TRABALHO DE GRADUAÇÃO

DRIVING MOBILE ROBOTS: A COMPARATIVE ANALYSIS BETWEEN SIGNAL FUSION METHODS FOR SYSTEMS WITH AIDED GUIDANCE

Carolina Sartori da Silva

Brasília, Maio de 2021

ENGENHARIA
MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

DRIVING MOBILE ROBOTS: A COMPARATIVE ANALYSIS BETWEEN SIGNAL FUSION METHODS FOR SYSTEMS WITH AIDED GUIDANCE

Carolina Sartori da Silva

Relatório submetido como requisito parcial de obtenção
de grau de Engenheiro de Controle e Automação

Banca Examinadora

Prof. Carla Cavalcante Koike, CiC/UnB _____
Orientadora

Prof. Flávio de Barros Vidal, CiC/UnB _____
Coorientador

Prof. Guilherme Caribé de Carvalho _____
Examinador interno

Jones Yudi Mori Alves da Silva _____
Examinador interno

Brasília, Maio de 2021

FICHA CATALOGRÁFICA

SARTORI DA SILVA, CAROLINA
Driving mobile robots: a comparative analysis between signal fusion methods for systems with aided guidance.

[Distrito Federal] 2021.

X, 69p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2021). Trabalho de Graduação – Universidade de Brasília.Faculdade de Tecnologia.

1. Mobile robots
2. Kalman filter
3. Signal fusion
4. Aided guidance

I. Mecatrônica/FT/UnB

REFERÊNCIA BIBLIOGRÁFICA

SARTORI DA SILVA, CAROLINA, (2021). Driving mobile robots: a comparative analysis between signal fusion methods for systems with aided guidance. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-$n°$03, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 84p.

CESSÃO DE DIREITOS

AUTOR: Carolina Sartori da Silva

TÍTULO DO TRABALHO DE GRADUAÇÃO: Driving mobile robots: a comparative analysis between signal fusion methods for systems with aided guidance.

GRAU: Engenheiro ANO: 2021

Carolina Sartori da Silva

Universidade de Brasília, Asa Norte.

70910-900 Brasília – DF – Brasil.

Dedicatória

Dedico este trabalho à minha mãe, Regina, meu pai, Ítalo, minha irmã, Isabela e à minha avó, Maria Cecília. Sem vocês isso não teria sido possível.

Carolina Sartori da Silva

Agradecimentos

Agradeço a todos que me acompanharam durante toda a minha graduação, colegas de curso e amigos, que tornaram essa longa jornada mais prazerosa e construtiva.
Aos meus colegas do intercâmbio em Torino, que se tornaram amigos que espero levar para toda a vida. Agradeço por todas conversas, risadas, viagens e principalmente pelos ensinamentos. Graças a vocês hoje sou uma pessoa e uma profissional melhor.
A todos os meus amigos da vida, que mesmo muitas vezes não entendendo os meus questionamentos e dúvidas com o curso, sempre me apoiaram, ajudaram e estiveram comigo. Vocês me fizeram a pessoa que sou hoje.
Agradeço também aos meus orientadores, que me apoiaram ao longo dos dois últimos anos no desenvolvimento deste trabalho e em meu crescimento profissional.
Por fim, agradeço à minha família, que sempre me apoiou em todas as decisões, todos os momentos de dificuldades e tornaram todo essa trajetória mais leve e feliz.

Carolina Sartori da Silva

RESUMO

A robótica é um campo da engenharia que têm se popularizado muito nos últimos anos, substituindo humanos em tarefas repetitivas ou que representem algum risco à saúde ou segurança do trabalhador. Entretanto, uma desvantagem da utilização de robôs é o alto custo dos treinamentos para os operadores, algumas vezes tornando seu emprego economicamente inviável. Um meio para solucionar este problema é usar sistemas de auxílio à guiagem do robô, permitindo que mesmo um operador inexperiente consiga utilizá-lo de forma satisfatória. Uma forma de implementar tais sistemas é usando redes neurais, que aprendem a sequência de comandos necessários para realizar determinada tarefa e podem auxiliar o operador em sua execução. Este trabalho propõe, então, uma implementação do filtro de Kalman para realizar a fusão do sinal de entrada do operador com o sinal de entrada da rede, tendo assim um único sinal de saída para o robô. Os objetivos deste trabalho são propor um algoritmo para o filtro de Kalman para realizar a fusão dos sinais, validar o filtro em simulações usando uma base de dados adquirida previamente e validar o filtro no robô testando com diversos usuários, quantitativa e qualitativamente, por meio de uma pesquisa sobre a percepção dos usuários. Para realizar a validação do filtro de Kalman como método de fusão de sinais, este foi comparado com outros dois métodos mais simples: network with 0.05 threshold (usa o sinal de saída da rede como entrada para o robô caso a diferença entre a entrada do usuário e a da rede seja maior que 5%) e mean with 0.05 threshold (usa a média aritmética entre a saída da rede e a entrada do usuário como entrada para o robô caso a diferença entre a entrada do usuário e a média seja maior que 5%). Depois de todas as etapas de validação, foi possível concluir que o filtro de Kalman garantiu trajetórias mais suaves e foi preferido por 37,5% dos usuários.

Palavras Chave: Robótica móvel, Filtro de Kalman, Fusão de sinais, Assitência de direção.

ABSTRACT

Robotics is a field of engineering that has become very popular in recent years, replacing humans in repetitive tasks and activities that represent a risk to the health or safety of the worker. However, a disadvantage of using robots is the high cost of training for operators, sometimes making the use of such robots economically unfeasible. One way to solve this problem is to use systems to help guiding the robot, allowing even an inexperienced operator to use it satisfactorily. One way to implement such strategies is to use neural networks, which learn the sequence of commands necessary to perform a given task and assist the operator in its execution. This work proposes an implementation of the Kalman filter to perform the fusion of the operator's input signal with the network's input signal, thus having a single output signal for the robot. The objectives of this work are to propose an algorithm for the Kalman filter to perform the fusion of the signals, to validate the filter in simulations using a previously acquired database and to validate the filter with the robot, testing with several users, quantitatively and qualitatively, by means of a survey on the perception of users. To perform the validation of the Kalman filter as a signal fusion method, it was compared with two other simpler methods: network with 0.05 threshold (uses the network output signal as input to the robot in case the difference between the user input and that of the network is greater than 5%) and mean with 0.05 threshold (uses the average between the network output and the user input as the input to the robot if the difference between the user input and the average is greater than 5%). After all the validation steps, it was possible to conclude that the Kalman filter guaranteed smoother trajectories and was preferred by 37.5% of users.

Keywords: Mobile robots, Kalman filter, Signal fusion, Aided guidance

Contents

List of Figures

List of Tables

List of Symbols

Latin Symbols

A	State matrix
B	Control or input matrix
C	Measured state matrix
E	Expectation of a random variable
Q	Covariance matrix of the measured state
R	Covariance of the noise of the posterior state
u	Control or input vector
x	States vector
z	Measured state

Greek Symbols

σ	Standard deviation
Σ	Covariance Matrix
μ	Mean vector
ϵ	Gaussian vector that describes the uncertainty introduced by the state transition
δ	Measurement noise

Grupos Adimensionais

$p(x)$	Probability Density Function of a random variable x
$p(x, y)$	Joint distribution of two random variables

Subscritos

t	Current instant in continuous time
$t-1$	Previous instant in continuous time
k	Current instant in discrete time
$k+1$	Following instant in continuous time

Siglas

CNN	Convolution Neural Networks
PDF	Probability Denstity Function
ROS	Robot Operating System

Chapter 1

Introduction

1.1 Context

Robots, as will be explained further in Chapter 2, Section 2.1, are machines that can be helpful if used and built right. They can be guided by an external operator or by an embedded computer.

In the robotics field, there are two main categories of robots: automated and teleoperated [1]. The first has the decision-making process made by an internal or external computer, without the necessity of human assistance. A human operator controls the latter, which is then responsible for all the decision-making process. Usually, the operator needs to be minimally skilled for the job. That need may represent a considerable cost, depending on the operation to be executed. The robot itself is already expensive equipment so that the total price can get high quickly.

To reduce the cost of education, the operator is driving the teleoperated robots with some autonomy. The final movement depends on the signal the operator sent and the decision taken by the robot's system.

Artificial intelligence, specifically neural networks, is an example of a decision-making algorithm used in this type of solution. That said, this work aims to provide a method to fuse the neural network's output, deciding the ideal trajectory with the input signal sent from the operator to make the trajectory intended by the operator as flawless as possible.

The use of robots with some autonomy, guaranteeing a smooth and safe trajectory, can be beneficial in several areas. Some examples are rovers to explore remote locations, robotic arms in the industry, and developing delicate activities.

1.2 Problem definition and objectives

As said in Section 1.1, a current obstacle of the industry regarding teleoperated robots is the cost of operator training to pilot the robot safely and efficiently.

One possible solution to that problem is using an embedded decision-making algorithm, usually

involving artificial intelligence, to help the operator drive through a safe trajectory. The problem one aims to propose a solution to is merging the output signal of the neural network embedded in the robot's system and the input signal given by an operator with little or no training.

Kalman filter provides a way to fuse two sensor signals by modelling the system in the space state form and using the two sensors as inputs. That way, the output would be the two input signals merged into one that can be used as input to the robot. This work uses a Kalman filter whose input is the neural network's output and the operator's input.

To realise this project, it will be used a database made by another student at the University of Brasília, Thúlio Noslen [2].

The main objective of this project is to propose and validate a method of signal fusion based on the Kalman Filter so that the user's input signal and that of the network can be properly used with the robot. Therefore, this objective can be divided into the following sub-objectives:

- Propose a Kalman Filter algorithm to fuse the two signals;
- Validate the filter in simulation using the database;
- Validate the filter in the robot, quantitatively, testing with several users;
- Validate the filter qualitatively, analysing the perception of those users.

1.3 Manuscript Presentation

This manuscript has the following chapters:

- Chapter 2 - Theoretical Foundations: gives a brief introduction to the concepts used to develop and understand this work;
- Chapter 3 - Methodology: the techniques used are defined here and the experimental procedures are listed;
- Chapter 4 - Results: shows the results obtained this far with the use of the proposed Kalman Filter algorithm;
- Chapter 5 - Conclusions: contains a summary of the work done and the final considerations regarding the results.

Chapter 2

Theoretical Foundations

This chapter describes briefly the theoretical foundations necessary to understand and to develop this work. It has a section about ***Robotics*** *2.1, one about* ***Probabilistic Robotics*** *2.2 and another about* ***Deep Learning*** *2.3.*

2.1 Robotics

2.1.1 Introduction

The field of robotics is mainly concerned with the study of machines that can replace humans in executing a task regarding decision-making and physical activities. This field has a deep cultural meaning since humans have been seeking something to serve as a substitute in some tasks and something that mimics our behaviour in the various interactions with the environment surrounding us.

Since the Greeks, one of the greatest ambitions of humankind has been to give life to their artefacts. We can see that clearly in several myths, like the Titan Prometheus, the giant Talus, and, more recently, the tale of Frankenstein. In the 1920s, already in the Industrial Age, the Czech playwright Karel Čapek introduced a mechanical creature whose purpose was to replace humans in subordinate labour duties, which he named *automaton*. On that same occasion, in the play *Rossum's Universal Robots (R.U.R.)*, he coined the term robot, at the time derived from the term *robota*, which means "executive labour" in Slav languages [1]. However, in Karel's science fiction tale, the automaton built by Rossum rises against humankind, unlike what happened in the following years.

A couple of decades later, in the 1940s, the Russian Isaac Asimov, a science fiction writer, described a robot as an automaton with a human appearance but deprived of any feelings. Its brain purely dictated its behaviour, which was programmed in such a way to satisfy specific rules of ethical behaviour defined by the human who built it. Asimov was also responsible for introducing the term robotics as being the science devoted to studying robots based on the *three fundamental laws* [1] :

- A robot may not injure a human being or, through inaction, allow a human being to come to harm;
- A robot must obey the orders given by human beings, except when such orders would conflict with the first law;
- A robot must protect its own existence, as long as such protection does not conflict with the first or the second law.

These laws were established in the fiction field, but they were carried out to the science field, setting rules of behaviour to consider as specifications of a robot design as an industrial product done by engineers. From that age, robots were seen as machines able to modify the environment they are in, independently of their exterior. In fact, robotics is often described as the science that studies the intelligent connection between perception and action.

Considering that, one may start describing the essential component of a robot: its mechanical system. In general, it is composed of a locomotion apparatus and a manipulation apparatus, the first usually being wheels, crawlers or mechanical legs, and the latter being mechanical arms, end-effectors or artificial hands. The combination of the locomotion apparatus and the manipulation apparatus is one way to categorize the robot.

Having defined the mechanical system, now one must describe the several abilities of a robot. The ones described in the following paragraphs are the capabilities to exert an action, percept the environment, and connect the action to perception.

The first one, the capability of exerting an action, is composed of both locomotion and manipulation abilities and is provided by an actuation system, which animates the mechanical components of the structure [1]. It refers to the context of motion control, dealing with drivers, transmissions and motors.

The second, the capability to percept the environment it is in, is carried out by the sensory system, which can acquire data regarding the mechanical system, both internal to the system and the external status of the environment. It refers to the context of sensors, materials properties, signal conditioning, information retrieval and data processing.

The third and last regards the connection of the first and the second. Therefore, it combines the perception of the environment with the execution of tasks in an intelligent fashion provided by the control system, which is responsible for commanding the execution of actions towards a goal set by the planning technique, respecting all constraints imposed by the environment and the robot itself. Its context is that of control and supervision of robot motions, artificial intelligence and expert system, as well as computer architecture and programming environment [1].

Having defined all that, one can easily see that robotics is an interdisciplinary subject concerning areas such as mechanics, control, electronics and computers.

2.1.2 Robot mechanical structure

The robot's mechanical structure is its key feature. As was said before, robots can be classified into two main classes by their mechanical structure: those with a fixed base, called robot manipulators, and those with mobile-based, called mobile robots.

The mechanical structure of a robot manipulator is, as described by Siciliano [1] composed of a sequence of links (rigid bodies) connected by joints (means of articulation). The manipulator itself is characterized by an arm that provides mobility, a wrist that ensures dexterity and an end-effector that performs the task. The fundamental structure is the serial or open kinematic chain. The kinematic chain is said to be open when there is only one sequence of links connecting its ends. Although that is the most common case, the manipulator can also have a closed kinematic chain when the sequence of links forms a loop.

