Master's Thesis

Master in Telecommunication Engineering

Autonomous mobility for an electronic wheelchair

Wiktor Jaworski

Supervisor: Jordi Carrabina Bordoll

Microelectronica i Sistemes Electronics

Escola Tècnica Superior d'Enginyeria (ETSE)
Universitat Autònoma de Barcelona (UAB)

June 2019

UAB

El sotasignant, *Jordi Carrabina Bordoll*, Professor de l'Escola Tècnica Superior d'Enginyeria (ETSE) de la Universitat Autònoma de Barcelona (UAB),

CERTIFICA:

Que el projecte presentat en aquesta memòria de Treball Final de Master ha estat realitzat sota la seva direcció per l'alumne Wiktor Jaworski.

I, perquè consti a tots els efectes, signa el present certificat.

Bellaterra, *25 Jun 2019*.

Signatura: Jordi Carrabina

Resum:

Aquí comença el resum del projecte...

Resumen:

A pesar del rápido desarrollo de las tecnologías medicas sector de la salud todavía no ofrece ningún remedio universal para las personas sufriendo de falta de control motor. Dependiente del rango de discapacidad las personas requieren rehabilitación y ayuda para realizar AC (actividades cotidianas). Para ayudar a las personas afectadas por discapacidad y relevar de algunos deberes la gente que los soporta se desarrolló la silla de ruedas electrónica. Una de las funciones de ya mencionada silla de ruedas debería ser la navegación autónoma con reconocimiento de voz. Entonces el objetivo principal de este proyecto fue extender la solución existente con todo el hardware y software necesarios para que la navegación autónoma sea posible. El proyecto resultado en creación de un sistema versátil capaz de mapear el espacio de trabajo y navegar en entornos también conocidos y desconocidos. El sistema permite detección y evitación dinámica de obstáculos, soporta comportamientos básicos de recuperación y acepta objetivos de navegación proporcionados por el software de reconocimiento de voz.

Summary:

Despite the rapid development of medical technologies the health sector does not yet offer any universal remedy for people suffering from permanent impairment of motor functions. Individuals depending on the range of disability require rehabilitation and help to perform the ALDs (activities of daily living). To aid people affected by the impairment and relieve from some duties the ones responsible for helping them the electronic wheelchair was developed. One of the functions of the electronic wheelchair is supposed to be autonomous navigation with speech recognition. The main objective of this project was to extend the existing electronic wheelchair solution with all necessary equipment and software necessary to make the autonomous navigation possible. As a result, a versatile system was created capable of mapping the working space and navigating in both known and unknown dynamic environments. The system allows dynamic obstacle detection and avoidance, basic recovery behaviors and accepts navigation goals provided by speech recognition.

Contents

List of Figures

List of Tables

List of Abbreviations

SCI	**S**pinal **c**ord **I**njury
WHO	**W**orld **H**ealth **O**rganization
ALS	**A**myotrophic **L**ateral **S**clerosis
ICU	**I**ntensive **c**are **U**nit
CAN	**C**ontroler **A**rea **N**etwork
SLAM	**S**imultaneous **l**ocalization **a**and **m**apping
DOF	**D**egrees **o**f **F**reedom
EC	**E**xteroceptive **S**ensor
PC	**P**roprioceptive **S**ensor
PaS	**Pa**ssive **S**ensor
AcS	**Ac**tive **S**ensor
LIDAR	**L**ight **d**etection **a**nd **r**anging
ToF	**T**ime **o**f **F**light
SONAR	**So**und **na**avigation **r**anging
vSLAM	**v**isual **S**imultaneous **l**ocalization **a**and **m**apping
RGB-D	**R**ed **G**reen **B**lue **D**epth
IMU	**I**nertial **M**easurement **U**nit
MEMS	**M**icro **E**lectro **M**echanical **S**ystem
RANSAC	**Ran**dom **Sa**mpling Consensus
LQE	**L**inear **Q**uadratic **E**stimation
IoT	textbfInternet **o**f **t**things
LTSM	**L**imultaneous **s**hort-**t**erm **m**emory
ROS	textbfRobot **O**perating **S**ystem
AHRS	**A**ttitude **H**eading **R**eporting **S**ystem
TCP	textbfTransmission **C**ontrol **P**rotocol
SPI	**S**erial **P**eripheral **I**nterface

Dedicated to the memory of my grandmother, Hanna, who despite the difficulties of amyotrophic lateral sclerosis she never lost her spirit and love for her family. You are gone but the dream of helping You made this journey possible.

Chapter 1

Introduction

1.1 Paraplegia, spinal cord injury, paraplegia and statistics

Paraplegia - Paralysis of the legs and lower body, typically caused by spinal injury or disease. From Greek paraplēgia, from paraplēssein 'strike at the side', from para 'beside' + plēssein 'to strike'. - The Oxford English Dictionary [1]

Impairment known as paraplegia is a very well-known problem and can be observed everyday in the streets of all modern human settlements. This chapter will not focus on one kind of motor impairment but will generally be addressing the problems connected with living with this disease.

1.1.1 Causes of paraplegia

There are many ways to become a victim of motor impairment disease but there are two main sources. First type of cause is the neural sicknesses of genetic or congenital nature. Aforementioned reason of sickness is minimally influenced by the human behavior factor. The second type of cause are the **SCI**s, the spinal cord injuries. Damaging the spine that holds the most imporant neuronal paths at certain point will cuts the connection to the lower parts of the body making person lose control over it. The severity of the injury can be very different depending on point and amount of damage delivered (*see: Table 1.1*).

According to **WHO** the staggering 90% of the causes are due to traumatic injuries. Research conducted by U.S. *National Spinal cord Injury Statistical Center* at *University of Alabama at Birmingham* reveals that the most common reasons of spinal cord injuries are vehicular crashes and falls (*see: Figure 1.1*) making up more than 71 percent of the cases [2].

Classification			
ASIA A	ASIA B	ASIA C	ASIA D
Complete loss of sensory function and motor skills below the injury	Some sensory function below the injury, but no motor function	Some motor function below level of injury, but half the muscles cannot move against gravity	More than half of the muscles below the level of injury can move against gravity

TABLE 1.1: SPI severity scale by American Spinal Injury Association

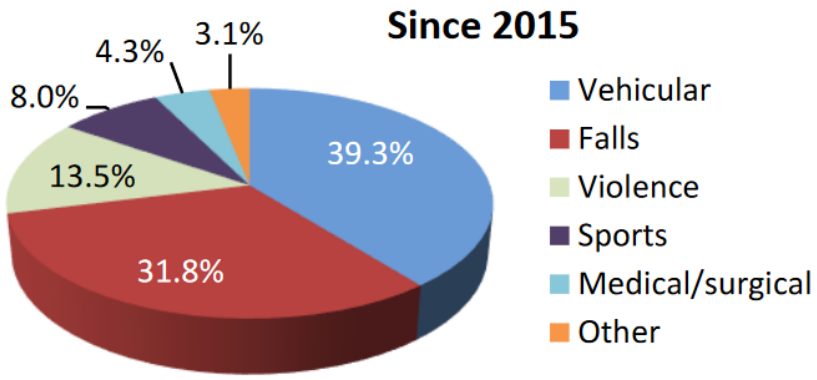


FIGURE 1.1: Causes of spinal cord injuries [2]

The other reason is violence (especially gunshot wounds) but this aspect of the statistical research cannot be generalized for more countries due to U.S. gun enforcement laws. The statistics vary very much in many countries. **WHO** states that causes of SCI vary depending on the region of the world [3]:

- In Easter Mediterraean and South-East Asia most people suffer from *SCI* because of the falls, making up to 40% of the cases;
- In African region leading cause are road traffic accidents - almost 70% of the cases;
- In Western Pacific region the vehicular accidents are also the main reason however it only reaches 55% of the cases.

The other, way less occurrent reasons of **SCI** are non-traumatic - mostly conditions such as tuberculosis, *spina bifida* or tumors.

1.1.2 Prevalence of paraplegia

Despite the rapid development of cutting-edge medical technologies there is still huge group of people suffering from paraplegia or tetraplegia *(paralysis of all four limbs)*. According to the Spanish Statistical Office - *Instituto Nacional de Estadistica* there were more than 485 thousand people affected by some form of motor coordination disability in 2008 [4]. The research done by U.S. *National Spinal Cord Injury Statistical Center* in years 2015-2019 estimate that the number of people suffering from motor disabilities in United States is increasing by approximately 17,730 new cases each year [2]. Studies indicate that roughly twice more males suffer from **SCI** than women with ratios minimum 2 to 1 among adults and the biggest risk of **SCI** occurs between ages of 20 to 29 and above 70 years old [3]. Most people (over 47%) suffer from Incomplete Tetraplegia - that is classified as **ASIA B** at *American Spinal Injury Association severity scale* (see Table 1.1). The other 40% are the victims of complete and incomplete paraplegia. The remaining 12.3% suffer from complete tetraplegia as shown in Figure 1.2.

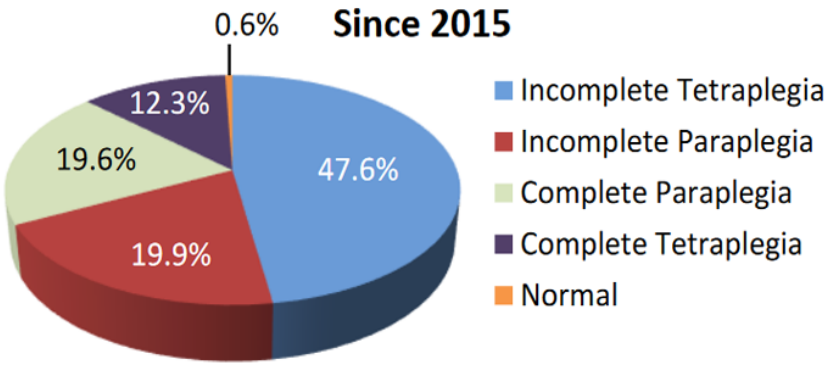


FIGURE 1.2: Statistical neurological severity of damage [2]

1.1.3 Consequences of the injury

As shown in Figure 1.2 less than 1% of the cases manage to undergo complete neurological and motor recovery. The remaining 99% of the people need to live with a very life-disrupting and, for now, medically unsolvable condition. The injury impacts both health and socio-economic aspects of life. From the medical point of view the **SCI** leads to chronic pain and discomfort. That leads to about 25% [2] of the cases to show significant signs of depression and many cases develop secondary conditions that not only are a liability but also may be a true thread to the life - *urinary tract infections, pressure ulcers,*

Status (%)	Injury	After 1 Year	After 10 Years	After 20 Years	After 40 years
66.0	17.4	23.0	28.8	31.8	31.8
8.1	7.5	3.1	1.0	0.3	0.0

TABLE 1.2: Percentage of employed people after in the following years after the **SCI** [2]

respiratory tract problems, deep vein thrombosis. The socio-economic impact of **SCI** is huge. Their participation in socio-communal life is very limited, due to both physical and mental barriers created by **SCI** the unemplyment rate of people with this condition is more than 60%. Only 17% of people with **SCI** are employed one year after the injury with increase to 32% after 30 years after injury (see Table 1.2).

The condition also increase the divorce rate rendering more than 20% of people with **SCI** to become divorced after 10 years from injury. According to **WHO** the life expectancy of people with **SCI** is significantly lower life expectancies of people without **SCI** with very high mortality rate during first year after the injury.

1.2 Objective

1.2.1 Control-Live project

Considering the disadvantages of living with **SCI** described in Section 1.1.3 there is still a very big demand for solutions aiming to make life of people with aforementioned condition better. This is the reason why **Control-Live** started developing a very special solution. They designed the mobile platform with functionalities of both electronic wheelchair and bed that are adapted for the people with limited motor ability. The reason for the construction of the device is making the life of people with motor disabilities and all the staff and family that are involved better (*as presented in Figure 1.3*). The quality of life both para/tetraplegic person and the assisting persons is greatly influenced by the accessibility of supporting technology. The greater the severity of the disability the greater contribution of other people is required to satisfy basic everyday needs of a human.

The modular platform developed by **Control-Live** can assume various configurations depending on desired use. The platform can transform in the

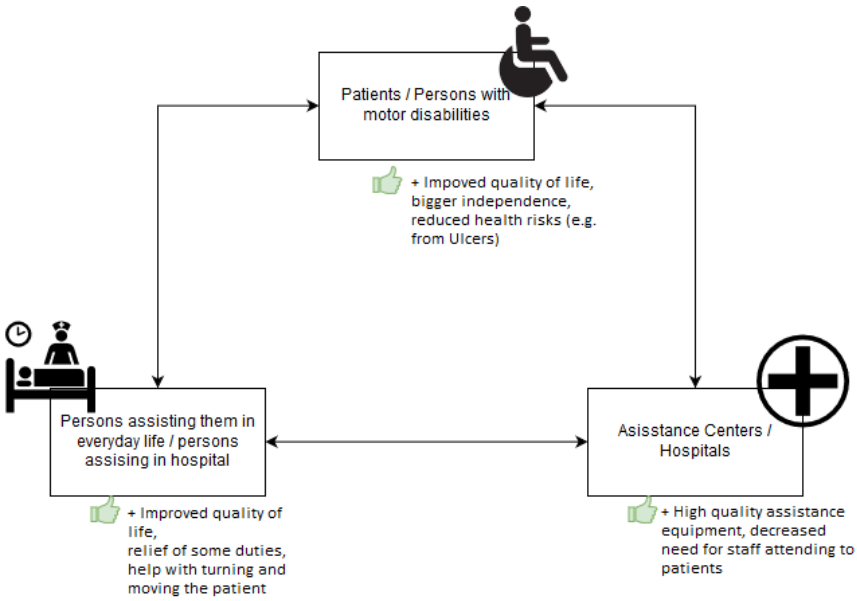


FIGURE 1.3: Target groups of platform developed by **Control-Live** and it's benefits

matter of seconds into the bed (*see Figure 1.4*) or wheelchair mode (*see Figure 1.5*).

Each module of the skeleton creating of the bed can move independently allowing the platform to enter various pre-programmed modes. This is very important feature because it is addressing the problem of great importance - pressure ulcers problem in hospital settings. The prevalence of aforementioned condition varies from 8.3% to 23% in European hospitals. Globally in year 2013 the documented death count caused by pressure ulcers reached up to **29 thousand** deaths [5]. But the target, as claimed by Control-Live, is way bigger than just people with ulcers, the platform will aim to help people with many more conditions such as:

- Spinal cord injury and tetraplegia;
- **ALS** (*Multiple Amyotrophic Latera Sclerosis*), **AMS** (*Advanced Multiple Sclerosis*), Muscular Dystrophies, other advanced degenerative diseases;
- Cereblar palsy, cerebrovascular accidents, strokes;
- Alzheimer's, dementias;
- **ICU** patients;
- Terminal sickness with no mobility.

FIGURE 1.4: Mobility platform assuming the bed configuration

Before the development of this thesis (widely developed the next section 1.2.2) the wheelchair was controlled only manually. The movement commands would be issued using special input unit (*see Figure 1.6*) that sends the joystick displacement to the control unit that translates it to differential drive action vector.

The aforementioned input unit is also equipped with buttons allowing user to control the joints of the platform according to his / hers preferences (*also seen at Figure 1.6*). The Input unit is connected to the controller unit with wider range of uses. The job of the control unit (*Figure 1.7*) is as mentioned before translating the orders of input unit but also communicating with all the platform subsystems using **CAN** bus. The control unit is also equipped with touch screen that allows user to select pre-defined positions of the platform. The positions of the joints can also be saved as a custom configuration.

Sometimes, depending on the severity of sickness or **SCI**, the person may not be able to select the desired position himself - for example if the person suffers from tetraplegia. This problem is addressed by built-in speech recognition module that allows user to control the joints of the platform, enter or exit the wheelchair mode or even make a call for help using spoken commands.

1.2.2 Objective of this work

Considering all the facilities described in Chapter 1.2.1 the wheelchair still lacks the few functions that are required for people with bigger severity

FIGURE 1.5: Mobility platform assuming the chair configuration

of paralysis. Some people may not be able to manually use the joystick to navigate the platform to desired location or may not be able to do it seamlessly. To address this problem the platform needs to be able to receive speech command and move itself to the location specified in the speech command. To do so, the platform needs to not only be able to receive and understand voice commands but to navigate autonomously according to those commands. Solving this problem in a safe, comfortable and useful for both user and environment is not easy. It requires solving the **S**imultaneous **L**ocalization **a**nd **M**apping problem **SLAM**. This is a still-open computational problem that seems easy to solve only at the first glance. The real work to solve it begun 1990 when Hugh F. Durrant-Whyte and his research group suggested few solutions to **SLAM** problem [6] (*see Figure 1.8*). Generally, for a robotic platform, to navigate in new areas there are three actions needed to be done:

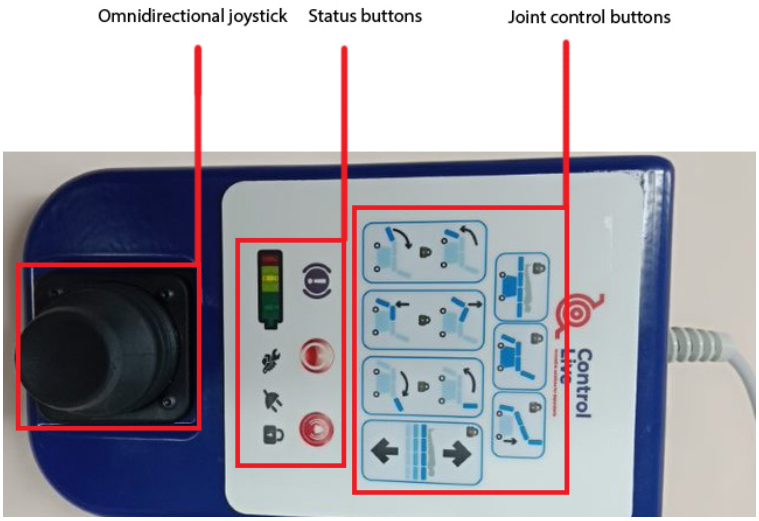


FIGURE 1.6: Input unit of the mobile platform

FIGURE 1.7: Control unit of the mobile platform

- Mapping - the platform needs to create the map of the environment it is currently occupying;
- Localization - as map is created, the platform needs to estimate its localization, to recognize where is it on the map;
- Path planning + Motion Control - the platform needs to create the plan using it's localization on the map and following it by issuing commands to drive that moves the physical platform base.

