl("file:///android_asset/videochat/index.html");
```

Example 5.5 Loading html file in the android application

The two web applications, for video chat and for pilot, are integrated in each navigation tap of the android application:

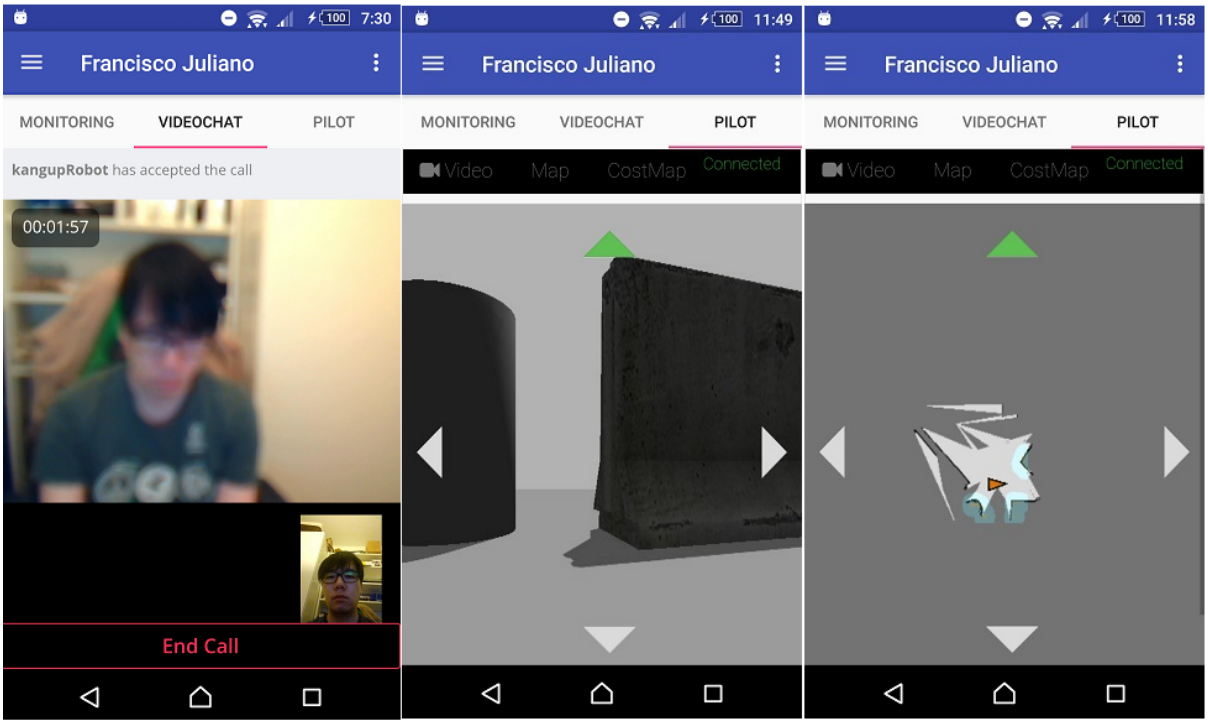


Figure 5.12 Video chat (left), ROS control with camera view (middle), and local cost map view (right)

5.4.1 Testing in Real World

To confirm that the developed telepresence system works well in the real world, the real TurtleBot was used. The test is performed by using developed android application without Wi-Fi connection. As a result the both video chat service and pilot service worked exactly the same as in the simulation without any problems.

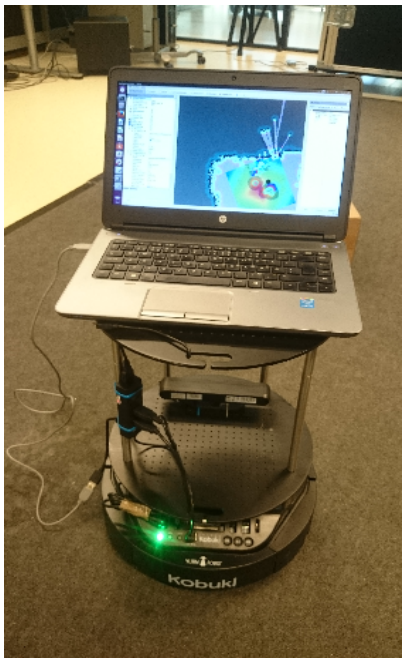


Figure 5.13 Interacting with Real TurtleBot

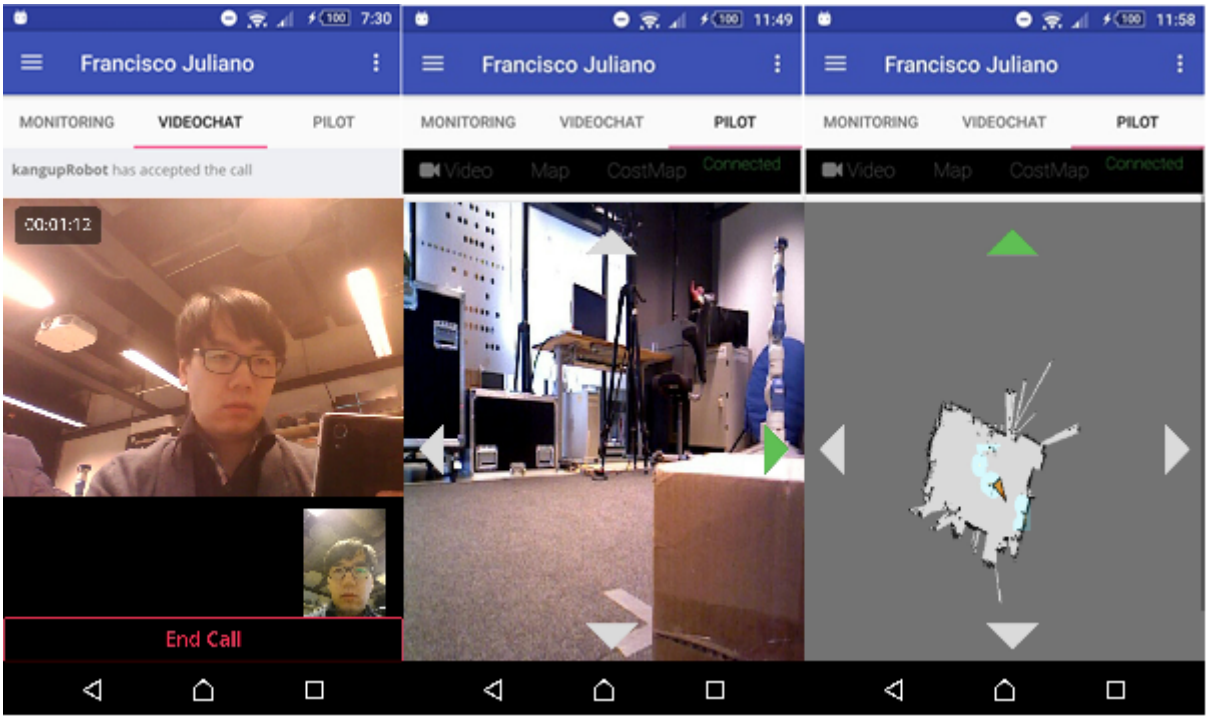


Figure 5.14 Video chat (left), Pilot with camera view (middle), and map view (right) in real world

6 Performance Testing

In this chapter the developed systems, Monitoring System and Robot Telepresence System will be tested in various ways by using several testing tools.

6.1 Methodology

As we have divided the systems into five parts, the performance testing will also be done with different methods for each divided part. The Monitoring System is divided into two parts: Server with Database and Android part, The Robot Telepresence System is divided into three parts: Pilot, Video Chat and Android part. Each part is tested by using different tools for analyzing different performance indicators. Due to the limited access to the TurtleBot robot, the performance test for Pilot system was performed in the simulation with virtual world.

In next two sections we introduce the definitions of performance indicators we are going to analyze and several performance testing tools we have used.

6.1.1 Performance Indicators

- *Latency*: The time an input reaches a system. Lower latency will simplify application development and increase web application scalability.
- *Response time*: The total time it takes from when a user makes a request until the user receives a response.
- *Memory usage*: A term used to describe how much memory is used in the android device for running the application.
- *CPU usage*: The percentage of CPU usage indicates how much of the processor's capacity is currently in use by the system.
- *Network traffic*: The amount of data moving across a network at a given point of time. Network traffic is also known as data traffic.

6.1.2 Performance Testing Tools

- *Apache JMeter* - An open source software, may be used to test performance both on static and dynamic resources, Web dynamic applications. This tool is used to test our

database server with "load testing" which simulates increasing number of concurrent users and transactions on a server and check the behavior of the server. By this test we can evaluate server's latency, response time and data transaction under a heavy load.

- *Android Device Monitor* - A standalone tool that is developed by Google and is installed together with Android Studio. It can be launched when the Android device is connected with desktop/laptop and the app is run by Android Studio. The tool helps us to profile the performance of our app by measuring and visualizing real time data such as memory, CPU, network and GPU usage of the Android device.

- *Chrome DevTools* - A set of debugging tools built into Google Chrome. The tool is used to test web applications we developed which can be run in the Chrome browser. The tool has a "Network" panel that provides insights into resources that are requested and downloaded over the network in real time, and "Performance"

- *Battery Historian* - An open source tool developed by Google. With the tool the user can receive a report that provides insight into a device's battery consumption over time. The tool is going to be used for analyzing battery consumption of the developed application.

6.2 Testing on Monitoring System

6.2.1 Load Testing on Database Server

The testing on Database Server is done by using Apache JMeter. The testing method is a load test that we make an use case scenario that sending HTTP Requests to the server and receiving responses and perform the test by a number of users. The tool allows us to make a thread group that can adjust number of users, loop count and duration of the test. As we can see in the following figure, we set up that 20 users run the scenario as many times as possible in 60 seconds.

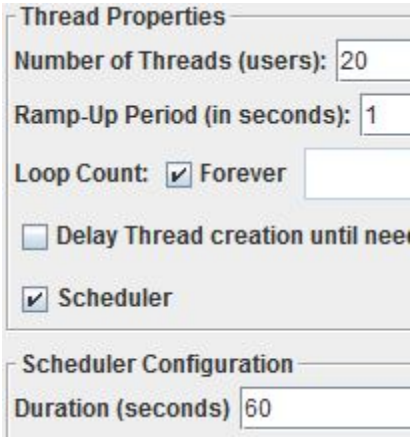


Figure 6.1 Configuration of the load test

The scenario of the test is basic use case of the monitoring service which consists of 6 steps :

1. Login with username and password.
   Request URL: {baseURL}/authenticateUser/{username}/{password}
2. Choose one elderly person in the list and receive measurement setting for making graph.
   Request URL: {baseURL}/getMeasurementsSetting
3. Request specific data of measurement of the elderly person for a certain period of time.
   Request URL: {baseURL}/getMeasurementsByPeriod/{group}/{date}/{periodType}
4. Change measurement type or period and request again.
   Request URL: {baseURL}/getMeasurementsByPeriod/{group}/{date}/{periodType}
5. Set reminder service at the current time.
   Request URL: {baseURL}/setReminder
6. Get up-to-the-minute reminder message
   Request URL: {baseURL}/getReminder

As the result we got 17135 samples in 60 seconds, and it means there were average 285 requests sent to the server every second. The summary of the tests is shown in the following figure:

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	17135	69	51	1377	38.77	0.00%	285.2/sec	143.84	113.72	516.4
TOTAL	17135	69	51	1377	38.77	0.00%	285.2/sec	143.84	113.72	516.4

Figure 6.2 Summary of the samples

The result shows that the average latency is low enough with 69ms. The minimum and maximum latency are 51ms and 1377ms respectively, but as we can see in the next figure (see figure 6.3), only 20 samples have over 800ms latency and the samples with high latency are run at the almost same time period, about 14:07:52 ~ 14:07:53. So we can infer that there was a "server lag" at that period.

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(ms)
15010	14:07:52.768	Thread Group 1-15	HTTP Request	1377		409	450	1377	0
14997	14:07:52.772	Thread Group 1-14	HTTP Request	1358		409	450	1358	0
14988	14:07:52.682	Thread Group 1-12	HTTP Request	981		409	450	981	0
14996	14:07:52.742	Thread Group 1-20	HTTP Request	968		510	383	968	0
14994	14:07:52.742	Thread Group 1-10	HTTP Request	963		491	383	963	0
14995	14:07:52.768	Thread Group 1-18	HTTP Request	942		510	383	942	0
14993	14:07:52.743	Thread Group 1-6	HTTP Request	933		510	383	933	0
14985	14:07:52.734	Thread Group 1-13	HTTP Request	927		465	413	927	0
14992	14:07:52.749	Thread Group 1-4	HTTP Request	920		460	413	920	0
14984	14:07:52.745	Thread Group 1-2	HTTP Request	913		483	413	913	0
14990	14:07:52.752	Thread Group 1-8	HTTP Request	913		465	413	913	0
14979	14:07:52.735	Thread Group 1-17	HTTP Request	912		472	413	912	0
14989	14:07:52.753	Thread Group 1-3	HTTP Request	911		460	413	911	0
14986	14:07:52.755	Thread Group 1-19	HTTP Request	907		465	413	907	0
14987	14:07:52.761	Thread Group 1-7	HTTP Request	902		460	413	902	0
14991	14:07:52.763	Thread Group 1-5	HTTP Request	902		460	413	902	0
14983	14:07:52.756	Thread Group 1-11	HTTP Request	895		460	413	895	0
14980	14:07:52.773	Thread Group 1-16	HTTP Request	874		630	392	874	0
14981	14:07:52.776	Thread Group 1-9	HTTP Request	872		648	392	872	0
