POLITECNICO DI MILANO

Corso di Laurea Magistrale in Ingegneria Informatica

Dipartimento di Elettronica, Informazione e Bioingegneria

Toward a Mobile User Interface for the Personal Mobility Kit of The ALMA Project

Supervisor: Prof. Matteo Matteucci

Co-supervisors: Prof. Sara Comai

Ing. Giulio Fontana

Graduation Thesis of:

Okan Çoban, matricola 796254

Academic year 2013-2014

Contents

List of Figures

Abstract

For most of the cases, people with different types of motor disabilities need assistance for mobility. Autonomous wheelchairs try to solve mobility problems for these people by assisting them to navigate. This assistance could be semi-autonomous or autonomous depending on the level of users' involvement in the process of navigation.

For both autonomous and semi-autonomous navigation of wheelchairs, user needs to be informed about the position, speed, direction, status of the wheelchair (moving, idle, waiting… etc.). User should also be able to configure and use the wheelchair easily and safely through an interface. This interface should be simple and easy to use to increase the usability of the wheelchair.

In Politecnico di Milano, there are two projects under development for solving problems of disabled people. Within *LURCH (Let Unleashed Robots Crawl the House) project,* a software system for robotic powered wheelchairs is developed. This system has capabilities of moving wheelchair autonomously on a given map, estimating odometry data (position, rotation and velocity) of the wheelchair, and detecting the obstacles around it using sensors. *LURCH* intends to solve navigation problems of people with motor disabilities by creating a system with software and hardware technologies. The developed solution is going to be *PMK (Personal Mobility Kit)* module of ALMA project and will be connected with *PNA (Personal Navigation Assistant)* module

of *ALMA (Ageing without Losing Mobility and Autonomy)* project. ALMA project aims to create a personal indoor navigation system for elderly and people with some disabilities.

In this work, a mobile application is proposed as the base of further developments to create a complete user interface for *PMK*. The purpose is to create a familiar and easy to use interface for the existing system so that people with motor disabilities can use the *PMK* easily and safely. The complete user interface will be merged with the mobile application developed as the user interface of *PNA.*

The application visualizes the data supplied by wheelchair in order to give feedback to users about the movement of the wheelchair on an indoor map and about the obstacles around wheelchair. *IndoorGML maps from ALMA servers* are proposed as a source to create occupancy data of maps. This approach creates a map source for *PMK* by generating new maps easily from already collected data without the need of *gmapping* technique. The application allows user to download a map in *IndoorGML* format. From this map, occupancy data is retrieved and a map for *PMK* is created and sent to wheelchair.

In order to communicate with *ROS (Robot Operating System)* framework, which runs on the wheelchair, *rosjava for android* framework is used in the application. Beside the developed mobile application, some modifications are made on ROS nodes of *PMK*.

Sommario

Per la maggior parte dei casi, le persone con diversi tipi di disabilità motorie hanno bisogno di assistenza per la mobilità. Carrozzine autonome per disabili cercano di risolvere i problemi di mobilità di queste persone aiutandoli a navigare. Dipende del livello di coinvolgimento degli utenti nel processo di navigazione, l'assistenza potrebbe essere semi-autonomo o autonomo.

Per la navigazione autonoma e semi-autonomo di carrozzina per disabili, l'utente deve essere informato circa la posizione, la velocità, la direzione e lo stato della carrozzina (in movimento, inattivo, in attesa ... eccetera). L'utente anche dovrebbe essere in grado di configurare e utilizzare la carrozzina con facilità e senza rischi usando un'interfaccia. Questa interfaccia deve essere semplice e facile da utilizzare per aumentare l'usabilità della carrozzina.

In Politecnico di Milano, ci sono due progetti in fase di sviluppo per risolvere i problemi delle persone disabili. All'interno progetto LURCH (Let Unleashed Robots Crawl the House), un sistema di software è sviluppato per la robotica carrozzina motorizzata. Questo sistema ha capacità di muoversi carrozzina autonomamente su una mappa, la stima dei dati odometria (posizione, rotazione e velocità) della carrozzina e individuare gli ostacoli intorno utilizzando sensori. LURCH intende risolvere i problemi di navigazione delle persone con disabilità motorie, creando un sistema di tecnologie software e hardware. La soluzione sviluppata sarà PMK (Personal Mobility Kit) modulo di progetto ALMA e sarà collegato con PNA (Personal Navigation Assistant) modulo di progetto ALMA (Ageing without Losing Mobility and Autonomy). Scopo di

progetto ALMA è creare un sistema di navigazione interna personale per anziani e le persone con alcune disabilità.

In questo lavoro, un'applicazione mobile si propone come la base di ulteriori sviluppi per creare un'interfaccia utente completa per PMK. Lo scopo è creare un ambiente familiare e facile da usare per il sistema esistente in modo che le persone con disabilità motorie possono utilizzare il PMK modo semplice e senza rischi. L'interfaccia utente completa sarà fusa con l'applicazione mobile sviluppata come l'interfaccia utente di PNA.

L'applicazione visualizza i dati forniti dai carrozzina per dare risposte agli utenti sul movimento della carrozzina su una mappa interna e sugli ostacoli intorno carrozzina. Mappe IndoorGML dai server ALMA sono proposti come fonte per creare i dati di occupazione di mappe. Questo approccio crea un fonte di mappe per PMK generando nuove mappe facilmente da dati già raccolti senza la necessità di tecnica di gmapping. L'applicazione permette all'utente di scaricare una mappa in formato IndoorGML. Da questa mappa, i dati di occupazione vengono recuperati e una mappa per PMK viene creato e inviato alla carrozzina.

Per comunicare con il framework ROS (Robot Operating System), che viene eseguita sulla carrozzina, framework rosjava per Android è utilizzato nell’applicazione. Oltre l'applicazione mobile sviluppata, alcune modifiche sono fatte su nodi ROS di PMK.

Chapter 1

Introduction

When we think of our daily lives, even very simple tasks, such as navigating around safely and easily, can be a big issue for people with different types of impairments. Most of the time these problems are ignored and everything is built for fully functioning human bodies. As a result, people having disabilities are being dependent on others to overcome these issues and this dependency lowers the quality of their lives.

As the society realizes the problems of disabled people, some solutions have been tried to ease their lives. These solutions try to enable people safely accomplishing their daily life activities without the need of other people. Solutions might include changing the environment and creating disabled friendly buildings. With the help of technology, disabled people are being supplied with necessary equipment to overcome these issues.

Most of the time, people with different types of motor disabilities need assistance for mobility. Autonomous wheelchairs try to solve mobility problems for these people by assisting them to navigate. This assistance could be semi-autonomous or autonomous depending on the level of users' involvement in the process of navigation.

For both autonomous and semi-autonomous navigation of wheelchairs, user needs to be informed about the position, speed, direction, status of the wheelchair (moving, idle, waiting… etc.). Users should also be able to configure and use the

wheelchair easily and safely through an interface. This interface should be simple and easy to increase the usability of the wheelchair.

There are some projects under development by Politecnico di Milano to solve problems of disabled people or elderly. *PMK (Personal Mobility Kit)* [3] is the software system for a robotic powered wheelchair developed in *LURCH (Let Unleashed Robots Crawl the House)* project. *LURCH* intends to solve navigation problems of people with motor disabilities by creating a system with software and hardware technologies.

PMK concerns autonomous robotics, a branch of robotics that deals with study and design of vehicles able to fulfil tasks without the need for human intervention. It has capabilities of moving wheelchair autonomously on a given map, estimating odometry data (position, rotation and velocity) of the wheelchair, and detecting the obstacles around it using sensors.

ALMA (*Ageing without Losing Mobility and Autonomy*) project [2] aims to create a personal navigation assistant for elderly and people with different types of disabilities. *PMK* developed by LURCH project will be combined with other modules of ALMA project to create a complete solution for helping disabled people and elderly to live safely and more independently.

Mobile devices are in a big part of our daily activities. With the help of mobile devices, many difficulties are now easily solved. Especially for people with different types of disabilities, many daily life tasks are easier using mobile devices. Mobile devices can be used for robotics for supplying a familiar interface to users so that users can easily and safely access to developed technologies. In this thesis, mobile application development is combined with robotics field in order to create an easy to use interface for people with motor disabilities to make the use of autonomous wheelchair easier and safer.

In this work, a mobile application is proposed as the base of further developments to create a complete user interface for *PMK*. The purpose is to create a familiar and easy to use interface for the existing system so that people with motor disabilities can use the *PMK* easily and safely.

The aim is to increase the usability of wheelchair by supplying a user interface that can be used and accessed easily. The intended users of the application are people with motor disabilities who are using the developed wheelchair. The application is also a proxy for *LURCH* and *PNA (Personal Navigation Assistant)* module of *ALMA,* which aims to create a personal assistant for elderly people or people suffering from different kinds of disabilities.

The project intends to use mobile devices to give users feedback about the movement state of the wheelchair and information about the obstacles around the wheelchair. It also offers possibility to manage the configurations of wheelchair easily by using an android device. The application can be used as a standalone application with *PMK (autonomous wheelchair)* without the need of connecting to *ALMA* project.

Application offers users use of mobile devices for (Figure 3):

- Downloading IndoorGML maps from the *ALMA* servers
- Converting downloaded maps into convenient format to be used by the wheelchair
- Connecting to wheelchair and getting information about map used by wheelchair
- Changing the map used by wheelchair with one of the maps from the device
- Visualizing the odometry information (position, direction on the map)
- Visualizing the sensor information (laser scan detecting the obstacles around and the giving the distances from wheelchair)
- Visualizing the wheelchair movement

The application was developed taking into consideration the latest software structure of the wheelchair. In the latest software structure [3], *ROS (Robot Operating System)* is used for providing the wheelchair with autonomous features like path planning and collision avoidance, while keeping it safe for both users and people around it.

Beside the mobile application developed, some modifications are made on the software running on the wheelchair in order to enable some tasks.

This work is going to be a base for the further developments for creating a complete user interface to *PMK*. The final interface will be merged with the user interface developed for *PNA*.

The structure of this document is below:

- Chapter 2 gives some background information about two projects (*ALMA* and *LURCH*) briefly. The purpose of these projects and the connection of this work with them are explained in detail.

- Chapter 3 describes the software architecture, along with the explanation of all design choices and specifications of the developed application.

- Chapter 4 explains software project in detail including the implementation of project and the software and technologies used for the development of the application.

- Chapter 5 explains how to set up a test and development environment in detail as a guide for the further development of the project and a user manual is included for users with some screenshots from the application.

- Chapter 6 draws conclusions, and some possible extension and improvement suggestions are suggested for project.

Chapter 2

Background Information

This work is built on top of these two projects. This chapter describes briefly these two projects (*LURCH and ALMA*). The general purposes of these projects are included as well as the detailed description of the parts related to application developed within this thesis.

2.1 LURCH

In 2007 the *AIRLab (Artificial Intelligence and Robotics Laboratory)* of *Politecnico di Milano* developed a robotic wheelchair called *LURCH (acronym for Let Unleashed Robots Crawl the House)*. It is based on a commercial wheelchair and implements both semi-autonomous and autonomous modes [3].

The first version of *LURCH* was equipped with [4]:

- 2 laser scanners in the front, each scanning the environment at 240 degrees;
- an IMU (Inertial Measurement Unit) to measure velocities;
- a camera that detects artificial landmarks;
- an on-board x86 computer;
- a touchscreen that helps the user interacting with the computer.

The wheelchair, basically, could detect obstacles by means of the two laser scanners, localize itself with the help of the camera, and have a velocity feedback given by the IMU. In semi-autonomous mode, the wheelchair could be driven with the on-board joystick or a joypad alike; the software was able to facilitate the user's movement, avoiding collisions [3].

In autonomous mode, the user could select a goal position through the touchscreen; the control software had the task of computing the path to reach the goal and executing it by sending the proper commands to the motors. In order to make the computer give commands to the motors, the connection between the joystick and the motors was cut, and an interface circuit was realized and put in the middle. Such board, basically, has two functions [3]:

- it reads the joystick positions and it sends them to the on-board computer;
- it translates the commands sent by the computer into voltage values that control the motors.

A peculiarity of this solution is that the joystick still keeps its functionality, as its position can always be known and employed, and it can be used in conjunction with other input devices [3].

Over the years, some changes to *LURCH* have been made. In particular:

- the *IMU* was removed, for its low accuracy in velocity measurements
- and for its high costs;
- the velocity feedback is now given by encoders mounted on the wheels;
- an odometry board was made in order to interface the computer with the encoders.

In the latest software structure configuration, *ROS (Robot Operating System)* [3] is used for providing the wheelchair with autonomous features like path planning and collision avoidance, while keeping it safe for both users and people around it. With its publish-and-subscribe paradigm and high portability, this framework improves extensibility and reuse of software modules. Moreover, the issue of robot localization has been studied.

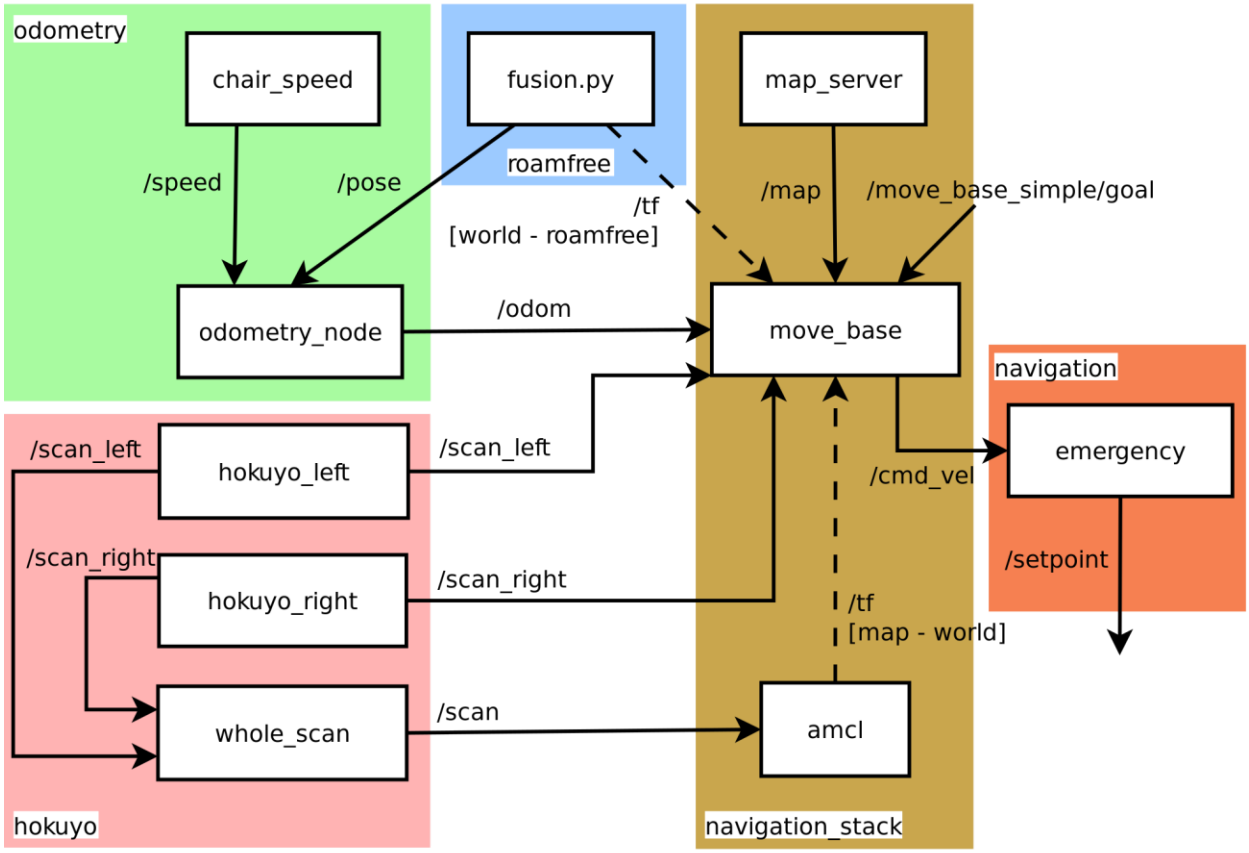

Figure 1: The autonomous drive architecture of the LURCH wheelchair

To this end, a new library for multisensor fusion and pose estimation, called *ROAMFREE (Robust Odometry Applying Multisensor Fusion to Reduce Estimation Errors)*, has been used. *ROAMFREE* fusion engine allows to merge odometry data coming from different sensors, in order to provide an estimate for the robot pose which is robust, meaning that it is less prone to errors. This method has been combined with an algorithm known in literature as *AMCL (Adaptive Monte Carlo Localization)*, in order to increase the robustness of the estimate by compensating, in many cases, the absence of absolute position sensors.

This thesis offers an android mobile application as the interface of *PMK* developed in *LURCH*. It will be merged with the user interface of *PNA* module from *ALMA* project.

In the development of the application, some of the ROS nodes running on the wheelchair are used by connecting via android device to get the published map, odometry information, and sensor information in order to give feedback to users. Moreover, a ros node (*map_server*) has been modified in order to fulfil some of the requirements. (More information about *ROS* and about the modification made on the *map_server* will be given in the Chapter 4.1 and Chapter 4.2 respectively).

The nodes used by the application are mainly *map_server* and nodes publishing odometry messages and laser scan messages.

Map server is used for publishing the information about the map data. With the modified version of the map server, it is also possible to update the published map on wheelchair with a new map from mobile device while the system is running without the need of restart. This map is used by the other nodes for navigating autonomously and detecting obstacles. In the mobile application, these maps are visualized on devices screen.

Odometry messages are published for giving data about the position, direction, and velocity of the wheelchair. Transform configuration of the map according to the real world is hold on a tree with tf which is a package that lets the user keep track of multiple coordinate frames over time. tf maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc. between any two coordinate frames at any desired point in time. These odometry messages are used for visualizing the position, direction, and velocity of the wheelchair.

Laser scan messages are used to publish the data about obstacles around the wheelchair. Obstacles are detected with the lasers on the wheelchair. Laser scan is used for autonomous movement. In the mobile application, this data will be visualized for users to give feedback about the obstacles around the wheelchair.

For the implementation of the connection to wheelchair from android application, "*rosjava for android*" has been adopted. (rosjava will be explained with more details in Chapter 4.1).

2.2 ALMA

ALMA [1] is an international project within the *AAL Ambient Assisted Living Joint Programme*, started on April 2, 2013 and involving the following partners:

Scuola Universitaria Professionale della Svizzera Italiana (Switzerland),
Politecnico di Milano (Italy),
Infosolution SpA (Italy),
VCA Technology Ltd. (UK),
Istituti Sociali di Chiasso (Switzerland),
Clinica Hildebrand (Switzerland),
University of Wuerzburg (Germany),
Degonda SA (Switzerland).

The aim of *ALMA* is to combine a set of advanced hardware and software technologies into an integrated, non-invasive and modular system in order to offer assistance for people affected by different types of impairments. In the context of *ALMA* end-users can be either patients or a healthcare structure.

ALMA project tackles the issue of not being able to move autonomously or effectively by combining a set of advanced hardware and software technologies into an integrated and modular system composed by [2]:

(i) an indoor localization system based on a network of low-cost/low-power RF emitters, to provide room level localization of people and objects;

(ii) an ad-hoc, autonomic hw/sw system based on networked smart cameras providing accurate indoor and outdoor localization and environment monitoring;

(iii) an intelligent system for the online planning of users' paths according to their specific needs, matching these with the actual state of the environment and the available resources;

(iv) a personal mobility kit for electric powered wheelchairs allowing them to perform automatic or assisted navigation and, additionally, to interact with the surrounding environment; (*PMK*)

(v) a personal navigation assistant supporting user-friendly interface to all the functionalities of the system, tailored to the specific user-defined requirements and physical limitations (e.g., vocal and tactile interfaces, ad-hoc devices). (PNA)

ALMA users [2] will be supported in their mobility to acquire knowledge about interesting locations (e.g., services, people, facilities, etc.), to select and follow an efficient and safe path to such destinations considering their needs and/or limitations or the status of the environment to present the resources provided by intelligent environments to the users so they can effectively access them with familiar instruments without feeling disoriented or overwhelmed by technology. *ALMA* aims at bringing to the mobility of a wide range of primary end-users a real advancement that will be measured in terms of the most appropriate metric: their own feeling of freedom and increased empowerment. At the same time secondary end-users, e.g., residences and hospitals, will leverage on the information collected by the system on the movements of their guests to monitor their ageing, to design personalized support services or rehabilitation paths.