SLAM faces the same casuality dilemma as question *"which came the first: the chicken or the egg?"* [6]. If the platform is using the same hardware to create the map and localize itself on it, after some time either its localization or the map become very uncertain and the platform will lose the ability to tell

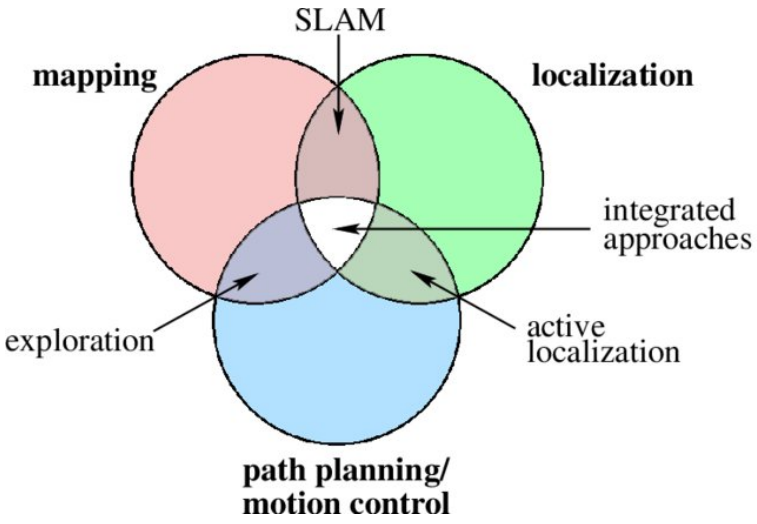


FIGURE 1.8: Main tasks connected with **SLAM** problem to generate accurate models of environment explained in *Robotic Mapping and Exploration* by Cyrill Stachniss [7]

where exactly it is or what exactly it is mapping. Both robot path and map become unknown and the estimate error increases ultimately rendering robot absolutely lost. The consequences of wrong pose estimate may have catastrophic consequences. The same problem but on a much bigger scale and with more random variables is solved in autonomous car challenges such as DARPA Challange (*see Figure 1.9*).

FIGURE 1.9: Great source of inspiration provided by Stannford Team at DARPA Grand Challange 2005 - Sebastian Thrun and *Stanley* - (SLAM-adapted Volkswagen Touareg), resting after being declared the official winners.

This Master Thesis is facing the **SLAM** problem with reduced complexity due to limitation to indoor spaces for the paths of the Control-Life mobile platform that let to much simpler cases that DARPA's complex outdoor roads. Thus, the objective of this Thesis is to provide autonomous mobility to the Control-Life platform what will be done by implementing solutions in following three domains: **SLAM**, active localization and integration with voice control.

1.2.3 Work plan

The entire work flow of the project was managed using the software *Microsoft Teams*. Most of the milestones were done sequentially but whenever possible the work was parallelized. The work flow was planned using *GanttProject* free software. Due to *research and development* nature of the the work plan throughout entire development of the project was changing unexpectedly. Many times the solution that seemed to be the best choice at the moment was shown not that satisfying few days later according to the requirements of the project and needed to be updated. The good example is the few additional days added to the *applying inertial unit and recalculating* due to need to recalibrate the entire system because of the new variable taken into the account that was the input data from accelerometer and triple-axis gyroscope (*Figure 1.11 and Figure 1.12*).

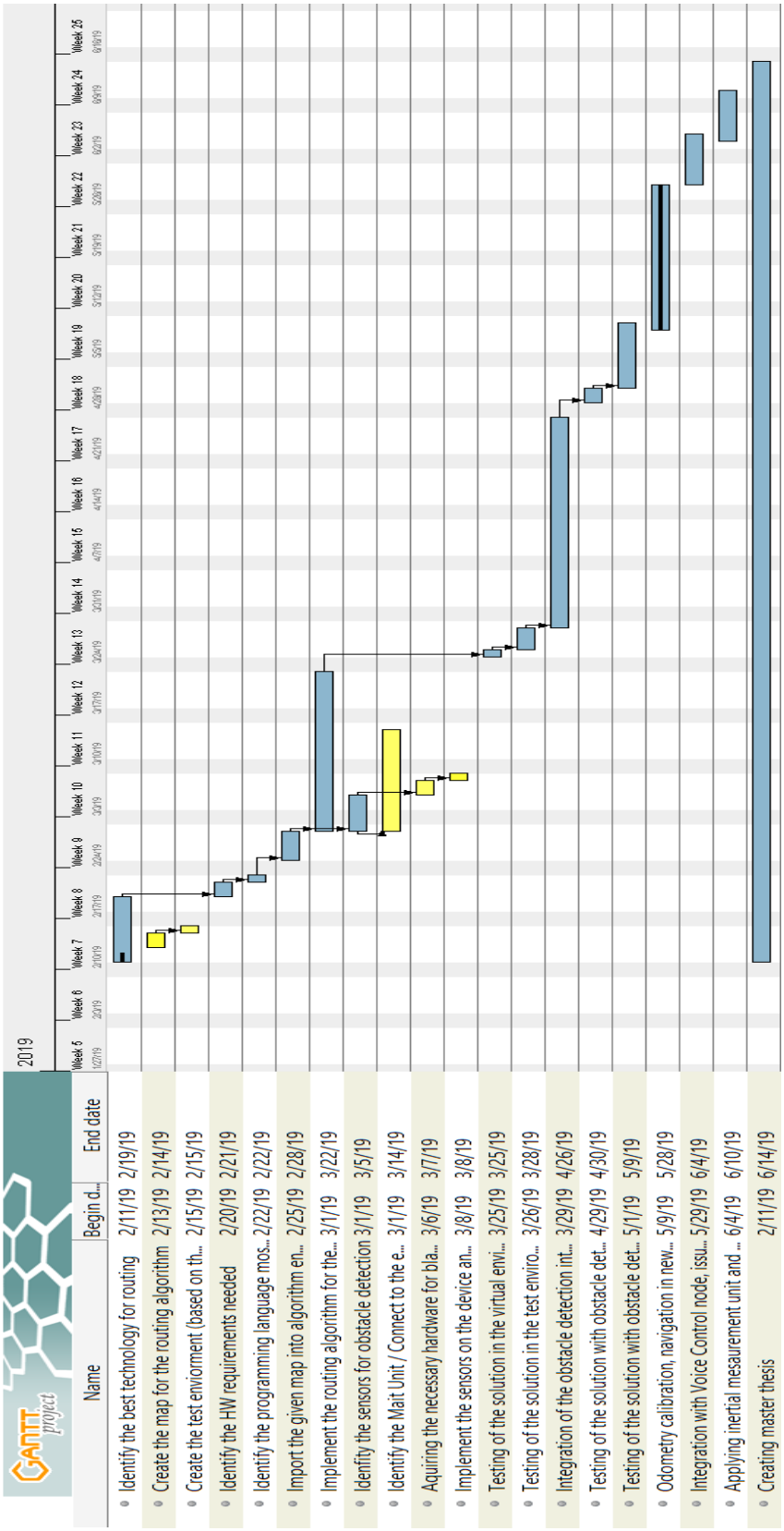


FIGURE 1.10: Work plan developed at the beginning of writing the paper.

Roadmap chronogram

Jun 18, 2019

2

Tasks

Name	Begin date	End date
Identify the best technology for routing	2/11/19	2/19/19
Create the map for the routing algorithm	2/13/19	2/14/19
Create the test enviorment (based on the created map)	2/15/19	2/15/19
Identify the HW requirements needed *Contact Khalid to investigate the possible solutions*	2/20/19	2/21/19
Identify the programming language most suitable for implementation	2/22/19	2/22/19
Import the given map into algorithm enviroment	2/25/19	2/28/19
Implement the routing algorithm for the imported map	3/1/19	3/22/19
Idenfity the sensors for obstacle detection *Contact the sub-contractor :*	3/1/19	3/5/19
Identify the Mait Unit / Connect to the existing peripherals	3/1/19	3/14/19
Aquiring the necessary hardware for autonomous movement	3/6/19	3/6/19
Implement the sensors on the device and connect to MCU	3/6/19	3/6/19
Testing of the solution in the virtual enviroment	3/25/19	3/25/19
Testing of the solution in the test enviroment	3/26/19	3/28/19
Integration of the obstacle detection into the routing algorithm	3/29/19	4/26/19
Testing of the solution with obstacle detection in virtual enviroment	4/29/19	4/30/19

FIGURE 1.11: Work flow with the dates, part 1.

Roadmap chronogram

Jun 18, 2019

3

Tasks

Name	Begin date	End date
Testing of the solution with obstacle detection in test enviroment	5/1/19	5/9/19
Odometry calibration, navigation in new maps	5/9/19	5/28/19
Integration with Voice Control node, issuing orders	5/29/19	6/4/19
Applying inertial mesaurement unit and recalculating	6/4/19	6/10/19
Creating master thesis	2/11/19	6/14/19

FIGURE 1.12: Work flow with the dates, part 2.

Chapter 2

State of the art

2.1 Generally about SLAM problem

After the brief introduction to **SLAM** in *Chapter 1* this section will explain some of the more in-depth aspects of the problem. **SLAM** addresses the problem of building the map of an unknown place by moving an agent while, at the same time localizing, itself in the aforementioned map and creating a path. **SLAM** is not single algorithm, it is a problem that can be addressed using very big variety of algorithms but for now there is not yet perfect solution proposed. We are going to approach **SLAM** as a open concept containing smaller atomized problems that can be solved following different solutions. Human beings solve the **SLAM** with relative ease due to very complex processes happening in the brain, but the general idea is the same. Let's say a person is entering a new building with many rooms - the person enters an unknown environment, looks around observing it, moves around and builds a model of this place. Then, using the same model person compares it with actual observations to know at which room he or she is at. We all do this at the same same time without even noticing, but teaching a artificial construct such as autonomous wheelchair to do it is a non-obvious task. To solve it, it is required to analyze the aforementioned exploration. The person is walking around, the equivalent of this for a autonomous platform is any kind of hardware allowing it to move - it can be a pair of motors in case of simple robots, propellers in case of drones or even manipulation of own robot body in cases that try to mimic the biological creatures (*about this in Chapter 2.2.1*). To create some kind of model of the environment, the mobile agent needs to perform some observations. In human case one of our many senses [8] can be used or most likely the fusion of data from the sensors (*see Figure 2.1*). That means that the autonomous platform needs to be able to gather information about surroundings using some kind of sensors. The variety of possible sensors is

huge, but there are few most used and most important for autonomous solutions (*more info in Chapter 2.2.2*). Later, provided the sensory data, a person looks for characteristic points in each room - called *landmarks* - that allow him or her to estimate in which room he or she is.

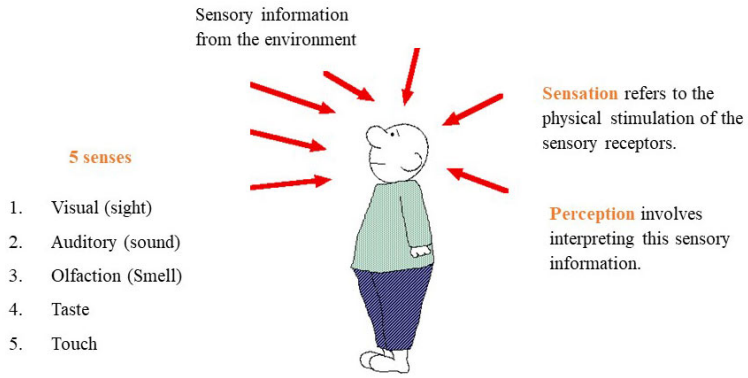


FIGURE 2.1: Humans naturally are using very many inputs to localize themselves in an unknown location. 5 most important senses for human to localize himself in the new environment, *Saul McLeod, Simplypsychology.com*

Providing the aforementioned sensory data the agent can then compare the observed landmarks to landmarks stored in the created model *for example the room A has white walls but room B has red walls*). What if two of the rooms look exactly alike? In case of people the solution can be simple because human can refer the room to the one next to it, or the other solution could be counting the steps and turns knowing exactly at what location the person is occupying. This information does not come from observing the environment but observing changes of own state.

To do the said process we need to follow these steps:

- Extract landmarks,
- Associate data,
- Estimate state,
- Update state,
- Update landmarks.

The following Figures (2.2, 2.3, 2.4, 2.5) were designed to explain that process in detail. First, the mobile agent estimates the location of the landmark using it's sensors (*Figure 2.2*).

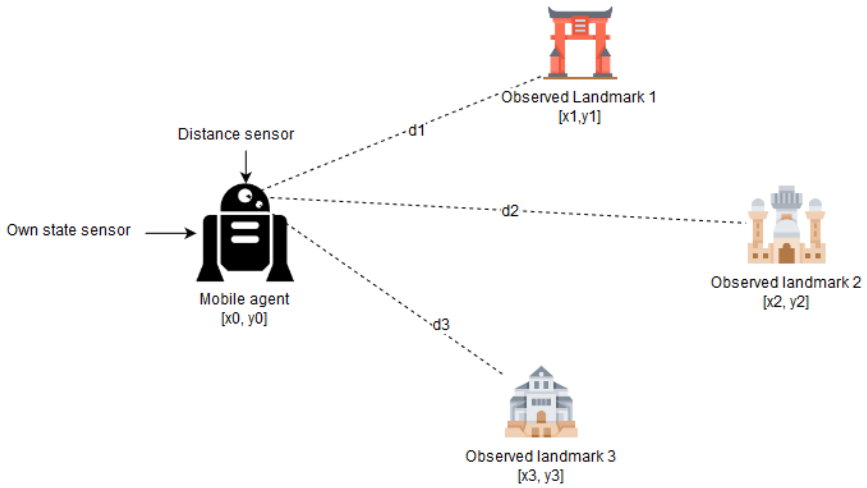


FIGURE 2.2: Moving agent

Then it moves to a new location whose coordinates are estimated by observation at the new state (*according to the analogy of counting your own steps*, *Figure 2.3*).

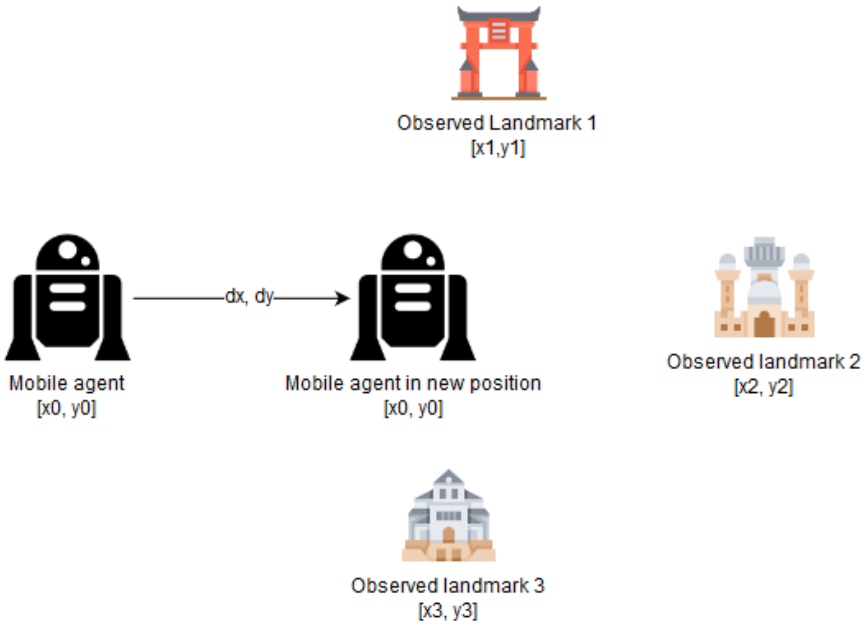


FIGURE 2.3: Agent enters new location

The external observation of landmark location does not correlate with the calculated position coming from the observation of the new state. The the imperfection of either sensor measurements that observe landmarks or on the computations to obtaining (*Figure 2.4*) lead to errors.

Later, depending on the algorithm and because of the uncertainty of pose estimation, the mobile agent will not know with 100% precision it's position,

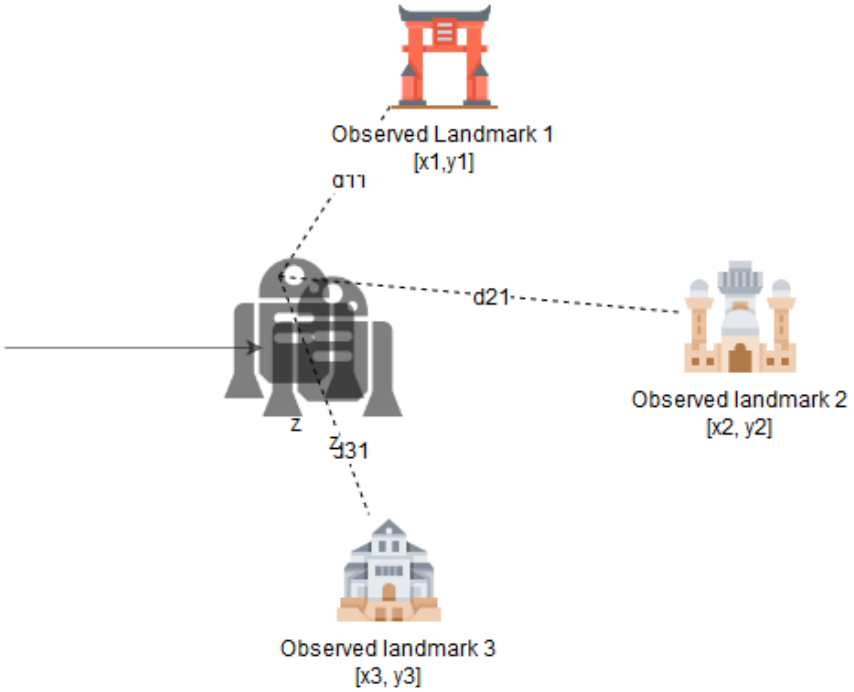


FIGURE 2.4: The uncertainty of agent position

but can estimate it's position according to the probabilistic rules applied in algorithm (*about that in Chapter 2.3.4*).

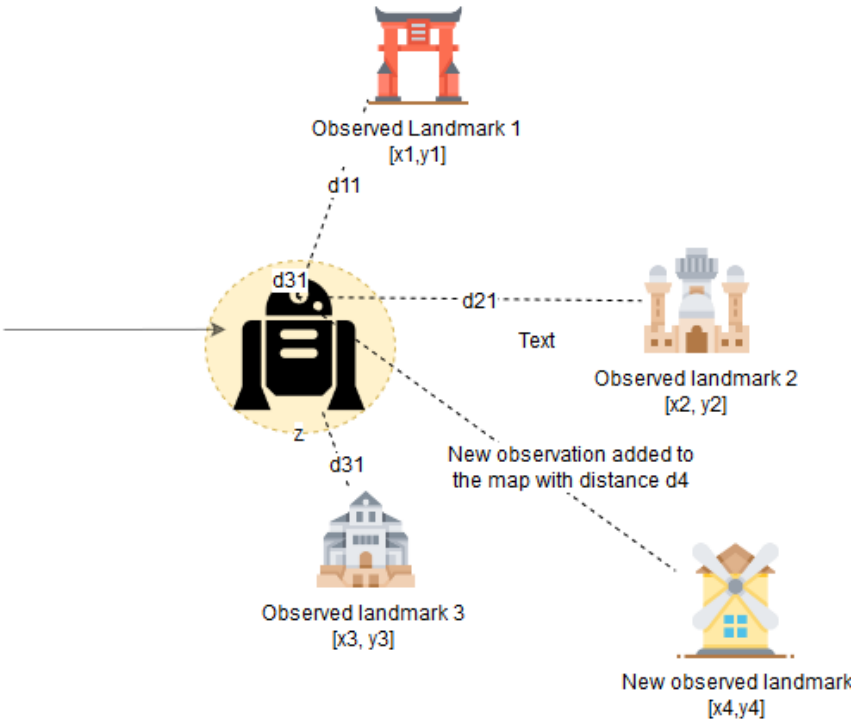


FIGURE 2.5: Agent estimating position using mathematical model and observing new landmarks, completing the **SLAM** time step loop

Afterwards, the platform repeats the process at each time step, summing up the entire process to:

- Movement of the mobile platform, the uncertainty of it's position increase;
- Mobile platform observes unique, interesting features in the environment - *landmarks*. The exact location of the landmarks is also uncertain due to culmination of imperfect precision of sensors and self pose estimate error;
- The platform uses both it's calculated localization and landmark observation to predict it's estimated location using statistical model.