The articulation between two consecutive links can be either a prismatic or a revolution joint. Both create a relative motion between the two links, but the first is translational, and the latter is rotational. Usually, revolution joints are preferred because of their compactness and reliability. In an open kinematic chain, each prismatic or revolute joint provides one degree of freedom (DOF) to the structure. However, in a closed kinematic chain, the number of DOFs is always smaller than the number of joints in the structure due to constraints imposed by the loop.

The degrees of freedom should be properly distributed along the structure so that it can have a sufficient number to execute the tasks it was designed for. If there are more DOFs than what is necessary to execute the task, the robot is said to be redundant. One may then define the concept of workspace, which is the portion of the environment that the manipulator's end-effector can access. The workspace is directly dependent on both the manipulator structure and the mechanical joints.

At last, there is a classification for the manipulators depending on the type and sequence of arm's DOFs. Therefore, the robots can be defined as Cartesian, cylindrical, spherical, SCARA or anthropomorphic [1].

The understanding of the basics of the mechanical structure of mobile robots is of the most importance for this work. Its main feature is the presence of a mobile base that freely allows the robot to move in the environment. One other aspect they differ from the manipulators is the use because they are much more common in service applications, so autonomous motion capabilities are required.

From a mechanical point of view, a mobile robot consists of one or more rigid bodies, with or without joints, equipped with a locomotion system. Knowing that we can separate two classes of mobile robots:

- Wheeled: is the most common class, consisting of a rigid body and a system of wheels that provide motion. Other rigid bodies can be attached to the main one through revolute joints.
- Legged: is less common and composed of multiple rigid bodies connected by prismatic or revolute joints, the latter being more used than the first. There is a large variety of

mechanical structures in this class, most of them inspired by the study of living organisms (biomimetic robotics).

The wheeled type will now be explained a bit further since it is the type of robot used in this project. Its basic mechanical element being, indeed, the wheel. According to Siciliano in [1], there are mainly three types of conventional wheels:

- Fixed wheel: it can rotate about an axis that goes through the centre of the wheel, being orthogonal to the wheel plane. It is rigidly attached to the chassis, and its orientation with respect to the wheel is constant. It can be seen on the left side of Figure 2.1.
- Steerable wheel: it has two axes of rotation, the first being the same as the fixed wheel, and the second is vertical and goes through the centre of the wheel. This allows the chassis to be in a different orientation with respect to the wheel. This type can be seen in the centre of Figure 2.1.
- Caster wheel: also has two axes of rotation, but the vertical one does not pass through the centre of the wheel but is displaced by a fixed offset. This allows the wheel to rotate automatically, aligning with the direction of the motion with ease. This type of wheel is mainly used to provide a supporting point for static balance without affecting the mobility of the base, as seen in supermarket shopping carts. An example can be seen on the right side of Figure 2.1.

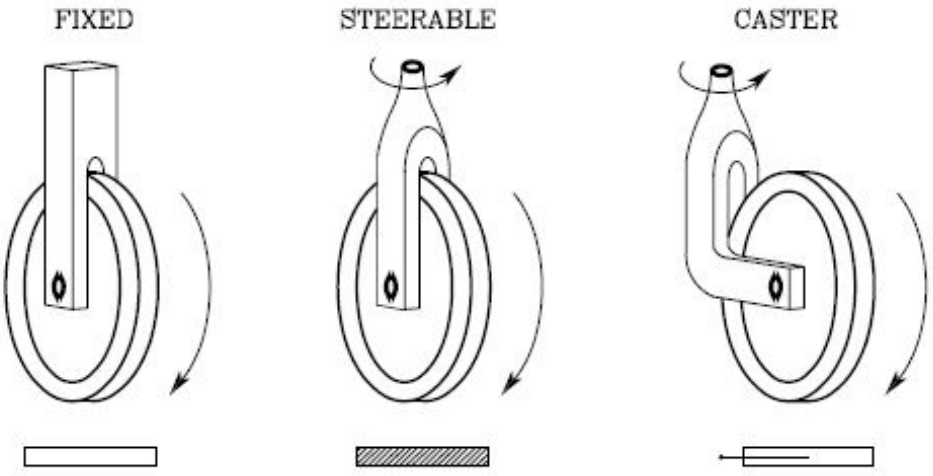

Figure 2.1: Main types of wheels - Adapted from [1].

Using those three types of wheels, one can provide a variety of kinematic structures. Only the most relevant combinations will be briefly described here.

The first is a differential drive, a vehicle with two fixed wheels, with a common axis of rotation, combined with one or more caster wheels, usually smaller than the fixed ones, to keep the structure statically stable. The two fixes wheels are separately controlled so that different velocities can

be applied. This allows the robot to turn freely and even rotate on the spot, depending on the controls applied. It can be seen in Figure 2.2 shown below.

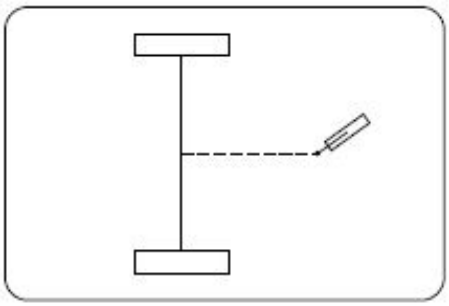

Figure 2.2: A differential-drive mobile robot - Adapted from [1].

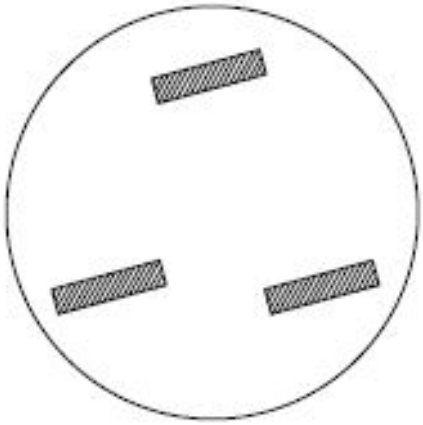

Figure 2.3: A synchro-drive mobile robot - Adapted from [1].

The second is a synchro-drive kinematic arrangement and is shown in Figure 2.3 above. As described by Siciliano [1], one can obtain that by aligning three steerable wheels, which are synchronously run by two motors through a mechanical coupling mechanism. One of the motors controls the rotation of the wheels around the vertical axis, affecting its orientation. The other one controls the rotation of the wheels regarding the horizontal axis, providing a driving force to the vehicle. A third motor is often used to rotate the upper part of the chassis independently, which can be useful to orient arbitrarily a directional sensor, for example.

The third is a tricycle vehicle, the one shown in Figure 2.4. It has three wheels, as the name suggests. Two of them are fixed, mounted on a rear axle, and the other is a steerable one, mounted in the front. The fixed wheels are controlled by a single motor which gives them traction, and the steerable one is driven by another motor that changes its orientation. Another configuration can be with the two fixed wheels passive and the steering wheel being controlled by two motors, one to give it orientation and the other to provide traction.

The fourth is the car-like vehicle, which is the one used in this project and is also the most common kind of vehicle in our day-to-day life. A simple model for it can be seen in Figure 2.5,

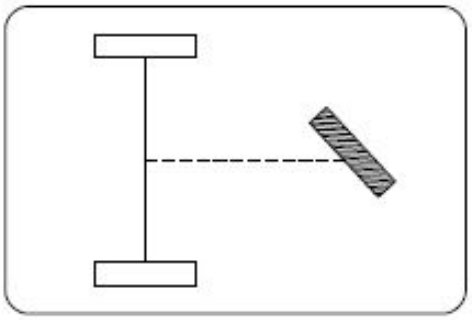

Figure 2.4: A tricycle mobile robot - Adapted from [1].

and it is the most relatable one since most of us have been in contact with machines with this configuration. It has two fixed wheels mounted on the rear axle and two steerable wheels mounted on the front. As in the tricycle, one motor provides traction, which can be located on the front or the rear, and another provides changes of orientation of the front wheels in respect to the chassis. It is important to emphasize that the two front wheels must have a different orientation when the vehicle is in a curved trajectory to avoid slippage. The internal wheel is slightly more steered than the external one. This is achievable by the use of a device called Ackermann steering.

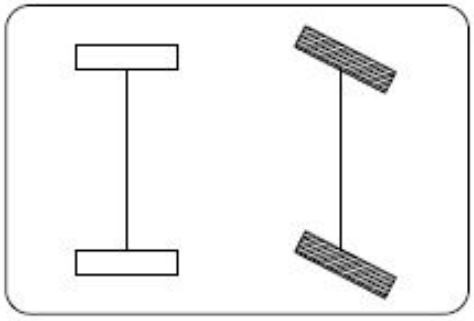

Figure 2.5: A car-like mobile robot - Adapted from [1].

The fifth and last is the omnidirectional vehicle, which has three caster wheels usually arranged in a symmetric pattern. The traction of the three wheels is provided by three different motors, keeping each wheel independent. This allows the vehicle to move instantaneously in any Cartesian direction, as it can re-orient itself on the spot. An example can be seen in Figure 2.6.

The workspace of a mobile robot, unlike the case of manipulators, is potentially unlimited. Although many mobile robots are subject to constraints on the admissible instantaneous motions, it does not prevent the possibility of attaining any position and orientation in the workspace. This also implies that the number of DOFs is smaller than the number of configuration variables [1].

Finally, one can think of merging the mechanical structure of a manipulator with that of a mobile robot. Such configuration is called mobile manipulator and combines the uses and the dexterity of a manipulator with the unlimited possibilities of workspaces provided by the mobile base.

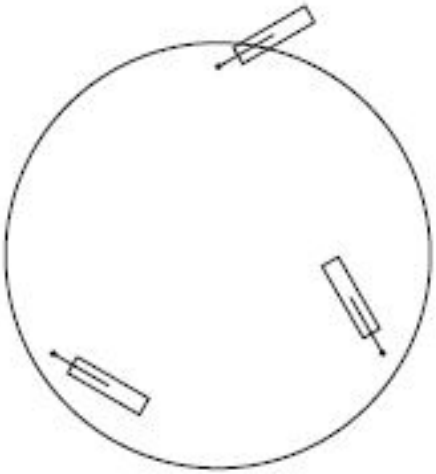

Figure 2.6: An omnidirectional mobile robot - Adapted from [1].

2.1.3 Industrial Robots

Industrial robotics is the field of robotics most concerned with robot design, control and applications in the industry. Its applications is that of operating in a structured environment, whose physical characteristics are mostly known *a priori*. Hence, limited autonomy is required [1].

The first industrial robots began to be developed in the 1960s, is highly influenced by two technologies: numerically controlled machines for precise manufacturing and teleoperators for remote radioactive material handling. Those first generations of industrial robots were characterized by versatility since they employed different end-effectors at the tip of the manipulator; adaptability to *a priori* unknown situations; positioning accuracy, by adopting feedback control techniques; and execution repeatability, since they could be programmed for various operations. This family of robots gained wide popularity because it allowed the realization of automated manufacturing systems.

One of the applications in the industrial field is the Automated Guided Vehicles (AGV), which are utilized to ensure the handling of parts and tools around the shop floor from one manufacturing cell to the next.

2.1.4 Advanced Robotics

The expression "advanced robotics" is commonly used to refer to the science of studying robots with marked characteristics of autonomy by operating in scarcely structured or unstructured environments, as stated by Siciliano [1].

The motivations for this field are many, ranging from the need for automata whenever human operators are safe or not available to the opportunity of developing products for potentially wide markets aimed at improving quality of life [1].

One subclass of industrial robots is the robots and is the category this work fits the most. The context is applying robots in areas where humans could not survive or would be exposed to unnecessary risk. Such robots can carry out exploration tasks and report data on the environment to a remote operator or even help find missing people in a disaster.

2.2 Probabilistic Robotics

2.2.1 Introduction

Rephrasing the concept of robotics said before in 2.1.1, robotics is the science of manipulating and perceiving the world through computer-controlled devices. Robotics systems perceive information on the environment they are in through sensors and apply forces through their manipulators. Nevertheless, in order to do those tasks, robots need to be able to contour the uncertainty that exists in the physical world.

The robot's environments are inherently unpredictable, as our world is. The challenge is making robots as capable of dealing with it as we are. Sensors are a good alternative, but they are limited in what they can perceive. Limitations arise from various factors, such as the range and the resolution. Sensors are also subject to noise, which makes the measurements made change in unpredictable ways and, by doing so, limits the ability to extract information. Another source of uncertainty is the actuation system, which involves motors and other mechanisms, being inherently unpredictable to some extent. Finally, the last primary source is algorithmic approximations because robots are real-time systems, but it limits the amount of computation that can be carried out.

That said, one can correctly assume that the uncertainty level depends directly on the application itself and its domain. That is the reason for using probabilistic robotic, because then "instead of relying on a single “best guess” as to what might be the case, probabilistic algorithms represent information by probability distributions over a whole space of guesses" [3]. This type of solution outperforms alternative techniques in various real-world applications.

The implications of that, according to Thrun [3], is the contrast with traditional programming techniques in robotics and probabilistic approaches, which tend to be more robust in the face of limitations, both model and sensor ones. Also, probabilistic algorithms have more flexible requirements on the accuracy of the models, relieving the programmer from the burden of coming up with an accurate model, which can be a problem in some situations.

However, as also stated by Thrun [3], all those advantages come at a price. Probabilistic algorithms make the system a lot more computationally complex and bring the need to approximate. This is due to the fact that probabilistic algorithms are inherently less efficient, and there is the need to consider a hole distribution of probability instead of a single value.

2.2.2 Recursive State Estimation

The core of probabilistic robotics is the idea of estimating states from sensor data, according to Thrun [3]. It aims to recover the state variables from the data obtained, so probabilistic state estimation algorithms compute the belief distributions over possible states. A simple example is the estimation of a mobile robot localization.

Although it is assumed that the reader has some basic knowledge of probability and statistics,

some basic concepts in probability will be introduced since they will be referred to later in the text. The first one is **random variables**, which are variables that, as described by Thrun [3], "can take on multiple values, and they do so according to specific probabilistic laws. The process of calculating these laws for random variables that are derived from other random variables and the observed data is called **probabilistic inference**". Then, there is the **Probability Density Function (PDF)**, and the PDF of a normal distribution is given by the following **Gaussian** function [3]:

$$p(x) = (2\pi\sigma^2)^{-\frac{1}{2}} exp\{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\} \tag{2.1}$$

The normal distribution assumes that x is a scalar value, but often it can be a multi-dimensional value, as in the case of this work. In that case, it is a normal distribution over a vector, and it called **multivariate**. Those functions are characterized by the following form of the density function, where μ is the **mean vector**, and Σ is a **covariance matrix**, whose characteristics are of being a positive semidefinite and symmetric matrix.

$$p(x) = (2\pi\Sigma)^{-\frac{1}{2}} exp\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\} \tag{2.2}$$

Both definitions, the equations 2.1and 2.2, described by Thrun in [3], are equivalent if $\Sigma = \sigma^2$ and x is a scalar instead of a vector. A couple of details that need to be explained is that the PDF value is not upper-bounded by 1 and that it will be silently assumed that all continuous random variables are measurable, so all continuous distributions actually possess densities.