The *ALMA* project developed a modular system of hardware and software components that can support or enhance the autonomous motion of people with different degrees of mobility and/or cognitive impairments. Each module of the ALMA system provides standalone functionalities that can be used to address individually one of the three mobility issues previously introduced, i.e., destination selection, path planning, and movement execution. The architecture of the whole system is shown in Figure 2.

This thesis proposes a mobile application as the user interface for the module which is a personal mobility kit for electric powered wheelchairs allowing them to perform automatic or assisted navigation and, additionally, to interact with the surrounding environment (*LURCH*). This application will be improved to be a complete user interface for *PMK,* and the final application will be merged with user interface for indoor navigation (*PNA).*

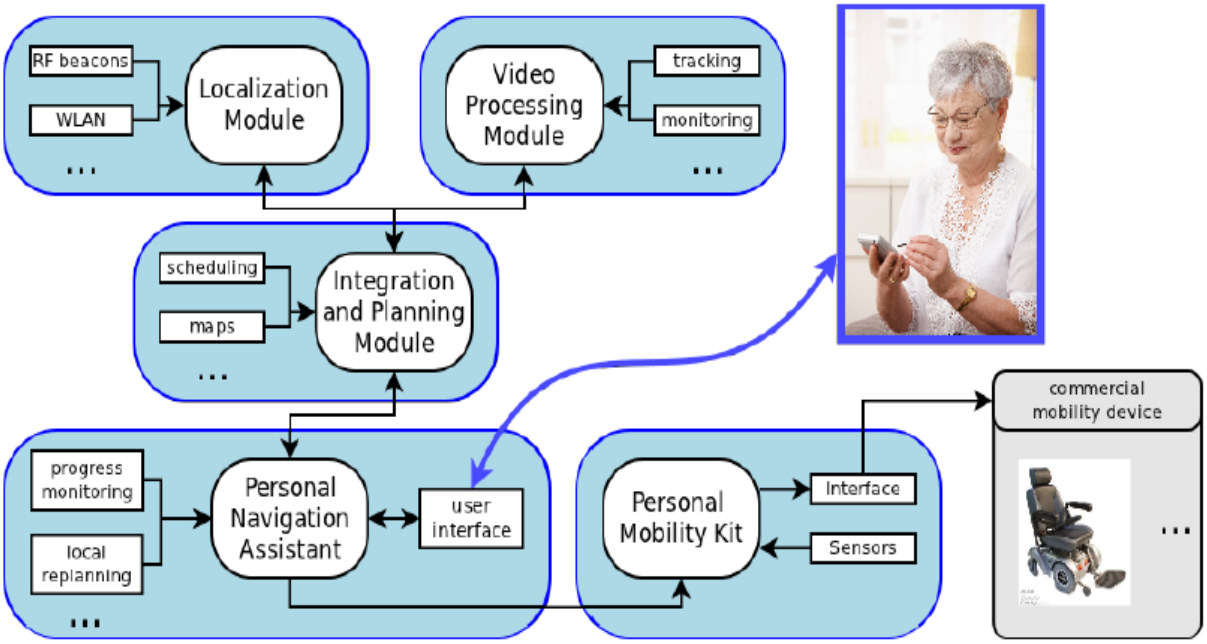

Figure 2: Functional modules of the ALMA system and their relationship

In *ALMA*, maps are specified as *IndoorGML* an *OGC* standard that extends the *GML (Geographic Markup Language)* with an application schema for indoor spaces. It takes inspiration from *CityGML* which is a recently introduced *OGC* standard to describe cities at various levels of detail. In particular, *CityGML* at Level-of-Detail 4 (LoD 4), provides classes to describe the geometry and semantics (door, room, floor, furniture, ...) of the interior of buildings.

The vision behind the *IndoorGML* standard (and other indoor maps development) is that in the near future, when we enter a building with a smart phone, we will be able to download a map, visualize it and get navigation instruction to the place we are looking for. The maps will contain enough information to be useful for people (and other agents) with different capabilities. *IndoorGML* will be explained with more details in Chapter 4.1.

The application connects to *ALMA* server, which stores the *IndoorGML* maps through any internet connection (wifi, 3g). Users download the map with the entered url into a specific folder on device to be parsed and used later on for generating the map in the proper format to be sent to wheelchair as well as for the visualization on the device. The format necessary for the wheelchair is *YAML* format with an image containing

occupancy data (occupancy grid map). The *IndoorGML* map is parsed and occupancy grid map is generated with resolution value decided by users.

Since the map is downloaded through a url entered by the user, the server could be any server publishing a proper *IndoorGML* map for connecting and downloading the map. Therefore, the application developed for this thesis does not depend on the *ALMA* server directly, but maps from server can be used. Other capabilities of the application can be used as a stand-alone application with *PMK*.

Chapter 3

System Architecture and Software Specifications

In this chapter, general architecture of the project is given including the relations with the *ALMA s*erver and *PMK*. The specifications of the application are described with some basic information. The specifications can be outlined as follows:

- Download an *IndoorGML* map from the *ALMA* server

- Parse the *IndoorGML* map to generate the maps in needed format

- Send the map to the wheelchair

- Retrieve the map metadata

- Visualize the wheelchair movement and the laser scan on mobile devices screen

3.1 System Architecture

As it was stated before, this thesis is working on the top of two projects previously developed by *Politecnico di Milano*, namely *ALMA* and *LURCH*. The overall architecture of the project and relations with the other projects can be seen in Figure 3.

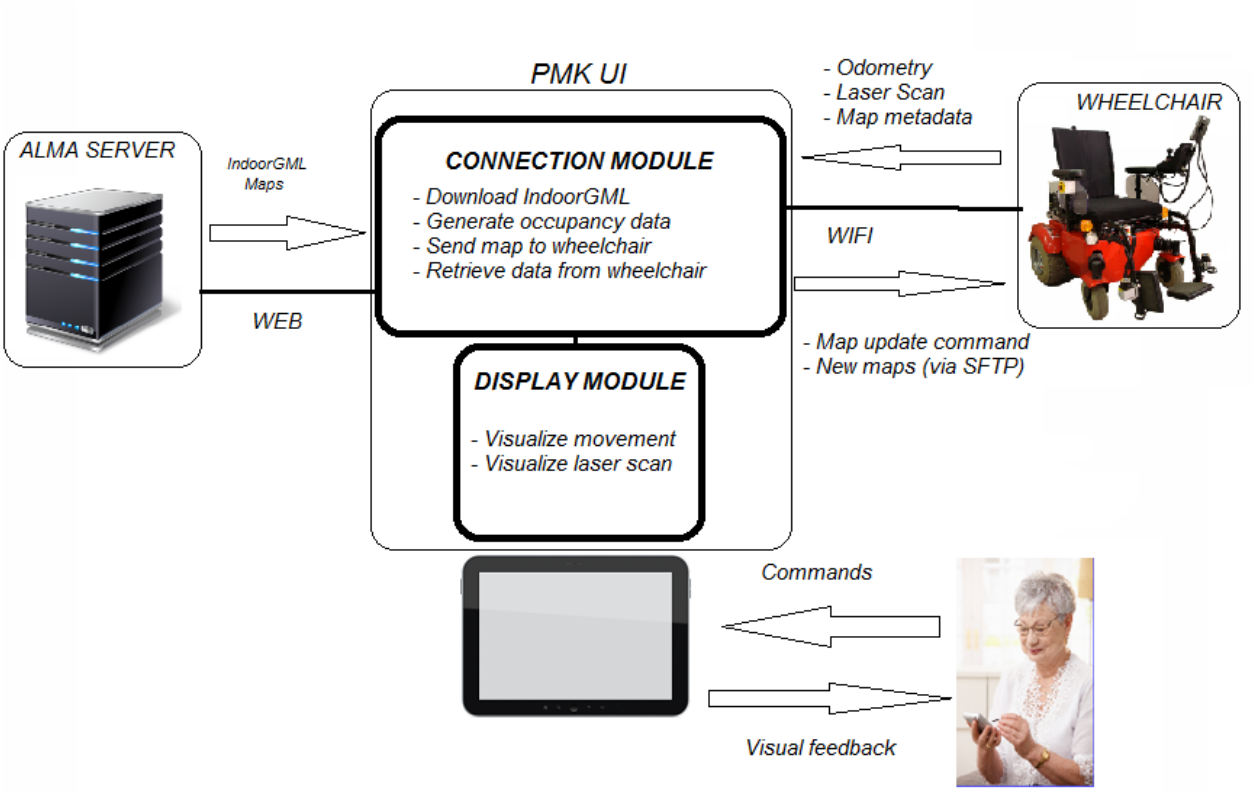

Figure 3: Architecture of the project and relations with other projects

The relation with *ALMA* project is not a mandatory one but a supportive one in the sense that the application developed for this thesis can also function without any connection to *ALMA* servers. *ALMA* server is used only for getting the *IndoorGML* maps as the source files to generate the occupancy grid maps to be used by wheelchair and for visualization purposes on the screen of the device. *IndoorGML* maps can be transferred to device manually or can be downloaded also from another server publishing maps online via a public url.

The format of the *IndoorGML* used as the source to generate occupancy data for *PMK* is considered as equal to the ones on *ALMA* server. For example, the resolution of source maps is assumed as 100 pixels per meter when generating the occupancy data.

The connection with *LURCH* project is mandatory for the application. This thesis is based on the implementation and structure of *PMK.* Therefore, the application itself, without the wheelchair, is not useful for the users. The connection to wheelchair is a

TCP/SFTP connection meaning that mobile device running the application should be connected to the same local network to be able to work properly.

The data for establishing a connection to the wheelchair is entered by user and saved for later use. This data include:

- *IP* of the wheelchair and the port for *ROS*

- Wheelchair node name

- Topic names of odometry layer and laser scan layer

- Username and password of the *SSH* protocol running on *PMK*.

When connected to wheelchair, users can:

- Send a map to wheelchair in order to change and reload the map used by the wheelchair

- Get the information about the current map, get the position, direction, and velocity of the wheelchair on the published map

- Get the sensor information from wheelchair in order to visualize it and give users an idea about obstacles around the wheelchair.

The application will be improved with some new features to have a complete user interface for *PMK* and will be integrated with user interface of *PNA* in the future.

3.2 Software Specifications

3.2.1 Download an *IndoorGML* map from *ALMA* servers

IndoorGML maps contain the necessary information for generating occupancy data of the maps needed by *PMK. In this work, IndoorGML* maps are proposed as the source of occupancy data map generation for *PMK*. The application allows users to

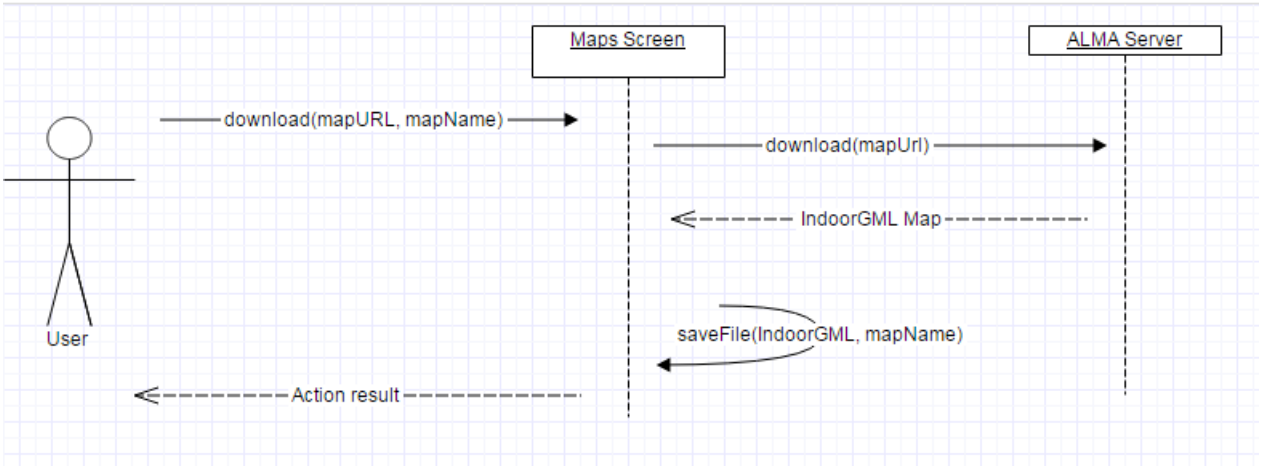

Figure 4: Sequence diagram for download IndoorGML Map

download *IndoorGML* maps published by the *ALMA* servers through a url. The url of the map to be downloaded should be entered by the user in the specified box. Moreover, the name for saving the file should be entered. File is downloaded into the folder*(ALMA_MAPS)* containing maps and files used by the application.

In order to have a successful download, mobile device should be connected to internet using 3g or wifi .

If *IndoorGML* url typed by user does not correspond to a proper *xml* file (IndoorGML file is a specialized type of xml file) or if there is a problem with the internet connection of the device, download action fails and user is informed with an alert dialog about possible reasons.

If there is already a map in the folder with the same name entered by user, the old file will be replaced with the new map or a new map will be created with the given name after user confirmation.

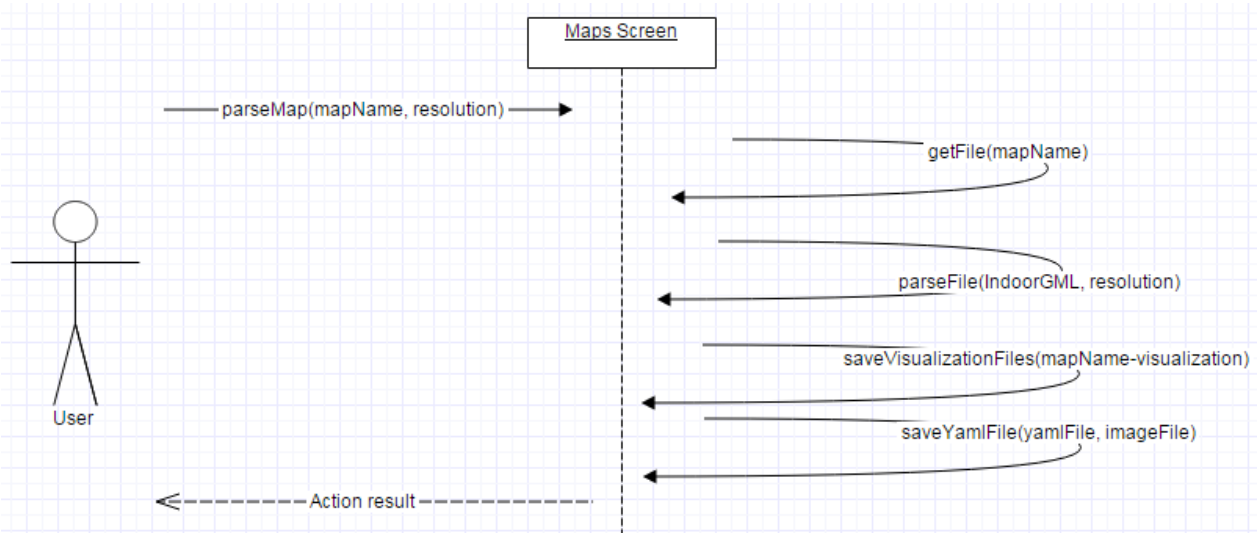

Figure 5: Sequence diagram for parse IndoorGML

3.2.2 Parse *IndoorGML* map to generate the maps in needed format

The previously downloaded or manually added *IndoorGML* maps can be parsed from the *ALMA_MAPS* folder on device in order to generate the maps in *YAML* format. *YAML* format contains an image file holding occupancy data and a text file holding metadata of the map. This format is used by *PMK* as the maps used for necessary tasks.

After chosing the name of the map to be parsed from the *IndoorGML* file list, which shows the *IndoorGML* files under *ALMA_MAPS* folder, user needs to enter the resolution value to be used in the creation of the occupany map.

If the *IndoorGML* file format is not correct, parse action fails. If the size of the image file to be created is too large, meaning that *IndoorGML* corresponds to a big map having large coordinate values, mobile device might be out of memory and application crashes. In this case, users are asked to use a smaller resolution value.

The parsed *IndoorGML* file is kept in *ALMA_MAPS* folder of device to be parsed again with a different resolution value if it is needed in the future. If a map is parsed again, old generated files for that map will be replaced with the newly generated ones.

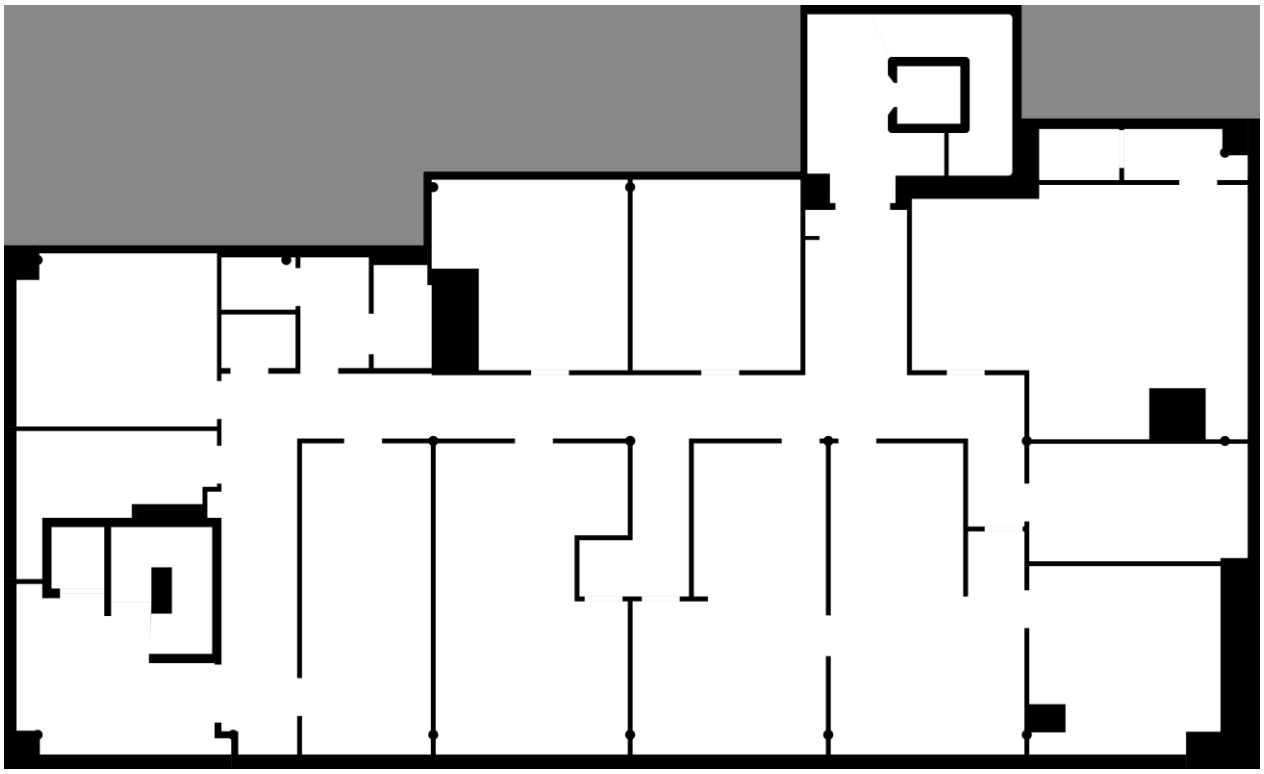

Figure 6: Example generated image containing occupancy data

When generation of the files is finished, there will be a pair of files with the same name, in png (See Figure 6) and *YAML* formats (*name.png* and *name.yaml*). Beside these files there will be another image file generated with a visualization suffix following same name (*name-visualization.png*). This file is going to be used for visualization of the map on devices screen. All these three files will be placed in the *ALMA_MAPS* folder containing also IndoorGML files previously downloaded. Two of these files are in the format to be used by map server of the wheelchair and can be sent and directly used by wheelchair.