2.2 The hardware - agents and sensors

The minimal system to set up a functioning agent capable of solving **SLAM** problem consists of one sensor that will observe the landmarks and some kind of moving platform. Thanks to robust algorithms and relatively big computational Heterogeneus Multiprocessor Systems-on-a-Chip (MPSoC) mobile phones processors, **SLAM** can be implemented on the mobile phones or similar portable and/or embedded platforms. The moving base would be human hand or whatever the phone is attached to and the sensor would be a monocular camera of the phone using multi-frame depth prediction algorithm to estimate the distance to the objects. This was already achieved in year 2014 [9].

FIGURE 2.6: Global localization from monocular slam on a mobile phone [9].

In this section the most popular mobile platforms and sensors will be discussed and their advantages and disadvantages compared. The general

informations and specifications are provided in *Section 2.2*, to skip to mathematical explanation of state of the art **SLAM** move to *Section 2.3*.

2.2.1 Mobile agents

The necessity of solving the simultaneous localization and mapping problem is getting more popular everyday due to many new inventions based on it. **SLAM** influence stretches from everyday life uses such as cleaning robots (*see Figure 2.7a*) or augmented reality smart-phone applications (*Figure 2.6*) to very specific task machines such as unmanned military drones or surface survey robots (*Figure 2.7b*).

(A) iRobot Roomba i7+, example of everyday use of **SLAM** technology

(B) Hovermap equipped unmanned aerial vehicle used to map the area created by *CSIRO Data 61*. A more sophisticated use of **SLAM**

FIGURE 2.7: Two examples of **SLAM** mobile agents

Depending on the environment and extend of use the mobile platforms will vary in complexity of their structure. When speaking of robots and their movement in space it is necessary to specify their *degree of freedom* - **DOF**. The degrees of freedom are group of independent parameters defining the mechanical system and are crucial to obtain the mathematical description of mobile platform movement [10]. Each agent has different movement possibilities that influence the drive and sensor complexity. Let's consider the three dimensional space perceivable by humans - all possible movements in this kind of space can be described combining 6 *parameters* that corresponds to 6 **DOF**. Every change of position and orientation in this space can be described using following parameters:

- Position of the object at axis forward axis - let's call it X;
- Position of the object at sides axis - Y;
- Position of the object at axis perpendicular to two aforementioned - Z;

- Angular orientation betwen X,Y - Yaw
- Angular orientation between X,Z - Pitch
- Angular orientation between Y,Z - Roll

Figure 2.8 explains visually the concept of degrees of freedom.

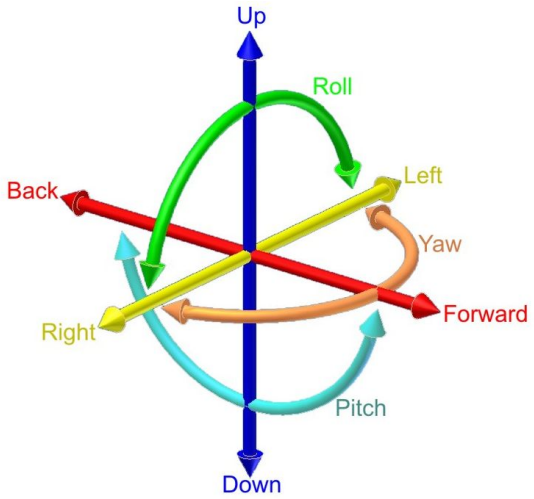


FIGURE 2.8: 6 Degrees of freedom in three dimensional space. [10]

The mobility of an agent can be calculated according to the **Gruebler Equation** that presents itself as following:

$$F = 3 * (N - 1) - 2 * J - H$$

Where **F** is the **DOF** for the kinematic chain, **N** is the number of parts, **J** is number of one **DOF** joints and **H** is the number of higher **DOF** joints. In the cases of mobile agents without any manipulators we can simplify their degrees of freedom considering only the movements of the agent in space (*see Table 2.1*).

2.2.2 Sensors

The key to getting the information about the external world are sensors, they provide the information to the machine just as the senses provide information to human brain. Basing on that both living creatures and machines create the model of the world surrounding them. Using the example from *Chapter 2.1* humans can estimate their own state both by using the external observations - that is looking around, hearing, smelling environment and using

Agent	**DOF**	Explanation
Roomba i7+	3DOF (x,y,yaw)	The robot due to its differential drive navigates in planar space and can only control its position in X,Y and it's angle in it.
Hovermap UAV	6DOF (x,y,z, roll, pitch, yaw)	The drone due to it's orientation in space and many rotors can move itself in three dimensional space and rotation in it.
Elevator	1DOF (y)	For the sake of example let's say elevator in the building is a mobile agent, it can only navigate in one dimension and is cannot rotate along any axis.

TABLE 2.1: Example **DOF** correlated to specific mobile agents

internal observations. Knowing the number of steps and their distance humans can estimate their positions just by evaluating own state. Machines can do the same and have special sensors designed for that [11]:

- Exteroceptive sensors **ES** - the devices used to gather information about the surroundings of mobile agents - sonars, laser scanners, cameras;
- Proprioceptive sensors **PS** - the devices designed to measure own displacement - wheel encoders, gyroscopes, accelerometers.

Sensors also can be classified depending on their interaction with environment:

- Passive sensors **PaS** - measure only entering energy from environment to the sensor - for example cameras, temperature probes, microphones.
- Active sensors **AcS**- obtain information about the ambient by emitting energy and observing environmental reaction to it - for example sonars, laser scanners, wheel encoders.

Below, *Table 2.2* presents sensors classified by the typical use and sorted by their complexity. The sensors highlighted with **bold text** are the sensors later used in *Chapter 4* and their subject is further explained in the *Subsections 2.2.3 and 2.2.4* of this chapter.

Selecting sensors for solving **SLAM** problem is not a trivial task, many characteristics such as:

Typical use	Sensor system	Exteroceptive Proprioceptive	Pasive Active
Proximity sensors	Contact bumpers	ES	Passive
	Proximity sensor	ES	Passive
Motor sensors	Brush encoders	PS	Passive
	Potentiometers	PS	Passive
	Resolvers	PS	Active
	Optical Encoders	PS	Active
	Magnetic Encoders	PS	Active
	Inductive Encoders	PS	Active
	Capacitive Encoders	PS	Active
Heading sensors	Compass	EC	Passive
	Gyroscope	PC	Passive
	Inclinometer	EC	Active or Passive
Localization sensors	GPS	EC	Active
	RF Beacons	EC	Active
	Reflective beacons	EC	Active
Ranging sensors	Reflectivity sensor	EC	Active
	Sonar	EC	Active
	Laser range sensor	EC	Active
	Optical triangulation sensor	EC	Active
	Laser scanner sensor	EC	Active
Motion sensors	Doppler sensor	EC	Active
	Accelerometer	PC	Passive
Vision sensors	Cameras/Depth Cameras	EC	Active

TABLE 2.2: Sensor classification by typical use.

- Range - upper and lower limits - for example proximity sensor of medium quality can measure 10 to 80 cm;
- Resolution - minimum difference between two measurements;
- Dynamic range - ratio between the maximum and minimum measurable input according to the formula

$$dynamic\ range = 10 \log_{10}(\frac{maximum\ input}{minimum\ input})$$

- Linearity - how linear is the relationship between sensor output and input signals;

- Sensitivity - rate of change between output to input
- Accuracy - difference between output and the ground truth

$$accuracy = 1 - \frac{measured\ value - ground\ truth}{ground\ truth}$$

- Precision - how easy it is to reproduce sensor results

$$precision = \frac{range}{standard\ deviation}$$

2.2.3 Exteroceptive sensors

At least one exteroceptive sensor of a good *accuracy, resolution precision and sensitivity* is obligatory for **SLAM**. These sensors are equivalent of touch or eyes for the humans. They are tasked with measuring the external state of the mobile agent, thanks to them machine running **SLAM** solving algorithms can detect landmarks and update the estimated pose of the robot.

Laser range sensor - LIDAR

LIDAR - **L**ight **d**etection **a**nd **r**anging - is a device that allows creating the model basing on distance measurements. The device spins / moves in certain range (*scans*) and returns the values of distance at each point of resolution. It uses known time of flight - **ToF** - calculate distance between sensor and an object. At one point of time the device emits a very short pulse of light and samples it. Knowing that speed of light is roughly $c = 299\ 792\ 458\ m/s$ it can calculate the distance to the object at one time stamp (*as seen at Figure 2.9*).

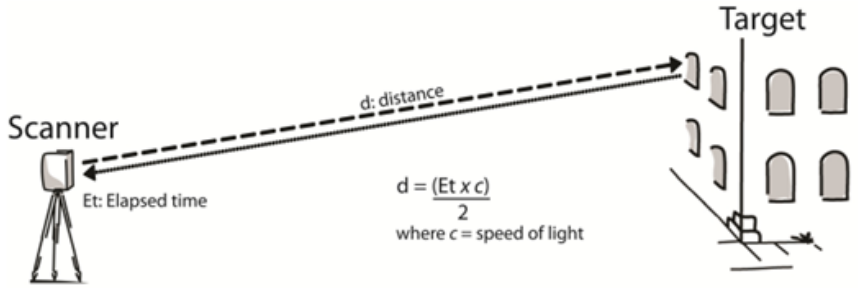


FIGURE 2.9: How the **LIDAR** works

By spinning fast **LIDAR** creates a set of measurements that can be correlated to the current sensor head position. Knowing sensor head position and distance there is sufficient data to create a cloud of points. The point cloud

later can be used to detect the landmarks in them and create a map (*see Figure 2.10*).

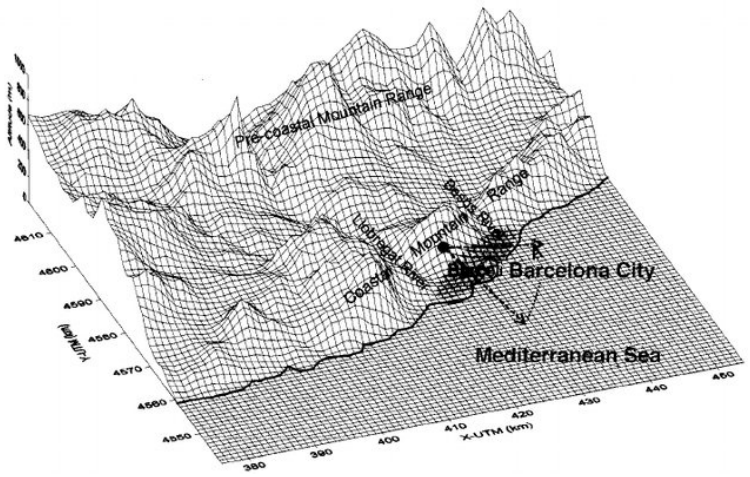


FIGURE 2.10: Point clouds maps produced by **LIDAR** sensors have many uses, for example mapping the terrain geography - this figure shows The Geographical Area of Barcelona Map created with the use of **LIDAR** unit in 1992. [12]

The obtained point clouds can be used to extract huge amount of features about the surrounding world. If the **LIDAR** used to survey the environment is of a good quality, meaning has satisfying sensor characteristics it can produce point clouds of quality good enough to get informations about shapes, possible passages and much more (*see Figure 2.11*).

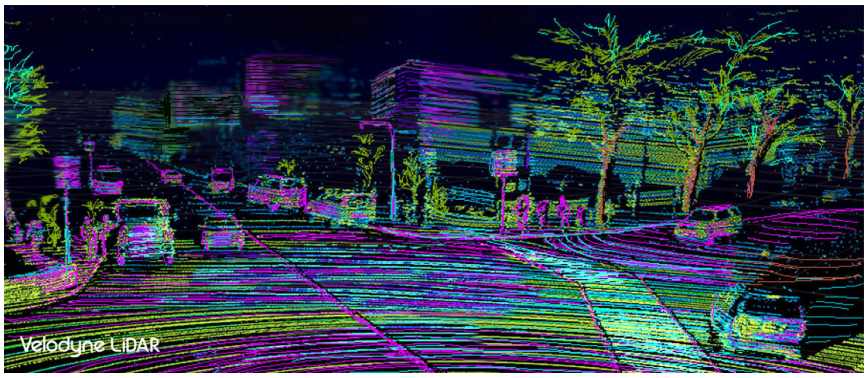


FIGURE 2.11: Pointcloud generated by High definition **LIDAR** system Velodyne Lidar created by Velodyne **LIDAR** Inc. [13]

LIDARs are one of the most often used sensors in **SLAM** solutions, and the laser scanner was also used in the subject of this thesis (*see Chapter 2.2.2*)

Sonar

SONAR - **So**und **na**avigation **r**anging is a device that works in a similar fashion to **LIDAR** but indead of using the electromagnetic light waves uses sound waves. Sonar sensors are commonly used because they are very cheap - the cheapest sonar unit is about 100 times cheaper than the cheapest lidar unit [14].

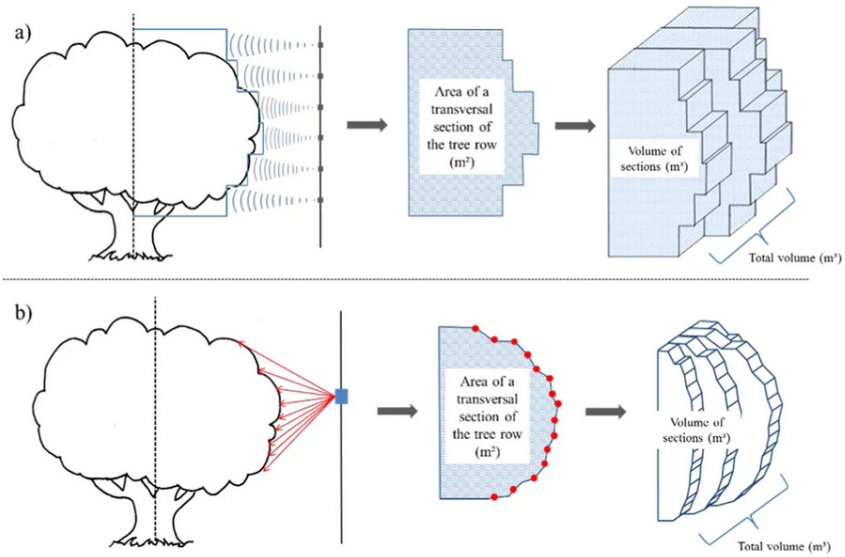


FIGURE 2.12: Different measures provided by **LIDAR** and ultrasonic **SONAR** [15]

Unfortunately the **LIDAR** has absolute superiority to **SONAR** sensors because of far greater resolution. The resolution gap is a product of different length of wave used for the survey of environment. The same surface being scanned by **LIDAR** and **SONAR** will be much more detailed in the **LIDAR** scans (as presented in *Figure 2.12*).

Cameras

Despite not using the camera in the subject of this paper cameras are worth mentioning when considering the state of the art **SLAM** solutions. There are two approaches to **SLAM** problem using cameras :

- **RGB-D** - Slam using special camera consisting usually of one or more cameras being standard RGB camera and the depth sensor such as structured light sensors. Usually the grid of infrared light is displayed and the warp in positions of the grid are calculated into depth image;
- **vSLAM** - Visual **SLAM** mostly using monocular camera.

The basic difference between these two approaches is that **RGB-D SLAM** has direct access to 3D information about structure of the environment. Stan-

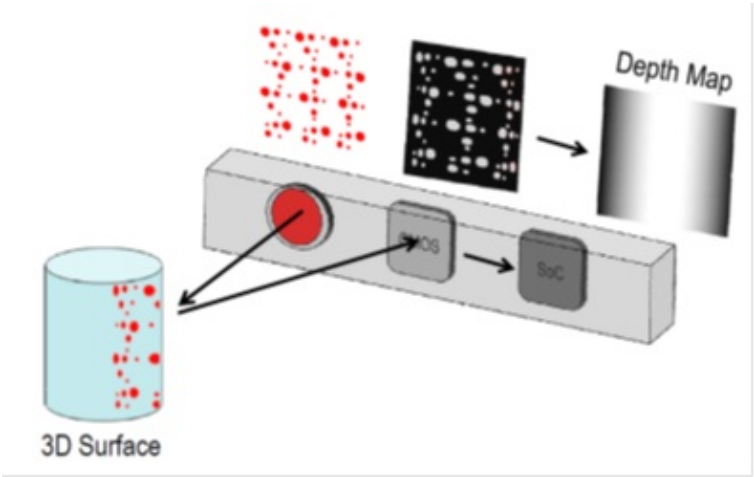


FIGURE 2.13: How the depth information is being extracted in RGB-D camera

dard **vSLAM** has to calculate the estimated depth of what it sees during the movement that is much more computationally complex. To run a **vSLAM** algorithm the following needs to be done:

- Initialize the process by observing the first frame and setting the parameters,
- Estimate camera motion
- Estimate depth
- Global optimization
- Relocation on the global map

In the case of **RGB-D** based methods the process is very similar to **vSLAM** case with the following changes:

- There is no need to estimate depth only using the camera frames, it can be calculated eaisly;
- The camera motion is much easier to identify thanks to three dimensional depth map created using grid displacement technique [16].

There are many advantages and disadvantages of **vSLAM** and **RGB-D** in closed buildings the **RGB-D** based methods are very good because of the instant depth information but it is limited to the distances from 1 to 5 metes [17]. The limitation is the product of difficulty detect the emitted grid pattern outside of closed scenarios.

2.2.4 Proprioceptive sensors

Proprioceptive sensors are not obligatory to implement basic **SLAM** but they are crucial for creating a reliable system. Measuring the internal values of mobile agent such as joint angles, position of wheels, battery levels or elevation level. In **SLAM** proprioceptive sensors are very important because the availability of good landmarks is not always guaranteed and the re-location of the mobile agent is not possible.

Rotary encoders

Rotary encoders is subtype of proprioceptive sensor that is extremely popular in automotive sector. Thanks to rotary encoders (*also known as shaft encoders*) the mobile agent can acquire information about the motor's current position. Aforementioned devices convert the motor's state to signal that is understandable by the controller - usually converting the physical parameter to a digital code. There are two types of rotary encoders :

- Incremental - reports only the position changes but cannot observe the absolute position of the motor;
- Absolute - always reports the absolute position of the motors, it always has one encoder value corresponding to one position of the motor.