14982	14:07:52.780	Thread Group 1-1	HTTP Request	869		648	392	868	0
15013	14:07:53.659	Thread Group 1-2	HTTP Request	519		409	450	519	49
15014	14:07:53.661	Thread Group 1-13	HTTP Request	518		409	450	518	53
15015	14:07:53.662	Thread Group 1-19	HTTP Request	517		409	450	517	0
15016	14:07:53.665	Thread Group 1-8	HTTP Request	514		409	450	514	52
15011	14:07:53.648	Thread Group 1-9	HTTP Request	498		668	394	498	51
15007	14:07:53.647	Thread Group 1-16	HTTP Request	495		668	394	495	0
15008	14:07:53.647	Thread Group 1-17	HTTP Request	495		465	413	495	0
15012	14:07:53.663	Thread Group 1-12	HTTP Request	494		510	383	494	51
15009	14:07:53.651	Thread Group 1-11	HTTP Request	491		472	413	491	55
15002	14:07:53.649	Thread Group 1-1	HTTP Request	490		668	394	490	56
15006	14:07:53.664	Thread Group 1-3	HTTP Request	478		472	413	478	53
15004	14:07:53.663	Thread Group 1-7	HTTP Request	477		472	413	477	51
15003	14:07:53.665	Thread Group 1-5	HTTP Request	474		472	413	474	54
15001	14:07:53.669	Thread Group 1-4	HTTP Request	469		472	413	469	51

Scroll automatically? Child samples? No of Samples 17135 Latest Sample 157 Average 69 Deviation 38

Figure 6.3 Detail information about each samples

The following graph is showing response time of the samples over time. Most of the values are around 60ms~70ms, but we can see the graph rises at point 14:07:50, when the server lag occurred.

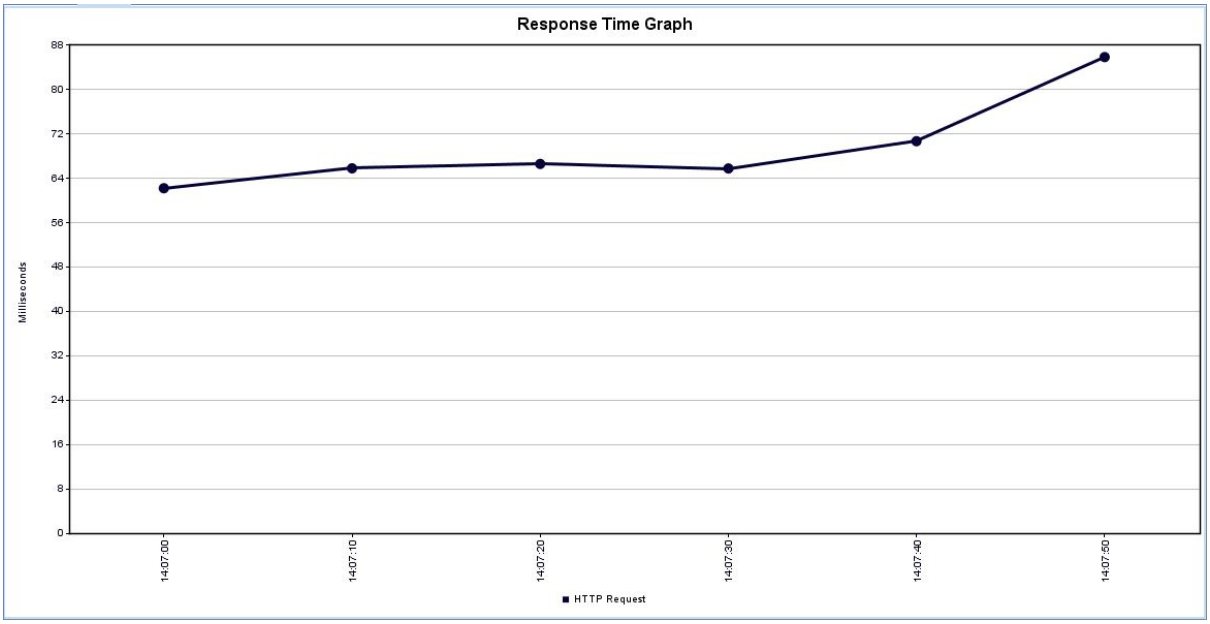


Figure 6.4 Response Time Graph of Load Test

The results of the load test indicate that the performance of the database server is fine enough to be used. There were some points that occurred server lag, but the average latency and the response time were low enough, and it shows the server has potential to tolerate more than 200 users simultaneously.

6.2.2 Monitoring System in Android Application

The following figure is Android Device Monitor that shows memory, CPU and Network usage of the Android device when we use Health Monitoring System in the application.

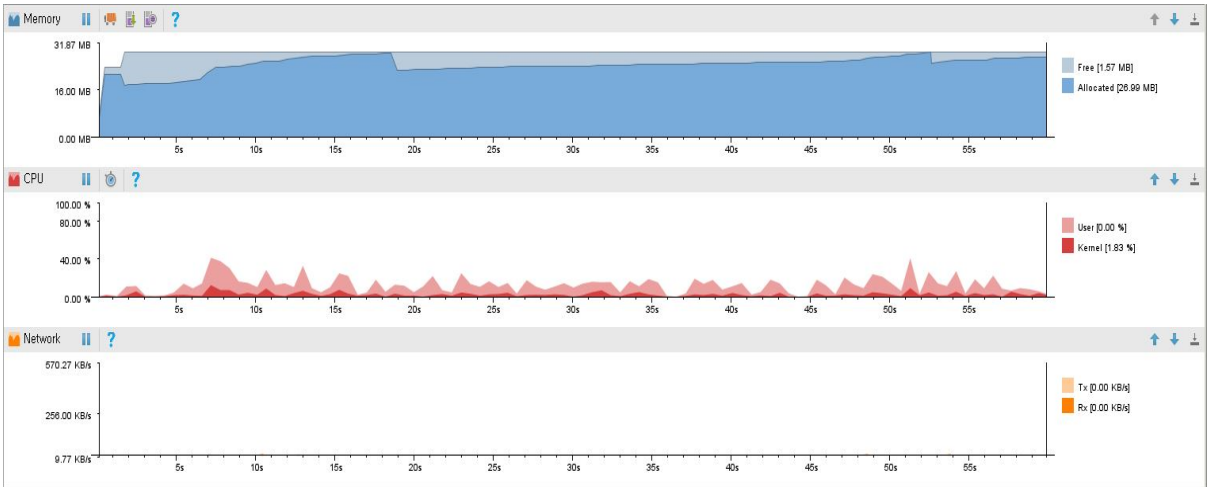


Figure 6.5 Performance for Monitoring System Analyzed by Android Device Monitor

In the graph of Memory usage, the y-axis displays the free and allocated RAM in megabytes and the x-axis shows the time elapsed. The amount of free memory, measured in megabytes, is shown in a light color, and allocated memory is a darker color. When there's a sharp drop in allocated memory that indicates a garbage collection event. The memory usage of our application was around 20 MB at first and there was some fluctuation in the graph, but the average memory usage was around 25MB.

The graph of CPU Monitor displays the CPU usage in real time used in the y-axis. The value is the percentage of total CPU time used in user mode and kernel mode. In user mode, the code must use system APIs to access hardware or memory, has access to virtual memory addresses only, and crashes are usually recoverable. In kernel mode, the code can directly access hardware, including physical memory addresses and crashes halt the device. In the graph there are small peaks that are occurred when the application sends HTTP request or receive responses. The average values are around 10~20% with the most of in user mode.

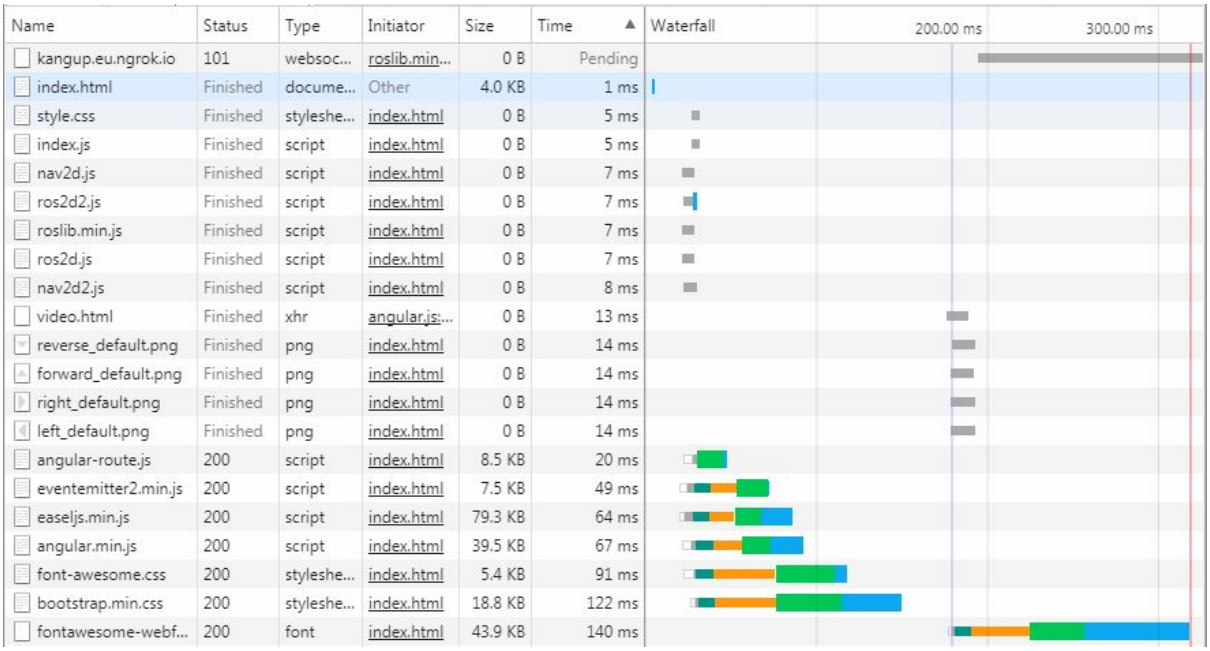
The Network Monitor tracks when our application is making network requests. Using this tool, we can monitor how much data is transferred when the application communicate with the database server. It shows the amount of time it takes for the device to transmit and receive kilobytes of data in x-axis, and the y-axis is in kilobytes per second. But as we saw in figure 6.3, the average bytes of transferred data is about 500bytes, and it's so small that we can't see any peak in the graph.

6.3 Testing on Telepresence System

6.3.1 Performance of Pilot Web Applications

The Telepresence system that has been developed is a hybrid application. So before we test the system in the android device, we can use a Google Chrome Devtool to analyze details about network traffic when we start the video chat and pilot systems. With this tool we can analyze what kinds of network transactions occurred when we use the application. In the Network panel we can see detailed information of each transaction the application performed as the “Time” shows how long time is spent by each transaction and the "size" shows the size of the transferred data. However, the speed of data transferring depends on the network performance of the desktop/laptop or mobile device, so it can vary from the results we are going to see. The testing is performed by two laptops with different Wi-Fi connections for making more real world situations.

The figure 6.6 shows transactions from when the application started to when the application finished with rendering the page. Here we can see those transactions is performed to load libraries and some image files. The times spent by the most of transactions are less than 50ms, and only two transactions spent time more than 100ms. The time spent by all the transactions for loading web page is less than 1 second as about 650ms.



Name	Status	Type	Initiator	Size	Time	Waterfall
kangup.eu.ngrok.io	101	websoc...	roslib.min...	0 B	Pending	
index.html	Finished	docume...	Other	4.0 KB	1 ms	
style.css	Finished	styleshe...	index.html	0 B	5 ms	
index.js	Finished	script	index.html	0 B	5 ms	
nav2d.js	Finished	script	index.html	0 B	7 ms	
ros2d2.js	Finished	script	index.html	0 B	7 ms	
roslib.min.js	Finished	script	index.html	0 B	7 ms	
ros2d.js	Finished	script	index.html	0 B	7 ms	
nav2d2.js	Finished	script	index.html	0 B	8 ms	
video.html	Finished	xhr	angular.js:...	0 B	13 ms	
reverse_default.png	Finished	png	index.html	0 B	14 ms	
forward_default.png	Finished	png	index.html	0 B	14 ms	
right_default.png	Finished	png	index.html	0 B	14 ms	
left_default.png	Finished	png	index.html	0 B	14 ms	
angular-route.js	200	script	index.html	8.5 KB	20 ms	
eventemitter2.min.js	200	script	index.html	7.5 KB	49 ms	
easeljs.min.js	200	script	index.html	79.3 KB	64 ms	
angular.min.js	200	script	index.html	39.5 KB	67 ms	
font-awesome.css	200	styleshe...	index.html	5.4 KB	91 ms	
bootstrap.min.css	200	styleshe...	index.html	18.8 KB	122 ms	
fontawesome-webf...	200	font	index.html	43.9 KB	140 ms	

Figure 6.6 Information of Transferred Data while Starting Pilot Web Application

The next two figures shows data transferring occurring when we use pilot system with camera view and map image respectively. The time spent by receiving data is very short as less than 1ms. The data for camera view is transferred continuously more than one time in every second, while the data for map image is transferred only when they were updated.

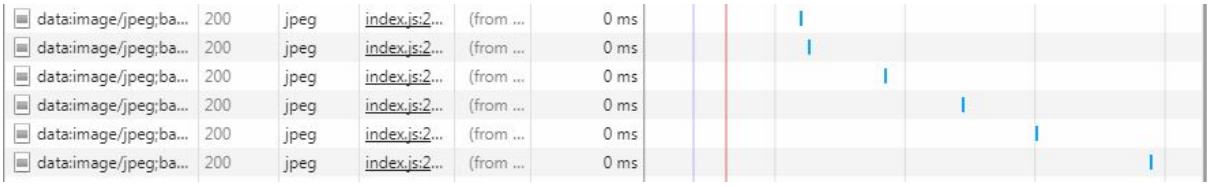


Figure 6.7 Transferred Camera View Data From TurtleBot

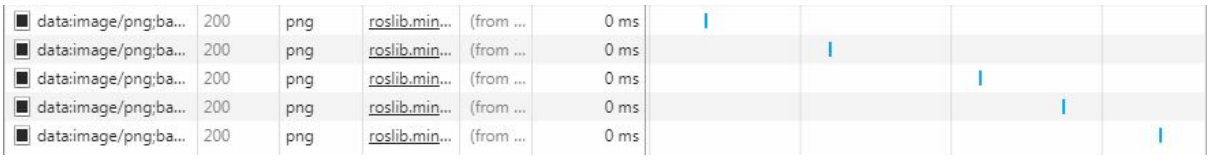


Figure 6.8 Transferred Map Data From TurtleBot

6.3.2 Performance of Video Chat Web Application

The video chat web application is also tested by the same method with the pilot web application. The figure 6.9 shows data transactions occurred when the web page is loaded. Like the pilot system the transactions were performed for using JavaScript libraries and CSS file and the time spent for rendering web page is shorter than pilot web application as 300ms.

Figure 6.9 Information of Transferred Data while Starting Video Chat Web Application