The generated image file to be sent to wheelchair will have the resolution chosen by user, but the visualization images have a fixed resolution (10pixels / meter). *IndoorGML* map files downloaded from *ALMA* servers have coordinates such that 1 pixel corresponds to 1 centimeter meaning a resolution of 100 pixels / meter.

```
image: map.png
resolution: 0.1
origin: [ 0, 0, 0]
occupied_thresh: 0.65
free_thresh: 0.196
negate: 0
lla_origin: 45.801324, 9.092891, -151.3
```

Figure 7: Example generated YAML file content

Two of the generated files are in png format with different resolution values, and these files encode occupancy data. One of these files is used for map visualization on devices screen. When user chooses to send a map to wheelchair, the other image file will be sent to wheelchair and wheelchair will be using this new map after this action.

The image generated describes the occupancy state of each position on the map with the color of the corresponding pixel. Thresholds in the *YAML* file are used to divide space occupancy to three categories as given in Figure 7. Image contains three colors:

- white pixels mean space is navigable
- black pixels mean space is occupied
- gray pixels mean unknown area

One of the generated files is in YAML format, which is a simple text file describing the map metadata and holds the name of the image file holding occupancy data. Beside the image file name, there are some other fields to describe the map such as resolution, origin, etc.

Required fields of *YAML* file used by *PMK* are:

- image: Path to the image file containing the occupancy data; can be absolute, or relative to the location of the *YAML* file

- resolution: Resolution of the map, meters / pixel

- origin: The 2-D pose of the lower-left pixel in the map, as (x, y, yaw), with yaw as counterclockwise rotation (yaw=0 means no rotation). Many parts of the system currently ignore yaw.

- occupied_thresh: Pixels with occupancy probability greater than this threshold are considered completely occupied.

- free_thresh: Pixels with occupancy probability less than this threshold are considered completely free.

- negate: Whether the white/black free/occupied semantics should be reversed (interpretation of thresholds is unaffected).

An extra field, which is not required by *PMK* is included in YAML file with the generation method implemented in this work:

- **lla_origin:** This field holds the translation and rotation of the map with respect to world coordinates which is taken from the *IndoorGML*.

3.2.3 Send map to wheelchair

Users can use the application to send a previously generated or manually transferred map to wheelchair and refresh the modified version of the map server node on wheelchair to start using this new map.

When user choose to send one of the map names from the list of maps, the generated pair of files are transferred to wheelchair via *SFTP* connection established using the settings configured by user. After wheelchair receives files the modified version of the map server is reloaded with the newly received map info. To reach this goal a service provided by the modified version of map server is called from mobile device If there already exists a map in the wheelchair with the same path and name, the old files are replaced with the new pair of files.

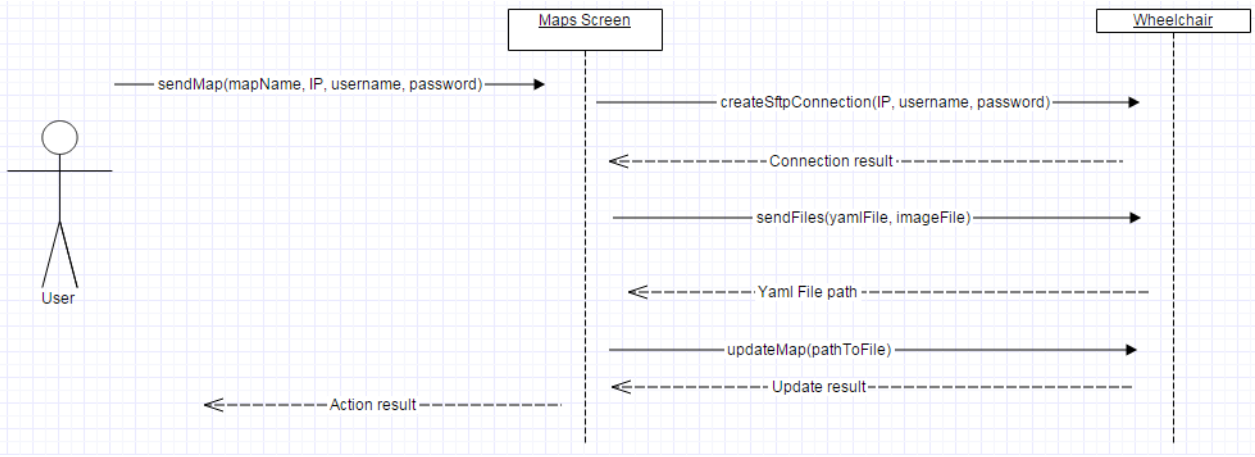

Figure 8: Sequence diagram for send map to wheelchair

In order to have a successful send and reload action, wheelchair and device should be connected to the same *LAN(Local Area Network)* and IP, port, username, password of the wheelchair should be properly set in the settings section of the application.

This action assumes that the modified version of the map server node is already running on the wheelchair. If modified version of the map server node is not running, this action will not have any effect on the published map even though it transfers the files to wheelchair. (More information will be given about the modified version of the map server in Chapter 4.2).

3.2.4 Retrieve map metadata

In the settings section of the application, users are asked to fill the necessary information for establishing connection to *PMK.* These values are used for retrieving information from wheelchair and making service calls to update map published on *PMK*.

Users can retrieve the published map information from the map_server node which is running on the wheelchair. This feature is used to test and verify the saved values of IP and port for the wheelchair. After saving the IP and port number in the settings page, users can try connecting to wheelchair. The verified values of IP and port

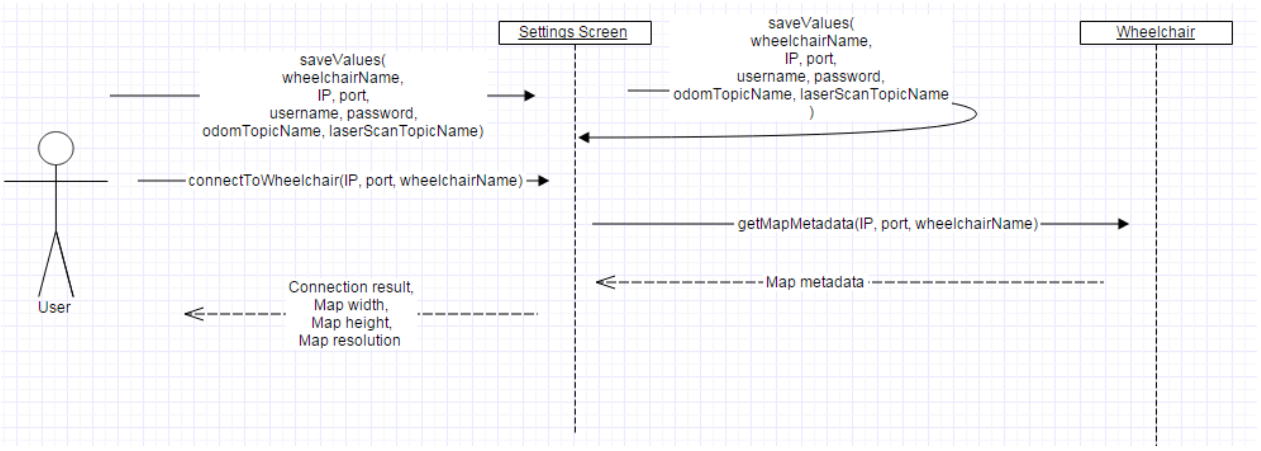

Figure 9: Sequence diagram for retrieve map metadata

will be also used in the other sections of the application where application needs connection to wheelchair such as retrieving data from a published topic or making a service call to update map published on *PMK*.

To be able to retrieve the map information, device should be connected to the same LAN with wheelchair. When user connects to wheelchair, the information about the published map from the wheelchair is seen on the screen and status seen on the screen changes from disconnected to connected.

3.2.5 Visualize wheelchair movement and laser scan

Application offers a way for visualization of the odometry information and laser scan information published from *PMK*. This information is used for showing position, direction, movement of the wheelchair on the map and laser scan giving information about the obstacles around the wheelchair.

The movement of the wheelchair is visualized on the device with a simple implementation (See Figure 11). On the main screen user will see the map name chosen for visualization. To change this map name user should choose a map and send it to wheelchair from the maps screen. After sending a map and going back to home screen, the name of the map will change from default chosen name to the sent one. Since the map is just sent to wheelchair, the map used by the wheelchair is known and correct map can be visualized on the device.

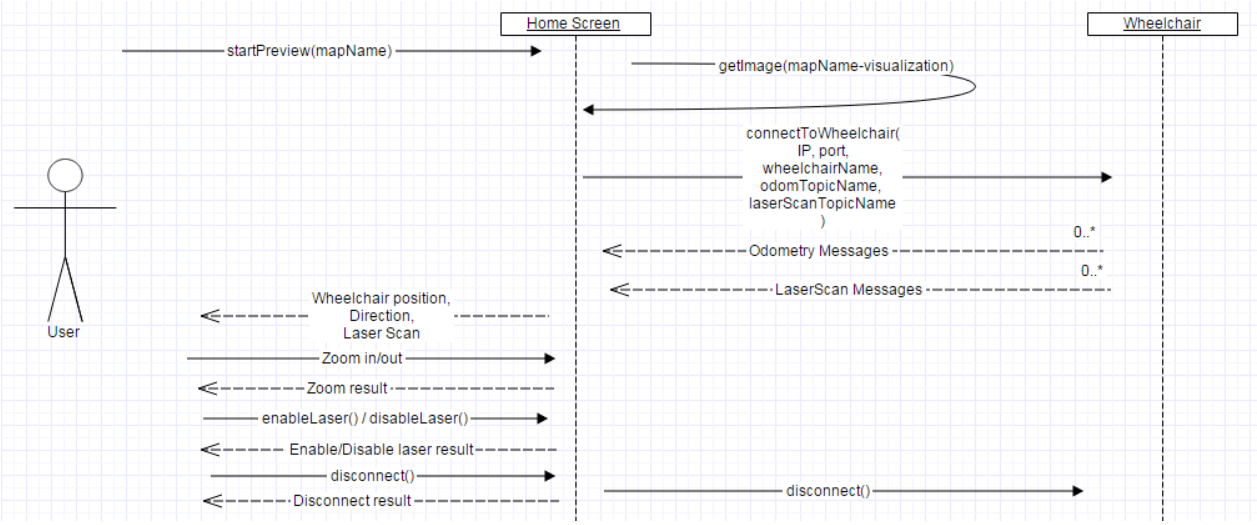

Figure 10: Sequence Diagram for visualization

If there is visualization image file containing the occupancy data for the given map name in *ALMA_MAPS* folder, users see a preview button on the main screen. When users click on this button, the correct map image is retrieved from the reserved folder. Users can see the value of zoom, which can be changed by pinch gestures on the imageview or with buttons on the upper right corner of the screen.

Application retrieves odometry messages from the wheelchair and shows the position of wheelchair on x y plane, as well as the direction of the movement. Two dots on the map simulating the position and the velocity (direction) of the wheelchair are drawn with different colors. For successful visualization of the wheelchairs movement, the device should be connected to the same local network with the wheelchair and values for wheelchair name, odometry topic name should be configured correctly.

When preview is running, users will have an option below disconnect button for toggling the sensor visualization on/off. User should correctly configure the name of the laser scan topic to be shown from settings screen to have a successful visualization.

When the preview is started, sensor visualization will be started as well. To hide the laser scan checkbox below the Disconnect button can be used. Laser scan messages created by sensor on wheelchair and published by *PMK* will be used to visualize the laser scan data on the map.

Users can zoom in or out with pinch gesture on the image or by using the buttons on the upper right corner. By default map is shown without zoom meaning whole map is shown at the beginning. If the map is zoomed, the part of the map to be shown decided according to the position of the wheelchair on the map.

If the wheelchair is not close to the boundaries of the map, a blue colored dot representing the wheelchair is shown in the middle of the map. As wheelchair moves around, the visualized part of the map is also changed so that wheelchair is always in the center of the map.

If the wheelchair is close to the boundaries of the map, wheelchair is not placed anymore in the middle of the visualized part of the map. The width and the height of the shown map are kept the same for that zoom value. Starting from the boundaries that wheelchair is close to, part of the map within the width and height of the map is shown and the point for representing wheelchair is moved around this part of the image.

Beside the current position of the wheelchair, direction of the wheelchair is also represented on the map with a smaller dot colored in yellow.

Laser scan data is represented by drawing red lines for each ray. For each ray starting from the position of the wheelchair and ending at a point calculated according to the distance of obstacles retrieved from laser scan and direction of the each ray.

The coordinates of the wheelchair are also shown as text on the screen and updated as wheelchair moves. Zoom value is also updated if it is changed by user while preview is running. The visualization can be stopped by clicking Disconnect button on the main screen and restarted later.

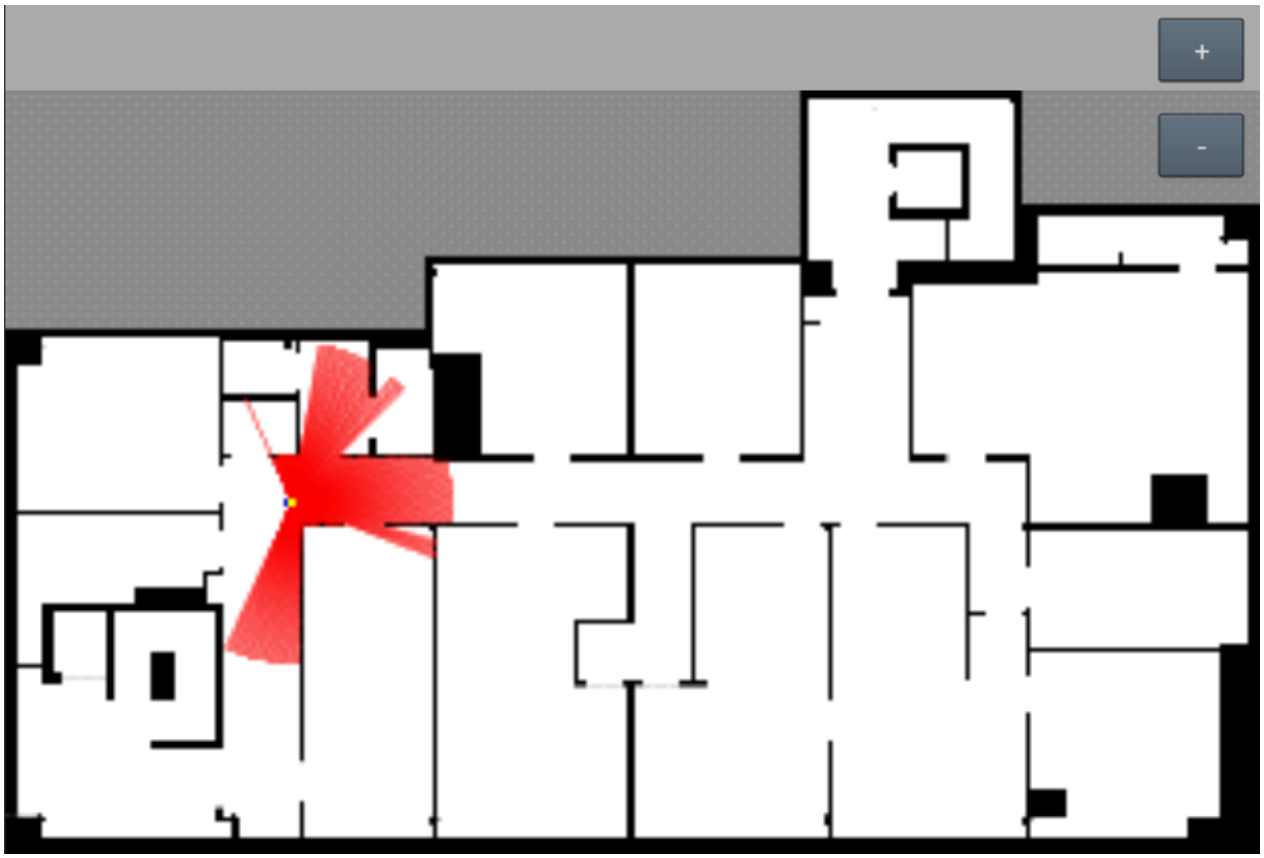

Figure 11: Example visualization of wheelchair and laser scan data

Chapter 4

Software Project

In this chapter, some background information is given about software and technologies used in the implementation of software project. File structure of mobile application, functionalities of the files, and implementation strategies are explained in detail. Modifications made on the map server running on the wheelchair are also included.

4.1 Used Software and Technologies

Software and Technologies used by the project are mainly:

- Android SDK and external libraries
- IndoorGML
- ROS
- ROSjava for android

4.1.1 Android SDK and External Libraries

In this work, android operating system is chosen as the mobile development environment. The minimum sdk version supported is 10 and screen layouts are prepared for tablets.

Different modules of the android sdk (software development kit) are used in the implementation of the application as well as some external libraries. Most of included

external libraries are necessary for making the application ROS*java* compatible and having ROS features in the application.

Used external libraries are mainly for:

- Parsing the *IndoorGML*(xml) files

- Connecting to *ALMA* servers

- Retrieving *IndoorGML* files from internet

- Sending generated maps to wheelchair via *SFTP* connection

- *ROS* enabling the application to be able to communicate with *ROS* framework used by *PMK*

The used permissions for the app are:

- android.permission.WRITE_EXTERNAL_STORAGE allows application to write external storage of the mobile device, which is needed for creating new files under devices file structure.

- android.permission.INTERNET allows application to connect internet using if the device is has access to internet. Internet connection is needed for downloading IndoorGML maps from ALMA server.

- android.permission.ACCESS_NETWORK_STATE allows application to check if the device is connected to wifi. This permission is needed to warn user if the device is not connected to wifi when application needs it to connect wheelchair.

- android.permission.WAKE_LOCK: allows application to keep the devices screen on while the application is running.

4.1.2 IndoorGML

As it was mentioned before, *IndoorGML* is used in *ALMA* project for storing the map information and it is used by the application developed within this thesis as an input for generating the occupancy grid map files.

IndoorGML is a candidate *OGC* standard for an open data model and *XML* schema for indoor spatial information. It aims to provide a common framework of representation and exchange of indoor spatial information. It is defined as an application schema of *OGC_Geographic Markup Language 3.2.1* [5].

The *IndoorGML* schema (data model) addresses the general problem of data exchange relevant to the indoor navigation of heterogeneous agents (pedestrian, disabled/impaired people, robots, motorized wheelchairs, ...). It was submitted as a proposal in September 2013 by an *OGC (Open Geospatial Consortium)* working group. It focuses on topological and semantic information and contains simpler geometrical information than that provided by buildings descriptions in other formats (like *IFC* and *CityGML*). Geometrical information can either be self-contained or refer to such external files.

The vision behind the *IndoorGML* standard (and other indoor maps development is that in the near future, when we enter a building with a smart phone, we will be able to download a map, visualize it and get navigation instruction to the place we are looking for. The maps will contain enough information to be useful for people (and other agents) with different capabilities [6, 7].

At the core, IndoorGML provides a multi-layered (topological) graph that can optionally contain geometrical information.

IndoorGML defines the following information about indoor space;

- Navigation context and constraints
- Space subdivisions and types of connectivity between spaces
- Geometric and semantic properties of spaces and connectivity
- Navigation networks (logical and metric) and their relationships

General concepts of IndoorGML [8]

- **Cellular space**: indoor space as a set of *cells*, which are defined as the smallest organizational or structural unit of indoor space.

- **Semantic representation**: Semantic is an important characteristic of cells. In IndoorGML, semantics is used for two purposes: to provide classification and to identify a cell and determines the connectivity between cells.

- Geometric representation: The geometry of 2D or 3D object may be optionally defined within IndoorGML according the data model defined by ISO 19107.

- Topological representation: Topology is an essential component of cellular space and IndoorGML.The Node-Relation Graph (NRG) represents topological relationships, e.g., adjacency and connectivity, among indoor objects.

- Multi-Layered Representation: A single indoor space is often semantically interpreted into different cellular spaces.

IndoorGML files contain 5 different layers (See Figures 12- 16 [23]):

- L1 – TOPOGRAHIC - Geometry
- L2 - TOPOGRAPHIC - Navigation
- L3 - SENSOR - Camera
- L4 - SENSOR - Localization
- L5 - TAGS - Semantic

In this work, only the first layer of the *IndoorGML* map, (See Figure 12), is used for the generation of occupancy grid map. In this layer, topographic geometry of the indoor environment is stored. Each polygon composing the overall map is defined by the corner points of the polygon. Semantic information such as doors are also included in this layer. For every polygon, it is stated if this polygon is a navigable space or not. There is also transition information in this layer but it is ignored in this work.