The optical incremental encoder usually has a code shield overlay on motor that is being observed by light sensor. The encoder disk is divided to smaller rings with displaced holes that allow sensor to detect various signals when the motor is moving. During the movement sensors on each ring output a binary value creating a square wave in time. Using the pulses in the wave the encoders can convert the number of pulses per revolution to the angular velocity of the wheel (*Figure 2.14* explains visually how the incremental encoder works).

In absolute encoders big advantage over incremental is that every angular position, as big as resolution can be, of the encoder can be read at any time. The absolute encoder contain similar components as incremental but in addition they have stationary mask installed between the light sensor and encoder disk. Mask at each point has unique code, as the shaft rotates, the light pattern received by sensors is also unique and later is translated into a code resulting in own unique bit sequence (see *Figure 2.15*).

Thanks to this the velocity can be controlled in a loop using **PID** - **p**roportional **i**ntegral **d**erivative controllers. Thanks to this mechanism mobile agent can

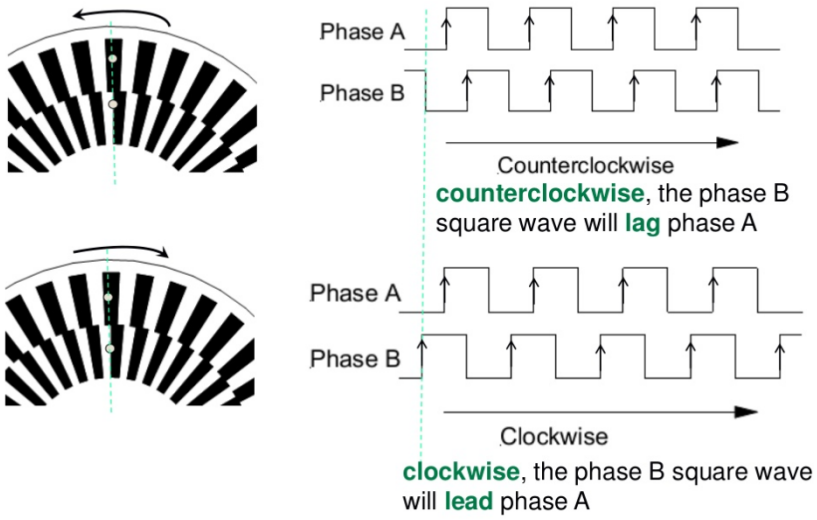


FIGURE 2.14: How incremental rotary encoder works [18]

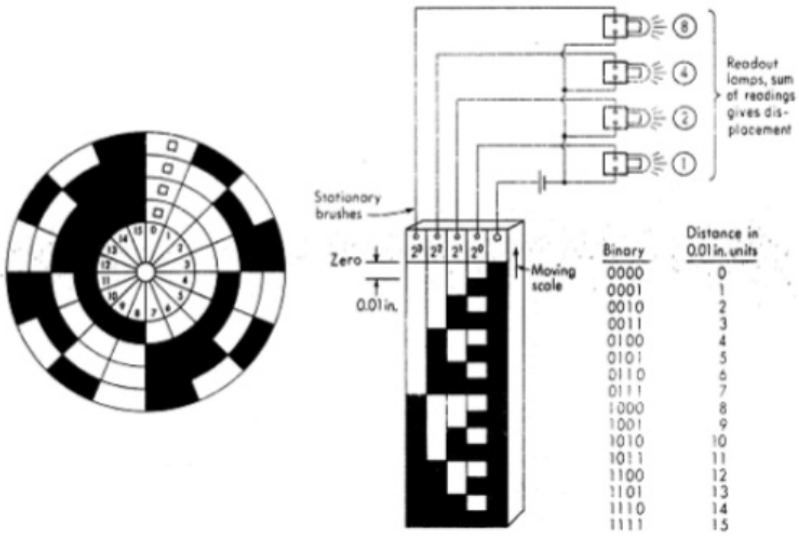


FIGURE 2.15: How absolute rotatory encoder works [18]

maintain the required velocity for example when friction between the wheels and the ground increase that is crucial for good working of almost any mobile agent (about that in *Chapter 4*). Getting the trustworthy position after longer period of navigation and estimating the pose using solely encoders is not realistic because of the *drift* problem (explained in *Chapter 2.3.1*)

Inertial Measurement Unit

IMUs track the displacement and orientation of mobile agent by comparing the series changes of acceleration and angular velocities to the starting pose. **IMU**s have wide variety of uses in everyday world such as:

- Ships (stabilization, navigation)

- Spaceships, Aircrafts (stabilization)
- Stabilized platforms, balancing platforms - *for example Segway HT*,
- Smartphones - (camera stablization, step counters, etc.)

The most important parts of an **IMU** are gyroscopes and accelerometers, either 1D, 2D or 3D. They can include other subsystems such as magnetometer but the two aforementioned are crucial for right working of the device. Tri-dimensional accelerometers measure linear acceleration in each axis (x,y,z) that are required by mobile agent to navigate (*Chapter 2.2.1*). The IMU when used on Earth and knowing it orientation can calculate roll and pitch values based on direction of natural gravitational force. The yaw parameter value cannot be estimated using only accelerometers because gravitational force is the point of reference. Vectors of roll and pitch are influenced by the gravitational force but yaw axis is perpendicular to the force of gravity hence making it impossible to get any information about it (as seen in *Figure 2.16*).

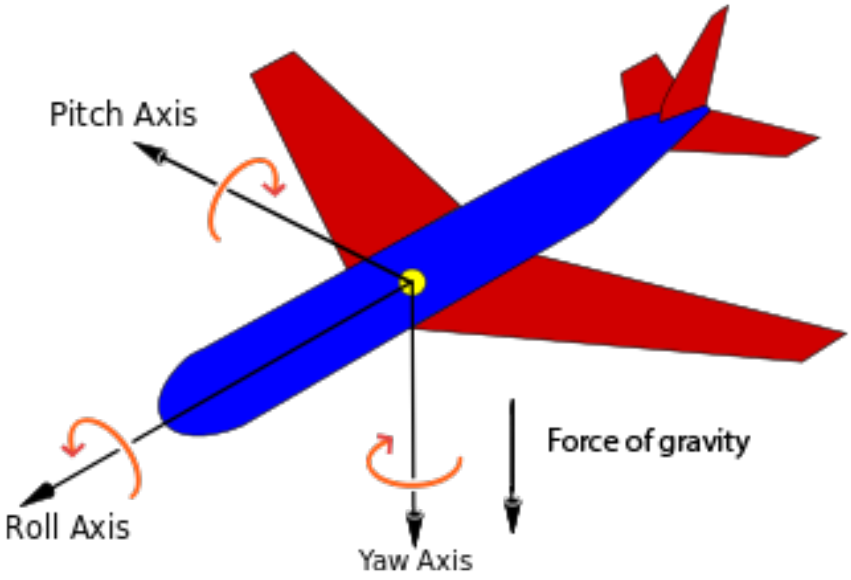


FIGURE 2.16: Yaw axis vector is perpendicular the gravitation vector

Gyroscopes measure the angular velocity of the mobile agent using the three axes. Observing the angular velocity over time can be used to calculate each angle change in pitch, roll and yaw. The readings are not as precise as readings of accelerometers due to how easily gyroscopes produce values with an error.

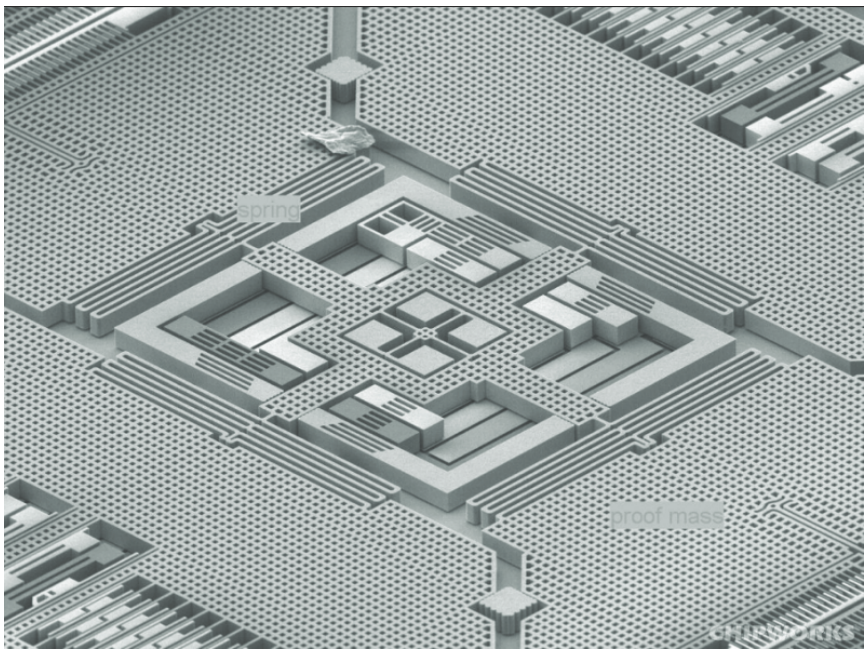


FIGURE 2.17: Microscopic structure of accelerometer

Nowadays the **IMU** systems are very small thanks to technology called **MEMS - Micro**electro**m**echanical systems. This is the technology combining the moving parts and electronics in devices at microscopic scales usually in silicon (*Figure 2.17*). Thanks to this the **IMU**s can be not bigger than few millimeters [19].

2.3 SLAM Algorithm

As mentioned previously **SLAM** two most important steps in solving the problem are :

- Predict, mobile agent localization and map state update using previous informations;
- Measurement update, getting new sensor data and matching it with predicted system state.

So far many **SLAM** methods were implemented using various prediction systems such as:

- Extended Kalman Filter and it's variations;
- Particle Filter methods;
- Expectation Maximization based methods.

The first case will be described more in-depth in following subsections. The others will not be discussed. Source of information about particle filter and expectation maximization methods can be found in [20]. Let's consider z observations of mobile agent from 1 to k:

$$o_{1:k} = o_1, o_2, \ldots, o_k$$

and c robot's controls from 1 to k:

$$c_{1:k} = c_1, o_c, \ldots, c_k$$

to complete the **SLAM** that is creating the entire mobile agent path p:

$$P_{1:k} = p_1, p_c, \ldots, p_k$$

using the map of landmarks:

$$m_{1:k} = m_1, m_c, \ldots, m_k$$

it is required to estimate the mapping between observations and landmarks hence correlating the detected landmarks with landmarks on the map and creating a path in time.

$$p(x_{0:t}, m | o_{1:t}, c_{1:t})$$

But the robot path and map are initially unknown and its location estimates are almost always erroneous basing solely on odometry (2.3.1) or observations [21]. Therefore it is important to provide good odometry, landmarks and data association method to determine absolute non-drifted mobile agent position.

2.3.1 Odometry

Odometry is mobile agent's position estimated by observing data from proprioceptive sensors (such as Wheel Encoders or **IMU**) by process called dead reckoning. Dead reckoning is the procedure of estimating own position using the last known location (e.g. initial position). Unfortunately this process has no way of correcting any errors. Small errors in measurements can result in huge displacement relatively to real position (see *Figure 2.18*). It is because the odometry errors are cumulative errors. Any **SLAM** system can exist without correcting odometry position using the landmarks.

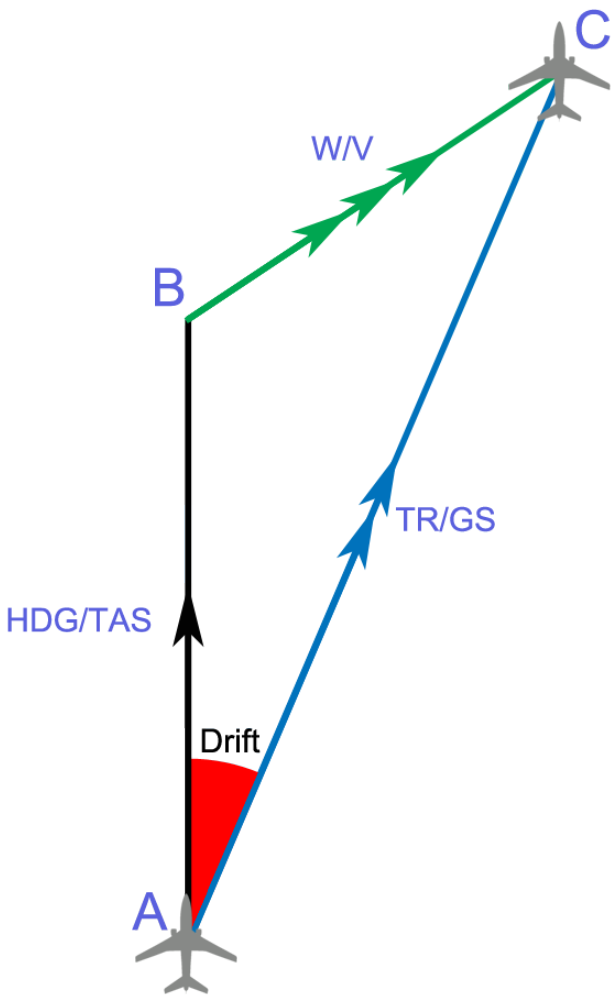


FIGURE 2.18: The odometry error (drift) after longer period of agent activity. A is last known position, B is the real position and C is the estimated position that is susceptible to culminative error.

The reason why the odometry is so important for **SLAM** systems is that sometimes there are not enough adequate landmarks observed to estimate robots' position. E.g. agent starts in initial location and observes the map with poor landmark diversity (see *Figure 2.19*):

Then, if only scans from exteroceptive sensor are provided when it moves to a new location, it cannot distinguish which pose is actually assuming because from its point of view, both observations are exactly the same (*Figure 2.20*), what can lead to large error.

The position is ambiguous unless the odometry data is used allowing the mobile agent to calculate which position it's actually occupying

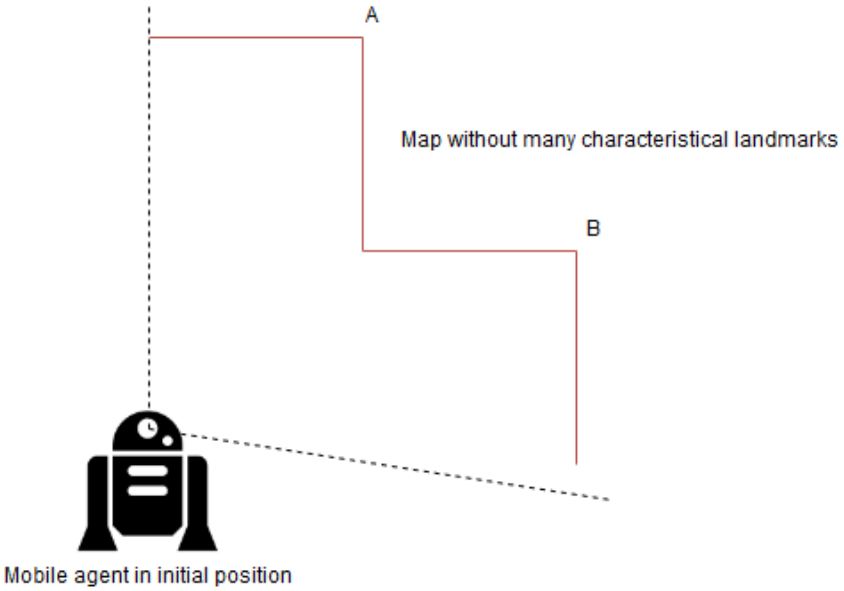
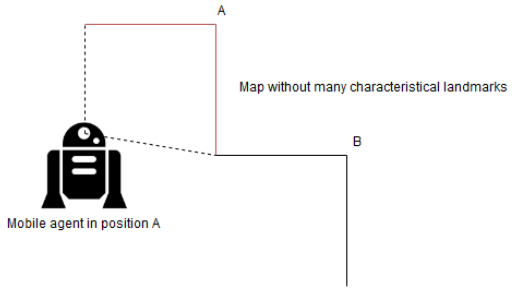
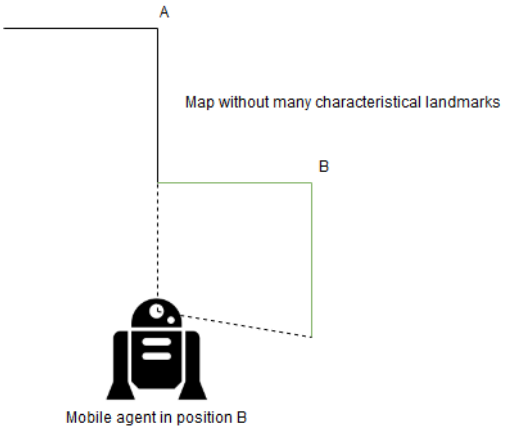


FIGURE 2.19: Mobile agent starts at the map with poor landmark diversity



(A) Robot in pose A



(B) Robot in pose B

FIGURE 2.20: Two position of robot at the map with poor landmarks

2.3.2 Landmarks and their extraction

Landmarks should be the features observed by the sensors that can be very easy distinguishable from rest of the environment (*Figure 2.21*). They will be used to future re-localization of the robot.

The nature of landmarks may vary very much on what type of exteroceptive sensor is used by mobile agent. Landmarks should be stationary objects because the agent's localization is calculated based on it's distance to the landmarks. The features of a good landmarks are:

- Stationary,

FIGURE 2.21: As an example - Sagrada Familia is an example of a good landmark for human classification, thanks to how unique is it can be easily used to estimate one's position

- Many unique in the environment,
- Distinguishable between each other,
- Easily re-observable.

There are many ways to extract landmarks depending on the type of sensor used. For example, landmarks observed by the monocular cameras can be edges of shapes, for them there are it many image processing techniques available. In this Thesis, only lamdkarks extracted with laser scanner methods will be mentioned because of the sensor used. Further references on visual landmarks can be fond in [22]. When speaking of laser scanner, the most used landmark extracting techniques are known are:

- Spike,
- **RANSAC** - **Ran**dom **Sa**mpling Consensus,
- Strategy mixes.

Spike extraction compares the received laser scanner values and looks for places where there are differences than exceed a certain threshold. That way the most significant changes are obtained and thus, is a good landmark extraction algorithm mostly in environments that have many irregularities and heavy edges.

RANSAC - this technique extracts the longer lines from the received values and later uses them and their composition as a landmark. The general spread is thresholded and the observed points belonging to same confined

space are being converted into a line landmark. It is a very good and reliable technique in indoor environments due to many regular wall segments in most situations.

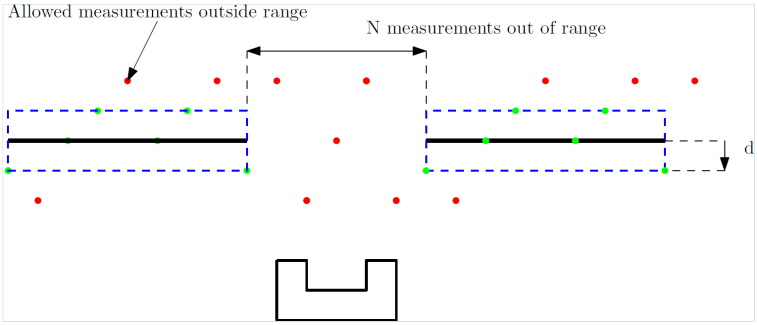


FIGURE 2.22: Landmark extraction with **RANSAC**

There are also mixed approaches that use both aforementioned techniques to detect walls but also a characteristic shapes that are not made of lines.