The figure 6.10 shows the data transactions occurring when the application tries to enter a room with certain room name and certain username. As we can see there were three transactions. The first one is making session for establishing peer to peer connection between two clients by using signaling server. The second and third one are transactions for making user data and login with that user data. Those three transactions spent relatively more time than other parts but it is hard to make them more faster since they are performed by the back end API QuickBlox.

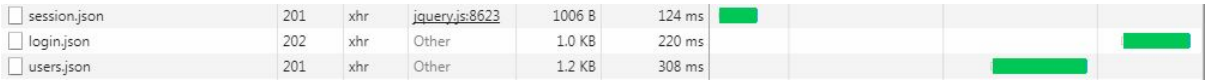


Figure 6.10 Information of Transferred Data while Establishing p2p connection

6.3.3 Telepresence System in Android Application

The Telepresence system is also tested in Android Device Monitor for analyzing memory, CPU, Network and GPU usage of the Android device. The figure 6.11 shows the result of the test for pilot system. As like the monitoring system the memory usage is between 20~30 MB, and there was a garbage collection. CPU usage was higher than monitoring system as up to 40%. In the Network traffic panel there were many peaks that shows data the application received continuously from TurtleBot.

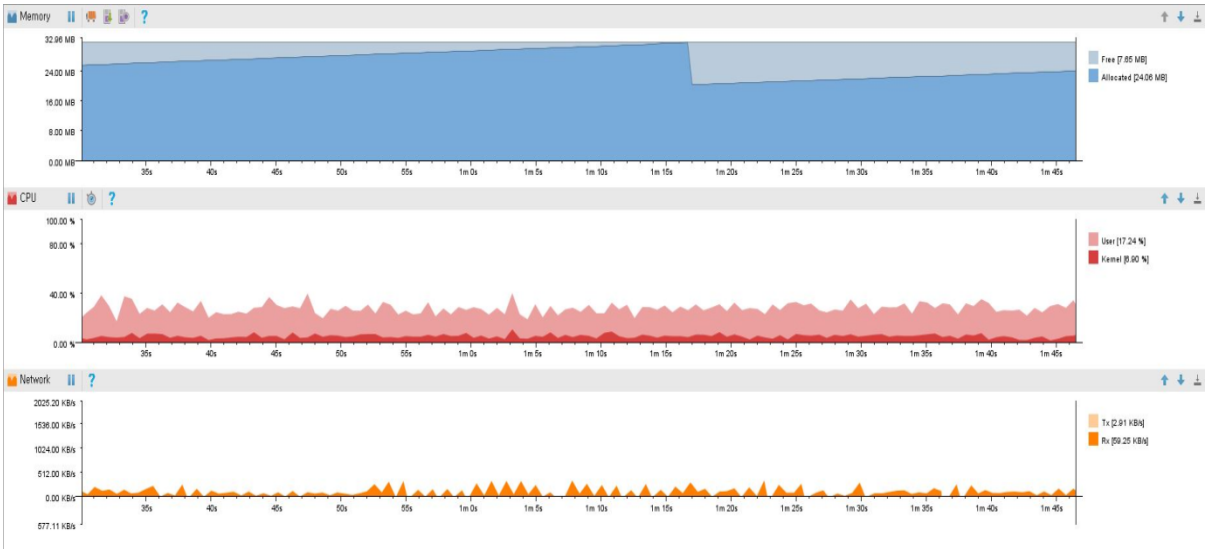


Figure 6.11 Performance for Pilot System Analyzed by Android Device Monitor

The figure 6.12 shows the result of the test for video chat system. The thing that is beneath our notice is that the CPU usage was higher than other systems as up to 50 %. In addition, as the network traffic there are both receiving and sending data as 100~300 KB/s. From these reasons we can infer that the video chat system uses more resources of the android device than other systems.

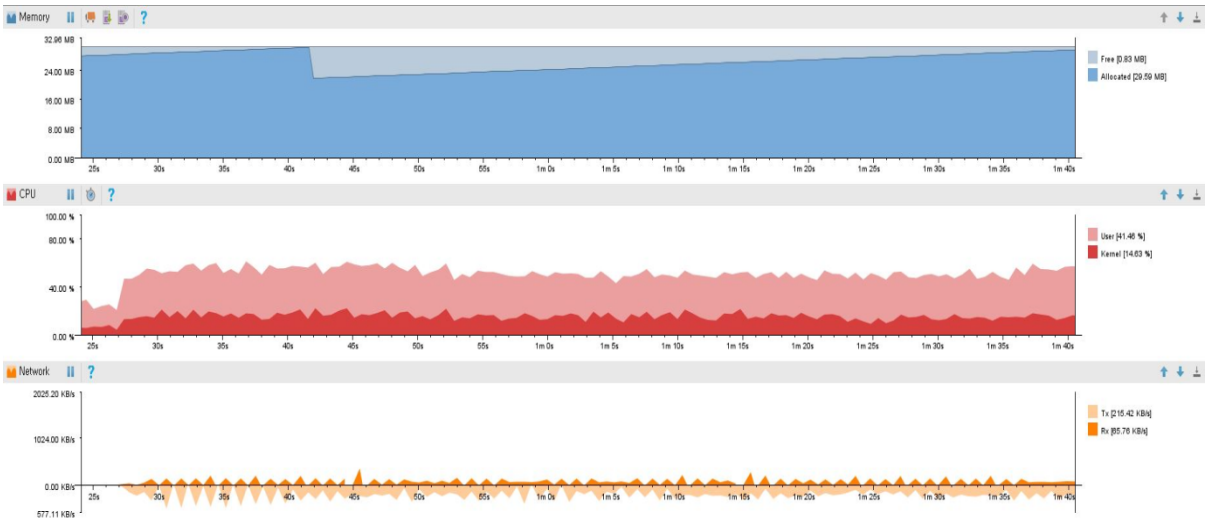


Figure 6.12 Performance for Video Chat System Analyzed by Android Device Monitor

6.4 Battery Usage of the Application

"Battery Historian" is a tool provided by Google, which enables the user evaluate battery usage of the android device in with time line. In order to test the battery usage I used each system in 1 hour. The black line in the following figure shows all the battery usage in 3 hours and where the battery is used for. The x-axis shows the time and the y-axis shows how the battery is used. For comparing with the other application, I used "Facebook" application in 1 hour, and the result is shown in the figure 6.14, the test was from 8 pm to 9 pm and battery consumption was about 10%.

My test was performed from 7 pm to 10 pm as we can see in the figure 6.14. From 7 pm to 8 pm the monitoring system was running in foreground of the device and I used some functions intermittently such as requesting other type of physiological data and reminding service. The battery usage was low as 2%.

From 8 pm to 9 pm, the video chat system was used as performed video chat with another

client. The battery usage was unexpectedly high as about 40%. This high battery usage can cause a problem for using the application in the real world.

From 9 pm to 10 pm, the pilot system was running in foreground. Since we know that the camera view uses relatively more resources than the map view, the test was performed with camera view. During the test, the robot was controlled by virtual controller intermittently. The test used about 10% of battery and it is low enough.

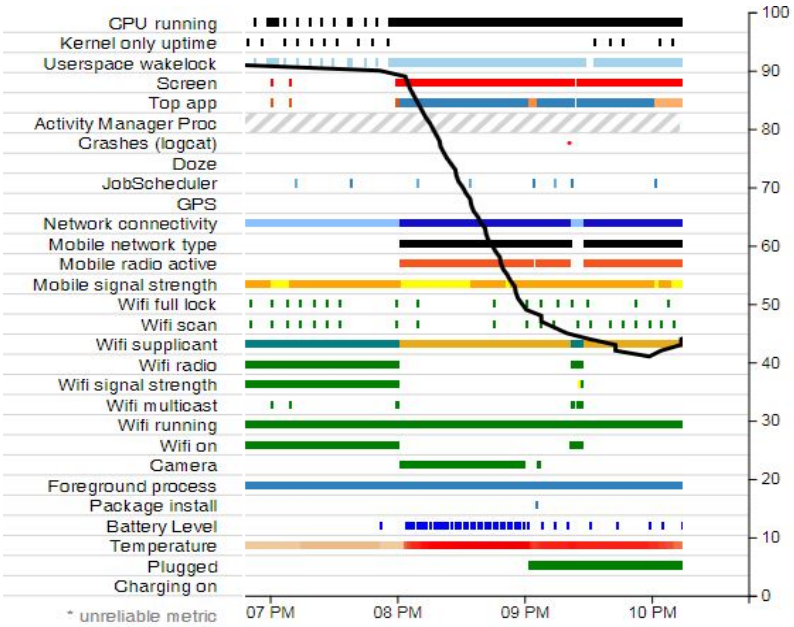


Figure 6.13 Battery usage for using the HMT application

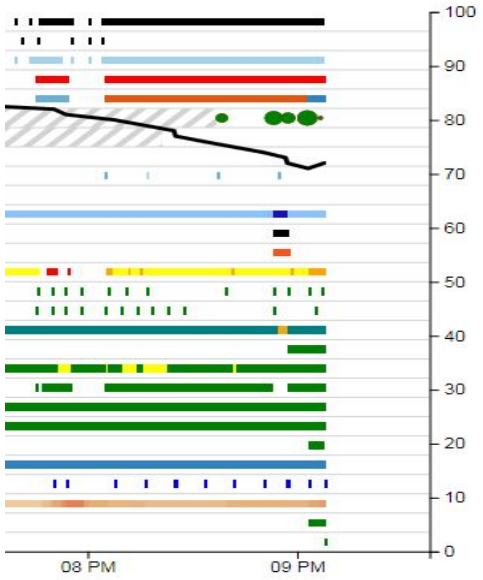


Figure 6.14 Battery usage for using "Facebook" application

Conclusions and Future Works

7 Conclusions

This thesis work intended to develop an android application for allowing caregivers to monitor elderly person's physiological data and remotely control remote robot "Turtlebot" for visiting patient virtually and performing video chat with elderly person. The fundamental idea is from "GiraffPlus" project and is designed to have mainly two systems: Health Monitoring System and Robot Telepresence System.

Health Monitoring System was developed as combination of 3 parts: server part, database part and android application. In this thesis we didn't develop a new sensor network, but we assumed the system use the same sensor network as the GiraffPlus project. The database server was developed as web application server (WAS) by using Apache Tomcat software and is run by Eclipse. The database was created by using MongoDB and connected with the server by using MongoTemplate API. Communication between android application and the server is performed by using REST API as the application can send and receive specific data to/from the server.

Robot Controlling System is composed of pilot system, video chat system and android application. The pilot system and video chat system was developed as web application and integrated into the android application. For the pilot system the application and "TurtleBot" is connected through "rosbridge" server and the robot is remotely controlled by using virtual controller. In order to teleoperating the robot, the application receives camera view, map image and local costmap image from TurtleBot. In costmap view the map image and the local costmap image is overlapped to allow the user navigate the robot more easily by seeing obstacles around the robot. The video chat system is developed by using WebRTC technology which provides peer to peer connection between two clients: android application for caregiver and web application for elderly person. An open source back-end API QuickBlox is used to implement signaling server for establishing peer to peer connection.

The developed systems were tested and analyzed in various aspects by using several testing

tools. The results shows that the developed application and systems have relatively high performance, but the one thing we have to concern is that the high resource consumption of the video chat service, in particular, the battery usage and network traffic were unexpectedly high.

8 Future Works

The android application and systems developed in this thesis still have a lot of potential to be improved by future works. In this chapter, I describe the remained problems of my systems and possible projects that can build upon the work I have done.

8.1 Usability Testing

The testing usability is very important for developing new software. By usability testing we can assess how easy user interfaces are to use. The usability of the application is not considered in this thesis, so the low fidelity UI of the developed application need to improved. By watching people trying to use the application in the real world, from the small things such as the positions of buttons or non-critical errors, to some critical errors that interrupt normal use or halt the application can be detected.

8.2 Improvement of the Telepresence System

The one of things that is needed to be improved for making more high fidelity application is the Telepresence system. The problem is that the two main components of the system, Pilot service and Video Chat service are working independently of each other. The biggest reason that the two services are developed separately is that they use different network connection from each other as the Pilot service uses "rosbridge" server and the Video Chat service uses WebRTC connection. Using those services separately causes some problems in the system, for example the video chat service can't be used before a call initiated by the application is accepted by the elderly person, while the pilot service can be used without any permission to control the robot. Actually combining two services into one system is not so difficult, as the