IndoorGML files used in this thesis have coordinates such that 1 point corresponds to 1 centimeter meaning that resolution of the map is 100 pixels / meter. In Chapter 4.2 process of interpreting *IndoorGML* file will be explained in detail with the specific tags used to retrieve the necessary information.

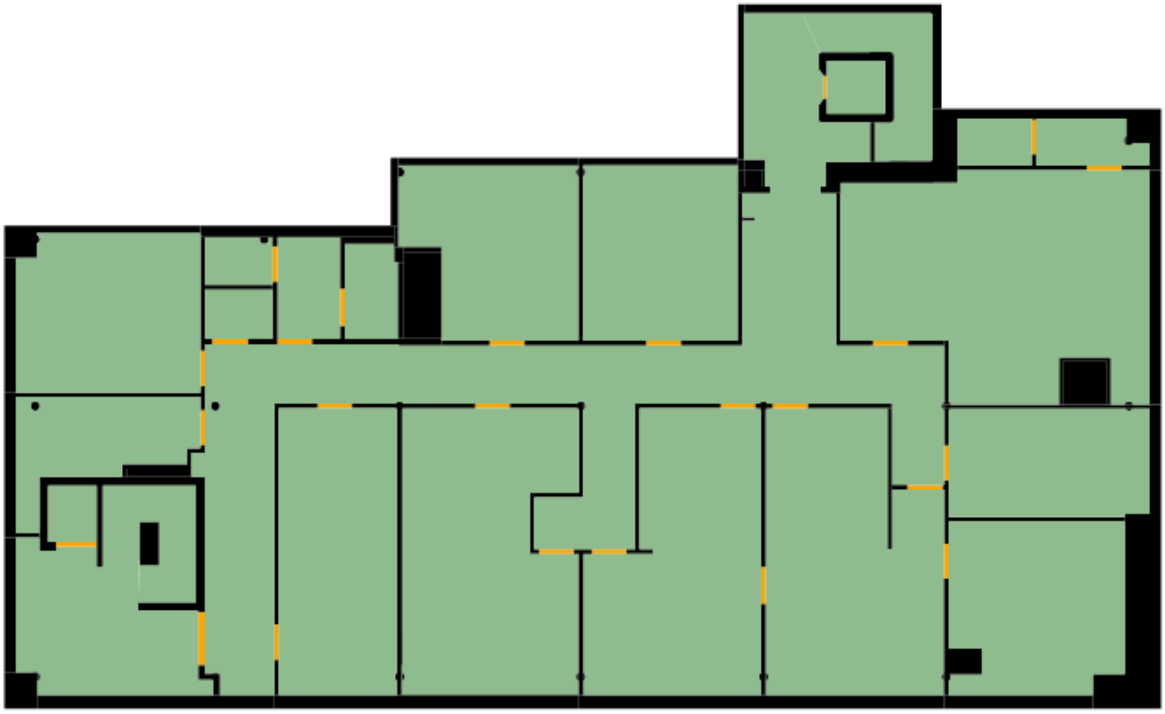

Figure 12: L1 - TOPOGRAPHIC - Geometry

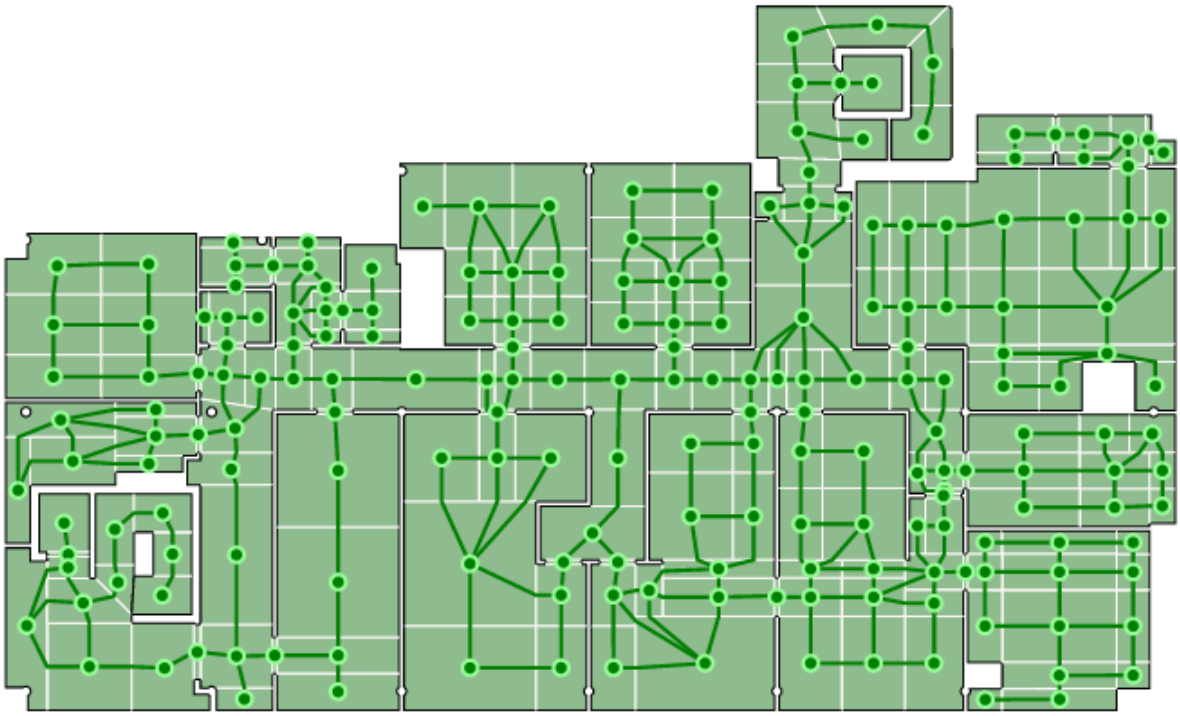

Figure 13: L2 - TOPOGRAHIC - Navigation

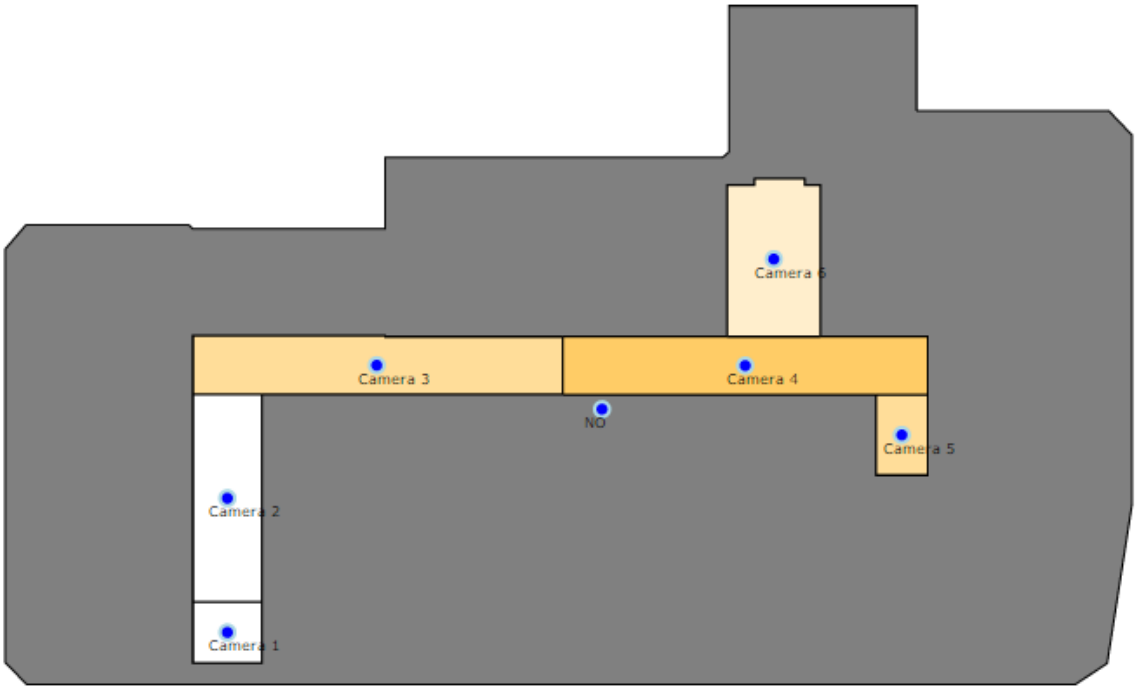

Figure 14: L3 - SENSOR - Camera

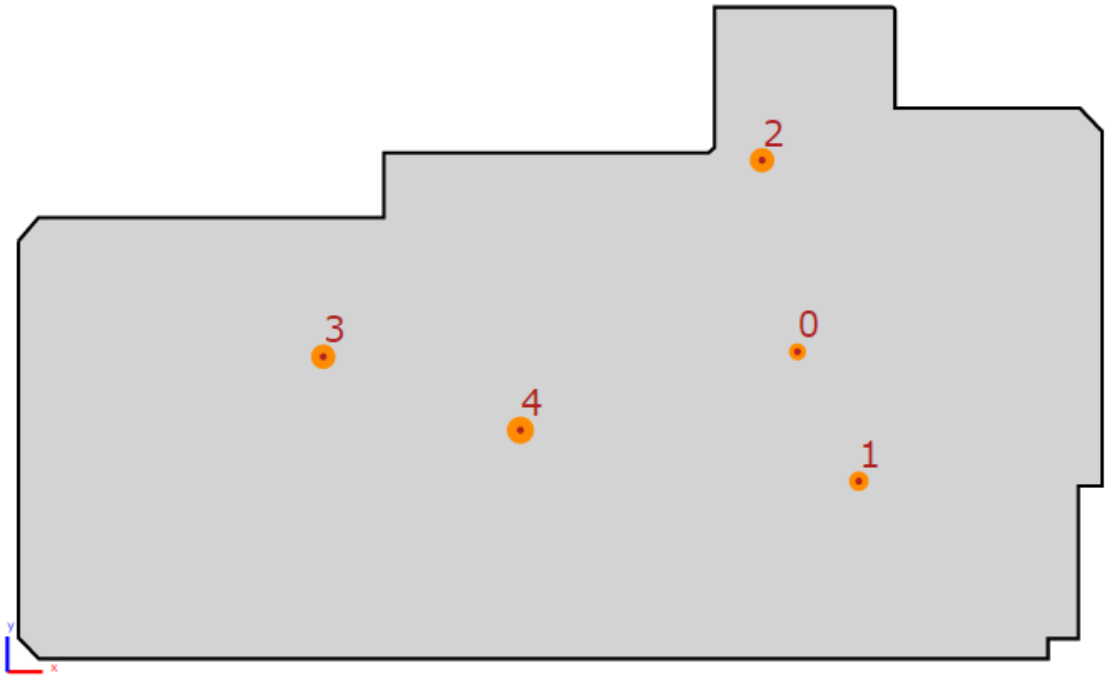

Figure 15: L4 - SENSOR - Localization

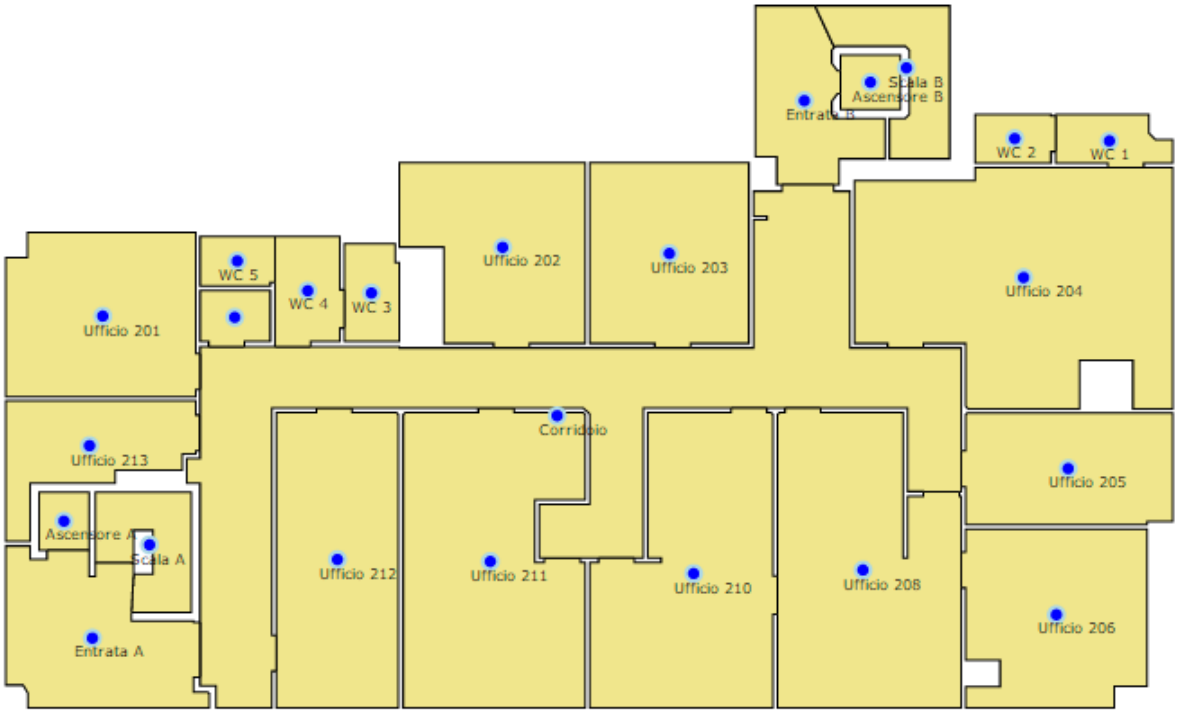

Figure 16: L5 - TAGS - Semantic

4.1.3 ROS

ROS (Robot Operating System) is an open-source operating system for robots, developed by the Stanford Artificial Intelligence Laboratory and by Willow Garage. More precisely, it is a meta-operating system, as it provides a structured communication layer above a host operating system. Its aim is to provide a general framework, suitable for the most common use cases in robotic software development [9].

a. Basic structure

A system built using *ROS* is made of a certain number of processes, potentially on a number of different hosts, connected at runtime in a peer-to-peer topology. Those processes are called nodes. In a typical robot application, each node is responsible for a specific task, often related to a particular part of the hardware. [3]

Nodes communicate with each other by passing messages. A message is a typed data structure. Standard primitive types, such as integer, float, etc. are supported, but programmers can also create custom messages and combine different types to produce more complex messages. A node sends a message by publishing it to a given topic. A node that is interested in a certain kind of data must subscribe to the proper topic. In general, many nodes publishing or subscribing to the same topic may exist, and a single node may publish or subscribe to multiple topics. Publishers and subscribers are not aware of each other's existence.

Although this topic-based model, which is founded on publish and subscribe paradigm, is very flexible and can be useful in many of the most common cases, it is not appropriate for synchronous transactions. To solve this problem, *ROS* provides the possibility to define a so-called service that is a pair of messages, one for the request and one for the reply. This is similar to what happens on Web services, which have request and response documents of well-defined types.

In order to let processes locate each other at runtime there is a module called master, which provides naming and registration services to the rest of the nodes in the *ROS* system. It tracks publishers and subscribers to topics and services. [10]

b. Main properties

The structure, made of independent nodes and messages, improves the reuse and extensibility of software projects. In fact, the encapsulation of code forced by this structure, makes it relatively easy to take a single node or a package, that is a set of nodes, from a project and put it into another project.

The only required effort is to adapt the new project to the interface of the retrieved nodes, namely message types and topic names, without having to touch their inner code. For these reasons, many generic nodes are provided, by the ROS team or by the community of programmers, and can be used in many cases directly out of the box or with little tuning. Among those, there is a variety of drivers for the most famous or common devices for robots, like sensors and input devices. The growing diffusion of

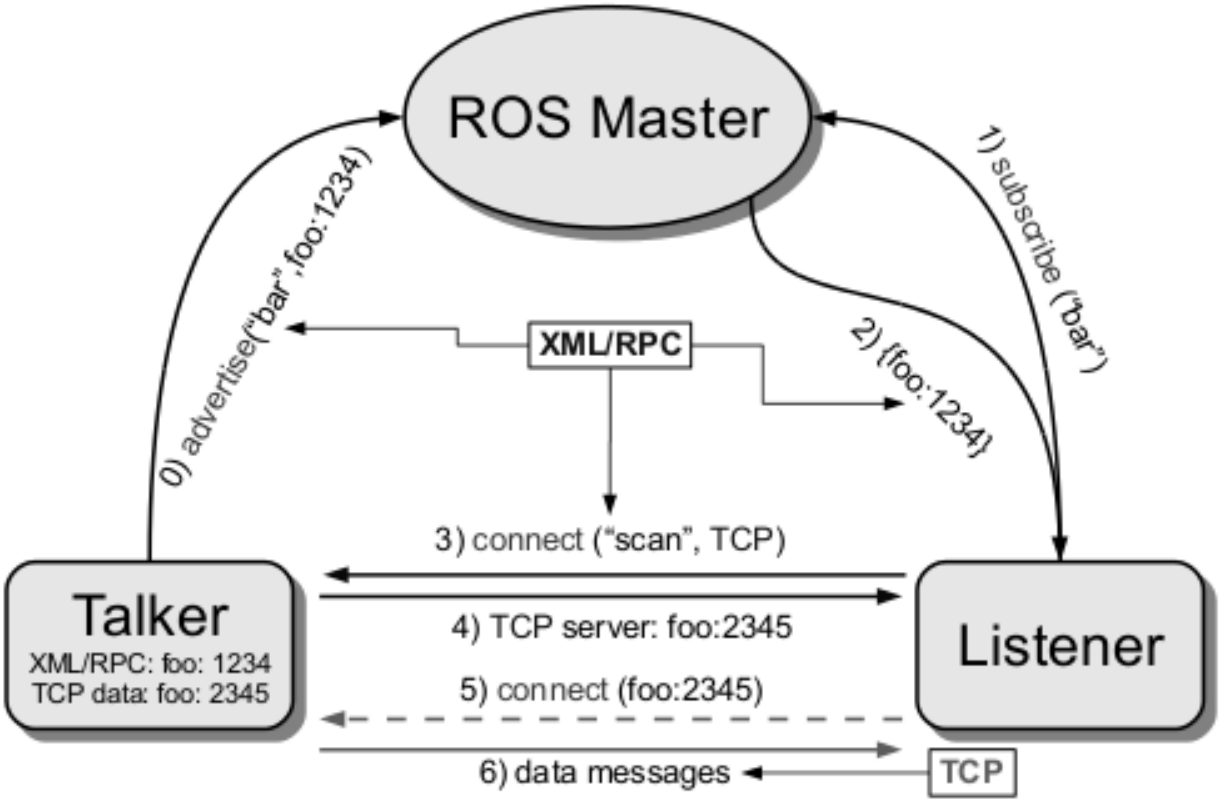

Figure 17: ROS basic structure [11]

ROS as a standard for robot software developing has increased the number of available solutions for many typical problems. [3]

Another important feature of ROS is that it allows communication between nodes written in different programming languages. This allows, to write some parts of the software in an interpreted language (like Python) to make those parts configurable and testable with less effort, and use more complex, compiled languages (like C++) to deal with tasks that have strict constraints in terms of time or memory consumption. ROS is not a monolithic development and runtime environment. On the contrary, it is composed of many small tools, able to perform various tasks. [3]

All these tools can be run by means of bash commands, so they are integrated in the normal operating system usage. These tools allow navigation through the source code tree, getting and setting configuration parameters, running single nodes or sets of

nodes, see which topics and nodes are running, visualizing messages published on a topic, and so on. This modular structure is useful when debugging, especially if the scope of investigation is a single part of the project, such as a single node. In fact, a node can be run, modified, and then rerun without having to restart the whole infrastructure: the graph composed by talkers and listeners is dynamically modifiable. ROS also provides specific tools for recording and playing back nodes, thus simplifying data analysis and research. [3]

Among the tools provided by ROS, an important role is played by graphic tools. Programmers can plot data and visualize graphs containing nodes, topics and relations between them. There is also a complete tool for data visualization, called rviz. This tool allows to view maps, reference frames, landmarks, planned paths, sensor data, and so on. In a nutshell, ROS allows to employ less effort in the coding and engineering parts, and to concentrate more on the core research. [3]

In this thesis, a few specific ros nodes and messages are used in the implementation to reach the goal intended. *map_server* node with some modification (will be explained in chapter 4.2) is used for publishing a map on the wheelchair and retrieving map info. Odometry messages are used to detect position, direction and velocity of the wheelchair and to visualize wheelchairs movement. Laser scan messages are used also for retrieving information about the obstacles around the wheelchair. These obstacles are detected by lasers. Laser scan can also be visualized on the map during the visualization of the wheelchair movement. Moreover, a custom service message is generated in order to create a service for updating the published map while map_server is running.

