SIMULTANEOUS LOCALIZATION AND MAPPING IN LIGHTWEIGHT AUTONOMOUS VEHICLES

YAGO LIZARRIBAR CARRILLO

Application of autonomous vehicle techniques in low-cost light electric platforms

Master of Science in Mechanical Engineering (Ing.)
Tecnun–University of Navarra

July 2018 – version 1.0

Yago Lizarribar Carrillo: *Simultaneous Localization and Mapping in Lightweight Autonomous Vehicles,* Application of autonomous vehicle techniques in low-cost light electric platforms, © July 2018

ABSTRACT

Self–driving vehicles will be a reality in the not so distant future of cities. Much of the attention today is focused on cars, although they face great challenges in years to come before their implementation, thus creating an ample spot for lightweight alternatives to blossom. This thesis is about one of those alternatives, the **Persuasive Electric Vehicle**, and the approaches taken to overcome the problems autonomous vehicles have. Specially interesting is the problem of **Simultaneous Localization and Mapping** due to its complexity and possibilities it opens to achieve fully autonomous platforms. In the chapters that follow, the reader will find how it was tackled, what are the results employing different techniques and how it could be improved so that autonomous lightweight vehicles can start to be seen on the streets of our cities.

Keywords— Autonomous Vehicles, Cities, Lightweight platforms, Shared Mobility, Simultaneous Localization and Mapping

RESUMEN

Los **vehículos autonómos** serán parte de la realidad de las ciudades en un futuro no muy lejano. La mayor parte de la atención está fijada en los coches, aunque deberán superar una gran cantidad obstáculos antes de ocupar las urbes, dejando un espacio de crecimiento para otras alternativas. Esta tesis es sobre una de esas alternativas, el **Persuasive Electric Vehicle**, y las metodologías implementadas para superar dichos obstáculos. Especialmente interesante es el problema de la **Localización Simultánea y Mapeado** debido a su complejidad y las posibilidades que abre para el desarrollo de plataformas completamente autonónomas. En los siguientes capítulos, el lector aprenderá como se ha afrontado dicho problema, cuáles son los resultados empleando diferentes técnicas y qué aspectos se debería reforzar para ver en el futuro vehículos autónomos ligeros circular por las ciudades del mundo.

Palabras clave— Vehículos Autónomos, Ciudades, Plataformas Ligeras, Movilidad compartida, Localización Simultánea y Mapeado

In other words, the rule about great scientific advances is that to make them you have to break the rules. Nobody has ever won a Nobel Prize doing what they're told, or even by following someone else's blueprints.

—Joichi Ito, Director of the MIT Media Lab

ACKNOWLEDGEMENTS

First of all, thank you **Kent Larson** for this amazing opportunity. Your decision of letting me in the City Science group opened my eyes and showed me worlds that I could not even imagine.

I am enormously grateful to **Michael**, **Abhishek** and **Phil** for their leadership of the mobility team and for taking me in without hesitation. Thanks to **Jerry** for all these months full of great moments and hard work side by side until very late hours. To **Justin, Luke** for all those lessons and and fun moments. To my Taiwanese family **Ting–Kai and Frank** for teaching me, for helping me and for welcoming me in your wonderful country.

To **Luis** and **Azucena** for all the invaluable help these months, for all those pieces of advice, for everything in general. And this gratefulness is extended to the whole City Science family: **Ana, Andrés, Ariel, Arnaud, Carson, Chrisoula, Jason, Laya, Lucas, Maggie, Markus, Mary, Nina, Roonan, Ryan, Thomas and Yasushi**.

To **Jose** and **Eduardo**, for those great talks about every subject imaginable and lessons to become a better researcher.

Many thanks to **Tecnun** for nominating me to go to these amazing place and all those professors that throughout these 6 years have inspired me and helped me become a better engineer and person.

To my family, my parents, my friends here and there, for their support both emotional and finantial. And specially to you, **I**...

THANK YOU ALL!

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

AI	Artificial Intelligence
API	Application Programming Interface
AV	Autonomous Vehicle
CS	City Science
CV	Computer Vision
DARPA	Defense Advanced Research Projects Agency
DWA	Dynamic Window Approach
EKF	Extended Kalman Filter
GPS	Global Positioning System
GPU	Graphics Processing Unit
ICP	Iterative Closest Point
IMU	Inertial Measurement Unit
LIDAR	Light Detection and Ranging
MDP	Markov Decision Process
ML	Media Lab
NDT	Normal Distributions Transform
PEV	Persuasive Electric Vehicle
ROS	Robot Operating System
RTK	Real Time Kinematics
SAV	Shared Autonomous Vehicle
SLAM	Simultaneous Localization and Mapping
TAF	Taipei Air Force
UKF	Unscented Kalman Filter
VMT	Vehicle-Miles Travelled

1

INTRODUCTION

This chapter will give a justification for the Persuasive Electric Vehicle (PEV) project. For that, the 3 main topics that caused this idea to flourish will be discussed, including their weak points and how PEV tackles them. These topics are: cities, autonomous vehicles and shared mobility.

THE FUTURE OF CITIES

In the coming decades, it is expected that population in urban areas will grow from 50% to 70% [32]. Furthermore, 95% of that expansion will happen in developing countries [28]. As a result of this migration important social and environmental challenges will arise such as fuel production, air pollution, waste management, urban transportation [12], etc.

To this day, cities are dealing with deeply concerning issues regarding transportation that will only get worse with the rapid growth it is expected in the next years [23, 45]:

- **Traffic congestion**: The number of private cars added to the amount of trucks, vans, taxis, buxes and rest of vehicles have lead to gridlocks in major cities across the globe. This causes increased traffic pollution, longer commutes and higher tensions among drivers.
- **Sprawling cities**: The rapid growth of cities has pushed people to the outskirts due to an exponential increase in the housing price. Therefore commutes become longer and transportation networks grow more complex.
- **Parking**: Parking is a major issue due to the amount of space it occupies, thus increasing the already high land demand inside urban environments.

Over the last decades, the term 'Smart City' has become more popular to describe cities that make investments in novel infrastructures and Information and Communication Technologies (ICTs) [32]. The purpose for this trend is to accelerate economic growth while managing resources more efficiently, being one of those resources transportation [6].

Among the various solutions that have been proposed to improve mobility infrastructure, 2 of them will be discussed due to their close link to the main topic of this thesis: **Autonomous Vehicles** and **Bicycle Sharing**.

AUTONOMOUS VEHICLES. FUTURE OF TRANSPORTATION?

The field of Robotics has gained a great deal of traction in the last years, whether it is for the field of biomedicine, construction or Artificial Intelligence [43]. And the field of Autonomous Vehicles (AVs) has followed this trend.

AVs, once utopy, are becoming every passing year a real possibility in the not-so-far future [1]. Apart from big names such as Google's Waymo, Tesla, Uber, Lyft or Nvidia, this decade has seen the surge of smaller companies dedicated to developing self-driving car technologies. The latter group is where companies like NuTonomy and Optimus Ride belong. Figure 1 shows some examples of these vehicles.

(a) Google's Waymo spinoff company

(b) Tesla's Autopilot

(c) Nutonomy's AV

(d) Optimus Ride's autobus

Figure 1: Current state of AVs

Some history on Self-Driving Vehicles

First attempts to build robotic vehicles date as early as 1920s [5] when the first radio controlled vehicles were designed. In the decade of 1980, there were serious attempts at building autonomous vehicles, such as the following projects:

- **Prometheus** (Program for a European Traffic with Highest Efficiency and Unprecedented Safety) project: The aim of this project was to build a civilian car to navigate on highways or cities (i. e.more structured environments) [42, 11]. The vehicle chosen was a Mercedes-Benz van designed by Ernst Dickmanns and during test drives on highways

it achieved a maximum speed of 96 km/h and drove for about 20 km[1].

- **ALV** (Autonomous Land Vehicle) project: This project was developed by DARPA in America (Prometheus was from Europe) in collaboration with Carnegie Mellon University. The van employed was able to navigate, plan routes and avoid obstacles in coarser terrains than the Prometheus van for 5 km and at a speed of 20 km/h [21, 13].

Both cars are shown in Figure 2.

(a) Prometheus van (b) DARPA's Autonomous Land Vehicle

Figure 2: The first serious attempts to build AVs

In 2003, DARPA launched the Grand Challenge [2] to promote the development of unmanned vehicles. The Challenge was to drive a car autonomously through an unknown off-road terrain. More specifically, a 142 mile-long course across the Mojave desert in less than 10 hours [38] [30]. There were 107 teams registered, 15 of which made it to the race, although none of the participants drove further than 5% of the total length. In 2005, the challenge was repeated and 5 teams managed to finish the race out of 197. Stanford University won that race with a car named Stanley (Figure 3).

In 2009, Google launched the Self-Driving Car project that became a company named Waymo. Since its inception, this project has achieved very interesting milestones and it is the company that has driven more miles autonomously by far [16].

Current state of autonomous driving and levels of autonomy

Nowadays, virtually almost every car has some sort technology embedded in it, that provides it with a certain level of autonomy or 'intelligence'[22]. Examples of these advancements are [37]:

- **Self-parking**: A couple years ago, automakers like Toyota, Mercedes or BMW started developing cars that could find parking spots and maneuver the car to eventually fit it in. These technologies generally

1 History on the car: https://www.mbscottsdale.com/blog/mercedes-benz-whensday-the-prometheus-project/

2 http://archive.darpa.mil/grandchallenge/

Figure 3: DARPA Grand Challenge's first winner: STANLEY

rely on ultrasonic sensors and cameras, taking into account cars' movement restrictions [24] (it will be explained in Chapter 2).

- **Automatic emergency-braking**: It detects imminent crashes and attempts to minimize the impact of them. These technology alerts the driver first with visual or sound warnings and if there is no response, the vehicle stops immediately.
- **Semi-autonomous drive system**: There are already cars on the market that can drive themselves autonomously on highway conditions. This is the case of Tesla's Autopilot or Cadillac's Super Cruise. These systems are capable of adapting to traffic conditions on highways and even steering on more complicated roads, but they still require the driver to pay attention to the road.

According to the Society of Automotive Engineers (SAE), there are 6 levels of automation[35] as show on Figure 4.

The first 3 levels (from 0 to 2) is where most cars lie nowadays, specially levels 1 and 2 thanks to the advances mentioned above. That means humans are still in charge of monitoring the environment. Tesla's Autopilot lies on level 3 though under certain conditions but it is still under development after the deadly crash in 2016 [2]. With regards to level 4 autonomy (in other words, car is able to full self-drive but under certain road and weather conditions), Uber and Waymo announced plans to test driverless taxis last November [4, 20].

Nevertheless, level 5 autonomous cars seem to be still a distant possibility. According to SAE (Figure 4), level 5 implies that no human intervention would be needed in any condition and that is not the case today, since AVs cannot be operated under heavy rain or snow conditions, unpaved roads or with diverse traffic [34].

SAE level	Name	Narrative Definition	Execution of Steering and Acceleration/ Deceleration	*Monitoring* of Driving Environment	Fallback Performance of *Dynamic Driving Task*	System Capability *(Driving Modes)*
***Human driver* monitors the driving environment**						
0	No Automation	the full-time performance by the *human driver* of all aspects of the *dynamic driving task*, even when enhanced by warning or intervention systems	Human driver	Human driver	Human driver	n/a
1	Driver Assistance	the *driving mode*-specific execution by a driver assistance system of either steering or acceleration/deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	Human driver and system	Human driver	Human driver	Some driving modes
2	Partial Automation	the *driving mode*-specific execution by one or more driver assistance systems of both steering and acceleration/ deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	**System**	Human driver	Human driver	Some driving modes
***Automated driving system* ("system") monitors the driving environment**						
3	Conditional Automation	the *driving mode*-specific performance by an *automated driving system* of all aspects of the dynamic driving task with the expectation that the *human driver* will respond appropriately to a *request to intervene*	System	**System**	Human driver	Some driving modes
4	High Automation	the *driving mode*-specific performance by an automated driving system of all aspects of the *dynamic driving task*, even if a *human driver* does not respond appropriately to a *request to intervene*	System	System	**System**	Some driving modes
5	Full Automation	the full-time performance by an *automated driving system* of all aspects of the *dynamic driving task* under all roadway and environmental conditions that can be managed by a *human driver*	System	System	System	**All driving modes**

Figure 4: Levels of Autonomy according to SAE

Advantages of AVs

The rise of self-driving cars and related technologies is not a product of coincidence rather than the general belief among scientists and industry that they will bring many benefits in terms of safety, productivity and wealth [22]. In the following paragraphs, some of these advantages will be discussed.

Road safety

ROAD SAFETY According to the World Health Organization (WHO), every year around 1.25 million people lose their lives in car accidents and 20-50 million people get injured [41]. In the United States human caused car accidents account for 90% of the crashes [9, 40]. By adopting AVs in large scales, distractions will happen less frequently, therefore reducing the number of deadly accidents.

Improved mobility

IMPROVED MOBILITY The daily commute is one of the tasks AVs could help many drivers with. On average, it takes 26 minutes [39] to go from home to work and back, time that could be spent doing more productive or rewarding tasks. Self-driving vehicles could be transformed into mobile bedrooms, playgrounds or offices to suit the demand of their users [citation], as it is Mercedes' self-driving concept car shows for example (Figure 5)[3].

Better Traffic Flow

BETTER TRAFFIC FLOW Some researchers have shown that AVs can improve traffic flow [36]. Since self-driving cars might be able to connect to

3 https://www.youtube.com/watch?v=8aEWHdduPwc

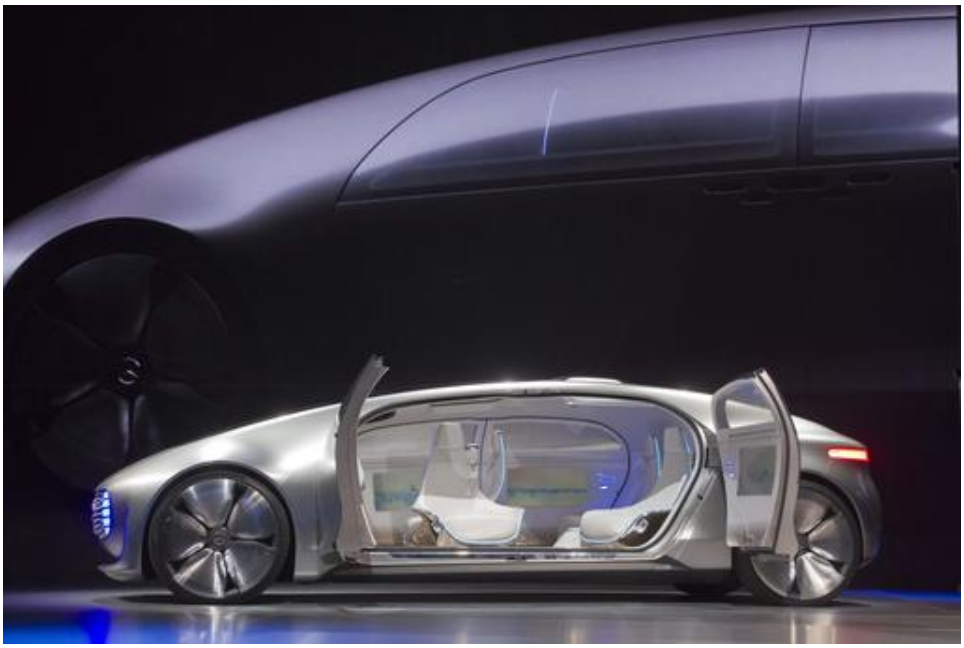


Figure 5: Mercedes self-driving car

other cars and features of the environment, and have lower response times, they can:

1. Anticipate events faster and allow for smoother braking and acceleration.
2. Use lanes and intersections more effectively by reducing gaps and platooning (Figure 6).

Figure 6: Platooning representation on a highway

Shared Networks

SHARED NETWORKS One of the key attactives for AV technology is that it could enable new travelling modes (combining provate and public) such as communally-owned shared vehicles [15]. Shared Autonomous Vehicles (SAVs) might potentially benefit the environment by reducing the need for car ownership and parking [10]. Users with similar destinations could share rides (in the same manner as with current carsharing services). However, the main advantage over what is available today, is that after the ride they could autonomously relocate themselves to more demanded areas, thus

reducing the need for parking lots that could be transformed in green areas [22].

Disadvantages of AVs

Even though self-driving technology has a vast potential to improve many aspects of society, it is not perfect and it raises concerns among researchers, policy makers and general public. Many of the positive aspects described above have their negative counterparts as it will be discussed in the next paragraphs.

HIGH COSTS

HIGH COSTS Even though costs could be reduced due to car reduction, the technology needed to transform a regular car into a self-driving one increases the prize dramatically. Each of the Light Detection and Ranging (LIDAR) sensors installed can cost between 30.000$ and 85.000$ [33] which makes AVs unaffordable for the vast majority of the population.

LIABILITY AND ETHICAL QUESTIONS

LIABILITY AND ETHICAL QUESTIONS As AVs become more pervasive in everyday landscape, questions will arise when these vehicles are involved in any type of accident [9]. When car's software fails, who is to blame: the software engineer, the company? How can an insurance company be convinced of the reliability of self-driving vehicles?

Moreover, if a crash is inevitable and the car has to decide between 2 evils, how can it make the decision? These ethical questions pose a major challenge when it comes to real world implementations and there is no clear answer. From MIT Media Lab's group Scalable Cooperation, an initiative called Moral Machine [25] was launched in order to create a discussion on these moral dilemmas and build a 'crowd-sourced' image of how intelligent machines should behave[4].

PUBLIC PERCEPTION

PUBLIC PERCEPTION Another issue to overcome is public perception. First of all, because when self-driving car technology reaches the mass market, professional drivers might start losing their jobs [22]. Trust on the technology is an important obstacle to overcome, since few people will feel uncomfortable at first, being the technology new and not sufficiently tested [17, 3].

SAFETY AND SECURITY

SAFETY AND SECURITY Electronic security is a major concern for every major car- and policymaker, and for the general public [29]. Hackers, terrorist groups or employees could target the intelligent system of the vehicles and cause major disruptions on the whole network [9], by forcing cars to collide or exposing users' locations.

4 A test can be taken with different scenarios: http://moralmachine.mit.edu/

When will AVs hit the road?

Due to these challenges and many other that arise when developing self-driving technologies, experts believe level 4 and 5 AVs will not be available to the general public until the 2040s–2050s [22, 27].

SHARED MOBILITY

Shared mobility has already been mentioned as a future field of application of AVs, but the growth it has undergone in past years by itself is worth noting, since it has crutial implications for the future of transport in cities.

Many car sharing companies have already achieved a great market share like Zipcar, Car2Go, or even Uber's latest services. However, this section will focus on bicycle sharing systems, due to their applications and benefits for cities across the world.

What is bicycle sharing?

Bicycle sharing or bikesharing corresponds to the public shared use of bicycle fleets that has gained attention these past years [8, 31] in many regions of the world such as Europe, US and China.

The case of China is specially striking, since many bikesharing startups have sprung out in the past years, and the biggest companies among those, Ofo and Mobike, already have 19 million bikes across the world [44].

Nevertheless, bicycle sharing it is not a radically new concept from this century. In fact, its origins trace back to the 1960s and there have 3 generations in bikesharing history [8]:

- The first generation started in 1965 and the concept was to provide with ordinary bikes that could be parked anywhere. However, bikes were vandalized very often.
- Second generation was born in 1993, when bikes had to be parked on specific docks or stations. Although it improved the previous versions, bicycles were still subject to theft due to user anonimity.
- Third generation (the current one) bikes already incoporate smart technologies like electronic locks or bicycle tracking to improve security.

The rise of this transportation method is not casual since it comes as an answer to 3 key aspects in cities [8]:

- **Increased cycle usage**: This way healthier lifestyles are promoted in urban environments.

- **Increase first/last mile connection**: Bike sharing can provide its users with connection from their homes/working areas to other means of transport such as buses, trains or subway. Moreover, bike fleets can be used to tranport not only people but packages and other goods across the city, thus avoiding traffic congestions caused by larger vans or trucks.
- **Reduce environmental impacts**: Biking is an emission–free means of transport and therefore it can help mitigate common concerns in cities like global climate change, energy supply or varying fuel prices [31].

What are the current disadvantages of these approaches?

Again there is no perfect solution to urban transportation and bicycle sharing suffers essentially from 2 issues.

SYSTEM REBALANCING: The first one, and this occurs to shared automobiles as well, is that they have to be manually relocated in order to guarantee the balance of the system. This implies arranging a logistics network around the city in order to ensure every area they are operated within has a minimum amount of bikes.

BIKE CEMETERIES: Because of these attempts to equilibrate the network, companies must provide with a higher amount of bicycles than necessary and keep higher stocks, all resulting in greater maintenance costs and many times bicycles end up in in landfills. Combining this with market saturation as it is happening in China's markets, 'bike cemeteries' are being formed in the outskirts of cities like Shanghai or Beijing (Figure 7) [7].

Figure 7: Unused bicycles end up forming immense cementeries around cities

PERSUASIVE ELECTRIC VEHICLE (PEV)

MIT Media Lab: Home of Innovation

To understand the PEV project, it is necessary first to put some context on the place of its inception, the Media Lab. According to their website, Media Lab is "an antidisciplinary research lab working to invent the future"[5]. In fact it is a place that gathers all sorts of people from various backgrounds such as Arts, Sociology, Computer Science or Engineering and aims at developing ideas that bring technology closer to humans. Since its foundation in 1985, more than 150 companies have spun out of its doors[6].

Some of the notable projects are:

- **Scratch programming language**: It is a block-based language initially developed for children but that it is used by people all ages around the world [26]
- **Harmonix**: Game development company that launched Rock Band and Guitar Hero.
- **E-ink**: Company that developed the electronic ink, now used in e-readers around the world [18].

City Science and its mission

City Science is one of the 25 groups of this ecosystem and it is where the PEV is located. The group, led by Professor Kent Larson, is aware of the challeges cities will face in the coming years and is focused on developing platforms and tools that will improve life in cities [19]. There are 3 main research topics at City Science:

- **CityScope**: It is a platform that aims at providing tools for cities and urban planners to model and design infrastructures with the use of simulations and real time data [14]. The whole system is built around a Lego table, where decision makers can 'play' and see the effects of their solutions in real-time (Figure 8).
- **Changing Places**: This group focuses on developing responsive places to live and work in future cities. 2 of the most famous projects are Cityhome which later became the spinoff Ori which provides robotic furniture (Figure 8), and Escape Pod, a transformable space for either working or resting.
- **Mobility-on-Demand**: This group is focused on efficient shared–use mobility systems and its main project is the aforementioned **PEV** which will be explained in more detail in the following lines.

5 https://www.media.mit.edu/research/?filter=groups
6 https://www.media.mit.edu/about/spin-off-companies/

(a) CityScope (b) Ori system

Figure 8: City Science's CityScope and Ori platforms

Reason for the PEV

The PEV is an autonomous, lightweight, low–cost, electric and shared tricycle (Figure 9) that aims to be an alternative to both autonomous vehicles and bikesharing systems.

Figure 9: PEV on the steets of Cambridge, Massachusetts (August 2017)

The PEV would be operated in 2 different modes:

- **People mover**: People would be able to call a PEV with a phone app (like Uber) and the vehicle would come to the user autonomously. Then the requester would have to ride it to the destination and after the ride, the PEV would leave on its own to another location.
- **Package delivery**: PEV could be used as a package delivery system while user demand is low (for example, at working hours). This way

it could help reduce traffic congestion caused in cities by trucks or vans.

It has already been explained taht both AVs and bikesharing systems have their downsides. The idea of the PEV was born to tackle the issues that arise on these approaches as Table 1 shows.

Aspect	AVs	Bike Sharing	PEV
Environmental Impact	Even though they might reduce traffic, environmental cost will be high	Minimal impact	Slightly bigger (when in autonomous mode) but minimal when riding it
Healthy lifestyle	They do not promote healthier lifestyles	Cycle usage increases physical activity	Passengers will have to pedal, thus promoting a healthier transport
Infrastructure	AVs need high-quality roads and special infrastructure that might be costly	They can run on current bike-lanes	It is designed to fit on bike lanes, so it would require less infrastructural support
Road safety	They are subject to strict regulations due to their risk, so that might slow down their deployment	It is less dangerous so regulation is not as strict	As it it is lightweigth and will operate at low speeds on bike lanes, risk will be lower than with self-driving cars. Therefore deplyoment might be faster
Parking	They need minimal parking	They need a notable amount of stations	They need minimal parking
Relocation	They can relocate automatically based on demand	The need to be manually relocated	They can relocate automatically based on demand

Table 1: Comparison between AVs, bikesharing systems and PEV

THESIS STRUCTURE

This thesis is divided as follows:

- **Chapter 2** explores the current technologies available for autonomous vehicles and describes the fundamental elements of the main aspects of self-driving vehicles.
- **Chapter 3** describes the field of Simultaneous Localization and Mapping (SLAM) more thoroughly, as well as providing with the latest approaches.
- **Chapter 4** analyzes different slam algorithms applied on a small autonomous robotic platform, that served as a testbed for the PEV.
- **Chapter 5** analyzes the slam approach on the PEV, and what are the best configurations for different scenarios.
- **Chapter 6** shows a simple budget for the project and its timeline.
- **Chapter 7** provides with conclusions on the results obtained on both platforms, improvements that could be tackled and sets the next steps for these vehicles.

2

AUTONOMOUS VEHICLE CONCEPTS

In essence, AVs are robots (with 2 degrees of freedom, generally) and more specifically fall in the branch called **Mobile Robotics**. The following pages will explore the basic concepts of it and what are the tools available to tackle this field.

WHAT IS MOBILE ROBOTICS

Examples of mobile robotics

The use of robots to assist or substitute human labour dates back to the 1960s [55] with industrial manipulators and to this day remains as one of the most successful applications of this field. However, generally these robots lack moving capabilities, thus they are limited to a narrow operating environment.

Mobile Robotics is a relatively recent branch of robotics which aims at developing platforms with moving capabilities [72], in order to achieve tasks that are impossible to do for fixed robots.

Applications in the field of mobile robotics range from:

- **Material delivery**: This is the case for Kiva Systems, company acquired by Amazon in 2012, and whose core idea is to use swarms of mini-robots to transport materials across the warehouse [56]. Mobile robots (in this case drones) have started to be used by companies like Amazon, Google and DHL to develop last mile delivery logistics [66].
- **Home applications**: This field is where the house cleaning robots called Roomba developed by iRobot lie. These platforms are equipped with different sensors that allow them to map their environment and navigate in order to clean the rooms of the household [74].
- **Autonomous Vehicles**: As it was shon on Chapter 1, there are many possible areas where self-driving vehicles could be applied.
- **Unknown Environment Exploration**: Robots with moving capabilities are employed to explore unknown environments in rescue missions [46] or in other planets [61].
- **Defense**: As it has happened with many other scientific advances in history, there is interest in developing mobile robotics with military goals. This is the case of Boston based company Boston Dynamics

(a) Kiva system's robot in a warehouse (b) iRobot's Roomba

Figure 10: Kiva System's robots and Roomba share some design principles

(a) NASA's Curiosity rover on Mars (b) Boston Dynamic's SpotMini (2018)

Figure 11: Built to explore the unknown, with different goals in mind

and their four-legged robot BigDog [71] capable of travelling through diverse outdoor terrains.

Robot control paradigms

The paradigms of control for mobile robotics are very similar to the ones used in robotics and control in general [49]. Many of the robots mentioned above follow the classical paradigm of sense-plan-act (Figure 12). The explanation of it is very straightforward: robot senses the environment, then feeds its algorithms with that data and produces an output, that goes into the system's actuators. That response is measured in the next iteration, forming a closed-loop.

Figure 12: Classical control paradigm

However, this approach requires a larger amount for computing power, which is not suitable for smaller platforms like the Roomba or other home robotics [49]. Therefore, these types of robots used a more reactive paradigm, where the planning step is removed and substituted by direct mapping of the sensor inputs to specific commands (Figure 13).

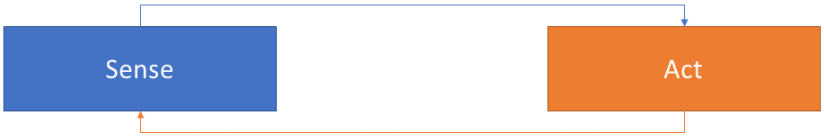


Figure 13: Reactive control paradigm

For the case of self-driving vehicles, the first approach is more suitable, since situations these robots encounter tend to be more complex and require a greater understanding of the environment [72]. That is why the classical paradigm is preferred and used in AVs. Figure 14, shows a more fine grained description of the control of an autonomous vehicle like the PEV.

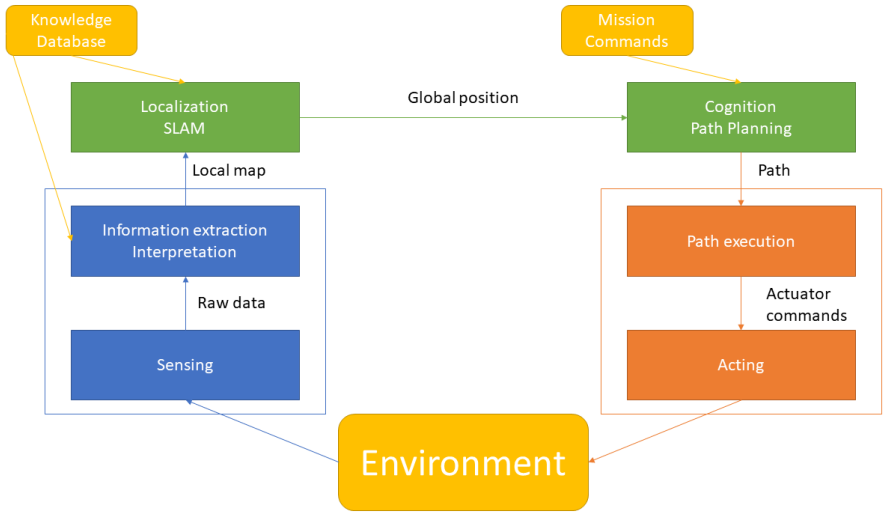


Figure 14: Extended control paradigm for mobile robots

The 3 aspects of control are present (sense-plan-act) but each one is subdivided in different categories. Each one of these constitutes is equally important to consider when building an autonomous platform and they will be described in the following section.

FUNDAMENTALS OF MOBILE ROBOTICS

Control and Locomotion

It has been mentioned before that classical robotic arms operate generally fixed to the ground in environments with moving objects, whereas mobile ones are in charge of navigating in mostly static scenarios. Therefore, it can be deduced that mobile robot's most fundamental feature is **locomotion** [72] and it plays an important role on the design of the platform.

Locomotion can take various forms but the most studied mobile robots are **legged** and **wheeled** (Figure 15).

(a) Boston Dynamics' Atlas

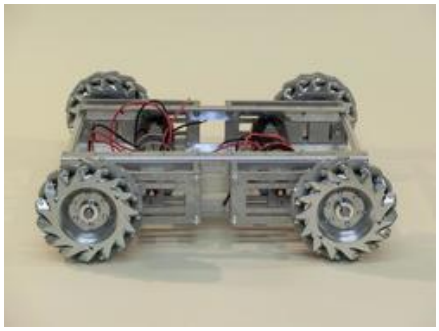
(b) Yujin Robot's turtlebot

Figure 15: Legged vs Wheeled robots

This thesis explores more in detail wheeled locomotion, since it is the basis for both robots that will be described in Chapter 4 and Chapter 5.

The wheel is by far the most popular mechanism for movement and there are various types [50]:

- **Differential drive**: These robots have 2 driving wheels and having each a motor attached. Movement varies due to 'differences' in the speed of both wheels.
- **Car drive (Ackerman steering)**: In this case one motor is in charge of the forward movement, whereas the other one steers the wheels.
- **Omnidirectional drive**: These robots are capable of moving in any direction at any time by using spherical, castor or Swedish wheels (Figure 16a).
- **Tracked locomotion**: When driving through loose terrains, wheels are put on a track so that the vehicle gains stability and maneuverability (for example in tanks as shown in Figure 16b).



(a) Robot with Omnidirectional drive

(b) Tanks make use of tracked locomotion

Figure 16: Omnidirectional and tracked motion examples

In the next paragraphs, differential drive and ackermann steering will be described in more detail, since those models were used in the Nexus Robot and the PEV respectively.

DIFFERENTIAL DRIVE There are many configurations for differential drive robots as it is illustrated in Figure 17. Lets suppose that right and left wheels rotate at velocities v_r and v_l respectively, and that they are separated a distance L (Figure 18).

Figure 17: Differential drive modes: 2 motorized wheels (left) or 2 motorized wheels + caster (right)

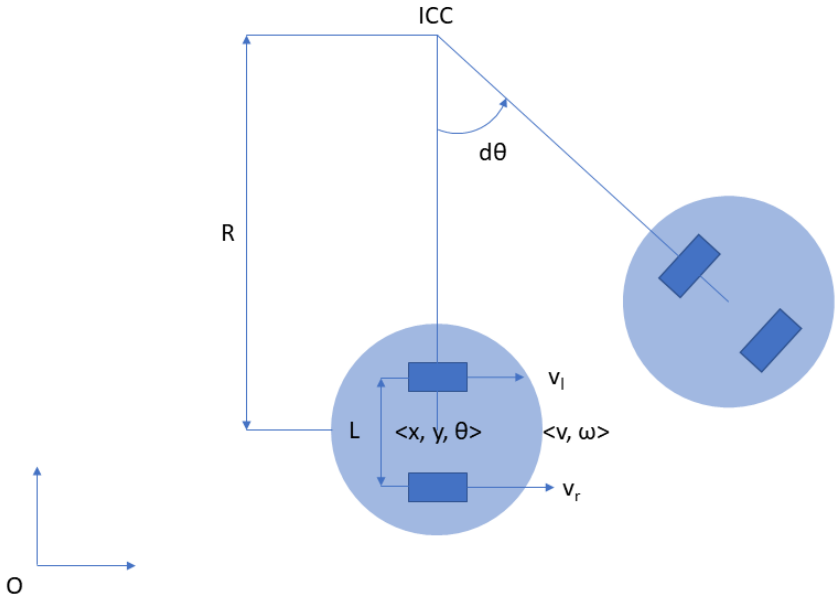


Figure 18: Instantaneous movement of a differential drive robot

The movement of the robot is modelled as a rotation at angular speed ω around the Instantaneous Center of Curvature, ICC (it is shown as R in Figure 18).

Both wheel velocities, R and ω are related through the following equations:

$$v_l = \omega \cdot (R - \frac{L}{2}) \tag{1}$$

$$v_r = \omega \cdot (R + \frac{L}{2}) \tag{2}$$

From which both R and ω can be obtained:

$$R = \frac{L}{2} \cdot \frac{v_r + v_l}{v_r - v_l} \tag{3}$$

$$\omega = \frac{v_r - v_l}{L} \tag{4}$$

From these values, the velocity of the robot, v is deduced:

$$v = \omega \cdot R = \frac{v_r + v_l}{2} \tag{5}$$

From these equations it can be deduced that differential drive vehicles are able to perform either pure translations (if v_r and v_l are equal) rotate in place (if v_r and v_l are opposite) [63].

Both linear and angular velocities are expressed in the local frames, so they are transformed to the base frame:

$${}^0\mathbf{v} = {}^0\mathbf{R}_A \cdot {}^A\mathbf{v} \tag{6}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{pmatrix} cos\,\theta & -sin\,\theta & 0 \\ sin\,\theta & cos\,\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{bmatrix} v \\ 0 \\ \omega \end{bmatrix} = \begin{bmatrix} v \cdot cos\,\theta \\ v \cdot sin\,\theta \\ \omega \end{bmatrix} \tag{7}$$

The trajectory is computed by integrating each term over time:

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} \int_0^t v(t)cos(t)dt \\ \int_0^t v(t)sin(t)dt \\ \int_0^t \omega(t)dt \end{bmatrix} = \begin{bmatrix} \frac{1}{2}\int_0^t (v_r(t) + v_l(t))cos(t)dt \\ \frac{1}{2}\int_0^t (v_r(t) + v_l(t))sin(t)dt \\ \frac{1}{L}\int_0^t (v_r(t) - v_l(t))dt \end{bmatrix} \tag{8}$$

Ackerman steering

ACKERMAN STEERING There are various modifications of the Ackerman steering vehicle as well (Figure 19). These vehicles need 2 commands, the drive speed v and the steering angle φ. Supposing a robot with Ackerman steering whose wheel distance is L, and that the movement can be modelled as a rotation around the ICC (same as differential drive).