pilot service is not available before the video chat service establish connection with the elderly person, but in that case the system becomes too complicated and using both services at the same time may cause the application uses unnecessary resources. In the future work, it is necessary to find an efficient way to combine these two systems.

8.3 Android Application for Elderly User

The system we designed in this thesis using android application and TurtleBot robot has still great potential to support elderly people living alone in their home environment. As an extension of our project, an android application designed to be used by elderly people with pilot system we developed can be developed. In addition, there are many open source project that use TurtleBot and ROS as an assistive robot. A typical project is "CoffeBot" project by "markwsilliman"[100], which autonomously finds way to the user and makes coffee whenever the user gives an order by using a web application and this system can enough be integrated into the android application.

The problem is that the elderly people need to have their own smart phone for using those applications. But as the figure 8.1 shows, about 40% of elderly people ages 65 and up have their smart phone and it is more than doubled in the past five years. Smartphone adoption among the elderly people is rising rapidly and it is becoming more and more common that elderly people using their own smart phones.

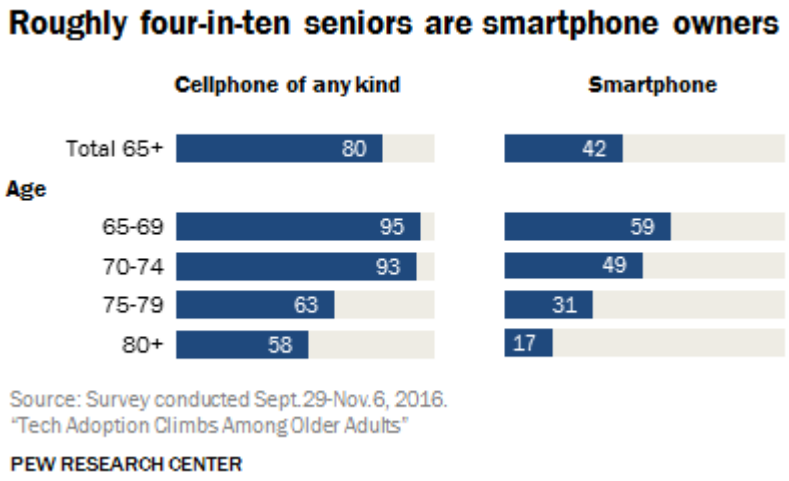


Figure 8.1 Percentage of U.S. adults ages 65 and older who own smart phone [99]

8.3.1 Collecting Physiological Data by Using Android Device

As a part of the android application for elderly user, collecting physiological data by using android device can be considered.

There are two ways to use android phone in monitoring system: i) Using wearable sensors with wireless data communication capability which is incorporated with the android phone to collect current user's health status [95], ii) Measuring physiological data by using android application.

The first solution is kind of "The Personal Mobile Hub (PMH)". An example of a wearable sensor can be an Electrocardiography (ECG) sensor to determine instant heart beat rate (HBR). Measurements of body temperature and current geographical location of the patient can be other valuable sources for remote diagnosis. Therefore, real-time information processing is highly desirable for remote health monitoring. Also, the Internet connection provides the flexibility and allows mobility in transferring the collected data to a central database [95]. But equipping wearable sensors is not permanent, and the elderly people may need at least one helper to use those sensors. So this solution is not suitable for our circumstance which aims for monitoring elderly people who are living alone at home.

For the second solution, there are numerous android application developed that have functions to measure user's physiological parameters. Some examples are shown in the figure 8.2. It includes user's heart rate, blood pressure and blood oxygen level. The principle to measure those parameters are for example pressing the screen with our finger, or place our finger over the LED flash on the back of android phone in a few seconds [96]. It is very low cost(almost free and open-sources), and can also be integrated in our developed application.

Figure 8.2 Examples of health-measuring android application

iCare Health Monitor Pro (left), Instant Heart Rate (middle) and Runatic Heart Rate (right)

However, it may still be burden for elderly user using those health-measuring applications because the measurements is not automatic, and it means we need to make the elderly user follow the instruction for measuring data at a certain time everyday for proper monitoring. As a possible solution, the reminder system which let the elderly user know when they take measurement can be helped.

References

1. Heerink, M., Krose, B., Wielinga, B., Evers, V., Human-Robot User Studies in Eldercare: Lessons Learned, 2006, p. 1.
2. Qiu, M., Dai, W., Gai, K., Mobile Applications Development with Android Technologies and Algorithms, 2017, p. 6.
3. Amalie F.A. Fadzlah., Readiness Measurement Model (RMM): Mathematical-Based Evaluation Technique for the Quantification of Knowledge Acquisition, Individual Understanding, and Interface Acceptance Dimensions of Software Applications on Handheld Devices, 2016, p. 2.
4. Turtlebot 2., URL: http://www.turtlebot.com/turtlebot2/
5. GiraffPlus project., URL: http://www.giraffplus.eu/
6. Broekens, J., Heerink, M., Rosendal, H., Assistive social robots in elderly care: a review, 2009, pp. 1-2.
7. Sabelli, A. M., Kanda, T., Hagita, N., A Conversational Robot in an Elderly Care Center: and Ethnographic Study, 2011, pp. 1-2.
8. Villarreal, J. J., Ljungblad, S., Experience Centred Design for a Robotic Eating Aid, 2011, p. 1.
9. Wynsberghe, A. V., Healthcare Robots: Ethics, Design and Implementation, 2014, p. 75.
10. Pollack, Martha E. et al., Pearl: A Mobile Robotic Assistant for the Elderly, 2002.
11. Giraff., URL: http://www.giraff.org/?lang=en
12. Costa, A., Julian, V., Novais, P., Personal Assistants: Emerging Computational Technologies, 2017, p. 79.
13. Wada, K., Shibata, T., Living with Seal Robots-Its Sociopsychological and Physiological Influences on the Elderly at a Care House, 2007, pp. 972-980.
14. Wada, K., Shibata, T., Social and Physiological Influences of Living with Seal Robots in an Elderly Care House for Two Months, 2008.
15. Stiehl, W.D., et al. The huggable: a therapeutic robotic companion for relational, affective touch, 2006.
16. Fujita, M., AIBO: Toward the Era of Digital Creatures, 2001.
17. Plaumbo, F., Ullberg, J., et al., Sensor Network Infrastructure for a Home Care Monitoring System, 2014, p. 3834.

18. Frennert, S. A., Forsberg, A., Östlund, B., Elderly people's perceptions of a telehealthcare system: Relative advantage, compability, complexity and observability, 2013.
19. Torbjørn S. D., Maged N. K. B., Robots in Health and Social Care: A Complementary Technology to Home Care and Telehealthcare?, 2013, p. 3.
20. Coradeschi, S., et al., GiraffPlus: Combining social interaction and long term monitoring for promoting independent living, 2013, pp. 1-2.
21. GiraffPlus project., URL: http://www.giraffplus.eu/
22. Cesta, A., Cortellessa, G., Tiberio, L., Long-term Evaluation of a Mobile Remote Presence Robot for the Elderly, 2011.
23. J. Gonzalez-Jimenez., et al., Technical Improvements of the Giraff Telepresence Robot based on Users' Evaluation, 2012, pp. 1-2.
24. Maria Linden., et al., GiraffPlus Project: D1.2 Technological Component Specifications, 2014, pp. 15-39.
25. Amedeo Cesta., et al., GiraffPlus Project: D4.1 The Interaction and Visualization Service and Personalization Module – Alpha Release, 2014, pp. 8-10.
26. Silvia Coradeschi., et al., GiraffPlus Project: D1.3 System Reference Architecture, 2014, pp. 9, 65, 69-72
27. Spring Framework., URL: https://spring.io/
28. MongoDB., https://www.mongodb.com/
29. GiraffPlus project: Pilot 2.0 User Guide., pp. 7-14.
30. What is application? - Definition from WhatIs.com., URL: http://searchsoftwarequality.techtarget.com/definition/application
31. What is Mobile Application? - Definition from Techopedia., URL: https://www.techopedia.com/definition/2953/mobile-application-mobile-app
32. IDC: Smartphone OS Market Share., URL: https://www.idc.com/promo/smartphone-market-share/os
33. What is Android? Applications in Android, Android Mobile App Development - Arokia IT., URL: http://www.arokiait.com/whatis-android.htm
34. Jobe, W., Native Apps vs. Mobile Web Apps, 2013, pp. 27-32