Beside these nodes and messages used directly in the implementation, some other nodes and tools were used for simulating the wheelchair. These nodes and tools will be explained in Chapter 5.1.

map_server Node:

map_server node provides publishers for map metadata and occupancy data of the map. Maps used by map server are stored in a pair of files. The *YAML* file describes the map meta-data, and names the image file. The image file encodes the occupancy data of the map. The information about YAML file and image holding occupancy data has been explained in the previous chapters.

With the modification on map_server node, a new feature, added. With this new feature map can be updated with a service call to map_server node. This helps to change the map being published by the map_server which is used by the wheelchair when the node is already running without the need of stoping and restarting the node.

Modification made on the map_server is explained briefly in Chapter 4-2.

Odometry Messages:

Odometry is the use of data from moving sensors to estimate change in position over time. Odometry is used by some robots to estimate (not determine) their position relative to a starting location[12]. This method is sensitive to errors due to the integration of velocity measurements over time to give position estimates.

ROS uses odometry to estimate the position of the robot. Especially the Navigation stack uses odometry. Using odometry messages, estimated position and the velocity of the wheelchair is retrieved and used for the visualization of the wheelchair movement on the map.

Compact message definition for Odometry message: [13]

std_msgs/Header_header
string child_frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist

Variable pose holds the estimated position and orientation info where variable twist holds angular and linear velocity. Orientation is given in quaternions instead of

Euler Angles since Euler Angles are limited by a phenomenon called "gimbal lock," which prevents them from measuring orientation when the pitch angle approaches +/- 90 degrees. A quaternion is a four-element vector that can be used to encode any rotation in a 3D coordinate system.

LaserScan Messages:

Laser scanners are commonly used sensors in robotics. LaserScan message is a type of sensor message that gives information about the obstacles around wheelchair.

Basically, laser rays are sent from the sensor to environment with a fixed angle increment between each ray within the angle ranges of the sensor. If there is an obstacle in the direction of the ray, this distance and intensity is detected. If ray does not hit any obstacle within the range that sensor can detect, infinity assigned for this rays distance. In this manner all the angles from defined minimum angle to defined maximum angle are controlled and the values of distances and intensities are saved.

LaserScan messages are used in the visualization part, by processing the retrieved message and drawing lines for each ray in the length of distance detected.

Compact message definition for LaserScan message: [14]

std_msgs/Header_header
float32 angle_min
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan time
float32 range_min
float32 range_max
float32[] ranges
float32[] intensities

4.1.4 ROSJava for android

rosjava_core is a pure *Java* implementation of *ROS*. It provides a client library that enables *Java* programmers to quickly interface with *ROS Topics, Services, and*

Parameters. It also provides a Java implementation of roscore. Because *ROS* is heavily dependent on network communication, *ROSJava* is asynchronous. [15]

ROS is also available to android with libraries developed based on *ROSJava* client and core libraries.

android_core is a collection of components and examples that are useful for developing *ROS* applications on Android.

android_core provides *Android Library Projects* to help for writing *ROS* applications for *Android*. The library projects are named for the Android API level they require (e.g. *android_gingerbread_mr1* and *android_honeycomb_mr2*). Each class or feature is defined in the library project that represents the minimum version of Android required for it to work. [16]

In this thesis android_gingerbread_mr1 (API level 10) library project is used. It is the lowest API level supported and provides the base Activity(RosActivity) and Service (NodeMainExecutorService) for executing and managing the lifecycle of your NodeMains. [16]

Beside the core dependencies of rosjava for android, there are also some other rosjava packages included in the application for specific features such as messages, services.

Moreover, service messages for communicating with the modified version of the map_server are generated and included as an external library and included to application. More information will be given about this generated jar file in Chapter 4-2.

4.2 Software Project Implementation

In this section, implementation of the software project is explained in detail. The general structure of the application, implementation of the android application, and modifications made on the map server node are included. In the sections explaining

android application implementation and map server modification, some important code pieces are shown and explained for guiding the future developers.

4.2.1 Android Application

4.2.1.1 Layouts and resources

In the application, there are three different screen serving for different functionalities of the project. These screens are *Home Screen*, *Settings Screen,* and *Maps Screen.* For each screen, there is a xml layout file under res/layout folder: *activity_main, activity_maps, activity_settings.*

Layout files define the user interface of the screens with components such as; buttons, edit text boxes, text views, image views, list views, and layouts. All three layouts have been divided into two parts. On the left buttons for navigation exists and on the right, there are other components for specific features of each pages. These layouts are updated from the activity classes according to the actions taken by user. Each screen has an activity class for managing the layout and responding to user inputs. Since the application is intended for tablets, all the screens are in landscape orientation. In all layout files, the flag for keeping the screen always on while the application is running is set to true.

Beside the layout files, there is a file named *strings.xml* under values folder. This file is used for holding the string values used in the application.

In *AndroidManifest* file, minimum sdk version is defined as 10 and the target is 21. Configuration of the application are made in this file. Moreover, for using some features permissions are added such as:

WRITE_EXTERNAL_STORAGE: allows application to create files under the external storage of the device.

INTERNET: allows application to connect internet.

ACCESS_NETWORK_STATE: allows application to get the information about the state of network connections.

WAKE_LOCK: allows application to hold the screen from locking itself.

Functionalities and components of each screen are described below:

Home Screen:

On the left part of the main screen users see navigation buttons for other pages. Below these buttons there is a panel for giving information to user. The chosen map name (when app is started default map name is shown) is shown on start. When application starts or main screen is viewed from other screens, if there is no wifi connection, user is warned since it is necessary to have a wifi connection for map preview. If there is an image file already created before and placed into *ALMA_MAP* folder under devices file structure for the chosen map, *Preview* button for starting the preview is shown under the information panel. When preview is started, a checkbox is seen below *Disconnect* button for retrieving and visualizing laser scan information. Users can enable and disable the laser scan visualization on the map using this checkbox.

On the right side of the main screen, there is a panel for showing the map and wheelchair movement. This is a simple imageview showing the related part of the map and wheelchair with position and velocity information on it. To see the map, user should be connected to same connection with wheelchair and press on preview button. Map will be updated as the wheelchair moves around. There will be two buttons on the right upper corner of the screen for zoom in and zoom out. The implementation will be explained in *MapViewer* class.

Settings screen:

On the left part of the settings screen there are navigation buttons for other screens. Below these buttons, there is a panel showing information about connection status and published map metadata.

By default the status is *Disconnected* and when user press on connect button on the right part of the screen, this status turns into *Connecting*. If connection is successful users see status as *Connected* and width, height, resolution information of the published map from the wheelchair. This connection is made only for testing the connection with IP and port number saved in settings screen. If there is no wifi connection and connect button is pressed, users are warned that for proper connection device should be connected to same local network with wheelchair.

On the right part of the screen, there are edit text boxes for entering wheelchair name which will be visualized on the map(this name is node name which publishes odometry and laser scan messages), odometry and laser scan topic names, IP and port number for connecting to wheelchair. Below these boxes, there are two edit boxes for username and password for the SSH framework running on wheelchair environment to be used for establishing a SFTP connection and sending map. Below these boxes there is a save button for saving the entered information and to be used in all other sections using connection to wheelchair. By default, these boxes are filled with default values. Next to save button, a button is placed for testing the saved values used for connecting to wheelchair as explained above.

Maps screen:

In the maps screen, similar to other screens, there is left panel for navigation buttons. On the right part, there are two tabs. By default users see the map list panel, which lists the maps in the *ALMA_MAP* folder with xml (*IndoorGML*) format. Users can choose a map and see parse or send buttons. If the map is not parsed before and YAML format is not generated yet, send button is not visible. After clicking on parse button, map is parsed with the given resolution and YAML format is generated with an image holding the occupancy data and and image to be used in visualization. Users can send the map to wheelchair by choosing the map from the list and clicking on send button. For connection, values of IP, username, and password are taken from already saved values in the settings screen. If there is no wifi connection user is warned with a dialog.

On the second tab, there is an option for downloading *IndoorGML* files by using the download url. By default, there is an example url and map name in the edit box widgets. When user clicks on dowload, file is downloaded and placed under *ALMA_MAPS* folder. Device should be connected to internet for downloading the file. If there is already a map with the entered name in *ALMA_MAPS* folder, the old file is replaced with the new one.

4.2.1.2 File structure and classes

The file structure of the project is shown below:

- →*i. activity*
  1. *BaseActivitiy*
  2. *MainActivity*
  3. *SettingsActivity*
  4. *MapsActivity*
- → *ii. almaui*
  - *a. ALMA*
  - → *b. utils*
    1. *AlmaMap*
    2. *AlmaMapFactory*
    3. *ServerAdapter*
    4. *MapViewer*
    5. *Alert*
    6. *OdomInfo*
    7. *LaserInfo*
    8. *MapTransferrer*
- → *iii. ros*
  1. *MapUpdateService*
  2. *OdomListener*
  3. *RetrieveMapInfo*
  4. *SensorListener*

i. Activity: Under the activity folder, there are activity classes for managing the user interface and implementing the logic with the help of other classes.

1. **BaseActivity:** This class is a parent class for other Activity classes in the project. Therefore, the shared methods and variables are placed in this class and they are inherited by the other activity classes.

There are three methods defining the navigation buttons callback handlers (mapsClickHandler, settingsClickHandler, homeClickHandler). These methods are mapped to corresponding buttons from the child activity classes for each screen.

Since NodeMainExecutor class, which is the class for executing ROS nodes in ROSjava environment, is used by all the activity classes, it is placed in this class. There are variables holding wheelchair uri (IP), wheelchair port, wheelchair username, wheelchair password and default values are assigned to these variables. With a method in BaseActivity class, saved values in Settings screen are taken from the shared preferences and replaced with the default values. This method is called at the beginning of each activity, so in all activities latest saved values are used.

When user exits from the application or changes the screen in the application, NodeMainExecutor class instance should be shut down so that the connection with wheelchair is closed for that activity. Therefore, whenever user navigates using the navigation buttons or presses on back button, if NodeMainExecutor is already running, shutdown method of this class is called inside the navigation button callback handler and back button pressed callback handler.

2. **MainActivity:** MainActivity class extends the BaseActivity class and holds the methods and variables for managing the main screen. It holds a variable for the name of the map to be shown in preview. This map name has a default value on start and is changed whenever user sends a new map to wheelchair. This class contains variables holding the odometry information and sensor information retrieved from wheelchair. These variables are modified within the OdomListener class or SensorListener class and used in visualization process.

   Moreover, mapViewer variable in MainActivity class is used to update the map preview and wheelchair position. Whenever a new odometry

message is received by OdomListener class, variables holding the rotation, position and velocity of the wheelchair are assigned with new values, values shown on the screen are updated and method for updating the map preview is called from mapViewer instance. All these are made inside a handler created in MainActivity. Also, sensor information is updated if show sensors option is selected and laserInfo in mapViewer instance is updated by the same handler.

In the initialization, the method for getting the latest values of wheelchair Ip, port, username, and password is called from the parent activity to be used later for connecting to wheelchair and getting the position information or laser scan information of wheelchair.

The image view for previewing the map has also zoom in and zoom out feature. Therefore, in MainActivity class there is a ScaleGestureDetector instance for receiving pinch gestures on map image and updating the view accordingly as well as callback for zoom in and out buttons.

When user starts the application main screen is viewed, which means MainActivity class is initialized. In the initialization, if the device is not connected to wifi, user is warned with a dialog informing that device should be connected to same wifi connection with wheelchair in order to preview the map. Same dialog is shown also when user tries to preview map by clicking on preview button without connecting to a wifi. If the image file for the chosen mapname does not exist under ALMA_MAPS folder, user is warned with a small message telling the map chosen does not have the corresponding image file. In this case user cannot see the Preview button. To be able to continue with the preview, user should go to maps screen, parse the corresponding file (download if necessary), or choose a map with generated image and send it to wheelchair.

In MainActivity class, there are methods for putting the image of the map on the imageview widget by initializing mapViewer instance, setting up

the handlers for zooming and updating the map view, starting the connection for receiving the odometry messages from wheelchair. These methods are called inside the callback handler of preview button. When the preview is on, it is possible to disconnect and stop the preview with another method which is the callback handler for disconnect button. This button replaces the preview button when preview is started and vice versa when preview is stopped. There is also methods for starting the sensor visualization.

3. **SettingsActivity:** SettingsActivity extends the BaseActivity and holds the methods and variables for managing the settings screen. There are variables holding the map information to be shown on the screen after the test connection, which can be established by using a button on this screen. There is also a variable holding the status of the connection, which is in disconnect state on start-up and becomes connecting when user tries to test the values saved in this screen. If the connection is successful, state becomes connected and map metadata information is shown on the status panel of the screen.

   In the initialization of SettingsActivity, the values of the wheelchair name, uri, port, username and password are retrieved from the shared preferences and placed to corresponding edit text views on the screen. User can edit these values and save by calling a method, which saves these values in shared preferences of the application to be used by all other methods in the application which connect to wheelchair.

   When user test the connection with connect button, if device is not connected to wifi, a similar dialog to the one in MainActivity warns the user about the wifi connection.

4. **MapsActivity:** MapsActivity class extends the BaseActivity class and contains the methods and variables for managing the maps screen. There is a ProgressDialog instance, which is shown whenever a task is processing in

the background such as downloading IndoorGML file, sending map to wheelchair, parsing IndoorGML file for generating the YAML formatted maps.

In maps screen there are two tabs. In the tab map list, which is shown by default, there is the list of maps in IndoorGML format, which are placed under ALMA_MAPS folder. In the initialization of the MapsActivity class, this list is created with an adapter. There is also a callback assigned to this list. When user selects a map from the list, on the right panel, the name chosen is shown as well as an edit text box for resolution to be used for parsing file and a button for calling the parse task. If the map has already parsed and YAML format is generated send button is also placed next to parse button.

For switching between tabs, there are two methods assigned to tab buttons. These methods update the layout and show the correct components on the screen.

On download map screen, there are two edit text boxes with a button for downloading the map using the uri in the edit text box above and saving with the name in the second edit text box.

For downloading, parsing, and sending map, there are different AsyncTask classes created and placed under MapsActivity class. These tasks are started within the callback handler methods of the corresponding buttons.

When download button is clicked, callback method gets the uri and map name written in the boxes, using these values calls the DownloadTask. DownloadTask method uses ServerAdapter class for connecting to AlmaServer and downloads the file. The progress dialog is shown until the task is finished. If download is successful, map list is updated with the new file. If the name chosen for the downloaded map is same as one of the indoorGML files under ALMA_MAPS folder, the file is replaced with the new file.

When a map is chosen from the list and parse button is clicked, resolution value entered is taken and used for calling ParseTask. ParseTask uses AlmaMapFactory class for generating the YAML formatted map. The progress dialog is shown until task finishes.

If user chooses a map and clicks on send button while the device is not connected to wifi, user is warned with a dialog telling that device should be connected to same wifi connection with wheelchair in order to send the map. If the connection is valid, SendTask is started from the callback handler of the button. SendTask uses MapTransferrer class for sending the chosen map in YAML format with the image file containing occupancy data to wheelchair.

After sending and locating these files under the file structure of the wheelchair, a service call to wheelchair is made by using MapUpdateService class. When a map is sent, this map becomes the one to be previewed on the main screen.

***ii. Almaui*:** Under almaui folder, there are utility classes, which are used by activity classes for accomplishing the tasks explained above.

***a.* ALMA:** ALMA class holds the uri for the default ALMA server and the default world to be used while downloading the indoorGML file. This value is shown when download tab in maps screen is viewed. User is supposed to modify the url on the screen in order to download a different map.

***b. Utils*:** Under utils folder there are supplementary classes explained below:

1. **AlmaMap:** AlmaMap class is a helper class for AlmaMapFactory and it holds the variables for generating a map from indoorGML file(mapName, xMax, yMax, xMin, yMin, translationX, translationY, rotation, resolution). There are getter and setter methods for all this variables. mapName holds the name of the map to be generated as well as being parsed. xMax, yMax, xMin, yMin variables are used for calculating the width and height of the image file to be generated.

These will take the lowest and highest values from the points defined in indoorGML file while parsing the indoorGML file inside AlmaMapFactory class.

Moreover, there are static functions inside this class which gives the path on the device for the map which is given as parameter to these functions. There are five static methods: one for general path for ALMA_MAP folder and others for xml, yaml, and two png formatted files for the given map name.

2. **AlmaMapFactory:** AlmaMapFactory class is used by ParseTask in MapsActivity and serves for parsing the indoorGML file, which is previously downloaded under ALMA_MAPS folder. There are six static methods in this class.

   First method, createAlmaMap, is called with a map name and resolution value. The xml file content is retrieved with another static method in this class (getFileToParse). An AlmaMap instance is created and the values are assigned to this instance to be used in the next steps of the process. Three other methods are called from the createAlmaMap method in the order: initializeMapValues, generateMap, and saveFiles.

   InitializeMapValues method is called with the almaMap instance and the content of the xml file, which has been retrieved previously. In this method, indoorGML file is parsed using an xml parser instance to set the variables in AlmaMap instance by reading the specific tags in indoorGML file (xMax, xMin, yMax, yMin, translationX, translationY, rotation).

   After initialization of the almaMap instance, occupancy data bitmap is generated by calling generateMap method with parameters: almaMap and xml file content. This method generates a bitmap by parsing the xml file content and using the previously retrieved

information about almaMap instance. The width and height of this bitmap is calculated by almaMap instance by taking into account the xMax, yMax, xMin, yMin values. Similar to initializeMapValues method, specific tags of the indoorGML file are parsed and polygons are drawn on the bitmap created.

The polygons are colored according to their availability for navigation with the logic below:

If polygon is navigable → white color
If polygon is obstacle or wall → black color
If an area is not defined by indoorGML tags → gray color

Polygons indicating doors are retrieved and painted into white color by generateMap method. The generated bitmap is resized according to the resolution value set by user and this resized bitmap is returned as the occupancy data image. Beside the bitmap image returned as a result, another image is created and saved with a fixed resolution (10 pixels / meter) in order to be used for visualization. This image file is created by calling saveVisualizationImage method and giving almaMap, bitmap generated, width and height values of the map to be created as parameters. This method simply resizes the image to given dimensions and saves under ALMA_MAPS folder.

After initializing almaMap instance and generating the bitmap with occupancy data, saveFiles method is called for creating and saving the YAML file and image file in png format. This method takes bitmap and almaMap instance, previously generated by other methods, as parameters and saves an image file in png format under ALMA_MAPS folder with the same map name given. Moreover, the YAML format holding the metadata of the map is created by using the information from almaMap instance. Image name, resolution assigned, origin, occupied and free threshold values are written into text file.

TranslationX, TranslationY and rotation values from the almaMap instance are also added as lla_origin value in this file. This yaml formatted file is saved under ALMA_MAPS folder as well as the image file generated which will be sent to wheelchair.

3. **ServerAdapter:** ServerAdapter class is used by DownloadTask in MapsActivity class. This class simply has a static method which creates a HTTP request with a Json object parser and gets the content of the file which is published on the url given as parameter. This content is saved as an xml file with the given name as a parameter under ALMA_MAPS folder. In order to have a successful download call, the device should be connected to internet.
4. **MapViewer:** MapViewer class is used by MainActivity class for visualization of map preview and wheelchair movement on the map. When user clicks on Preview button on main screen, an instance of mapViewer is created with parameters:

   imageView(the widget on the screen for viewing the map image),
   mapImage(full bitmap image of the map),
   positionX(text view showing the position of wheelchair on x plane),
   positionY(text view showing the position of wheelchair on y plane),
   width(width of the image part to be shown on screen with zoom),
   height(height of the image part to be shown on screen with zoom).