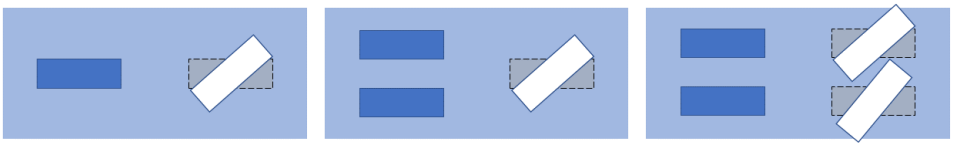


Figure 19: Ackerman drive modes: bicycle (left), tricycle (center), car (right)

Obtaining the linear speed is trvial (it is s itself), thus only the calculation of the angular speed is needed. Looking at Figure 20, it can be seen that:

$$v = \rho \cdot \omega \tag{9}$$

$$\frac{L}{\rho} = tan\,\varphi \tag{10}$$

This yields the result for ω:

$$\omega = \frac{v}{L} tan\, \varphi \tag{11}$$

The velocities in global coordinates and the translation are computed with the same equations the differential drive robots use. However, there is a fundamental difference between both locomotions: for Ackerman steering robots, it is not possible to rotate in place since that would require the backwheels to slide instead of roll [63].

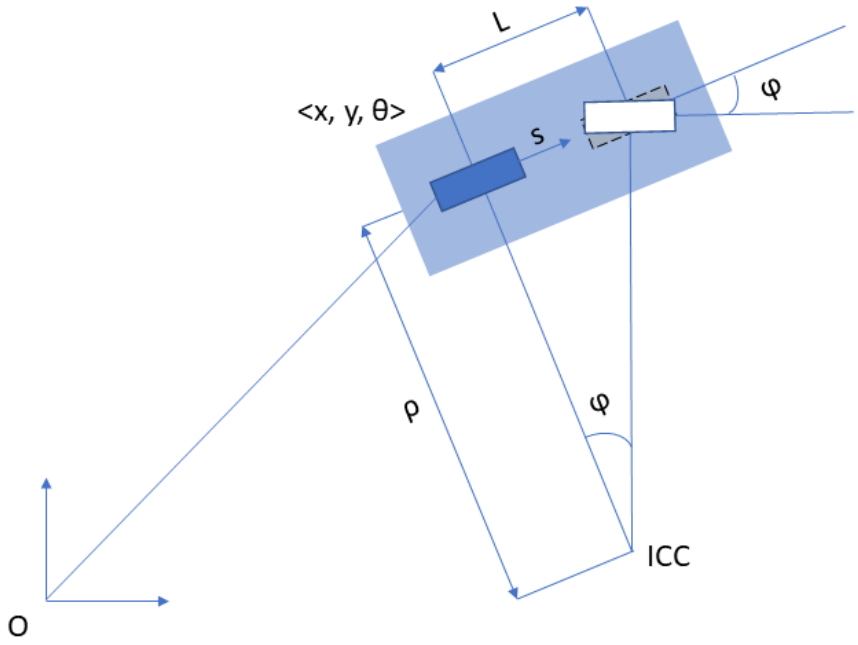


Figure 20: Instantaneous movement of an Ackerman drive robot

Perception

Apart from kinematics, acquiring information from the environment is an essential feature, not only of mobile robotics but many other systems [72]. **Sensors** are the ones in charge of that information gathering and they can be divided in [52, 51]:

- **Active/Passive**: Active sensors emit energy to the environment and measure the response (lasers, radars) and passive sensors measure external inputs (cameras, tactiles).
- **Proprioceptive/Exteroceptive**: Whether the sensor measures internal (speed, joint angles) or external (distance, sound) values.

Sensors can be categorized based on their specific function as well. In the next paragraphs some of these sensors will be described, specially the ones used on the PEV, but there is a wealth of other options that are more suited to other applications [58, 48].

Dead–reckoning and Odometry sensors

DEAD–RECKONING AND ODOMETRY SENSORS Dead–reckoning is the process of determining the current position based on previous state knowledge and the undertaken actions [48, 47]. One of the most important and widely used implementations of dead–reckoning is **odometry**, since it is accurate in the short–term, although it drifts over time [47].

The most used sensors to measure odometry are **optical encoders**, which consists of a light source passing through a punched disc producing a number of pulses per rotation. Counting the number of pulses (p) in a Δt, given that the disc has N pulses per revolution can provide the angular velocity:

$$\omega = \frac{p}{N}\frac{2\pi}{\Delta t}\left[\frac{rad}{s}\right] \tag{12}$$

If a second channel is added and shifted 90 degrees, rotation direction can be measured as well as position determination becomes more precise (Figure 21).

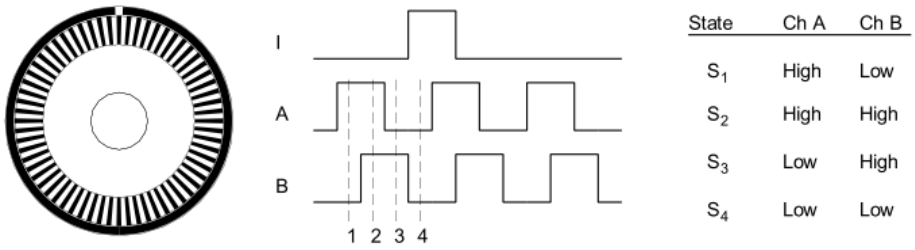


Figure 21: Rotary encoder working principle

Heading sensors

HEADING SENSORS Heading sensors can help compensate the errors produced in odometry by small differences in orientation [48]. This type of sensors are also called **Inertial sensors** and there are mainly 2 types [47]:

- **Accelerometers**: They measure acceleration in one or more axes. However, they perform poorly on many mobile robot applications if used alone.
- **Gyroscopes**: These sensors can measure orientation on 3 axes and are of a notable importance since they correct the odometry measurements when there are small orientation changes.

Recently, another type of sensor has arisen in the field of mobile robotics called **Inertial Measurment Unit (IMU)**, which combines both accelerometer and gyroscope (sometimes adding a magnetometer for heading).

Active/Passive beacons

ACTIVE/PASSIVE BEACONS The principle behind this technology has been used for centuries and it is to determine the absolute position of the robot based on the distance to landmarks or objects whose location is well known [72]. This mapping of the position of the robot is done via **trilateration** or **triangulation** [47].

Here 2 different systems can be defined: **Ground–based beacons** and **Global Positioning System (GPS)**. They vary in that the former uses features or infrastructures located on the ground whereas the latter makes use of a network of satellites specifically built for that purpose (Figure 22).

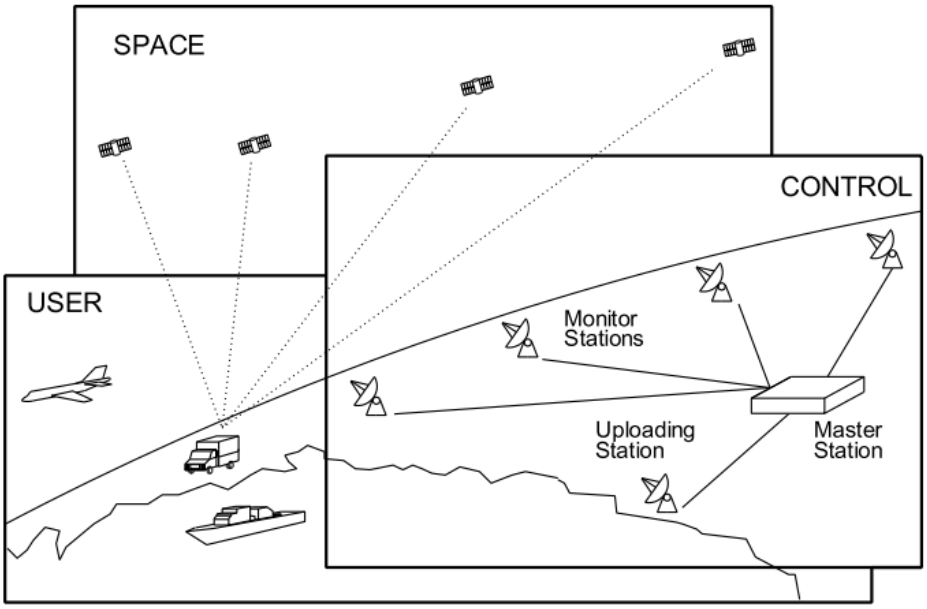


Figure 22: GPS network overview

One of the issues these technologie has is that even though it works satisfactorily in outdoor 'open' environments, it is not as suitable in indoor or more 'closed' regions (urban areas with large buildings) [48, 47].

Active ranging sensors

ACTIVE RANGING SENSORS This sensors are mostly used for map building and localization. Among all the varieties, the **Time–of–flight** sensors are the most utilized. These devices send a wave (sound or electromagnetic) and knowing the speed at which they operate, the distance travelled is calculated:

$$d = \frac{v \cdot t}{2} \tag{13}$$

2 types of time-of-flight sensors will be described:

- **Ultrasonic**: This kind of sensor sends a packet pressure waves to determine distance of objects. However, their range goes from 12 cm to 5 m making them unfit for large environments.
- **Laser rangefinders or LIDARs**: It improves the ultrasonic sensor by using infrared light, and it normally adds a rotating mirror so it can cover a wider area. The range of these sensors goes up to 300 m[1]. Generally, LiDAR output is a detailed 2D or 3D pointcloud of the surroundings, which often is better than camera+radar based systems. That is why is considered to be the single most important sensor in

1 https://velodynelidar.com/vls-128.html

Autonomous Vehicles [57] and it is also one of the most expensive components.

VISION–BASED SENSORS

VISION–BASED SENSORS Vision is one of the most powerful senses humans have. It provides with a great deal of information about the surrounding environments and currently there is a lot of effort into building machines with similar capabilities. Recent advances in machine–learning (and deep–learning in particular) [64], combined with the existence of many frameworks such as Tensorflow have improved tasks that previously where extremely difficult, such as object detection, scene understanding or lane detection.

The most used vision–based sensors are cameras, among which 2 types can be diferenced:

- **Monocular cameras**: These are well known devices, that have been available in the market for decades.
- **Stereo cameras**: Essentially, the concept is to combine the input images of 2 cameras that watch the scene from different perspectives, and by comparing points that appear on both (conjugate pair), depth information can be obtained (Figure 23).

Figure 23: ZED stereo camera

Localization

The problem of localization (which is related to the problem of SLAM that will be discussed in Chapter 3) is to determine the position of the robot within the environment [72, 96].

Recalling the control paradigms in Figures 14 and 13, it is not difficult to see that the former needs a map–based localization system whilst the latter does not. However, the reactive paradigm is not robust to changes in the environment, since it would require to change the rules every time it changed. The classic control one is more scalable beause just by giving the robot another map, it could start navigating in a shorter amount of time.

When a robot is localized it means that the pose of the robot $\mathbf{x_t} = (x\ y\ \theta)$ is known with respect to the global frame. Nevertheless, pose cannot be sensed, it must be inferred from other sources and sensors are not noise–free, thus making the problem of localization a difficult one [96].

Map representation

MAP REPRESENTATION The quality of the map is a critical component for a robust localization, therefore much attention has to be paid when choosing a representation [72].

There are mainly 2 manners of representing maps:

- **Continuous representation**: They contain the exact description of their environment. The main advantage is their accuracy at a higher computational cost.
- **Decomposed representation**: These type of maps are a higher level abstraction of the area, which results in loss of accuracy but may be useful if the jey features are preserved. One of the most common forms are **occupancy grid maps**, where the map is discretized an assigned a value if occupied or free.

Dimensions of localization

DIMENSIONS OF LOCALIZATION Localization algorithms have a number of divisions [96]:

- **Global/Local**: In global localization the initial position is unknown, whereas in the local one it is known to be confined in a certain region.
- **Static/Dynamic**: The former are those whose objects remain still and the latter have features that change their position over time.
- **Passive/Active**: In passive localization, the algorithm is limited to observation. In active localization, the algorithm can affect the robot motion to facilitate it.

Localization approaches

LOCALIZATION APPROACHES With regards to the specific algorithms for localization, the main approach is **probabilistic localization** [96]. In these branch are included Markov, Extended and Unscented Kalman Filter (EKF and UKF), Grid and MonteCarlo (MCL) localizations. Figure 24 shows a working example of a particle filter based localization algorithm, similar to MCL.

(a) Not localized (b) Localized

Figure 24: How probabilistic localization works

Those are the most popular algorithms though not unique. Other solutions are landmark–based navigation or route–based navigation.

Planning and Navigation

Recalling Figure 14, it can be inferred that motion planning is the branch that develops algorithms to translate high–level information such maps and human commands into low–level instructions to the controllers [63].

Planning and Navigation has 2 goals: reach the desired/commanded location as quickly as possible and avoid obstacles/collisions [53]. To achieve that, this task is usually divided in 2 layers as shown in Figure 25.

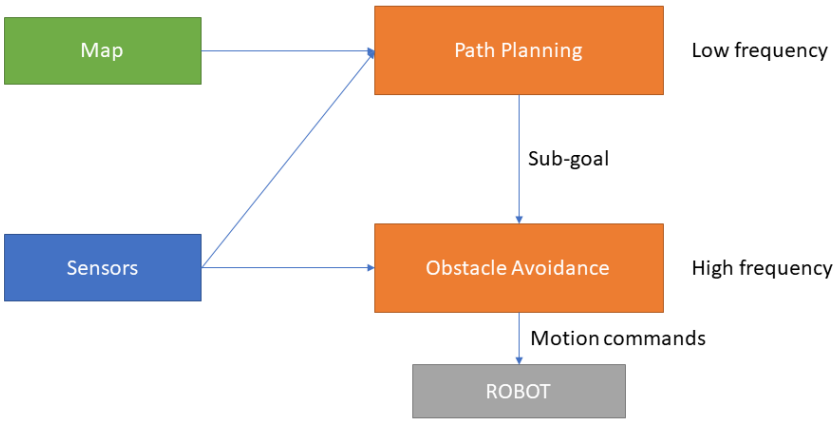


Figure 25: Motion planning workflow

Path Planning

PATH PLANNING Path planning can be considered as the long term strategy to reach the goal [72]. The objective of this layer is to identify a path that moves to the desired location without hitting any obstacles. For that purpose, the following information is required: The **start** pose of the robot, the **goal** pose, the geometrical description of the **robot** and the representation of the **environment**.

The field of path planning was deeply studied for industrial manipulators, and in fact they are more complex cases than differential drive or ackerman steering robots.

Even though path planning is a real–world problem, when approaching the problem it is often represented in the **configuration space**, where every state can be represented with k values $q_1, q_2 \dots q_k$, being k the number of degrees of freedom. That space is then discretized and the algorithm is applied. Some of the most common algorithms for path planning are [63, 53]:

- **Search algorithms**: Such as Dijkstra and A*.
- **Road map planning**: Voronoi diagrams are a very common variety.
- **Cell decomposition**.
- **Rapidly Exploring Random Trees (RRT)**
- **Markov Decision Processes (MDP)**.

OBSTACLE AVOIDANCE Whereas path planning is focused on searching global optimal solutions, obstacle avoidance searches on the local space and modifies the trajectory based on the obstacles detected by sensors. There are many existing algorithms for obstacle avoidance but one of the most utilized (even though there exist more optimal solutions nowadays) is the Dynamic Window Approach (DWA) [59] (Figure 26).

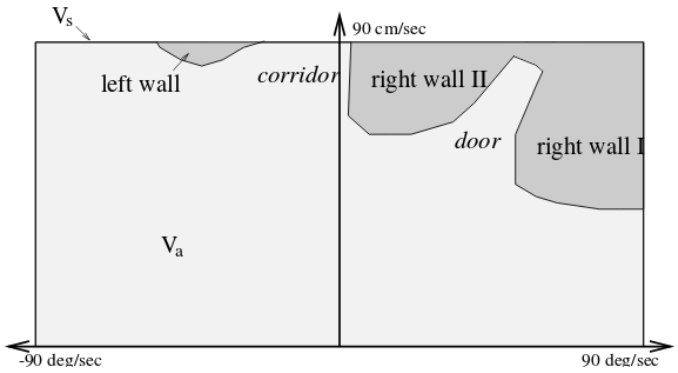


Figure 26: DWA velocity search space

This algorithm searches the admissible pair of velocities $< v, \omega >$ based on the dynamic constraints and surrounding obstacles that maximize the function:

$$G(v,\omega)) = \sigma(\alpha \cdot heading(v,\omega) + \beta \cdot dist(v,\omega) + \gamma \cdot vel(v,\omega)) \tag{14}$$

where *heading* measures the distance to the goal (favoring movements towards it), *dist* the distance to the closest obstacle (favoring clearance) and *vel* the velocity of the robot (favoring higher speeds).

ROBOT OPERATING SYSTEM (ROS)

With the rise of robotics, a notable amount of software packages has proliferated these years, either for industrial or research purposes:

- **Industrial robotics software**: Almost every major robot manufacturer ships with their own programming language as it is shown on Table 2:
- **Research software**: Some of the libraries used are: Peter Corke's Robotic Toolbox for Matlab [54], Microsoft Robotics Developer Studio, Robotics Library and many others.

These are not the only items available, in fact the field of robotics is too broad, and there is no perfect solution that covers all the aspects of the it. However, there is one tool that is very popular among researchers and companies around the world and that is called Robot Operating System (ROS).

Company Name	Language Name
ABB	RAPID
Kuka	KRL (Kuka Robot Language)
Comau	PDL2
Yaskawa	INFORM
Fanuc	Karel

Table 2: Industrial robots' programming languages

Why and what is it?

Every year, robotics grows in size and scaling software becomes a daunting challenge [69]. This is because robotics gathers the fields of Mechanical and Electrical Engineering with Computer Science, thus software must contain driver level software up to perception and higher levels of abstraction.

Since very few researchers have sufficient knowledge on all the aspects mentioned above, code reuse is a must of any robotics software.

Therefore, in 2008, researchers from Stanford University, University of Southern California (UCSC) and Willow Garage began developing ROS as a continuation of previous software packages such as Stanford's STAIR [68] and Willow Garage's Personal Robots Program [75]. ROS continued to grow and in 2014, the Open-Source Robotics Foundation (OSRF) took charge of the development and maintenance of the ecosystem[2].

ROS was built with the following design goals in mind [69, 70] (Figure Figure 27 shows a representation of those concepts):

Figure 27: Capabilities of ROS

- **Peer to peer**: ROS consists on different processes that run in a peer–to-peer network exchanging information between them. This way several machines can be attached, performing each one different tasks. For instance, a robot can have several onboard machines sensing and sending actuation commands via ethernet, while offboard machines connected to the previous ones wirelessly can take charge of 'heavier' tasks such as SLAM or Computer Vision (Figure 28).

Figure 28: Schematic of the networking capabilities of ROS

2 For more on the history of ROS: http://www.ros.org/history/

- **Tools based**: Instead of an enormous standalone application, ROS is composed of several tools with various functionalities, such as navigating the structure, visualizing sensor inputs, compiling, and many others.
- **Multi-lingual**: As every programming language excels at different tasks, ROS provides client libraries for a great variety of languages [3], being the main ones C++, Python and LISP
- **Thin**: ROS conventions encourage developers to create standalone algorithms and drivers in order to promote code reuse. Those software packages can then be wrapped around ROS and be used on the netowork.
- **Free and Open-Source**: ROS full source is publicly available and anyone can make contributions to its core. This core is licensed under BSD, thus allowing for either commercial or non commercial use.Individual modules or components can have their own licensing.

In a more formal definition, ROS is an open-source framework that includes a collection of tools, libraries and conventions to allow for robotic system development [70].

Structure of ROS

Basic Concepts

BASIC CONCEPTS The structure of ROS is composed of **nodes**, **messages**, **topics** and **services** [69, 67]:

- **Nodes**: They are the equivalent of software modules and perform one specific computation. When running an application built on ROS, it will very often be composed of several nodes communicating with each other.
- **Messages**: They are data structures that nodes use to communicate. Messages can be composed of basic types or other messages as shown on Listing 2.1, where a first file `point.msg` is created to contain 2 variables referring to the position of an object, and the second one `points.msg` will store a vector of those points:

Listing 2.1: Example message files

```
# First file: point.msg
float32 x
float32 y
# Second file: pointarray.msg
std_msgs/header header
point[] points
```

3 http://wiki.ros.org/Client Libraries

- **Topics**: They act as the containers for the messages. A single node can *publish* messages to a variety of topics, so that other nodes can *subscribe* to one or more of those topics and perform their computations. Topic names must always start with "/" (i. e."/topic_name").
- **Services**: Topics can be read/written by any node and sometimes a more bidirectional form of communication is wanted between 2 nodes, in a similar fashion as web services employ request and response documents. That is why services exist (Listing 2.2).

Listing 2.2: Example service file

```
# File: myService.srv
# Here goes the request
int32 x
int32 y
---
# Here goes the response
bool success
int32 multiplication
```

When many nodes and topics are running at the same time, debugging might become difficult. That is why ROS provides a tool called `rqt_graph`, in order to visualize all dependencies among processes (Figure 29).

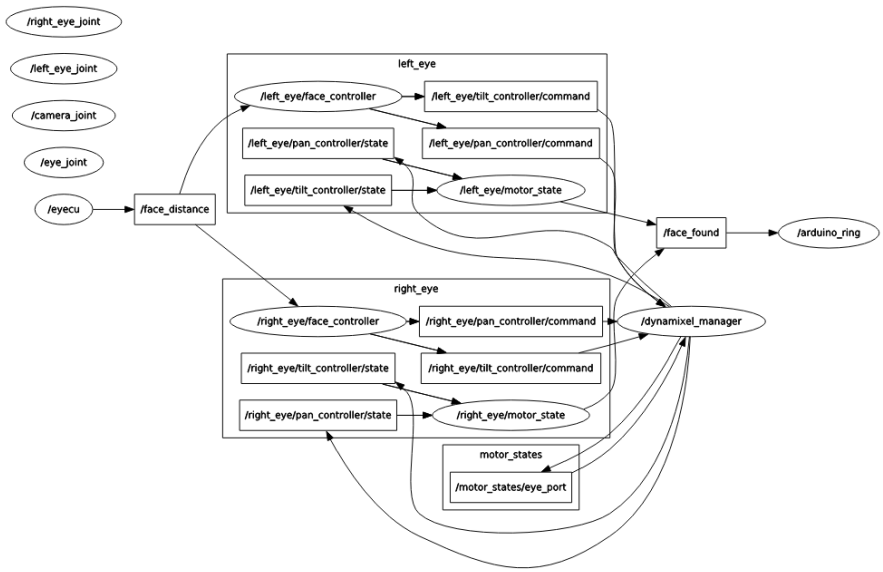


Figure 29: ROS graph example. The flow is represented as: nodes (circles) send messages (arrows) to topics (boxes)

BUILD SYSTEM

BUILD SYSTEM ROS is packaged with several tools to produce libraries, executables and scripts in both C++ and Python (and other languages with

a little tweaking) [70]. The key concepts here are: **packages**, **workspaces** and **catkin**:

- **Packages**: In ROS, all software, data and documentation is organized into packages. Packages are organized in folders (source code in `src/`, message files in `msg/`, launch files in `launch/`) and need to have 2 files, `CMakeLists.txt` and `package.xml` with instructions for the build tools.
- **Workspaces**: Packages are bundled in directories called workspaces, that by convention are called `exampleworkspace_ws`. Figure 30 shows the structure of workspaces, where packages must be located in the `src/` folder, and the `install/` and `devel/` directories contain the executables and libraries.

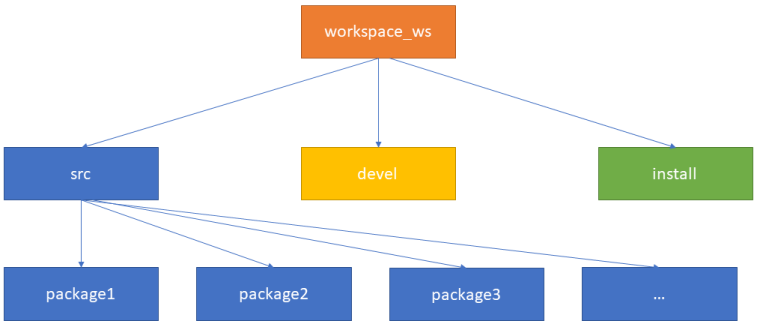


Figure 30: Workspace structure

- **Catkin**: It is the offitial build system for ROS. More specifically, it defines some macros and Python scripts to extend the capabilities of CMake, so that building ROS packages is easier [4].

In order to create the executables, ROS provides the tool `catkin_make` and it is used as shown in Listing 2.3:

Listing 2.3: Use of catkin_make

```
# Go to workspace
user@host:~$ cd workspace_ws
# Compile packages
user@host:~/wokspace_ws$ catkin_make
# If no errors appear
user@host:~/wokspace_ws$ source devel/setup.bash
# This way other ROS tools can find the executables
```

RUNNING THE SYSTEM

RUNNING THE SYSTEM To start any ROS session, there are 3 major components: **roscore**, **rosrun** and **roslaunch**:

4 More information on: http://wiki.ros.org/catkin/conceptual_overview

- **roscore**: This command launches the ROS master, which is in charge of providing nodes with information to establish communication with other nodes. If the core is not running, nodes cannot find each other and ROS will not start.
- **rosrun**: It searches for the specified node in the specified package and starts it: `rosrun <package> <executable> <arguments>`.
- **roslaunch**: When the number of nodes and parameters grows in size, it is possible to group several nodes in an `.xml` file and run them with the `roslaunch` command (Listing 2.4).

Listing 2.4: Example of launch file

```
<launch>
<!-- Node to launch necessary files from Jetson -->

  <!-- Launch hackbike serial -->
  <node pkg="hackbike" type="hackbike_serial" name="
     hackbike_serial_node" output="screen"/>

  <arg name="mode" default="pedestrian"/>

  <!-- Launch data sender node -->
  <param name="mode" value="$(arg mode)"/>
  <node pkg="panasonic" type="send_data_to_bike.py"
     name="data_sender_node" output="screen"/>

  <!-- Start tensorflow node -->
  <arg name="tensorflow" default="false"/>
  <group if="$(arg tensorflow)">
    <include file="$(find panasonic)/launch/
       start_tensorflow.launch"/>
  </group>

  <include file="$(find panasonic)/launch/
     start_arduino.launch"/>

</launch>
```

VISUALIZATION AND SIMULATION

VISUALIZATION AND SIMULATION It has already been stated that ROS is a tools-based platform, and 2 of those main tools are **RViz** and **Gazebo**:

- **RViz**: It is the standard visualization tool for ROS, capable of rendering 3D models, sensed images and pointclouds [60]. It is widely used for controlling robotic systems based on ROS (Figure 31)
- **Gazebo**: It is an open–source robot simulator with support for multiple robots in complex environments [62]. It is a different platform from ROS, but they can be bundled together (Figure 32).

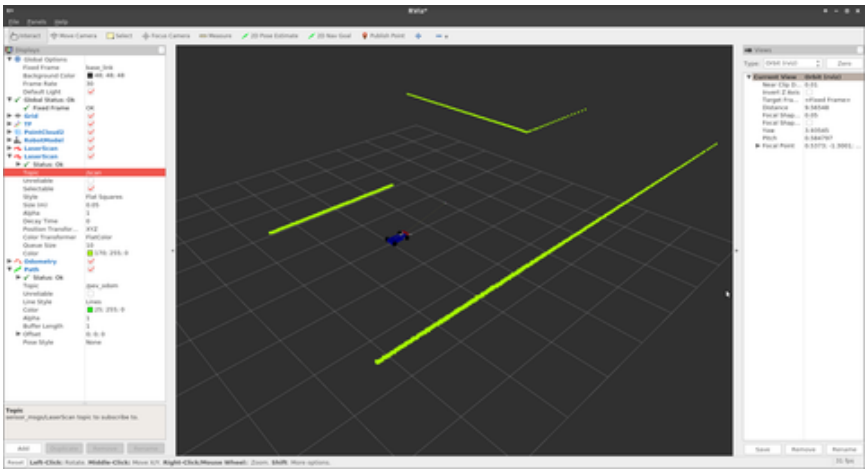


Figure 31: RViz showing the points received by a LiDAR

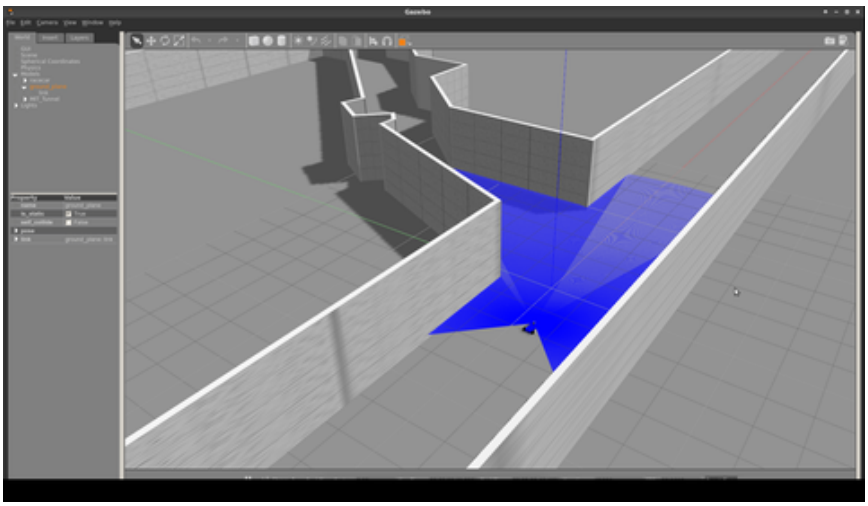


Figure 32: Small robot in Gazebo

Navigation stack

ROS provides a collection of packages that allow for 2D indoor navigation called Navigation Stack [65]. Basically, this framework reads odometry, laser data and transforms and outputs velocity commands to the controllers. Figure 33 shows a conceptual map of the Navigation Stack.

The main nodes provided by the stack are:

- **Global costmap**: It reads the information of a static map and creates a 2D layer with the obstacles detected in it. This layer will remain mostly unchanged.
- **Local costmap**: It creates a smaller 2D costmap with the available laser data that gets updated frequently in order to account for dynamic obstacles.

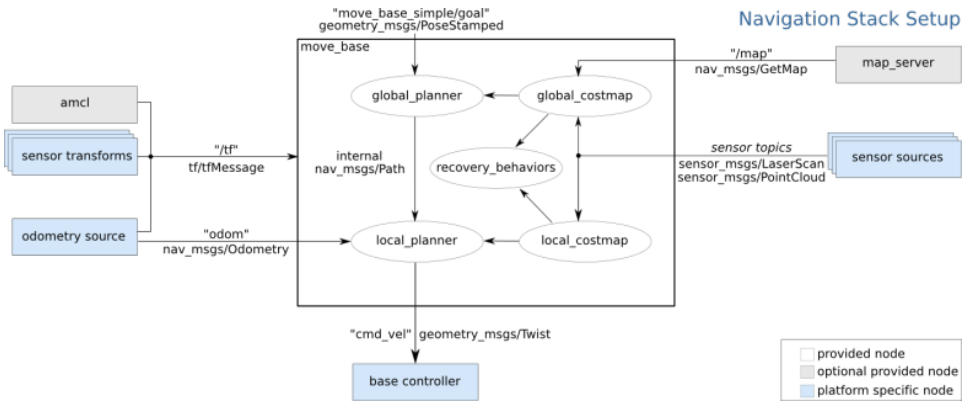


Figure 33: Navigation Stack overview

- **Global planner**: With the information from the global costmap and a desired goal, it provides a high-level plan for the robot to follow. It uses algorithms like Dijkstra or A* and it does not take into account any kinematic or dynamic restrictions.
- **Local planner**: It reads the global plan and the local costmap and outputs velocity commands that follow the global plan as closely as possible while avoiding obstacles. This planner takes into account the constraints the robot might have.

The navigation stack is a useful tool but has its limitations as well. First, it is optimized for 2D navigation which is less accurate and not particularly suitable for large environments. Second, it was designed for indoor navigation not for outdoor scenarios. And third, it only considers differential and holonomic drive robots.

However, it is possible to modify some of the nodes provided to extend the functionalities of the stack, as it has been done with the PEV.

3

SIMULTANEOUS LOCALIZATION AND MAPPING (SLAM)

Simultaneous Localization and Mapping (**SLAM**) or Concurrent Mapping and Localization (**CML**) is one of the most fundamental problems in robotics [81, 96, 97]. It tackles the task of placing a robot in an unknown environment and being able to build a map while keeping track of the robot's own position on that map.

Since SLAM is one of the core features of mobile robotics, its applications range in the same field such as ground, indoor, underwater or air vehicles (Figure 34).

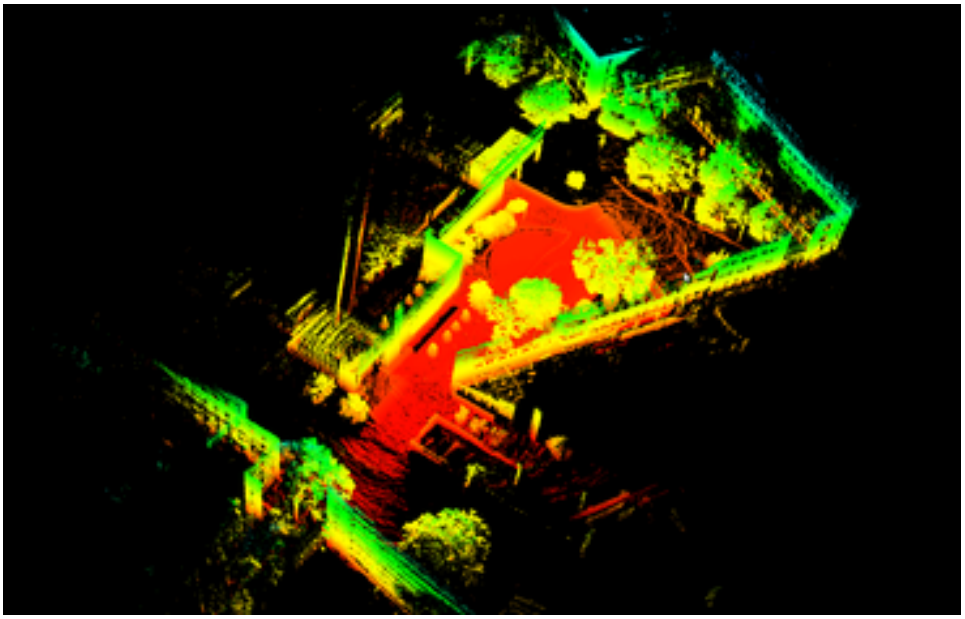


Figure 34: 3D example of SLAM techniques on the PEV

For its nature, SLAM is considered to be one of the toughest challenges of mobile robotics [80, 96] and it is often referred as a **chicken–and–egg** problem:

- In order for a robot to correct errors from odometry and localize in the environment it needs a map.
- In order to build a map, the robot needs to know its position and orientation with respect to the environment.

This interdependence between both problems is why they have to be approached at the same time, thus being named 'Simultaneous'.

SLAM is seen as the 'holy grail' of AVs and mobile robots in general [81], since once a robust method of building rich maps is found, it will allow for fully autonomous vehicles.

In fact, the SLAM problem has been solved in the theoretical plane, where many equally valid variants can be found; the problem arises when those algorithms are applied in real–life systems.

SLAM FUNDAMENTALS

Probabilistic Robotics

The approach to SLAM is generally done in terms of probability instead of using single "best guess" values. This manner of tackling SLAM (and many other fields in mobile robotics) results to be very useful since neither sensors nor actuators are noise–free. By integrating those uncertainties into the models, controls can be made more robust, thus improving the performance of the robot.

The core of probabilistic robotics is **state estimation**, that is estimating environment and internal variables using sensor data.

Denoting time with letter t, the state and environmental variables can be defined as [96, 97]:

- **State**: State is usually denoted with letter x and is composed of position and orientation (relative to global frame), more commonly called **pose**. In flat navigation, the state comprises the tuple $(x\ y\ \theta)$. Thus, the state at time t is denoted as x_t, and the collection of states from time 0 to t, namely the **path**, is:

$$X_t = \{x_0, x_1, x_2, \ldots, x_t\} \tag{15}$$

  Moreover, states can be *complete* or *incomplete*. A state is called *complete* when the prior knowledge of state, control and perception does not add more information. However, in real–life situations, not all aspects of the robot are known, being that called *incomplete*.

- **Control actions**: Control actions change the state of the system (e. g., manipulation, forward movement) and data associated to these actions contain information on how the state changes. One of the most common sources of data is *odometry* (Section 2.2.2). Even though the sources of odometry are sensors, they measure the change of state, being equally valid for the purpose.