2.3.3 Associating data

Data association problem is the most crucial problem considering **SLAM**. If the landmarks cannot be identified there is no viable way to estimate mobile agent's position in an environment [23]. If the landmarks are not enough unique, hard to observe or there are not sufficiently many of them the situation presented in *Figure 2.20* happens. In practice the following problems happen always when trying to re-observe landmarks:

- Observe and save landmark but never re-observe it again rendering it useless;
- Associating one landmark with another, different one;
- Not re-observing any landmarks at any moment of time.

Most **SLAM** algorithms consider a landmark a good landmark only after it complies with certain criteria and is seen enough times. Once the landmark is observed, it is associated with other near landmarks using using nearest-neighbor approach. It is connected by calculating the euclidean distance to the neighbor. Later, the **EKF** algorithm can check if this landmark is in the database by checking if it is is inside certain distange range (with a given uncertainty) [24].

2.3.4 Extended Kalman Filter

EKF - **E**xtender **K**alman **F**ilter, also called **LQE** (**l**inear **q**uadratic **e**stimation) is a tool used to estimate mobile agent pose given the data from landmark observations and odometry. **EKF** can even be used for three dimensional space but in this paper two dimensional model will be explained because of the chosen sensor and robot operation dimensions.

EKF can be broke down into 3 steps:

- Estimating the mobile pose using the data from odometry;
- Updating the estimated pose using data association of landmakrs;
- Adding new landmarks if possible to new state.

The **EKF** requires few matrices to properly estimate the state:

- **X** - System state matrix - holds the mobile agent's coordinates and angular offset;
- **P** - Covariance matrix - holds the information how strong are some of the agent's position correlated and how strong correlation show some landmarks with the others. This one is a complex matrix and will not be discussed in depth in this paper (to read more consult [25]);
- **K** - Kalman gain matrix - it holds the information how accurate the position can be when observing selected landmarks. It is used to tell how much the position is corrected. This matrix settles what should be trusted more: odometry or re-observation;
- **H** - Mesaurement Jacobian matrix - it holds information how *range* and *bearing* of estimated landmark change as (x, y, θ) change it takes into account the noise (not discussed here [25]).
- **W, Q** - Procedural noise matrix - In **EKF** the noise of process and odometry is treated as Gaussian noise, this matrix holds the noise according to how well performs the odometry
- **R, V** - Measurement noise matrix - In **EKF** the noise of measurement devices is also considered to be imperfect and is described using Gaussian noise.

At the beginning, the map is initialized with it's origin being the robot's initial pose. At this moment, there is no uncertainty because it is the first

step. Then, the mobile agent moves and produces the new state with x being the x coordinate, y being the y coordinate and θ being the angle.:

$$(x, y, \theta) = (x_{t-1} + \frac{\partial x}{\partial t}, y_{t-1} + \frac{\partial y}{\partial t}, \theta_{t-1} + \frac{\partial \theta}{\partial t})$$

During this process the **Q** matrix is updated. At this moment agent's position is uncertain due to natural odometry error. Re-observation of some known landmarks is necessary to detect agent's state in map. Having the new position from estimate from previous formula:

$$(x, y, \theta)$$

and agent's new position it is possible to calculate landmark's expected position:

$$(\lambda_x, \lambda_y)$$

obtaining the estimated *range* to the landmark and it's *bearing*. Then we ob-

$$\begin{bmatrix} \text{range} \\ \text{bearing} \end{bmatrix} = \begin{bmatrix} \sqrt{(\lambda_x - x)^2 + (\lambda_y - y)^2} + v_r \\ \tan^{-1}\left(\frac{\lambda_y - y}{\lambda_x - x}\right) - \theta + v_\theta \end{bmatrix}$$

FIGURE 2.23: Calculating the estimated range and bearing of landmark

tain the landmark aquired and **H** matrix. Finally, the **K** matrix is calculated according to :

$$K = P * H^T * (H * P * H^T * + V * R * V^T)^{-1}$$

At this moment, the system knows the estimated position of the mobile agent and new state can be calculated using z and h as displacement of range and bearing.

$$X = X + K * (z - h)$$

Last step is adding new landmarks to the state X where mobile agent is now. New rows and columns need to be added to the **P** matrix and new landmarks correlated to it. Then, all steps need to be repeated successfully solving **SLAM** process.

2.3.5 Path and motion planning

Once the mobile agent knows the map of the place and its location, it needs to be able to plan a path to desired point without colliding with any obstacle. There is a difference between path and motion planning:

- Path planning takes care of creating the optimal, safe path;
- Motion planning uses the information about mobile agent's body and makes it follow the path according to own actuators.

Planning is being done at higher abstraction layer, the real world map (*Figure 2.24*) is converted to a workspace where the problem can be easily solved (*Figure 2.25*).

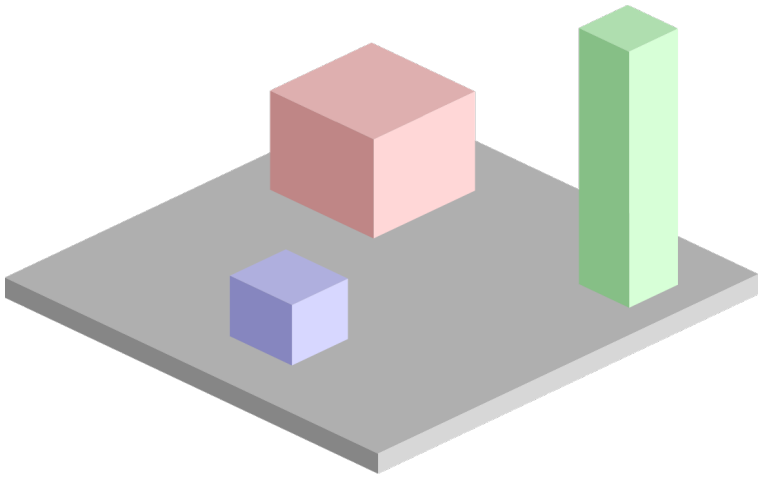


FIGURE 2.24: Real scenario before moving to higher abstraction layer

The aforementioned workspace is exactly what is created when the map is constructed using the **LIDAR** laser scanner.

The workspace is converted into a pixelmap where dark pixels represent the space occupied and white pixels represent the free space. The motion planning aspects make sure the mobile agent can pass every narrow space that the turns are not violent [26]. First, the optimal path is being calculated using one of the planning algorithms and then the trajectory is being optimized using the motion planner. To do so the map is being converted to a graph using each pixel (or group of pixels for sake of computational complexity) as a vertex (see *Figure 2.26*). The shortest path in graph is calculated using one of the popular algorithms such as **A*** or **Dijkstra** [26].

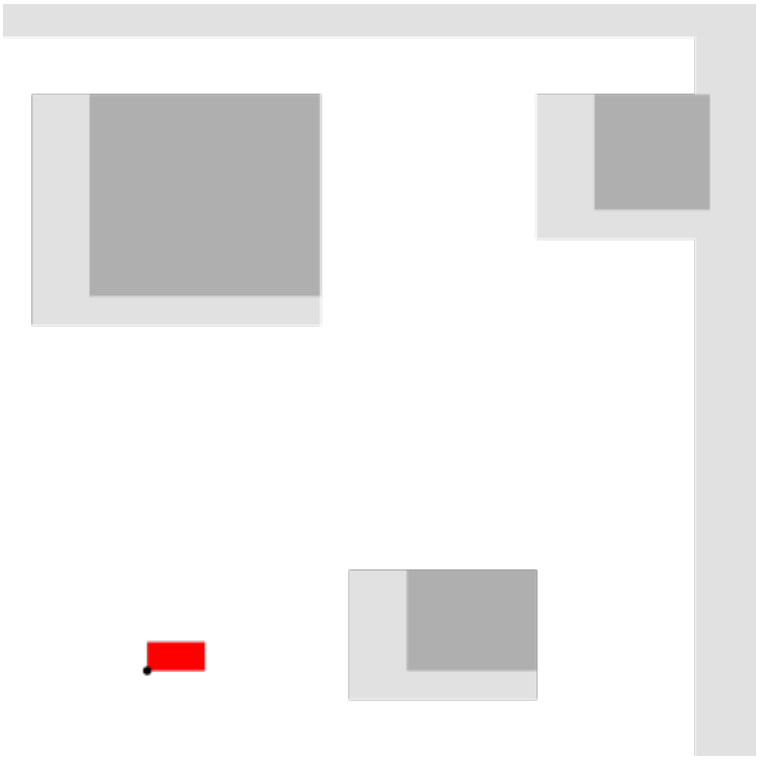


FIGURE 2.25: Real scenario before moving to a higher abstraction layer

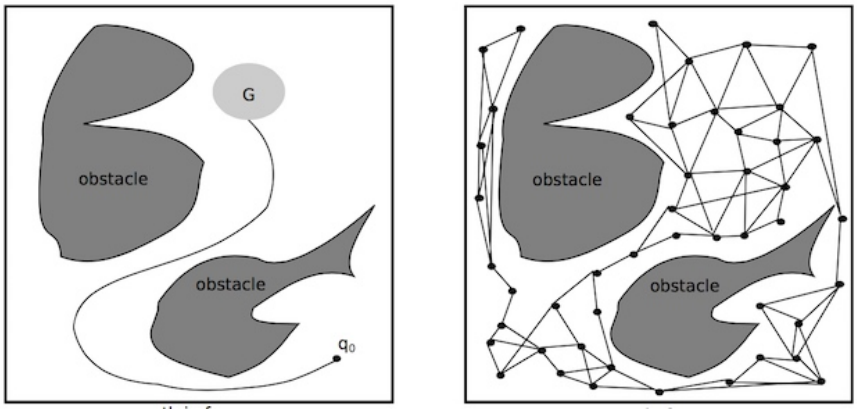


FIGURE 2.26: Motion planning algorithm output on the left and path planning on graph on the right. *Source: Duke Robotics*

2.4 Speech recognition

Speech recognition is an increasingly popular subject these days. Thanks to development of voice recognition models and **IoT** - **I**nternet **o**f **t**things most of modern smartphones are equipped with some extend of embedded voice recognition. The estimates say that more than 50% of all on-line searches will be done using voice recognition by the year 2020 [27]. The voice control makes interfacing with machine for humans more approachable and natural.

Difference	On-line voice recognition	Off-line voice recognition
Internet connection	Required	Not required
Word error	Very low	Medium-High
Training data access	Very high	Low
Reaction time	Depends on connection	Very low

TABLE 2.3: Advantages and disadvantages of on-line and off-line speech recognition

For some people, especially the people with **SCI** it may be one of few instant methods of communicate with the machine. There are many solutions to speech recognition problem but there is always a common process for all the algorithms (*Figure 2.27*):

- First, the voice is captured using appropriate device such as microphone;
- Then the recorded audio is pre-processed removing noise, splitting audio into characteristic for human speech smaller chunks (words) and extracting the most important features, it is done using **DSP** algorithms that will not be discussed in this paper, more about them here [28];
- Then. acoustic models are applied to identify spoken words in said language with dictionary. Today these models are mostly **LSTM** deep neural networks [28];
- Identified words are compared with language references and the output text is generated.

Voice recognition can either work on-line or off-line. The on-line models send the pre-processed audio to the on-line service that performs the rest of the steps and returns the recognized text. The off-line models have to manage all the calculations and recognition by themselves. Thus, the system cannot handle complex tasks. *Table 2.3* shows the advantages and disadvantages of both approaches.

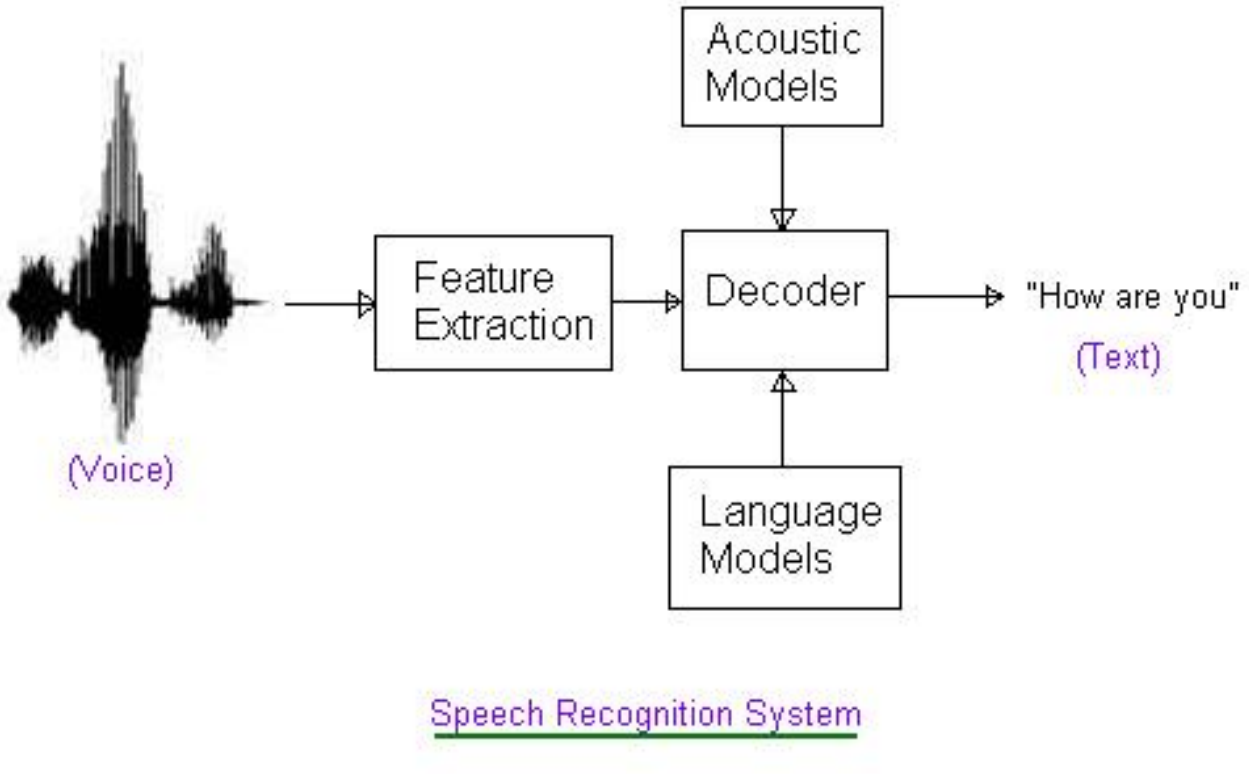


FIGURE 2.27: Voice recognition system layout *Source: http://www.rfwireless-world.com/Terminology/automatic-speech-recognition-system.html*

Chapter 3

Selected tools and methodology

This chapter will discuss all the necessary tools and reason for choosing them. If necessary the comparison to other choices will be presented. Implementing the entire solution from the scratch would be very redundant and according to one of the most archetypes of human ingenuity, useless. According to the good programming practices [29] and metaphor *don't reinvent the wheel* many subsystems of the solution are using third-party libraries as long as software license marks them as *open source*.

3.1 Software

This section will contain the description of the autonomous platform. Each subsection contains the crucial components for the proper function of developed system. It contains:

- Environment that handles basic low-level system operations and set of drivers that will be able to fetch data from sensors *Subsection 3.1.1*;
- Set of libraries and open source software required to perform **SLAM** process - *Subsection 3.1.2*;
- Software written to connect all the aforementioned libraries into working system - *Subsection 3.1.3*.

3.1.1 Mobile agents operating system

The selected operating system is *Linux* system under *Ubuntu 18.04.2 LTS* distribution. The *Linux* operating system is one of the least popular systems for everyday use (desktop computer) (see *Figure 3.1*) it is the most popular solution in supercomputers, embedded solutions and automotive [30].

Linux systems are superior in the means of reliability, security, freedom and cost compared to other operating systems on the market [30]. The main

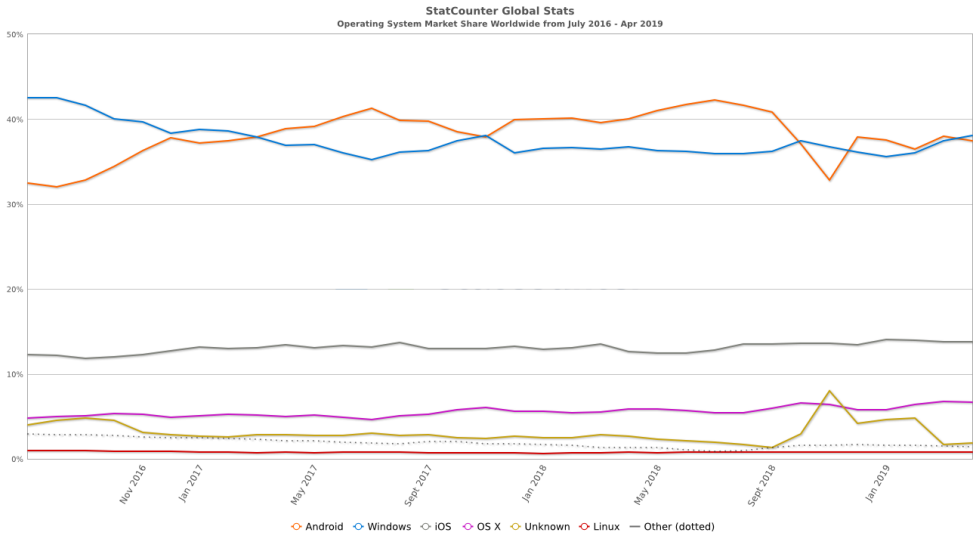


FIGURE 3.1: Dekstop operating system market share [31]

reason of choosing the *Ubuntu* is the distribution's commonness and active community. Linux has the tools not available on the other operating systems such as tools allowing mounting the serial interfaces as network sockets. Another extremely important reason for choosing *Linux* is it's integration with **ROS** (*Subsection 3.1.2*). The latest and most patched distributions of **ROS** are available on this platform and it is developed mostly int this environment. *Ubuntu* also offers a very good and intuitive graphical interface and console editors with syntax highlight.

3.1.2 ROS - Robot operating system

Despite its name, **ROS** not an operating system. In reality, it is a package of open-source community-driven tools, libraries and protocols designed to standardize and make robotic solutions more approachable. **ROS** is licensed with *Berkeley Software Distribution Licenses* making it open-source and available for anyone. The main advantage of **ROS** is that is developed by community (not a closed team) and it keeps the interoperability and protocols between hardware and software. **ROS** takes advantage of possibilities of *Linux*-based systems and creates a infrastructure of **Nodes** communicating via sockets in peer-to-peer manner. The infrastructure of **ROS** presents itself as following *Table 3.1* presents.

Each used subsystem is contained in a **package**, when package is launched by **ROS** Master Node initializes the topics and nodes contained in it and oversees them according to the *Figure 3.2*. The chosen distribution of **ROS** Melodic Morenia due to it's support of x64 Linux system and 2018 features.