   If preview is already called once and clicked again, instead of creating a new instance updateMap method of the previously created instance is called with mapImage, width, and height values.

   MapViewer class has fields for the values given as parameter in initialization and an extra field for holding the part of the image to be shown which changes dynamically according to the position of the wheelchair and zoom value set by user. There is also LaserInfo class instance, which is updated by the handler in MainActivity setting the

values retrieved from wheelchair with SensorListener class. LaserInfo instance is used for adding the sensor information on the map.

UpdateMap method of this class is called for updating the previously created mapViewer and restart preview, after disconnecting the previous preview. It simply updates the bitmap image by recycling the old one and replacing it with the new one from parameter. Variables holding the width and height are also updated with the new parameters.

UpdateViewerZoom method is called from MainActivity class whenever a pinch gesture is detected. The width and the height of the map being previewed are recalculated with the new zoom value from parameter. For calculating the new width and height, the original width and height of the bitmap image is divided by the value:

*(zoom + 1) * 0.5*

The text widget showing the zoom value on the screen is updated from MainActivity class after zooming.

Whenever a new odometry message retrieved by MainActivity class, setPosition method of the MapViewer class is called in order to update the position of the wheelchair and change the part of the image shown on the screen. This method first updates the values of the text widgets on screen showing the position of the wheelchair on x y plane. The part of the map shown on the screen is calculated according to width, height values after zoom. If the wheelchair is not close to boundaries of the map, it is placed in the middle of the preview. If the wheelchair is close to boundaries of the map, the closest part of image to wheelchair with calculated width and height is shown and wheelchair is not placed in the middle anymore.

After calculating the part of image to show, wheelchair is drawn as a blue circle and velocity as a smaller yellow circle on the map showing the next position that wheelchair will move (the direction of the wheelchair). If the sensor visualization is enabled, results of the laser scan is visualized on the map by drawing a red line for each laser by using the values retrieved from LaserInfo instance which is updated with new values by SensorListener class. After calculating the current status of the map (map preview, wheelchair position, direction and laser scan optionally), image view on the screen is updated with the created bitmap image. If wheelchair is out of the boundaries of the map, only the visible part of the laser scan or wheelchair is drawn on the map.

5. **Alert:** Alert class has a static function for creating an alert dialog easily by giving context, title for dialog, and message of the dialog. This method is called by the activity classes whenever an alert dialog is needed to give warning or information to users.
6. **OdomInfo:** OdomInfo class is created to store the values retrieved from the odometry topic of the wheelchair. There are fields and getter/setter methods for holding x, y plane coordinates of wheelchair position in meters, x, y, z, w values of the orientation (quaternion values) to be used in calculating the wheelchair direction, and linear velocity of wheelchair on x, y planes.

MainActivity has and OdomInfo instance and values of this instance is updated by OdomListener class with the values retrieved from odom topic of the wheelchair. These values are used for calling the setPosition method of MapViewer instance in order to calculate the new preview image.

The angle in degrees, which shows the rotation of the wheelchair from the default direction is calculated using quaternion

values (only z and w) within convertAngleToDegrees method with the formula given below:

```
if z > 0
        angle_in_degrees = 2 * acos(w)
else
        angle_in_degrees = -2 * acos(w)
```

7. **LaserInfo:** LaserInfo class is created to store values laser scan values retrieved from wheelchair by using SensorListener class. There are fields and getter/setter methods for the values of:

   angleMin (minimum angle of the laser scan with the direction of wheelchair),

   angleMax (maximum angle of the laser scan with the direction of wheelchair),

   angleIncrement (angle difference between two laser scans),

   rangeMax (maximum distance that laser can measure in meters),

   rangeMin (minimum distance for laser),

   ranges (list of distances in meters scanned for each angle within the range of laser at a given time instance).

   The values are used for drawing the sensor information on the map preview if users choose the option of showing sensor for the given laser name while preview is on. In setPosition method of MapViewer instance, for each angle that laser can scan, a line is drawn on the preview image by calculating position and direction of each ray relative to the position and direction of the wheelchair.

8. **MapTransferrer:** MapTransferrer class is used for sending the map to wheelchair by SendTask of MapsActivity. There is a static method called transferMap which takes the name of the map to be sent, ip, username, and password of wheelchair as parameters and creates a

SFTP connection using these values. Via this connection YAML file and the png file holding data for the map chosen are sent and placed under ALMA_MAPS folder on wheelchairs file structure.

iii. ***Ros*****:** Under ros folder, there are classes for connecting to ros master running on the wheelchair in order to retrieve information or modify the current running nodes. This classes are called from activity classes using NodeMainExecutor instances, and they connect to wheelchair using the ros message passing paradigm. In order to have successful calls with these classes, device should be connected to same wifi connection with the wheelchair and the values for connection (IP, port) should be set correctly.

1. **MapUpdateService:** MapUpdateService class used by MapsActivity class in order to update the modified version of the map service node running on wheelchair after sending a new map to wheelchair. It uses generated MapUpdateResponse and MapUpdateRequest classes, which are imported into project as a jar file as service messages to send the map path and retrieve the result of the action. The path of map on the wheelchair is set as the path value in MapUpdateRequest class and this request is sent to wheelchair. If the running instance of the map server is not the modified one within this project, the update call is not going to be successful. After a successful map update call, map server node running on wheelchair will start publishing the new map under the given path.

2. **OdomListener:** OdomListener class is used by MainActivity class in order to get the position, direction and velocity (next point to be navigated) of the wheelchair on the map. When user starts the preview, and instance of OdomListener class is started and a handler is assigned to receive update message from OdomListener. This handler is passed as a parameter when initializing the OdomListener instance. Whenever a new odometry message is received from the wheelchair, variables in MainActivity class holding the position and velocity of the wheelchair in x/y planes are updated with

received data and an update call is made to handler in MainActivity in order to make a call to setPosition method in MapViewer class.

3. **RetrieveMapInfo:** RetrieveMapInfo class is used by SettingsActivity class for testing the connection between wheelchair and device. When user clicks on connect button after saving the correct values of the Ip and port number of the wheelchair, an instance of this class is started. This class simply listens to map metadata published from the map server node running on the wheelchair. When the metadata of the map is retrieved, a message to the handler, which is created and send to this class as parameter in the constructor, in order to update the values shown on the screen with the ones just received. Width, height, and resolution of the map published from map server is retrieved and the connected status in the SettingsActivity is changed to true after a successful call.

4. **SensorListener:** SensorListener class is used by MainActivity similar to OdomListener class, but instead of odometry messages, purpose is to retrieve laser scan messages from wheelchair. Laser scan messages give information about the obstacles around the wheelchair and distance between wheelchair and these obstacles or walls. When user fills the name of the correct laser to be listened and starts sensor visualization while preview is on, this class starts to listen to the topic publishing the laser scan results. When creating an instance, same handler for OdomListener is passed as parameter as well as the names for wheelchair and laser to be listened. A new node is created which subscribes to the given topic, which publishes the laser scan messages and whenever a new message arrives, LaserInfo instance in MainActivity is updated with the new values from the message.

4.2.1.3 Implementation of the specifications

In this section, some code pieces used in the implementation of different tasks will be explained in detail.

a. Downloading IndoorGML file:

Downloading and creating the IndoorGML file is done by a static method in ServerAdapter class. This method is called from doInBackground method of DownloadTask instance in MapsActivity class with the url and map name entered by user into edit boxes on the screen:

ServerAdapter.downloadMap(downloadUrl, downloadedMapName);

Inside the downloadMap method a HttpRequestFactory is created from HTTP_TRANSPORT, in order to create requests using the JSON object parser with JSON_FACTORY.

Content of the file from the url is parsed as string using a request build from requestFactory. After getting the content of the file DocumentBuilderFactory is used for creating a document instance and this document is used to create a DOMSource instance. A new file is created under ALMA_MAPS folder if there is no file created before with the same name. DOMSource instance is transformed by a Transformer and content is saved in xml format into this file.

b. Parsing indoorGML file:

IndoorGML file is parsed by AlmaMapFactory class using AlmaMap class as a model. The process of parsing IndoorGML file and generating YAML file and image with occupancy grid maps is divided into a pipeline with 4 steps. Parsing starts from parseTask subclass of MapsActivity with a call to createAlmaMap method of AlmaMapFactory with mapname and resolution:

AlmaMapFactory.createAlmaMap(chosenMap, resolution);

Before parsing starts the first step is getting the content of the IndoorGML file with getFileToParse method which basically reads the content of the file from file system of the device.

```
String data = getFileToParse(mapName);
```

After initializing an instance of AlmaMap class with map name and resolution, initial values for creating the bitmap image for occupancy grid data are retrieved by calling initializeMap method with parameters almamap, and content:

```
initializeMapValues(almaMap, data);
```

Inside this method, an xml parser instance created for content. To retrieve the width and height values of the final image, content is parsed according to tag names. Only the first layer of the IndoorGML format is used. With a loop over the branches of the xml tree, for each start event, tag names are checked to decide which data that branch corresponds.

```
currentTagName = parser.getName();
if (currentTagName.equals("gml:Polygon")) {
    inPolygon = true;
} else if (currentTagName.equals("indoorCore:SpaceLayer")) {
    String layer = parser.getAttributeValue(0);
    if (!layer.equals("L1")) {
        endOfFirstLayer = true;
    }
} else if (currentTagName.equals("gml:posList")) {
    if (inPolygon) {
        inDataItemTag = true;
    }
} else if (currentTagName.equals("indoorNavi:translation")) {
    inTranslation = true;
} else if (currentTagName.equals("indoorNavi:rotation")) {
    inRotation = true;
}
```

"gml:Polygon" tag switches a flag to inform the parser is inside one of polygons which creates the map. "indoorCore:SpaceLayer" tag is checked if the L1 layer is finished. "gml:posList" is used to get the points of the polygons which will be used in text event capture to decide the max and min coordinates over x and y planes so the image size can be calculated. *"indoorNavi:translation"* and *"indoorNavi:rotation"* tag names are used to switch flags for saying translation and rotation of the map according to world coordinates will be in the next text event.

Inside the text event, the flags are controlled to decide type of retrieved data and data is processed according to these flags. At the end of process used flags are reseted.

If tag being parsed is point list of a polygon, with a loop over points of polygon maximum and minimum coordinates calculated for x and y planes are updated with new values.

```
almaMap.setxMax(Math.max(almaMap.getxMax(), Double.valueOf(split[i])));
almaMap.setyMax(Math.max(almaMap.getyMax(), Double.valueOf(split[i + 1])));
almaMap.setxMin(Math.min(almaMap.getxMin(), Double.valueOf(split[i])));
almaMap.setyMin(Math.min(almaMap.getyMin(), Double.valueOf(split[i + 1])));
```

If data corresponds to translation or rotation these values are set in AlmaMap. If it is end of firstlayer, process is finished and AlmaMap instance is initialized with necessary values to generate files.

With generate file method following the initialization bitmap image is created and occupancy information is encoded.

```
Bitmap bitmap = generateMap(almaMap, data);
```

In this method, bitmap image is created with the width and height calculated by using the minimum and maximum points of x, y planes retrieved with previous method. Similar to initialize method, content is parsed one more time. In the second loop over branches of xml tree, polygons and position list are retrieved as before but with addition flags for deciding if the polygon is navigable with tag name *"indoorNavi:NavigableSpace"* or a connection space (a door) with tag name *"indoorNavi:ConnectionSpace".*

When the text events are captured if data corresponds to points of polygon, A path is created to be drawn on bitmap image following all the corners of the polygon.

```
Path wallpath = new Path();
wallpath.reset();
double firstX = Double.valueOf(split[0]) - almaMap.getxMin();
double firstY = Double.valueOf(split[1]) - almaMap.getyMin();
wallpath.moveTo((float)firstX, (float)(almaMap.getHeight() - firstY));
for(int i = 2; i < split.length; i = i + 2) {
```

```
    double x = Double.valueOf(split[i]) - almaMap.getxMin();
    double y = Double.valueOf(split[i + 1]) - almaMap.getyMin();
    wallpath.lineTo((float)x, (float)(almaMap.getHeight() - y));
}
wallpath.lineTo((float)firstX, (float)(almaMap.getHeight() - firstY));
```

As it can be seen for calculating the y coordinate, the number retrieved is subtracted from the height of the image since the images origin is upper left corner, while the world's origin is given as lower left corner.

The polygon is drawn on the bitmap and filled with a color code: white meaning navigable space or a door, and black meaning obstacles or walls.

When all the polygons defined in the indoorGML file are drawn, image file is saved with a fixed resolution (10 pixels / meter) for visualization purposes.

```
saveVisualizationFile(almaMap, Bitmap.createScaledBitmap(bitmap,
(int)(almaMap.getWidth() / 10), (int)(almaMap.getHeight() / 10), false));
```

The image is resized according to the resolution value by dividing the width and height to *"almaMap.getResolution() / 0.01"* to get an image with correct resolution and result image is returned as the output.

In the final step two files are saved according to data collected:

```
saveFiles(bitmap, almaMap);
```

In this method, image file is saved under ALMA_MAPS folder in png format and given map name. YAML files content is generated with values and it is created and placed in the same folder.

```
String yamlData = "image: " + almaMap.getMapName() + ".png \n" +
      "resolution: " + String.valueOf(almaMap.getResolution()) + "\n" +
      "origin: [ 0, 0, 0] \n" +
      "occupied_thresh: 0.65 \n" +
      "free_thresh: 0.196 \n" +
      "negate: 0 \n" +
      "lla_origin: " + (almaMap.getTranslationX() - almaMap.getxMin()) + ", " +
(almaMap.getTranslationY() - almaMap.getyMin()) + ", " + almaMap.getRotation();
```

c. Sending map to wheelchair:

Generated files are sent to wheelchair using MapTransferrer class from SendTask subclass of MapsActivity.

final String path = MapTransferrer.transferMap(mapNameToSent, wheelchairUri, wheelchairUsername, wheelchairPassword);

Within this function a secure ssh session is created by using the values given as parameters:

```
Session session = jsch.getSession(wheelchairUsername, ip , 22);
session.setPassword(wheelchairPassword);
```

From this session an SFTP channel is created and if the folder does not exist under the home folder of the username on wheelchair file system, it is created. Through this channel, both YAML file and png image file are read from device folder, sent to wheelchair and placed under ALMA_MAPS folder.

The path to YAML file on wheelchair file structure is returned as output. This path will be sent to map server in order to update the map being published with the new map:

return "/home/" + wheelchairUsername + "/" + FOLDER_NAME + "/" + mapNameToSent + ".yaml";

d. Connection to ROS nodes of PMK:

For connecting to ros nodes running on wheelchair in order to retrieve information or make a service call, NodeMainExecutor class from rosjava library is used. For sending a service message to modified version of map server node, MapUpdateService instance is executed with necessary parameters.

```
NodeConfiguration mapPathConfig = NodeConfiguration.newPrivate();
mapPathConfig.setMasterUri(URI.create(wheelchairUri + ":" + wheelchairPort));
mapPathConfig.setNodeName("MapUpdate");
NodeMain mapUpdate = new MapUpdateService(path);
e.execute(mapUpdate, mapPathConfig);
```

MapUpdateService creates a service clients and calls the service with the path to YAML file on the wheelchair file system:

```
        ServiceClient<MapUpdateRequest, MapUpdateResponse> client =
connectedNode.newServiceClient(SERVICE_NAME, SERVICE_TYPE);
        MapUpdateRequest request = client.newMessage();
        request.setPath(pathToMap);
        client.call(request, new ServiceResponseListener<MapUpdateResponse>() {
                …
        }
```

MapUpdateRequest and MapUpdateResponse classes are generated according to custom service message used by modified map server node and included as an external library to application. Request holds a string variable for the path and response holds a boolean variable to inform about the result of call stating if it successful. Generation of these classes will be explained in map server modification section.

For retrieving map metadata being published from map server node, a subscriber for map metadata is created in RetrieveMapInfo class. This node is executed from SettingsActivity in a similar way to MapUpdateService call by changing the node name and creating node as an instance of RetrieveMapInfo class instead of MapUpdateService class. Inside this class a subscriber for map_metadata topic is created and added a message listener which assigns the values of map width, height and resolution to fields in SettingsActivity and calls the handler to inform that values are updated.

Similar to retrieving map metadata, listening to odometry messages is done by creating a subscriber for odom topic for the chosen wheelchair name. This node is created in the same manner with previous nodes by using OdomListener instance. A message listener is added to subscriber for updating the values of position on x, y plane, orientation quaternion, and linear velocities on x, y plane in the OdomInfo instance of MainActivity with each messages retrieved. After assigning values, handler is called to inform the MainActivity for updating the position and orientation of the wheelchair on the map.

The same implementation is valid for retrieving laser scan information, where the node is created as an instance of SensorListener class. In this class a subscriber for the topic with the given name on the given wheelchair is created. A similar message listener

added for updating values of laser rangeMin, rangeMax, angleMin, angleMax, angleIncrement, and ranges in LaserInfo instance of MainActivity. MainActivity is informed about the new message by sending a message to handler.

e. Visualization of wheelchair movement and laser scan:

For the visualization of wheelchair MapViewer class is implemented such a way that map is placed and the wheelchair position, direction, and laser scan information are drawn on the top of map image. When preview is starting, wifi connection is checked and if there is no wifi connection an alert dialog is shown to user asserting preview cannot start without connection to wheelchair, since it is necessary to retrieve the necessary data for preview. If the device is connected to wifi a scale gesture detector is initialized for detecting zoom in or out events. Zoom value changes discretely with every single event detected.

Width and height of the image to be shown on the screen is calculated according to the zoom value:

```
int width = (int) (mapImage.getWidth() / ((zoom + 1) * 0.5));
int height = (int) (mapImage.getHeight() / ((zoom + 1) * 0.5));
```

If the MapViewer instance is not initialized before a new instance is created. If the preview is being called second time after disconnecting, instead of creating a new MapViewer instance, old one is being updated with values.

```
if(mapViewer == null) {
        mapViewer = new MapViewer(imageView, mapImage, positionX, positionY,
width, height);
} else {
    mapViewer.updateMap(mapImage, width, height);
}
```

After initializing the MapViewer instance, handler is initialized for receiving update flags for odometry messages from the OdomListener and laser scan messages from

SensorListener. OdomListener is executed with wheelchair ip and port in order to get the odometry messeges and update the position and direction of the wheelchair.

In the initialization of the MapViewer instance, image view for drawing the preview image, the original map image to be used, width, height values showing the width and height of the map that will be visible, text widgets showing the position on x, y coordinates are assigned to fields. A LaserInfo instance is created and wheelchair is position is set to default in order to have the map shown on the screen right after the initialization.

If MapViewer is being updated instead of being created from scratch, the old image holding the map is recycled and new values for image, width and height are assigned.

Whenever a zoom event is detected, either by using the zoom in/zoom out messages on the right upper corner or using pinch gesture, width and height values deciding the part of the image to be shown are updated:

```
this.width = mapImage.getWidth() / ((zoom + 1) * 0.5);
this.height = mapImage.getHeight() / ((zoom + 1) * 0.5);
```

If a laser scan message is retrieved, the values of the laser scan are assigned to LaserInfo instance in this class.