  Control data is usually denoted with u, and u_t denotes the action that brings the system from state x_{t-1} to x_t. Actions are collected in the vector:

$$U_t = \{u_0, u_1, u_2, \ldots, u_t\} \tag{16}$$

- **Environment measurements**: Sensors such as lidars or camera perform *observations* to gain knowledge of the state of the system. Measurement data is denoted with z and z_t corresponds to a specific measurement at time t. Z_t will denote the collection of all measurements from 0 o to t:

$$Z_t = \{z_0, z_1, z_2, \ldots, z_t\} \tag{17}$$

Once the main variables have been defined, they are usually gathered into a probability distribution called *belief* or *state of knowledge*, that represents the pose at time t taking into account the previous data:

$$bel(x_t) = p(x_t|z_{1:t}, u_{1:t}) \tag{18}$$

There is a variant of the belief, called prediction, that estimates the pose without taking into account the latest sensor measurement and that is denoted $\overline{bel}(x_t)$:

$$\overline{bel}(x_t) = p(x_t|z_{1:t-1}, u_{1:t}) \tag{19}$$

Having defined these variables, now the Bayes filter can be deduced.

Bayes filter

The bayes filter is a technique to peform state estimation, based on previous beliefs and sensor measurements. It is of essential importance in Probabilistic Robotics, since it is the basis for practically all other approaches in the field [80, 81].

The goal is to determine the law that provides the probability distribution of the belief at time t, $bel(x_t) = p(x_t|z_{1:t}, u_{1:t})$. For that we start with the combined probability $p(x_t, z_{1:t}, u_{1:t})$ and apply the **Bayes rule** (hence the name):

$$\begin{aligned} p(x_t, z_{1:t}, u_{1:t}) &= p(z_{1:t-1}, u_{1:t}) \cdot p(z_t|z_{1:t-1}, u_{1:t}) \cdot p(x_t|z_{1:t}, u_{1:t}) \\ &= p(z_{1:t-1}, u_{1:t}) \cdot p(x_t|z_{1:t-1}, u_{1:t}) \cdot p(z_t|x_t, z_{1:t-1}, u_t) \end{aligned} \tag{20}$$

Considering that $p(z_{1:t-1}, u_{1:t})$ appears on both sides, it can be cancelled out. Then, reordering the elements of the equation:

$$\begin{aligned} bel(x_t) = p(x_t|z_{1:t}, u_{1:t}) &= \frac{p(z_t|x_t, z_{1:t-1}, u_t) \cdot p(x_t|z_{1:t-1}, u_{1:t})}{p(z_t|z_{1:t-1}, u_{1:t})} \\ &= \eta \cdot p(z_t|x_t, z_{1:t-1}, u_t) \cdot \overline{bel}(x_t) \end{aligned} \tag{21}$$

Where parameter η is called *normalizer*. If it is assumed that the state is complete, then the measurement z_t does not depend on the previous measurements or controls (this is called **Markov assumption**), thus it can be written:

$$p(z_t|x_t, z_{1:t-1}, u_t) = p(z_t|x_t) \tag{22}$$

As for the prediction, $\overline{bel}(x_t)$, applying the *Theorem of Total Probability* the following is obtained:

$$\begin{aligned}\overline{bel}(x_t) &= p(x_t|z_{1:t-1}, u_{1:t}) \\ &= \int p(x_t|x_{t-1}, z_{1:t-1}, u_{1:t}) \cdot p(x_{t-1}|z_{1:t-1}, u_{1:t}) \cdot \mathrm{d}x_{t-1}\end{aligned} \tag{23}$$

Applying again the Markov assumption to $p(x_t|x_{t-1}, z_{1:t-1}, u_{1:t})$, the resulting probability distribution is (u_t is not cancelled since it carries information on the change from x_{t-1} to x_t):

$$p(x_t|x_{t-1}, z_{1:t-1}, u_{1:t}) = p(x_t|x_{t-1}, u_t) \tag{24}$$

As for the second term, u_t can be taken away, as it does not predict state x_{t-1}:

$$p(x_{t-1}|z_{1:t-1}, u_{1:t}) = p(x_{t-1}|z_{1:t-1}, u_{1:t-1}) = bel(x_{t-1}) \tag{25}$$

After these changes, Equation 21 is converted to:

$$\begin{aligned}bel(x_t) &= \eta \cdot p(z_t|x_t) \cdot \overline{bel}(x_t) \\ &= \eta \cdot p(z_t|x_t) \cdot \int p(x_t|x_{t-1}, u_t) \cdot bel(x_{t-1}) \cdot \mathrm{d}x_{t-1}\end{aligned} \tag{26}$$

There are 2 important components in Equation 26, namely the *measurement probability* $p(z_t|x_t)$, and the *state transition probability* $p(x_t|x_{t-1}, u_t)$. In [96], *observation models* and *motion models* are provided, that allow to describe those probability distributions more accurately (Figure 35).

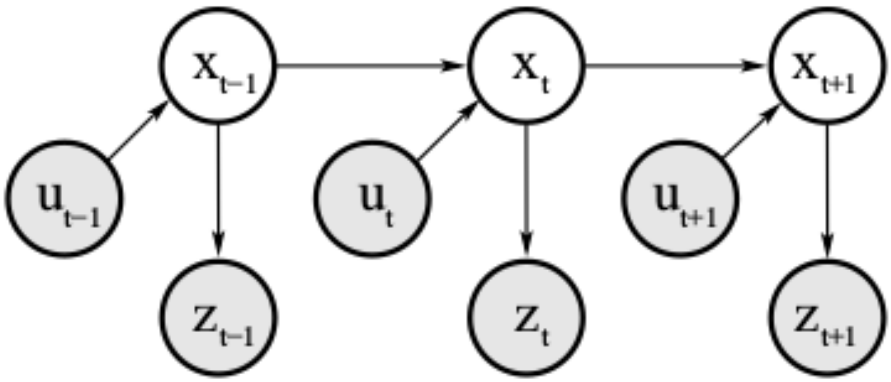


Figure 35: Schematic of state determination using Bayes filter

Probabilistic robotics in SLAM

When tackling SLAM, there is another variable that needs to be defined and it corresponds to the map features m_i, being m the set of the n features that compose the **map**:

$$m = \{m_1, m_2, \ldots, m_n\} \tag{27}$$

It is possible to add the map to the Bayes filter (Equation 26), since it can be considered as an extension of the state of the system. Therefore the bayes filter becomes:

$$p(x_t, m|z_{1:t}, u_{1:t}) = \eta \cdot p(z_t|x_t, m) \cdot p(x_t, m|z_{1:t-1}, u_{1:t}) \tag{28}$$

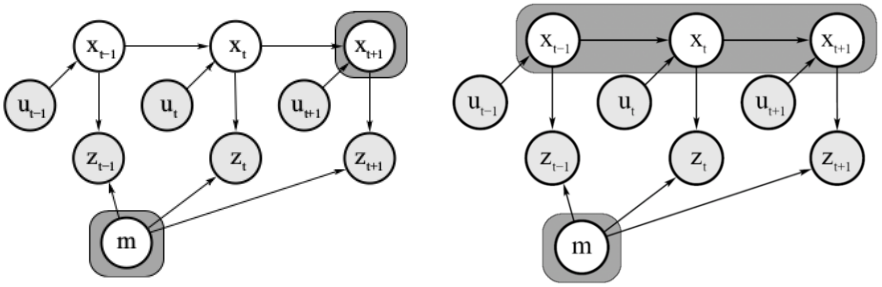
In the probabilistic realm, there are 2 main forms of SLAM (Figure 36):

- **Online SLAM**: This approach only seeks to recover the most recent pose and map as it is stated in Equation 28. The previous poses and measurements get discarded in this approach.
- **Full SLAM**: In this case, instead of the current position, the whole path is estimated, yielding the following equation:

  $$p(x_{0:t}, m|z_{1:t}, u_{1:t}) \tag{29}$$

  It is possible to derive the online SLAM from the full SLAM, by integrating all the poses throughout time [80]:

  $$p(x_t, m|z_{1:t}, u_{1:t}) = \int \int \cdots \int p(x_{1:t}, m|z_{1:t}, u_{1:t}) \mathrm{d}x_1 \mathrm{d}x_2 \ldots \mathrm{d}x_{t-1} \tag{30}$$



(a) Online SLAM (b) Full SLAM

Figure 36: Schematic of the 2 SLAM approaches

SLAM dimensions

The most usual differences among the SLAM algorithms are defined in the following paragraphs [97]:

- **Volumetric/Feature–based**: In volumetric SLAM, the map is sampled at realistic resolutions, thus the computational complexity grows notably. In feature–based maps, there are algorithms that process sensor readings and extract the key landmarks of the environment. The latter group is more data efficient, but it may lose some important sensor information.
- **Topological/Metric**: Topological maps are those that only characterize the environment with basic relationships, whereas metric ones provide more accurate descriptions of the surroundings (Figure 37).

(a) Topological (b) Metric

Figure 37: 2 maps of Boston: Subway (Topological) vs Satellite (Metric)

- **Known/Unknown correspondence**: Correspondence refers to matching sensed features from different timestamps. Some SLAM algorithms assume the identity of landmarks is known while others do not. The problem of determining those correspondences is known as *data association* and it is one of the toughest challenges in SLAM.
- **Static/Dynamic**: The same way as it was described in localization (Section 2.2.3), environments can be static if the environment remains the same over time, or dynamic if there are moving parts. Most of SLAM algorithms suppose static worlds.
- **Small/Large uncertainty**: Algorithms that allow for small uncertainties require that robots move along simple paths back and forth. If the path is more complex and there are multiple paths to return to the starting point, it will cause more uncertainty. In these cases the ability of the algorithm to perform **loop closure** correctly is crutial. Loop closure is the task of matching a previously visirted area with the current sensor readings [90]. Today, most SLAM algorithms implement more or less sophisticated loop–closing modules.
- **Active/Passive**: As in localization, when the SLAM algorithm is passive, it just observes to the movement of the robots and estimates the location and the map. In active SLAM, the algorithm itself takes control of the vehicle, resulting in more accurate maps. Nevertheless, the majority of SLAM approaches belong to the passive field.
- **Single/Multi–robot**: Most approaches use the single–robot assumption, though the multi–robot exploration is gaining adepts in the last years. In the latter form, robots are given relative positions to each other and a form of communication among them.

SLAM VARIETIES

Having set the basics of the SLAM problem, it is time to describe the 3 main SLAM paradigms from which most of the algorithms stem: **Kalman filter SLAM**, **Particle filter SLAM** and **GraphSLAM**.

Kalman Filter SLAM

Kalman filter SLAM (more specifically Extended Kalman Filter (EKF)) is one of the earliest versions of SLAM [96, 97]. As its name suggests it is based on Kalman filter approaches, which are methods devised by R.E. Kalman in the 1960s to solve the discrete linear filtering problem [79].

Kalman filter and its siblings assume that states can be modelled by gaussian distributions [93]:

$$p(x) = det(2\pi\Sigma)^{-\frac{1}{2}} \cdot exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)) \tag{31}$$

Where x is the state vector, μ is the mean vector and Σ is the covariance matrix.

Kalman filter then models the state prediction and the measurement as:

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t \tag{32}$$
$$z_t = C_t x_t + \delta_t \tag{33}$$

where A_t is a $(n \times n)$ matrix that describes the evolution from $t-1$ to t without controls, B_t is a $(n \times l)$ matrix that describes the changes induced by control u_t, C_t a $(k \times n)$ matrix that maps the state to the observations and ϵ_t, δ_t are zero–mean random variables representing the noise in controls and measurements.

The filter described in Equation 33 is only valid for linear systems, which is not the case in many real life environments [79, 92]. Instead, the EKF supposes non linear functions that map the prediction and measurement:

$$x_t = g(x_{t-1}, u_t) + \epsilon_t \tag{34}$$
$$z_t = h(x_t) + \delta_t \tag{35}$$

Functions g and h are then linearized via Taylor expansion around the mean of x_{t-1} to calculate the posterior:

$$g(x_{t-1}, u_t) \approx g(\mu_{t-1}, u_t) + \frac{\partial g(\mu_{t-1}, u_t)}{\partial x_{t-1}}(x_{t-1} - \mu_{t-1}) \tag{36}$$
$$h(x_t) \approx h(\overline{\mu}_t) + \frac{\partial h(\overline{\mu}_t)}{\partial x_t}(x_t - \overline{\mu}_t) \tag{37}$$

When it comes to EKF SLAM, maps are *feature based*, that is, composed of a collection of landmarks. The robot state estimation is the *combined* state of x_t and the coordinates of the landmarks, m:

$$\begin{aligned} y_t &= \begin{pmatrix} x_t \\ m \end{pmatrix} \\ &= (x\ y\ \theta\ m_{1,x}\ m_{1,y}\ m_{2,x}\ m_{2,y}\ \dots\ m_{N,x}\ m_{N,y})^T \end{aligned} \tag{38}$$

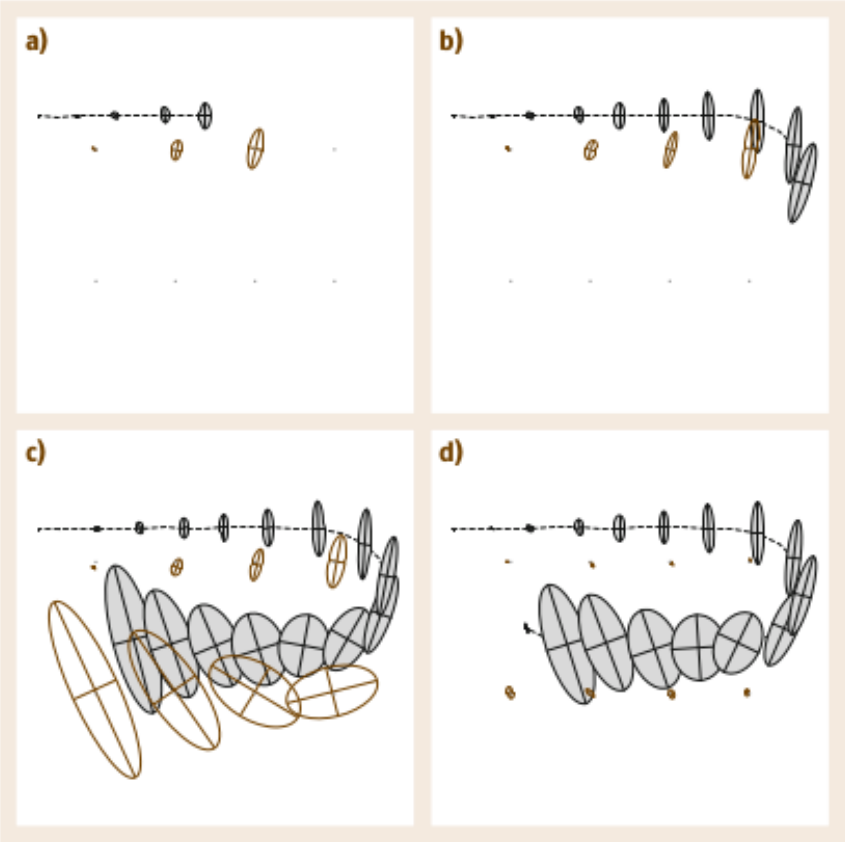


Figure 38: EKF SLAM example

where μ_t is a $2N+3$ size vector representing the mean at time t and Σ_t is a $2N+3$ matrix that describes the covariance matrix.

The filter cycle of the algorithm is the following [92]: 1. State prediction, 2. Measurement prediction, 3. Measurement, 4. Data association and 5. State update.

EKF SLAM has been applied successfully in many environments from underwater to aerial spaces. Figure 38 shows an example of the application of EKF SLAM using 8 landmarks. From a) to c) the robot pose uncertainty grows until it detects the first feature again and the pose uncertainty decreases.

However, it has its drawbacks. Due to the nature of the covariance matrix, every update step takes a quadratic time, thus making the feature number low (less than 1000 features) [96].

Because the number of features needs to remain low, *data association* tends to become more difficult. Data association is matching the most likely measurement to a specific landmark. EKF is generally more fragile to other methods to incorrect data associations. Therefore EKF SLAM algorithms are not particularly scalable to larger problems.

Modern EKF algorithms implement some features to avoid the issues of the earliest approaches. Furthermore, there is another approach called

Extended Information Filter (EIF) described by the information matrix and the information vector [77, 91]:

$$\Omega = \Sigma^{-1} \tag{39}$$

$$\zeta = \Sigma^{-1}\mu \tag{40}$$

This approach is interesting because many of the off–diagonal components of the *normalized* information matrix are 0, making it a sparse matrix and making update steps more efficient.

Particle Filter SLAM

The particle filter approach seeks to sample the probability distribution with discrete states (known as particles), instead of approximating using Gaussians. Each particle represents the posterior belief of the system and has associated a weight,w, that corresponds to the likelihood of that particle [95, 96]:

$$\chi_t = \{< x^{[j]},\ w^{[j]} >\} \quad j = 1, \ldots, J \tag{41}$$

The steps of the particle filter are:

1. **Sample from proposal distribution**: it is represented by letter π and in this case it is coincident with the motion model distribution:

$$x_t^{[j]} \sim \pi(x_t| \ldots) = p(x_t|x_{1:t-1}, u_t) \tag{42}$$

   After this step, the predicted particles are obtained, $\overline{\chi}_t$, which is the representation of $\overline{bel}(x_t)$ in the Bayes Filter.

2. **Calculate the weights for each particle**: With the particles in x_t, the weight of each one is calculated by dividing the target distribution with the proposal wich is proportional to the observation model:

$$w_t^{[j]} = \frac{target}{proposal} \propto p(z_t|x_t^{[j]}) \tag{43}$$

3. **Resampling**: In this step a sample i is drawn with probability $w_t^{[i]}$ and this process is repeated J times. In this last step, the more likely particles are kept whereas the ones that are more unlikely get discarded.

The problem of particle filters applied to SLAM is that the state space tends to have a notable size (Equation 38). Particle filters scale exponentially, therefore if the number of landmarks is considerable the problem becomes untractable.

FastSLAM, originally developed by [89] is an algorithm that exploits the independence among the different map features Figure 39. In the original implementation, it only applied to feature based maps.

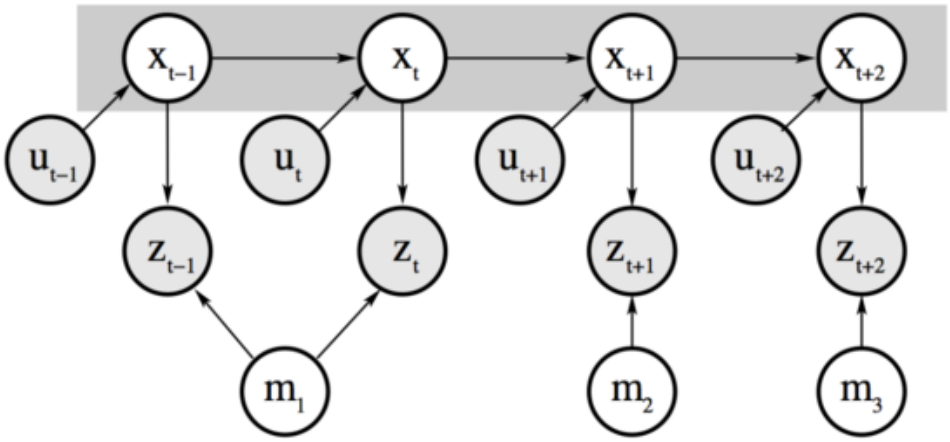


Figure 39: Conceptual view of how each landmark is disconnected from the rest

The belief at time t can be factored as:

$$p(x_{0:t}, m|z_{1:t}, u_{1:t}) = p(x_{0:t}|z_{1:t}, u_{1:t}) \cdot \prod_{i=1}^{N} p(m_i|x_{0:t}, z_{1:t}) \tag{44}$$

where each of the landmarks represents a 2–dimension gaussian distribution with mean and covariance $\mu_{t,n}^{[j]}$ and $\Sigma_{t,n}^{[j]}$ respectively. Due to a technique called *Rao–Blackwellization* for every particle the map posteriors can be estimated independently.

The posterior estimation is done as follows:

1. **Sampling**: A set of particles if sampled from the motion model:

$$x_t^{[j]} \sim p(x_t|x_{t-1}^{[j]}, u_t) \tag{45}$$

2. **Update the observed features estimates**: This is done for each particle and the update is done employing the EKF. For any unobserved feature the gaussian remains the same:

$$\langle \mu_{t,n}^{[j]}, \Sigma_{t,n}^{[j]} \rangle = \langle \mu_{t-1,n}^{[j]}, \Sigma_{t-1,n}^{[j]} \rangle \tag{46}$$

3. **Resampling**: First the weights are calculated with $p(x_{0:t}|z_{1:t}, u_{1:t})$ as target and $p(x_{0:t}|z_{1:t-1}, u_{1:t})$ as proposal distributions. Each weight $w_t^{[j]}$ can be obtained via [96, 97]:

$$\begin{aligned} w_t^{[j]} &= \frac{target(x^{[j]})}{proposal(x^{[j]})} = \frac{p(x_{0:t}^{[j]}|z_{1:t}, u_{1:t})}{p(x_{0:t}^{[j]}|z_{1:t-1}, u_{1:t})} \\ &= \frac{\eta p(z_t|x_{0:t}^{[j]}, z_{1:t-1}) \cdot p(x_{0:t}^{[j]}|z_{1:t-1}, u_{1:t})}{p(x_{0:t}^{[j]}|z_{1:t-1}, u_{1:t})} \\ &= \eta p(z_t|x_{0:t}^{[j]}, z_{1:t-1}) \\ &= \eta \int p(z_t|x_{0:t}^{[j]}, z_{1:t-1}, m_i) \cdot p(m_i|x_{0:t}^{[j]}, z_{1:t-1}) \cdot \mathrm{d}m_i \\ &= \eta \int p(z_t|x_{0:t}^{[j]}, m_i) \cdot p(m_i|x_{0:t}^{[j]}, z_{1:t-1}) \cdot \mathrm{d}m_i \end{aligned} \tag{47}$$

Both probability distributions inside the integral can be approximated by Gaussian distributions and the weights can be computed with an EKF. After obtaining the weights for each particle, the new set is created being the particles with bigger weights more likely to 'survive' to the next round.

The FastSLAM algorithm has some notable properties [89, 97]: 1. it solves both full and online SLAM problems, 2. each particle can make its own data association hypothesis and 3. it is possible to implement it very efficiently.

Apart from the feature–based FastSLAM, there exists a Grid–Map based implementation of the algorithm, which is the basis for the **GMapping** algorithm that will be explained in Section 3.3.1.

Graph–based SLAM

The last of the approaches that will be discused in this thesis is the **Graph–based SLAM**. The intuition behind this algorithm is as follows [94]:

- **Graph**: Corresponds to the whole SLAM problem.
- **Nodes**: Correspond to landmarks and robot locations.
- **Edges**: Correspond to spatial constraints due to motion or observations.

The whole purpose of the Graph SLAM is therefore to find a node distribution that minimizes the measurement errors.

Generally, Graph–based SLAM algorithms have the structure shown in Figure 40, where the front–end takes the measurements are obtained by *matching* observations, and the backend, where given the measurements it builds the constraints and optimizes the graph.

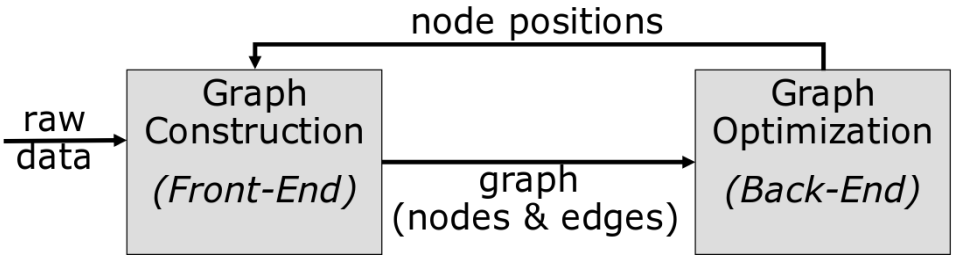


Figure 40: Graph–based SLAM structure

BACKEND:

BACKEND: The definition of the optimization is described in the following paragraphs [82]. The vector describing the pose of every node i is defined $x = (x_1, x_2, \ldots, x_N)^T$. z_{ij} and Ω_{ij} are the mean of the measurement and the information matrix from node i to j. $\hat{z}_{ij}(x_i, x_j)$ is the prediction of a measurement given nodes i and j.

Therefore an error function can be defined as:

$$e_{ij} = z_{ij} - \hat{z}_{ij}(x_i, x_j) \tag{48}$$

$\mathbf{F}(x)$ is the negative log likelihood:

$$\mathbf{F}(x) = \sum_{\langle i,j \rangle} e_{ij}^T \cdot \Omega_{ij} \cdot e_{ij} \tag{49}$$

and the objective is to minimize $\mathbf{F}(x)$:

$$x^* = argmin\ \mathbf{F}(x) \tag{50}$$

Equation 50 corresponds to a **non–linear least squares** problem and it can be solved using well-known methods such as gradient descent, Newton–Gauss, Lavenberg–Marquardt [82].

Generally, this is done by linearizing the error function with a first order Taylor expansion around the initial guess $\breve{x}$:

$$\begin{aligned} e_{ij}(\breve{x}_i + \Delta x_i, \breve{x}_j + \Delta x_j) &= e_{ij}(\breve{x} + \Delta x) \\ &\approx e_{ij} + J_{ij}\Delta x \end{aligned} \tag{51}$$

where J_{ij} is the jacobian matrix.

In terms of F_{ij} the equation results in:

$$\begin{aligned} & F_{ij}(\breve{x} + \Delta x) \\ &= (e_{ij}(\breve{x} + \Delta x)^T \cdot \Omega_{ij} \cdot e_{ij}(\breve{x} + \Delta x) \\ &\approx (e_{ij} + J_{ij}\Delta x)^T \cdot \Omega_{ij} \cdot (e_{ij} + J_{ij}\Delta x) \\ &= e_{ij}^T \Omega_{ij} e_{ij} + 2e_{ij}^T J_{ij}\Delta x + \Delta x J_{ij}^T \Omega_{ij} J_{ij} \Delta x^T \\ &= c_{ij} + 2b_{ij}\Delta x + \Delta x^T H_{ij} \Delta x \end{aligned} \tag{52}$$

The function $\mathrm{F}(x)$ then becomes:

$$\begin{aligned} \mathbf{F}(\breve{x} + \Delta x) &= \sum_{\langle i,j \rangle} F_{ij}(x + \Delta x) \\ &\approx \sum_{\langle i,j \rangle} c_{ij} + 2b_{ij}\Delta x + \Delta x^T H_{ij} \Delta x \\ &= \mathbf{c} + 2\mathbf{b}^T \Delta x + \Delta x^T \mathbf{H} \Delta x \end{aligned} \tag{53}$$

Equation 53 can be minimized by solving the linear system:

$$\mathbf{H}\Delta x^* = -\mathbf{b} \tag{54}$$

and therefore the solution becomes:

$$x^* = \breve{x} + \Delta x^* \tag{55}$$

FRONTEND:

FRONTEND: The frontend of the Graph–based SLAM obtains the measurements by performing a **scan–matching** of current sensor data with previous points. The algorithms that tackle this problem are **Iterative Closest Point (ICP)**, **Correlative matching**, **Normal Distributions Transform (NDT)** among many others.

Graph SLAM algorithms have 2 key advantages [97] in comparison with EKF SLAM: The update of the map is constant in time and the memory utilization grows linearly. However, the optimization step is resource consuming.

One of the features of these algorithms is that they serve for both 2D and 3D maps [82]. The 'only' change that has to be made is to substitute the laser scans with 3D pointclouds. In fact Figure 34 was built using a Graph–based SLAM algorithm.

In the past years, Graph SLAM algorithms have been subject to profound research and have become the state–of–the–art algorithms in this field [96, 97]. Some of the biggest SLAM maps have been created using this approach and contain up to 10^8 features.

OPEN-SOURCE SLAM ALGORITHMS

Fortunately, these algorithms have already been implemented in software packages and many of them are available in the open–source community. What is more, due to the success of the ROS ecosystem, generally SLAM algorithms come bundled with ROS libraries, thus making them easier to interface.

In the following pages, the main algorithms used to develop the SLAM systems of both the Nexus Robot and the PEV will be described.

SLAM Gmapping

As it was mentioned on Section 3.2.2, **Gmapping** is a particle filter SLAM variant that employs grid maps instead of feature–based maps [83, 84], that incorporates some improvements to the original algorithms:

- **Scan–matching**: Before applying the particle filter, the pose is corrected using a scan–matcher to maximize the likelihood of the current pose and map:

$$x_t^* = \underset{x_t}{argmax}\{p(z_t|x_t, m_{t-1})p(x_t|u_t, x_{t-1}^*)\} \tag{56}$$

- **Improving the proposal distribution**: Recalling from Section 3.2.2, the importance weights are a result of the ratio between the target distribution $p(x_{0:t}|z_{1:t}, u_{1:t})$ and the proposal $\pi(x_{0:t}|z_{1:t}, u_{1:t})$. Gmapping computes this proposal taking into account the most recent scan, thus improving accuracy:

$$p(x_t|m_{t-1}^{[j]}, x_{t-1}^{[j]}, z_t, u_t) = \frac{p(z_t|m_{t-1}^{[j]}, x_t)p(x_t|x_{t-1}^{[j]}, u_t)}{p(z_t|m_{t-1}^{[j]}, x_{t-1}^{[j]}, u_t)} \tag{57}$$

- **Adaptive resampling**: Even though resampling is necessary since the number of particles is finite and the less likely ones must be removed, the process might also remove good particles. Therefore Gmapping computes the following parameter:

$$N_{eff} = \frac{1}{\sum_{i=1}^{N} (\tilde{w}^{[j]})^2} \tag{58}$$

  where $\tilde{w}^{(i)}$ is the normalized weight. The algorithm will do a resampling step whenever N_{eff} drops below $N/2$, being N the number of particles. Thus, the resampling operation is done when needed.

Gmapping is one of the first open–source alogorithms to appear on the ROS ecosystem[1] and it blends naturally with the Navigation Stack. Some of the main characteristics of this ROS package are:

- **Subscribed topics**: The main topics are `/tf` to obtain the rigid body transforms and `scan` to obtain the laser messages. The necessary transforms are:
  - `laser_frame` → `base_link`. Transformation between the frame of the range sensor and the main frame of the robot.
  - `base_link` → `odom`. Transformation between the base frame and an odometry frame.
- **Published topics**: Gmapping publishes the **occupancy grid map** to the `/map` topic.
- **Provided transforms**: Gmapping provides the transform `map` → `odom` with the robot's pose within the map.
- **Parameters**: Table 3 shows a list of the main parameters of the algorithm. The full list can be found on the ROS Wiki

Name	Default	Description
~base_frame	"base_link"	Frame attached to mobile base
~map_frame	"map"	Frame attached to the map
~odom_frame	"odom"	Frame attached to odometry
~map_update_interval	5	Time (s) between map updates
~maxUrange	80	Usable range of the laser
~iterations	5	Max iterations of the scanmatcher
~linearUpdate	1.0	Process scan every x meters
~angularUpdate	0.5	Process scan every x radians
~particles	30	Number of particles in the filter
~resampleThreshold	0.5	N_{eff} limit to resample

Table 3: Gmapping main parameters

1 http://wiki.ros.org/gmapping

Hector SLAM

Hector SLAM is another ROS–based SLAM approach that fully integrates with the Navigation Stack and that is capable of performing 3D motion estimation [88]. Figure 41 shows the Hector SLAM system overview.

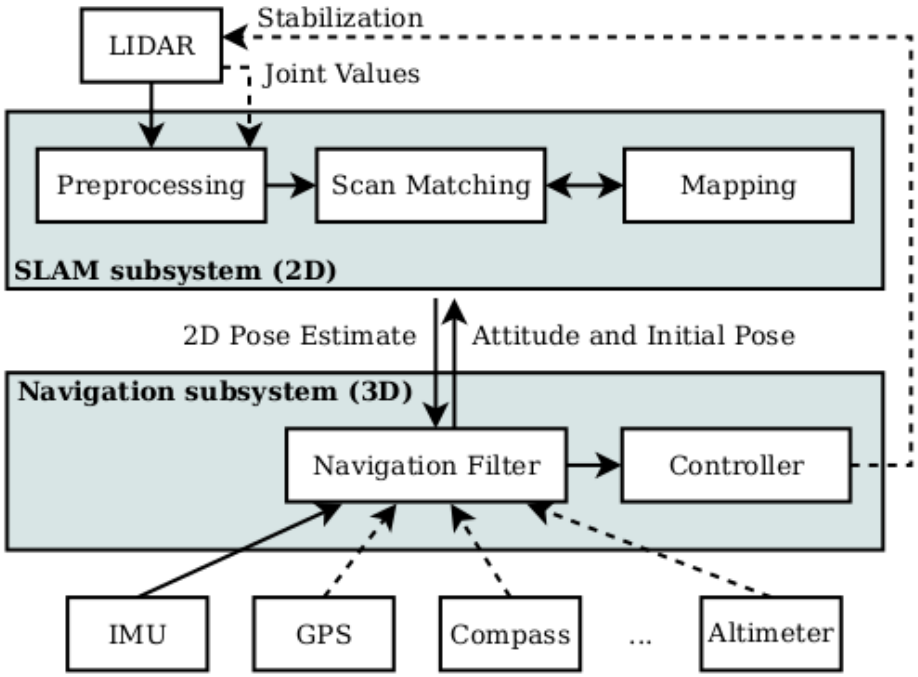


Figure 41: Hector SLAM overview

There are 2 clear layers in the system:

- **SLAM subsystem**: It reads the scans from the laser, performs the scan matching opperation and stores the results on a grid–map in 2 dimensions. This algorithm does not divide the SLAM between frontend and backend as it was explained in Section 3.2.3: instead, it relies on the scan–matching operation to update the map. Maps are divided into multiple resolution layers to gain more consistency in the localization.
- **Navigation subsystem**: It estimates the 6D pose by employing an EKF: the 2D pose given by the SLAM layer is augmented with inertial sensors or GPS. The 3D state vector is defined as $\mathbf{x} = \{\mathbf{\Omega}^T\ \mathbf{p}^T\ \mathbf{v}^T\}^T$ with $\mathbf{\Omega} = \{\phi, \theta, \varphi\}^T$ the roll–pitch–yaw angles, $\mathbf{p} = \{p_x, p_y, p_z\}^T$ the position and $\mathbf{v} = \{v_x, v_y, v_z\}$ the velocity in the navigation frame.

Because it does not perform any graph optimization on the backend, this algorithm is best suited for smaller scale scenarios with less loop closures to execute. However, this algorithm has one interesting feature which is the possibility of not using odometry.

With regards to the Hector SLAM ROS API, here are some of the main features of this package:

- **Subscribed topics**: The main topics are `/tf` and `scan` like in Gmapping. The necessary transforms are:
  - `laser_frame` → `base_link`. Transformation between the frame of the range sensor and the main frame of the robot.
  - `base_link` → `odom`. Only if the odom frame is provided by an external node.
- **Published topics**: Hector SLAM publishes the map to `/map` and the pose of the robot to `/poseupdate`.
- **Provided transforms**: Hector SLAM provides the transform `map` → `odom` with the robot's pose within the map.
- **Parameters**: Table 4 shows a list of the principal parameters available for Hector SLAM

Name	Default	Description
~base_frame	"base_link"	Frame attached to mobile base
~map_frame	"map"	Frame attached to the map
~odom_frame	"odom"	Frame attached to odometry. It can be set to "base_link" when no odometry is provided
~map_update_distance_thresh	0.4	Threshold for map update after linear motion
~map_update_distance_thresh	0.9	Threshold for map update after angular motion
~map_multi_res_levels	3	Number of grid levels for the map
~laser_min_dist	0.4	Min distance to consider a scan measurement
~laser_max_dist	30	Max distance for a scan measurement
~pub_map_odom_transform	True	Determine if `map` → `odom` transform should be published by the system
~scan_subscriber_queue_size	5	Queue for the scan measurements

Table 4: Hector SLAM main parameters

Google Cartographer

Cartographer is a relatively recent SLAM package (2016) developed by Google [85], that aims at providing real–time 2D mapping in indoor environments.

Cartographer is an example of a Graph–based SLAM algorithm with 2 definite layers (Figure 42):

- **Local SLAM**: The local SLAM matches every scan with a small portion of the world called submap. To perform that matching it employs the Ceres solver [76].