Part of the system	Role
Master node	This node manages naming and registration of other nodes. It is responsible of enabling and tracking communication between other nodes. Master node also stores and inputs parameters (if any are given) to nodes later initialized.
Nodes	Nodes are the atomic processes that are responsible of solving one (or more) of the tasks in the system such as converting raw data from sensors for another node to analyze and create map from. The nodes publish or listen to Topics that allow them to communicate direcly with another nodes.
Messages	Messages is the data sent between nodes, usually the message follows a certain protocol. It is considered to be a data structure consisting fields of another types (such as integers, floating point values, arrays of the aforementioned primitive types)
Topics	Topics are the main transport route in **ROS**. They funcion as tunnels for messages published by nodes. Topics can be subscribed to or published. One node may subscribe to or publish many topics depending on how many information it needs to share / receive.
Service	Services work in similar fashion to topics but instead of simply publishing or subscribing them, they can be activated on demand. One node needs to send a request for any given information before it is being published by a certain service.

TABLE 3.1: Elements building **ROS**

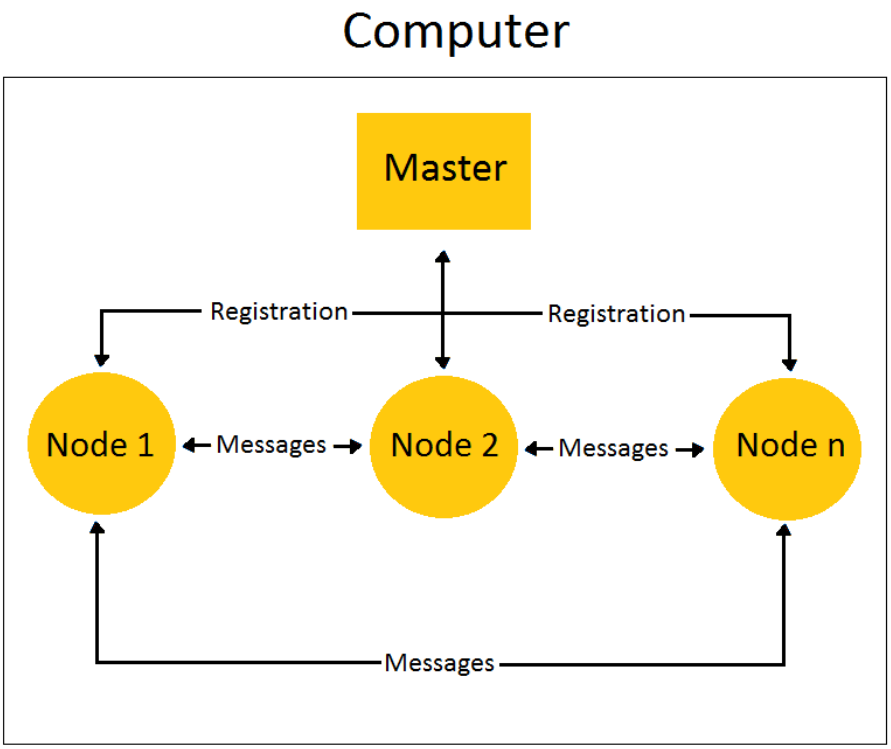


FIGURE 3.2: Structure of the **ROS** based system, *Source: docs.erlerobotics.com*

3.1.3 Programming languages and tools

The programming languages used are mostly **Python** and **C++** because of the variety of usages in **ROS**. **C++** is programming language appropriate for creating drivers for sensors. C++ is good for working on low levels of abstraction and lightweight making the created binaries light. C++ was also chosen because of it's legacy and huge availability of various libraries for creating drivers. Python on the other hand is scripting language and thanks to no need to compile the code every time it is used, it is perfect for rapid prototyping of some logic of the project. The basic **ROS** library set is available for both languages. Another used programming-related tool is **XML** - **E**xtensible **M**arkup **L**anguage. It is used in **ROS** environment to set the general parameters and create *launch* files for the nodes.

3.2 Hardware

The physical aspect of the mobile agent is what will define many parameters in the implementation. Due to nature of the project - that is creating the autonomous solution for the **Control-Live** platform it needs to satisfy some

```
<?xml version="1.0"?>
<launch>
<node name="rplidarNode" pkg="rplidar_ros" type="rplidarNode" output="screen
    ">
<param name="serial_port" type="string" value="/dev/ttyUSB3"/>
<param name="serial_baudrate" type="int" value="115200"/>
<param name="frame_id" type="string" value="laser"/>
<param name="inverted" type="bool" value="true"/>
<param name="angle_compensate" type="bool" value="true"/>
</node>
```

FIGURE 3.3: Example code for node launcher in **ROS**

pre-selected requirements. The choice of sensors was strongly influenced by target environment (indoor). The hardware of the mobile agent remains open to new extensions and sensors aside from already connected.

3.2.1 Mobile agent

The mobile agent prototype should resemble the final solution created by **Control-Live**. That's why the base frame of the agent is exactly the same frame as the one used for the electronic wheelchair. The agent is being moved by two motors that propel it and 4 additional wheels used for stabilization of the frame (*Figure 3.4*).

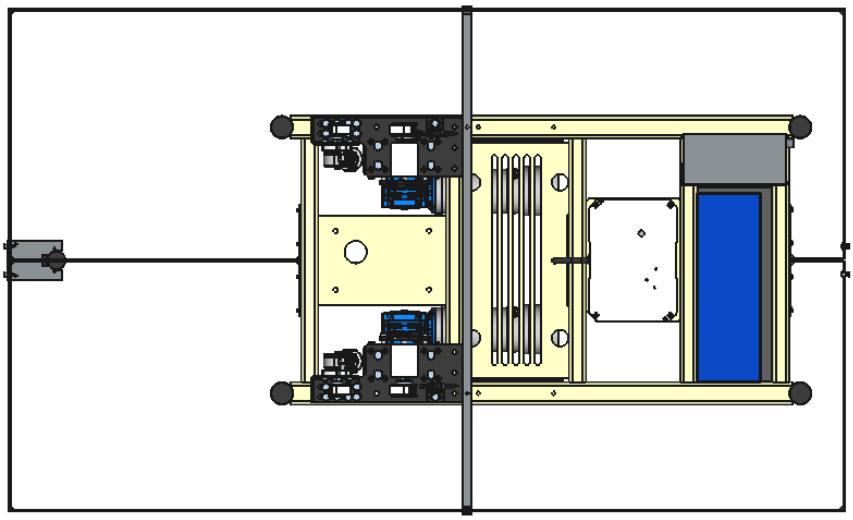


FIGURE 3.4: Top-view of mobile agent schematic

The prototype was suited with metal frame that corresponds to the size of the upper part of the wheelchair - it is installed to create fine borders for prototype to navigate (*Figure 3.5*).

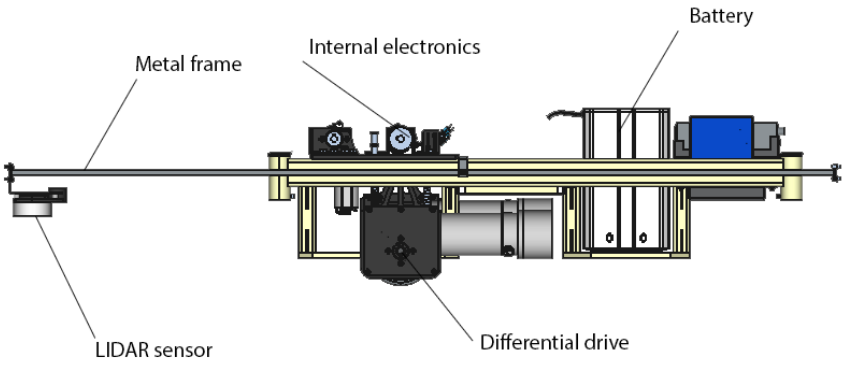


FIGURE 3.5: Side-view of mobile agent schematic

3.2.2 Exteroceptive sensors

At this moment the prototype is using only one exteroceptive sensor - **LIDAR**. Because of the closed and regular nature of environment the 2D **LIDAR** was estimated to be the best solution. The **LIDAR** is mounted at the front side of the prototype and covers 270 degree range of scans. It is enough to detect many characteristic landmarks. Unfortunately the average price of laser range scanners is quite high. Due to it's satisfactory quality to price ratio of **RPLIDAR A1M8 360 degree laser scanner** (see *Figure 3.6*) it was chosen to be the main exteroceptive sensor for the prototype.

FIGURE 3.6: RPLIDAR A1M8, Main exteroceptive sensor of the prototype

Sensor	Detection Range	Sample Rate	Coverage	Price
Sweep V1	1-40 meters	2 KHz	360 degrees	350$
RPLidar A1M8	**0.4-12 meters**	**2 KHz**	**360 degrees**	**112$**
HCSR04 Ultrasonic	0.6-4 meters	10 KHz	15 degrees	4$
Hokuyo URG-04LX	0.2-56 meters	28 KHz	360 degrees	1900$
Hokuyo UTM-30LX	0.2-30 meters	25 KHz	250 degrees	4500$

TABLE 3.2: Comparison of potential extreroceptive sensors

The RPLIDAR A1M8 after initial testing was producing the satisfying and clear scans (*Figure 3.7*) of the environment hence it was selected as main exteroceptive sensor.

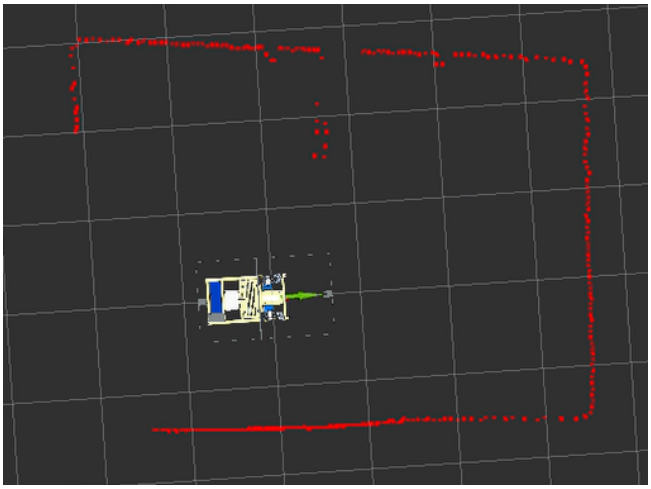


FIGURE 3.7: Scan using RPLIDAR A1M8

3.2.3 Proprioceptive sensors

To acquire the odometry the initial plan assumed using only wheel encoders to estimate the displacement of the mobile agent. Unfortunately the *odometry drift* (see *Chapter 2.3.1*) was too big for the SLAM algorithm to work properly - see *Figure 3.8*.

The problem was solved by adding the additional proprioceptive sensor - inertial navigation unit. With the data fusion from both sensors the odometry error was minimal and allowed the good working of **SLAM** algorithm.

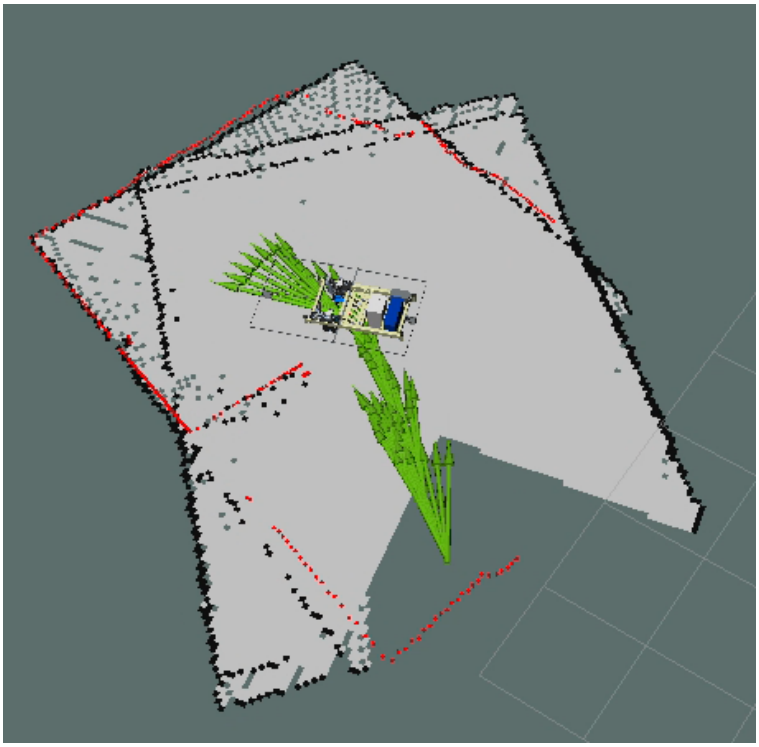


FIGURE 3.8: Mobile agent's estimation error using only laser scanner and wheel encoders, the odometry (green arrows) error was too big to precisely estimate agent's state, therefore detected landmarks could not be associated with the remembered ones and were added as a new set of landmarks generating the displacement in the map.)

Wheel Encoders

To move the prototype, two **TGM3 24VDC** motor units were used. Each equipped with incremental magnetic encoders providing the wheel speed (in pulses per second) as shown in *Figure 3.9*.

The encoders use hall effect [32] to generate alternating pulses. Later the calculated pulses per second are sent over the internal prototype network to the control unit that runs **ROS** software.

Inertial Navigation Unit

The selected **IMU** for the prototype is **SparkFun MPU-9250 9DoF** (*Figure 3.10*). This unit was chosen because of it can be easily reprogrammed and the sensors can be re-calibrated thus vastly improving it's performance.

Another important reason for choosing **MPU-9250** is that its price to quality ratio. Because of it's programmability **MPU-9250** can be modified to fuse the output of it's sensors internally using Direction Cosine Matrix algorithm. On-board chip can use it to become **AHRS** system - **A**ttitude **H**eading

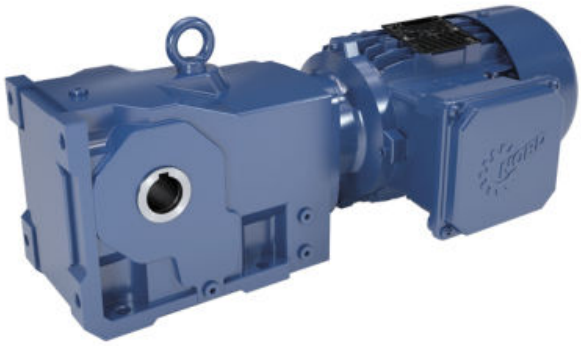


FIGURE 3.9: TGM3 Motors with built-in encoders used for calculating the odometry

FIGURE 3.10: SparkFun MPU-9250 9DoF sensor used in prototype

Reporting **S**ystem allowing the mobile agent to correct it's odometry taking into the account the heading of **IMU** [33].

3.2.4 Speech recognition

The voice control development kit selected is **vicCONTROL DSP3** (*Figure 3.11* developed by *voice INTER connect*. The choice of speech recognition software was pre-made by **Control-Live** and the mobile agent needed to be adapted to it. The voice recognition unit contains it's own speech recognition model, algorithms and the microphone. **vicCONTROL DSP3** navigates the speech recognition patterns using dialogs - allowing user to design own communication protocols. It also supports the voice beam-forming using microphone arrays and voice-over-ip emergency call system [34].

FIGURE 3.11: vicCONTROL DSP3

The device is programmed using own language, when voice recognition is done the appropriate detected phrase and state is sent via serial connection to the main control unit) that later.

Chapter 4

Implementation

This chapter will introduce the system implementation, it's structure and interactions between subcomponents.

4.1 System overview

The entire system contains itself in three distinct environments and communicates with subcomponents using three different protocols:

- **ROS** environment - home to all nodes required to perform **SLAM** logic;
- Serial devices - almost all serial-connected peripheral devices such as sensors or speech recognition board;
- **CAN** devices - entire hardware of mobile agent, from main motors to motors powering wheelchair configuration changes.

Both **CAN** and **ROS** environment have internal communications whereas serial devices are not exchanging information between subcomponents.

4.1.1 ROS environment

ROS environment is where the brain of mobile agent. Peer-to-peer node system that communicate with each other while being supervised by master node. The protocol used in agent implementation is **TCPROS** [35].

The *master* node works as lookup-name service for the nodes other subscribe to it. Then they can communicate with other nodes directly using the TCP layer sockets (see *Figure* 4.2). The TCP layer transmission is being handled directly by *Linux* operating system.

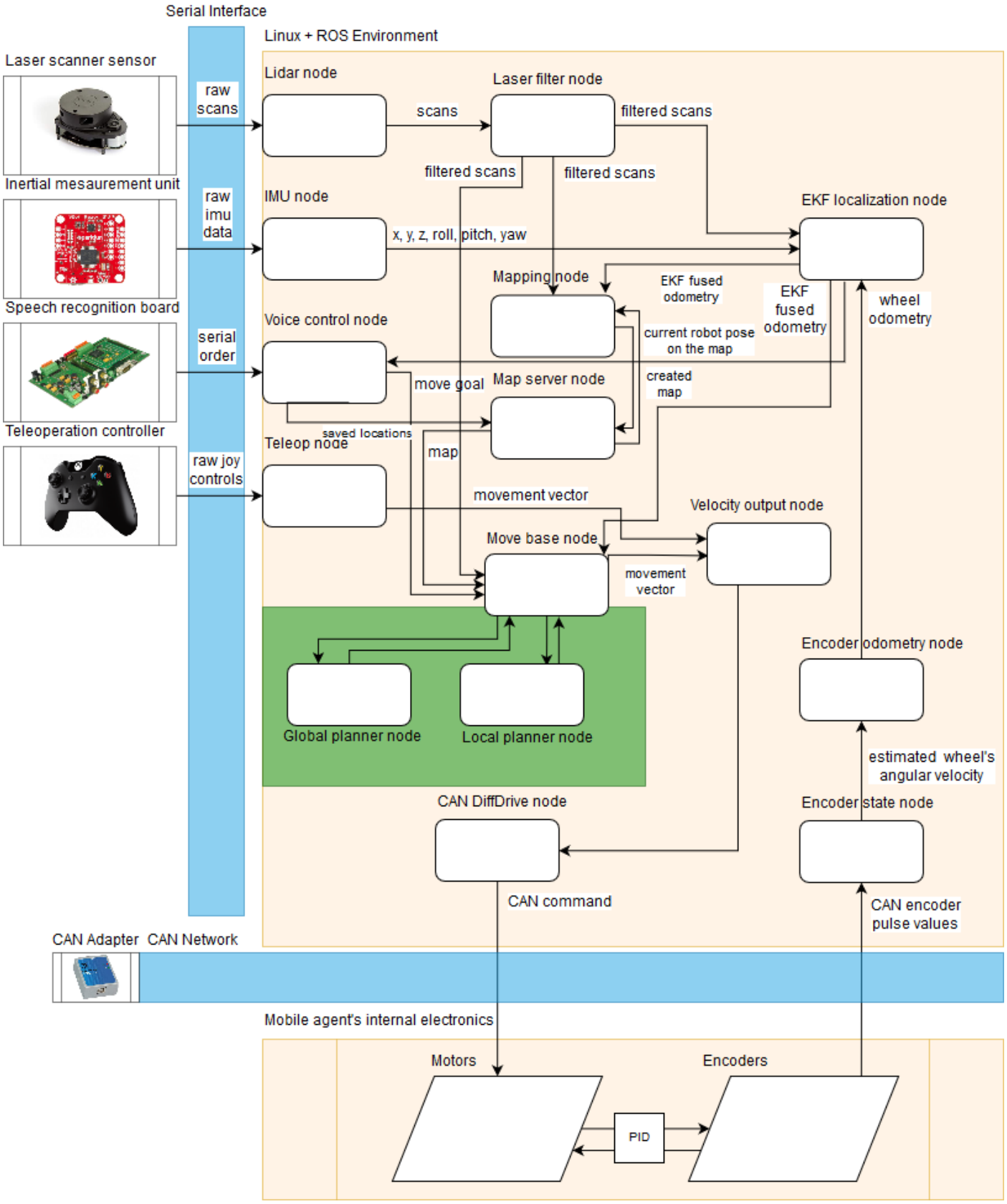


FIGURE 4.1: Diagram of the mobile agent's system and internal interaction

4.1.2 Serial devices

The communication with sensors is mostly one-sided - **ROS** environment receives information from the sensors and boards. Because of the little need for

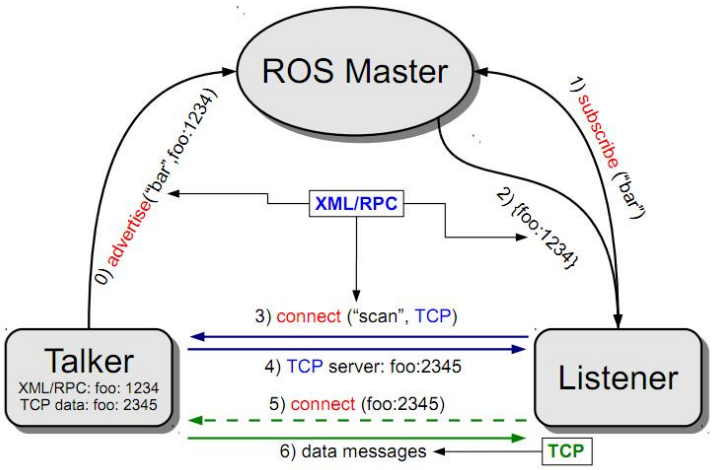


FIGURE 4.2: TCP ROS communication example

parallelization of data streams and high available clock rate serial protocols are perfect for receiving information from sensors. The serial protocol is a typical embedded systems solution - **SPI** - **S**erial **P**eripheral **I**nterface. It was selected for various reasons :

- No transceivers needed - that allows for easy integration of new sensors and scalability;
- High output data rate compared to other serial protocols - good for sensors with high refresh rate;
- Only four pins usage - allows more possibles and scalability when prototyping.