For retrieved odometry message, setPosition method is called in order to redraw the preview with new values. After updating the text views showing the coordinates of x, y plane, x and y values retrieved in meters are converted to pixels. Since all the visualization image files have a resolution of 10 pixels / meter and origin of the image save is different than the origin of the coordinates received x and y values in pixels are calculated as:

```
y = mapImage.getHeight() - (y * 10);
x = x * 10;
```

After converting all values in pixel coordinates, starting point of the image going to be viewed are calculated. Since wheelchair is going to be in the center of the image, half of the width and height are substracted from the coordinates:

```
int startX = (int)(x - (width / 2));
int startY = (int)(y - (height / 2));
```

If wheelchair is close to boundaries, it can not be placed in the center of the image that will be viewed. Therefore, calculated starting points should be checked if they are out of the boundaries as well as the end points (startX + width, startY + height):

```
int positionToStartX = (int)((mapImage.getWidth() - x) > (width / 2) ? startX :
mapImage.getWidth() - width);
int positionToStartY = (int)((mapImage.getHeight() - y) > (height / 2) ? startY :
mapImage.getHeight() - height);
```

Firstly, end points of the image that will be viewed are checked. If they are in the map, initial values are used. If they exceed the width and height of the whole map, starting points are selected for fitting the end points on the boundaries of the whole map.

```
positionToStartX = Math.max(positionToStartX, 0);
positionToStartY = Math.max(positionToStartY, 0);
```

After checking the endpoints, calculated start coordinates are checked if they are negative. This means wheelchair cannot be in the center of the shown image and starting points of the image will be viewed should be fitting the starting boundaries of the whole map. With the final values the part of image that will be viewed on the screen is selected from the whole map.

```
partToShow = Bitmap.createBitmap(mapImage, positionToStartX,
positionToStartY, (int) width, (int) height);
```

The wheelchairs coordinates are calculated according to the starting point of the image being viewed:

```
double wheelchairX = x - positionToStartX;
double wheelchairY = y - positionToStartY;
```

Over the image a blue circle is drawn at the position calculated using a canvas. If the laser scan is also going to be visualized, a loop over the range values for each ray starts. For each ray, angle between the base direction and the ray is calculated using the data from LaserInfo which is being updated with every laser scan message:

```
float rayDegree = (float) Math.toDegrees(laserInfo.getAngleMax() - (rayNumber *
laserInfo.getAngleIncrement()));
 matrix.postRotate((float)degree + rayDegree);
```

The exact coordinates of the rays end point is calculated by rotating the rays distance. Angle is calculated in degrees by summing the wheelchairs rotation angle and calculated for the specific ray. Wheelchairs rotation angle is calculated from quaternion values inside OdomInfo class and passed to setPosition method as a parameter.

```
if orientationZ > 0
    degree = -1 * Math.toDegrees(Math.acos(w) * 2);
else
    Math.toDegrees(Math.acos(w) * 2);
```

The distance of the ray is checked if it is lower than rangeMin or higher than rangeMax for laser. To show the ray on the map a red line is drawn from wheelchairs position to calculated end point.

```
canvas.drawLine((float)wheelchairX, (float)wheelchairY,
(float)wheelchairX + rayEnd[0] * 10, (float)wheelchairY + rayEnd[1] * 10, paint);
```

After drawing the wheelchair circle and laser scan according to position and rotation of wheelchair, velocity information and rotation of the wheelchair is interpreted and next point that wheelchair will navigate is drawn as a yellow circle which has half the radius of the wheelchair circle.

```
canvas.drawCircle((float)(wheelchairX + point[0] * 10), (float)(wheelchairY
+ point[1] * 10), wheelchairRadius / 2, paint);
```

The final result has a blue circle for wheelchair, a yellow circle for the next position and the laser scan data with red lines on the viewed part of the image according to zoom value. With every odometry message received, the image viewed is updated and wheelchair movement is visualized with laser scan.

4.2.2 Map Server Modification

a. General Information

In the default implementation of the map server, there is a service for detecting the map requests and there are publisher topics for map metadata and occupancy data. However, there is no service or method to load a new map while map server is running. In order to change the map being published map server is supposed to be stopped and started with the new YAML file path.

To fulfill the specifications of the application, a service is added to MapServer class in main.cpp file. This service serves for updating the published map with a new map while map server is running. The only modified file is main.cpp file. The new implementation can be compiled by using catkin_make command after setting up the proper development environment following the tutorials. Moreover, MapUpdate.srv file is created and new classes to be used in the service callback are generated with compile. In a similar way java classes are generated from the same MapUpdate.srv file as an external library and included to android application.

MapUpdate.srv file content:

string path
---
bool done

This content means that, there will be a string variable holding the path of map that is going to be published and there will be a boolean variable stating if the update is successful or not.

After compiling the same srv file in rosjava environment by following the tutorial for setting up a development environment, a jar library file is created holding three classes:

MapUpdate
MapUpdateRequest
MapUpdateResponse

These files are used for calling the update service from rosjava environment, in this case from the android application. Path of the map on the wheelchair file system is assigned to path variable in MapUpdateRequest and when the update is finished, MapUpdateResponse is retrieved with a true value assigned done field.

b. Implementation:

In the original implementation of map server, inside the constructor of the class, all the initialization is done, map is loaded, publishers are created. Since it is needed to update the published map, meaning loading a new file and publish new values over publishers, a new method, loadMap, is extracted. This method does exactly the same things that original implementation does in the constructor for loading a YAML file, publishing both map metadata and occupancy data. This new method is called from the constructor in order to load and publish the map given as parameter. After loading the map, new service is advertised similar to “static_map” service so that requests for update can be detected.

```
MapServer(const std::string& fname, double res) {
        fnameToLoad = fname.c_str();
         resValue = res;
         loadMap();
         service = n.advertiseService("static_map", &MapServer::mapCallback, this);
                updateService = n.advertiseService("MapUpdate",
&MapServer::updateMap, this);
 }
```

The new service added, MapUpdate, simply calls a method, which loads the map with the new path to the YAML file. The new path is retrieved from the path field of sent request. After assigning the path, loadMap method is called in order to load the map and publish new values. If the update is successful, variable in the response class is assigned to true.

```
bool updateMap(map_server_pmk::MapUpdate::Request &req,
               map_server_pmk::MapUpdate::Response &res) {
    fnameToLoad = req.path;
    ROS_INFO("Map update callback");
    loadMap();
    res.done = true;
    return true;
}
```

With these modifications, a new node map_server_pmk is created which has same properties with default map_server node and an extra service for updating the map while the node is already running.

Chapter 5

Development and Usage Guide

In this chapter, there will be information about the tools used in development process and explanation about how to setup a development/test environment. Moreover, a user manual for the end users is included with some screenshots from the application. This section is intended to guide future developers of the application. In this work, all the development is done using Ubuntu 14.04 as the operating system. Samsung Galaxy Tab2 10.1 tablet running with Android version 4.2.2 and Samsung Galaxy Ace Plus running with 2.3.3 are used for testing and developing as mobile devices in the system.

5.1 Tools used for development

5.1.1 Android Studio

For the development of the mobile application, android studio IDE is used. Android Studio is the official IDE for Android. Instead of this tool, Eclipse IDE with android development plugin can be used for development. However, there might be some incompatibilities in terms of configuration and a migration will be necessary for working in Eclipse.

Android Stuido can be dowloaded as a bundle with android development sdk, which enables the android development. If Eclipse IDE is going to be used, android sdk should be downloaded and path to this sdk should be configured to be able to use android libraries.

According to android developers tutorial [17]:

Before setting up Android Studio, JDK 6 or higher (the JRE alone is not sufficient) must be installed, JDK 7 is required when developing for Android 5.0 and higher.

To set up Android Studio on Linux:

1. Unpack the downloaded ZIP file into an appropriate location for your applications.
2. To launch Android Studio, navigate to the android-studio/bin/ directory in a terminal and execute studio.sh.

You may want to add android-studio/bin/ to your PATH environmental variable so that you can start Android Studio from any directory.

3. If the SDK is not already installed, follow the setup wizard to install the SDK and any necessary SDK tools.

Note: It might be needed to install the ia32-libs, lib32ncurses5-dev, and lib32stdc++6 packages. These packages are required to support 32-bit apps on a 64-bit machine.

After following these steps Android Studio is ready and loaded with the Android developer tools, but there are still a couple packages need to be added to make Android SDK complete. By default, the Android SDK does not include everything to start developing. The SDK separates tools, platforms, and other components into packages to be download as needed using the Android SDK Manager [18]. When Android SDK Manager is started from android studio, some default packages are selected and as a minimum when setting up the Android SDK, the latest tools and Android platform should be downloaded. Other packages are not needed for the development of the application explained here.

5.1.2 STDR simulator

For development and testing, a simulator is needed so that communication to wheelchair can be developed faster. As a simulator, a ros environment is needed first and creation of a ros environment will be explained in the following sections. For basic functionalities of simulator, a roscore master node can be started and necessary nodes publishing topics and supplying service can be added. There will be explanation about a robot simulator, which makes this process easier.

STDR Simulator implements a distributed, server-client based architecture. Each node can run in a different machine and communicate using ros interfaces. STDR Simulator, also provides a GUI developed in QT, for visualization purposes and more. The GUI, is not necessary for the simulator to run and its functionalities can be performed using command-line tools provided with the package.

The **STDR Simulator** available packages are [19]:

- **stdr_server:** Implements synchronization and coordination functionalities of STDR Simulator.
- **stdr_robot:** Provides robot, sensor implementation, using nodelets for stdr_server to load them.
- **stdr_parser:** Provides a library to STDR Simulator, to parse yaml and xml description files.
- **stdr_gui:** A gui in Qt for visualization purposes in STDR Simulator.
- **stdr_msgs**: Provides msgs, services and actions for STDR Simulator.
- **stdr_launchers:** Launch files, to easily bringup server, robots, guis
- **stdr_resources:** Provides robot and sensor descripiton files for STDR Simulator.
- **stdr_samples:** Provides sample codes to demonstrate STDR simulator functionalities.

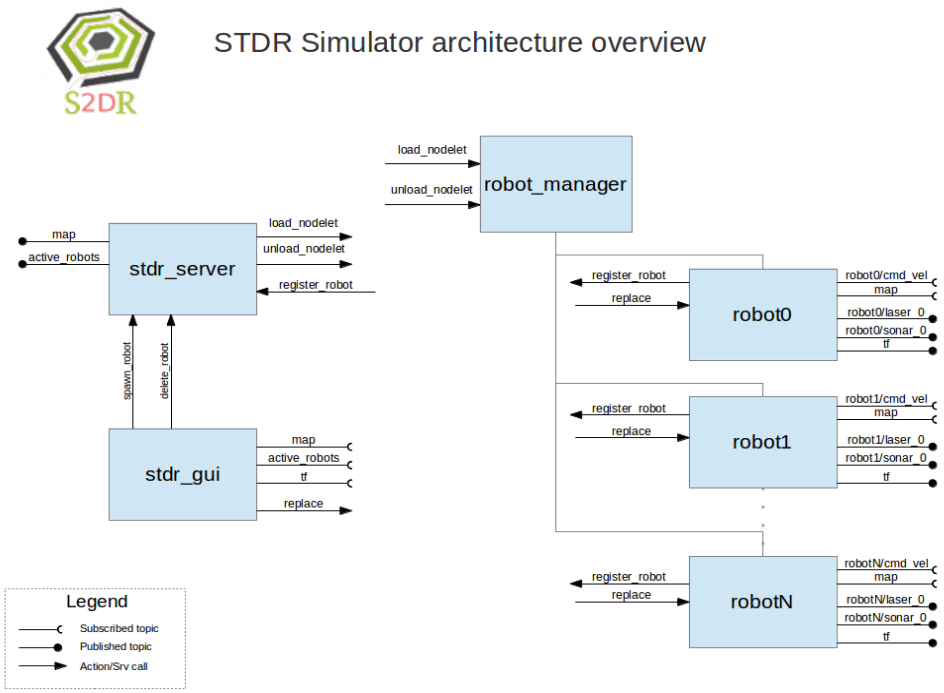

Figure 18: STDR simulator architecture overview [19]

To simulate the wheelchair, an instance of STDR simulator is started and with the help of gui a map is inserted. On this map, a pre-created robot is added. Using the teleop interface running on a different terminal robot can be moved around. After setting up this structure, the necessary topics needed for simulating the application such as laser scan messages, odometry messages, map server node will be ready.

Since map server instance is modified, part related to update of the map from device with update map call can only be tested by running this node after creating a new ros master node and running modified map server node in a different terminal. To be able to use this map server instance with STDR simulator, there should be some modification in map server such as a service for adding a robot.

Figure 19: STDR simulator GUI with map, robot, and laser scan

5.1.3 RVIZ

RVIZ is a 3D visualization tool for ROS. Below commands are used to install RVIZ in ubuntu from a terminal:

sudo apt-get install ros-indigo-rviz
source /opt/ros/indigo/setup.bash
roscore &

To start using the tool, one of the below commands is enough to run it from a terminal:

rviz or *rosrun rviz rviz*

RVIZ is used to visualize the map, odometry, and laser scan in 3D. For visualizing these layers, necessary topics should be chosen from the menu of add button. For showing the odometry tf layer should be added.[20]

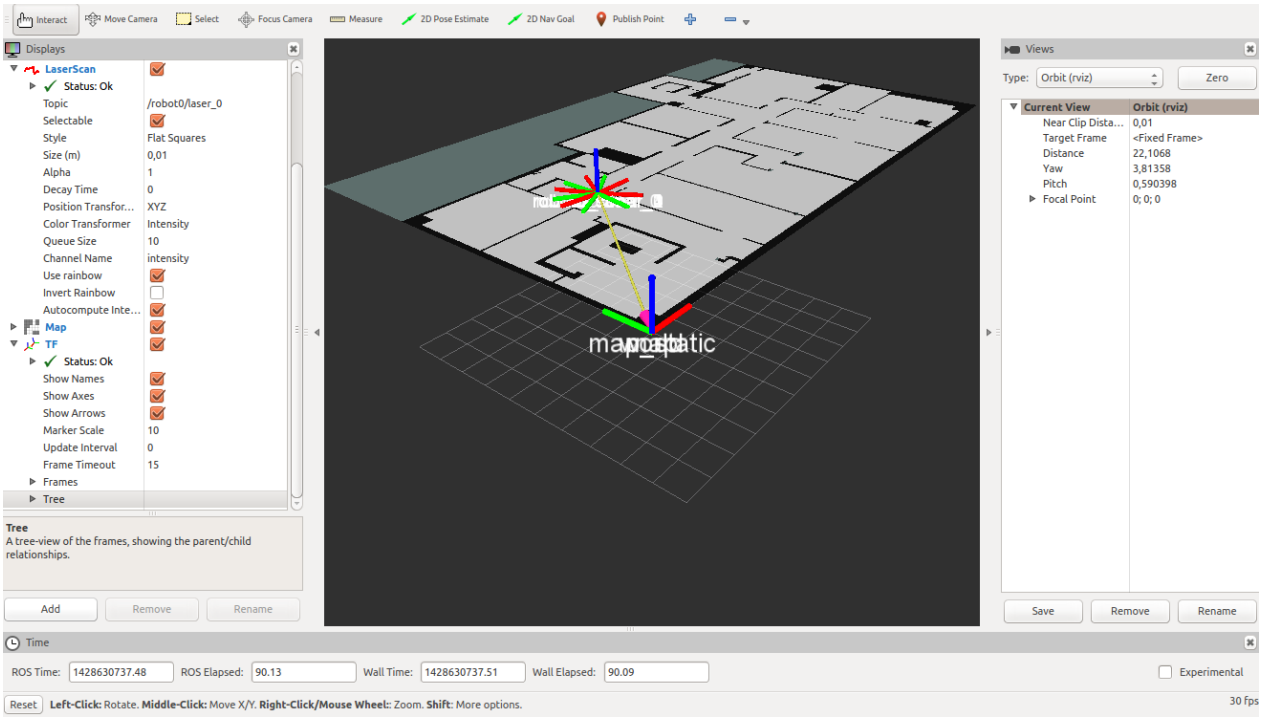

Figure 20: rviz screen with map, tf and laser scan layers

5.2 Setting up the development environment

5.2.1 Installing the application on a mobile device

The application can be installed on a mobile device by following these 6 steps:

1- In order to run the application, Android Studio should be installed and android sdk should be configured as explained in section 5.1.1. Android Studio IDE is started by navigating to bin subfolder of installment from a terminal and running ./studio.sh command.

2- Previously downloaded source code of the application is imported from Import Project option by navigating to the folder containing the project. If all the configurations are done correctly project should be ready to install on the device.

3- The mobile device should be connected to the computer with a usb cable. The necessary usb drivers should have been installed from Android SDK Manager.
4- Before installing the application on the device, developer options should be turned on. In the new version of android devices, developer options are hidden by default. To activate the developer options from the “About Device” option in the settings, the “build number” of the device should be tapped 7 times. After enabling, developer options will be present on the settings list.
5- USB debugging third party app installation should be enabled. When the device is connected with the correct configurations, a dialog will be present to allow the computer and after taping on ok device should be ready for installation.
6- When all the configurations on the device and computer are done, application can be installed by clicking on run option from Android Studio. For debugging purposes, debug option should be used to install the application.

5.2.2 Setting ROS environment and running simulator

For simulating the wheelchair from the computer, ROS environment should be created. In this work, indigo version of the ROS framework is used.

According to installation tutorial of ROS Indigo [21]:

Ubuntu repositories should be configured to allow "restricted," "universe," and "multiverse." Ubuntu guide [ref] can be followed for *instructions* on doing this.

Setup computer to accept software from packages.ros.org:

sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu trusty main" > /etc/apt/sources.list.d/ros-latest.list'

Setup keys:

wget https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -O - | sudo apt-key add

Make sure Debian package index is up to date:

sudo apt-get update

If there are dependency issues, for installing some additional system dependencies:

Ubuntu Trusty 14.04.2:

sudo apt-get install xserver-xorg-dev-lts-utopic mesa-common-dev-lts-utopic libxatracker-dev-lts-utopic libopenvg1-mesa-dev-lts-utopic libgles2-mesa-dev-lts-utopic libgles1-mesa-dev-lts-utopic libgl1-mesa-dev-lts-utopic libgbm-dev-lts-utopic libegl1-mesa-dev-lts-utopic

Ubuntu 14.04:

sudo apt-get install libgl1-mesa-dev-lts-utopic

After setting these configurations for Desktop install with ROS, rqt, rviz, and robot-generic libraries:

sudo apt-get install ros-indigo-desktop

For installing individual packages replacing the underscores with dashes of the package name :

sudo apt-get install ros-indigo-PACKAGE

Before using ROS, rosdep should be initialized which enables easily installing system dependencies for source wanted to be compiled and is required to run some core components in ROS.

sudo rosdep init
rosdep update

For adding ROS environment variables automatically to bash session every time a new shell launched:

```
echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

For creating a catkin workspace:

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/src
catkin_init_workspace
cd ~/catkin_ws/
catkin_make
```

catkin_make command is a convenience tool for working with catkin workspaces.[22] In the current directory there should be 'build' and 'devel' folders. Inside the 'devel' folder there are several setup.*sh files. Sourcing any of these files will overlay this workspace on top of the environment. To source new setup.*sh file:

```
source devel/setup.bash
```

Now workspace should be ready to build new packages. Source code of modified version of map server(map_server_pmk) should be placed under src folder in catkin_ws folder. STDR simulatorshould be cloned in to src folder. After clonning packages, workspace can be build.

```
mkdir src
cd src
//command for cloning map_server_pmk goes here
git clone https://github.com/stdr-simulator-ros-pkg/stdr_simulator.git
cd ..
rosdep install --from-paths src --ignore-src --rosdistro $ROS_DISTRO
catkin_make
```

To add the workspace to ROS environment generated setup file should be sourced:

```
. ~/catkin_ws/devel/setup.bash
```

For driving the robot from terminal teleop_twist_keyboard package should be installed:

```
sudo apt-get install ros-indigo-teleop-twist-keyboard
```

After following the instructions above, simulator can be started and tested by connecting with mobile device.

Similar to catkin workspace, rosjava workspace is needed for generating the library of MapUpdate service in jar format which is already included in the application:

mkdir -p ~/rosjava
wstool init -j4 ~/rosjava/src
https://raw.githubusercontent.com/rosjava/rosjava/indigo/rosjava.rosinstall
source /opt/ros/indigo/setup.bash
cd ~/rosjava
rosdep update
rosdep install --from-paths src -i -y

After clonning the “map_server_pmk” package in the “src” folder, under “rosjava” workspace library including the necessary classes to make service call from android is generated with a build:

catkin_make

Generated jar file can be found under:

~/rosjava/build/map_server_pmk/java/map_server_pmk/build/libs

To start simulator following command should be run from a shell:

export ROS_IP={ip-of-computer}

Simulator without map and robot can be started with from the same shell:

roslaunch stdr_launchers server_no_map.launch

To test update map feature of application, an instance of map_server_pmk should be started in a different shell while ros master is running. This test can be done also by starting a master with *roscore* command in a different shell instead of simulator.

rosrun map_server_pmk map_server {path-to-yaml-file}

To test other features gui of the simulator can be used. Gui can be started with:

roslaunch stdr_gui stdr_gui.launch

Map and robot (pandora robot example can be used) can be initialized from gui using the options on the upper panel. After starting the map, robot can be chosen and a click on the map adds the robot instance. To simulate the movement of the wheelchair in a different shell:

rosrun teleop_twist_keyboard teleop_twist_keyboard.py

cmd_vel:=robot0/cmd_vel

On the shell, instructions for moving the robot with keyboard inputs can be found.