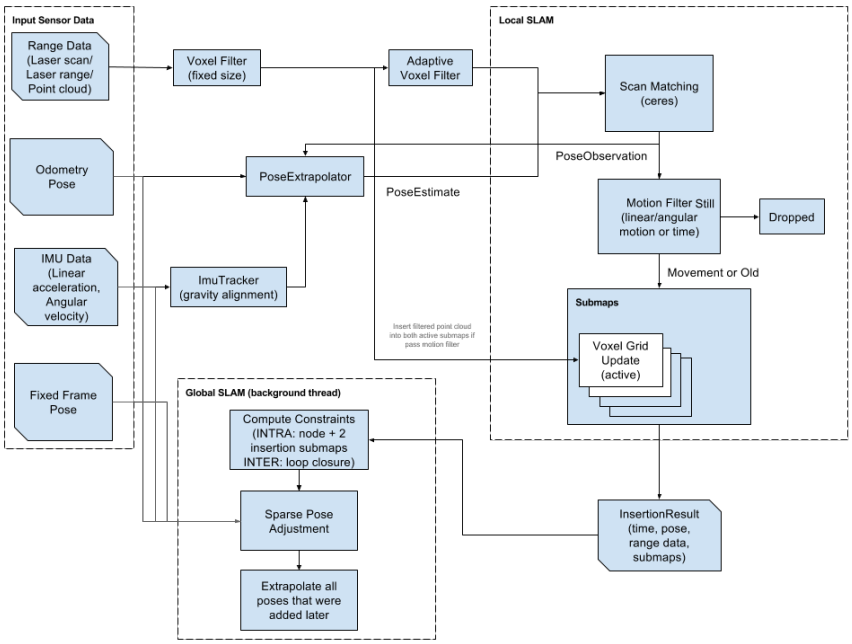


Figure 42: Cartographer system overview

- **Global SLAM**: Matching scans against a submap accumulates errors over time. That is why every few seconds Cartographer takes those submaps with their underlying constraints and performs a loop–closure optimization employing once again the Ceres–solver.

As of July 2018, Cartographer provides both 2D and 3D SLAM and localization options with a extense variety of sensors that can be added: 2D/3D LIDARs, IMU, odometry information and GPS among others.

Cartographer provides as well ROS wrappers that allow its use with the existing stack of packages:

- **Subscribed topics**: The topics that it subsribes are (Table 5):

Topic	Message	Description
scan	sensor_msgs/LaserScan	Planar laser scanners
echoes	sensor_msgs/MultiEchoLaserScan	Axially rotating planar laser scanners
points2	sensor_msgs/PointCloud2	2D/3D pointclouds
imu	sensor_msgs/Imu	Optional in 2D, required in 3D
odom	nav_msgs/Odometry	Optional in both modes

Table 5: Cartographer subscribed topics

- **Published topics**: `scan_matched_points2` with the pointcloud and `submap_list` with a list of all submaps.
- **Transforms**:
  - **Required**: `tracking_frame` → `published_frame`

  - **Provided**: `published_frame` → `map_frame`

- **Parameters**: The amount of features and parameters that can be tweaked is considerable as it is shown below:
  - **General options**: Main sensors, the frame names and SLAM main options.
  - **Map builder**: SLAM method (2D or 3D) and the number of threads.
  - **Trajectory builder**: Options for the scan–matcher and the trajectory.
  - **Pose graph**: Options for the global optimizer

After the information given above, it can be noted that Cartographer is a powerful tool that enables SLAM in relatively large areas. A notable example is the mapping of the whole first floor of the Deutsches Museum as shown in Figure 43.

Figure 43: 2D map of the Deutsches Museum done with Cartographer

Autoware and Normal Distributions Transform (NDT)

The last software tool employed is Autoware, a ROS–based open–source software package aiming at developing outdoor autonomous car technologies [86, 87].

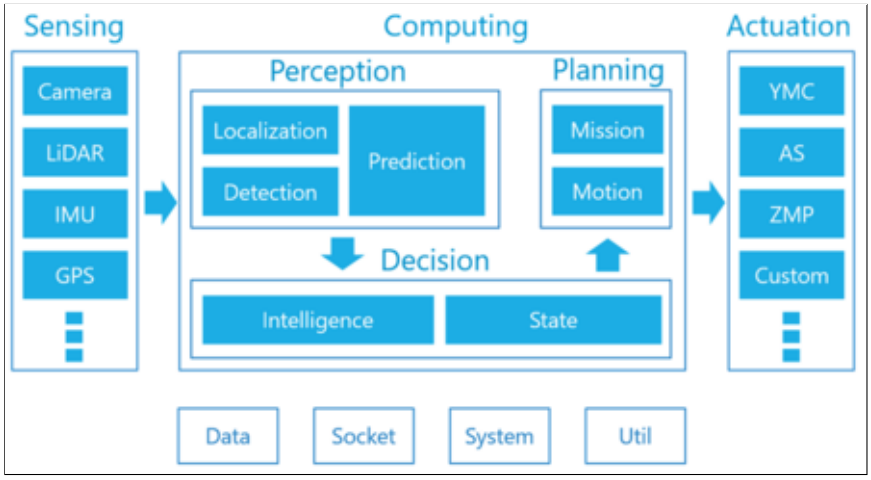


Figure 44: Autoware functionalities

It is not a SLAM algorithm per–se, but a standalone module that provides libraries for motion planning, perception and actuation (Figure 44). Among the perception packages several 3D SLAM and localization algorithms can be found, that merge 3D LiDAR, IMU and GPS sensors among others.

The 3D SLAM algorithms rely on scan–matching (like in Hector SLAM), and concretely they apply the 3D version of the Normal Distributions Transform (NDT) [78]. After each localization, the algorithm registers the pointcloud and updates the 3D map. Figure 34 is an example of a map built using this technique.

NDT mapping can be used with the Graphical User Interface (GUI) shown in Figure 45 or as a standalone module.

Figure 45: Autoware GUI

If used as a standalone module, the required aspects are:

- **Subscribed topics**: Subscribed topics are shown in Table 6

Topic	Message	Description
points_raw	sensor_msgs/PointCloud2	3D pointcloud
vehicle/odom	nav_msgs/Odometry	Optional
imu_raw	sensor_msgs/Imu	Optional
ndt_mapping	ConfigNdtMapping	Configuration for autoware
ndt_mapping_output	ConfigNdtMappingOutput	Configuration for the pointcloud

Table 6: Autoware's NDT subscribed topics

- **Published topics**: NDT mapping publishes a single topic with the 3D map of the environment called `/ndt_map`.
- **Transforms**: The required transforms are `sensor_frame` → `base_link`.
- **Parameters**: Parameters are specified in Table 7

Name	Default	Description
~tf_xxx	no default	These are 6 parameters for the x, y, x, roll, pitch and yaw transform between base and map
~method_type	0	Specifies type of pointcloud
~odom_frame	"odom"	Frame attached to odometry. It can be set to "base_link" when no odometry is provided
~use_imu	False	Specifies if IMU is used
~imu_upside_down	False	Specifies if IMU is turned 180°
~imu_topic	/imu_raw	Topic to listen IMU messages
~use_odom	False	Specifies if odometry is used

Table 7: Autoware's NDT mapping main parameters

4

NEXUS ROBOT

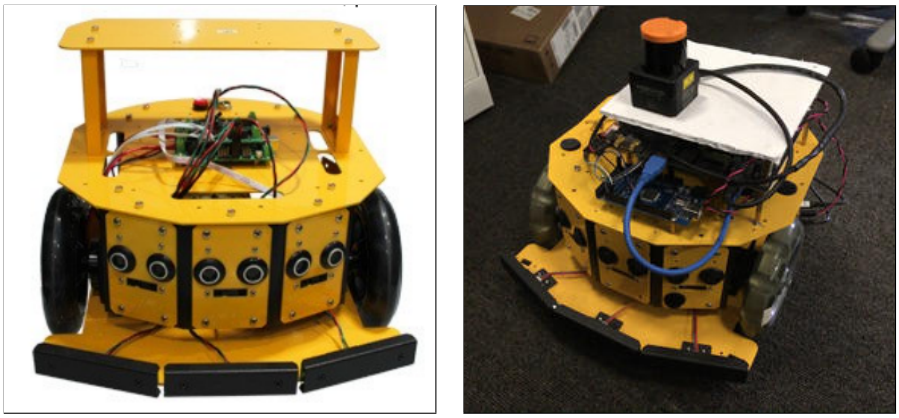
This chapter will talk about the Nexus Robot, a small 2–wheeled robot that was used as a testbed for different SLAM algorithms, before applying them on the PEV.

Another purpose for this robot is to serve as an educational platform, by creating interfaces for students of different segments to learn about mobile robots and Autonomous Vehicles.

The last reason to research on these smaller robots is to explore the idea of using them as a 'last meter' delivery system, making intra–office communication much faster and bridging the gap between a platform like the PEV and the end user/client.

DESCRIPTION OF THE ROBOT

The Nexus Robot is a 2 wheeled differential drive platform as shown in Figure 46, that is used for educational robotics and costs around 500$.



(a) Nexus robot as sold (b) Setup

Figure 46: Nexus robot

The body of the robot is made of aluminum and its dimensions are 30cm in circumference and 20cm in height. Due to its size it is ideal for indoor navigation and the fact that is a differential drive improves its manouverability in those environments.

Generally, it does not come with very advanced sensors, thus in order to perform autonomous navigation tasks its capabilities had to be enhanced

with more sensors and computing units. Those will be described in the following sections.

Hardware

Sensors

sensors The sensors that come with and were added to the platform are:

- **Sonar**: The robot comes with 3 low–range ultrasonic sensors that are originally intended for obstacle detection. However, they were not used in this experiments.
- **Bumper sensors**: It also comes with 3 bumber sensors to detect collisions but those have not been used either.
- **Wheel encoders**: Each wheel comes with a 2–channel encoder (A and B) to calculate the linear and rotational speeds.
- **Lidars**: 2 diiferent lidars were used throught the experiments (Figure 47):
  - RPLidar A2: It is an 12m range lidar, with a 360º field of view and outputs 4000 points per/second (400 per revolution). The connection is done via USB adapter.
  - Hokuyo UST-10LX: The range of this lidar is of 10m, with a 270º field of view and more than 40.000 points per second (1080 per scan). This laser uses ethernet to connect to the computer.

(a) RPLidar A2 (b) Hokuyo UST-10LX

Figure 47: Lidars used in the Nexus robot

Actuators

actuators The robot comes with 2 12V DC motors, one for each wheel and each one has 2 control pins: one for rotation direction and one for speed.

Control

control Control comes from 2 platforms:

- Controller board: Connected to a 12V power supply, it is in charge of sending power and control commands to the DC motors. It is

equipped with an ATMega 168 and can be programmed with the Arduino IDE, but the low amount of memory (16KB) makes the controller's behavior unstable when bridged to ROS.

- Arduino Mega: It is one of the most complex Arduino boards (Figure 48), with 54 digital pins, 16 analog pins and a memory of 256KB. That capacity is important since the board was bridged to communicate with the ROS system, and ROS libraries occupy a notable amount of space on microcontrollers.

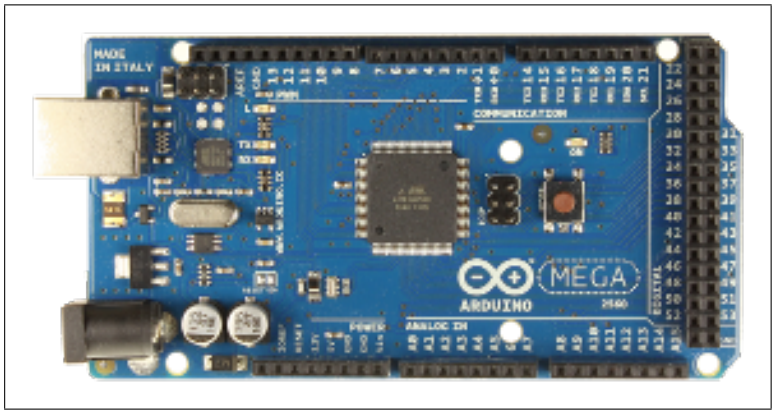


Figure 48: Arduino Mega board layout

Computing

COMPUTING The computations onboard are performed by a Jetson TX2 Developer platform developed by Nvidia (Figure 49). This device is part of a branch of development computers aimed at tasks like autonomous navigation or deep learning. Some of the features are: 32GB hard–drive, 8GB of RAM and a 256–core GPU.

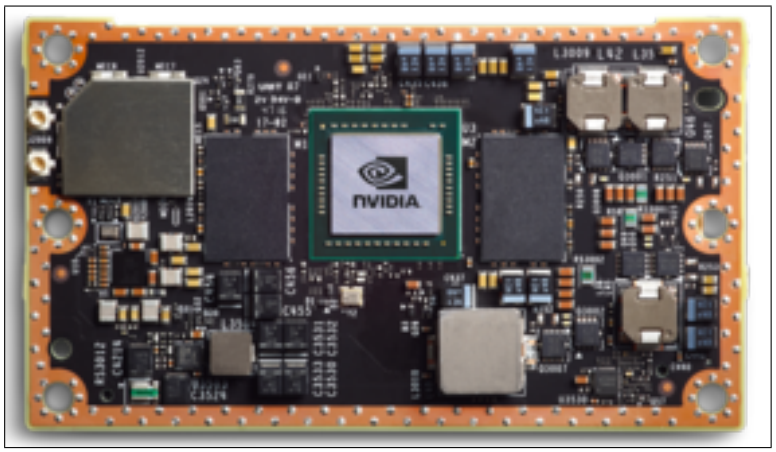


Figure 49: Jetson TX2 module

The Operating System is Ubuntu 16.04 and it has libraries such as CUDA, CUDNN, OpenCV and TensorRT preinstalled in order to perform all sorts of deep learning and artificial intelligence tasks that require parallel com-

puting. It is also compatible with ROS, thus making the TX2 an interesting device for self–driving vehicles.

SETUP

The conceptual setup of the Nexus robot is depicted in Figure 50.

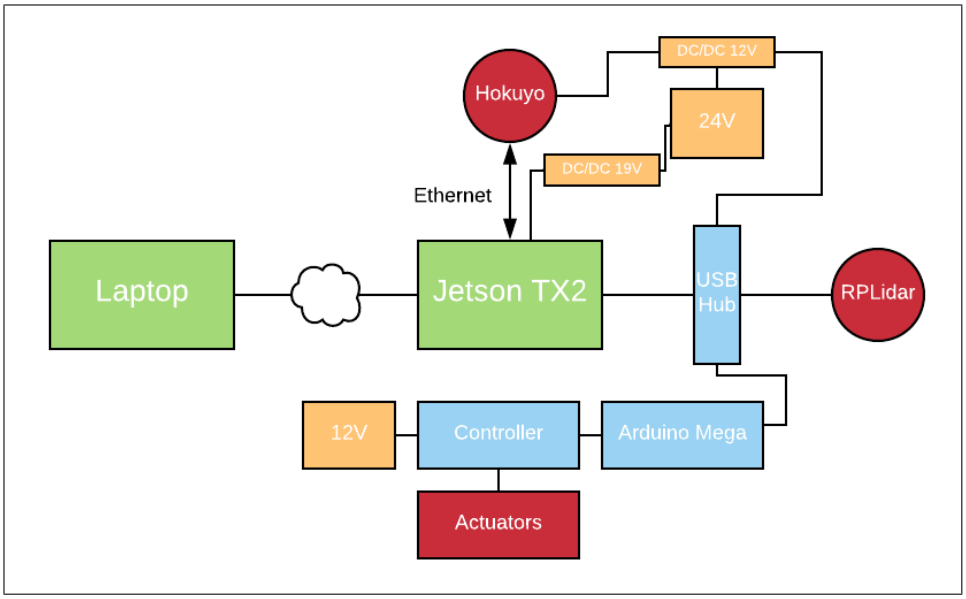


Figure 50: Nexus setup conceptual overview

As it can be seen, the Jetson TX2 is in charge of running all the sensors and controllers and then it is connected via WiFi to the laptop. The latter stores the data from the experiments and visualizes it on RVIZ. This connection is possible thanks to the networking capabilities of ROS.

Control

It was briefly mentioned above how the Arduino controlled the actuators of the robot. In this subsection this explanation will be expanded.

Both motors are set to Pulse–Width Modulation (PWM) mode and each one has 2 pins to be set, as shown in Table 8.

Pin	Name	Description
4	M1	Direction control of Motor 1
5	E1	PWM control of Motor 1
6	E2	PWM control of Motor 2
7	M2	Direction control of Motor 2

Table 8: Pin controls of the motors

Motors 1 and 2 correspond to the left and right wheels. As for the encoders, each one has 12 pulses per revolution, 2 channels and a gearbox ratio of 64:1. That yields $64 \cdot 12 \cdot 4 = 3072$ pulses per revolution. In order to obtain a pulse every time there is a change, 2 interrupts have to be attached to each encoder, one for channel. From the Arduino IDE this can be done as in Listing 4.1, where ELA and ELB are the pins connected to channel A and B of the left encoder.

Listing 4.1: Attaching interrupts

```
attachInterrupt(digitalPinToInterrupt(ELA), doEncoderlA
    , CHANGE);
attachInterrupt(digitalPinToInterrupt(ELB), doEncoderlB
    , CHANGE);
```

In Figure 51, a schematic of the wiring of the system is shown.

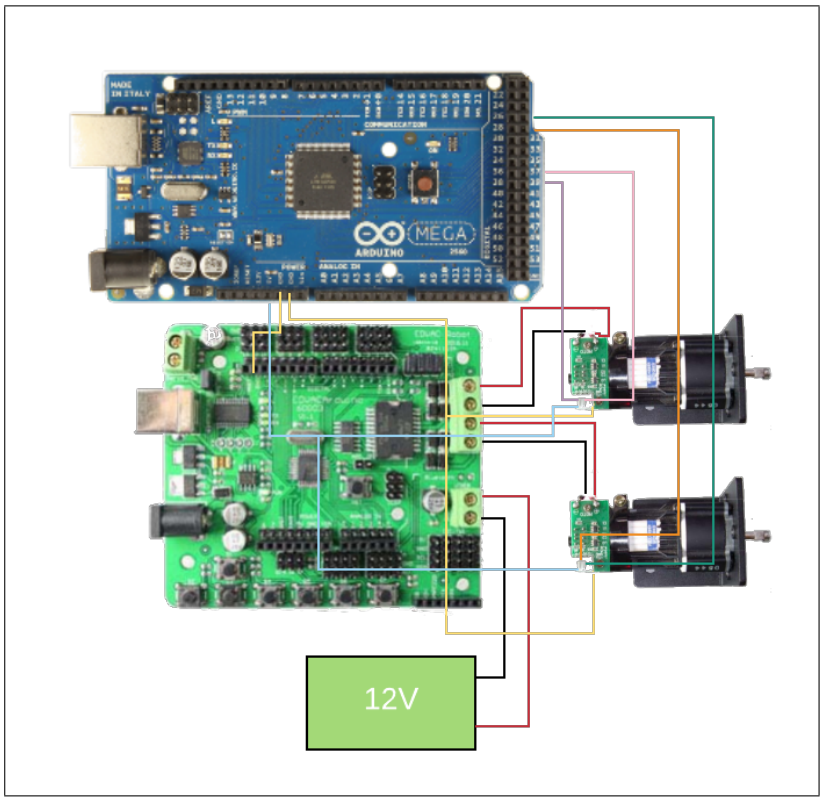


Figure 51: Nexus robot control wiring

Speed commands

Speed commands come in the form of the ROS message type `Twist.h` and then are translated to left and right wheel speed. 2 different sources output speed commands:

- **Joystick**: When recording data to map the environment a Logitech gamepad was used to control the robot.
- **Navigation system**: When running in autonomous mode, speed commands come from the navigation stack, specifically the local planner.

ROS package

All the functionalities of the Nexus Robot have been gathered in a collection of ROS packages that can be found on Github. Table 9 is offered as an explanation of the different modules provided:

Package name	Description
minion_teleop	Scripts to run the joystick and send speed commands to the robot
nexus_robot_2wd	Contains the Arduino code, hardware rules and odometry nodes
nexus_robot_cartographer	Scripts to run Google's cartographer on the nexus robot
nexus_robot_localization	Contains the autonomous navigation software (localization, path planning and move_base)
nexus_robot_mapping	Scripts to run Gmapping or Hector SLAM
nexus_robot_msgs	Custom ROS messages for the platform

Table 9: ROS packages to control the nexus robot

INDOOR SLAM WITH NEXUS

Once the main aspects of the Nexus robot setup have been described, this section will show the results of the SLAM performed in 3 different scenerarios: **City Science group's lunch room**, **City Science group** and the **Media Lab's third floor**.

The algorithms employed with the Nexus robot have been **Gmapping** and **Hector SLAM**. For every environment Gmapping will be shown first and then Hector.

Autonomous navigation workflow

For every environment the procedure to perform SLAM and subsequent navigation was the same: 4 test runs are performed, and basic sensor data is recorded into bag files: encoder tics and laser data.

2 of those recorded files were used to perform SLAM and a considerable amount of maps was obtained. The best map of the batch was then used to test localization and path planning algorithms. Once the results from tests were correct, online runs were carried.

Lunch Room

The lunch room is a small (5×5 m^2) space where the City Science has its meetings. It is an ideal place to start with mapping algorithms and measure their robustness to different parameters.

GMAPPING IN LUNCH ROOM

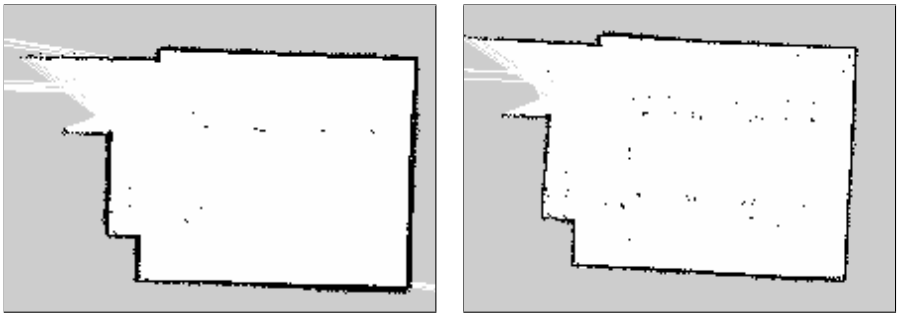
GMAPPING IN LUNCH ROOM The tests performed with gmapping are shown in Table 10:

Parameter	Range
Default	-
Linear update rate	[0.1-1.0]
Angular update rate	[0.1-1.0]
Number of iterations for the scanmatcher	[0-10]
Number of particles in the filter	[1-30]

Table 10: Tests with Gmapping in the lunch room

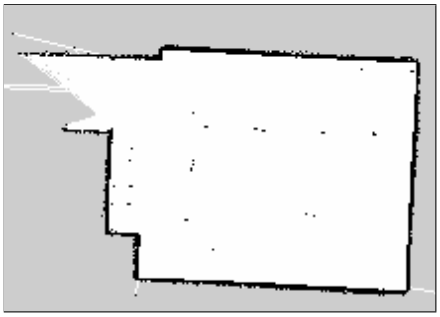
Figure 52 shows the occupancy grid map with the default values. The result is satisfactory and it would suffice to peform navigation. However, in order to test the robustness of the algorithm, the rest of the cases are shown.

Figure 52: Lunch room map with defaults for Gmapping



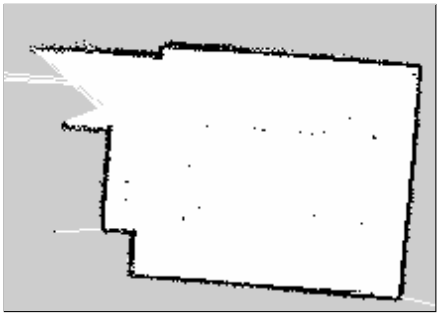
(a) Linear update every 0.1 m

(b) Linear update every 1.0 m



(c) Angular update every 0.1 rad

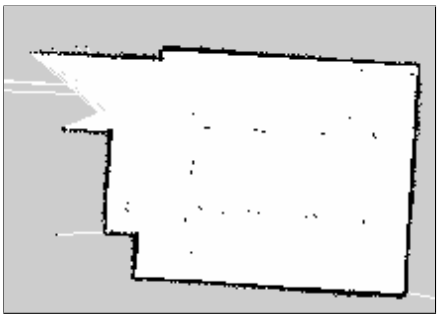
(d) Angular update every 1.0 rad



(e) 0 iterations

(f) 10 iterations

(g) 1 particle



(h) 10 particles

Figure 53: Lunch room map varying the linear update rate

As it can be sheen on Figure 53, the results are very similar for a wealth of variations in the parameters, therefore it can be concluded that Gmapping has a robust behavior in this scenario.

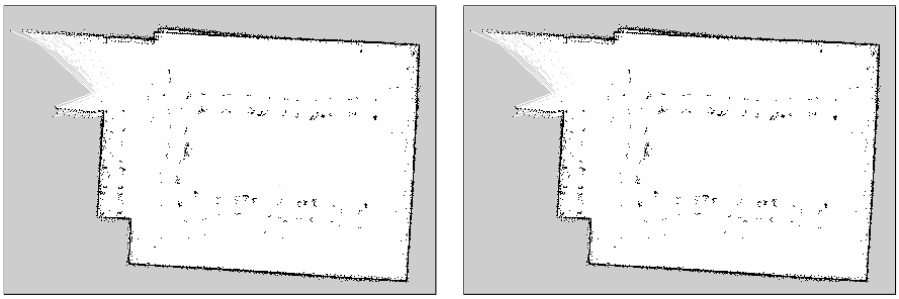
HECTOR IN LUNCH ROOM

HECTOR INLUNCH ROOM For Hector SLAM, since it can run with or without odometry, tests were run with both cases in mind. Table 11 summarizes the cases of study (with and without odometry)

Parameter	Range
Default	-
Multimap resolution	2 & 3
Linear update rate	[0.2-0.6]
Angular update rate	[0.4-1.4]

Table 11: Tests with Hector in the lunch room

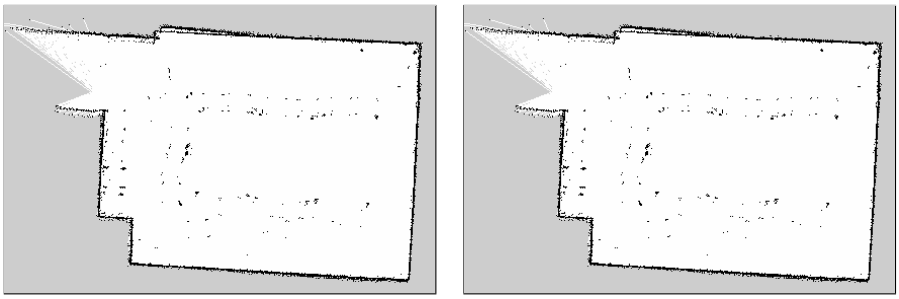
Figure 54 shows, the results of the default maps obtained and there is not much difference between them. However, on the left corner of the map there is a small misadjustment that was not seen with Gmapping.



(a) With odometry (b) Without odometry

Figure 54: Default results for Hector SLAM (Lunch Room)

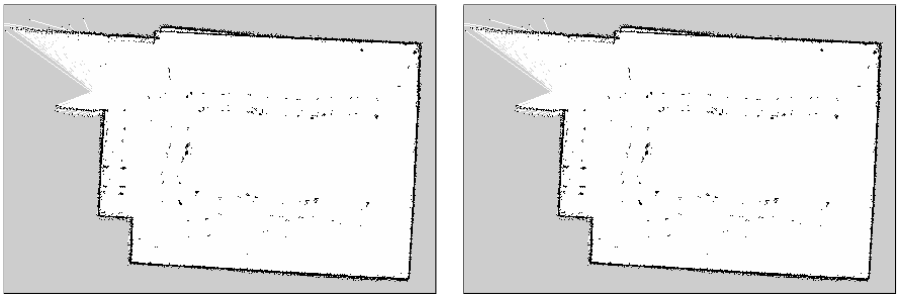
This problem is persistent across all the configurations no matter if using odometry or not (Figures 55 and 56).



(a) With odometry (b) Without odometry

Figure 55: Linear update at 0.2 m with Hector (Lunch Room)

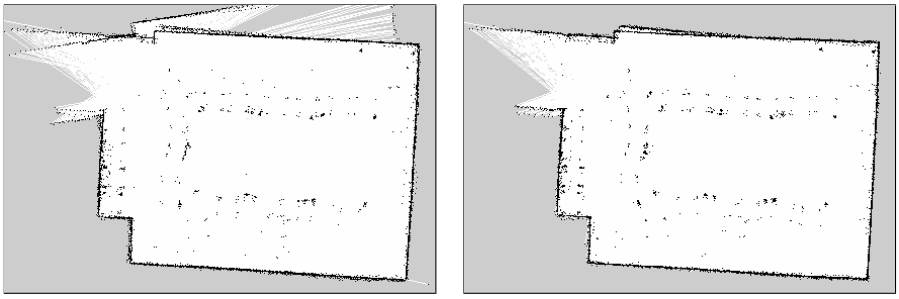
Thus, it could be due to the map multi resolution parameter, since maps are divided into coarser layers and matches are done with the previous



(a) With odometry (b) Without odometry

Figure 56: Angular update at 0.4 rad with Hector (Lunch Room)

map. In Figure 57, the resolution was changed to 2 and the drift is higher, possibly meaning that the scanmatcher needs more layers. With 4 maps, the drift is barely appreciable, but there is not much change when using 3 layers.



(a) Map resolution 2 (b) Map resolution 4

Figure 57: Varying the map resolution with Hector (Lunch Room)

Both maps in Figure 57 were obtained without odometry, since again there was not any significant change between using or not using it.

The outcome of the mapping process in the lunch room is that Gmapping behaves in a robust fashion and delivers acceptable results without tinkering. On the other hand, with Hector SLAM, results are consistent across configurations but in the overall it lags behind Gmapping.

City Science Lab

The next environment in which the Nexus robot was tested was the City Science (CS) group's laboratory. This room is bigger (around 400m²) and is full of features and obstacles, therefore it is and interesting place to perform SLAM in.

GMAPPING IN CS

GMAPPING IN CS The different parameters tested are shown in Table 12. The default map is shown in Figure 58. The results are acceptable for most of the area except for the left side of the map, where it is not complete.

Parameter	Range
Default	-
Linear update rate	[0.2-0.6]
Angular update rate	[0.4-1.4]
Number of particles in the filter	[1-50]
Occupied threshold	[0.3-0.9]

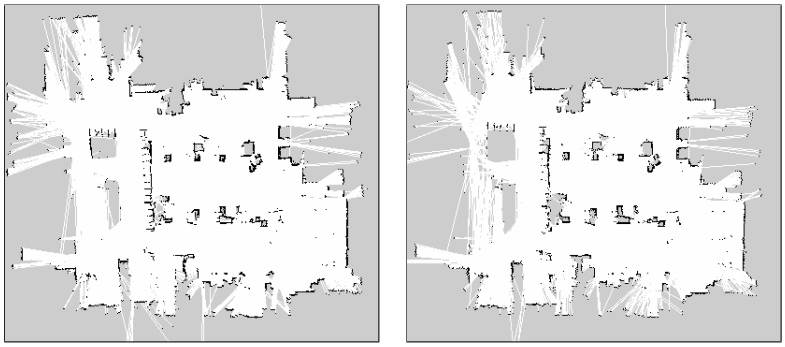
Table 12: Tests with Gmapping in CS laboratory

The cause for this to happen would be that region is surrounded by glass doors and acrylic surfaces (grey rectangle embedded on the left part) that compose the base of the CityScope platforms.

Figure 58: CS with defaults for Gmapping

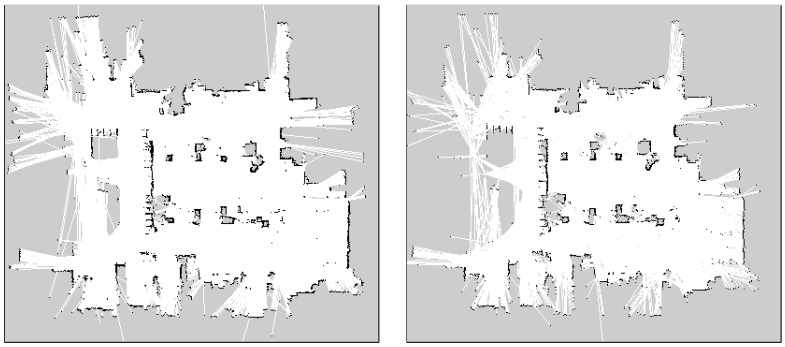
In order to mitigate this issue, the approaches taken were to vary the linear and angular update rates, as well as the occupied threshold and the particles. Results are shown in Figure 59

From these results, the folowing conclusions can be obtained: increasing the update rate improves the area, but in the case of the linear update there is a limit to the improvement, since with 0.1m, the map starts drifting. For the angular case, this does not occur and a continued lowering of the rate value improves the overall quality.



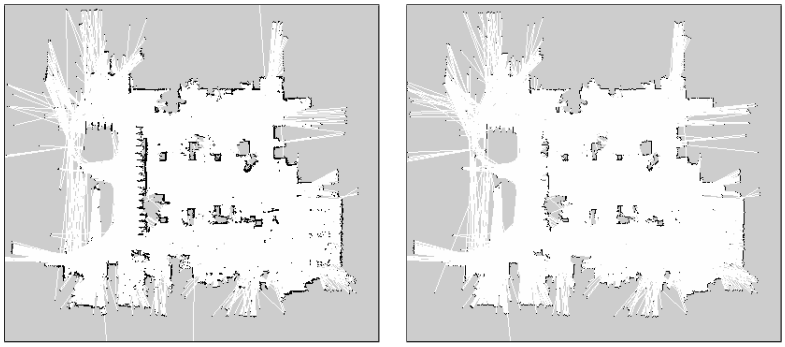
(a) Linear update at 0.1 m

(b) Linear update at 0.5 m



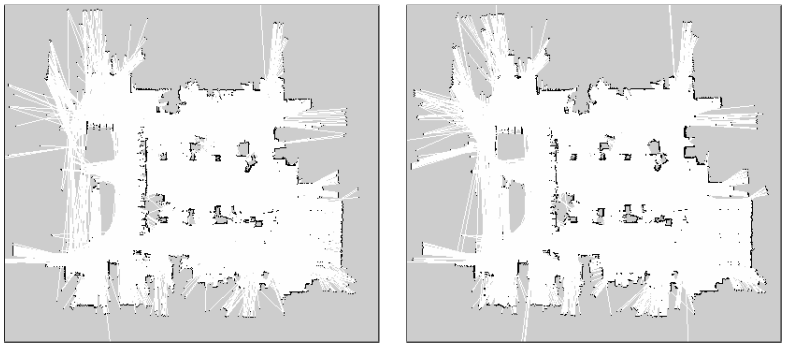
(c) Angular update at 0.1 rad

(d) Angular update at 1.0 rad



(e) Occupancy threshold of 0.3

(f) Occupancy threshold of 0.8



(g) 5 particles

(h) 50 particles

Figure 59: CS results with Gmapping

The reason to vary the occupied threshold in this case was to mitigate the effect of small obstacles or features. In fact, with an occupied threshold of 0.8 there are fewer 'isolated' points but the overall result is 'blurrier'.

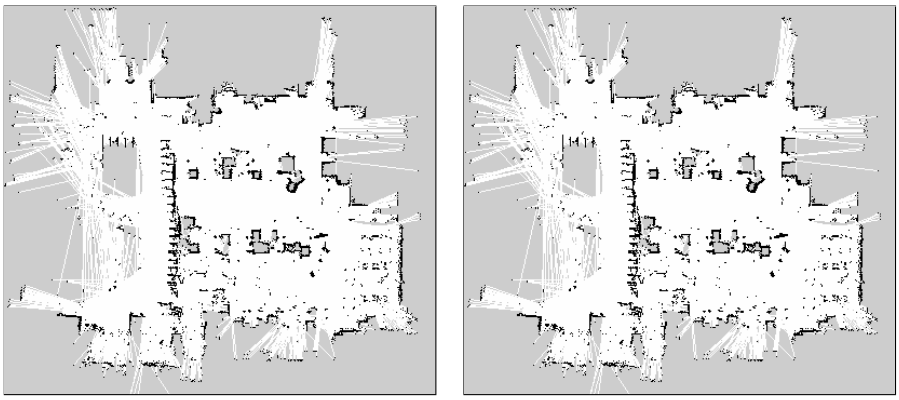
With regards to the number of particles, it seems that increasing their number helps mititigate the voids on the left region, due to the fact that with more particles there are more maps and they are able to approximate reality better.