The following two concepts are the **joint distribution** and **independence**, and the latter depends on the first. The joint distribution of two random variables X and Y is given by [3]:

$$p(x, y) = p(X = x \ \ and \ \ Y = y) \tag{2.3}$$

And if X and Y are independent, it becomes:

$$p(x, y) = p(x)p(y) \tag{2.4}$$

However, often random variables carry information about other random variables. In that case, it is called **conditional probability**, and it is denoted by [3]:

$$p(x|y) = p(X = x|Y = y) \tag{2.5}$$

If $p(y) > 0$, the conditional probability is defined by:

$$p(x|y) = \frac{p(x,y)}{p(y)} \tag{2.6}$$

And if X and Y are independent, it is assumed that Y tell us nothing about the value of X, so the probability is given by:

$$p(x|y) = \frac{p(x)p(y)}{p(y)} = p(x) \tag{2.7}$$

Then, following the definition of conditional probability is the theorem of **total probability**. It is defined as follows, where the equation 2.8 represents the definition for discrete systems and the equation 2.9 represents the continuous systems.

$$p(x) = \Sigma_y p(x|y)p(y) \tag{2.8}$$

$$p(x) = \int p(x|y)p(y)dy \tag{2.9}$$

If $p(x|y)$ or $p(y)$ are equal to zero, then the product $p(x|y)p(y)$ is also zero, regardless of the value of the remaining factor.

Of equal importance is the **Bayes rule**, which relates a conditional dependence $p(x|y)$ to the "opposite" dependency, $p(y|x)$. As in the total probability theorem, the rule requires $p(y) > 0$, as it acts as a **normalizer**, making sure the result is not greater than 1. The equation 2.10 represents the definition for discrete systems and the equation 2.11 represents the continuous systems.

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} = \frac{p(y|x)p(x)}{\Sigma_{x'} p(y|x')p(x')} \tag{2.10}$$

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} = \frac{p(y|x)p(x)}{\int p(y|x')p(x')dx'} \tag{2.11}$$

The probability $p(x|y)$ showed above is called the **posterior probability distribution** over X. Thus the Bayes rule provides a convenient way to compute a posterior probability using the "opposite" conditional probability $p(y|x)$ [3]. That said, if one is interested in inferring a quantity x from sensor y, the Bayes rules allows that through the calculation of the probability of data y assuming the value x.

Now, one can condition the rule for combining probabilities of independent random variables on other variables:

$$p(x|z) = p(x|z, y) \tag{2.12}$$

$$p(y|z) = p(y|z, x) \tag{2.13}$$

This rule applies in every case where the variable y carries no information about the variable x if the third variable z is known. That is called a **conditional independence**, that must not be confused with **absolute independence**.

The last two concepts to be treated here are the **expectation** of a random variable and its **covariance**. The first is given by the equations 2.14 and 2.15, for discrete systems and continous systems respectively, and the latter is given by 2.16.

$$E[X] = \Sigma_x x p(x) \tag{2.14}$$

$$E[X] = \int x p(x) dx \tag{2.15}$$

$$Cov[X] = E[X - E[X]^2] = E[X^2] - E[X]^2 \tag{2.16}$$

Using the concepts explained above, one can now understand how the robot percept the environment using sensors. **Perception** is the robot's ability to use sensor measurements to obtain information about the state of the environment it is in. The result of that interaction is simply called measurement, but sometimes can also be referred to as observation or percept. The robot may also keep a record of all past sensor measurements, and that is called **data**. Therefore, the environment measurement data provides information about the momentary state of that environment.

The robot can also record all past **control action**, which is called **control data** and carries information about the change of state in the environment. An alternative source of data are odometers, a specific type of sensor that measures the revolution of a robot‘s wheels. As such, they provide information about a change of state, thus being considered control data.

The evolution of state and measurement is governed by probabilistic law, as said by Thrun [3]. Assuming that a state x is complete, meaning it is the best predictor one can have about the future, its information about past measurements and states is sufficient. In particular, x_{t-1} is a

sufficient statistic of all previous measurements and controls up to this point in time. Other than that, only the control signal u_t matters if the state x_{t-1} is known.

Writing that in probabilistic terms, one can have the following equality, which is also an example of conditional independence:

$$p(x_t|x_{0:t-1}, z_{1:t-1}, u_{1:t}) = p(x_t|x_{t-1}, u_t) \tag{2.17}$$

The right half of the expression above, equation 2.17, is called the **space transition probability**. It specifies how the environmental state evolves over time as a function of robot controls u_t. The environment must be considered stochastic, so the expression must be a probability distribution, not a deterministic function. Although it is indexed regarding the variable t, meaning time, sometimes the state transition is not time-dependent, so it can also be indexed as $p(x'^{u,x)}$, where x' is the successor and x is the predecessor state.

2.2.3 Gaussian Filters

This section will now introduce an important and specific family of recursive state estimator, called **Gaussian Filters**. This type of filter is based on the implementation of Bayes Filters for continuous spaces, but those will not be explained here for the sake of keeping this work brief. Gaussian filters are also the most popular family of techniques up to date, according to Thrun [3], and all Gaussian techniques share the basic idea that beliefs are represented by multivariate normal distributions. The equation 2.18 is the same as the equation 2.2.

$$p(x) = (2\pi\Sigma)^{-\frac{1}{2}} exp\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\} \tag{2.18}$$

Two sets of parameters characterize the density over the variable x:

- The mean μ, that is a vector with the same dimensionality as the state x;
- The covariance Σ, that is a quadratic matrix, symmetric and positive-semidefinite, and its dimensionality is the same as the state x squared.

The idea that one may have to represent the posterior estimation by a Gaussian has essential characteristics to be considered. First, Gaussians are unimodal, meaning they possess a single maximum. This characteristic is suitable for most robotic problems since the posterior is usually focused around the true state with a small margin of uncertainty [3]. However, it is a poor choice for many global estimation problems since many distinct hypotheses may exist.

Entering now a bit more in the topic of interest, the **Kalman Filter** will now be introduced. It is the most popular and probably the most studied technique for implementing Bayes Filters, being

invented by Swerling in 1958 and Kalman in 1960, thus the name. It is basically a technique for filtering and predicting linear Gaussian systems, implementing belief computation for continuous states. Thus, it does not apply to discrete or hybrid state spaces.

The first important concept is the **Gaussian Posterior**. The Kalman filter represents the belief in terms of time t, the belief µt and the covariance Σ_t. The posterior consists of the following affirmations:

1. The state transition probability, defined in the equation 2.19, must be a linear function in its arguments with added Gaussian noise, as shown in the equation 2.20, where x_t and x_{t-1} are state vectors, u_t is the control vector, and both A_t and B_t are matrixes. The vectors are represented as in 3.1.

$$p(x_t|u_t, x_{t-1}) \tag{2.19}$$

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t \tag{2.20}$$

$$x_t = \begin{bmatrix} x_{1,t} \\ x_{2,t} \\ ... \\ x_{n,t} \end{bmatrix} \text{ and } u_t = \begin{bmatrix} u_{1,t} \\ u_{2,t} \\ ... \\ u_{n,t} \end{bmatrix}$$

Thus, as described by Thrun [3], if one multiplies the satate and control vector by A_t and B_t, respectively, the state transition function becomes linear in its arguments. So, the Kalman filter assumes a system with linear dynamics. The random variable ϵ_t is a Gaussian vector that describes the uncertainty introduced by the state transition. It has the same dimensions as the state vector, its mean is zero and R_t will denote its covariance.

The mean of the posterior state is given by the equation 2.21 and the covariance R_t.

$$p(x_t|u_t, x_{t-1}) = det(2\pi R_t)^{-\frac{1}{2}} exp\{-\frac{1}{2}(x_t - A_t x_{t-1} - B_t u_t)^T T_t^{-1}(x_t - A_t x_{t-1} - B_t u_t\} \tag{2.21}$$

2. The measurement probability $p(z_t|x_t)$ must also be linear in its arguments, with an added Gaussian noise, where C_t is a matrix with dimensions kxn, k being the dimension of the measurement vector z_t. The vector δ_t describes the measurement noise, and its distribution is a multivariate Gaussian with zero mean and covariance Q_t [3].

$$z_t = C_t x_t + \delta_t \tag{2.22}$$

The measurement probability is then given by the equation 2.23 [3].

$$p(z_t|x_t) = det(2\pi Q_t)^{-\frac{1}{2}} exp\{-\frac{1}{2}(z_t - C_t x_t)^T Q_t^{-1}(z_t - C_t x_t)\} \tag{2.23}$$

3. Last but not least, the initial belief $bel(x_0)$ must have a normal distribution, where μ_0 is the mean and Σ_0 is the covariance. The belief is represented as shown in the equation 2.24 [3].

$$bel(x_0) = p(x_0) = det(2\pi\Sigma_0)^{-\frac{1}{2}} exp\{-\frac{1}{2}(x_0 - \mu_0)^T \Sigma_0^{-1}(x_0 - \mu_0)\} \tag{2.24}$$

2.2.3.1 The Kalman Filter Algorithm

The mathematical algorithm, as stated in Thruns book [3], can be seen on the Table 2.1.

	Algorithm Kalman_filter(μ_{t-1}, Σ_{t-1}, u_t, z_t)
1:	$\overline{\mu_t} = A_t\mu_{t-1} + B_t u_t$
2:	$\overline{\Sigma_t} = A_t\Sigma_{t-1}A_t^T + R_t$
3:	$K_t = \overline{\Sigma_t}C_t^T(C_t\overline{\Sigma_t}C_t^T + Q_t)^{-1}$
4:	$\mu_t = \overline{\mu_t} + K_t(z_t - C_t\overline{\mu_t})$
5:	$\Sigma_t = (I - K_t C_t)\overline{\Sigma_t}$
6:	return μ_t, Σ_t

Table 2.1: Kalman Filter Algorithm, step by step, for linear Gaussian state transitions and measurements.

Kalman filters uses as input the belief at time $t-1$, represented by μ_{t-1} and Σ_{t-1}. To update its parameters, it requires the control signal u_t and the measurement z_t. Then, the output is the belief at time t, represented by μ_t and Σ_t. As stated by Thrun [3], the update of the mean value is done using the deterministic version of the state transition function 2.20, and the update of the value of the covariance is considering the fact that states depend on previous states, doing that through the matrix A_t. Then, the **Kalman Gain** is the variable K_t. It specifies the degree of incorporation of the measurement into the new estimated state.

Seeing the algorithm shown in the Table 2.1, one may see that the Kalman filter is computationally quite efficient, as stated by Thrun [3], so it makes sense that it is the most popular algorithm in probabilistic robotics today.

2.3 Deep Learning

Deep Learning is a specific type of machine learning. As the name suggests, it is a deeper application of the concepts of machine learning, so to understand the concepts of deep learning properly, it is necessary to introduce machine learning basics.

2.3.1 Machine Learning Basics

Currently, it is common for one to find oneself with an absurd amount of data available about almost any topic. That is because we are in the era of big data. This situation calls for automated methods of data analysis, or data science as some may call it.

That automated method is what machine learning provides us. In particular, it can be defined as a combination of methods that detect patterns in data automatically and then uses the uncovered patterns to predict, or better-said estimate, future data. Besides that, it can also be used to perform several kinds of decision making under uncertainty.

Knowing that, it becomes evident that a good way to solve such problems is to use probability theory, as that can be applied to solve any problem involving uncertainty. In machine learning, the uncertainty can come in several forms: what is the best model for a data set? What is the best prediction? What should be the subsequent measurement to assure the best set of data?

One can divide machine learning into three main types, even though the last one is still barely used. The first and simplest one is the **predictive** or **supervised learning** approach, where the goal is to learn a mapping from the given inputs to the given outputs, starting from a labelled set of input-output pairs. This set is called the **training set**. This type is equivalent to giving someone the question and the correct answer to learn a subject [4].

One simple example given by Murphy (2012) [5] is that each training input is a D-dimensional vector of numbers representing the height and the weight of a person. Those are called **features** (or **attributes** or even **covariates**). In general, the input set will be a complex structured object. The output or **response variable** can be anything, but it is usually assumed to be a **classification**, a **pattern recognition** or a **regression**.

The second type of machine learning is named **unsupervised learning** or **descriptive learning**. In this one, the goal is to find interesting patterns to solve the problem at hand. Because of that, it can also be sometimes called **knowledge discovery**. This kind of problem is much less defined than the first one since the algorithm does not know which patterns to look for, and there is no obvious error metric to be used. As the name suggests, it is indeed unsupervised, meaning that the machine is left to decide which patterns are more important and how it will define them. This is equivalent to telling someone to learn about machine learning and leaving one to figure out by oneself how to do it and which books to use.

The third and last type, and also somewhat less commonly used one, is called **reinforcement learning**. As the name suggests, it is similar to the reinforcement teaching commonly used with kids, where there is some occasional reward or punishment signal associated with the machine's

behaviour. This type will not be explored further in this work since it is not relevant to the other concepts used here.

Two other concepts that are worth describing is **overfitting** and **underfitting**. Those are problems that one working with neural networks needs to be careful about since they can ruin the whole project if not considered. **Overfitting** is when the function describing the pattern is too fitted to the training data. The function can perfectly describe the input data, but when faced with a different set of data, it gives the wrong results. **Underfitting** is usually the problem encountered when trying to avoid overfitting, being when the function is too general and can not even describe properly the training data so that a different set will give an even worse result. There are several methods for fixing those problems, but they are unfortunately out of the scope of this work. However, it can be found in Murphy's book [5], and on Goodfellow's book [4].

2.3.2 Convolutional Networks

Convolutional Networks, also known as **convolutional neural networks** (CNNs), were first introduced by Yann LeCun in 1989. It is a specialized kind of neural network for processing data known to have a grid-like topology. Two simple examples of this type of neural network can be time-series data and image data, the first being represented as a 1D grid with samples at regular time intervals and the latter a 2D grid of pixels.