35. Web, Hybrid Or Native Apps? What's The Difference?., URL: https://www.mobiloud.com/blog/web-hybrid-native-apps/
36. Web app vs. Native app - App Press., URL: https://www.app-press.com/blog/web-app-

vs-native-app
37. Types of Mobile Applications., URL: https://www.linkedin.com/pulse/types-mobile-applications-kaushik-bhatti-
38. What's the difference between a web site and a web application? - Stack Overflow., URL: https://stackoverflow.com/questions/8694922/whats-the-difference-between-a-web-site-and-a-web-application
39. What is a web application? - 3Squared Ltd., URL: https://www.3squared.com/2011/02/what-is-a-web-application/
40. Mobile: Native Apps, Web Apps, and Hybrid Apps., URL: https://www.nngroup.com/articles/mobile-native-apps/
41. Mobile apps vs. PB desktop apps., URL: https://www.appeon.com/support/documents/appeon_online_help/2.0/development_guidelines/ch02s01.html
42. Touch vs. Mouse - The Hipper Element., URL: http://thehipperelement.com/post/47937273876/touch-vs-mouse
43. Mobile vs. Desktop: 10 Key Differences - ParadoxLabs., URL: https://www.paradoxlabs.com/blog/mobile-vs-desktop-10-key-differences/
44. 10 Ways the Mobile Web is Different., URL: https://www.elated.com/articles/10-ways-the-mobile-web-is-different/
45. Spring Framework., URL: https://spring.io/
46. Apache Tomcat., URL: http://tomcat.apache.org/
47. WebRTC Home., URL: https://webrtc.org/
48. Apache Cordova., URL: https://cordova.apache.org/
49. Getting Started with WebRTC - HTML5 Rocks., URL: https://www.html5rocks.com/ko/tutorials/webrtc/basics/
50. ROS.org - Powering the world's robots., URL: http://www.ros.org/
51. Jakobsson, C., Peer-to-peer communication in web browsers using WebRTC, 2015.
52. rosbridge_suite - ROS Wiki., URL: http://wiki.ros.org/rosbridge_suite
53. Johnson, R., et al., Spring java/j2ee Application Framework, 2004.
54. The Spring Web MVC Framework - TechTarget., URL: http://media.techtarget.com/tss/static/articles/content/AgileJavaDev/SpringMVC.pdf
55. What is Web server? - Definition from WhatIs.com., URL:

http://whatis.techtarget.com/definition/Web-server
56. App server, Web server: what's the difference? - JavaWorld., URL: https://www.javaworld.com/article/2077354/learn-java/app-server-web-server-what-s-the-difference.html
57. Srivastava, A., Bhargava, A., Understanding Application Servers, 2003, pp. 1-4.
58. DR. M. Hoche., Structure Capital, Assessment, Compliance and More, 2016, p. 100.
59. Most popular Java application servers: 2017 edition, URL: https://plumbr.eu/blog/java/most-popular-java-application-servers-2017-edition
60. Press, W., Tomcat for beginning Web developers, 2005, p. 1.
61. Dedoimedo., Apache Web server Complete Guide, p.11.
62. Introduction to Spring Framework., URL: https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/overview.html
63. Fredrich, T., RESTful Service Best Practices., 2012.
64. Richardson, L., Ruby, S., RESTful Web Services., 2007.
65. Stateless Spring Security on REST API - Complex to Simple., URL: https://malalanayake.wordpress.com/2014/06/30/stateless-spring-security-on-rest-api/
66. Alex, B., Taylor, L., Winch, R., Hillert, G., Spring Security Reference, 2015
67. JSON Web Tokens - jwt.io., URL: https://jwt.io/
68. JSON Web Token Tutorial: An Example in Laravel and AngularJS., URL: https://www.toptal.com/web/cookie-free-authentication-with-json-web-tokens-an-example-in-laravel-and-angularjs
69. Sapthami, R., Anisha, P. Rodrigues., Study on Handling Semi-structured Data using NoSQL Database, 2016.
70. Palumbo, F., et al., GiraffPlus project: D2.1 First Prototype of sensors, Giraff platform and network system report, 2014.
71. Palumbo, F., et al., GiraffPlus project: D2.2 Second Prototype of sensors, Giraff platform and network system, 2014.
72. Spring Data MongoDB., URL: https://docs.spring.io/spring-data/mongodb/docs/current/reference/html/
73. Spring 4 MongoDB Example (MongoTemplate CRUD)., URL: http://www.technicalkeeda.com/spring-tutorials/spring-4-mongodb-example
74. Introduction to Spring Data MongoDB - Daeldung., URL:

http://www.baeldung.com/spring-data-mongodb-tutorial
75. Thambawita, V., et al., Low Cost Telepresence Robot, 2011, p. 2.
76. Cesta, A., et al., Enabling Social Interaction Through Embodiment in Excite, 2013, pp. 1-3.
77. Foustanas, N., Helping Elderly Users control a Telepresence Robot With a Touch Screen, 2015, pp. 4-14.
78. TurtleBot Archives - JetsonHacks., URL: http://www.jetsonhacks.com/tag/turtlebot/
79. Jose Manuel Ferrandez Vicente., Natural and Artificial Models in Computation and Biology, 2013, p. 142.
80. Quigley, M., ROS: and open-source Robot Operating System.
81. Martinez, A., Fernandez, E., Learning ROS for Robotics Programming, 2013, pp. 8-13.
82. Jason M. O'Kane., A Gentil Introduction to ROS, 2014.
83. Sam Dutton, WebRTC in the real world: STUN, TURN and signaling, 2013, URL: https://www.html5rocks.com/en/tutorials/webrtc/infrastructure/
84. roslibjs - ROS Wiki., URL: http://wiki.ros.org/roslibjs
85. ros2djs - ROS Wiki., URL: http://wiki.ros.org/ros2djs
86. nav2djs - ROS Wiki., URL: http://wiki.ros.org/nav2djs
87. rviz - ROS Wiki., URL: http://wiki.ros.org/rviz
88. gmapping - ROS Wiki., URL: http://wiki.ros.org/gmapping
89. amcl - ROS Wiki., URL: http://wiki.ros.org/amcl
90. QuickBlox Backend: Cloud Communication Backend API As A Service For Mobile And Web Apps., URL: https://quickblox.com/
91. Web video / voice calls on WebRTC, code samples., URL: https://quickblox.com/developers/Sample-webrtc-web#Integrate_video_calling_to_your_application
92. S. Coradeschi., GiraffPlus: Combining social interaction and long term monitoring for promoting independent living, 2013, p. 1.
93. Atallah, L., Lo, B., Yang, G.Z., Siegemund, F., Wirelessly Accessible Sensor Populations (WASP) for Elderly Care Monitoring, 2008, pp. 1-2.
94. Yang, G.Z., Body Sensor Networks, 2006.
95. Mutha, N., et al., Patient Health Monitoring Using Android Application, 2007.
96. How to Check Your Heart Rate on Any Android Phone - Android., URL:

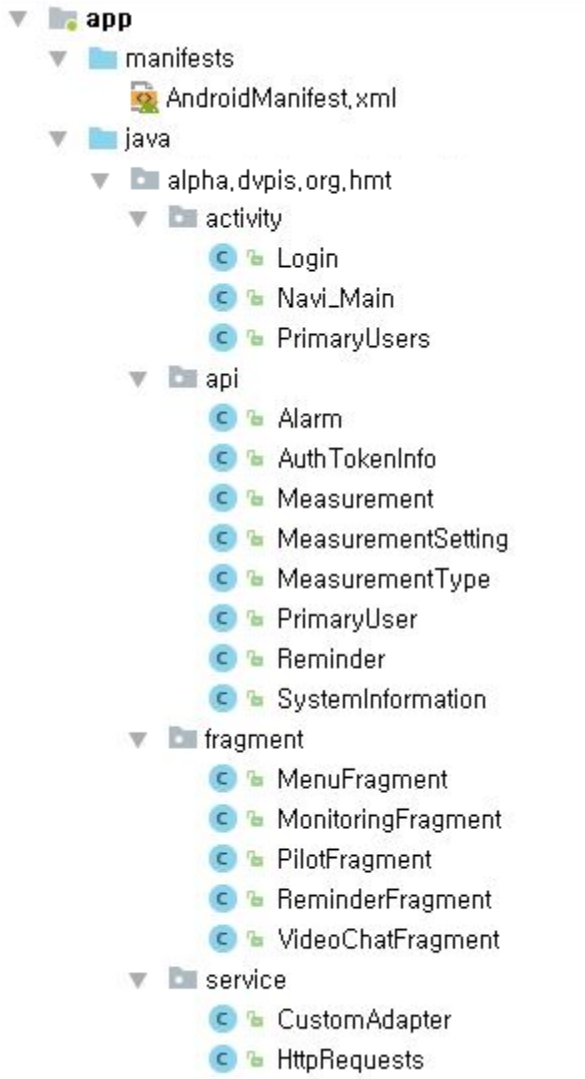
https://android.gadgethacks.com/how-to/check-your-heart-rate-any-android-phone-0164070/

97. Apache JMeter., URL: http://jmeter.apache.org/index.html
98. Nimbledroid., URL: https://nimbledroid.com/
99. Technology use among seniours - Pew Research Center., URL: http://www.pewinternet.org/2017/05/17/technology-use-among-seniors/
100. GitHub - markwsilliman/turtlebot-web-app: Web app for CoffeeBot., URL: https://github.com/markwsilliman/turtlebot-web-app
101. How do I start Mongo DB from Windows? - Stack Overflow., URL: https://stackoverflow.com/questions/20796714/how-do-i-start-mongo-db-from-windows
102. Silvia Coradeschi., et al., GiraffPlus Project: D5.1 Preliminary technological integration (Pre-T-Int) report, 2012, pp. 27-49
103. Aggregation Pipeline - MongoDB Manual 3.4., URL: https://docs.mongodb.com/manual/core/aggregation-pipeline/
104. Elastic Beanstalk Applications., URL: https://eu-west-1.console.aws.amazon.com/elasticbeanstalk/home?region=eu-west-1#/applications
105. MongoDB Hosting: Database-as-a-Service by mLab., URL: https://mlab.com/
106. Packages - ROS Wiki., URL: http://wiki.ros.org/Packages
107. GitHub - QuickBlox/quickblox-javascript-sdk: JavaScript SDK of QuickBlox cloud backend platform., URL: https://github.com/QuickBlox/quickblox-javascript-sdk
108. ngrok - secure introspectable tunnels to localhost., URL: https://ngrok.com/product
109. Robot Web Tools., URL: http://robotwebtools.org/

Appendices

Appendix A: Android Part

Structure of android project



HTTPRequests.java