HECTOR IN CS The parameters chosen with Hector SLAM are shown in Table 13 (again with and without odometry):

Parameter	Range
Default	-
Multimap resolution	[2-6]
Linear update rate	[0.2-0.6]
Angular update rate	[0.4-1.4]

Table 13: Tests with Hector in CS laboratory

The default maps obtained in both cases show a drift on the lower part that causes the map to be unacceptable (Figure 60). As Hector SLAM does not perform any background optimization and relies only on the scanmatcher, when the lidar does not detect obstacles (as it happens on the left area) the state becomes drifted.

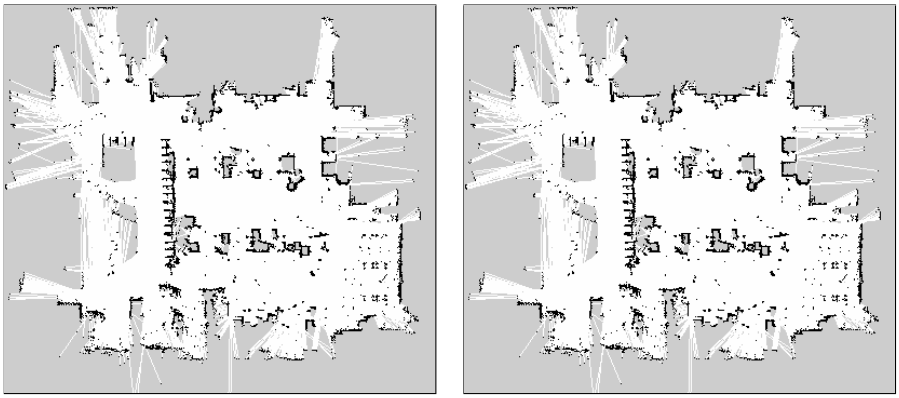


(a) With odometry (b) Without odometry

Figure 60: Default results for Hector SLAM (CS)

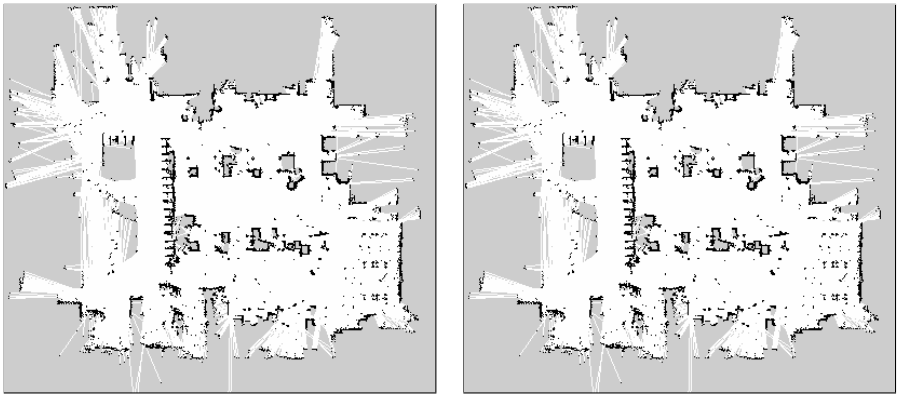
To solve this issue 2 approaches were taken: the first one was to increase both linear and angular update rates (Figures 61 and 62) and the second one to vary the map resolution (63). Again there is no observable difference between using odometry or not meaning that the scanmatcher is barely affected by the errors due to motion estimation.

It can be observed than in the case of the CS laboratory, only the update rates play and important role when tackling the mapping issues. Specially the variation of the linear rate has more visible effects on the improvement of the occupancy grid map.



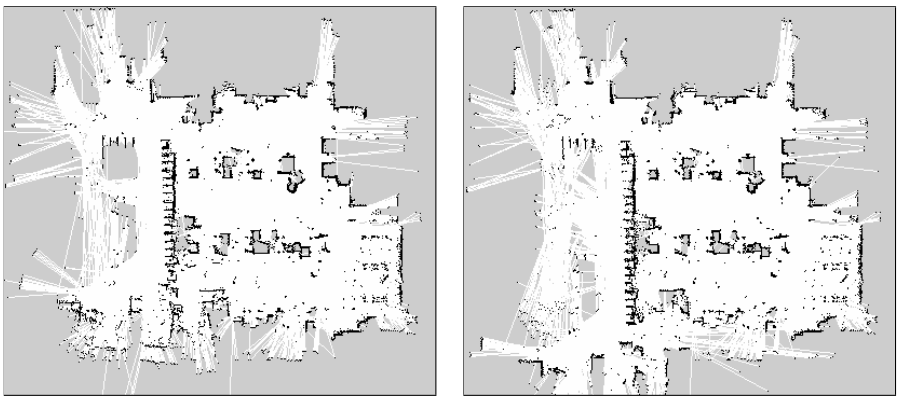
(a) With odometry (b) Without odometry

Figure 61: Linear update rate 0.2 m with Hector (CS)



(a) With odometry (b) Without odometry

Figure 62: Angular update rate 0.1 rad with Hector (CS)



(a) Map resolution 2 (b) Map resolution 4

Figure 63: Varying map resolution with Hector (CS)

This time again results with Gmapping are slightly better. With regards to Hector, for these relatively small scenarios, odometry does not have any effect on the final result of the SLAM.

Media Lab 3rd floor

The last environment tested with the robot was the Media Lab's (ML) third floor. Concerning the area, it is slightly bigger than the City Science laboratory (around 600 m²), but it has both long, featureless corridors and larger open areas.

GMAPPING IN ML

GMAPPING IN ML Since this is a more difficult environment, most of the parameters were varied in order to observe their effect on the environment. everything is summarized on Table 14.

Parameter	Range
Default	-
Linear update rate	[0.1-1.0]
Angular update rate	[0.1-1.0]
Iterations of the scanmatcher	[1-10]
Number of particles in the filter	[5-50]

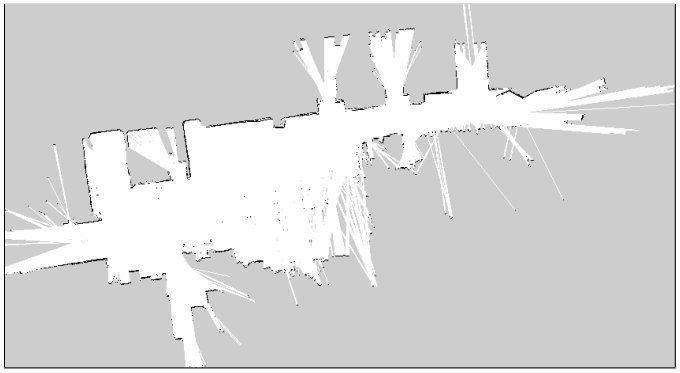
Table 14: Tests with Gmapping in ML third floor

As with the previous areas, first the default result will be shown (Figure 64).

Figure 64: ML with defaults for Gmapping

The result in general is acceptable, even though some of the edges are not perfectly matched. As it has been learnt from previous experiences, this effect can be improved by either increasing the linear/angular rates, the iterations of the scanmatcher and the particles in the filter. These results are shown in Figure 65. The differences are very subtle, but the best results are obtained by increasing the iterations (Figure 65(c)).

(a) Linear update at 0.1 m



(b) Angular update at 0.1 m

(c) 10 iterations for the scanmatcher

(d) 50 particles on the filter

Figure 65: Media Lab results with Gmapping

HECTOR IN ML For Hector mapping, parameters are described on Table 15. Results with and without odometry are the same in this case, therefore for the sake of space, results will be given without odometry.

Parameter	Range
Default	-
Multimap resolution	[2-3]
Linear update rate	[0.2-0.4]
Angular update rate	[0.4-0.9]

Table 15: Tests with Hector in ML third floor

The default result appears on Figure 66. In this case it can be noted that the first outcome with Hector SLAM seems slightly better than with Gmapping. Edges are more defined, there is less drift, fewer voids on the central region and the tables and chairs are captured by the mapping process.

Figure 66: Default result for Hector SLAM (ML)

The solutions proposed to improve the mapping of the third floor are shown on Figure 67 and the strategy is the same as before: increasing the update rate and maintaining map resolution in values of 3 and 2.

The clearest map is obtained with a linear update of 0.2 m, but the difference is barely noticeable. Features in all maps appear distinct enough and there is no drift on the edges.

The case of the third floor is the first where the results with Hector SLAM are better than with Gmapping. It could be inferred that it could be due to the fact that Gmapping does not scale properly when environments increase in size, but it is still possible to build reasonably good maps in this area.

(a) Linear update at 0.2 m

(b) Angular update at 0.4

(c) 50 particles on the filter

Figure 67: Results for different configurations in Hector (ML)

5

PERSUASIVE ELECTRIC VEHICLE (PEV)

PEV has already been presented in (Chapter 1), as a lightweight, low–cost, electric and autonomous platform, that would operate in the intersection of AVs (Figure 68) and bicycle sharing services.

Figure 68: PEV from the side

In this chapter, a more technical report of the vehicle will be done, as well as results from the SLAM on the different environments selected for the PEV.

DESCRIPTION OF THE VEHICLE

General specifications

The PEV weights 50 kg in vacuum is composed of of 2 main parts:

- **Steel base:** The white rickshaw shown in Figure 68 is made of stainless steel and weights 30 kg (including wheels). It connects to all three wheels and housess both motors and the battery.
- **Aluminum frame**: The exterior part of the PEV is made in aluminum sheets that weight in total 10 kg. The front part forms a curved shape and inside it houses the sensor batteries, power supply and connections.

The maximum load it could carry is around 120 kg which is enough to carry most people or a notable amount of packages, therefore it could accomplish its 2 main goals: people mover and package delivery.

The PEV would eventually operate on bike lanes, therefore speed is limited to 20 km/h, but the maximum speed the engine can achieve is up to 30 km/h.

The engine is fed with 10 Ah Lithium-Ion battery that can provide an autonomoy of 40 km. Sensors powered through a different system and can last for 1 hour without recharging.

Last, the PEV is provided with cellular communication so that it can communicate with remote machines and receive updates on the system.

The general details of the PEV are summarized on Table 16

Parameter	Value
Weight (kg)	50.0
Max. load (kg)	120.0
Max. speed (km/h)	30
Avg. speed (km/h)	20.0
Battery Capacity (Ah)	10.0
Autonomy (km)	40
Autonomy (h)	1
Communications	4G LTE cellular

Table 16: Specifications for the PEV

Hardware and sensors

In the following paragraphs, all the components that conform the system will be described:

Sensors

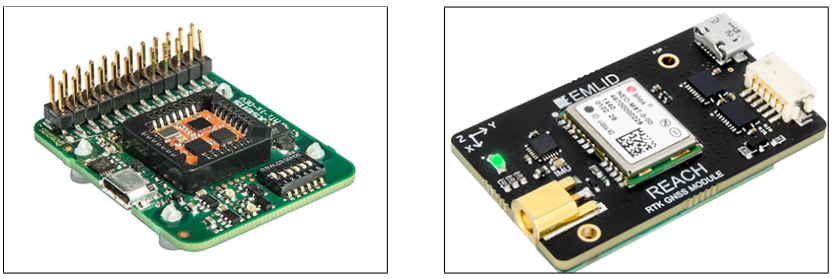
sensors The PEV has the following sensors:

- **Lidar** PEV has 3 lidars installed: 1 3D lidar and 2 2D lidars (Figure 69).
  - Velodyne VLP16: This 3D lidar has a 100 m range, with a 360°field of view and generally rotates at 600 rpm. it has 16 channels covering an angle of ±15°and generates 600.000 points per second. Tha main use of this lidar is to perform both SLAM and localization.
  - Hokuyo UTM-30LX: There are 2 lidars of this kind one located at 0°and the other tilted 30°. They are mainly employed for obstacle detection but they can be used as input for 2D SLAM algorithms. The field of view is of 270°producing 57.600 points per second.

(a) Velodyne VLP16 (b) Hokuyo UTM-30LX

Figure 69: Lidars installed on the PEV

- **Cameras** The PEV has 2 different types of cameras:
  - Monocular cameras: 4 Logitech C920 are located on top of the aluminum frame in such a way to have a 360°field of view. These cameras are used to perform object recognition and traffic light detection.
  - Stereo cameras: 2 ZED cameras are mounted on the PEV, one of the front and one on the back. They are used for short range recognition, lane detection and they provide odometry estimates.
- **Encoders**: The PEV has 2 encoders located on the back wheel and the steering bar. Their signals are then translated to speed commands according to the Ackermann steering equations.
- **Inertial Measurement Units**: Two Xsens MTi-1 IMUs (Figure 70(a)) are located on the front of the PEV, one horizontal to the floor plane and one tilted 30°downwards. These devices measure acceleration, orientation and magnetic field, and are fused with the encoders and GPS to improve the accuracy of state estimation.
- **GPS**: PEV has a Emlid Reach RTK GPS receiver with Real Time Kinematics (RTK) functionality and that achieves precisions of centimeters (Figure 70(b)).



(a) IMU (b) GPS receiver

Figure 70: Two types of sensors installed on the PEV

ACTUATORS The main actuators of the PEV have alreayd been defined: The **middrive** engine and the **steering** motor. Both are controlled by a **'Vedder' Electric Speed Control (VESC)** which is an open–source speed control just as shown in Figure 71. Apart from controlling motors, it allows to control servos and read from pulse encoders.

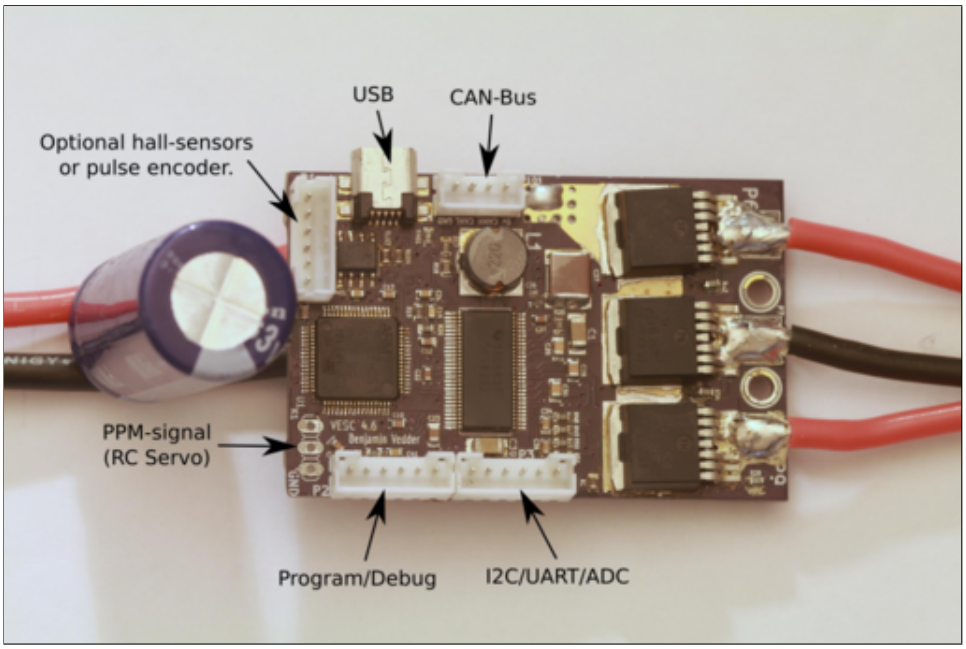


Figure 71: VESC electronic circuit

The parameters for both motors can be configured using the BLDC tool provided by the same company and shown in Figure 72. This software module also provides visualization tabs to check the state of the system.

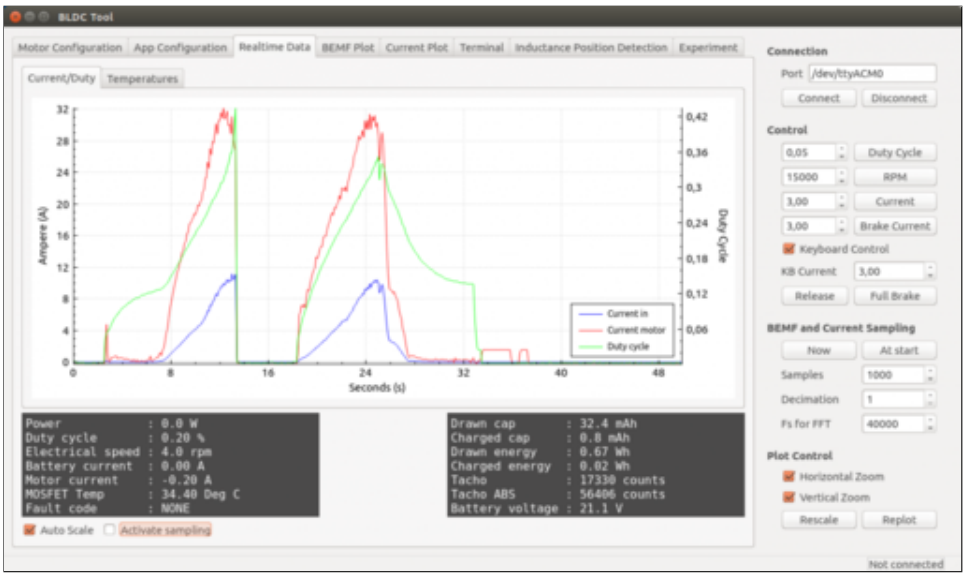


Figure 72: BLDC tool to configure VESC

In order to **brake**, the PEV also features a **servo motor** attached to the rear wheel that pulls the break when actuated. It is also controlled by the middrive VESC module.

Power supply

POWER SUPPLY As for the power system of the PEV, there are 2 main sources:

- **Lithium–Ion battery**: This battery powers the middrive engine, operates at 48 V and has 10 Ah of charge.
- **Lead–Acid batteries**: There are 4 12 V lead–acid batteries mounted on the PEV. 2 of them are mounted in series and are attached to the steering motor. The other 2 are connected in parallel and run the sensors, USB hubs and computers.

Computing

COMPUTING The 'brains' of the PEV are divided into:

- **Jetson TX2**: There is one Jetson TX2 that takes the input of the four cameras and publishes it on the system. To process large images at high speed, this Jetson TX2 is connected to a PCI express module with four USB 3.0 ports just as shown on Figure 73

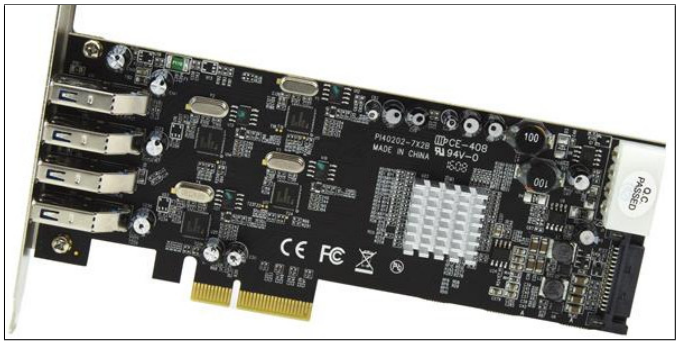


Figure 73: PCI express module

- **Laptop**: The main computer is a Dell Precision 5520 with an 3.0 GHz Intel Xeon E3-1505M CPU, 16 GB DDR4 RAM and Nvidia Quadro M1200 4 GB Graphics Processing Unit (GPU).

Communication

COMMUNICATION PEV has wireless communications thanks to an TP–Link TL-MR6400 router to which a generic SIM card can be attached, adding 4G LTE connectivity to the PEV.

SETUP

A conceptual overview of the PEV's setup is shown in Figure 74. It is more complex than the Nexus robot but it shares the same principles. Sensors are handled by the laptop except for the cameras, and the computing units send commands to the actuators.

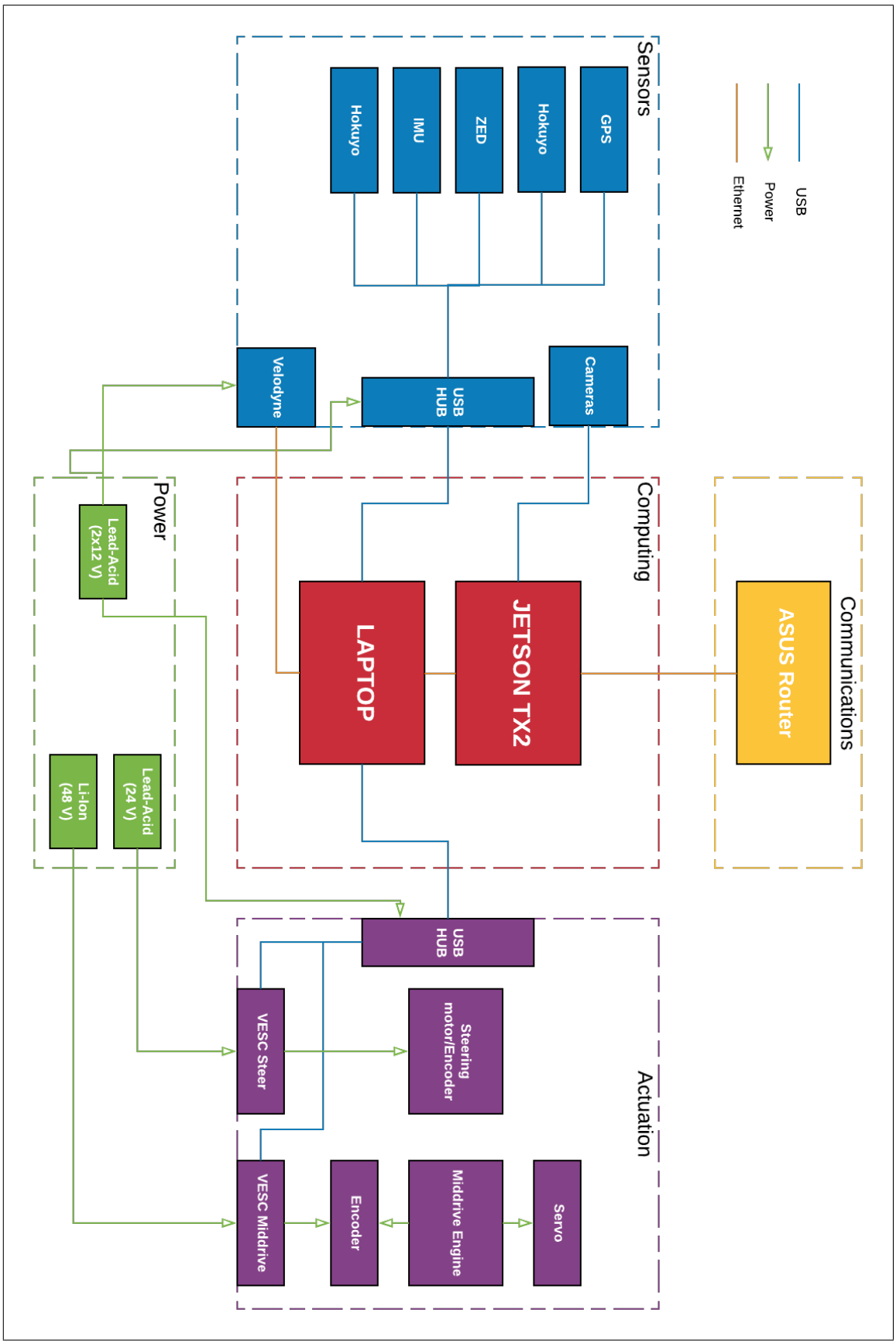


Figure 74: PEV setup conceptual overview

The **master** of the ROS system is the laptop, but due to the wired communication, each computing unit can share information over the network. Both units are connected to the the router which can receive commands from smartphones and other devices.

Physically, all these components are housed on the front part, as it was mentioned before. Components are divided into 4 layers, and all sensors' cables are routed to their corresponding layer:

- **First layer**: It houses the lead–acid batteries.
- **Second layer**: It houses an HVAC system that has not been mentioned because it is not in use, although it is on the road map.
- **Third layer**: Here, the USB hubs, ethernet switch and power boxes are located.
- **Fourth layer**: Places the Jetson TX2

Control

Control of the system is similar to the Nexus Robot, as speed commands come from 2 different sources: **Joystick** or **Navigation Stack**.

However, the control is more sophisticated, since the joystick can set the PEV in 3 different modes:

- **Free mode**: In this state the PEV does not listen to any speed command and can move freely.
- **Autonomous mode**: While no speed commands are received, the servo brakes the wheel. When it receives input from either the joystick or the navigation stack, it starts moving autonomously.
- **Emergency Braking**: In this mode, the servo brakes the wheel and the PEV does not take any speed commands.

Besides this, speed commands are not sent directly to the VESC modules, a **speed multiplexor** is used instead. The workflow of the multiplexor is as follows:

- **Inputs**: Navigation Stack and Joystick emit messages of the type `Twist` and `Joy`, so they are converted to `AckermannDriveStamped`.
- **Multiplexor**: Based on priority levels, the multiplexor outputs the speed command of the input with highest priority.
- **Ackermann to VESC node**: This node takes the speed command from the multiplexor, the joystick commands and the feedback from the VESC. If set in autonomous mode, it outputs one command to each of VESC modules, corrected with a PID controller.
- **VESC nodes**: Send the commands to the motors and feedback to the previous node in order to close the loop.

In Figure 75, an schematic of the whole control process is displayed.

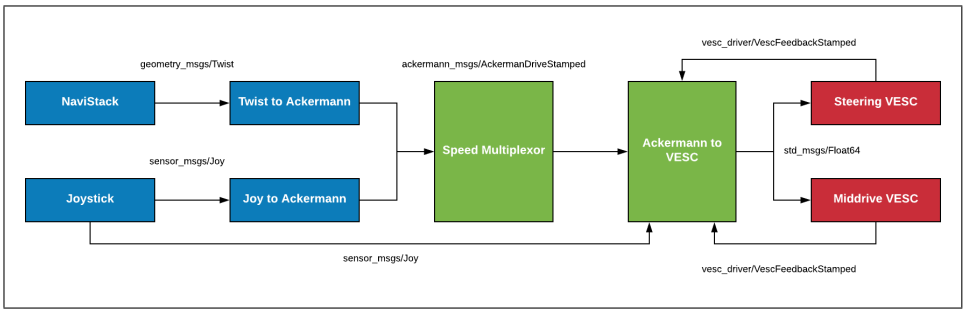


Figure 75: Schematic of PEV's control system

ROS package

The ROS packages for the PEV is described in Table 17:

Package	Description
firmware	Code for the different controllers the PEV has (i.e, Arduino, Raspberry Pi)
pev_autoware	Necessary packages and messages from Autoware to run NDT mapping and localization
pev_cmd_mux	Nodes of the multiplexor and its configuration files
pev_description	3D model of the PEV in .urdf format
pev_hardware_rule	Hardware rules of all the sensors of the PEV
pev_mapping	Configuration files for all the mapping algorithms
pev_move	Package to perform autonomous navigation. Contains nodes to send commands to motors, configuration files of localization algorithms and motion planners' parameters
pev_msgs	Custom services and messages for the PEV
pev_octomap	Package to perform 3D mapping. Deprecated
pev_odom	Nodes and configuration files to transform feedback from motors to odometry measurements
pev_path_server	Nodes to save and load predefined paths for autonomous navigation
pev_sensors	Launch files of all the sensors of the PEV

Table 17: ROS packages for the PEV

For SLAM, the most important packages are: **pev_mapping**, **pev_autoware**, **pev_description**, **pev_sensors** and **pev_odom**.

INDOOR/OUTDOOR SLAM WITH PEV

Three different scenarios were tested with the PEV: one indoor and 2 outdoor. The indoor scenario corresponds to **Media Lab's 6th floor**, where as the outdoor environments are **MIT Media Lab's courtyard** and **Taipei's Air Force Base**.

Because the data was collected in such diverse areas, the 4 SLAM algorithms have been employed (not all in all cases). Recalling from (Chapter 3) those algorithms are: **Gmapping**, **Hector SLAM**, **Google's Cartographer** and **Autoware**.

Media Lab 6th floor

The 6th floor of the Media Lab is very well-known in the Boston area for its impressive views and it is often used for all kinds of conferences and venues. It is a relatively large open–space (800 m^2), therefore it is ideal to test the PEV when new features are developed or when the weather conditions do not allow for outdoor testing.

GMAPPING ON 6TH FLOOR

GMAPPING ON 6TH FLOOR The first algortihm to be tested is the Gmapping algorithm. As it was done with the Nexus robot, several parameters were varied to find the optimal configuration (Table 18):

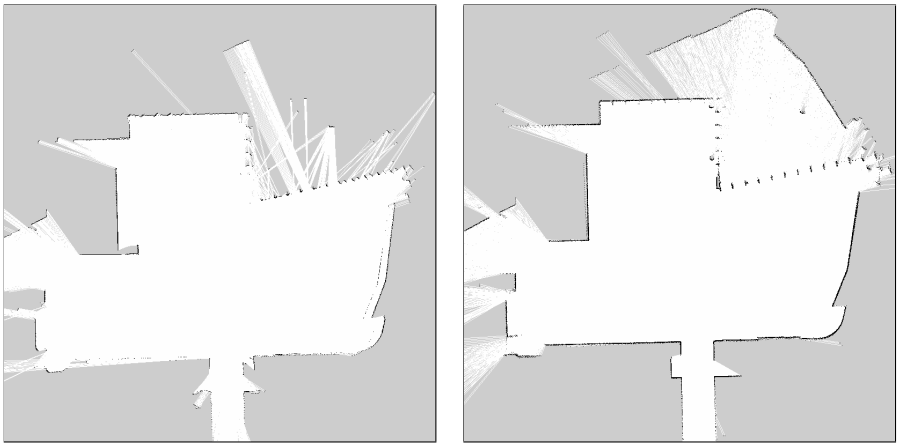
Parameter	Range
Default	-
Linear update rate	[0.1-1.0]
Angular update rate	[0.1-1.0]
Iterations for the scanmatcher	[0-10]

Table 18: Tests with Gmapping in the 6th floor

The results for Gmapping are shown in Figure 76. As it can be seen, there is not much difference in any of the cases studied. It is true that in the default case, edges of the map appear to be 'blurrier' but this issue is mitigated by applying the same reasoning as with the Nexus Robot.

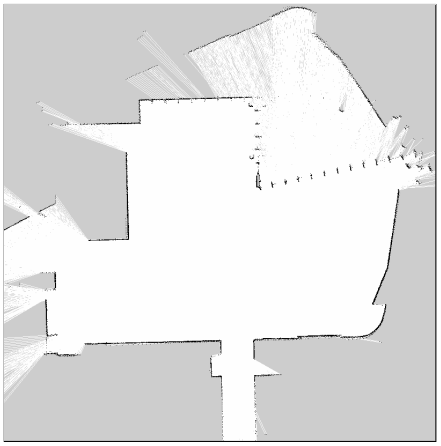
Generally speaking, the quality of the maps obtained is more than acceptable for autonomous navigation with only a little adjustment to be made on the upper right area (edges are not detected because doors are made of glass).

As a matter of fact, the odometry data (which is necessary to run the Gmapping algorithm) is not provided by the wheel and motor encoders, but by a ROS library that estimates odometry from laser [98]. As it can be seen, the quality of the estimation is good enough, since results are satisfactory.

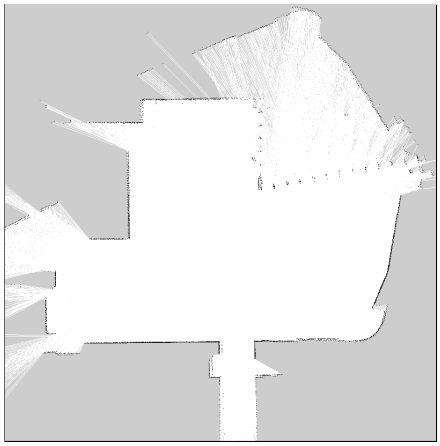


(a) Default values

(b) Linear update of 0.1 m



(c) Angular update of 0.1 rad



(d) 10 iterations

Figure 76: Gmapping on the 6th floor

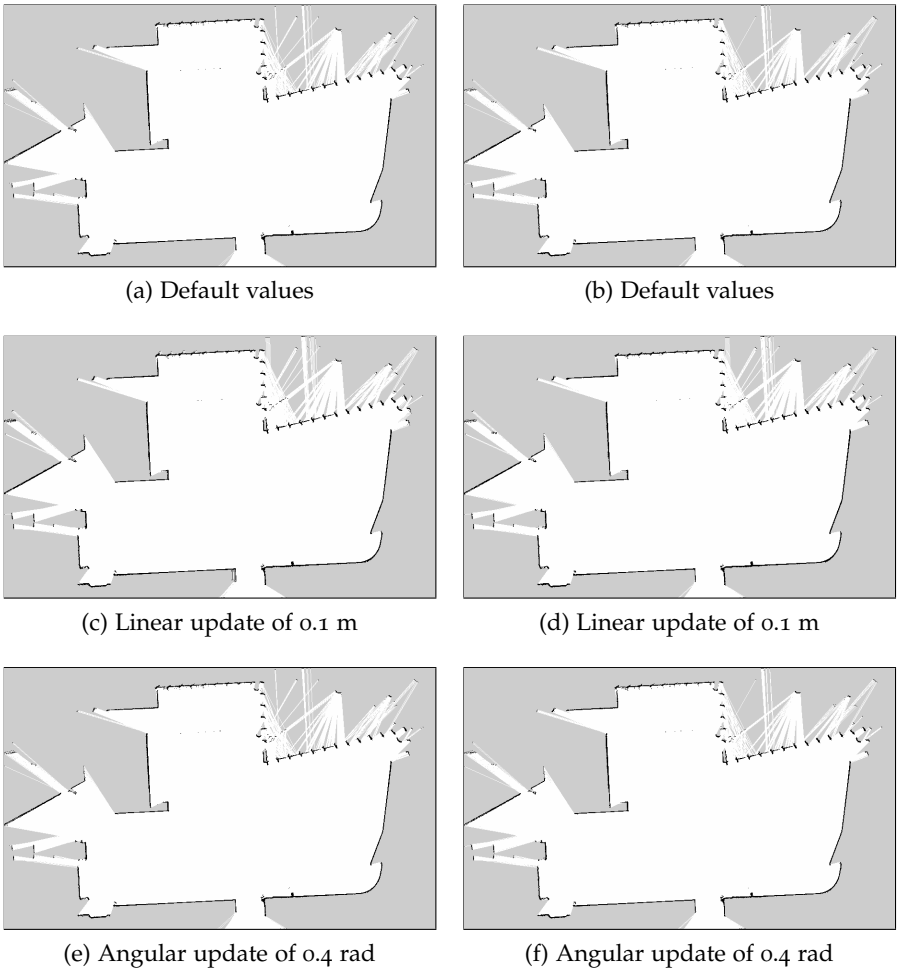
HECTOR ON 6TH FLOOR

HECTOR ON 6TH FLOOR Recalling from previous chapters, odometry is optional in Hector SLAM, and generally there is not much difference. For that reason, results are provided without using it. Table 19 summarizes the variations tested:

Parameter	Range
Default	-
Linear update rate	[0.1-1.0]
Angular update rate	[0.1-1.0]
Map resolution	2 & 3

Table 19: Tests with Hector in the 6th floor

The results (Figure 77), show that there is no appreciable difference in any of the cases tested. However, the accuracy is slightly better than Gmapping, as edges are sharper and there is less diffusion.