A convolutional neural network employs the mathematical operation **convolution**, which is a specialized type of linear operation. A convolution network can be described simply as a neural network that uses a convolution in place of general matrix multiplications in at least one of their layers, according to Goodfellow [4].

Using convolution offers a couple of advantages since it is suitable for working with inputs of variable sizes. That brings the possibility of working with sparse interactions, parameter sharing and equivariant representations.

2.4 Previous work

This work was based on the work done previously by another student, Thúlio Noslen. As described by him in [2], the work "proposed the use of a deep LSTM network in order to create a steering assistance module from data collected from an experienced pilot". The robot model used was the Pioneer 3AT, the same one used in this work, as is explained in Section 3.7.

To do so, the first step was to create a database composed of the input data from an experienced pilot, composing the ideal trajectory for each route. The second step was to propose, train and validate two deep LSTM networks, one for each route, that could learn the patterns of the ideal driving. The third and last step was to validate the network with inexperienced users in real-time.

As said earlier, two routes were used: one with an eight-shape trajectory and another that goes around one of the buildings in UnB. The routes can be seen, respectively, in Figures 2.7 and 2.8. Each trajectory was used to train one LSTM network that learned the driving patterns of an experienced driver and could then help an inexperienced user drive a softer and overall better trajectory.

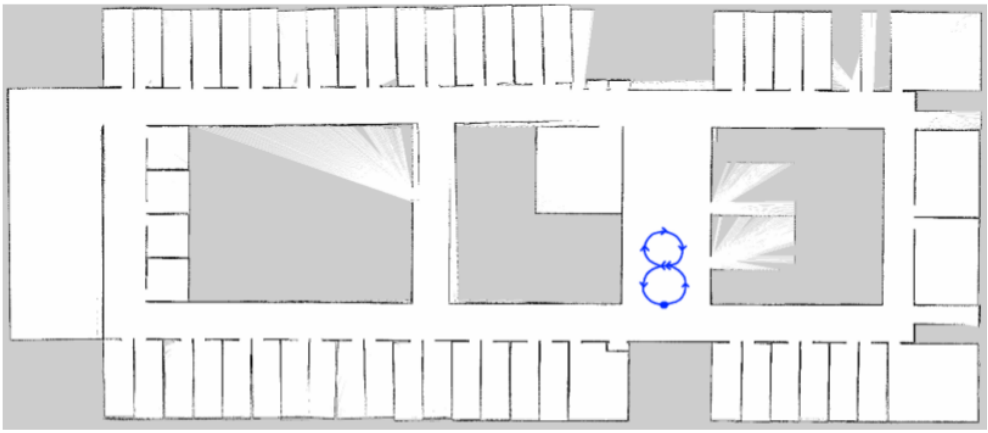

Figure 2.7: Route in an eight shape - Source: Thúlio Noslen [2].

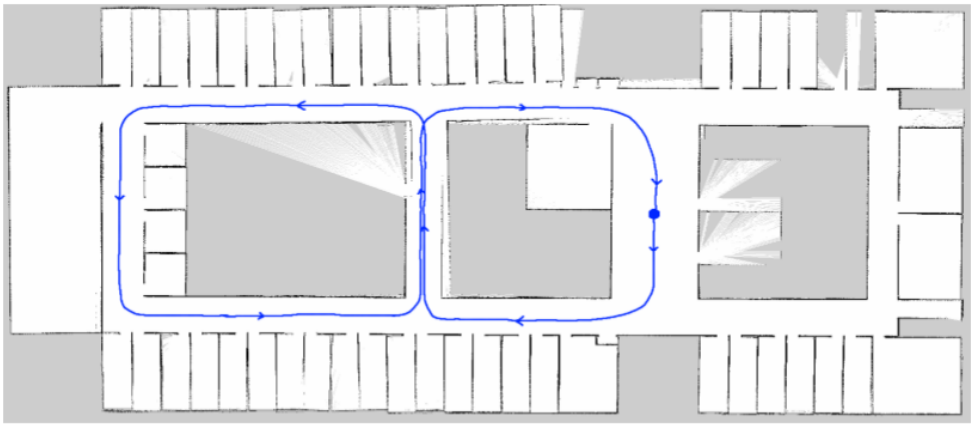

Figure 2.8: Complete route - Source: Thúlio Noslen [2].

When validating the proposed algorithm with the inexperienced users, along with the algorithm

turned off, four simple signal fusion were implemented:

- **Network with 0.05 threshold:** if the difference between the user input and the network input is bellow 5%, then the user input is used directly as the output. If it is above 5%, the network input is used as output.
- **Network with 0.20 threshold:** if the difference between the user input and the network input is bellow 20%, then the user input is used directly as the output. If it is above 20%, the network input is used as output.
- **Mean with 0.05 threshold:** if the difference between the user input and the average between the user input and the network input is bellow 5%, then the user input is used directly as the output. If it is above 5%, the average is used as output.
- **Mean with 0.20 threshold:** if the difference between the user input and the average between the user input and the network input is bellow 5%, then the user input is used directly as the output. If it is above 5%, the average is used as output.
- **Off:** the user input is used as output directly.

With the networks trained and the signal fusion methods implemented, validation with inexperienced users could be made. This validation was done with 14 users, each one testing two different fusion methods driving the robot in person. During the tests, the user input, network input and system output data was recorded, allowing both the analysis was done by Noslen in [2] and [6] and the execution of this present work. The routes and the database created with the user validation step are explained in further details Section 3.3.

Noslen's full work can be seen in [2] and [6].

Chapter 3

Methodology

This chapter explains the mathematical model proposed for the system; the implemented Kalman Filter Algorithm; the database used to validate the Kalman Filter implemented, and how the tests were carried out to do the final validation.

3.1 System Modeling

The system used in this work can be seen in Figure 3.1 and is composed of four main parts:

- **User:** the user is the main input of the system since it feeds both the signal fusion system and the neural network;
- **Neural network:** has the user commands as input and outputs the corrected command, being the other input to the signal fusion system.
- **Signal fusion system:** in this work, the signal fusion system studied is the Kalman filter, which receives two input signals and outputs one fused signal.
- **Robot:** in this diagram is represented by a car.

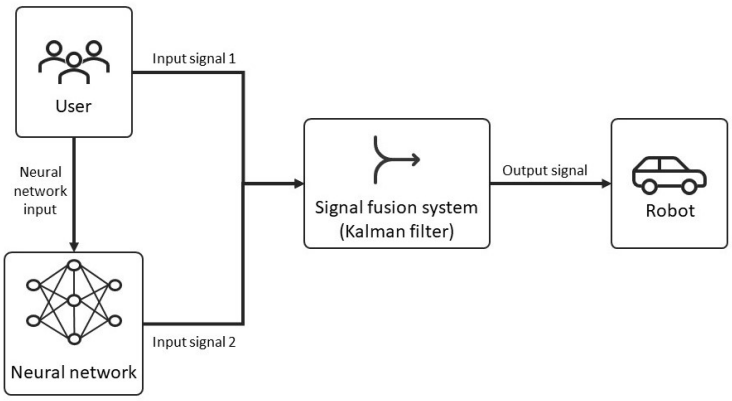

Figure 3.1: Simplified diagram of the modeled system.

The system shown in Figure 3.1, in order to work with the Kalman Filter, needs to be in the space states format, as shown in the Equation 2.20. The specific model for this system is described by the Equation3.1, that is an adaptation of the Equation 2.20. The matrix A is shown below, and the matrix was actually divided into B_1 and B_2. That division was made to simplify the modelling process and to the process to modify the weights later. The vector with the inputs u_t was also divided since there are two inputs. One is the neural network output, and the other is the user input. That could also be a single vector, as the matrix B could be only one, but it was divided to simplify the modelling process.

$$x[k+1] = Ax[k] + B_1u_1[k] + B_2u_2[k] + \epsilon[k] \tag{3.1}$$

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},\ B_1 = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \text{ and } B_2 = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix}$$

The matrices B_1 and B_2 are identical because the first implementation considers that the network and the input from the user should have the same weight on the output signal.

The output is given by the Equation 3.2.

$$y[k] = Cx[k] \tag{3.2}$$

3.2 Implementation and validation of the Kalman Filter

The language chosen to implement the Kalman Filter Algorithm was Python, and it was chosen due to its simplicity and number of relevant libraries in the data analysis field, as well as being compatible with ROS (Robot Operating System). ROS is the software used to implement the system proposed working with the robot, as will be explained in Section 3.5.

The algorithm is based on the mathematical model showed in Subsection 2.2.3.1. The noises used are Gaussian white noises. The observation noise is constant and does not depend on the data set analysed. The covariance used to generate the noise is constant and set to $5e^{-4}$ for the X-axis and $5e^{-3}$ for the Y-axis. Those values are completely arbitrary and will be validated on the next step of the project when the algorithm is implemented with the robot. The noise of the process depends on the data set used, so it is calculated using the covariance of the data.

The recursive part of the algorithm is implemented considering the system is online so that it can be implemented on the robot without the need for many adaptations. To implement that, the algorithm gets one set of data at a time, as if the user and the neural network had just generated it, and works as if it does not have access to "future" data. The covariance is re-calculated each time and used to generate the process noise. Then, both the process noise and the observation

noise are used to calculate the other matrices and the updated state. The updated state is then computed and plotted in a graph.

The output of the filter is the mean of the updated state, but it must be validated to determine the best approach to be used with the robot.

3.3 Database

The database used to validate the proposed model was made by another student, Thúlio Noslen [2], and contains all the data obtained during the experiments done by him separated into folders by user.

The robot used was a Pioneer 3AT, and the input method was an Xbox One controller with a built-in joystick. The value ranges for the joystick signal the ones shown in Table 3.1. One crucial observation is that the axes for the robot and the control are reversed, so in the robot, the y-axis means the robot is turning (left or right), and the x-axis means backwards and forward.

Table 3.1: Ranges of values of the output signal from the Joystick.

Axis	**0 to 1**	**0 to -1**
X	Turn to the right	Turn to the left
Y	Forward	Backward

The experiment consisted of letting an inexperienced user pilot the robot through two routes, the first is through almost the whole second floor of the Computer Science building at the University of Brasília, as shown in Figure 3.2, and the second is doing an eight-shaped route at the same building, as shown in Figure 3.3.

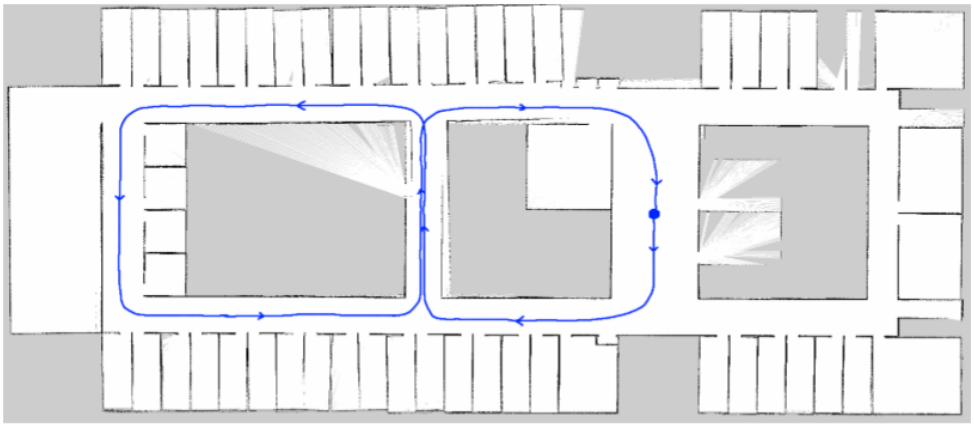

Figure 3.2: Complete route - Source: Thúlio Noslen [2].

The neural network used to build this database was trained by Thúlio Noslen, and the method used to do it can be found in his work [2]. It was trained by a good pilot, whose piloting is assumed to be soft, without sudden manoeuvres, there is no acceleration in straight lines and the curves are done with small acceleration, regardless of the robot's velocity.

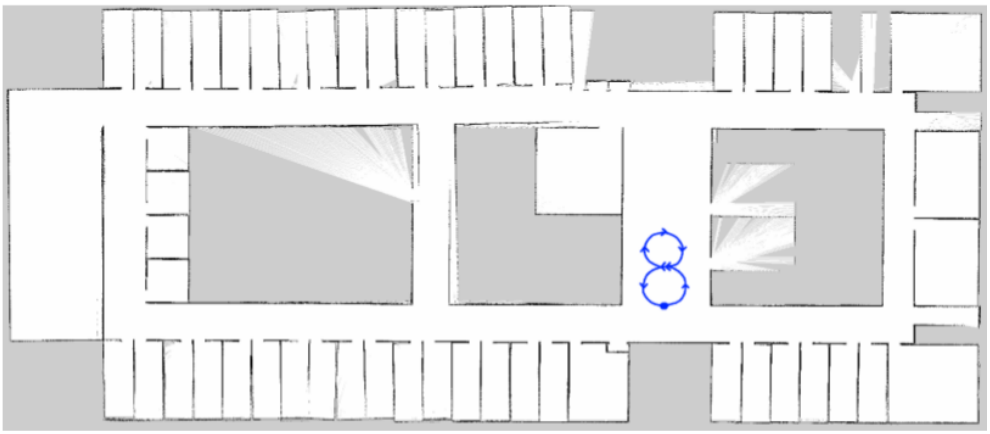

Figure 3.3: Route in an eight shape - Source: Thúlio Noslen [2].

The database has a few variables in it, all listed and described in the Table 3.2.

Table 3.2: Data Base Variables.

Variable	**Description**
datetime	saves the date and time the experiment was performed
enabled	shows if the system is enabled or if the user is driving on its own
network	shows which network is being used
method	method used to combine the two inputs for the robot
thresh	saves the value of the threshold used to merge the signals
diff	difference between the two inputs for the robot
acting	shows if the network is acting on the movement or not
vx	robot's velocity in x
vz	robot's velocity in z
jb	enable button for the joystick
jpx	joystick command in x
jpy	joystick command in y
jcx	neural network command in x
jcy	neural network command in y
jox	output command signal in x
joy	output command signal in y

The data described is then used as input to the Kalman Filter algorithm to generate an output to feed the robot.

3.4 Quantitative validation of the model

It is necessary to validate the proposed model analytically before testing it in simulations to ensure the Kalman Filter is working properly. To do so, the quantitative validation was done by

comparing the Kalman Filter's output signal with the ones from Thúlio's work [2]. The database is defined in Section 3.3.