The **SPI** configuration in mobile agent's implementation is as presented in *Table 4.1*.

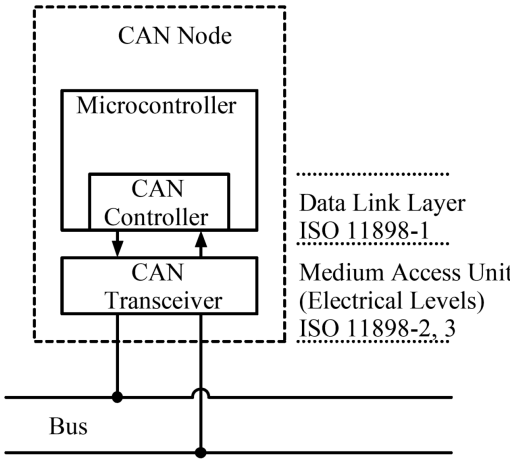
Serial line parameter	Value
Baud-rate	115200 bps
Data bits	8 b
Stop bits	1 b
Parity	None
Flow control	XON / XOFF

TABLE 4.1: Serial configuration of the implementation

4.1.3 CAN network

CAN - **C**ontroller **A**rea **N**etwork is the subtype of serial bus that hold most of internal mobile agent electronics communication. **CAN** is in internal electronics due to it's many safety countermeasures. To communicate with the **CAN** bus of the mobile agent the system requires to use special device that will modulate the given data to **CAN** standards (see *Figures 4.3a and 4.3b*).

(A) Selected CAN Node - VSCOM CAN to USB converter



(B) General CAN Node layout

FIGURE 4.3: CAN communication device

The standards used are *ISO 11898-1* data link layer standard and *ISO 11898-2,3* medium access unit standards. There is only one **CAN** node in the system because all required components are connected to the same bus.s The system uses two type of the messages:

- Velocity messages issuing the velocity commands to the mobile agent's motors;
- Encoder messages containing the information about the current motor turning speed.

The message layout will be described in *Section 4.2.6*.

4.2 Nodes overview

Each node has it's own unique task in the **ROS** system and communicates with selected nodes. There are three types of nodes in the system:

- **SLAM** node - nodes responsible for solving task connected to slam - implementing **EKF**, moving the agent, detecting landmarks;
- Sensor nodes - nodes responsible for obtaining the data from sensors and converting it to standard usable by the system;
- Utility nodes - nodes used for development (not required for **SLAM**) and very useful for prototyping.

4.2.1 Laser and laser filter nodes

Laser node and laser filter node together create a series chain (see *Figure 4.1*). The laser scanner node connects to the **SPI** socket in *Linux* system obtaining raw scan values:

$$scan = (d_1, d_2, \ldots, d_n),\ n = scanner\ resolution$$

and converting them to the **ROS** *laser scan message* standard [36]. That means the observed data will be stored in array of primitive *float* type and fraught with timestamps and then put into message that contains basic info about the laser scanner.

$$LaserMessage = (header, scan, timestamp)$$

This message enters the filter node that will be scanning the range and intensities arrays and remove all the messages that are in the range of the space that mobile agent's body occupy. That way, the **ROS** obtains obscured laser information (*Figure 4.4*).

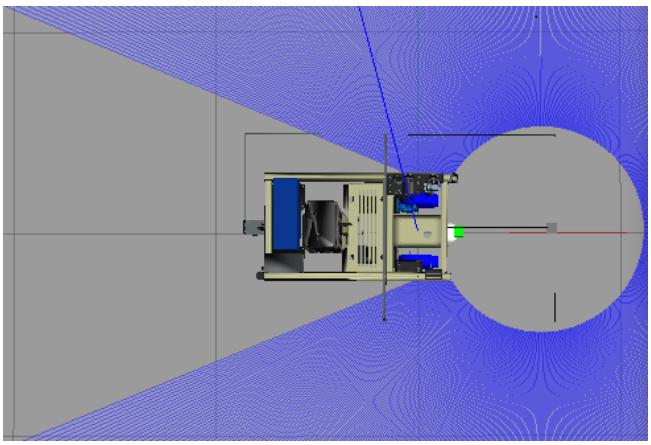


FIGURE 4.4: Visualization of edited laser scan data

4.2.2 IMU node

The **razor 9dof IMU** was reprogrammed to function as attitude and heading reference system. The sensor implements its own kalman filter and calculates the angular and linear velocities. Then it returns the values to the system as set of three matrices:

$$IMU\ message = (covariance, angular\ velocity, linear\ velocity)$$

The additional covariance matrix is calculated by the **IMU** and measures the reliability of the generated data. The node connects to the **SPI** socket associated with the **IMU** and converts the received messages to **ROS** *IMU message* standard [37] that holds the same fields as raw message transferred through serial interface.

4.2.3 Voice control node

The voice control node is connected on the **SPI** socket associated with the **vicCONTROL DSP3**. The board was reprogrammed to follow speech dialog controlling the mobile platform's configuration and autonomous navigation goals (see *Figure 4.5*).

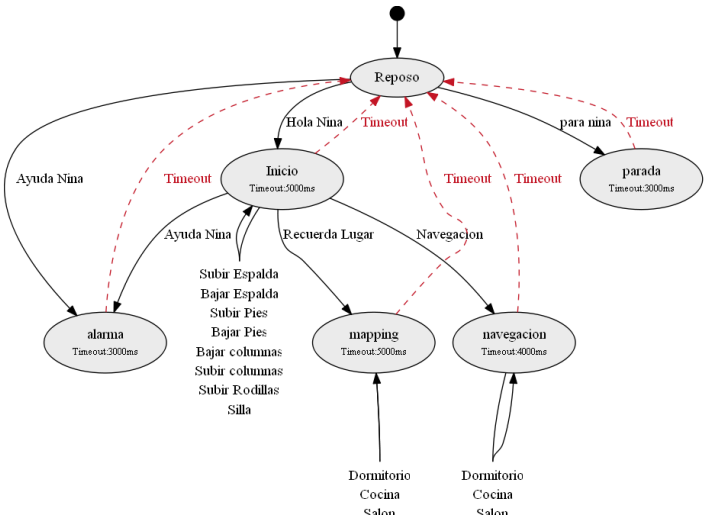


FIGURE 4.5: Dialog flow programmed into voice control board

When talking, user utilizes board's natural language processing software to operate on the graph. Depending on its state, the board sends the appropriate message via serial to voice control node. Then the voice control node

recognizes the given command and depending on its nature saves the current position or gives the new goal to move to. The node provides the set of saved way points to map and set of goals to move base.

4.2.4 Teleoperation node

Teleoperation node is used for taking the manual control of the mobile agent. Teleoperation node is connected via serial to the external controller. It accepts various input devices such as joystick controller, mouse or keyboard. First the input data such as:

$$joystick = (axisx, axisy, buttons)$$

is taken from the **SPI** socket associated with the controller and is calculated into linear ang angular velocities vectors.

$$message = (linear, angular)$$

4.2.5 Velocity output node

This node is responsible of converting the values sent by nodes described in 4.2.4 and 4.2.6 into the **ROS** *geometry_msgs* message. After collecting the commands from two (or more) topics it forwards the command to the node responsible for translating the messages to the **CAN** (4.2.7). The velocity node is also responsible for multiplexing the input commands. The manual controls have always priority over the move base node commands.

4.2.6 Move base node

Move base node is the core node responsible for robot movement. It is a stack of many state-of-the-art solutions for:

- Generating optimal path,
- Generating motion plan,
- Avoiding obstacles,
- Performing recovery behaviors.

The move base node subscribes to following topics:

- Map, basic higher abstraction level environment where the navigation problems will be solved. The map is provided by *map server node*;
- Goal, the point that will be the final destination of the generated path, it is provided by *voice control node*;
- Scan messages, used for dynamic obstacle handling and creating local costmaps;
- Odometry - the estimated mobile agent location calculated by the **EKF** fusion of odometry of various sensors.

And published on the following topics:

- Velocity output - the wanted mobile agent's velocity at given time moment;
- Feedback - state of the navigation at given time moment - idle, moving etc.;

Provided odometry and laser data and goal the navigation can start working on creating the path. The map provided by *map server node* is converted into costmaps. Costmaps are the two dimensional occupancy grids. Based on them (and various parameters) the move base node can create a motion plan for the mobile agent. There are two types of costmaps used in the node:

- Global costmap - costmap created based on the map provided by the map server node;
- Local costamp - overlay of the part of global costmap in radius around the mobile agent and dynamic detections done by laser scanner (obstacles observed by the agent);

Later, the assisting global planner node generates the path taking into the account the mobile agent's shape [38]. In my implementation the agent's shape is approximated by two circles at the front and back side of the agent. The representation in the **ROS** allows performing maneuvers such as U-Turns. The algorithm used to generate the path is the heuristic **A-star**. The map pixels are used as the set of vertexes with edges being the connections between groups of white (unoccupied) pixels. Later, the generated global path is locally optimized for robot motion by the local planner that takes into the account the dynamic occuring obstacles (see *Figure 4.6*). As the mobile agent odometry estimation proceeds through the map, the move base issues different velocity commands that will be subscribed to by *differential drive node 4.2.7*.

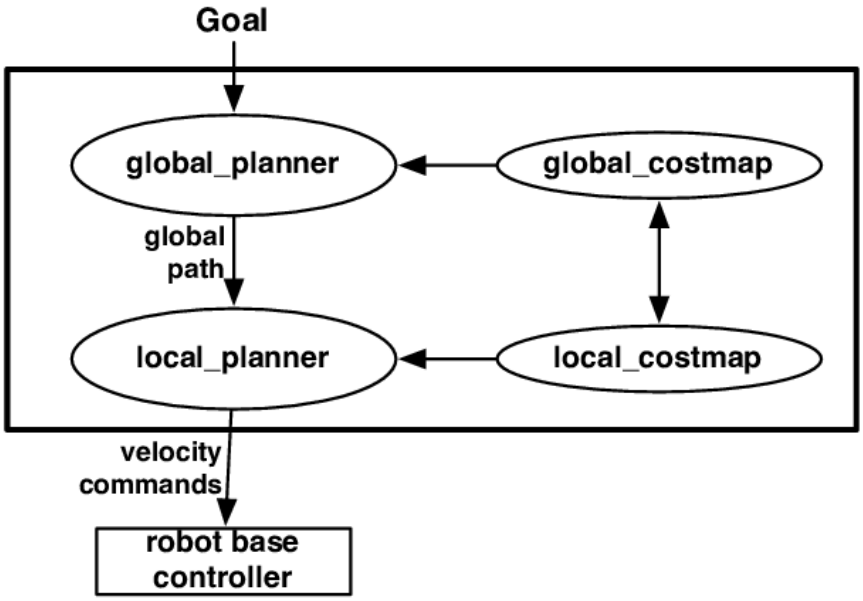


FIGURE 4.6: Move base workflow [38]

4.2.7 Differential drive node, encoder state node, wheel odometry node

The differential drive node and encoder state node are directly interacting with the **CAN** network. They are associated with socket forwarding **CAN** data. Socket is created using open source drivers for *Linux* system, allowing to set up the connected *CAN controller device* to be seen as network device (see *Figure 4.7*).

Differential drive node subscribes to velocity output node hence receiving the velocity commands from *move base node* and *teleop node*. . Differential drive node converts the message received by velocity node:

$$message = (linear, angular)$$

$$message = (V, \omega)$$

into separate speed commands for both of mobile agent's motors (v_{right}, v_{left}) using following logic. To do so it also needs to know the L diameter of the wheel:

$$V_{right} = \frac{2V + \omega L}{2}$$

$$V_{left} = \frac{2 * V - \omega L}{2}$$

Then calculated values coded into 16 bits and sent via the **CAN** controller to the mobile agent's internal electronics.

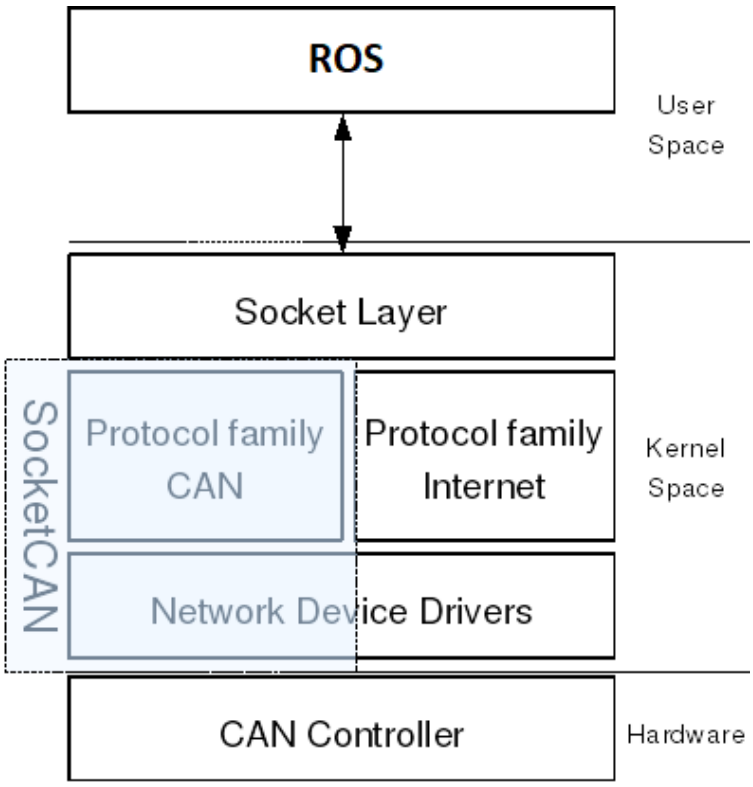


FIGURE 4.7: SocketCAN communication layer

4.2.8 Mapping and map server nodes

Mapping utilizes *Hector mapping* open source package. It allows constructing the map provided the laser scanner with high refresh rate. The mapping node is capable of providing 2D robot state estimates on its own but it is not satisfactory for the precision requirements of the mobile agent's system so it uses odometry provided by **EKF** node. The node uses software written by **Team Hector** - the winners of many automotive-focused contests [39]. That node creates the map using the estimated pose and laser scanner. The map created by mapping node is represented as two dimensional grid where each cell (pixel) occupancy is expressed by probability function. Later the map is saved to the mapping server. Mapping server that provides services for other nodes to simply recover the saved map.

4.2.9 EKF Localization node

This node is also heart of the **SLAM** process. The node performs the process explained in *Chapter* 2.3.4. Node estimates odometry based on scan matching technique (landmarks), IMU readings and wheel odometry. This node estimates pose based on the relative pose differences generated from aforementioned mesaurements taking into the account the covariance of each of them. If the observations and the odometry done by dead reckoning provide similar state, the uncertainty is modified and covariance matrix updated. Depending on performance of estimation from various sources their reliability in covariance matrix is updated.

Chapter 5

Results and discussion

5.1 Autonomous mobility for electronic wheelchair

The construction and implementation of the **SLAM** was a continuous process with quite satisfactory from the technological point of view. Physical construction proven to be the simpler task to complete. Providing wide set of tools in **Control-Live** base and created by professional mobile agent body there were no many needs for physical changes. *Figure 5.1* presents the mobile base.

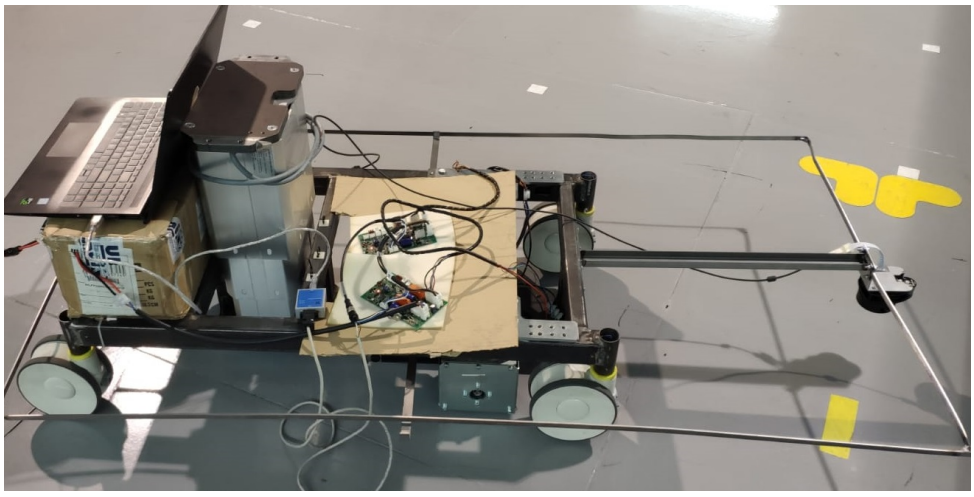


FIGURE 5.1: Final appearance of the autonomous electronic wheelchair

Before integrating the laser scanner sensor with the mobile agent the tests that aimed to produce proof-of-concept map were undertaken. After implementing the mapping algorithm without any dosimetry sources, the creation the map proved to be impossible. The pose update was solved by adding the scan matching algorithm that estimated the robot pose. That was the first step in creating the maps. First tests aimed to recreate the layout of few rooms in an apartment. For the sake of comparison the handmade top-view

of the house was prepared and afterwards the first mapping procedure was performed. The laser scanner unit was manually moved the the following apartment around on a stable platform and the results turned out to be very satisfying - *Figure 5.2*.