(a) Default values (b) Default values

(c) Linear update of 0.1 m (d) Linear update of 0.1 m

(e) Angular update of 0.4 rad (f) Angular update of 0.4 rad

Figure 77: Hector on the 6th floor. Map resolution 3 (left) and 2 (right)

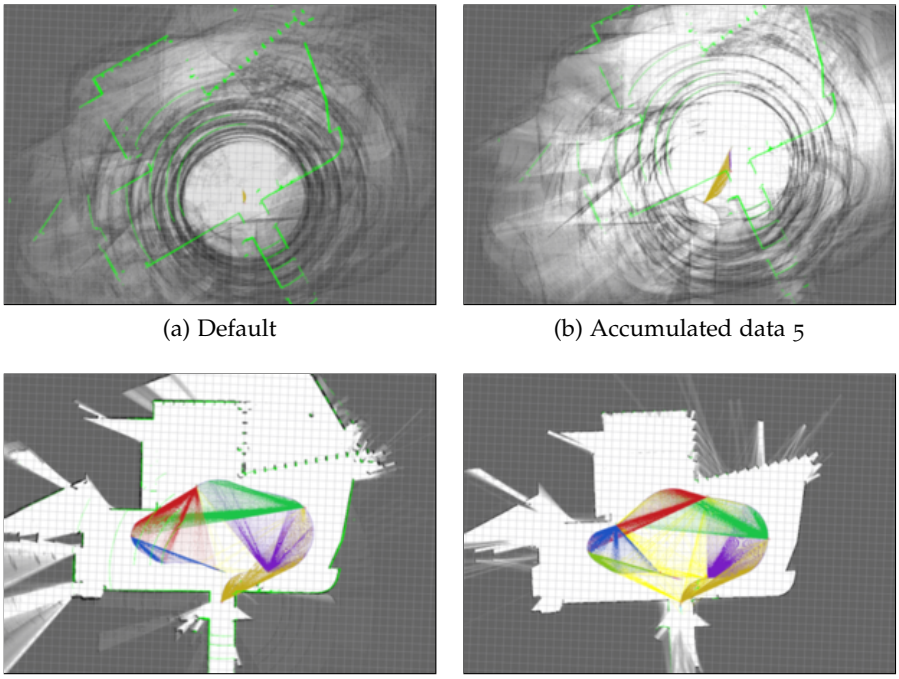
CARTOGRAPHER ON 6TH FLOOR

CARTOGRAPHER ON 6TH FLOOR Tuning Cartographer is not as straight forward as tuning Hector SLAM or Gmapping. The number of parameters is much larger and not every parameter is documented on the API. Furthermore, Cartographer can run with or without odometry and IMU when mapping in 2 dimensions.

The parameters that have been tuned are shown in Table 20. The default parameters are obtained from the `backpack_2d.lua` file provided by cartographer. Results with IMU are shown on Figure 78 and without it on Figure 79.

Parameter	**Range**
Default	-
Accumulated range data	[1-10]
Lidar type	[Velodyne and Hokuyo]

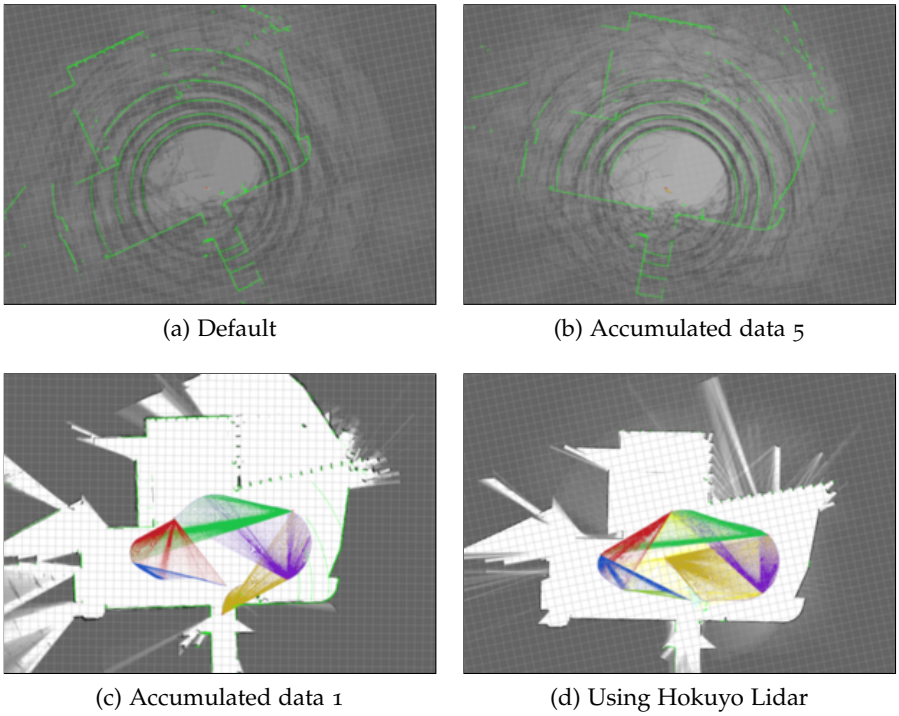
Table 20: Tests with Cartographer in the 6th floor



(a) Default (b) Accumulated data 5

(c) Accumulated data 1 (d) Using Hokuyo Lidar

Figure 78: Cartographer on the 6th floor (with IMU)



(a) Default (b) Accumulated data 5

(c) Accumulated data 1 (d) Using Hokuyo Lidar

Figure 79: Cartographer on the 6th floor (without IMU)

First thing to notice is that when accumulated laser data grows, the SLAM process results in incomprehensible maps. The difference of using the 3D Velodyne lidar or the 2D Hokuyo is the higer range of the former,

which results in more features mapped. Another aspect to point out is that there is virtually no difference between using IMU or relying solely on the scanmatcher. Finally, when using the 2D Lidar, it can be seen that the results are as good as with the previous approaches.

It can be concluded that for this environment Cartographer's theoretical advantage over Gmapping and Hector is unappreciable.

MIT Media Lab's Courtyard

On the back door of the Media Lab there is an extense courtyard that serves as an outdoor testing site for the PEV (Figure 80). The area is much larger than the previous studied environments (2000 m^2), although it is 'closed' in the sense that it does not communicate with the street directly. Therefore, it serves as a middleground between indoor and pure outdoor areas.

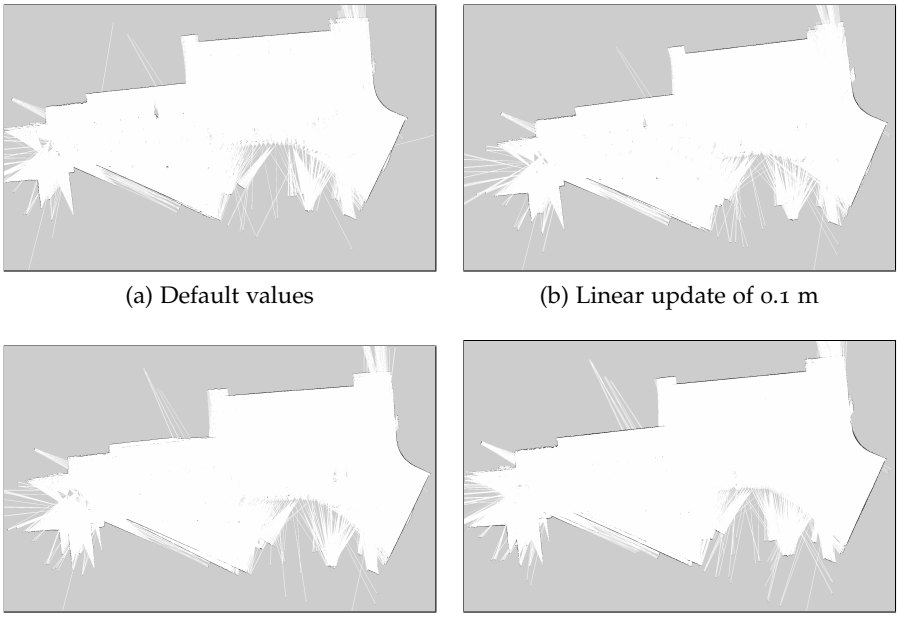
Figure 80: Satellite image of the courtyard

GMAPPING ON COURTYARD

GMAPPING ON COURTYARD The parameters that were varied in the courtyard are summarized on Table 21, and various results can be seen on Figure 81.

Parameter	Range
Default	-
Linear update rate	[0.1-0.5]
Angular update rate	[0.1-0.5]
Iterations for the scanmatcher	[0-10]

Table 21: Tests with Gmapping on the courtyard



(a) Default values (b) Linear update of 0.1 m

(c) Angular update of 0.2 rad (d) 10 iterations

Figure 81: Gmapping on the courtyard

None of the configurations tried througout the experiments improved the quality of the above shown maps. Although quality is not acceptable for autonomous navigation, the main shaoe of the area is conserved. The biggest gap that appears on the lower end of the map is due to the 30 m limit of the Hokuyo lidar.

With some manual adjustments, however, these maps could be utilized for localization.

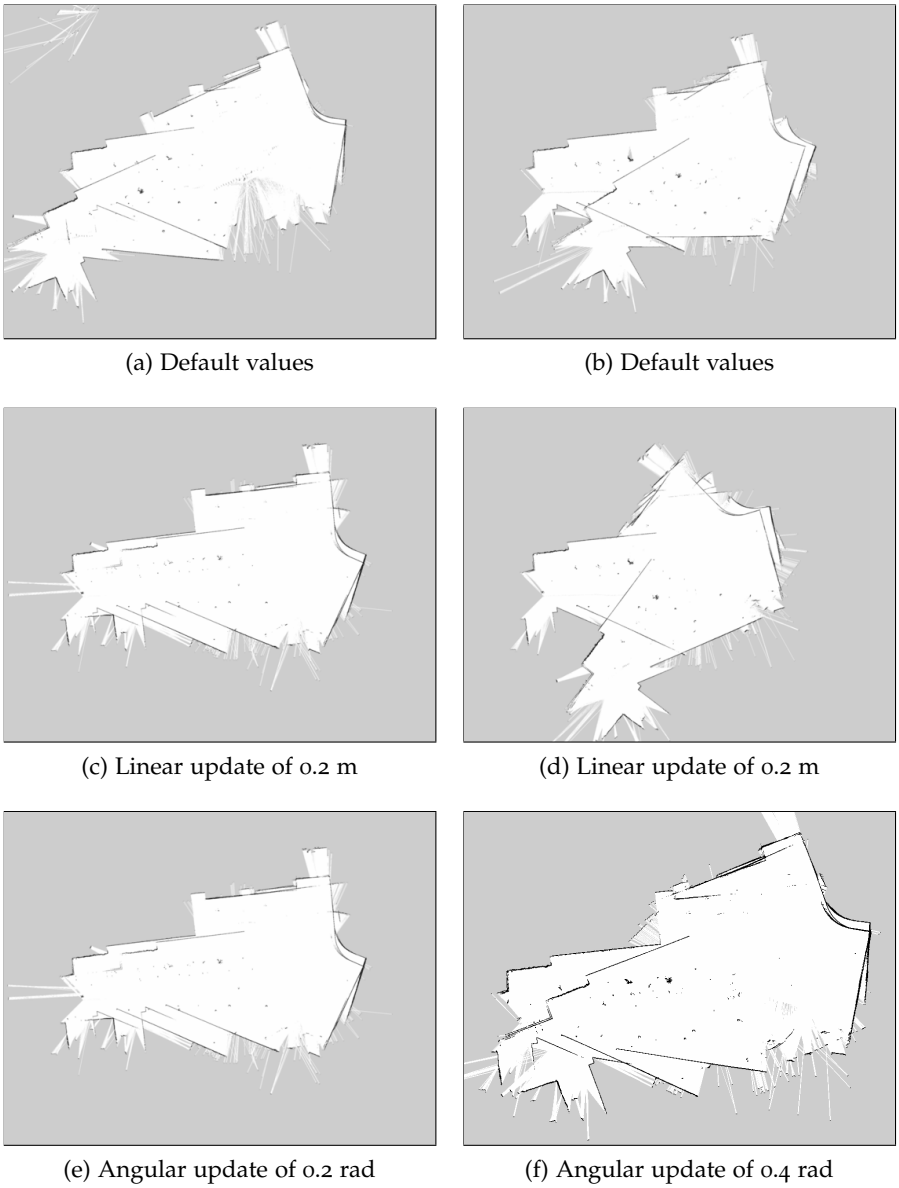
HECTOR ON COURTYARD

HECTOR ON COURTYARD With regards to employing Hector SLAM on the courtyard, Table 22 gathers the variations tested.

Parameter	Range
Default	-
Linear update rate	[0.1-1.0]
Angular update rate	[0.1-1.0]
Map resolution	2 & 3

Table 22: Tests with Hector on the courtyard

As it can be seen on Figure 82, none of the results for the Hector SLAM is acceptable, nor it could be manually adjusted. The reason why this might be ocurring is that Hector SLAM relies on the scan matcher but the central area of the courtyard is too wide for the 30 m laser scanner to find features. Therefore, the matching operation fails, leading to bad results. To improve this, it would be necssary a longer range sensor.



(a) Default values (b) Default values

(c) Linear update of 0.2 m (d) Linear update of 0.2 m

(e) Angular update of 0.2 rad (f) Angular update of 0.4 rad

Figure 82: Hector on the courtyard. Map resolution 3 (left) and 2 (right)

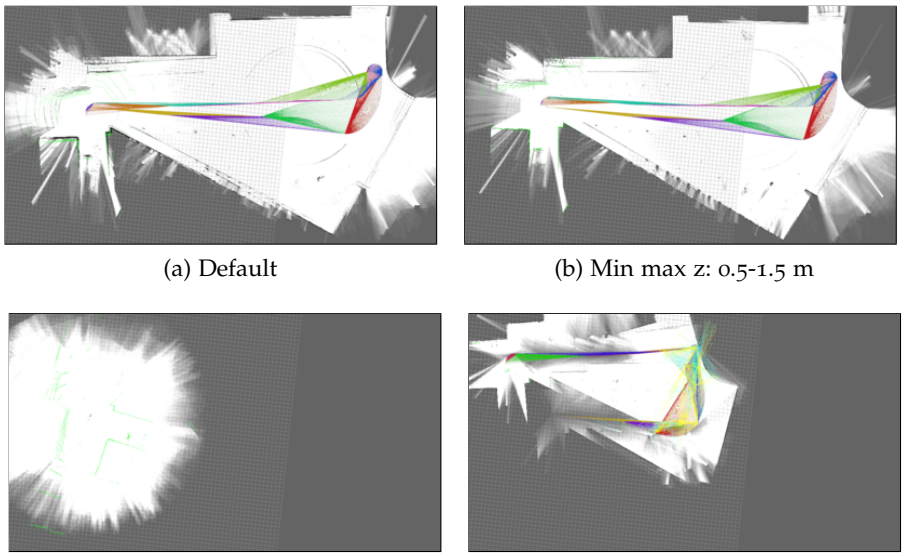
CARTOGRAPHER ON COURTYARD

CARTOGRAPHER ON COURTYARD With regards to Cartographer, the configurations tested can be checked on Table 23. Results with IMU and without are displayed on Figure 83 and Figure 84.

The configuration for the best result achieved in the 6th floor does not provide a perfect solution but it can serve as a starting point. The parameters that have the more direct influence on the quality of the map are the translational and rotational weights. On Figure 83(b) and Figure 84(b) these weights were set to 2 and 5, and yield very good results. Adding to that, the range of 0.5-1.5 m improves the speed of the computation and results in the most accurate maps for each case.

Parameter	Range
Default	-
Max z laser	[0.5-2.0]
Min z laser	[-0.5-1.0]
Translational weight	[2-200]
Rotational weight	[5-500]
Lidar type	[Velodyne and Hokuyo]

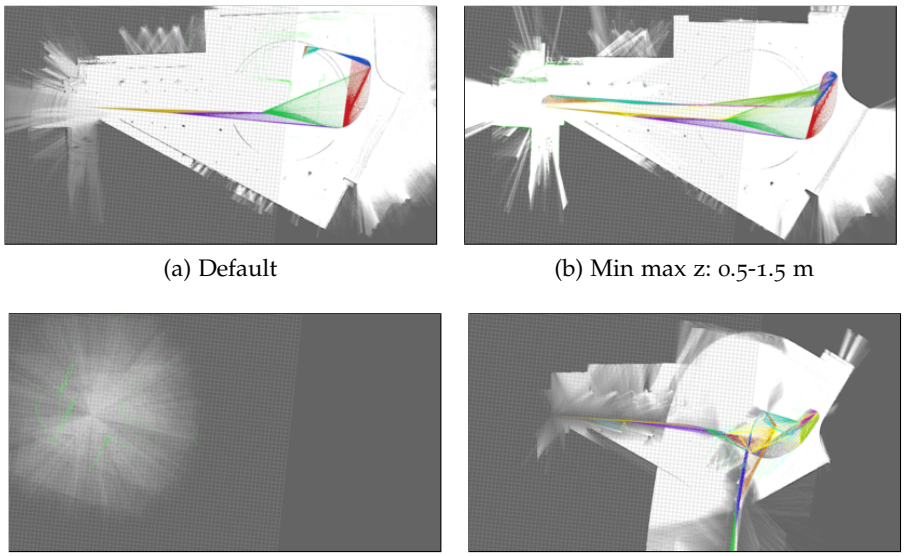
Table 23: Tests with Cartographer on the courtyard



(a) Default (b) Min max z: 0.5-1.5 m

(c) Trans/Rot. weights: 200/500 (d) Using Hokuyo Lidar

Figure 83: Cartographer on the courtyard (with IMU)



(a) Default (b) Min max z: 0.5-1.5 m

(c) Trans/Rot. weights: 200/500 (d) Using Hokuyo Lidar

Figure 84: Cartographer on the courtyard(without IMU)

In these tests the difference of using IMU and not using it is more notorius. However, what seems counterintuitive is that the best results are achieved when IMU is not considered. A possible explanation could be miscalibration of inertial sensors, but the lidar-only approach still provides more consistent results when using data from other days.

NDT ON COURTYARD

NDT ON COURTYARD NDT mapping does not allow to change any of its parameters, except for the use of IMU. The processing of the file took around 15 minutes and in that time the memory consumption of the algorithm was of 5 GB.

The output of NDT mapping is a **3D pointcloud**, with a considerable point density (~80 MB), as shown in Figure 85 and Figure 86.

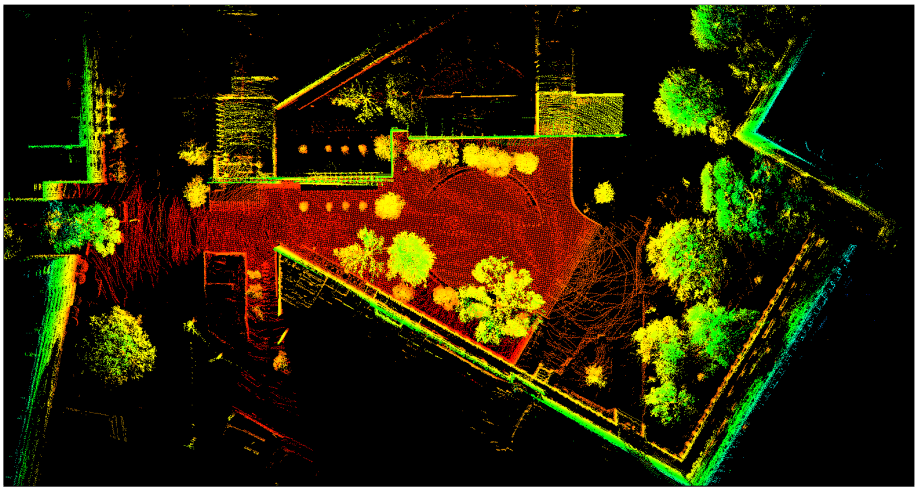


Figure 85: NDT mapping on the courtyard

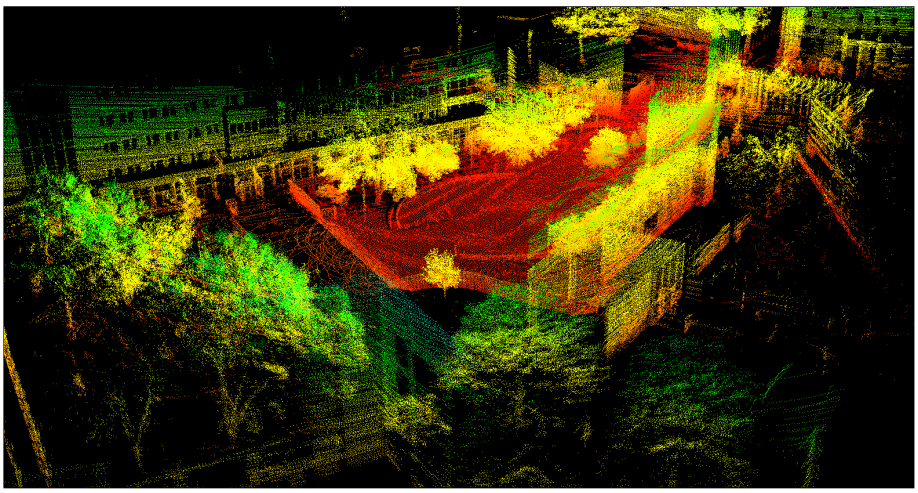


Figure 86: Detail of the result of NDT mapping

When performing localization with these pointclouds, a voxel grid filter will need to be applied. The tool developed for that purpose can be found on Github.

Taipei's Air Force Base (TAF)

Taipei's Air Force used to have a 1 km² area in the city of Taipei, Taiwan for their military operations, but in the last decaade it was transformed into an innovation space. With the permission of Taipei's government, the PEV was allowed to test on the area shown in Figure 87

Figure 87: Aerial view of TAF

The specific region recorded is the front part, since it features bike lanes and pedestrian walks, thus making it an interesting area to test how the vehicle adapts to urban real–life scenarios.

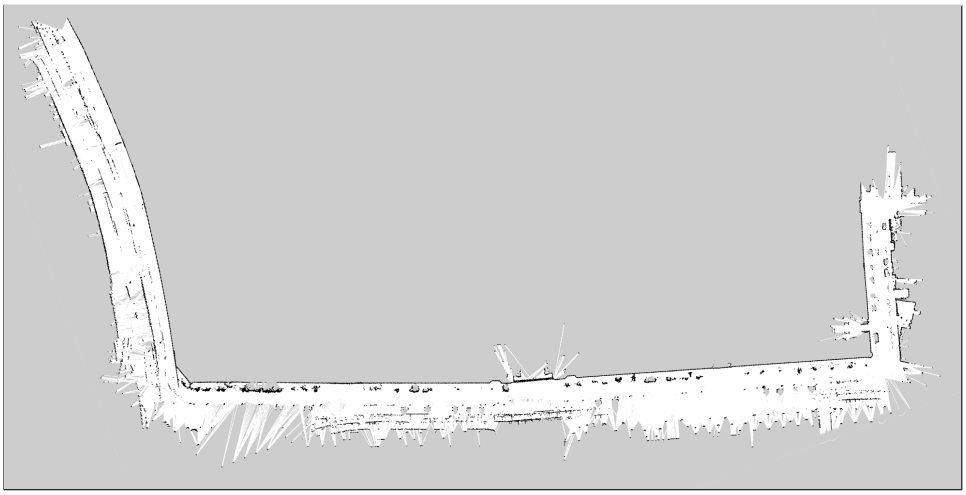
HECTOR ON TAF

HECTOR ON TAF As in previous cases, Table 24 summarizes the tested approaches:

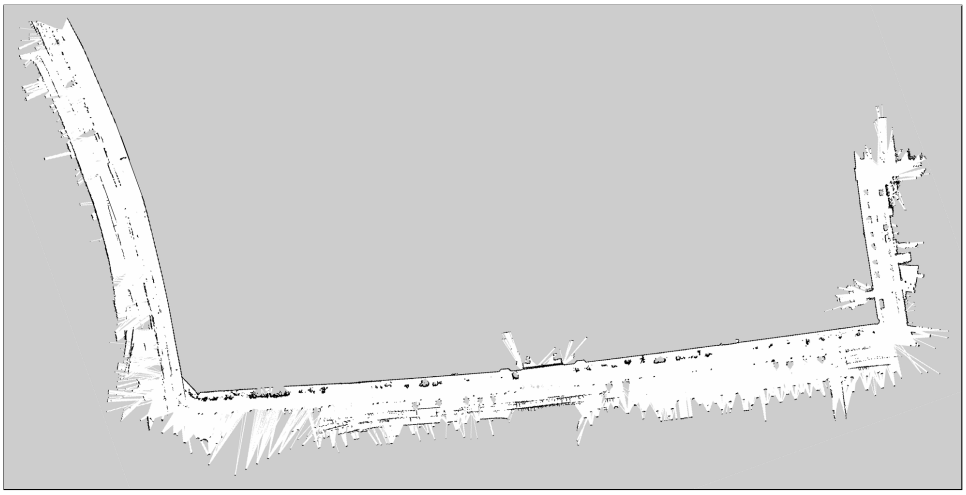
Parameter	Range
Default	-
Linear update rate	[0.1-1.0]
Angular update rate	[0.1-1.0]

Table 24: Tests with Hector on TAF

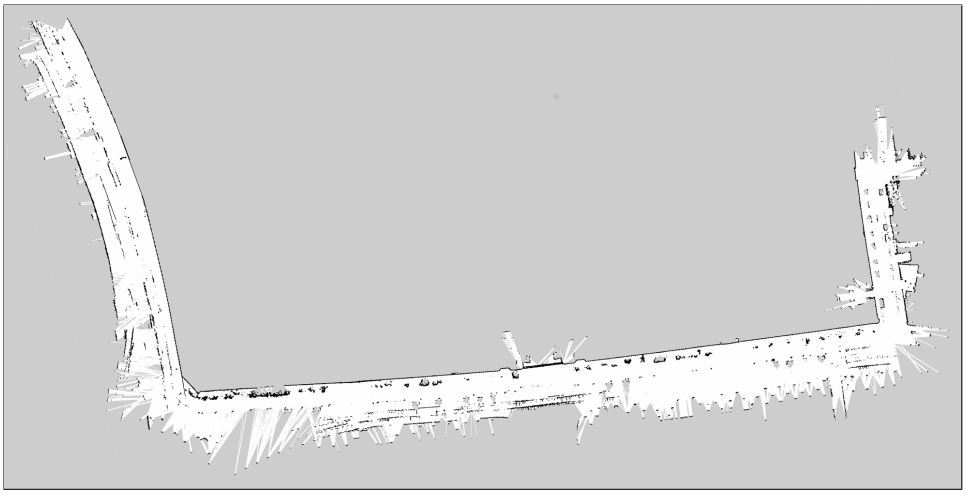
In Figure 88, results of the SLAM are provided. It is notable the quality of the maps obtained in this case compared to the courtyard. This is due to the fact that even though it is a longer path, the PEV is always surrounded by feature–rich environments, thus resulting in low localization errors.



(a) Default values



(b) Linear update of 0.2 m

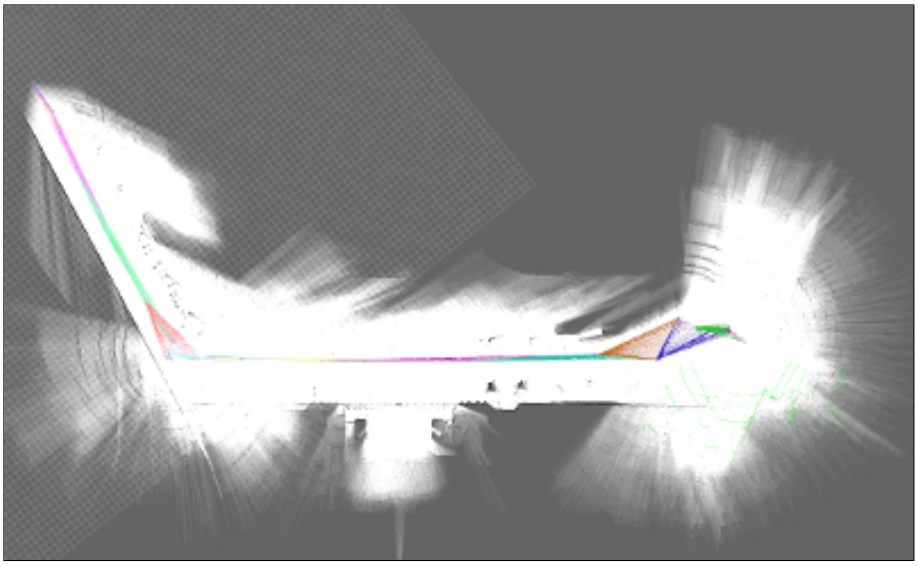


(c) Angular update of 0.4 rad

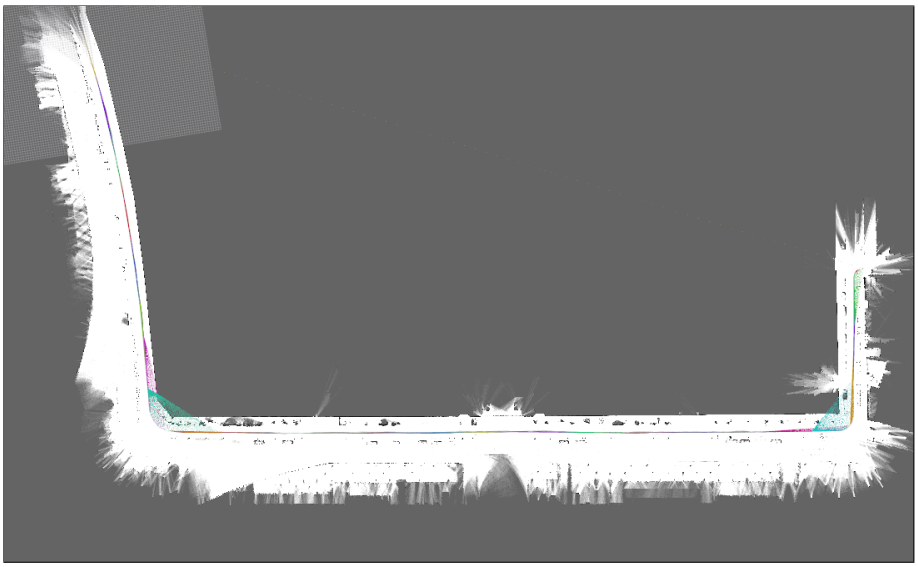
Figure 88: Hector on TAF

The default map has a small error on the turn of the left part, which is shortened with regards to the actual road, but it is mitigated by increasing the linear and angular update rates.

CARTOGRAPHER ON TAF With regards to Cartographer, the reader might have noticed that it is more sensible to parameter changes than Hector or Gmapping. For TAF, the best combination of parameters achieved on the courtyard was kept and the results are shown in Figure 89



(a) Best result with IMU



(b) Best result without IMU

Figure 89: Cartographer on TAF

Again, the best results are achieved without IMU. And the reason that could explain this is the IMU discalibration, but nothing was seen on the trajectories with or without inertial measurement.

NDT ON TAF Lastly, the results for NDT mapping on TAF are shown on Figure 90 and Figure 91. Each map took 1 hour to process, taking up 5 GB of the system memory. The resulting pointclouds had a size of ~350 MB so in order to display them, a voxel grid filter had to be applied as mentioned previously.

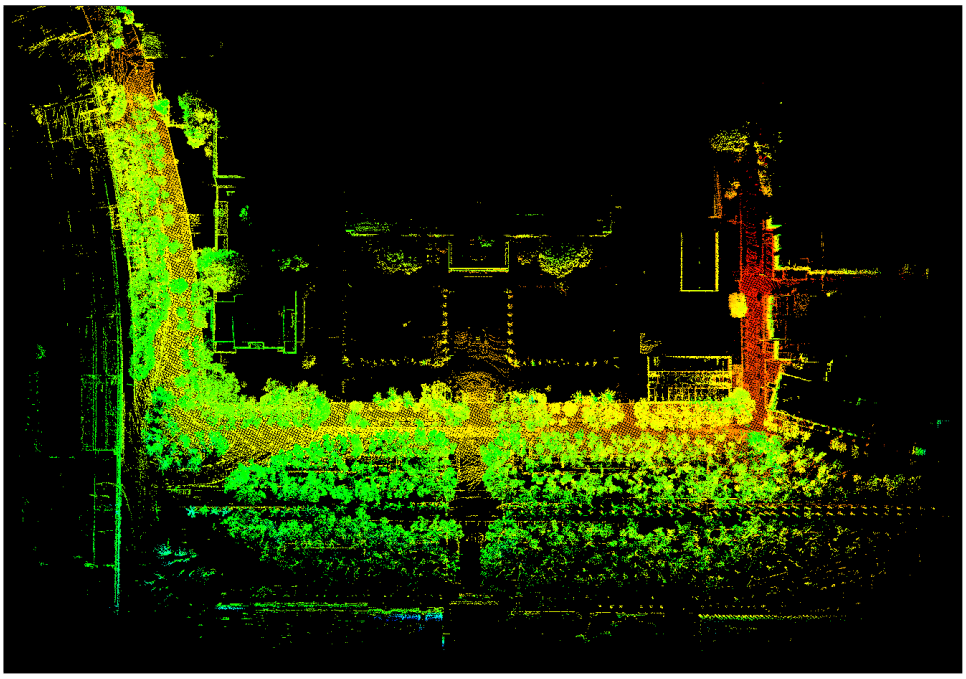


Figure 90: NDT mapping on TAF

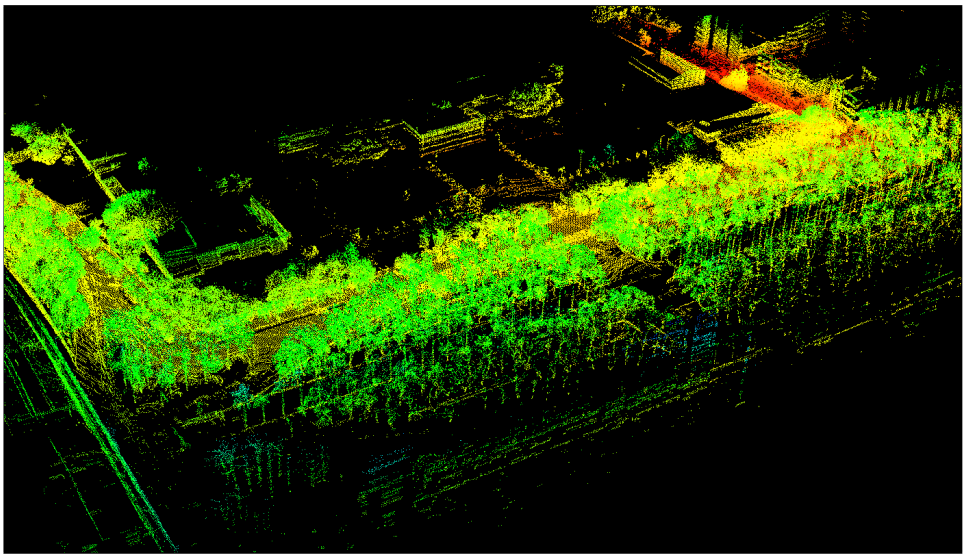


Figure 91: Detail of the result of NDT mapping

6

PROJECT COSTS

This section covers the cost of the main components to develop this thesis, namely: hardware and sensors; software and labour.

HARDWARE

Nexus robot

The cost of materials for the Nexus robot is shown in Table 25:

Component	Site	Unit price ($/h)	Qty	Total ($)
Nexus Robot	Taobao	500	1	500
Arduino Mega	Arduino	35	1	35
Jetson TX2	Nvidia	610	1	610
RPLidar A2	Seedstudio	319	1	319
Hokuyo UST-10LX	Robotshop	1,680	1	1,680
NiMH battery	Micro Center	20	3	60
USB hub 3.0	Amazon	50	1	50
Total				**3,254**

Table 25: Cost of the nexus Robot

PEV

The main structure of the PEV was already built so there are no machining costs associated to the fabrication process. The main expenditures were sensors,a s shown on Table 26:

Component	Site	Unit price ($/h)	Qty	Total ($)
Velodyne VLP16	Velodyne	8,000	1	8,000
Hokuyo UTM-30LX	Hokuyo	4,770	2	9,540
Xsens MTi-1 IMU	Xsens	399	2	798
ZED Stereo camera	Stereolabs	449	2	898
Logtech C930-e camera	Robotshop	85	4	340
Emlid Reach RTK GPS	Emlid	570	1	570

(Continued from previous page)

Component	Site	Unit price ($/h)	Qty	Total ($)
VESC	Amazon	75	2	150
Jetson TX2	Nvidia	610	1	610
Dell Precision 5520	Dell	2,949	1	2,949
Lead–acid battery	MicroCenter	115	4	460
Total				**24,315**

Table 26: Cost of the PEV

SOFTWARE

All of the software utilized was open–source and free, therefore it was costless. Nevertheless, it is listed in Table 27 ofr documentation purposes:

Name	Developer
Ubuntu 16.04 LTS	Canonical
Robot Operating System (ROS)	OSRF
OpenCV	OpenCV
Tensorflow	Google
CUDA	Nvidia
Autoware	CPFL

Table 27: Software employed

LABOR

Labor costs are estimated from the annual salaries of the human labor involved in the development of the project (Table 28).

Labor	Salary ($/h)	Qty (h)	Total ($)
Research assistant	12	960	11,520
Principal Investigator	85	10	850
Total			**12,370**

Table 28: Human labor cost

TOTAL COST

Assuming a 5 year amortization rate, for the sensors and equipment for the vehicles, the cost of the project for the period is shwon on (Table 29):

Field	Total ($)	Amortization (yr)	Per-year ($)
Hardware			
- Nexus Robot	3,254	5	650
- PEV	24,315	5	4,863
Software	0	0	0
Labor	12,370	0	12,370
Total			**17,883**

Table 29: Total cost for the period

7

CONCLUSIONS AND FUTURE IMPROVEMENTS

In this chapter, the main conclusions drawn from the experiments will be presented, as well as future improvements and plans for the platforms.