The comparison was made in two steps:

1. Comparing the graphs generated by the program with the output from the database and the output of the implemented filter. The updated states in X and Y are also analysed to make sure the results are coherent considering the path.
2. Analysing the mean error between the outputs from the database and the output of the implemented filter [7].

The following guidelines guided the graphical comparison:

- The comparison between the output obtained from the database and the updated states, the expected results are for the two data sets to be close, but not identical. Both graphs should overlap or at least partially overlap.
- The analysis of the updated states in the eight-shaped route, for the states in X, the expected result is something similar to a squared wave, but with much noise. For the updated states in Y, it is expected to be close to one most of the time.
- The analysis of the updated states in the building route, for the states in X, the expected result is a signal with some values close to one and minus one, but most of it should be around zero. For the states in Y, the expected result is similar to the eight-shaped route, so it is expected to be close to one most of the time.

The values showed in the graphs have the same format as the input signal from the user, which means it has the same format as the output of the joystick used to make the database. The maximum value is 1, and the minimum is -1. On the X-axis, 1 means left and -1 means right, and on the Y-axis, 1 means forward and -1 means backwards. If the value is around zero, it means there is almost no imposed acceleration.

The mean error was calculated using the updated states in each axis (X and Y) and their respective values from the database, and then the percentage value of this error is calculated so that it is possible to judge the results as good or bad following a pre-established maximum value [7].

The mean error for each axis should then be calculated using the relation shown in 3.3. It is necessary to use $(Updated\ state)_{i+1}$ because the states calculated with the Kalman filter have an offset of one in the algorithm implemented.

$$Average\ \ error = \frac{\Sigma_{i=0}^{n-1}[(Updated\ state)_{i+1} - (Database\ state)_i]}{n} \tag{3.3}$$

The following criteria should guide the mean error comparison:

- An error smaller than 5% is considered acceptable since the outputs are supposed to be close, but not identical;
- The analysis should be made comparing the X-axis and Y-axis separately so that it is possible to tell in which axis the modelled system needs adjustments.

The results of this quantitative validation can be found in Chapter 4, Section 4.1.

3.5 Softwares used

The software used to simulate the robot were ROS (Robot Operating System) and Gazebo. The first one is, as described in its website [8], "an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers". The second, Gazebo, is a robust 3D robot simulator that has, according to its website [9], several features, such as dynamics simulation, sensors and noise, robot models and extensive command-line tools. Also, it seamlessly integrates with ROS. In addition to that, there is a package ready to use with the Pioneer robot model used in this work, that will be described in subsection 3.7, called p3at_tutorial.

The specifications of the software used to implement the simulation were selected due to the package used to implement the robot, described in subsection 3.7 and already employed by Thúlio [2]. The specifications can be seen in Table 3.3.

Software	**Specifications**
Operating system	Ubuntu 18.04
ROS	ROS Melodic
Gazebo	Gazebo 9
Python	Version 2.7

Table 3.3: System software specifications.

3.6 Implementation of the Kalman Filter Algorithm for Simulation

The proposed model was validated (Section 4.1), so the Kalman Filter Algorithm implemented could then be adjusted to work in simulation with the robot. To implement that, the system implemented by Thúlio in his work [2] is used, including the Kalman Filter as a new method of signal fusion.

The implemented Kalman Filter, previously described in Section 3.2, was adjusted in order to

receive the data from the simulation and act as an online signal fusion method. To implement that, the following modifications had to be made:

- Only the values from instants *k* and *k-1* are saved in variables and used in the calculations, and the previous values are saved for future analysis.
- The mean, used to calculate the covariance and both process and observation noises, is calculated cumulatively and considers all previous values. This calculation was implemented for the sake of simplicity, and, as stated by Chui in [10], it is a convenient way to calculate the mean for real-time applications using Kalman filtering.

3.7 System implemented

The robot model used in this work is the Pioneer 3-AT, a general-purpose mobile robot with four steerable wheels with rough tires, as described in ROS website [11]. This model can be used in ROS using a few drivers, such as "rosaria", and both model and launch files are available to download on ROS website [12]. The package used to implement these simulations is called p3at_tutorial and can be found to download on authors GitHub [13]. This package was chosen because it is very stable and well documented, so the use is simple and easy to combine with the remaining work since it uses the node cmd_vel to implement the robots' movements.

Since the original work done by Thúlio [2] used an Xbox One controller as an input method, the same one was used here in the first phase, and no changes were made to the input algorithm used by him in his work. The initial idea was to perform all tests using this method, but since the pandemic requires social distancing and in-person encounters are not recommended, the second phase was done remotely using the user's keyboard as input. For the implementation using the controller as the input method, the buttons were mapped in the input algorithm, as shown in Table 3.4.

With the model defined and working, it was necessary to implement the robot's control. Because of the Coronavirus pandemic, it was necessary to divide the tests into two phases:

- **Phase 1:** was performed in person and with a group of 13 users, using an Xbox One controller as the input method;
- **Phase 2:** was performed remotely with a group of 20 users, using the keyboard as the input method.

Using the keyboard as the input method, the second phase required a few adjustments in the input method algorithm originally used by Thúlio. The main difference between those input methods is that the keyboard is a discrete signal composed of several step functions that can have significant distances between values of consecutive sampling times. At the same time, the Xbox controller is much closer to a continuous signal, allowing the user greater control over the changes in the input signal. This relation between discrete and continuous signals can be seen

Button name	Index	Used to:
Left joystick	0 and 1	Control the robot
Power	8	Turn system on and off
Left bumper	4	Enable movement
X	3	Switches network
A	0	Switches method
B	1	Switches threshold
Right bumper	5	Marker

Table 3.4: Xbox One control button mapping.

in more detail in the references [14], and [15] and the description of the conversion of analogical to digital signal, done for the Xbox controller, in this case, can be seen in [16]. Aside from the adjustments made, it is important to remember that since the system was originally made to work with a continuous input signal from the Xbox controller, which also allows a fine adjustment of the trajectory, different behaviour can be expected when using a discrete signal coming from the keyboard. This difference in behaviour are detailed in Chapter 4, specifically in Subsection 4.2.2.

As with the controller, it was necessary to map some keys to control the robot, as can be seen in Table 3.5. Since the main goal here was to simulate a continuous signal such as the one from the controller, the approach was different from what is usually done when controlling a robot with the keyboard. There are two main differences here:

- The first is that four extra keys were added to the system to simulate the diagonal position of the joystick in the controller. This means a signal is sent both on the X-axis and the Y-axis simultaneously, allowing the robot to have an angular velocity and take turns and go forward.
- The second is that the velocity is not increased by clicking a specific key once, but instead is increased by repeatedly clicking on the key of the direction the user wants to increase the speed. That aimed to simulate the fine adjustment achieved with the joystick since the velocity would increase by 5% at a time.

3.8 Simulation and tests

As stated in Sections 3.5 and 3.7, the simulation and tests were divided into two phases: the first was done in person, using an Xbox controller as the input method, and the second was done remotely, using the user's keyboard as the input method. Both were executed using a virtual machine with Linux Ubuntu, as described in Section 3.5, the only difference being the input method. This implied changing the input algorithm, and, for the second phase, another software was required to allow the user to access the computer and use the robot remotely. To implement that, a software named Parsec [17], commonly used to play shared-screen games remotely was

Key	X	Y
i	1	0
j	0	1
l	0	-1
,	-1	0
u	1	1
o	1	-1
.	-1	-1
m	-1	1

Table 3.5: Keyboard key mapping.

used to allow the user to access the computer, seeing the screen and using their keyboard as input with low latency. This software was chosen for three main reasons:

- It works on the main modern operating systems, such as Windows 7+, macOS 10.11+ and Ubuntu 18.04, however hosting is available only for Windows 8.1+;
- It allows low latency peer-to-peer connection, delivering 60 fps HD video over the network and virtually lag-free [17];
- It is free to use and lightweight, so it would be easy for any of the users to download and use and would not compromise too much the host computer computing system.

To validate the Kalman filter as a signal fusion method, it was necessary to compare it with other signal fusion methods. To allow that, two more straightforward methods were used:

- **Network with threshold of 0.05:** if the difference between the users' input and the network's input is smaller than the chosen threshold of 0.05 (5%), then the user input is used directly as input for the robot. If it is equal to or bigger than the threshold, then the network's input is used directly as input for the robot.
- **Mean with threshold of 0.05:** if the difference between the users' input and the network's input is smaller than the chosen threshold of 0.05 (5%), then the user input is used directly as input for the robot. If it is equal to or bigger than the threshold, the arithmetic average of both input signals is used as input for the robot.

Since the objective here is to find which method the user prefers, the tests had to be random to ensure the results are not biased. That said, six test sequences were predefined, and each user would get a different one so that each method would be the first one, the middle one and the last one for at least one of the users. Every user had one round to learn how to use the controller or the keyboard and get used to the simulation. After the first round, the user would start the tests following the assigned sequence. The sequences used can be seen in Table 3.6. Also, as explained in further details in Section 3.9, each test received a colour code so that the user's opinion would not be influenced by the method implemented. The colour code is as follows:

- **Network with threshold of 0.05:** red;
- **Mean with threshold of 0.05:** yellow;
- **Kalman filter:** green.

Test	Order
A	Red, yellow and green
B	Red, green and yellow
C	Yellow, red and green
D	Yellow, green and yellow
E	Green, red and yellow
F	Green, yellow and red

Table 3.6: Test sequences.

3.8.1 Trajectory

The user's path was established in the simulation using traffic cones, which indicate where the curves should happen. Figure 3.4 shows a perspective view of the path that will be executed, being possible to differentiate well the objects in the scene. Figure 3.5 presents the trajectory considered ideal for carrying out the indicated route, making a zigzag between the six cones equally spaced, going around the last cone and then returning to the beginning.

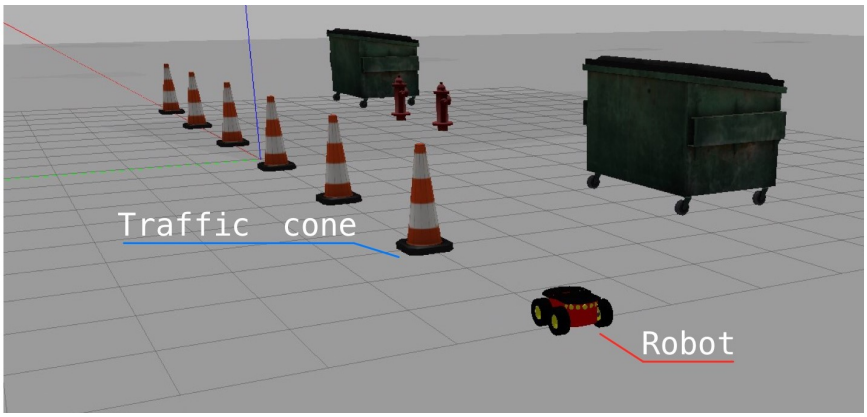

Figure 3.4: Perspective view of the robot path.

This trajectory is similar to the eight-shaped trajectory implemented by Thúlio [2], but longer and slightly more complicated.

Figure 3.5: Superior view of the trajectory, with indication of the ideal route.

3.8.2 First phase: using the Xbox controller

For the tests executed in person, each user received the following set of information:

- The main goal is to complete the trajectory as smoothly as possible, going between the cones until the end and coming back to the start point;
- In order for the system to work, one has to keep pressing the left bumper of the controller at all times, since as soon as one lets it go, the robot will stop responding;
- One has one round, the first one, to learn how to use the robot, from the second round forward the tests will begin;
- One will not be aware of the method being tested only after the questionnaire is responded to to avoid any bias.

After receiving the initial information, the users could begin the simulation, starting with the learning round. Since some of the users presented some difficulties using the controller, sometimes it was necessary to abort the test and start over, as will be further explained in Chapter 4.

3.8.3 Second phase: using the keyboard

For the tests performed remotely, each user received the following set of information:

- The main goal is to complete the trajectory as smoothly as possible, going between the cones until the end and coming back to the start point;
- The keys you can use to control the robot are the following:
  - i: go forward;

  - j: turn left;
  - l: turn right;
  - u: turn left while going forward;
  - o: turn right while going forward;
  - space bar: stop the robot.

- One should prioritize using u and o while driving and moving backwards is not allowed;
- One has one round, the first one, to learn how to use the robot, from the second round forward the tests will begin;
- One will not be aware of the method being tested, only after the questionnaire is responded, to avoid any bias.

As explained for the tests in the first phase, in Subsection 3.8.2, after receiving the initial information, the users were allowed to begin the simulation, starting with the learning round. The difference in the second phase was that at the end of each round, the whole system would be restarted to guarantee all rounds for all users were the same, and the robot always began in the same position. That was necessary here and not in the in-person tests due to the considerable increase in difficulty, so to avoid differences between users, it was set as the standard for all tests. This will be further explained in Chapter 4.

3.9 Evaluation questionnaire

The questionnaire was used to acquire the user's perception of which method was the most comfortable to drive the robot with and which one allowed to perform smoother curves. To achieve that, the survey must question if the user about previous experience with robots or heavy machinery and experience with the use of joystick or keyboard for games, depending on the type of test the user performed. According to Meyer in [18], it is essential to evaluate whether the user knows how to use the system being tested and is comfortable with it since both knowledge and comfort with the system can increase or decrease the difficulty in its use and even jeopardize its functioning.

The questionnaire was always applied at the end of the test after all three methods were executed, and the user only knew the colour code so that their responses would not be biased.

Since the tests were divided into two phases, it was necessary to use two different surveys. The main evaluation is the same in both, but the survey for the second phase had a few more questions to evaluate if the internet connection interfered with the execution of the tests and to verify if the user was able to complete all four rounds. The questions for each questionnaire can be seen in the following subsections.

3.9.1 Questionnaire for tests in first phase, using the Xbox controller

The questions used in this questionnaire where all in Portuguese, since the all the tests were carried out in Brasília, Brazil, so they are translated to English here. The original questions, in Portuguese, can be found in the attachments at the end of this document.

The first set of questions aims to acquire general information about the user, with some socioeconomic questions. After that, there are three questions about previous experience with video games and using a joystick.

1. How old are you?
    - Less than 20 years old;
    - 21 to 25 years old;
    - 26 to 30 years old;
    - 31 to 35 years old;
    - 36 to 40 years old;
    - 41 to 45 years old;
    - 45 to 50 years old;
    - 51 to 55 years old;
    - 56 to 60 years old;
    - More than 60 years old.
2. Which gender do you identify with the most?
    - Female;
    - Male;
    - I prefer not to declare.
3. What is your level of education?
    - Incomplete elementary school;
    - Complete elementary school;
    - Incomplete high school;
    - Complete high school;
    - Incomplete higher education;
    - Complete higher education;
    - MBA, postgraduate or specialization;
    - Master;
    - PHD.
4. Do you have experience with operating robots or machinery?