```
public class HttpRequests extends AsyncTask<String, String, String> {

    public interface AsyncResponse {
        void authenticateUser(String token, String[] primaryNames);
        void getMeasurementsByPeriod(int[] measurements);
        void getMeasurementsSetting(MeasurementSetting setting);
        void getReminder(Reminder reminder);
        void ResourceAccessException();
    }

    public AsyncResponse delegate = null;

    public HttpRequests(AsyncResponse delegate){
        restTemplate = new RestTemplate();
        restTemplate.getMessageConverters().add(new  MappingJackson2HttpMessageConverter());

        this.delegate = delegate;
    }

    RestTemplate restTemplate;
    HttpEntity<String> httpEntity;
    HttpEntity<Reminder> httpEntity_reminder;
    HttpHeaders header;
    String token;
    String[] primaryNames;
    int[] measurements;
    MeasurementSetting setting;
    Reminder reminder;
    Reminder reminder_httpentity;
    String baseUrl = "http://dvpis.rpbqq3term.eu-west-1.elasticbeanstalk.com/";

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
    }

    @Override
    protected void onPostExecute(String s) {
        switch (s){
            case "authenticateUser":
                delegate.authenticateUser(token, primaryNames);
            case "getMeasurementsSetting":
                delegate.getMeasurementsSetting(setting);
            case "getMeasurementsByPeriod":
                delegate.getMeasurementsByPeriod(measurements);
            case "getReminder":
                delegate.getReminder(reminder);
            case "ResourceAccessException":
                delegate.ResourceAccessException();
            default:
```

```
                    break;
            }
        }

        @Override
        protected String doInBackground(String... params) {

            switch (params[0]){
                case "authenticateUser":
                    httpEntity = new HttpEntity<String>(new HttpHeaders());

                    ResponseEntity<String[]> response_auth;

                    try {
                        response_auth = restTemplate.exchange(baseUrl +
"/authenticateUser/test/test", HttpMethod.GET, httpEntity, String[].class);
                    }catch (ResourceAccessException e){
                        return "ResourceAccessException";
                    }

                    token = response_auth.getHeaders().getAuthorization();
                    primaryNames = response_auth.getBody();
                    return params[0];
                case "getMeasurementsSetting":
                    header = new HttpHeaders();
                    header.add("Authorization", params[1]);
                    header.add("primaryUser", params[2]);
                    httpEntity = new HttpEntity<String>(header);

                    ResponseEntity<MeasurementSetting> response_setting =
restTemplate.exchange(baseUrl+"/getMeasurementsSetting/", HttpMethod.GET, httpEntity,
MeasurementSetting.class);

                    setting = response_setting.getBody();
                    return params[0];

                case "getMeasurementsByPeriod":
                    System.out.println("parameters!: " + params[0] + " " + params[1] + " " +
params[2] + " " + params[3] + " " + params[4] + " " + params[5]);
                    header = new HttpHeaders();
                    header.add("Authorization", params[1]);
                    header.add("primaryUser", params[2]);
                    httpEntity = new HttpEntity<String>(header);

                    ResponseEntity<int[]> response_meas =
restTemplate.exchange(baseUrl+"/getMeasurementsByPeriod/" + params[3] + "/" + params[4] + "/" +
params[5], HttpMethod.GET, httpEntity, int[].class);
                    measurements = response_meas.getBody();
                    return params[0];

                case "setReminder":
                    header = new HttpHeaders();
                    header.add("Authorization", params[1]);
```

```
                header.setContentType(MediaType.APPLICATION_JSON);

                reminder_httpentity = new Reminder(Long.parseLong(params[2]), params[3]);

                httpEntity_reminder = new HttpEntity<Reminder>(reminder_httpentity, header);

                ResponseEntity<Reminder> response_setReminder =
restTemplate.exchange(baseUrl+"/setReminder", HttpMethod.POST, httpEntity_reminder,
Reminder.class);
                return params[0];

            case "getReminder":
                header = new HttpHeaders();
                header.add("Authorization", params[1]);
                httpEntity = new HttpEntity<String>(header);

                ResponseEntity<Reminder> response_reminder =
restTemplate.exchange(baseUrl+"/getReminder", HttpMethod.GET, httpEntity, Reminder.class);
                reminder = response_reminder.getBody();
                return params[0];

            case "deleteReminder":
                header = new HttpHeaders();
                header.add("Authorization", params[1]);
                header.setContentType(MediaType.APPLICATION_JSON);

                reminder_httpentity = new Reminder(Long.parseLong(params[2]), params[3]);

                httpEntity_reminder = new HttpEntity<Reminder>(reminder_httpentity, header);

                ResponseEntity<Reminder> response_deleteReminder =
restTemplate.exchange(baseUrl+"/deleteReminder", HttpMethod.POST, httpEntity_reminder,
Reminder.class);
                return params[0];
        }
        return null;
    }
}
```

MonitoringFragment.java

```
public class MonitoringFragment extends Fragment {
    TextView from_date;
    Button from_button;
    DatePickerDialog datePickerDialog;
    Date date;
    LineGraphSeries<DataPoint> series;
    GraphView graph;
    Spinner spinner;
    String[] measurementTypes;
    String primaryName;
```

```
    int[] measurement_values;
    String type;
    int type_int;
    String type_str;
    String curDate_str;
    MeasurementSetting setting;
    String token;
    Long curDateMilli;
    int curYear;
    int curMonth;
    int curDay;
    Date parseDate;
    Date[] days;
    String[] days_str;

    HttpRequests getMeasurementValues;
    HttpRequests getReminder;

    int oneDayMilli = 1000 * 60 * 60 * 24;

    public static MonitoringFragment newInstance(String token, String name) {

        Bundle args = new Bundle();

        MonitoringFragment fragment = new MonitoringFragment();

        args.putString("name", name);
        args.putString("token", token);
        fragment.setArguments(args);

        return fragment;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        token = getArguments().getString("token");
        primaryName = getArguments().getString("name");
    }

    @Override
    public void onResume() {
        super.onResume();
    }

    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        View view    = inflater.inflate(R.layout.activity_monitoring, container, false);

        testData = new TestData();

        from_date = (TextView) view.findViewById(R.id.textView);
        from_button = (Button) view.findViewById(R.id.button);
```

```
        graph = (GraphView) view.findViewById(R.id.graph);

        spinner = (Spinner) view.findViewById(R.id.spinner);
        spinner.setPrompt("Measurement types");

        final HttpRequests getMeasurementsSetting = new HttpRequests(new
HttpRequests.AsyncResponse() {
            @Override
            public void getMeasurementsByPeriod(int[] measurements) {}
            @Override
            public void authenticateUser(String token_out, String[] primaryNames_out) {}
            @Override
            public void getReminder(Reminder reminder) {}
            public void ResourceAccessException(){}

            @Override
            public void getMeasurementsSetting(MeasurementSetting setting_out) {
                setting = setting_out;

                measurementTypes = setting.getDescriptions();
                type = measurementTypes[0];
                type_int = 0;

                ArrayAdapter<String> adapter = new ArrayAdapter<>(
                        getActivity(),
                        android.R.layout.simple_list_item_1,
                        measurementTypes
                );

                spinner.setAdapter(adapter);

                spinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener()
{
                    @Override
                    public void onItemSelected(AdapterView<?> parent, View view, int
position, long id) {

                        type = (String) spinner.getSelectedItem();
                        type_int = position;
                        deleteGraph();

datePickerDialog.getDatePicker().setMinDate(setting.getStartDates()[position]);

datePickerDialog.getDatePicker().setMaxDate(setting.getLastDates()[position]);

                        type_str = String.valueOf(setting.getGroups()[type_int]);
                        curDate_str = String.valueOf(curDateMilli);

                        getMeasurementValues = new HttpRequests(new
HttpRequests.AsyncResponse() {

                            @Override
```

```
                                public void authenticateUser(String token, String[] primaryNames)
{}
                                @Override
                                public void getMeasurementsSetting(MeasurementSetting setting)
{}
                                @Override
                                public void getReminder(Reminder reminder) {}
                                public void ResourceAccessException(){}

                                @Override
                                public void getMeasurementsByPeriod(int[] measurements) {
                                    measurement_values = measurements;
                                    days = new Date[5];
                                    days_str = new String[5];
                                    Date d = parseDate;

                                    for(int i = 0; i < 5; i++){
                                        days_str[i] = String.valueOf(d.getDate());
                                        days[i] = d;
                                        d = new Date(d.getTime() + oneDayMilli);
                                    }
                                    deleteGraph();
                                    drawGraph(measurement_values, curDay, days, days_str);
                                }
                            });
                            getMeasurementValues.execute("getMeasurementsByPeriod", token,
primaryName, type_str, curDate_str, "5");

                        }

                        @Override
                        public void onNothingSelected(AdapterView<?> parent) {

                        }
                    });

                    date = new Date();
                    Calendar cal=Calendar.getInstance();
                    cal.setTimeInMillis(setting.getLastDates()[0]);
                    curDateMilli = setting.getLastDates()[0] - (oneDayMilli * 4);
                    parseDate = new Date(curDateMilli);

                    curYear = parseDate.getYear() + 1900;
                    curMonth = parseDate.getMonth() + 1;
                    curDay = parseDate.getDate();

                    datePickerDialog = new DatePickerDialog(getActivity(), new
DatePickerDialog.OnDateSetListener() {
                        @Override
                        public void onDateSet(DatePicker view, int year, int month, int
dayOfMonth) {
                            month = month + 1;
                            String dateStr = dayOfMonth + "/" + month + "/" + year;
```

```
                    from_date.setText(dateStr);

                    curYear = year;
                    curMonth = month;
                    curDay = dayOfMonth;

                    DateFormat formatter = new SimpleDateFormat("dd/MM/yyyy");
                    parseDate = null;
                    try {
                        parseDate = formatter.parse(dateStr);
                    } catch (ParseException e) {
                        e.printStackTrace();
                    }

                    curDateMilli = parseDate.getTime();
                    while(curDateMilli >= (setting.getLastDates()[type_int]-(oneDayMilli *
4))){
                        curDateMilli = curDateMilli - oneDayMilli;
                    }
                    type_str = String.valueOf(setting.getGroups()[type_int]);
                    curDate_str = String.valueOf(curDateMilli);

                    getMeasurementValues = new HttpRequests(new
HttpRequests.AsyncResponse() {
                        @Override
                        public void authenticateUser(String token, String[] primaryNames)
{}
                        @Override
                        public void getMeasurementsSetting(MeasurementSetting setting)
{}
                        @Override
                        public void getReminder(Reminder reminder) {}
                        public void ResourceAccessException(){}
                        @Override
                        public void getMeasurementsByPeriod(int[] measurements) {
                            measurement_values = measurements;
                            days = new Date[5];
                            days_str = new String[5];
                            Date d = new Date(curDateMilli);
                            for(int i = 0; i < 5; i++){
                                days_str[i] = String.valueOf(d.getDate());
                                days[i] = d;
                                d = new Date(d.getTime() + oneDayMilli);
                            }
                            deleteGraph();
                            drawGraph(measurement_values, curDay, days, days_str);
                        }

                    });
                    getMeasurementValues.execute("getMeasurementsByPeriod", token,
primaryName, type_str, curDate_str, "5");

                }
```

```
                }, date.getYear()+1900, date.getMonth(), date.getDay());

                datePickerDialog.getDatePicker().setMinDate(setting.getStartDates()[0]);
                datePickerDialog.getDatePicker().setMaxDate(setting.getLastDates()[0]);

                datePickerDialog.updateDate(curYear, curMonth, curDay);
                from_date.setText(curDay + "/" + curMonth + "/" + curYear);

                from_button.setOnClickListener(new View.OnClickListener() {
                    @Override
                    public void onClick(View v) {
                        datePickerDialog.show();
                    }
                });

                getReminder = new HttpRequests(new HttpRequests.AsyncResponse() {

                    @Override
                    public void authenticateUser(String token, String[] primaryNames) {}
                    @Override
                    public void getMeasurementsSetting(MeasurementSetting setting) {}
                    @Override
                    public void getMeasurementsByPeriod(int[] measurements) {}
                    public void ResourceAccessException(){}
                    @Override
                    public void getReminder(Reminder reminder) {}
                });
            }
        });

        getMeasurementsSetting.execute("getMeasurementsSetting", token, primaryName);

        return view;
    }

    public void drawGraph(int[] measurement_values, int startDate, Date[] days, String[] days_str){

        String[] ylabel = new String[measurement_values.length];

        for(int i = 0; i < measurement_values.length; i++){
            ylabel[i] = days_str[i];
        }

        series = new LineGraphSeries<DataPoint>();
        for(int i = 0; i < measurement_values.length; i++) {
            series.appendData(new DataPoint(i, measurement_values[i]), true, 180);
        }