CONCLUSIONS ON SLAM

The main conclusion for the SLAM algorithmms is that there is not a 'one size fits all' solution. Each one performs better under a range of all the possible conditions.

Gmapping has proven accurate and robust enough in most indoor environments, without requiring barely any parameter variations. However, the fact it relies on odometry makes it complicated to use when that information is not available or it does not have good quality.

Hector SLAM overcomes the problem of odometry by employing a more accurate scanmatcher. The issue that the algorithm has is that when the number of features decreases, this overdependency on the scanmatcher causes the algorithm to provide bad results, just as it was seen on the courtyard.

Google's Cartographer offers a complete solution for both problems of SLAM and localization, thus being in advantage over the previous approaches, as they only perform SLAM. Tuning the Cartographer is generally more difficult and can take up days to achieve acceptable quality maps. However, when key parameters are hit, solutions tend to be better than with Gmapping or Hector SLAM.

Another feature Cartographer has is the possibility of 3D mapping, but since tuning is even more time consuming, and results do not vary significantly from 2D SLAM (at least for the PEV), it has not been explored in-depth.

NDT mappping is a very powerful tool and its result are unmatched when it comes to outdoor scenarios. For the PEV to ever hut the road, it seems that the correct approach will be algorithms like NDT SLAM.

The general conclusion for the SLAM process is that the quality of the laser sensor has the most noticeable impact on the algorithms. With modern scanmatching techniques, it is possible to build maps by only utilizing lidars and neglecting the rest of sensors.

Among lidars, it has been shown that higher ranges and 3D architectures tend to perform better at the cost of higher computing requirements. Nev-

ertheless, this century has already seen a major improvement in computing power and this trend will likely continue upwards.

One of the sensors that has not been used is the RTK GPS (generally reserved for localization tests), but in the future, it will be integrated within the SLAM workflow. Fortunately, both Cartographer and NDT mapping are capable of integrating GPS information, so transition will presumably be easier.

CONCLUSIONS ON NEXUS AND PEV

Looking at the cost for both platforms, it seems that the purpose of building low-cost platforms is out of reach for the moment. However, this high prices is due to 2 reasons:

- They both are **research projects**, therefore sensors and hardware has to be requested on demand. If they were to be produced in mass, price would go down considerably.
- Another fact to point out is the high price for **Lidar**, which accounts for 60% of the price on the Nexus robot and 70% of the PEV. Until very recently, lidars had a narrower market share but the development of AV companies is expected to lower down their price. In fact, the price of the Velodyne VLP16 was already cut down to 4,000 $.

HOW DOES FUTURE LOOK LIKE?

Considering the SLAM problem, an interesting branch that it is not fully developed yet is **multi robot mapping**. Despite the technical difficulties of this approach, it is very interesting since it would allow for faster and broader mapping, and that is a key factor when operating in large outdoor environments.

Another problem that has to be tackled is large pointcloud management. Currently the maximum size for a message to be passed on the ROS system is of 1 GB, which is not enough to hold the map of a city. That is why new approaches need to be developed if the PEV starts operating in large urban environments.

As for the PEV and the Nexus Robot, there are many plans in the roadmap for both.

The former will continue to be developed along with new platforms, in order to explore new manners of educating students on autonomous vehicles, and expanding new concepts of in-office delivery. One of the experiments that will be developed on the next months is a deployment of a number of these robots to be used as the delivery system of the City Science group.

With regards to the PEV some of these plans are:

- The next months will see trials of the PEV to navigate autonomously in Taipei, Japan and Boston.
- Since social acceptance is a key factor for the deployment of AVs, a great deal of research around the PEV is focused on developing **Human-Machine Interactions**.
- When the PEV was concieved, it was determined that it would not be a closed-form solution. In fact. the core idea of team has always been to provide a framework for building **autonomous lightweight 3-wheeled vehicles**. Part of the work these days is focused on identifying the key components and modularizing the system, so that anyone in the future can build a version of the 'PEV'.

The number of possibilities does not end here, and it depends mostly on how far society can look into the future of Autonomous Vehicles and its lightweight branches. The aim of developing the PEV is not to impose it over everything else, but to provoke a debate over new and more sustainable means of transport or and think about how to make cities attractive places for their dwellers.

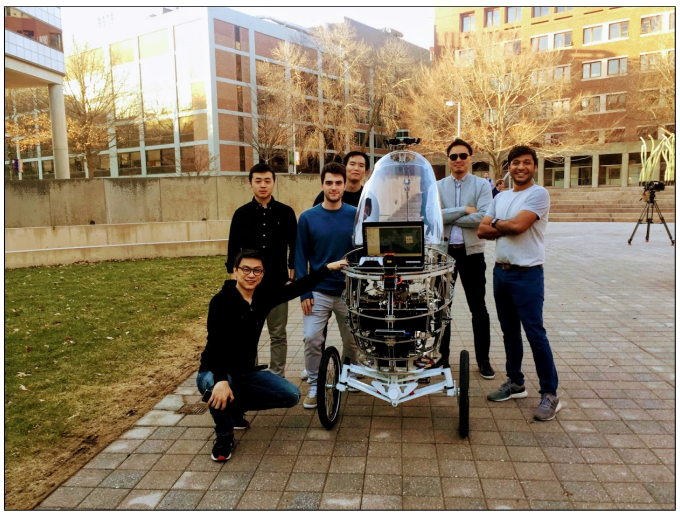


Figure 92: Alone we can do so little, together we can do so much

BIBLIOGRAPHY

CHAPTER 01

[1] Alex Davies. The Wired guide to Self-Driving Cars. *Wired Magazine, Conde NAST, www. wired. com*, 2018.

[2] Andrew Hawkins. Tesla's Autopilot is supposed to deliver full self-driving, so why does it feel stuck in the past? *The Verge*, oct 2017.

[3] Prateek Bansal and Kara M Kockelman. Forecasting Americans' long-term adoption of connected and autonomous vehicle technologies. *Transportation Research Part A: Policy and Practice*, 95:49–63, 2017.

[4] Mark Bergen. Alphabet Launches the First Taxi Service With No Human Drivers. *Bloomberg*, nov 2017.

[5] Keshav Bimbraw. Autonomous Cars: Past, Present and Future - A Review of the Developments in the Last Century, the Present Scenario and the Expected Future of Autonomous Vehicle Technology. *Proceedings of the 12th International Conference on Informatics in Control, Automation and Robotics*, (January 2015):191–198, 2015.

[6] Andrea Caragliu, Chiara del Bo, and Peter Nijkamp. Smart cities in Europe. *Journal of Urban Technology*, 18(2):65–82, 2011.

[7] Charlie Campbell. The Trouble with Sharing: China's Bike Fever Has Reached Saturation Point. *Time*, 2018.

[8] Paul DeMaio. Bike-sharing: History, impacts, models of provision, and future. *Journal of public transportation*, 12(4):3, 2009.

[9] Daniel J. Fagnant and Kara Kockelman. Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations. *Transportation Research Part A: Policy and Practice*, 77:167–181, jul 2015.

[10] Daniel J. Fagnant and Kara M. Kockelman. The travel and environmental implications of shared autonomous vehicles, using agent-based model scenarios. *Transportation Research Part C: Emerging Technologies*, 40:1–13, mar 2014.

[11] Margaret Galer Flyte. The safe design of in–vehicle information and support systems: the human factors issues. *International journal of vehicle design*, 16(2-3):158–169, 1995.

[12] Herbert Girardet. Cities, People, Planet. *John Wiley & Sons*, (I):1–15, 2008.

[13] Yoshimasa Goto and Anthony Stentz. Mobile robot navigation: The CMU system. In *IEEE expert*. Citeseer, 1987.

[14] Arnaud Grignard, Núria Macià, Luis Alonso Pastor, Ariel Noyman, Yan Zhang, and Kent Larson. Cityscope andorra: a multi-level interactive and tangible agent-based visualization. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1939–1940. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[15] Chana J Haboucha, Robert Ishaq, and Yoram Shiftan. User preferences regarding autonomous vehicles. *Transportation Research Part C: Emerging Technologies*, 78:37–49, 2017.

[16] L Hook and R Waters. Google's Waymo passes milestone in driverless car race. *Financial Times. https://www. ft. com/content/dc281ed2-c425-11e7-b2bb-322b2cb39656. Accessed*, 28, 2018.

[17] Daniel Howard. Public Perceptions of Self-driving Cars: The Case of Berkeley, California. *MS Transportation Engineering*, 2014(1):21, 2014.

[18] Jacobson, Joseph M., Paul S. Drzaic, and Ian D Morrison. Electrophoretic displays using nanoparticles, mar 2001.

[19] Kent Larson. Brilliant designs to fit more people in every city. *TED Talk*, 2012.

[20] Timothy B. Lee. Fully driverless cars could be months away. *Arstechnica*, oct 2017.

[21] Robert D Leighty. DARPA ALV (Autonomous Land Vehicle) Summary. Technical report, 1986.

[22] Todd Litman. Autonomous Vehicle Implementation Predictions: Implications for Transport Planning. *Transportation Research Board Annual Meeting*, (2014):36–42, 2014.

[23] Anna Mikołajczyk. Transportation problems of contemporary cities and opportunities to solve the problems through innovative management. *World Scientific News*, 72:482–487, 2017.

[24] Christian Paromtchik, Igor; Laugier. Autonomous parallel parking of a nonholonomic vehicle. In *Proceedings of Conference on Intelligent Vehicles*, pages 13–18, 1996.

[25] Iyad Rahwan, Jean-Francois Bonnefon, and Azim Shariff. Moral Machine. *Moral Machine. Np*, 2016.

[26] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, and Brian Silverman. Scratch: programming for all. *Communications of the ACM*, 52(11):60–67, 2009.

[27] R Rowe. Self-Driving Cars, Timeline, 2015.

[28] Pericle Salvini. Urban robotics: Towards responsible innovations for our cities. *Robotics and Autonomous Systems*, 100:278–286, 2018.

[29] Brandon Schoettle and Michael Sivak. Public Opinion About Self-Driving Vehicles in China, India, Japan, The U.S., The U.K. and Australia. (UMTRI-2014-30 (October)):1–85, 2014.

[30] Guna Seetharaman, Arun Lakhotia, and Erik Philip Blasch. Unmanned vehicles come of age: The DARPA grand challenge. *Computer*, 39(12), 2006.

[31] Susan Shaheen, Stacey Guzman, and Hua Zhang. Bikesharing in Europe, the Americas, and Asia: past, present, and future. *Transportation Research Record: Journal of the Transportation Research Board*, (2143):159–167, 2010.

[32] Hossein Shahrokni, Bram van der Heijde, David Lazarevic, and Nils Brandt. Big Data GIS Analytics Towards Efficient Waste Management in Stockholm. *Proceedings of the 2014 conference ICT for Sustainability*, (Ict4s):140–147, 2014.

[33] Nick Shchetko. Laser eyes pose price hurdle for driverless cars. *The Wall Street Journal*, 21, 2014.

[34] Tom Simonite. Prepare to be Underwhelmed by 2021's Autonomous Cars. aug 2016.

[35] Society of Automotive Engineers. Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles. 2018.

[36] Alireza Talebpour and Hani S Mahmassani. Influence of connected and autonomous vehicles on traffic flow stability and throughput. *Transportation Research Part C: Emerging Technologies*, 71:143–163, 2016.

[37] Cadie Thompson. These are the tech features you should get in your next car. *Business Insider*, 2017.

[38] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, and Gabriel Hoffmann. Stanley: The robot that won the DARPA Grand Challenge. *Journal of field Robotics*, 23(9):661–692, 2006.

[39] Elisabeth Uhlemann. Connected-Vehicles Applications Are Emerging [Connected Vehicles]. *IEEE Vehicular Technology Magazine*, 11(1):25–29, 2016.

[40] U.S. Department of and National Highway Traffic Safety Administration Transportation. National Motor Vehicle Crash Causation Survey Report to Congress. (July):1–47, 2008.

[41] World Health Organization. Road traffic injuries, 2018.

[42] M Xie, L Trassoudaine, J Alizon, M Thonnat, and J Gallice. Active and intelligent sensing of road obstacles: Application to the European

Eureka-PROMETHEUS project. In *Computer Vision, 1993. Proceedings., Fourth International Conference on*, pages 616–623. IEEE, 1993.

[43] Guang-Zhong Yang, Jim Bellingham, Pierre E. Dupont, Peer Fischer, Luciano Floridi, Robert Full, Neil Jacobstein, Vijay Kumar, Marcia McNutt, Robert Merrifield, Bradley J. Nelson, Brian Scassellati, Mariarosaria Taddeo, Russell Taylor, Manuela Veloso, Zhong Lin Wang, and Robert Wood. The grand challenges of Science Robotics. *Science Robotics*, 3(14):eaar7650, 2018.

[44] Yang, Yuang. Bike sharing in China: Ofo, Mobike and the lure of two wheels. *Financial Times. https://www. ft. com/content/dc281ed2-c425-11e7-b2bb-322b2cb39656. Accessed*, 2018.

[45] K Zavitsas, I Kaparias, M G H Bell, and M Tomassini. Transport problems in cities. *ISIS*, 6:5, 2010.

CHAPTER 02

[46] Markus Bernard, Konstantin Kondak, Ivan Maza, and Anibal Ollero. Autonomous transportation and deployment with aerial robots for search and rescue missions. *Journal of Field Robotics*, 28(6):914–931, 2011.

[47] J Borenstein, H R Everett, L Feng, and D Wehe. Mobile Robot Positioning Sensors and Techniques. *Journal of Robotic Systems, Special Issue on Mobile Robots*, 14(4):231–249, 1997.

[48] Johann Borenstein, H R Everett, and Liqiang Feng. Where am I? Sensors and methods for mobile robot positioning. *University of Michigan*, 119:120, 1996.

[49] Wolfram Burgard. Introduction to Mobile Robotics: Robot Control Paradigms. page 21, 2017.

[50] Wolfram Burgard. Introduction to Mobile Robotics: Wheeled Locomotion. page 29, 2017.

[51] Wolfram Burgard. Probabilistic Sensor Models. 2017.

[52] Wolfram Burgard. Proximity Sensors. 2017.

[53] Wolfram Burgard. Robot Motion Planning. page 151, 2017.

[54] Peter Corke. *Robotics , Vision*.

[55] John J. Craig. Introduction to Robotics: Mechanics and Control 3rd. *Prentice Hall*, 1(3):408, 2004.

[56] Raffaello D'Andrea. Guest editorial: A revolution in the warehouse: A retrospective on Kiva Systems and the grand challenges ahead. *IEEE Transactions on Automation Science and Engineering*, 9(4):638–639, 2012.

[57] Alex Davies. What Is Lidar, Why Do Self-Driving Cars Need It, and Can It See Nerf Bullets? 2018.

[58] Thor I. Fossen, Kristin Y Pettersen, and Henk Nijmeijer. *Sensing and Control for Autonomous Vehicles*. 2017.

[59] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[60] David Gossow, Adam Leeper, Dave Hershberger, and Matei Ciocarlie. ROS topics: Interactive markers: 3-D user interfaces for ROS applications. *IEEE Robotics and Automation Magazine*, 18(4):14–15, 2011.

[61] John P. Grotzinger. Analysis of Surface Materials by the Curiosity Mars Rover X-ray Diffraction Results from Soil Diversity and Hydration at Gale Crater , Mars. *Science*, 341(September):2012–2014, 2013.

[62] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, 3:2149–2154, 2004.

[63] Steven M. LaValle. Planning algorithms. *Planning Algorithms*, 9780521862:1–826, 2006.

[64] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[65] Eitan Marder-Eppstein, Eric Berger, Tully Foote, Brian Gerkey, and Kurt Konolige. The office marathon: Robust navigation in an indoor office environment. In *International Conference on Robotics and Automation*, 2010.

[66] H. Menouar, I. Guvenc, K. Akkaya, A.S. Uluagac, A. Kadri, and A. Tuncer. UAV-enabled intelligent transportation systems for the smart city: Applications and challenges. *IEEE Communications Magazine*, 55(3):22–28, 2017.

[67] Jason M. O'Kane and Jason M. O. Kane. *A gentle introduction to ROS*. 2013.

[68] Morgan Quigley, Eric Berger, and Andrew Y Ng. Stair: Hardware and software architecture. In *AAAI 2007 Robotics Workshop, Vancouver, BC*, pages 31–37, 2007.

[69] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Mg. ROS: an open-source Robot Operating System. *Icra*, 3(Figure 1):5, 2009.

[70] Morgan Quigley, Brian Gerkey, and William D Smart. *Programming Robots with ROS: a practical introduction to the Robot Operating System*. " O'Reilly Media, Inc.", 2015.

[71] Marc Raibert. BigDog, the rough-terrain quadruped robot. *IFAC Proceedings Volumes (IFAC-PapersOnline)*, 17(1 PART 1):6–9, 2008.

[72] Roland Siegwart and Illah R Nourbakhsh. *Introduction to Autonomous Mobile Robots*, volume 23. 2004.

[73] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[74] Elizabeth Woyke. Roomba to Rule the Smart Home. *MIT Technology Review*, dec 2017.

[75] Keenan A Wyrobek, Eric H Berger, H F Machiel Van der Loos, and J Kenneth Salisbury. Towards a personal robotics development platform: Rationale and design of an intrinsically safe personal robot. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 2165–2170. IEEE, 2008.

CHAPTER 03

[76] Sameer Agarwal, Keir Mierle, and Others. Ceres solver. http://ceres-solver.org.

[77] Tim Bailey and Hugh Durrant-Whyte. Simultaneous localization and mapping (SLAM): Part II. *IEEE Robotics & Automation Magazine*, 13(3):108–117, 2006.

[78] Peter Biber and Wolfgang Straßer. The normal distributions transform: A new approach to laser scan matching.

[79] Gary Bishop and Greg Welch. An introduction to the Kalman filter. *Proc of SIGGRAPH, Course*, 8(27599-3175):59, 2001.

[80] Wolfram Burgard. Simultaneous Localization and Mapping. 2017.

[81] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: Part I. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[82] Giorgio Grisetti, Rainer Kummerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based SLAM. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, 2010.

[83] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improving grid-based SLAM with Rao-Blackwellized particle filters by adaptive proposals and selective resampling. *Proceedings - IEEE International Conference on Robotics and Automation*, 2005:2432–2437, 2005.

[84] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with Rao-Blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, 2007.

[85] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-Time Loop Closure in 2D LIDAR SLAM. pages 1271–1278, 2016.

[86] Shinpei Kato, Eijiro Takeuchi, Yoshio Ishiguro, Yoshiki Ninomiya, Kazuya Takeda, and Tsuyoshi Hamada. An open approach to autonomous vehicles. *IEEE Micro*, 35(6):60–68, 2015.

[87] Shinpei Kato, Shota Tokunaga, Yuya Maruyama, Seiya Maeda, Manato Hirabayashi, Yuki Kitsukawa, Abraham Monrroy, Tomohito Ando, Yusuke Fujii, and Takuya Azumi. Autoware on board: enabling autonomous vehicles with embedded systems. In *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*, pages 287–296. IEEE Press, 2018.

[88] Stefan Kohlbrecher, Stefan Kohlbrecher, Oskar Von Stryk, Johannes Meyer, and Uwe Klingauf. A flexible and scalable SLAM system with full 3D motion estimation A Flexible and Scalable SLAM System with Full 3D Motion Estimation. (November):0–5, 2011.

[89] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. *Proc. of 8th National Conference on Artificial Intelligence/14th Conference on Innovative Applications of Artificial Intelligence*, 68(2):593–598, 2002.

[90] Paul Newman and Kin Ho. SLAM-loop closing with visually salient features. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 635–642. IEEE, 2005.

[91] Cyrill Stachniss. Extended Information Filter. 2014.

[92] Cyrill Stachniss. Extended Kalman Filter SLAM. 2014.

[93] Cyrill Stachniss. Introduction to Robot Mapping. 2014.

[94] Cyrill Stachniss. Least Squares Approach to SLAM. 2014.

[95] Cyrill Stachniss. Short Introduction to Particle Filters and Monte Carlo Localization. 2014.

[96] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[97] David R. Tobergte and Shirley Curtis. *Handbook of Robotics*, volume 53. 2013.

CHAPTER 05

[98] Mariano Jaimez, Javier G. Monroy, and Javier González-Jiménez. Planar odometry from a radial laser scanner. a range flow-based approach. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4479–4485, 2016.

A

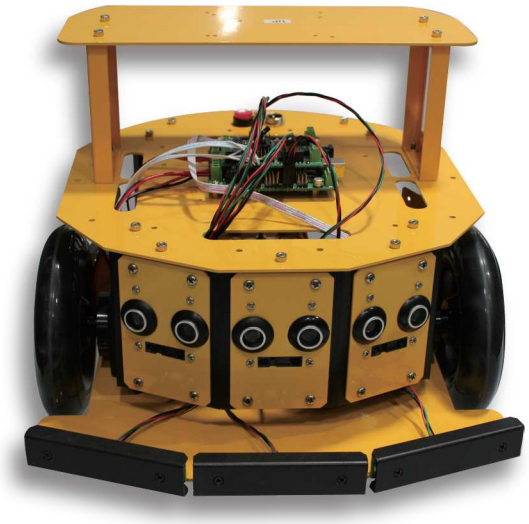
SPECIFICATION SHEETS

In the following pages, the specification sheets of the sensors and modules employed is provided:

- Nexus Robot.
- RPLidar A2.
- Hokuyo UST-10LX.
- Jetson TX2 module.
- Velodyne VLP 16.
- Hokuyo UTM-30LX.
- Xsens MTi-1.
- Vedder ESC (VESC).

NEXUS robot

2WD mobile robot kit 10004



This robot kit provides an economical introduction to the world of robotics. It has 2 drive wheels and a freewheel. It includes a serials of sensors making it aware of the surrounding: Sonar sensors to detect the obstructions, IR distance measure sensors used as a fall-arrest detector, bumper sensors to make it turn around while run into obstruction in its way. It is based on Arduino microcontroller. Its aluminum alloy body is firm enough to be mounted with extension equipments.

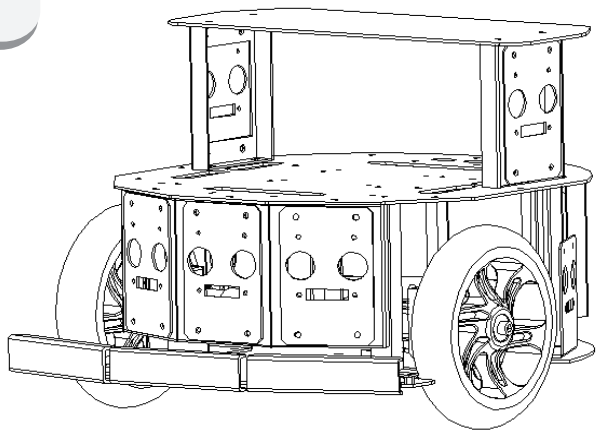
Aluminum alloy frame | Fall detect sensor | Ultrasonic sensor | IR sensor | Encoder | Progarmmable | Easily expand

Features:

- 2 wheels drive
- Aluminum alloy body
- IR sensors and ultrasonic sensors mounted as range finder
- Fall detect sensors equipped to avoid fall
- Bumper sensors to prevent collision
- Still spare mounts for sensors
- Easily expand with pre-drilled holes
- Second and third plates for extended components

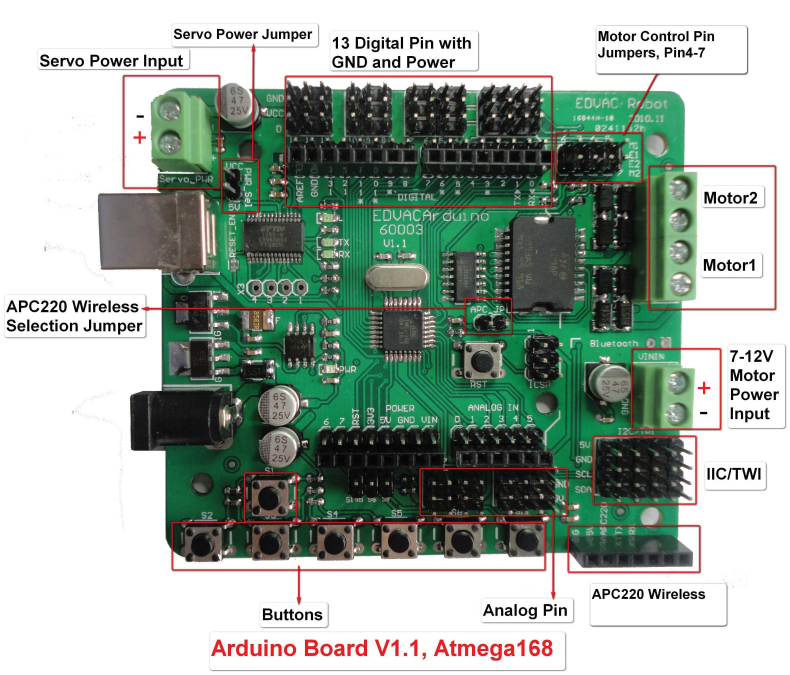
Parts included:

- 135mm PU Wheel X 2
- Faulhaber 12V DC Coreless Motor X 2
- Arduino 328 ControllerX1
- Arduino IO Expansion X1
- Ultrasonic Range Finders X 3
- GP2Y0A21 Infrared Sensor X3
- Bumper Sensors X 3
- Fall Detect IR Sensors X 2
- 12V Ni-Mh BatteryX1
- 12V ChargerX1



NEXUS robot

2WD mobile robot kit 10004



Specifications:

Chassis	Appearance	Circular
	Length	357mm
	Width	313mm
	Height	267mm
	Chassis Height	39mm
	Wheel Tread	280mm
	Wheel Base	160mm
	Coupled Mode	Compaction
	Material	Aluminum
	Color	Yellow, Black
	Speed	0.8m/s
	Drive Mode	2 wheels drive, 1 directional
	Climbing Capacity	20degree
	Load Capacity	10kg
	min-ITX Compatible	Yes
Wheel	Diameter	143mm
	Thickness	28mm
	Material	PU

Motor	Type	Faulhaber 12V DC Coreless Motor
	Power	17W
	RPM	120rpm
	Diameter	30mm
	Length	42mm
	Total Length	85mm
	Diameter of Shaft	6mm
	Length of Shaft	35mm
	No Load Current	75m
	Load Current	1400mA
	Gearbox Ratio	64:1
Encoder	Type	Optical
	Encoder Phase	AB
	Encoder Resolution	12CPR
Battery and Charger	Battery	12V Ni-Mh
	Slow Charger	100~240V In, 2.4~12V Out
	Duration of Charge	2 hours
	Running Time	0.5 hour

Microcontroller Specification	Atmega 328
	14 Channels Digital I/O
	6 PWM Channels (Pin11,Pin10,Pin9,Pin6,Pin5,Pin3)
	8 Channels 10-bit Analog I/O
	USB interface
	Auto sensing/switching power input
	ICSP header for direct program download
	Serial Interface TTL Level
	Support AREF
	Support Male and Female Pin Header
	Integrated sockets for APC220 RF Module
	Five IIC Interface Pin Sets
	Two way Motor Drive with 2A maximum current
	7 key inputs
	DC Supply:USB Powered or External 7V~12V DC
	DC Output:5V /3.3V DC and External Power Output
	Dimension:90x80mm
IO expansion board	To support RS485 interface or dirve 4 motors

SLAMTEC

2017-05-15 rev.1.0

RPLIDAR A2

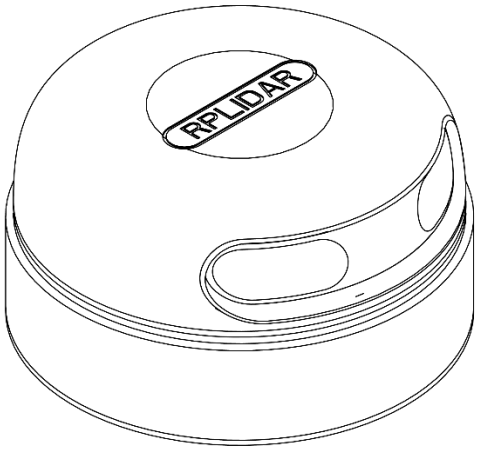
Low Cost 360 Degree Laser Range Scanner

Introduction and Datasheet

Model: A2M7

A2M8

OPTMAG 4K



www.slamtec.com

Measurement Performance

- For Model A2M7/A2M8 Only

Item	Unit	Min	Typical	Max	Comments
Distance Range	Meter(m)	0.15	-	8	Based om white objects with 70% reflectivity
Angular Range	Degree	-	0-360	-	-
Distance Resolution	mm	-	<0.5 <1% of the distance	-	<1.5 meters All distance range*
Angular Resolution	Degree	0.45	0.9	1.35	10Hz scan rate
Sample Duration	Millisecond(ms)	-	0.25	-	-
Sample Frequency	Hz	2000	4000	4100	
Scan Rate	Hz	5	10	15	The rate is for a round of scan. The typical value is measured when RPLIDAR takes 400 samples per scan

Figure 2-1 RPLIDAR Performance

Note: the triangulation range system resolution changes along with distance.

Laser Power Specification

- For Model A2M7/A2M8 Only

Item	Unit	Min	Typical	Max	Comments
Laser wavelength	Nanometer(nm)	775	785	795	Infrared Light Band
Laser power	Milliwatt (mW)	-	3	5	Peak power
Pulse length	Microsecond(us)	60	87	90	-
Laser Safety Class	-	-	FDA Class I	-	-

Figure 2-2 RPLIDAR Optical Specification

Note: the laser power listed above is the peak power and the actual average power is much lower than the value.

Date: 2016.06.20

Scanning Laser Range Finder
Smart-URG mini
UST-10LX
Specification

CE
RoHS

Symbol	Amended Reason	Pages	Date	Amended by	Ref.No
△2×3	Improved document quality.	3,4,6	2016.06.20	Y.Kamioka	RS-00629
△1×1	Product code no deleted.	1	2015.03.02	T.Kasahara	RS-00554

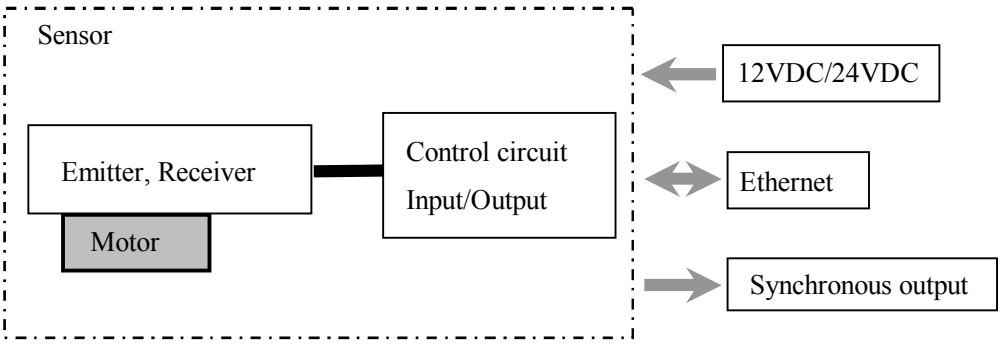
Approved by	Checked by	Drawn by	Designed by	Title	UST-10LX Specification	
T.Kamitani	M.Maeda	Y.Kamioka	A.Yamamoto	Drawing No.	C-42-04077	1/6

1. General

This sensor uses a laser source to scan 270° field of view. Positions of objects in the range are calculated with step angle and distance. Sensor outputs these data through communication channel.

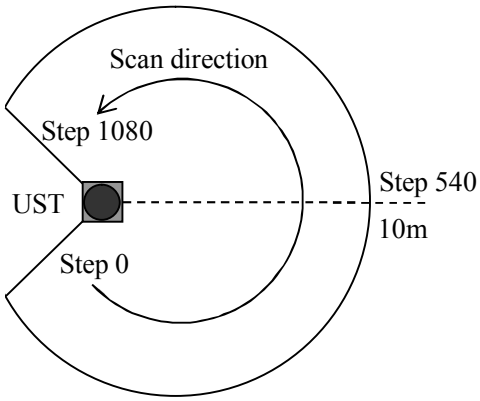
2. Structure

2-1.Strucure diagram



2-2. Laser scanning image

Measurement steps 1081
Detection angle 270°
Angular resolution 0.25°



3. Important notes

(1) This sensor is not a safety device/tool.

(2) This sensor cannot be used for human body detection as per the machinery directives.

(3) Hokuyo products are not developed and manufactured for the use in weapons, equipments or related technologies intended for destroying human lives or causing mass destruction. If such possibilities or usages are revealed, the sales of Hokuyo products to those customers might be halted by the laws of Japan such as Foreign Exchange Law, Foreign Trade Law or Export Trade control order. In addition, Hokuyo products are for the purpose of maintaining the global peace and security in accordance with the above law of Japan.

4. Specifications

Product name	Scanning Laser Range Finder
Model	UST-10LX
Supply voltage	12VDC/24VDC (Operation range 10 to 30V ripple within 10%)
Supply current △2	150mA or less (When using DC24V) (during start up 450mA is necessary.)
Light source	Laser semiconductor (905nm) Laser class 1 (IEC60825-1:2007)
Detection range	0.06m to 10m (white Kent sheet) 0.06m to 4m (diffuse reflectance 10%) Max. detection distance : 30m
Accuracy	±40mm (*1)
Repeated accuracy	σ< 30mm (*1)
Scan angle	270°
Scan speed	25ms (Motor speed 2400rpm)
Angular resolution	0.25°
Start up time	Within 10 sec (start up time differs if malfunction is detected during start up)
Input	IP reset input, photo-coupler input (current 4mA at ON)
Output	Synchronous Output, photo coupler open collector output 30VDC 50mA MAX.
Interface	Ethernet 100BASE-TX
LED display	Power supply LED display (Blue): Blinks during start up and malfunction state.
Surrounding intensity	Less than 15,000lx Note : Avoid direct sunlight or other illumination sources as it may cause sensor malfunction
Ambient temperature and humidity	-10°C to +50°C, below 85%RH (without dew, frost)
Storage temperature and humidity	-30°C to +70°C, below 85%RH (without dew, frost)
Vibration resistance	10 to 55Hz double amplitude of 1.5mm for 2hrs in each X, Y, and Z direction 55 to 200Hz 98m / s^2 sweep of 2min for 1hr in each X,Y and Z direction
Vibration resistance (Operating)	55 to 150Hz 19.6m / s^2 sweep of 2min for 30min in each X,Y and Z direction
Shock resistance	196m/s^2 (20G) X,Y and Z direction 10 times.
EMC standards	(EMI) EN61326-1:2013 EN55011:2009 + A1:2010 (EMS) EN61326-1:2013 EN61000-4-2:2009 EN61000-4-3:2006 + A1:2008 + A2:2010 EN61000-4-4:2012 EN61000-4-6:2009 EN61000-4-8:2010
Protective Structure	IP65
Weight	130g (Excluding cable)
Material	Front case: Polycarbonate, Rear case: Aluminum
Dimensions (W×D×H)	50×50×70mm (sensor only)

(*1) Under the factory standard testing conditions using white Kent sheet.

5. Measurement Data

Distance Value (x)	Meaning
x < 21	Output numerical number “4” as Measurement error
21 ≦ x ≦ 30000	Valid distance [mm]
x > 30000	Output numerical number “65533” as Measurement error (object does not exists or object has low reflectivity)

6. Connection

6-1. Power source, I/O cable

Cable length: 1000mm Flying lead cable (AWG28)

Color	Signal
Red	COM Input +
Gray	COM Output -
Light Blue	IP Reset Input
Orange	Synchronous Output
Brown	+VIN (12VDC/24VDC)
Blue	-VIN

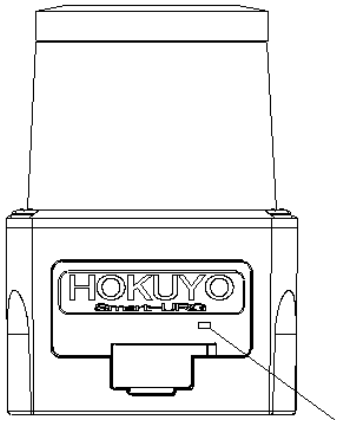
Note: Direction of Inputs and Outputs are mentioned from the sensor’s side.