- Yes;
- No.

5. How familiar are you with video games?
   - I am not;
   - A little;
   - A lot.
6. How skilled are you with the use of joystick controls for games?
   - I am not;
   - A little;
   - A lot.

The main objective of the second set of questions was to collect information regarding the user's general perception of the tests performed. The questions can be seen below.

1. What was the sequence used?
   - A;
   - B;
   - C;
   - D;
   - E;
   - F.
2. Was there a significant difference between the methods used?
   - Yes;
   - No.
3. If so, which one was the most comfortable to drive the robot?
   - Method red;
   - Method yellow;
   - Method green.
4. If not, please describe your experience with the tests performed.
5. Describe the points that you liked the most in the method chosen as the most comfortable one.
6. Besides comfort, on what else did you notice a difference between the methods used?

Finally, the last three sections aimed to evaluate the user's perception of each method tested. Since all three sections have the same questions, changing only the colour of the method evaluated, only the first one will be listed here.

1. What is your perception about the interference of the red method in the guidance of the robot?
    - Very little interference;
    - Little interference;
    - Moderate;
    - Too much interference.
2. How comfortable was it to drive the robot using the red method?
    - Not comfortable;
    - Moderate;
    - Very comfortable.
3. In your perception, the red method should:
    - Interfere more;
    - Interfere less;
    - The interference is adequate.

3.9.2 Questionnaire for second phase tests, using the keyboard

The first five questions are the same used in the first phase questionnaire (3.9.1) with difference only in the 6th question, since in the second phase is critical to know whether the user is skilled with keyboard for games, instead of being skilled with a joystick.

6. How skilled are you with the use of keyboards for games?
    - I am not;
    - A little;
    - A lot.

The main objective of the second set of questions was to collect information regarding the user's general perception of the tests performed. Here, a few questions were added to the second phase questionnaire to acquire information regarding the internet connection and whether the user could finish all the tests. Since the other questions are the same as the ones used for the first phase (3.9.1), only the new ones will be listed below.

2. Did the internet connection interfere with the testing?

- Yes;
- No;
- Could not tell.

3. Was the software used to perform the tests adequate?
    - Yes;
    - No.
4. Were you able to finish all tests?
    - Yes;
    - No.
5. If not, what prevented you from finishing the tests?

At last, the last three sections are the same as the ones used for the in-person questionnaire and can be seen in Subsection 3.9.1.

Chapter 4

Results

This chapter presents the results obtained as well as their critical analysis.

This chapter will present the results of the analysis described in Chapter 3. The first validation was done using the database acquired by Noslen in his work [2], starting with a graphical validation and then doing a percentage mean error analysis. After this validation, since the proposed filter passed all the criteria listed in 3.2, the system was then adjusted to work with the simulated robot in order to allow the second validation.

The simulations were performed with 33 users, and those tests were divided into two phases, described in Section 3.8. This validation consisted of two different analysis:

- **User perception:** it was acquired via a questionnaire, described in Section 3.9. This composes the qualitative validation of the Kalman filter as a signal fusion method.
- **Trajectories:** this analysis composes the quantitative validation of the model, consisting of analysing the data acquired during the simulations and comparing the different methods tests.

4.1 Quantitative validation of the Kalman Filter using the previous database

The quantitative validation of the model using previous data, as explained in Section 3.4, was divided into two stages: using charts and using the percentage mean error. Those results can be seen in the following subsections.

4.1.1 Graphics validation

The graphic validation consisted of comparing the graphs generated with the output from the database and the output of the implemented filter. The updated states in X and Y are also analysed to make sure the results are coherent considering the robot's path. Both outputs were

plotted in the same graph in order to allow the analysis. This was a simple validation to check whether the results were coherent or not, so the data from only two users were used.

The data set presents the results obtained. Two different users were chosen randomly, user **am** and **gm**, and the data from their eight-shaped route and building route was used as a base comparison for this analysis. The data sets used are the ones listed below.

1. User am, eight-shaped route (Subsection 4.1.1.1);
2. User gm, eight-shaped route (Subsection 4.1.1.1);
3. User am, building route (Subsection 4.1.1.2);
4. User gm, building route (Subsection 4.1.1.2);

For the comparison between the output in the database and the updated states in the Kalman filter, the expected results are for the two data sets to be close but not necessarily identical. Both charts should, at least partially, overlap.

Analysing the updated states in the eight-shaped route, for the states in X, the expected result is something similar to a squared wave but with a lot of noise. This result is expected because the updated states in X should have the same format as the signal from the Xbox controller in X. Since this route is the eight-shaped one, one should expect constant movements from one side to the other, mimicking a squared wave. The noises seen are expected because the user can choose to make the curve with more or less intensity by varying the input value in X, so adjustments during the trajectory are expected. The updated states in Y are expected to be close to one most of the time since it represents the linear velocity. Since the robot is moving forward during the test, it is expected to be close to one, whereas that is the maximum value it can achieve.

Finally, in the analysis of the actualised states in the building route, for the states in X, the expected result is a signal with some values close to one and minus one, but most of it should be around zero. This is expected because the building route had mainly straight lines and only a few turns, so the user only needed to change the input signal in X to correct the trajectory during the straight parts or perform the curves. For the states in Y, the expected result is similar to the eight-shaped route, so it is expected to be close to one most of the time.

4.1.1.1 Eight-shaped route

The batch of results from user **am** is showed in Figures 4.1, 4.2 and 4.5. The results from user **gm** are showed in Figures 4.3, 4.4 and 4.6.

The Figures 4.1 and 4.2 show the actualized states in X and Y, respectively, from user **am**. The graphs for user **gm** are shown in Figures 4.3 and 4.4. Analysing the graphs, it is clear that the pattern follows the expected behaviour defined in the Section 3.4. The states in X are in a shape similar to a squared wave and the states in Y are near to one most of the time.

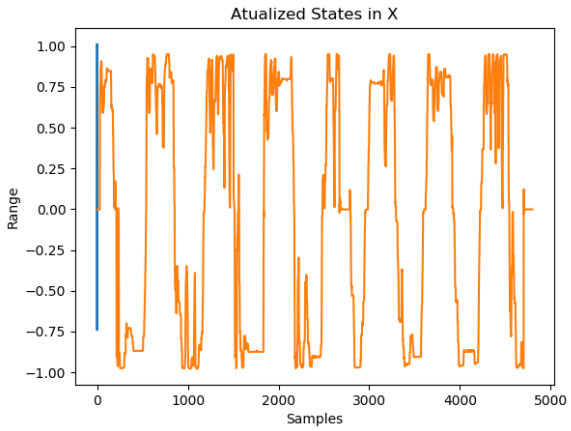

Figure 4.1: Updated states in X - eight-shaped route, user **am**.

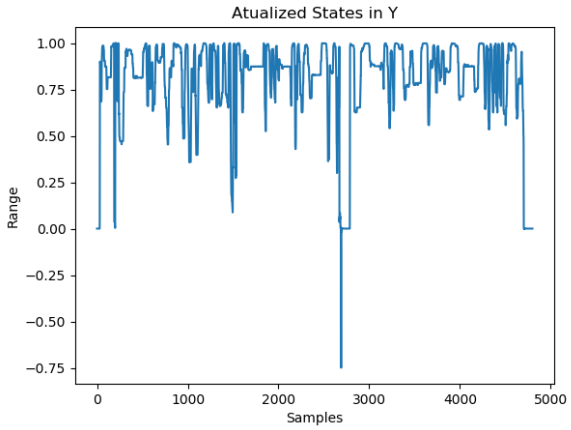

Figure 4.2: Updated states in Y - eight-shaped route, user **am**.

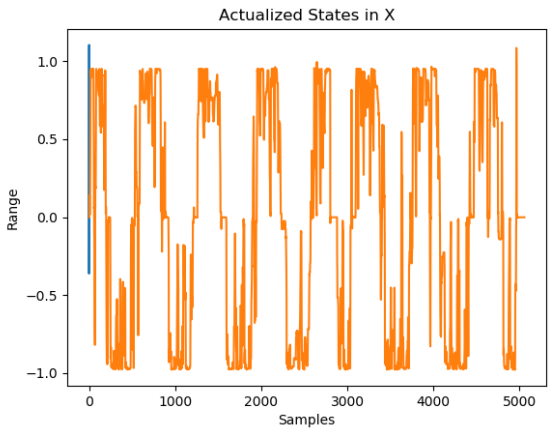

Figure 4.3: Actualized states in X - eight-shaped route, user **gm**.

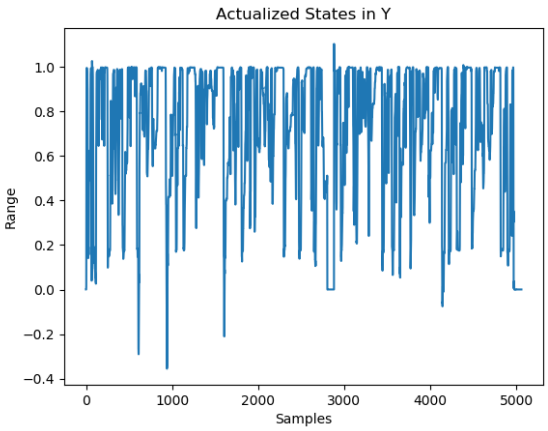

Figure 4.4: Actualized states in Y - eight-shaped route, user **gm**.

The last analysis for the eight-shaped route can be done combining the signals in one graph, as shown in Figures 4.5 and 4.6. This combination results in the representation of the movement the joystick would make in order to create the signals in X and Y, shown in Figures 4.1 and 4.2 for user **am** and in Figures 4.3 and 4.4 for user **gm**. Considering that the route is composed mainly of curves, the user is mostly going forward while constantly turning left and right to perform the eight-shaped route, and this representation is expected to be shaped similar to the upper part of an umbrella. The point in the centre happens whenever the user lets the joystick loose, and it goes back to its original position, and user **am** has one line going the opposite direction because at some point, it tried to move the robot backwards. The updated states overlap the states from the database for both users, validating the proposed Kalman filter.

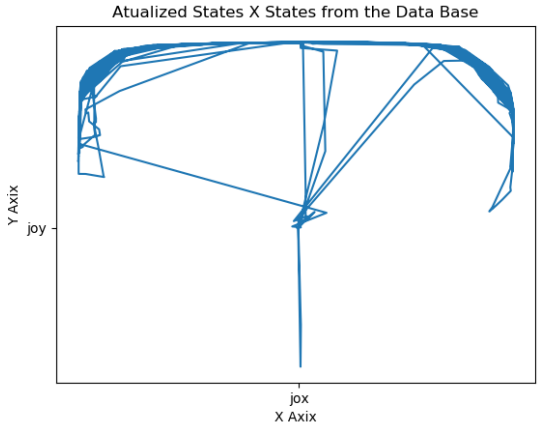

Figure 4.5: Comparison between the Updated states and the Output from the Data Base - eight-shaped route, user am.

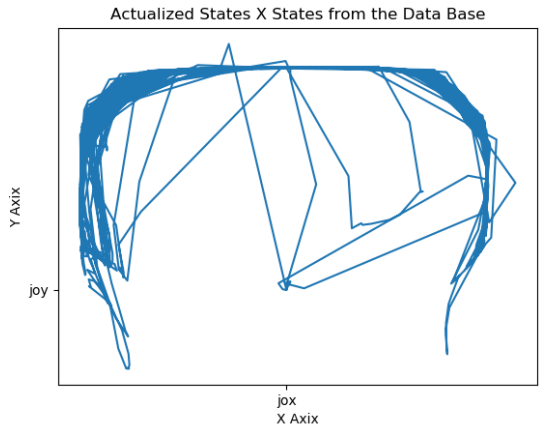

Figure 4.6: Comparison between the actualized states and the Output from the Data Base - eight-shaped route, user gm.

4.1.1.2 Building route

The batch of results form user **am** is showed in the Figures 4.7, 4.8 and 4.11. The results from user **gm** are shown in Figures 4.9, 4.10 and 4.12.

The Figures 4.7 and 4.8 show the updated states in X and Y, respectively, for user **am**. The Figures 4.9 and 4.10 show the updated states for user **gm**. Analysing the graphs, one can see that the pattern follows again the expected behaviour defined in Section 3.4. The states in X vary between one and minus one, but most of the time, it is smaller than $0,5$, therefore considered to be close to zero. It has those variations because the trajectory is quite long, and the user needs to correct it sometimes before the robot arrives at the next corner. The states in Y are at one most of the time, confirming that the robot was moving forward most of the time.

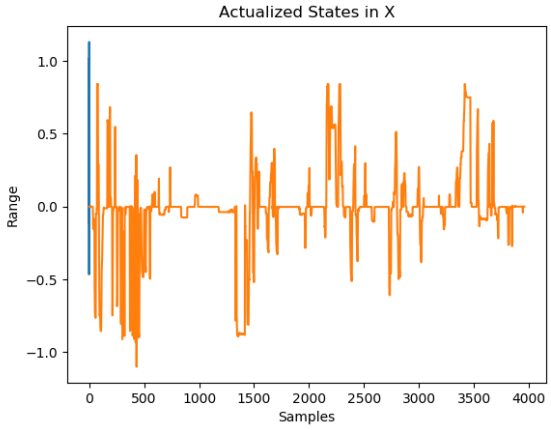

Figure 4.7: Updated states in X - building route, user **am**.

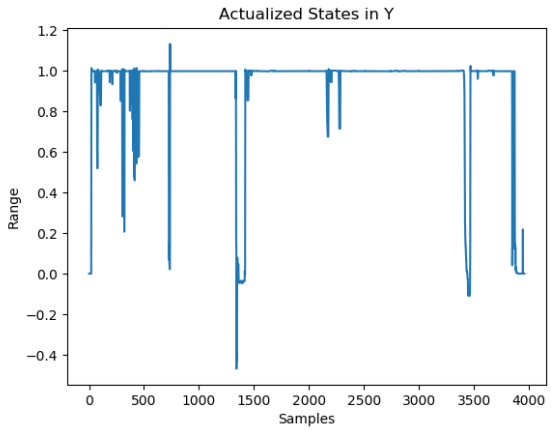

Figure 4.8: Updated states in Y - building route, user **am**.