All the configurations should be ready for testing. To be able to test features properly mobile device and computer running ROS master should be connected to same wifi. Following section explains how to use the application giving instructions to test after setting up the environment. As it was mentioned before, to test update map feature, map_server_pmk node should be running. Other features can be tested by using GUI for configurations. If the loaded map is a big image file, GUI might crash. To load a new map, ros master should be stopped and a new instance without map should be started again. Robot can be removed by using right click menu of the robot.

5.2.3 User manual

5.2.3.1 Starting the app: Home Screen

When the application is started, the HOME screen is displayed. (See Figure 21) On the left panel, there are the buttons for navigating between screens. Below these buttons, there is information panel, which gives information about the current map that will be previewed. Right panel of the main screen is empty if the preview is not started. By default, the last map used by the user is selected as the current map. The current map can be selected from available maps list in MAPS Screen. To change the chosen map, a map from the list should be sent to wheelchair in maps screen.

If there is no image file for the current map (the last used map is chosen by default), user is warned with a dialog and option for preview is hidden (Figure 22).

When user navigates to home screen, if wifi of device is not enabled, a dialog warns user (Figure 23).

5.2.3.2 Settings Screen

SETTINGS screen (Figure 24) can be viewed by clicking on the settings button from navigation panel. This screen is used to configure of variables to connect wheelchair. In a similar way to the HOME screen, there are navigation buttons on the left panel. Below navigation buttons there is an information panel showing the status of the test connection and width, height, and resolution of the map published by the wheelchair. On the right panel, there is a form for setting the variables for the connection. Default values, which are the values used for the last connection, are set on start. Below the form, there are buttons to save the values and testing connection.

If wifi is disabled while trying to connect, a warning dialog is shown (Figure 23).

Values from settings screen and their meaning:

Wheelchair Name: ROS node name for the wheelchair
Wheelchair IP: IP of the wheelchair
Wheelchair Port: Port of wheelchair for ROS
Odometry Topic: Topic name to get odometry messages from the wheelchair
Laser Topic:Topic name to get laser scan messages from the wheelchair
Wheelchair Username: Username for SSH connection to wheelchair
Wheelchair Password: Password for SSH connection to wheelchair

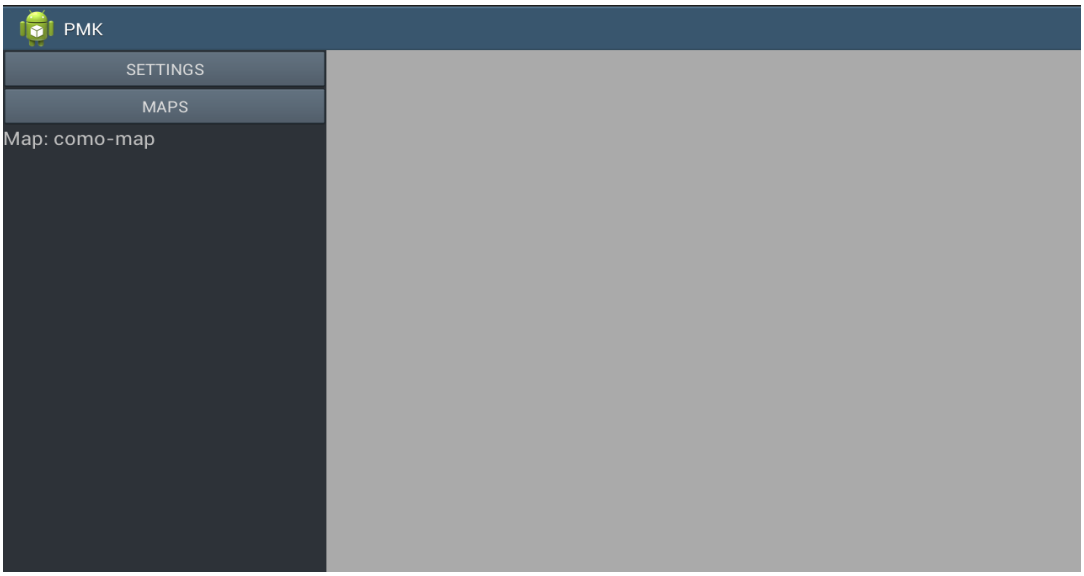

Figure 21: Home screen on start

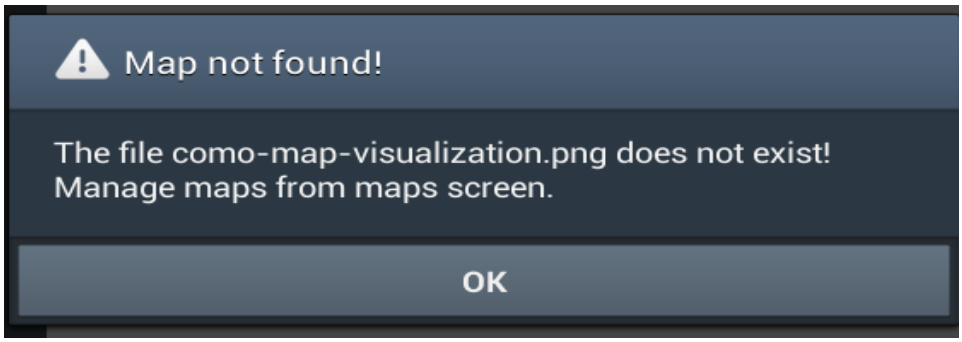

Figure 22: Alert dialog when map does not exist

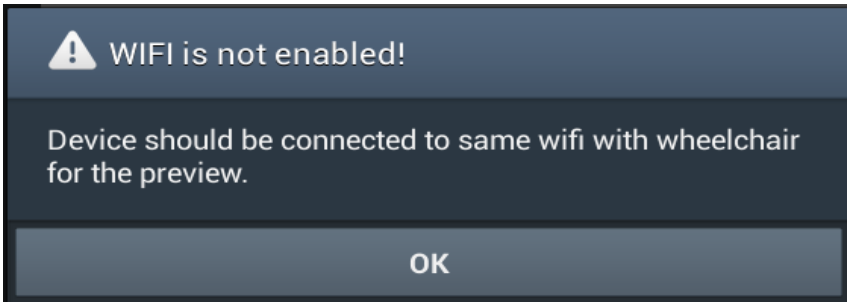

Figure 23: Alert dialog when wifi is disabled

5.2.3.3 Maps Screen

MAPS screen has navigation panel on the left and two tabs on the right (Figure 25). On the first tab, there is a list of already downloaded (available) maps. When a map is selected by tapping on one of the names from the list of maps, on the right of the list there is the parse option for generating the necessary files for wheelchair with a resolution value given by the user. If the files are already generated, there is also a send option for sending the map to the wheelchair and change the chosen map to be previewed in the home screen.

Parse button retrieves IndoorGML file from devices storage for the selected map from the list and generates two files to send to the wheelchair. These files are; YAML file containing metadata of the map, and image file with png format containing the occupancy data of the map. Moreover, another image file is generated for visualization of the map on HOME screen. This image file also holds the occupancy data for the map, however the resolution of this image is a fixed value (10 pixels per meter) while the image to send to the wheelchair has the resolution value typed by user for parse action.

Send button connects to the wheelchair and sends the selected map from the list to the wheelchair. Generated YAML file and png file from the storage of the device are sent to wheelchair using the variables (IP, username, password) configured in SETTINGS Screen (Figure 24). After sending these files, map server running on the wheelchair is updated so that wheelchair will start using the new map, which is just sent from the application.

In the download section (Figure 26) there is the form and button for downloading a map from ALMA server. User should enter the url of the map and a name for the downloaded map. In order to use a map for visualization in HOME Screen, user should parse the downloaded map with a resolution value from the other tab in MAPS Screen containing the list of available maps. After parsing the downloaded IndoorGML file, generated files by parse action should be sent to wheelchair using send button.

If download action fails, there is a dialog for warning user to control internet connection and url of the map.

Figure 24: Settings screen with map info after a successful test connection

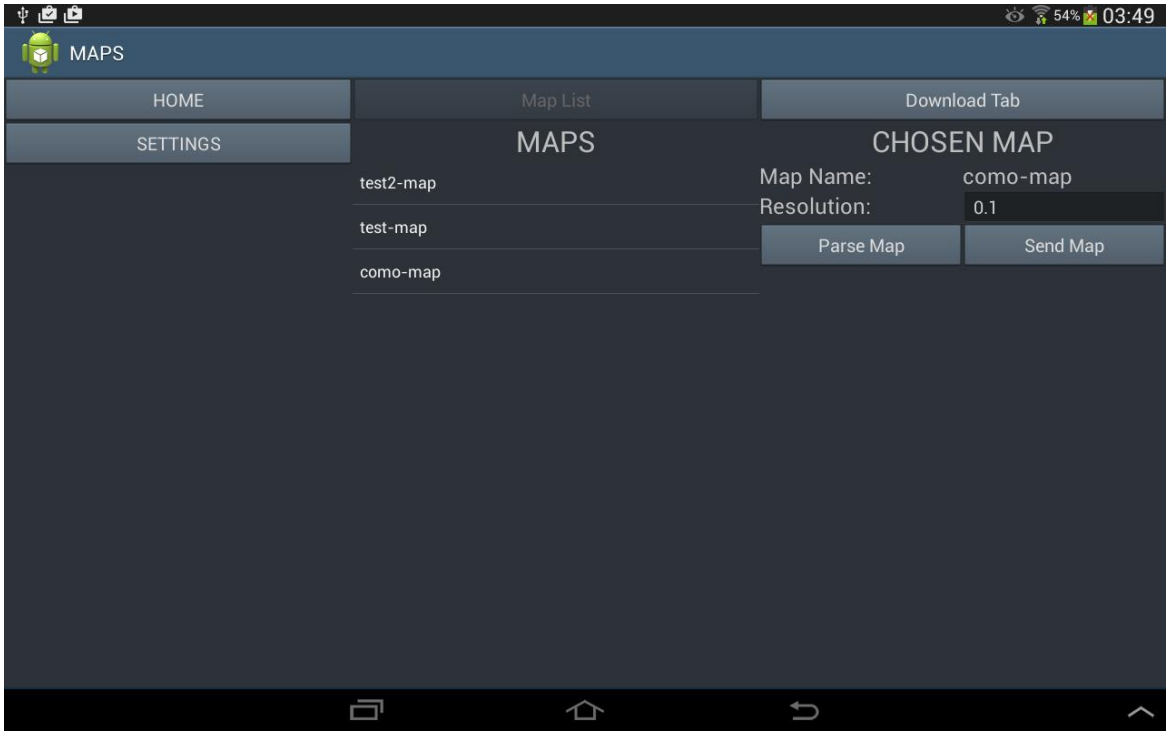

Figure 25: Maps screen with a selected map from list

5.2.3.4 Home Screen with Preview

After sending a map to wheelchair, the current map to be previewed is chosen as the one sent to wheelchair. In the home screen preview can be started for this map, using preview button.

While preview is running, position of the wheelchair on the map and value of the zoom is shown on info panel (Figure 27). Laser scan can be enabled by ticking the show sensor checkbox (Figure 28). User can zoom in/zoom out using the buttons on the right upper corner of the screen or by using pinch gesture on the map. Disconnect button is used to stop visualization of the wheelchair movement.

When the preview is running with laser scan, there is a blue circle representing the wheelchair, a yellow circle representing the direction of the wheelchair, and red lines representing the laser scan retrieved from the wheelchair. If red lines are shorter than their maximum range, an obstacle is detected in that direction.

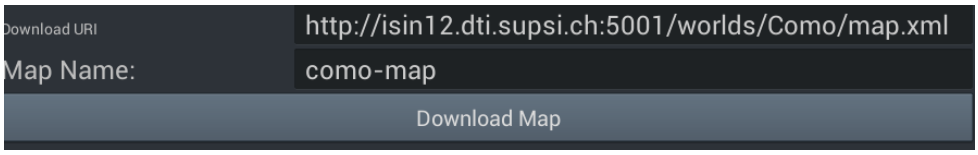

Figure 26: Download section in maps screen

Figure 27: Preview without laser scan

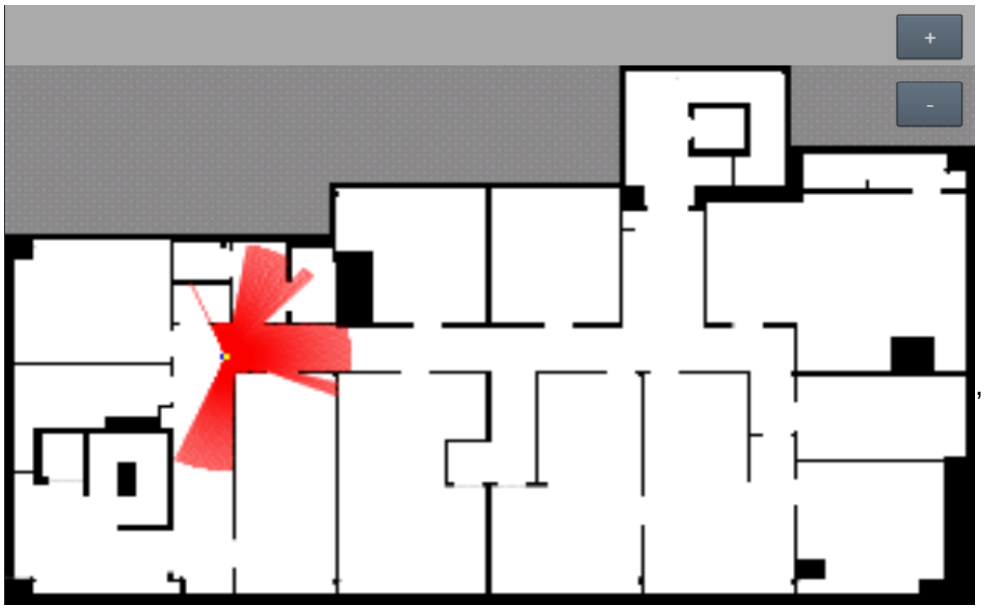

Figure 28: Visualization with laser scan

Chapter 6

Conclusions and Future Work

In this thesis, the problem tackled is necessity of an easy to use interface for previously developed autonomous wheelchairs. Users should be able to receive feedback about the wheelchair movement and they should be able to configure and use the wheelchair easily with a familiar user interface.

The result of the work is an android application as the user interface for Personal Mobility Kit of ALMA project. This application will be a base for the future improvements to have a complete user interface. The complete user interface will be integrated with user interface of PNA (Personal Navigation Assistant) module of ALMA project.

With the proposed implementation, occupancy data maps are generated successfully from IndoorGML maps in the format needed by PMK. With this attempt wheelchair can access to occupancy data of maps easier and quicker comparing to the creation of maps using gmapping method. These maps are sent to the wheelchair via application and new maps can be used by the wheelchair without requiring a restart of the overall system. Having the proper data of maps for the wheelchair easier, makes the system more adaptable and extends the use of system to more users in different environments.

Integration between android environment and ROS framework running on the wheelchair has been established using rosjava for android framework. This enables mobile devices to retrieve collected or generated data of wheelchair and to give commands to wheelchair by calling services of the wheelchair.

Data retrieved from wheelchair is used for the visualization of the wheelchair odometry (position, direction, velocity on the map), wheelchair movement and laser scan (obstacles around). The visualization is implemented in a simple way and aims to give feedback to users about the current state of the system.

The connection between the wheelchair and the mobile device needs a good wifi connection to be able to give real time response and feedback to users. This aspect can be improved by giving options to use different types of connections such as bluetooth technology.

For the generation of occupancy data from IndoorGML files, users are supposed to enter the direct url address of the map on ALMA server. This implementation decreases the usability of the application. A better solution could be giving a list of available maps to user in order to choose and download the required map. In order to implement such an improvement, ALMA server needs to be modified for publishing the list of available maps.

Since occupancy data can be generated by using ROS framework and laser scans with gmapping, a new service can be introduced to retrieve the map published by wheelchair to device for visualization purposes.

Users are supposed to fill some configuration values such as published topic names of the odometry messages and laser scan messages. Considering that users will not have technical knowledge, this configuration process needs to be simplified. A solution could be creating a fixed service on the wheelchair software system publishing the necessary data in a way that application can recognize and use automatically for configuration. This would reduce the required input to basic data needed to establish a connection and all the rest can be automatized. A new ROS node can be implemented which encapsulates all the necessary features to have one single communication

channel between two modules of the system (wheelchair and mobile device). This approach would ease the implementation of a similar solution for devices running with different operating systems such as IOS, windows mobile.

Visualization process can be improved and different types of feedback methods such as audial can be added beside the visual one. All different types of input methods supported by mobile devices can be introduced for getting commands from users.

Some extra features can be introduced in the future such as driving the wheelchair with user commands.

Bibliography

[1] F. Fontana, M. Gianfreda, *Design and implementation of an android based personal indoor navigation assistant*, Master's thesis, Politecnico di Milano, 2013.

[2*] D1.1 End user and user interface requirements*, Deliverable of ALMA Project, 2013.

[3] L. Calabrese, *Robust Odometry, Localization and Autonomous Navigation on a Robotic Wheelchair*, Master's thesis, Politecnico di Milano, 2014.

[4] S. Ceriani, *Sviluppo di una carrozzina autonoma d'ausilio ai disabili motori*, Master's thesis, Politecnico di Milano, 2008.

[5] JL Ki-Joune Li and Jiyeong Lee. *Indoor spatial awareness initiative and standard for in-door spatial data*. In Proceedings of *IROS 2010 Workshop on Standardization for Service Robot*, volume 18, 2010.

[6] Marcus Goetz and Alexander Zipf. *Extending openstreetmap to indoor environments: bringing volunteered geographic information to the next level*. Rumor M, Zlatanova S, ledoux H (eds) *Urban and regional data management*, Udms Annual, 2011.

[7] Michael Worboys, *Modeling indoor space*. In Proceedings of the *3rd ACM SIGSPATIAL International Workshop on Indoor Spatial Awareness*, pages 1–6. ACM, 2011.

[8] Ki-Joune Li, *OGC Candidate Standard for Indoor Spatial Information IndoorGML documentation*, accessed April 10, 2015, last updated 09, 2013, http://indoorgml.net/

[9] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, *ROS: an open-source Robot Operating System, IEEE International Conference on Robotics and Automation*, 2009.

[10] *ROS Documentation*, accessed April 10, 2015, http://wiki.ros.org

[11] R.B. Rusu, *ROS - Robot Operating System*, Tutorial Slides, November 1, 2010

[12] *ROS Odometry messages documentation*, accessed April 10, 2015, http://docs.ros.org/api/nav_msgs/html/msg/Odometry.html

[13] *ROS Odometry Tutorial*, accessed April 10, 2015, http://wiki.ros.org/navigation/Tutorials/RobotSetup/Odom

[14] *ROS Laser scan messages documentation*, accessed April 10, 2015, http://docs.ros.org/api/sensor_msgs/html/msg/LaserScan.html

[15] *ROSJAVA_CORE Documentation*, accessed April 10, 2015, http://rosjava.github.io/rosjava_core/latest/

[16] *ANDROİD_CORE Documentation*, accessed April 10, 2015, http://rosjava.github.io/android_core/latest/

[17] *Android Studio and SDK dowload tutorial*, accessed April 10, 2015, https://developer.android.com/sdk/installing/index.html?pkg=studio

[18] *Android SDK install packages tutorial*, accessed April 10, 2015, https://developer.android.com/sdk/installing/adding-packages.html

[19] *STDR simulator tutorial*, accessed April 10, 2015, http://wiki.ros.org/stdr_simulator/Tutorials

[20] *RVIZ User Guide*, accessed April 10, 2015, http://wiki.ros.org/rviz/UserGuide

[21] *Installing and Configuring Your ROS Environment*, accessed April 10, 2015, http://wiki.ros.org/indigo/Installation/Ubuntu

[22] *Building a ROS package tutorial*, accessed April 10, 2015, http://wiki.ros.org/ROS/Tutorials/BuildingPackages

[23] *ALMA Integration and Planning Module*, ALMA Server, accessed April 15, 2015, http://isin12.dti.supsi.ch:5001/worlds/ComoSim/map