6-2. Ethernet cable

Cable length: 300mm

Color	Signal
Blue	TX+
White	TX-
Orange	RX+
Yellow	RX-

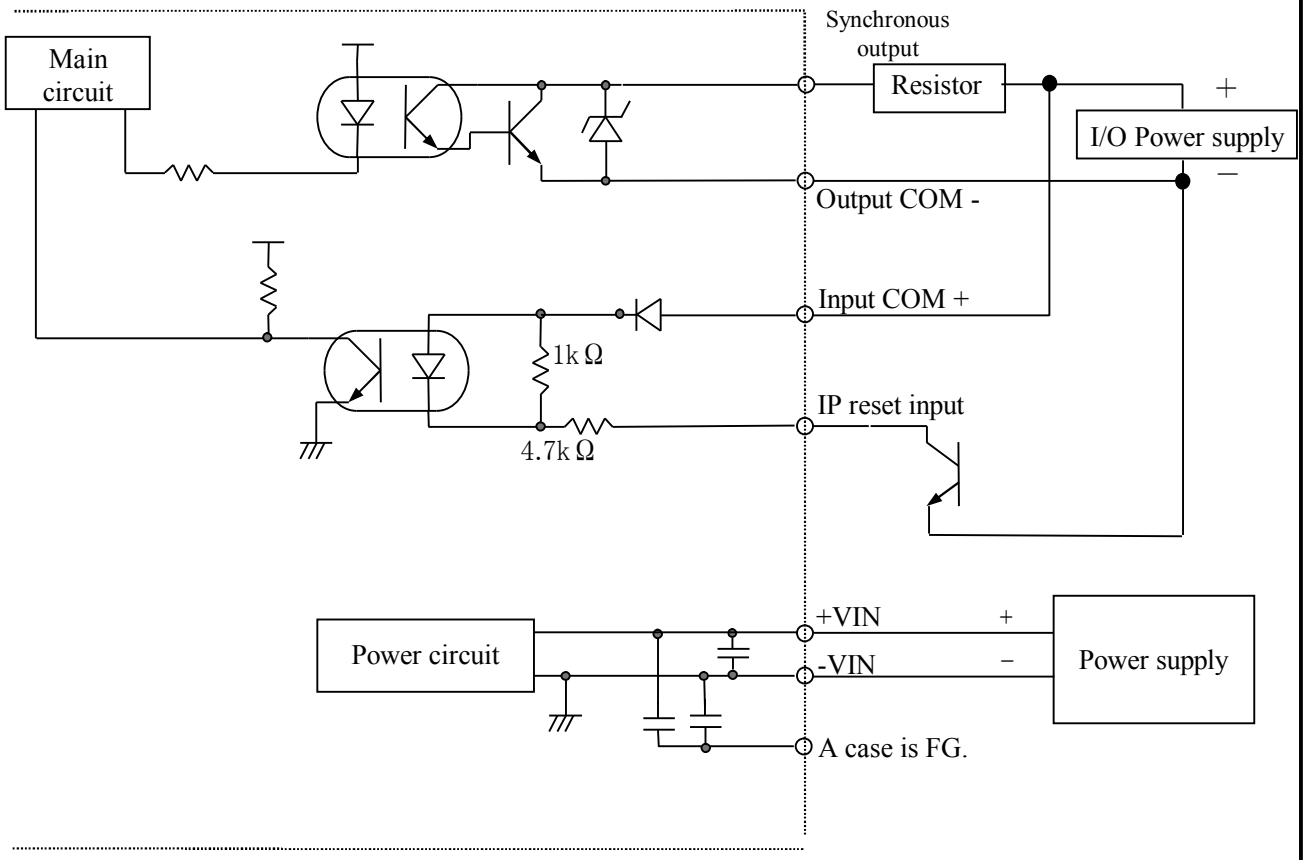
7. LED display ⚠2



Power supply display

(Blinks during start up and malfunction state)

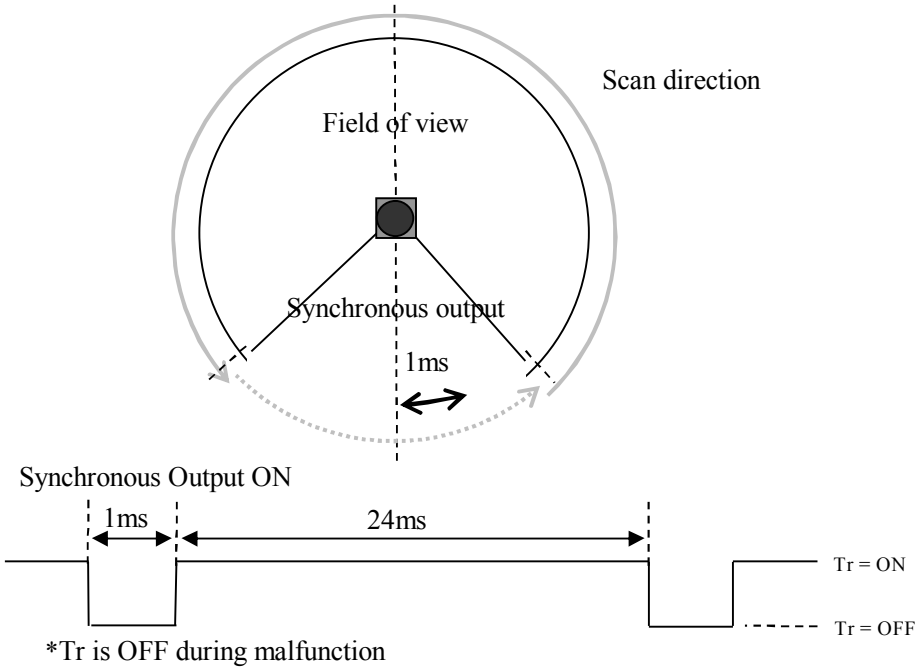
8. Output circuit



9. Control signal

9-1. Synchronous output

1 pulse is approximately 1ms. Output signal synchronization timing chart is shown as below.



10. Ethernet Setting ⚠2

1. The setting value is as below.

   IP Initial value :192.168.0.10

   Port number :10940

2. About Initialization of IP address

   To reset IP address to the factory default value, please provide power to input circuit (see Section 8) and connect IP RESET LINE to COM- for more than 2 sec.

   After IP RESET LINE is disconnected from COM- or opened, the sensor LED blinks and the sensor start to reboot.

11. Cautions for operation

This sensor uses high speed processing components that generate heat during operation.

The heat is concentrated at the bottom of the unit. When mounting, please attach the bottom of the unit to a good heat sink. A 200mm x 200mm x 2mm aluminum plate is recommended as a heat sink.

If multiple sensors are installed side by side, a sensor might mistake the laser pulses of other units as its own and the detection error occurs. When it happens, usually the error lasts for one or two steps of measurement. Please use software filters to handle this type of error.

DATA SHEET

NVIDIA Jetson TX2 System-on-Module

Pascal GPU + ARMv8 + 8GB LPDDR4 + 32GB eMMC + WLAN/BT

Description

The NVIDIA® Jetson TX2 System-on-Module (SOM) redefines possibility; a combination of performance, power efficiency, integrated deep learning capabilities and rich I/O remove the barriers to a new generation of products. The Jetson TX2 is ideal for many applications including (but not limited to): Intelligent Video Analytics (IVA), Drones, Robotics, Gaming Devices, Virtual Reality (VR), Augmented Reality (AR) and Portable Medical Devices. Superior performance, robust design and reduced complexity in system integration results in more advanced products getting to market faster.

The Jetson TX2 module integrates:

- **256 core NVIDIA Pascal GPU**. Fully supports all modern graphics APIs, unified shaders and is GPU compute capable. The GPU supports all the same features as discrete NVIDIA GPUs, including extensive compute APIs and libraries including CUDA. Highly power optimized for best performance in embedded use cases.
- **ARMv8 (64-bit) Multi-Processor CPU Complex**. Two CPU clusters connected by a high-performance coherent interconnect fabric designed by NVIDIA; enables simultaneous operation of both CPU clusters for a true heterogeneous multi-processing (HMP) environment. The **Denver 2 (Dual-Core)** CPU clusters is optimized for higher single-thread performance; the ARM **Cortex-A57 MPCore (Quad-Core)** CPU clusters is better suited for multi-threaded applications and lighter loads.
- **Advanced HD Video Encoder**. Recording of 4K ultra-high-definition video at 60fps. Supports H.265 and H.264 BP/MP/HP/MVC, VP9 and VP8 encoding.
- **Advanced HD Video Decoder**. Playback of 4K ultra-high-definition video at 60fps with up to 12-bit pixels. Supports H.265, H.264, VP9, VP8 VC-1, MPEG-2, and MPEG-4 video standards.
- **Display Controller Subsystem**. Two multi-mode (eDP/DP/HDMI) outputs and up to 8-lanes of MIPI-DSI output. Multiple line pixel storage allows more memory-efficient scaling operations and pixel fetching. Hardware display surface rotation is also provided for bandwidth reduction in mobile applications.
- **128-bit Memory Controller**.128-bit DRAM interface providing high bandwidth LPDDR4 support.
- **8GB LPDDR4 and 32 GB eMMC memory** integrated on the module
- **1.4Gpix/s Advanced image signal processing**: Hardware accelerated still-image and video capture path, with advanced ISP.
- **Audio Processing Engine**. Audio subsystem enables full hardware support for multi-channel audio over multiple interfaces.

Description		Jetson TX2 System-on-Module*
Pascal GPU ◊		
256-core GPU | End-to-end lossless compression | Tile Caching | OpenGL® 4.5 | OpenGL® ES 3.2 | Vulkan® 1.0 | CUDA® 8.0 | GPGPU		
Maximum Operating Frequency		1.12GHz
CPU Complex ‡		
ARMv8 (64-bit) heterogeneous multi-processing (HMP) CPU architecture; two CPU clusters (6 processor cores) connected by a high-performance coherent interconnect fabric. NVIDIA Denver 2 (Dual-Core) Processor: L1 Cache: 128KB L1 instruction cache (I-cache) per core; 64KB L1 data cache (D-cache) per core | L2 Unified Cache: 2MB ARM® Cortex® -A57 MPCore (Quad-Core) Processor: L1 Cache: 48KB L1 instruction cache (I-cache) per core; 32KB L1 data cache (D-cache) per core | L2 Unified Cache: 2MB		
Maximum Operating Frequency per Core NVIDIA Denver 2 ARM Cortex-A57		 2.0GHz 2.0GHz
HD Video & JPEG		
Video Decode(Number of Streams Supported): H.265 (†): Main 10, Main 8 H.265 (†): Main 444 H.264 (†): Baseline, Main, High H.264 (†): MVC Stereo (per view) VP9 (†): Profile 0 (8-bit) and 2 (10 and 12-bit) VP8: All MPEG1/2: Main MPEG4: SP/AP VC1: SP/MP/AP		 (2x) 2160p60 | (4x) 2160p30 | (7x) 1080p60 | (14x) 1080p30 2160p60 | (2x) 2160p30 | (3x) 1080p60 | (7x) 1080p30 (2x) 2160p60 | (4x) 2160p30 | (7x) 1080p60 | (14x) 1080p30 2160p60 | 2160p30 | 1080p60 | 1080p30 (2x) 2160p60 | (4x) 2160p30 | (7x) 1080p60 | (14x) 1080p30 2160p60 | (2x) 2160p30 | (4x) 1080p60 | (8x) 1080p30 2160p60 | (2x) 2160p30 | (4x) 1080p60 | (8x) 1080p30 (4x) 1080p60 | (8x) 1080p30 (2x) 1080p60 | (4x) 1080p30
Video Encode (Number of Streams Supported): H.265 H.264: Baseline, Main, High WEBM VP9 WEBM VP8		 2160p60 | (3x) 2160p30 | (4x)1080p60 | (8x) 1080P30 2160p60 | (3x) 2160p30 | (7x) 1080p60 | (14x) 1080p30 2160p30 | (3x) 1080p60 | (7x) 1080p30 2160p30 | (3x) 1080p60 | (6x) 1080p30
JPEG (Decode & Encode)		600 MP/sec
Audio Subsystem		
Industry-standard High Definition Audio (HDA) controller provides a multi-channel audio path to the HDMI interface | 4 x I2S | DMIC | DSPK | 2 x I and Q baseband data channels | PDM in/out		
Display Controller Subsystem		
Support for DSI, HDMI, DP and eDP | Two multi-mode eDP/DP/HDMI outputs.		
Captive Panel		
MIPI-DSI (1.5Gbps/lane)	Max Resolution	Support for Single x4 or Dual x4 links | 2560x1600 at 60Hz
eDP 1.4 (HBR2 5.4Gbps)	Max Resolution	3840x2160 at 60Hz
External Display		
HDMI 2.0a/b (6Gbps)	Max Resolution	3840x2160 at 60Hz
DP 1.2a (HBR2 5.4 Gbps)	Max Resolution	3840x2160 at 60Hz
Imaging System		
Dedicated RAW to YUV processing engine process up to 1.4Gpix/s | MIPI CSI 2.0 up to 2.5Gbps (per lane) | Support for x4 and x2 configurations (up to 3 x4-lane or 6 x2-lane cameras)		
Clocks		
System clock: 38.4 MHz | Sleep clock: 32.768 KHz | Dynamic clock scaling and clock source selection		
Boot Sources		
Internal eMMC and USB (recovery mode)		

Description	Jetson TX2 System-on-Module*
Security	
Secure memory with video protection region for protection of intermediate results | Configurable secure DRAM regions for code and data protection | Hardware acceleration for AES 128/192/256 encryption and decryption to be used for secure boot and multimedia Digital Rights Management (DRM) | Hardware acceleration for AES CMAC, SHA-1, SHA-256, SHA-384, and SHA-512 algorithms | 2048-bit RSA HW for PKC boot| HW Random number generator (RNG) SP800-90 | TrustZone technology support for DRAM, peripherals | SE/TSEC with side channel counter-measures for AES | RSA-3096 and ECC-512/521 supported via PKA	
Memory ††	
128-bit DRAM interface | Secure External Memory Access Using TrustZone Technology | System MMU	
Memory Type	4ch x 32-bit LPDDR4
Maximum Memory Bus Frequency (up to)	1866MHz
Memory Capacity	8GB
Storage	
eMMC 5.1 Flash Storage	
Bus Width	8-bit
Maximum Bus Frequency	200MHz (HS400)
Storage Capacity	32GB
Connectivity	
WLAN	
Radio type	IEEE 802.11a/b/g/n/ac dual-band 2x2 MIMO
Maximum transfer rate	866.7Mbps
Bluetooth	
Version level	4.1
Maximum transfer rate	3MB/s
Networking	
10/100/1000 BASE-T Ethernet | IEEE 802.3u Media Access Controller (MAC) | Embedded memory	
Peripheral Interfaces Δ	
XHCI host controller with integrated PHY: (up to) 3 x USB 3.0, 3 x USB 2.0 | USB 3.0 device controller with integrated PHY | 5-lane PCIe: two x1 and one x4 controllers | SATA (1 port) | SD/MMC controller (supporting eMMC 5.1, SD 4.0, SDHOST 4.0 and SDIO 3.0) | 5 x UART | 3 x SPI | 8 x I^2C | 2 x CAN | 4 x I2S: support I^2S, RJM, LJM, PCM, TDM (multi-slot mode) | GPIOs	
Operating Requirements ♦	
Temperature Range	-25C – 80C
Module Power	7.5W (Max-Q) / 15W (Max-P)
Power Input	5.5V – 19.6V
Applications	
Intelligent Video Analytics, Drones, Robotics, Industrial automation, Gaming, and more.	

* Refer to the software release feature list for current software support.

◊ GPU Maximum Operating Frequency: 1.3GHz supported in boost mode.
Product is based on a published Khronos Specification and is expected to pass the Khronos Conformance Process. Current conformance status can be found at www.khronos.org/conformance.

‡ CPU Maximum Operating Frequency: 1-4 core = up to 2.0GHz; greater than 4 core = up to 1.4GHz

(†) For max supported number of instances: bitrate not to exceed 15 Mbps per HD stream (i.e., 1080p30), overall effective bitrate is less than or equal to 240 Mbps

†† Dependent on board layout. Refer to Interface Design Guide for layout guidelines.

Δ Refer to the Interface Design Guide and *Parker Series SoC Technical Reference Manual* to determine which peripheral interface options can be simultaneously exposed.

♦ Refer to the *Jetson TX2 OEM Product Design Guide* and *Jetson TX2 Thermal Design Guide* for evaluating product power and thermal solution requirements. See the software documentation for information on changing the default power mode (default: Max-P).

Revision History

Version	Date	Description
1.0	MAY 1, 2017	Initial Release
1.1	JUN 30, 2017	Summary: added Module Power to table Added Environmental & Mechanical Screening section; includes reliability tests and standards used to evaluate module robustness. Package Drawing and Dimensions: updated package drawing

Laser type

Scanning Laser Range Finder

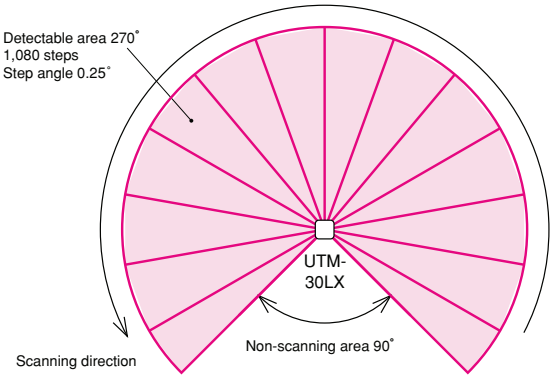
UTM-30LX FDA approval

Long distance scanning 30m!

UTM-30LX is a 2-dimensional laser sensor for measuring the distance to the objects.

- Wide range scanning, 30m and 270˚ .
- Available for outdoor use because of 100,000lux for ambient illuminance and IP64 for protective structure.
- High-speed response, 25msec.
- 12VDC.

System structure



Note) The above figure shows the detectable area for white Kent sheet (500mm×500mm). Max.detection ditance is 30m. Detection distance may vary with size and object.

Specifications

Kinds	Data output type (serial type)
Model No.	UTM-30LX
Power source	12VDC ±10%
Current consumption	700mA or less (rush current approx.1A)
Light source	Semiconductor laser diode λ=905nm (FDA approval, Laser safety class 1)
Detectable object	500×500mm white sheet or more
Scanning range	0.1 to 30m
Scanning accuracy	0.1 to 10m: ±30mm, 10 to 30m: ±50mm*
Scanning angle	270˚
Resolution	1mm
Angular Resolution	Step angle: approx.0.25˚ (360˚ /1,440 steps)
Beam diameter	Approx. ϕ400mm (at 30m)
Scanning time	25msec/scan
Interface	USB2.0 (Full Speed)
Communicating specifications	Exclusive command (SCIP Ver.2.0)
Output	OUTPUT 1 pce, synchronous output
Indication lamps	Power lamp (green): Lights up when normal operation, Operation lamp (red): Lights up when normal operation
Connection	Power and synchronous output: cable 2m, USB:2m cable with type A plug
Ambient illuminance note)	Halogen/mercury lamp: 10,000lux or less, incandescent lamp: 6,000lux or less
Ambient temperature	-10 to +50℃ (-25 to +75℃when stored)
Ambient humidity	85%RH or less, not icing, not condensing

Insulation resistance	10MΩ 500VDC megger
Vibration resistance	Double amplitude 1.5mm, 10 to 55Hz, each 2 hour in X, Y and Z directions
Impact resistance	196m/s^2, each 3 time in X, Y and Z directions
Protective structure	IP64 (IEC standard)
Life	5 years (motor life, vary depending on use conditions)
Noise	25dB or less (at 300mm)
Case materials	Polycarbonate
Weight	Approx.370g (including cable)

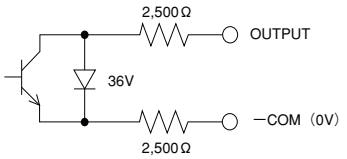
*It may reduce the accuracy when receing strong light like sunlight etc. directly.

Note This sensor is not a safety device/tool.

Note This sensor is not for use in military applications.

■ Connection

Output circuit



Wiring table

● **Cable 1 power and output**

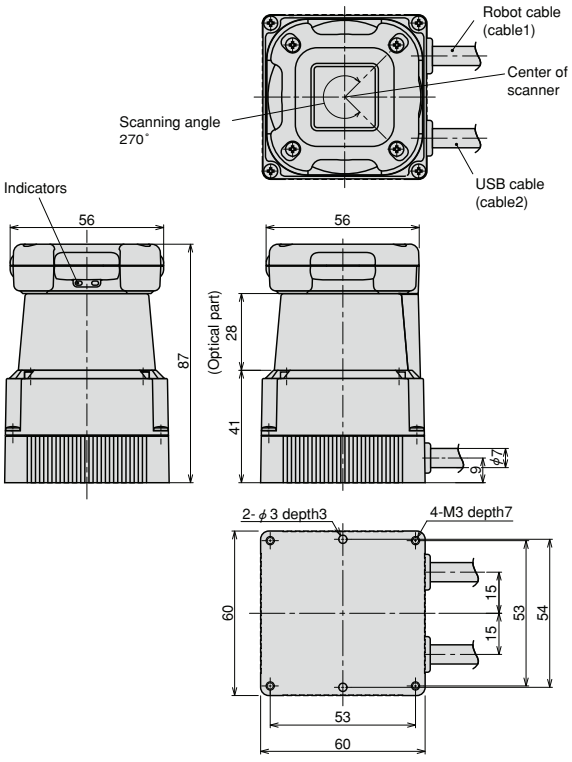
Cable colors	Signals
Brown	+VIN (12VDC)
Blue	-VIN (0V)*
Green	Synchronous output
White	COM (0V)*

*Power and output (0V) are not connected inside.

Note I/O direction is on the basis of UTM-30LX.

● **Cable2 USB Type A(4 pins)**

■ External dimension

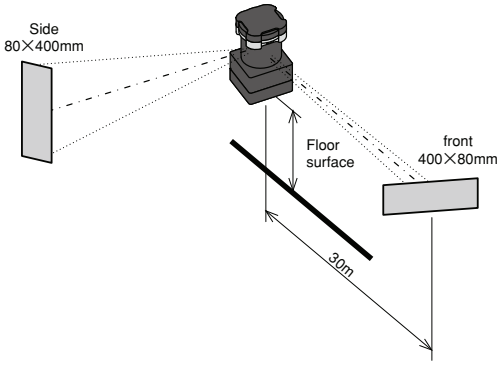


■ Caution for installation

(1) When installation, don't close light-projection/reception window or interrupt area.

(2) Don't make a wiring with high-voltage line or load line because of avoiding noise or surge induction.

(3) Install it 200mm or more away from floor. If 200mm or less, install it 1° upward. Spread of sensor beam is φ400mm (Reference value) at 30m.



■ Supplement

- Scanning direction is counterclockwise from topview.
- About USB driver
  It is connected as software COM port through CDC (Communication Device Class). It can be handled as well as COM port from applicatoin program of host. But this doesn't provide plug & play function.
- This sensor has higher radiation value because of high speed processing and so please install it with radiation plate at the bottom. (Recommend: aluminum plate with 200×200×2) because radiation is found on the bottom cover.
- It may make a false detection in case of close installation.
  In case that, make filtering processing of data.

Velodyne LiDAR PUCK™

VLP-16

Velodyne's new VLP-16 sensor is the smallest, newest, and most advanced product in Velodyne's 3D LiDAR product range. Vastly more cost-effective than similarly priced sensors, and developed with mass production in mind, it retains the key features of Velodyne's breakthroughs in LiDAR: Real-time, 360°, 3D distance and calibrated reflectivity measurements.

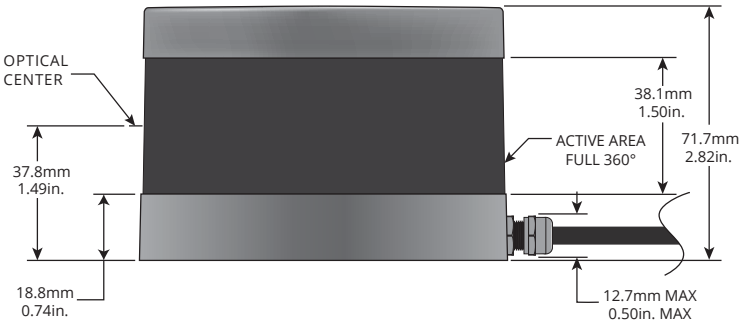
VLP-16

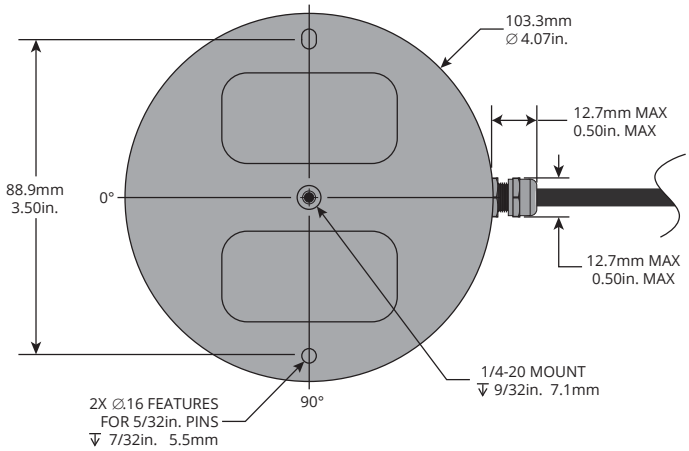
3D - Real Time - LiDAR

The VLP-16 has a range of 100m, and the sensor's low power consumption (~8W), light weight (830 grams), compact footprint (~Ø103mm x 72mm), and dual return capability make it ideal for UAVs and other mobile applications.

Velodyne's LiDAR Puck supports 16 channels, ~300,000 points/sec, a 360° horizontal field of view and a 30° vertical field of view, with +/- 15° up and down. The Velodyne LiDAR Puck does not have visible rotating parts, making it highly resilient in challenging environments (Rated IP67).

DIMENSIONS



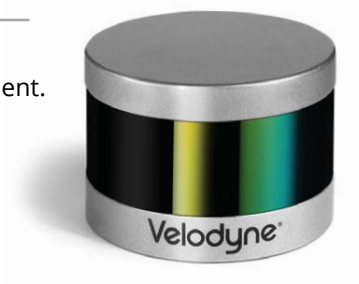


VLP-16

Real-Time 3D LiDAR

The HDL-32E provides high definition 3-dimensional information about the surrounding environment.

PUCK™



	Specifications:
Sensor:	• Time of flight distance measurement with calibrated reflectivities • 16 channels • Measurement range up to 100 meters • Accuracy: +/- 3 cm (typical) • Dual returns • Field of view (vertical): 30° (+15° to -15°) • Angular resolution (vertical): 2° • Field of view (horizontal/azimuth): 360° • Angular resolution (horizontal/azimuth): 0.1° - 0.4° • Rotation rate: 5 - 20 Hz • Integrated web server for easy monitoring and configuration
Laser:	• Class 1 - eye safe • 903 nm wavelength
Mechanical/ Electrical/ Operational	• Power consumption: 8 W (typical) • Operating voltage: 9 - 32 VDC (with interface box and regulated power supply) • Weight: 830 grams (without cabling) • Dimensions: 103 mm diameter x 72 mm height • Shock: 500 m/sec² amplitude, 11 msec duration • Vibration: 5 Hz to 2000 Hz, 3G rms • Environmental Protection: IP67 • Operating temperature -10° to +60° C • Storage temperature - 40° to +105° C
Output:	• Up to 0.3 million points/second • 100 Mbps Ethernet connection • UDP packets containing - Distances - Calibrated reflectivities - Rotation angles - Synchronized time stamps (µs resolution) • $GPRMC NMEA sentence from GPS receiver (GPS not included)

63-9229 Rev-B

CE CLASS 1 LASER PRODUCT

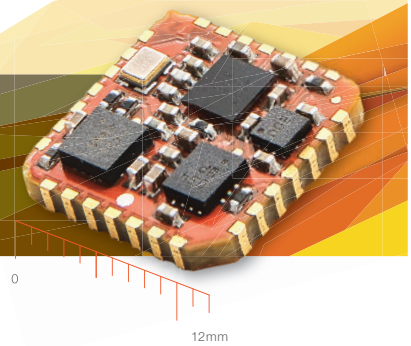
Velodyne Acoustics, Inc.
345 Digital Drive, Morgan Hill, CA 95037
lidar@velodyne.com

408.465.2800

MTi 1-series

IMU, VRU and AHRS

- ✔ Uniform software/hardware interface over product lifetime (no EOL)
- ✔ Always best-in-class inertial sensors incorporated
- ✔ Industry-leading signal processing pipeline and orientation algorithm
- ✔ API-compatible with all Xsens Motion Trackers



The MTi 1-series is a self-contained Attitude Heading and Reference System (AHRS), Vertical Reference Unit (VRU) and Inertial Measurement Unit (IMU) as a 12.1 x 12.1 mm module. The Xsens-optimized strapdown algorithm (AttitudeEngine™) performs high-speed dead-reckoning calculations at 1 kHz allowing accurate capture of high frequency motions. Xsens' industry-leading sensor fusion algorithm (XKF3™) provides high accuracy and sensor auto-calibration in a cost-effective module for a wide range of (embedded) applications. It relieves users from the design, integration and maintenance of gyroscopes, accelerometers and other sensors. The roll and pitch accuracy of 0.8 deg under dynamic conditions allow for integration in demanding applications.

Miniature aerial vehicles

- Delivery drones
- Video drones
- Agricultural UAVs

Ultra lightweight
Vibration rejection

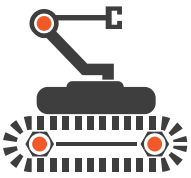
Machinery

- Satcom on the Move (SotM)
- Construction machinery
- Ship monitoring

Extremely low power
Motion on Demand

Robotics

- Autonomous agriculture
- Warehouse automation
- Robotic arms



Robust heading tracking
ROS node support

Other applications

- Handheld devices
- Pedestrian navigation
- VR/AR and HMDs
- Navigation aiding

Unlimited possibilities
Flexible design

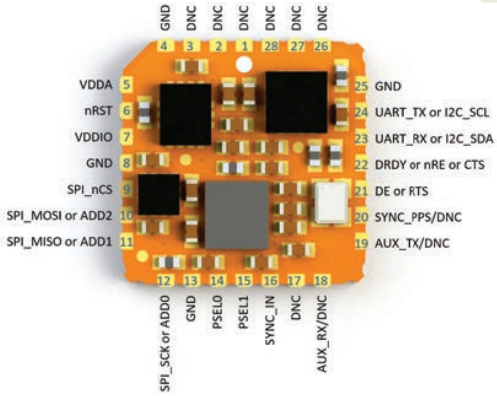
Ordering information

Part Number	Output	Packing Method
MTi-1	IMU; inertial data	Tray (containing 20 modules)
MTi-2	VRU; inertial data, roll/pitch, heading tracking	Tray (containing 20 modules)
MTi-3	AHRS; inertial data, roll/pitch/yaw	Tray (containing 20 modules)
		Reels available from 250 units
MTi-3-DK	Development kit for MTi 1-series	Development Kit

Specifications MTi 1-series

Orientation accuracy	
Roll/Pitch (static)	0.5° 1σ RMS
Roll/pitch (dynamic)	0.8° 1σ RMS
Yaw (dynamic)	2° 1σ RMS
Inertial sensor performance	
Gyroscope full-scale range	±2000°/s
Gyroscope bias stability	10 deg/hr
Gyroscope noise density	0.007°/s/√Hz
Gyroscope non-linearity	0.1% FS
Accelerometer full-scale range	±16 g
Accelerometer bias stability	0.03 mg
Accelerometer noise density	120 µg/√Hz
Accelerometer non-linearity	0.5% FS
System specifications	
Power consumption	44 mW @ 3V
Input voltage	2.19 to 3.6V
Package	SMD, footprint compatible with JEDEC PLCC-28
Size	12.1 x 12.1 x 2.55 mm
Weight	<1 g
Packaging	Tray (20 modules) Reel (250 modules)
Interfacing	
Hardware interface	I²C, SPI, UART (selectable)
Software interface	Xsens Xbus binary protocol Driver source code supplied
Output data rate	0-800 Hz

PIN LAYOUT

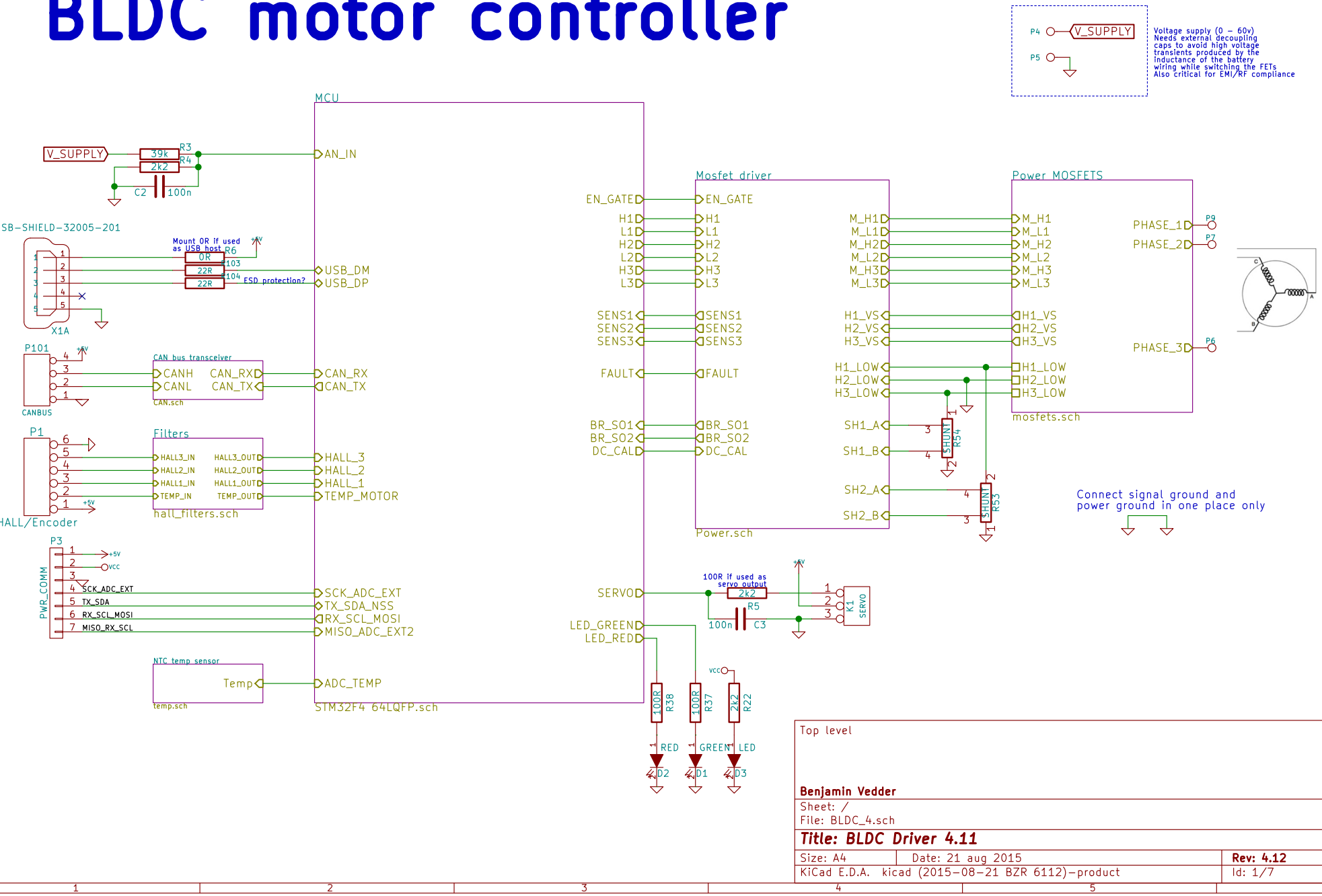


DEVELOPMENT KIT

In order to get started with the MTi 1-series, an extensive development kit for characterization and prototyping is available:

- Shield board including MTi-3 module and USB calbe
- Arduino header compatible shield board
- Easy to use connection (micro USB), access to I²C/SPI/UART
- Arduino header compatible shield board
- Full functionality and pin configuration
- Intuitive MT Software Suite (Linux / Windows GUI)
- SDK with drivers and embedded software examples

BLDC motor controller



Top level

Benjamin Vedder

Sheet: /
File: BLDC_4.sch

Title: BLDC Driver 4.11

Size: A4	Date: 21 aug 2015	**Rev: 4.12**
KiCad E.D.A. kicad (2015-08-21 BZR 6112)-product		Id: 1/7