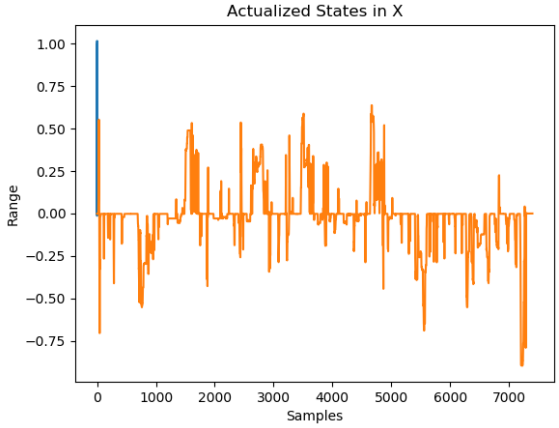

Figure 4.9: Actualized states in X - building route, user **gm**.

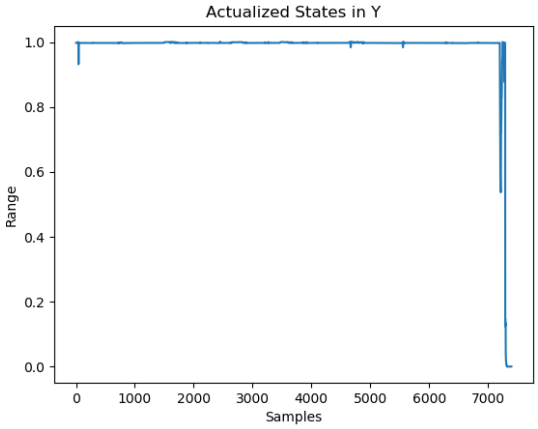

Figure 4.10: Actualized states in Y - building route, user **gm**.

The last two graphs, shown in Figures 4.11 and 4.12, show the combination of the updated

states in X and Y for each user. As explained in 4.1.1.1, this representation is expected to have a shape similar to the upper part of an umbrella since it is the movement the joystick is supposed to make to generate the input signal. The updated states overlap the states from the database for both users, validating the proposed Kalman filter.

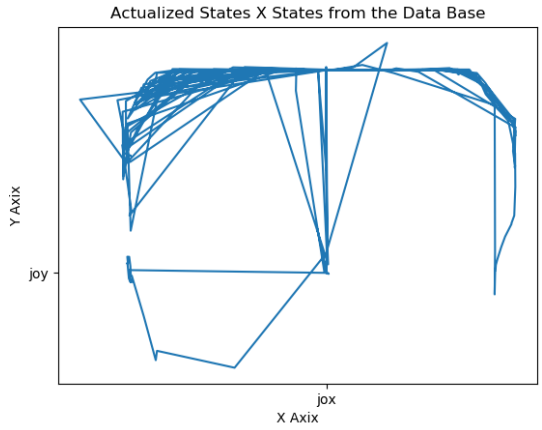

Figure 4.11: Comparison between the actualized states and the Output from the Data Base - building route, user **am**.

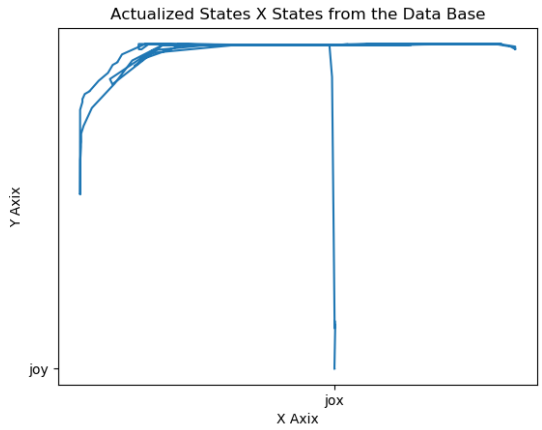

Figure 4.12: Comparison between the actualized states and the Output from the Data Base - building route, user **gm**.

4.1.2 Percentage mean error analysis

The graphical analysis done previously is a good starting point, but one can go further and analyse the results analytically. The following tables show the results of the mean error for each trajectory analysed, that is, the building and the eight shape. All signal fusion methods used by Thúlio in his work [2] were compared to the Kalman filter:

- **Network with 0.05 threshold:** if the difference between the user input and the network

input is bellow 5%, then the user input is used directly as the output. If it is above 5%, the network input is used as output.

- **Network with 0.20 threshold:** if the difference between the user input and the network input is bellow 20%, then the user input is used directly as the output. If it is above 20%, the network input is used as output.
- **Mean with 0.05 threshold:** if the difference between the user input and the average between the user input and the network input is bellow 5%, then the user input is used directly as the output. If it is above 5%, the average is used as output.
- **Mean with 0.20 threshold:** if the difference between the user input and the average between the user input and the network input is bellow 5%, then the user input is used directly as the output. If it is above 5%, the average is used as output.
- **Off:** the user input is used as output directly.

Table 4.1: Average error for the building trajectory

Method	**Absolute mean error**		**Percentage mean error**	
	X	**Y**	**X**	**Y**
Network with 0.05 threshold	0.00668026	-0.02550249	2.0655%	2.5413%
Network with 0.20 threshold	0.00226939	-0.02194258	1.1969%	2.1856%
Mean with 0.05 threshold	0.00098402	-0.00733841	0.2849%	0.7331%
Mean with 0.2 threshold	0.00122919	0.00672693	0.2454%	0.6727%
Off	0.00000267	0.00000804	0.008639%	0.002831%

Table 4.2: Average error for the eight trajectory

Method	**Absolute mean error**		**Percentage mean error**	
	X	**Y**	**X**	**Y**
Network with 0.05 threshold	0.00283381	-0.06303956	0.9058%	6.3495%
Network with 0.20 threshold	-0.01439740	-0.03264266	1.8123%	3.2643%
Mean with 0.05 threshold	-0.00764797	-0.02243746	0.7647%	2.2437%
Mean with 0.2 threshold	-0.00435120	0.01002543	0.4351%	1.3816%
Off	0.00002455	0.00009597	0.002455%	0.009597%

Analysing the results shown in the tables above, one can see clearly that the percentile mean error is acceptable for all cases, staying below under 5% and in most cases staying under 1%. The only case where the error was above 5% is in the first one in Table 4.2, and even in that case, it was close enough to be considered acceptable.

Another detail that one should notice in the tables is the accuracy the filter has when the test type is with both the mean algorithm and the net off. In those, the error is considerably lower. The main explanation is that the whole system has more error when the other test types are executed.

Table 4.3: Average error both trajectories

Method	Absolute mean error		Percentage mean error	
	X	**Y**	**X**	**Y**
Network with 0.05 threshold	0.00465581	-0.04525884	1.4552%	4.5456%
Network with 0.20 threshold	-0.00513807	-0.02669817	1.4705%	2.6650%
Mean with 0.05 threshold	-0.00353466	-0.01495741	0.5339%	1.4954%
Mean with 0.2 threshold	-0.00130788	0.00633079	0.3317%	0.9677%
Off	0.00001361	0.00005201	0.005547%	0.006214%

Having analysed the tables shown and using 5% as the limit of acceptable error, the values of the actualised state vector (used as the output of the filter and the comparison value for this analysis), the filter is validated and can be used as-is as the project goes on.

4.2 Simulated tests

The results of the tests will be analysed in this Section. For this analysis, there are two essential sources of information: the users' perception, which will compose the qualitative validation of the system, and the trajectories recorded from the tests, which will compose the quantitative validation. It is essential to consider, also, that the tests were carried out in two phases: in the first phase, the users used a joystick as the input method, and in the second phase, the users used a keyboard as the input method. This difference in the input methods must be considered when analysing the results for both the qualitative and quantitative validations.

In total, 32 tests were done to validate the system: 13 were in person, and 19 were done remotely, using the keyboard as the input method. Also, three users participated in both phases, and all three of them reported that it was significantly more challenging to control the robot with the keyboard than with the joystick.

The distribution of the tests can be seen in Table 4.4. Since the users tested the methods in different orders, it can be assumed that this factor did not favour one method or another. Both the qualitative analysis of users' preference and the quantitative analysis of the trajectories can be done considering that all methods were tested in the same conditions. Only the first order (A - Network, mean and Kalman filter) was done two times more than the others, and it happened solely because of the number of users.

4.2.1 User perceptions - Qualitative validation

The qualitative validation was made by analysing the responses of the perception questionnaire, presented in Section 3.9. The first thing one should look for in this analysis is which method most users preferred and any difference between the two different phases. This can be seen in Table 4.5 and its graphic representation in Figure 4.13.

Table 4.4: Tests performed, classified by the order of the methods used.

Order		**All tests**		**Phase 1 (joystick)**		**Phase 2 (keyboard)**	
A	Network, mean and Kalman filter	7	21.88%	3	9.38%	4	12.50%
B	Network, Kalman filter and mean	5	15.63%	2	6.25%	3	9.38%
C	Mean, network and Kalman filter	5	15.63%	2	6.25%	3	9.38%
D	Mean, Kalman filter and network	5	15.63%	2	6.25%	3	9.38%
E	Kalman filter, network and mean	5	15.63%	2	6.25%	3	9.38%
F	Kalman filter, mean and network	5	15.63%	2	6.25%	3	9.38%
Total		**32 users**	**100%**	**13 users**	**41%**	**19 users**	**59%**

Table 4.5: User preference analysis, considering both phases.

Test	**All tests**		**Phase 1 (joystick)**		**Phase 2 (keyboard)**	
Red (Network with 0.05 threshold)	8	25.00%	3	9.38%	5	15.63%
Yellow (Mean with 0.05 threshold)	12	37.50%	5	15.63%	7	21.88%
Green (Kalman Filter)	12	37.50%	5	15.63%	7	21.88%
TOTAL	**32 users**	**100%**	**13 users**	**41%**	**19 users**	**59%**

The first and most straightforward conclusion that one can draw from those values is that the method red (Network with 0.05 threshold) was the least liked by the users, both in remote and in-person tests. The main explanation for that perception is that it is the most rigid method, so it gives less freedom of choice during the test to the user, and most people do not feel comfortable with that. The second conclusion one can draw is that regarding the comfort during the tests, both the yellow and the green methods had the same number of users preferring them. That said, it is not possible to determine which method was best with just this set of data.

Table 4.6: User preference analysis, considering only the order of the tests and not each method (for both keyboard and joystick).

Test order	**All tests**		**Phase 1 (joystick)**		**Phase 2 (keyboard)**	
First	8	25.00%	1	3.13%	7	21.88%
Second	9	28.13%	5	15.63%	4	12.50%
Third	15	46.88%	7	21.88%	8	25.00%
TOTAL	**32 users**	**100%**	**13 users**	**41%**	**19 users**	**59%**

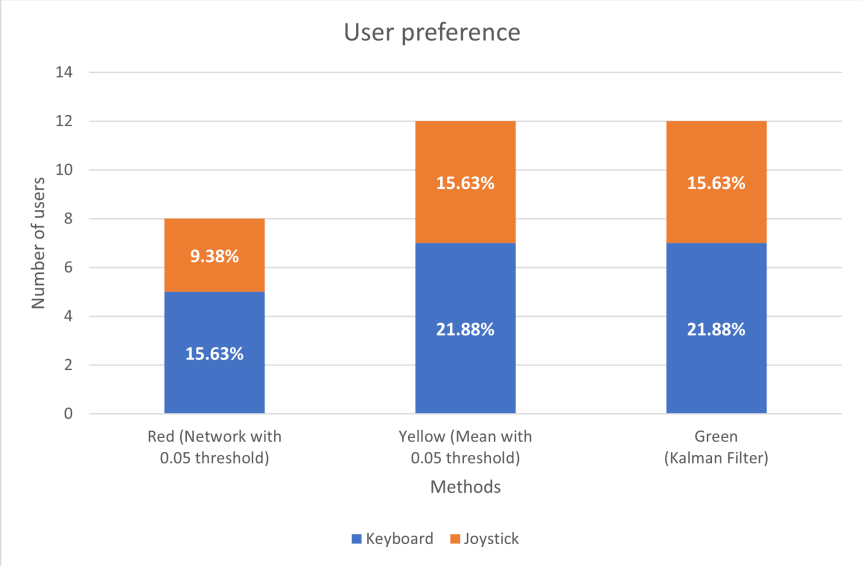

Figure 4.13: Analysis of user preference, considering both joystick (in person) and keyboard (remote) tests.

The next thing that must be analysed from the questionnaire, considering user preference solely, is if there is any relation between the order the users did the tests and the test they chose as most comfortable. This information is shown in Table 4.6 and in Figure 4.14, its graphic representation.

Analysing the graph shown in Figure 4.14, one can easily see that most users (46.88% of them) preferred the last method they tried. That was expected since the users tend to feel more comfortable with the whole system after using it longer. Also, it implies that user perception, although very important, cannot be the only parameter analysed to decide which method is better.

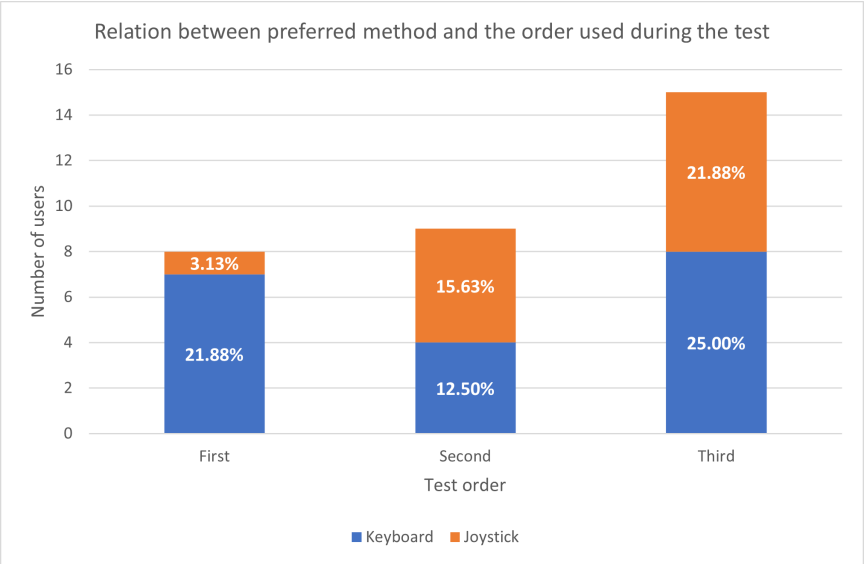

Figure 4.14: Analysis of user preference, considering only the order of the tests and not each method (for both keyboard and joystick).