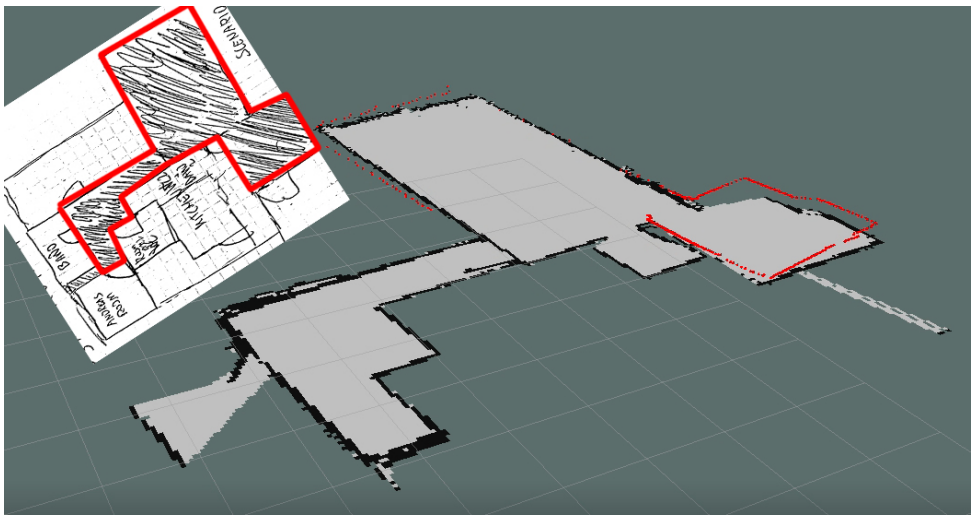


FIGURE 5.2: First ever created map in the development of autnomous electronic wheelchair

Having the basic mapping done, it was crucial to integrate the laser scanner with mobile agent. As soon as the basic interfacing with **CAN** network was implemented, the laser scanner was mounted to the structure of the mobile agent. Thanks to the **Control-Live** team, the integration to the base process went smoothly, letting the platform to be equipped with exteroceptive sensor (*Figure 5.3*).

First the sensor did not provide stable measurements and the investigation was undertaken. The imperfections in map creation were accumulated errors produced by various factors:

- Unstable and vibrating frame of the mobile agent base;
- Fast turns let scan matching algorithm to fail;
- Lack of odometry.

The frame was stabilized using additional weight and screws visible in *Figure 5.3*. Fast turn problem was addressed by reducing the in-place maximal turn speed of the mobile agent but in unknown environment the agent would lose the estimation of it's pose anyway. To address this problem, odometry was added to the system. First approach to odometry was generating odometry data based on built-in motor encoders. Unfortunately, the

FIGURE 5.3: Laser scanner unit connected to the mobile agent

motor encoder data was not yet published in **CAN** bus. During the waiting time for the base frame firmware update interfacing with it was further developed. Finally having the encoder data the odometry estimation was possible - (*Figure 5.4*).

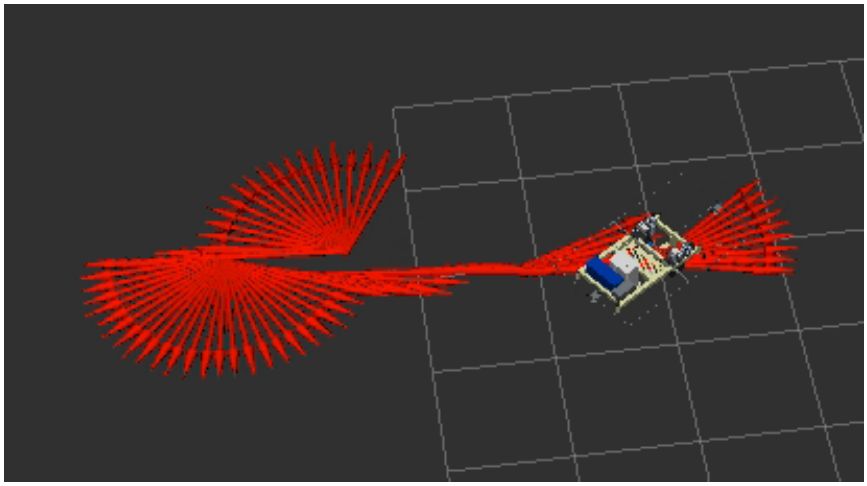


FIGURE 5.4: Mobile agent displacement visualization based only on wheel odometry

After further upgrade using the **IMU** mobile agent was able to estimate it's pose with good precision. The mobile agent's performance increased greatly. The test scenario was built according to the target environment - that

is space standards compatible with *iHFG International Health Facility Guidelines* (*Figure 5.5*).

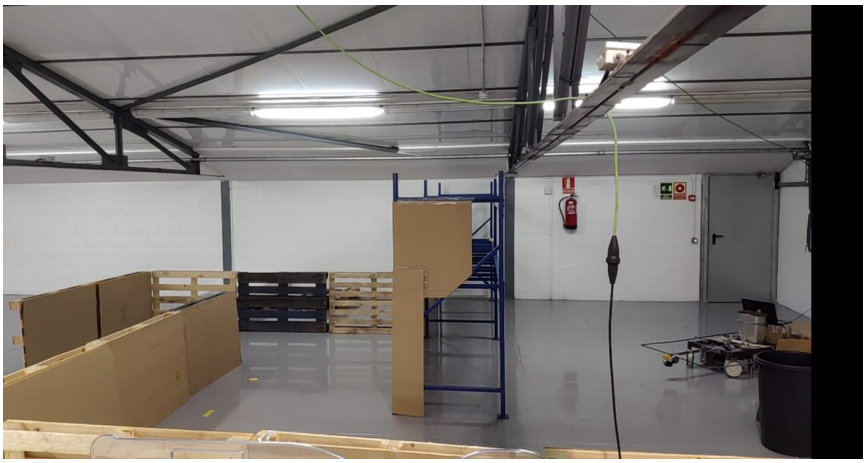


FIGURE 5.5: Mobile agent test scenario

Using the final stack, the autonomous platform was able to navigate in the created environment with or without prior knowledge of the map or without it (*Figure 5.6*).

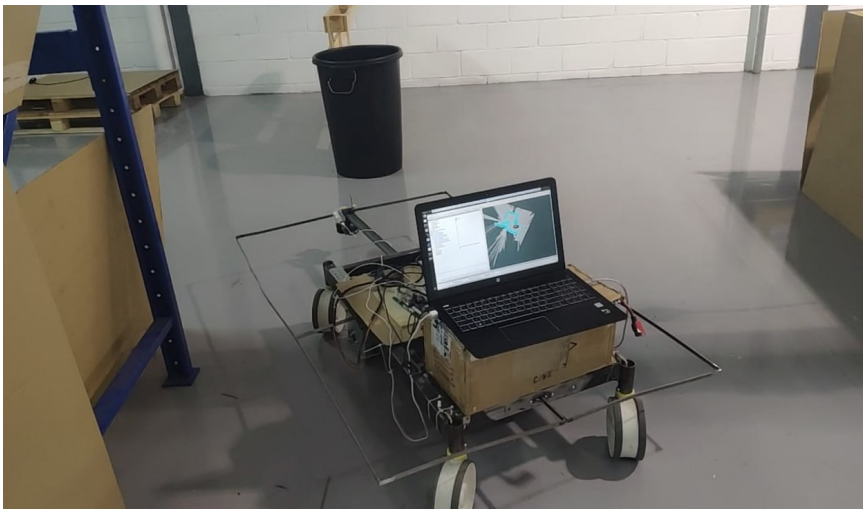


FIGURE 5.6: Autonomous platform performing **SLAM** - moving to new destination in new environment

The platform is fully capable of receiving voice commands, saving waypoints and moving between them when performing **SLAM** (*Figure 6.1*).

Chapter 6

Conclusions

Concluding - Applying autonomous mobility to any kind of mobile platform is not a trivial task. The **SLAM** methods are very susceptible to erroneous data. The **SLAM** algorithms behave very different in physical world compared to ideal simulations due to different sources of noise and imperfect sensors. This proves to be yet another example that world cannot be described in fixed, perfect parameters but it is necessary to create probabilistic models to work with. Creating project such as this one was a great challenge and much more further development needs to be undertaken before the reliable autonomous mobility can be created.

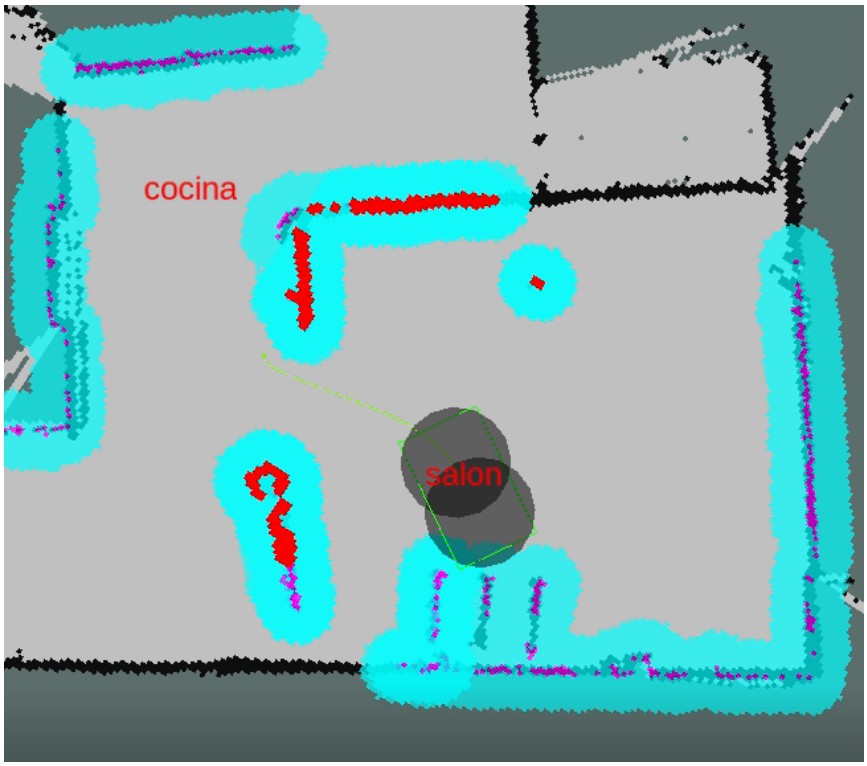


FIGURE 6.1: System visualization of mobile agent moving between known waypoints

Nevertheless, the project can be called a success thanks to completing following achievements:

- Expansion of the existing platform physically with new sensors and finding optimal locations for them,
- Creation of central unit that can operate between various environments (such as **CAN** or serial) and can be easily scaled to utilize new hardware,
- Creation of a dialog-based program that suits the navigation needs,
- Implenentation of the autonomous navigation algorithms thus solving the **SLAM** problem,
- Sensor calibration to produce the reliable data,
- Fusion of the sensor data using probabilistic algorithms, thus allowing the reliable estimation of the robot localization.

6.1 Open problems and further development

This project can be called a proof of concept of autonomous mobility for electronic wheelchair. The autonomy of a mobile platform designed for people with **SCI** is a noble cause but integrating it into real-life solution is far from done. To assure the safety of the solution many tests must be undertaken in different environments and sources of noise. System also requires addition of additional recovery behaviors and qualitative testing done to measure the levels of comfort and trust of potential users.

Bibliography

[1] The Oxford English Dictionary. *Paraplegia definition*. 2019. URL: https://www.lexico.com/en/definition/paraplegia.

[2] National spinal cord injury statistical center. *Spinal Cord Injury Facts and Figures at a Glance*. 2017. URL: https://www.nscisc.uab.edu.

[3] World Health Organization. *Spinal cord injury: as many as 500 000 people suffer each year*. 2013. URL: https://www.who.int/mediacentre/news/releases/2013/spinal-cord-injury-20131202/en/.

[4] Instituto Nacional de Estadistica. *Population with paralysis-related disabilities*. 2017. URL: https://www.ine.es.

[5] McInnes Elizabeth. "Support surfaces for pressure ulcer prevention". In: *a* (2015). DOI: doi:10.1002/14651858.CD001735.pub5.ISSN1469-493X.

[6] H.F. Leonard J.J.; Durrant-whyte. "Simultaneous map building and localization for an autonomous mobile robot". In: *I* 322.10 (1991), pp. 1442–1447. DOI: doi:10.1109/IROS.1991.174711.

[7] Cyrill Stachniss. "Robotic Mapping and Exploration". In: *a* (2015). DOI: ISBN978-3-642-01096-5.

[8] S. A. McLeod. *Visual perception theory*. 2018. URL: https://www.simplypsychology.org/perception-theories.html.

[9] J. Ventura. "Global localization from monocular slam on a mobile phone". In: *IEEE Virtual Reality* (2014).

[10] William Narmontas. *Limit degrees of freedom in development*. 2016. URL: https://www.scalawilliam.com/1612/limit-degrees-of-freedom/.

[11] Chris Clark. *COS 495 - Lecture 7 Autonomous Robot Navigation*. Princeton University. URL: https://www.cs.princeton.edu/courses/archive/fall11/cos495/COS495-Lecture7-SensorCharacteristics.pdf (visited on 06/04/2019).

[12] Cecilia Soriano, José M Baldasano, William Buttler, Kurt R. Moore. *Circulatory Patterns of Air Pollutants within the Barcelona Air Basin in a Summertime situation: Lidar and Numerical Approaches*. Princeton University. 2000. URL: 10.1023/A:1018726923826.

[13] INC. David S. Hall VELODYNE LIDAR. "High definition lidar system". US7969558B2. 2007.

[14] Charles Birdsong John Carlin. "Evaluation of Cost Effective Sensor Combinations for a Vehicle Precrash Detection System". In: California Polytechnic State University, San Luis Obispo (2005).

[15] Joan Rosell Jose Molin. "Application of light detection and ranging and ultrasonic sensors to high-throughput phenotyping and precision horticulture: current status and challenges". In: *International Journal of Engineering and Technical Research* (2018). DOI: 10.1038/s41438-018-0043-0ID.

[16] HD Mckay PJ Besl. "A method for registration of 3-d shapes." In: *a* Pattern Analysis and Machine Intelligence (1992).

[17] Qing Zhu Shengjun Tang. "Enhanced RGB-D Mapping Method for Detailed 3D Indoor and Outdoor Modeling". In: *a* Pattern Analysis and Machine Intelligence (2016).

[18] Mohammed Ibrahim. *MDP Mechatronics Lecture 06: Sensors*. Ain Shams University, Faculty of Engineering. 2019. URL: https://www.slideshare.net/MohamedAtef80/lec-06sensors.

[19] Bosch. *IMU Sizes*. 2015. URL: https://www.bosch-sensortec.com/bst/products/allproducts/bmi260.

[20] Josep Aulinas et al. "The SLAM problem: a survey". In: *The SLAM problem: a survey*. Vol. 184. Jan. 2008, pp. 363–371. DOI: 10.3233/978-1-58603-925-7-363.

[21] Wolfram Burgard, Cyrill Stachniss, Kai Arras, Maren Bennewitz. *SLAM: Simultaneous Localization and Mappin*. 2018. URL: http://ais.informatik.uni-freiburg.de/teaching/ss12/robotics/slides/12-slam.pdf.

[22] Patric Jensfelt Simone Frintrop and Henrik Christensen. *Detecting Useful Landmarks for Visual SLAM*. 2015. URL: https://pdfs.semanticscholar.org/abfe/5cb9c4dfea873273e84a05ca810d145705b9.pdf.

[23] Joaquim SALVI Josep AULINAS Yvan PETILLOT and Xavier LLADÓ. *The SLAM problem: a survey*. 2017. URL: http://eia.udg.es/~qsalvi/papers/2008-CCIAa.pdf.

[24] J.M.M. Montiel D. Ortín J. Neira. *Relocation using Laser and Vision*. 2004.

[25] Søren Riisgaard and Morten Rufus Blas. "SLAM for Dummies". In: *a* (2014).

[26] H. Choset. "Principles of Robot Motion: Theory, Algorithms, and Implementation". In: *m* (2005).

[27] Christi Olson. *Just say it: The future of search is voice and personal digital assistants*. 2017. URL: https://www.campaignlive.co.uk/article/just-say-it-future-search-voice-personal-digital-assistants/1392459.

[28] Olle Ferling Johannes Koch. *Speech Recognition using a DSP*. 2017. URL: https://www.eit.lth.se/fileadmin/eit/courses/etin80/2017/reports/speech-recognition.pdf.

[29] Mark Schankerman Josh Lerner. "The Comingled Code: Open Source and Economic Development". In: *Cambridge, MA: MIT Press* (2010).

[30] Md Javedul Ferdous and Niladri Shekhar. "Comparison on Booting Process and Operating Systems Features". In: *2017* (2018).

[31] StatCounter Global Stats. *Operating System Market Share Worldwide*. 2019. URL: http://gs.statcounter.com/os-market-share#monthly-201607-201904.

[32] Hyperphysics. *Hall Effect*. 2010. URL: http://hyperphysics.phy-astr.gsu.edu/hbase/magnetic/Hall.html.

[33] Kristof Robot Tang Tiong Yew. *Sparkfun Razor IMU 9DOF ROS Package documentation*. 2017. URL: http://wiki.ros.org/razorimu9dof.

[34] voice INTER connect. *Voice Control, Beamforming, Echo Cancellation, Intercom*. 2018. URL: https://www.voiceinterconnect.de/en/.

[35] ROS. *TCPROS*. 2018. URL: http://wiki.ros.org/ROS/TCPROS.

[36] ROS. *LaserScan Message*. 2017. URL: http://docs.ros.org/melodic/api/sensormsgs/html/msg/LaserScan.html.

[37] ROS. *IMU Message*. 2017. URL: http://docs.ros.org/melodic/api/sensormsgs/html/msg/Imu.html.

[38] ROS. *Move base*. 2017. URL: www.ros.org/wiki/move%20base.

[39] Stefan Kohlbrecher. *Hector SLAM Tutorial*. 2012. URL: http://tedusar.eu/cms/sites/tedusar.eu.cms/files/Hector%5C_SLAM%5C_USAR%5C_Kohlbrecher%5C_RRSS%5C_Graz%5C_2012.pdf.