        StaticLabelsFormatter staticLabelsFormatter = new StaticLabelsFormatter(graph);
        staticLabelsFormatter.setHorizontalLabels(ylabel);
        graph.getGridLabelRenderer().setLabelFormatter(staticLabelsFormatter);
        graph.getGridLabelRenderer().setNumHorizontalLabels(ylabel.length);
```

```
            graph.addSeries(series);
        }

        public void deleteGraph(){
            graph.removeAllSeries();
        }
}
```

VideoChatFragment.java

```
public class VideoChatFragment extends Fragment
{
    WebView webview;

    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        View view = inflater.inflate(R.layout.activity_videochat, container, false);

        webview = (WebView) view.findViewById(R.id.webView);

        this.getActivity().setVolumeControlStream(AudioManager.STREAM_VOICE_CALL);

        webview.setWebChromeClient(new WebChromeClient() {
            @Override
            public void onPermissionRequest(final PermissionRequest request) {
                getActivity().runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        request.grant(request.getResources());
                    }
                });
            }
        });

        WebSettings webSetting = webview.getSettings();
        webSetting.setJavaScriptEnabled(true);
        webSetting.setJavaScriptCanOpenWindowsAutomatically(true);
        webSetting.setDomStorageEnabled(true);
        webSetting.setAllowFileAccessFromFileURLs(true);
        webSetting.setAllowUniversalAccessFromFileURLs(true);
        webSetting.setMediaPlaybackRequiresUserGesture(false);

        webview.loadUrl("file:///android_asset/videochat/index.html");

        return view;
    }
}
```

Appendix B: Server Part and Database Part

Example of the Demo Database


```
{
    "_id" : ObjectId("58628f6b91749269398c1cba"),
    "username" : "userName",
    "password" : "userPass",
    "name" : "",
    "primaryUsers" : [
        {
            "name" : "Francisco Juliano",
            "videoChatName" : "hmt_robot_Francisco",
            "measurements" : [
                {
                    "date" : NumberLong(1483574400000),
                    "value" : 73.0,
                    "measurementType" : {
                        "group" : 3,
                        "name" : "DBP",
                        "description" : "Diastolic blood pressure",
                        "measurementUnits" : "mmHg"
                    }
                },
                {
                    "date" : NumberLong(1483574400000),
                    "value" : 173.0,
                    "measurementType" : {
                        "group" : 2,
                        "name" : "SBP",
                        "description" : "Systolic blood pressure",
                        "measurementUnits" : "mmHg"
                    }
                },
                {
                    "date" : NumberLong(1483574400000),
                    "value" : 79.0,
                    "measurementType" : {
                        "group" : 4,
                        "name" : "HR",
                        "description" : "Heart rate",
                        "measurementUnits" : "bpm"
                    }
                },
                {
                    "date" : NumberLong(1483660800000),
                    "value" : 133.0,
                    "measurementType" : {
                        "group" : 3,
                        "name" : "DBP",
                        "description" : "Diastolic blood pressure",
                        "measurementUnits" : "mmHg"
```

```
                        }
                    },
                    {
                        "date" : NumberLong(1483660800000),
                        "value" : 73.0,
                        "measurementType" : {
                            "group" : 2,
                            "name" : "SBP",
                            "description" : "Systolic blood pressure",
                            "measurementUnits" : "mmHg"
                        }
                    },
                ],
            },
            {
                "name" : "Maria Perez",
            },
            {
                "name" : "Luisa Jimenez",
            }
        ],
        "systemInformation" : {
            "lastLogonDate" : NumberLong(1347456156577),
            "lastLogoutDate" : NumberLong(1347456146577),
            "sessionExpirationDate" : NumberLong(1347456166577),
            "videoChatName" : "hmt_application_userName"
        },
        "reminders" : [
            {
                "date": NumberLong(1510137900000),
                "description": "test description"
            }
        ]
}
```

Structure of database server project

HomeController.java

```
@RestController
public class HomeController {

	private final MongoController mongoService = new MongoController();
	private final JwtService jwtService = new JwtService();


	@RequestMapping(value = "/authenticateUser/{username}/{password}",
					method = RequestMethod.GET,
					produces = APPLICATION_JSON_VALUE)
	public ResponseEntity<String[]> authenticateUser(@PathVariable String username,

		@PathVariable String password,

		HttpServletResponse response){

		String[] output_primaryUserNames = mongoService.authenticateUser(username,
password);

		if(output_primaryUserNames == null){
			return new ResponseEntity<String[]>(HttpStatus.NOT_FOUND);
		}else{
			String token = jwtService.tokenFor(username);
			response.setHeader("Authorization", token);

			return new ResponseEntity<String[]>(output_primaryUserNames,
HttpStatus.OK);
		}
	}

	@RequestMapping(value = "/getMeasurementsSetting",
							method = RequestMethod.GET,
produces = APPLICATION_JSON_VALUE)
	public ResponseEntity<MeasurementSetting> getMeasurementsSetting(HttpServletRequest
request) throws SignatureException {
		String secondaryName = getNameFromToken(request);
		String primaryName = request.getHeader("primaryUser");

		return new
ResponseEntity<MeasurementSetting>(mongoService.getMeasurementsSetting(secondaryName,
primaryName), HttpStatus.OK);
	}

	@RequestMapping(value = "/getMeasurementsByPeriod/{group}/{date}/{periodType}",
							method = RequestMethod.GET,
produces = APPLICATION_JSON_VALUE)
	public ResponseEntity<int[]> getMeasurementsByPeriod(@PathVariable int group,
@PathVariable long date,

	@PathVariable int periodType, HttpServletRequest request) throws SignatureException {
```

```
		String secondaryName = getNameFromToken(request);
		String primaryName = request.getHeader("primaryUser");

		int[] measurements = mongoService.getMeasurementsByPeriod(secondaryName,
primaryName, group, date, periodType);
		return new ResponseEntity<int[]>(measurements, HttpStatus.OK);
	}

	@RequestMapping(value = "/getSystemInformation",
							method = RequestMethod.GET,
produces = APPLICATION_JSON_VALUE)
	public SystemInformation getSystemInformation(HttpServletRequest request) throws
SignatureException{

		String secondaryName = getNameFromToken(request);
		SystemInformation information =
mongoService.getSystemInformation(secondaryName);

		return information;
	}

	@RequestMapping(value = "/getLastAlarms/{num}",
				method = RequestMethod.GET, produces =
APPLICATION_JSON_VALUE)
	public ArrayList<Alarm> getLastAlarms(@PathVariable int num, HttpServletRequest
request) throws SignatureException{

		String secondaryName = getNameFromToken(request);
		String primaryName = request.getHeader("primaryUser");

		ArrayList<Alarm> alarms = mongoService.getLastAlarms(secondaryName,
primaryName, num);
		return alarms;
	}

	@RequestMapping(value = "/setReminder", method = RequestMethod.POST, produces =
APPLICATION_JSON_VALUE)
	public void setReminder(HttpServletRequest request, @RequestBody Reminder reminder)
throws SignatureException {
		String secondaryName = getNameFromToken(request);

		mongoService.setReminder(secondaryName, reminder);
	}

	@RequestMapping(value = "/getReminder", method = RequestMethod.GET, produces =
APPLICATION_JSON_VALUE)
	public Reminder getReminder(HttpServletRequest request) throws SignatureException {
		String secondaryName = getNameFromToken(request);

		return mongoService.getReminder(secondaryName);
	}
```

```
	@RequestMapping(value = "/deleteReminder", method = RequestMethod.POST, produces =
APPLICATION_JSON_VALUE)
	public void deleteReminder(HttpServletRequest request, @RequestBody Reminder
reminder) throws SignatureException {
		String secondaryName = getNameFromToken(request);

		mongoService.deleteReminder(secondaryName, reminder);
	}

	@RequestMapping(value = "/logout", method = RequestMethod.POST)
	public void userLogout(HttpServletRequest request) throws SignatureException{
		String secondaryName = getNameFromToken(request);
		mongoService.userLogout(secondaryName);
	}

	public String getNameFromToken(HttpServletRequest request) throws SignatureException{
		String token = request.getHeader("Authorization");
		Jws<Claims> claims = jwtService.verify(token);
		String secondaryName = claims.getBody().getSubject().toString();

		return secondaryName;
	}
}
```

MongoController.java

```
public class MongoController {
	// Set credentials
	MongoCredential credential = MongoCredential.createScramSha1Credential("kangup",
"dvpis_database", "0215".toCharArray());
	ServerAddress serverAddress = new ServerAddress("ds039261.mlab.com", 39261);

	// Mongo Client
	MongoClient mongoClient = new MongoClient(serverAddress,Arrays.asList(credential));

	// Mongo DB Factory
	SimpleMongoDbFactory simpleMongoDbFactory = new SimpleMongoDbFactory(
			mongoClient, "dvpis_database");

	MongoTemplate mongoTemplate = new MongoTemplate(simpleMongoDbFactory);

	public SecondaryUser findSecondary(String username){
		Criteria criteria = new Criteria("username");
		criteria.is(username);
		Query query = new Query(criteria);

		SecondaryUser user = mongoTemplate.findOne(query, SecondaryUser.class,
"Secondary_Users");
```

```
		return user;
	}

	public PrimaryUser findPrimary(SecondaryUser secondaryUser, String primaryName){
		PrimaryUser primaryUser = null;
		for(PrimaryUser primary : secondaryUser.getPrimaryUsers()){
			if(primary.getName().equals(primaryName)){
				primaryUser = primary;
			}
		}
		return primaryUser;
	}

	public String[] listPrimaryUsers(String secondaryName){
		SecondaryUser user = findSecondary(secondaryName);
		String[] primaryUsers = new String[user.getPrimaryUsers().size()];
		int i = 0;
		for(PrimaryUser primary : user.getPrimaryUsers()){
			primaryUsers[i++] = primary.getName();
		}
		return primaryUsers;
	}

	public String[] authenticateUser(String secondaryName, String password){
		AggregationOperation unwind = Aggregation.unwind("primaryUsers");
		AggregationOperation match =
Aggregation.match(Criteria.where("username").is(secondaryName).and("password").is(password));
		AggregationOperation group =
Aggregation.group("null").first("systemInformation").as("systemInformation")

					.push("$primaryUsers.name").as("output_primaryUserNames");

		Aggregation aggregation = Aggregation.newAggregation(unwind,match,group);

		List<QueryOutput> result = mongoTemplate.aggregate(aggregation,
"Secondary_Users", QueryOutput.class).getMappedResults();

		if(result.isEmpty()) return null;
		else{
			userLogin(secondaryName, result.get(0).getSystemInformation());
			return result.get(0).getOutput_primaryUserNames();
		}
	}

	public MeasurementSetting getMeasurementsSetting(String secondaryName, String
primaryName){
		AggregationOperation match1 =
Aggregation.match(Criteria.where("username").is(secondaryName));
		AggregationOperation unwind1 = Aggregation.unwind("primaryUsers");
		AggregationOperation match2 =
Aggregation.match(Criteria.where("primaryUsers.name").is(primaryName));
		AggregationOperation unwind2 =
Aggregation.unwind("primaryUsers.measurements");
```

```
                AggregationOperation sort = Aggregation.sort(Sort.Direction.ASC,
"primaryUsers.measurements.measurementType.group");
                AggregationOperation group =
Aggregation.group("primaryUsers.measurements.measurementType.group")

        .first("primaryUsers.measurements.measurementType.description").as("description")

        .min("primaryUsers.measurements.date").as("startDate")

        .max("primaryUsers.measurements.date").as("lastDate");

                Aggregation aggregation =
Aggregation.newAggregation(match1,unwind1,match2,unwind2,sort,group);

                AggregationResults<DBObject> result = mongoTemplate.aggregate(aggregation,
"Secondary_Users", DBObject.class);

                List<DBObject> settings = result.getMappedResults();
                MeasurementSetting setting = new MeasurementSetting(settings.size());
                int i = 0;
                for(DBObject ob : settings){
                        setting.getGroups()[i] = (int) ob.get("_id");
                        setting.getDescriptions()[i] = (String) ob.get("description");
                        setting.getStartDates()[i] = (long) ob.get("startDate");
                        setting.getLastDates()[i++] = (long) ob.get("lastDate");
                }

                return setting;
        }

        public int[] getMeasurementsByPeriod(String secondaryName, String primaryName,

                                        int group, long date, int periodType){

                long endDate = getMaxDate(date, periodType);

                AggregationOperation match1 =
Aggregation.match(Criteria.where("username").is(secondaryName));
                AggregationOperation unwind1 = Aggregation.unwind("primaryUsers");
                AggregationOperation match2 =
Aggregation.match(Criteria.where("primaryUsers.name").is(primaryName));
                AggregationOperation unwind2 =
Aggregation.unwind("primaryUsers.measurements");
                AggregationOperation match =
Aggregation.match(Criteria.where("primaryUsers.measurements.measurementType.group").is(group)

        .andOperator(Criteria.where("primaryUsers.measurements.date").gte(date),

Criteria.where("primaryUsers.measurements.date").lte(endDate)));

                AggregationOperation sort = Aggregation.sort(Sort.Direction.ASC,
"primaryUsers.measurements.date");
                AggregationOperation group_query =
```

```
Aggregation.group("null").push("$primaryUsers.measurements.value").as("output_measurements");

                Aggregation aggregation =
Aggregation.newAggregation(match1,unwind1,match2,unwind2,match,sort,group_query);

                AggregationResults<QueryOutput> result = mongoTemplate.aggregate(aggregation,
"Secondary_Users", QueryOutput.class);

                return result.getMappedResults().get(0).getOutput_measurements();
        }

        public SystemInformation getSystemInformation(String secondaryName){
                AggregationOperation match =
Aggregation.match(Criteria.where("username").is(secondaryName));
                AggregationOperation project = Aggregation.project("systemInformation");

                Aggregation aggregation = Aggregation.newAggregation(match,project);

                AggregationResults<QueryOutput> result = mongoTemplate.aggregate(aggregation,
"Secondary_Users", QueryOutput.class);

                return result.getMappedResults().get(0).getSystemInformation();
        }

        public ArrayList<Alarm> getLastAlarms(String secondaryName, String primaryName, int
num){
                AggregationOperation unwind1 = Aggregation.unwind("primaryUsers");
                AggregationOperation unwind2 = Aggregation.unwind("primaryUsers.alarms");
                AggregationOperation match =
Aggregation.match(Criteria.where("username").is(secondaryName)

        .and("primaryUsers.name").is(primaryName));
                AggregationOperation sort = Aggregation.sort(Sort.Direction.ASC,
"primaryUsers.alarms.date");
                AggregationOperation limit = Aggregation.limit(num);
                AggregationOperation group =
Aggregation.group("null").push("$primaryUsers.alarms").as("output_alarms");

                Aggregation aggregation =
Aggregation.newAggregation(unwind1,unwind2,match,sort,limit,group);

                AggregationResults<QueryOutput> result = mongoTemplate.aggregate(aggregation,
"Secondary_Users", QueryOutput.class);

                return result.getMappedResults().get(0).getOutput_alarms();
        }

        public void updateInformation(String secondaryName, SystemInformation information){
                Criteria criteria = new Criteria("username");
                criteria.is(secondaryName);
                Query query = new Query(criteria);

                Update update = new Update();
```

```
		update.set("systemInformation", information);

		mongoTemplate.updateFirst(query, update, "Secondary_Users");
}

public void userLogin(String secondaryName, SystemInformation information){
		long date = new java.util.Date().getTime();
		long expiration = date + (2 * 3600000);
		information.setLastLogonDate(date);
		information.setSessionExpirationDate(expiration);
		updateInformation(secondaryName, information);
}

public void userLogout(String secondaryName){
		SystemInformation information = getSystemInformation(secondaryName);
		long date = new java.util.Date().getTime();
		information.setLastLogoutDate(date);

		updateInformation(secondaryName, information);
}

public void setReminder(String secondaryName, Reminder reminder){
		ArrayList<Reminder> reminders = getReminders(secondaryName);

		if(reminders == null){
			reminders = new ArrayList<Reminder>();
		}

		reminders.add(reminder);
		Criteria criteria = new Criteria("username");
		criteria.is(secondaryName);
		Query query = new Query(criteria);

		Update update = new Update();
		update.set("reminders", reminders);

		mongoTemplate.updateFirst(query, update, "Secondary_Users");
}

public Reminder getReminder(String secondaryName){
		ArrayList<Reminder> reminders = getReminders(secondaryName);
		if(reminders != null) {
			Reminder reminder = reminders.get(0);

			return reminder;
		}else return null;
}

public void deleteReminder(String secondaryName, Reminder reminder){
		ArrayList<Reminder> reminders = getReminders(secondaryName);
		Reminder r;
		if(reminders != null){
			for(int i = 0; i < reminders.size(); i++){
```

```
                                        r = reminders.get(i);
                                        if(r.getDate()==reminder.getDate() &&
r.getDescription().equals(reminder.getDescription())){
                                                reminders.remove(i);
                                                break;
                                        }
                                }
                                Criteria criteria = new Criteria("username");
                                criteria.is(secondaryName);
                                Query query = new Query(criteria);

                                Update update = new Update();
                                update.set("reminders", reminders);

                                mongoTemplate.updateFirst(query, update, "Secondary_Users");
                        }
                }

                public ArrayList<Reminder> getReminders(String secondaryName){
                        AggregationOperation match =
Aggregation.match(Criteria.where("username").is(secondaryName));
                        AggregationOperation unwind = Aggregation.unwind("reminders");
                        AggregationOperation sort = Aggregation.sort(Sort.Direction.ASC,
"reminders.date");
                        AggregationOperation group =
Aggregation.group("null").push("$reminders").as("reminders");
                       Aggregation aggregation = Aggregation.newAggregation(match,unwind,sort,group);

                        AggregationResults<QueryOutput> result = mongoTemplate.aggregate(aggregation,
"Secondary_Users", QueryOutput.class);

                        if(!result.getMappedResults().isEmpty()) {
                                ArrayList<Reminder> reminders = (ArrayList<Reminder>)
result.getMappedResults().get(0).getReminders();
                                return reminders;
                        }

                        return null;

                }

                public long getMaxDate(long date, int periodType){
                        long oneDayInMilliSecond = 86400000;

                        date = date + (oneDayInMilliSecond * periodType);
                        return date;
                }
}
```

Appendix C: Pilot System

index.js

```
// Connecting to ROS
// -----------------
var ros = new ROSLIB.Ros();

ros.connect('ws://kangup.eu.ngrok.io');

// If there is an error on the backend, an 'error' emit will be emitted.
ros.on('error', function(error) {
    document.getElementById('connecting').style.display = 'none';
    document.getElementById('connected').style.display = 'none';
    document.getElementById('closed').style.display = 'none';
    document.getElementById('error').style.display = 'inline';
    console.log(error);
});

// Find out exactly when we made a connection.
ros.on('connection', function() {
    console.log('Connection made!');
    document.getElementById('connecting').style.display = 'none';
    document.getElementById('error').style.display = 'none';
    document.getElementById('closed').style.display = 'none';
    document.getElementById('connected').style.display = 'inline';
});

ros.on('close', function() {
    console.log('Connection closed.');
    document.getElementById('connecting').style.display = 'none';
    document.getElementById('connected').style.display = 'none';
    document.getElementById('closed').style.display = 'inline';
});

// First, we create a Topic object with details of the topic's name and message type.
var cmdVel = new ROSLIB.Topic({
    ros : ros,
    name : '/cmd_vel_mux/input/teleop',
    messageType : 'geometry_msgs/Twist'
});

// Then we create the payload to be published. The object we pass in to ros.Message matches the
// fields defined in the geometry_msgs/Twist.msg definition.
var twist = new ROSLIB.Message({
    linear : {
        x : 0,
        y : 0,
        z : 0
    },
    angular : {
        x : 0,
```

```
        y : 0,
        z : 0
    }
});

var timer = null;
var elements = document.getElementsByClassName('button');

function move(evt){
    if(evt == 'forward'){
        mouseUp();
        twist.linear.x = 0.25;
        btn_forward.parentElement.className += " active";
        timer = setInterval(function(){
            cmdVel.publish(twist);
        }, 100);
    }else if(evt == 'left'){
        mouseUp();
        twist.angular.z = 0.75;
        //twist.linear.x = 0.25;
        btn_left.parentElement.className += " active";
        timer = setInterval(function(){
            cmdVel.publish(twist);
        }, 100);
    }else if(evt == 'right'){
        mouseUp();
        twist.angular.z = -0.75;
        //twist.linear.x = 0.25;
        btn_right.parentElement.className += " active";
        timer = setInterval(function(){
            cmdVel.publish(twist);
        }, 100);
    }else if(evt == 'stop'){
        mouseUp();
        twist.angular.z = 0;
        twist.linear.x = 0;
        btn_stop.parentElement.className += " active";
        mouseUp();
    }else if(evt == 'reverse'){
        mouseUp();
        twist.linear.x = -0.2;
        btn_reverse.parentElement.className += " active";
        timer = setInterval(function(){
            cmdVel.publish(twist);
        }, 100);
    }
}

document.onkeydown = function(e) {
        switch (e.keyCode) {
            case 37:
                if(timer == null) {
                    //console.log("press left");
```

```
                move("left");
            }
            break;
        case 38:
            if(timer == null) {
                //console.log("press forward");
                move("forward");
            }
            break;
        case 39:
            if(timer == null) {
                //console.log("press right");
                move("right");
            }
            break;
        case 40:
            if(timer == null) {
                //console.log("press reverse");
                move("reverse");
            }
            break;
    }
};

document.onkeyup = function(){
    clearInterval(timer);
    timer = null;
    twist.linear.x = twist.linear.y = twist.angular.z = 0;

    for (var i = 0; i < elements.length; i++) {
        elements[i].className = "button";
    }
}

function mouseUp(){
    clearInterval(timer);
    timer = null;
    twist.linear.x = twist.linear.y = twist.angular.z = 0;

    for (var i = 0; i < elements.length; i++) {
        elements[i].className = "button";
    }
}

var myApp = angular.module('myApp', ['ngRoute']);

myApp.config(function($routeProvider){
    $routeProvider
        .when('/', {
            templateUrl : 'pages/video.html',
            controller : 'videoController'
        })
        .when('/video', {
```

```
            templateUrl : 'pages/video.html',
            controller : 'videoController'
        })
        .when('/map', {
            templateUrl : 'pages/map.html',
            controller : 'mapController'
        })
});

myApp.controller('videoController', function($scope){

    var sub_image = new ROSLIB.Topic({
        ros : ros,
        name : '/camera/rgb/image_raw/compressed',
        messageType : 'sensor_msgs/CompressedImage'
    });

    sub_image.subscribe(function(message){
        var imageData = "" + message.data;
        var style = "width:" + window.innerWidth + "px;height:" + window.innerHeight + "px;";
        var image_element = document.getElementById('ros_image');
        if(image_element != null){
            image_element.setAttribute('src', imageData);
            document.getElementById('ros_image').setAttribute('style', style);
        }
    })

    cmdVel.publish(twist);
});

myApp.controller('mapController', function($scope){
    // Create the main viewer.
    var viewer = new ROS2D.Viewer({
        divID : 'map',
        width : window.innerWidth,
        height : window.innerHeight
    });

    // Setup the nav client.
    var nav = NAV2D.OccupancyGridClientNav({
        ros : ros,
        rootObject : viewer.scene,
        viewer : viewer,
        continuous : true,
        serverName : '/move_base',
        actionName : '/move_base_msgs/MoveBaseAction',
        costmap : '/move_base/local_costmap/costmap',
        withOrientation : true
    });

    var viewer2 = new ROS2D2.CostMapViewer({
        divID : 'costmap',
        width : window.innerWidth,
```

```
      height : window.innerHeight
    });

    var nav2 = NAV2D2.CostMapClientNav({
      ros : ros,
      rootObject : viewer2.scene,
      viewer : viewer2,
      continuous : true,
      topic : '/move_base/local_costmap/costmap',
      serverName : '/move_base',
      actionName : '/move_base_msgs/MoveBaseAction',
      withOrientation : true,
      costmap : true
    });
});
```