Going forward with the analysis of the data obtained with the questionnaire, one can start the socioeconomic analysis of respondents.

Table 4.7: Gender of the users.

Gender	**All tests**		**Phase 1 (joystick)**		**Phase 2 (keyboard)**	
Female	16	50.00%	8	25.00%	8	25.00%
Male	16	50.00%	5	15.63%	11	34.38%
I prefer not to declare	0	0.00%	0	0.00%	0	0.00%
TOTAL	**32 users**	**100%**	**13 users**	**41%**	**19 users**	**59%**

The first data to be analysed is the user's gender. From Table 4.7, one can see that considering both phases, 50% of the users declared themselves to be female, and 50% declared to be male. None of the users preferred not to declare their gender. This is a balanced distribution, even though the user's gender is not assumed to interfere with the results and will not be considered in the quantitative analysis.

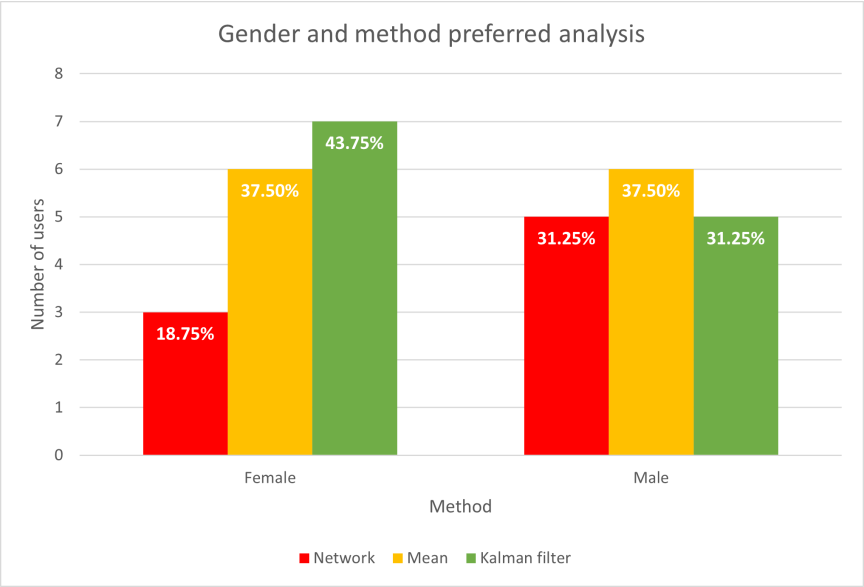

Figure 4.15: Analysis of the relationship between gender and preferred method by the user.

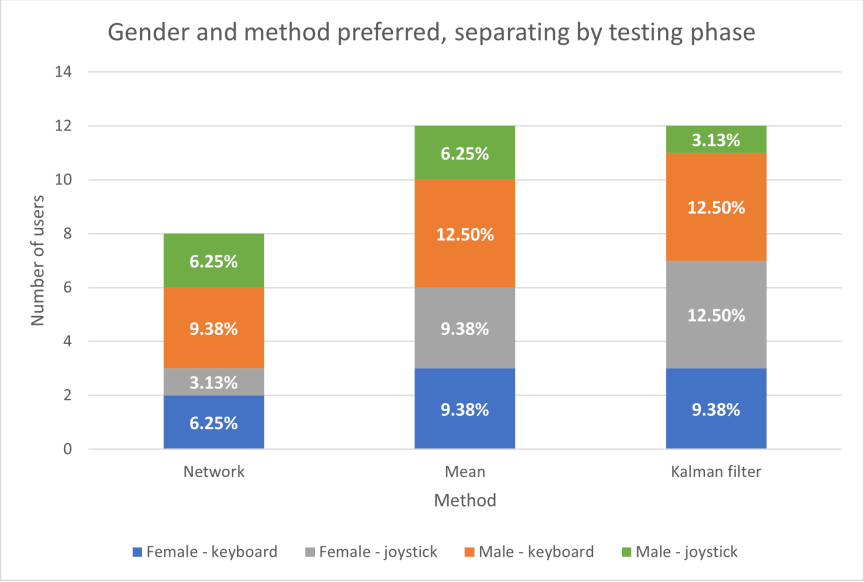

Figure 4.16: Analysis of the relationship between gender and preferred method by the user, considering which phase the test was performed.

Analysing Figures 4.15 and 4.16, one can see that more female users preferred the Kalman filter method than the other two methods, while male users preferred the mean method. Also, one can see that the least liked method amongst women was the network one.

The following analysis to be made is regarding the age and education level of the users. From the data shown in Tables 4.8 and 4.9, one can see that most users are still in university or have completed a higher education degree. That resonates well with the age of the users since most of them fall in one of the first three age groups, being up to 30 years old.

Table 4.8: User's age.

Age	**All tests**		**Phase 1 (joystick)**		**Phase 2 (keyboard)**	
Less than 20	4	12.50%	3	9.38%	1	3.13%
21 to 25	17	53.13%	4	12.50%	13	40.63%
26 to 30	4	12.50%	2	6.25%	2	6.25%
31 to 35	0	0.00%	0	0.00%	0	0.00%
36 to 40	1	3.13%	0	0.00%	1	3.13%
41 to 45	0	0.00%	0	0.00%	0	0.00%
46 to 50	4	12.50%	3	9.38%	1	3.13%
51 to 56	2	6.25%	1	3.13%	1	3.13%
56 to 60	0	0.00%	0	0.00%	0	0.00%
More than 60	0	0.00%	0	0.00%	0	0.00%
TOTAL	**32**	**100%**	**13**	**41%**	**19**	**59%**

Table 4.9: Level of education.

Education level	**All tests**		**Phase 1 (joystick)**		**Phase 2 (keyboard)**	
Incomplete elementary school	0	0.00%	0	0.00%	0	0.00%
Complete elementary school	0	0.00%	0	0.00%	0	0.00%
Incomplete high school	1	3.13%	1	3.13%	0	0.00%
Complete high school	0	0.00%	0	0.00%	0	0.00%
Incomplete higher education	18	56.25%	5	15.63%	13	40.63%
Complete higher education	5	15.63%	1	3.13%	4	12.50%
MBA, postgraduate or specialization	5	15.63%	4	12.50%	1	3.13%
Master	1	3.13%	1	3.13%	0	0.00%
PhD	2	6.25%	1	3.13%	1	3.13%
TOTAL	**32**	**100%**	**13**	**41%**	**19**	**59%**

The following information to be analysed is the users experience with operating robots or machinery of any kind. The responses from this question are summarised in Table 4.10 and one can see that most users do not have experience with robots or machinery. That is a good indicator because most of them fall in the desired target audience of inexperienced users.

Table 4.10: Experience with operating robots or machinery.

Answer	**All tests**		**Phase 1 (joystick)**		**Phase 2 (keyboard)**	
Yes	2	6.25%	0	0.00%	2	6.25%
No	30	93.75%	13	40.63%	17	53.13%
TOTAL	**32**	**100%**	**13**	**41%**	**19**	**59%**

The last information to be evaluated is familiarity with video games and skills with the input method, whose data can be seen in Tables 4.11 and 4.12, respectively. Form Table 4.11, one can

see that almost all users (84.88%) declared to have some familiarity with video games, most of them (46.88%) having said to have "a lot". Also, in Table 4.12, the pattern is repeated. Most of the users (78..13%) declared to have skills to some degree with the used input method (keyboard or joystick), most of them (46.88%) saying to have "a lot".

Table 4.11: How familiar the users are with video games.

Answer	**All tests**		**Phase 1 (joystick)**		**Phase 2 (keyboard)**	
I am not	5	15.63%	3	9.38%	2	6.25%
A little	12	37.50%	5	15.63%	7	21.88%
A lot	15	46.88%	5	15.63%	10	31.25%
TOTAL	**32**	**100%**	**13**	**41%**	**19**	**59%**

Table 4.12: How skilled the users are with the input method (keyboard or joystick) for games.

Answer	**All tests**		**Phase 1 (joystick)**		**Phase 2 (keyboard)**	
I am not	7	21.88%	5	15.63%	2	6.25%
A little	10	31.25%	3	9.38%	7	21.88%
A lot	15	46.88%	5	15.63%	10	31.25%
TOTAL	**32**	**100%**	**13**	**41%**	**19**	**59%**

Last but not least, for the remote test is vital to analyse how many users were disturbed by the internet connection. In Table 4.13, one can see that most of the users (68.42%) did not have any problems.

Table 4.13: Internet interference with the execution of the tests.

Answer	**Phase 2 (keyboard)**	
Yes	5	26.32%
No	13	68.42%
Could not tell	1	5.26%

4.2.1.1 Red method - Network with 0.05 threshold

Starting the analysis of the perception of users about each specific method, one can see in Table 4.14 that most users (65.63%) felt that the red method had little or very little interference in the robot's piloting, even though this was the method that most interfered in the trajectory, as it will be seen in Subsection 4.2.2. The data shown in Table 4.16 show that although most users feel this method has little or very little interference, most of them (56.25%) answered that the interference is adequate.

When analysing the data shown in Table 4.15, one can see that most users felt that the comfort was moderate in the red method. However, as will be seen in Subsection 4.2.2, this method was the one most of the users faced difficulties while driving, even some of them could not finish the

Table 4.14: Users perception regarding the interference of the red method (network with 0.05 threshold).

Answer	**All tests**		**Phase 1 (joystick)**		**Phase 2 (keyboard)**	
Very little interference	13	40.63%	4	12.50%	9	28.13%
Little interference	8	25.00%	2	6.25%	6	18.75%
Moderate interference	8	25.00%	4	12.50%	4	12.50%
Too much interference	3	9.38%	3	9.38%	0	0.00%
TOTAL	**32**	**100%**	**13**	**41%**	**19**	**59%**

intended trajectory on the first try because they would hit one of the traffic cones or go too far from the route and had to start over: this only happened in this method.

Table 4.15: Users perception of comfort while driving with the red method (network with 0.05 threshold).

Answer	**All tests**		**Phase 1 (joystick)**		**Phase 2 (keyboard)**	
Very comfortable	7	21.88%	4	12.50%	3	9.38%
Moderate	14	43.75%	6	18.75%	8	25.00%
Not comfortable	11	34.38%	3	9.38%	8	25.00%
TOTAL	**32**	**100%**	**13**	**41%**	**19**	**59%**

Table 4.16: Users perception of how the red method (network with 0.05 threshold) could be improved, regarding its interference.

Answer	**All tests**		**Phase 1 (joystick)**		**Phase 2 (keyboard)**	
Interfere more	4	12.50%	2	6.25%	2	6.25%
Interfere less	10	31.25%	6	18.75%	4	12.50%
The interference is adequate	18	56.25%	5	15.63%	13	40.63%
TOTAL	**32**	**100%**	**13**	**41%**	**19**	**59%**

4.2.1.2 Yellow method - Mean with 0.05 threshold

The same analysis done for the red method in 4.2.1.1 can be done for the yellow method as well. Starting with the perception of interference, one can see from Table 4.17 that most users (56.26%) found that this method had little or very little interference in the driving. However, contradicting that, the data in Table 4.19 shows that most users (56.25%) would prefer if the yellow method interfered less.

Analysing the comfort the users felt while driving the robot with the yellow method, one can see from Table 4.18 that less than one-fifth of the users felt this method was not comfortable to drive with, and 43.75% of them felt it was very comfortable. This was the method that more users pointed out to be very comfortable to drive with. Considering that most users that liked

Table 4.17: Users perception regarding the interference of the yellow method (mean with 0.05 threshold).

Answer	**All tests**		**Phase 1 (joystick)**		**Phase 2 (keyboard)**	
Very little interference	7	21.88%	3	9.38%	4	12.50%
Little interference	11	34.38%	4	12.50%	7	21.88%
Moderate interference	12	37.50%	5	15.63%	7	21.88%
Too much interference	2	6.25%	1	3.13%	1	3.13%
TOTAL	**32**	**100%**	**13**	**41%**	**19**	**59%**

this method said they felt the robot responded faster to the inputs made, both in the remote and the in-person tests, it makes sense that only a few users would feel uncomfortable using it.

Table 4.18: Users perception of comfort while driving with the yellow method (mean with 0.05 threshold).

Answer	**All tests**		**Phase 1 (joystick)**		**Phase 2 (keyboard)**	
Very comfortable	14	43.75%	5	15.63%	9	28.13%
Moderate	12	37.50%	6	18.75%	6	18.75%
Not comfortable	6	18.75%	2	6.25%	4	12.50%
TOTAL	**32**	**100%**	**13**	**41%**	**19**	**59%**

Table 4.19: Users perception of how the yellow method (mean with 0.05 threshold) could be improved, regarding its interference.

Answer	**All tests**		**Phase 1 (joystick)**		**Phase 2 (keyboard)**	
Interfere more	8	25.00%	3	9.38%	5	15.63%
Interfere less	18	56.25%	7	21.88%	11	34.38%
The interference is adequate	6	18.75%	3	9.38%	3	9.38%
TOTAL	**32**	**100%**	**13**	**41%**	**19**	**59%**

4.2.1.3 Green method - Kalman filter

Lastly, one can analyse the users' perception of the Kalman filter method. This was the one in which more users felt the interference during the trajectory, as can be seen in Table 4.20 that 43.88% of them felt the method had moderate or too much interference. Corroborating with that information, in Table 4.22 one can see that 50% of the users felt that this method should interfere less.

However, 40.63% of users said that this method was very comfortable to drive with, and less than 30% said it was uncomfortable. This points out that even if the user feels that the method is interfering, it does not mean that it is bad. In fact, 7 users (1 from the first phase and 6 from the second phase) even pointed out during the tests that the method was indeed interfering with

Table 4.20: Users perception regarding the interference of the green method (Kalman filter).

Answer	**All tests**		**Phase 1 (joystick)**		**Phase 2 (keyboard)**	
Very little interference	3	9.38%	3	9.38%	0	0.00%
Little interference	14	43.75%	6	18.75%	8	25.00%
Moderate interference	9	28.13%	2	6.25%	7	21.88%
Too much interference	6	18.75%	2	6.25%	4	12.50%
TOTAL	**32**	**100%**	**13**	**41%**	**19**	**59%**

the driving and felt that they drove better because of it.

Table 4.21: Users perception of comfort while driving with the green method (Kalman filter).

Answer	**All tests**		**Phase 1 (joystick)**		**Phase 2 (keyboard)**	
Very comfortable	13	40.63%	5	15.63%	8	25.00%
Moderate	10	31.25%	5	15.63%	5	15.63%
Not comfortable	9	28.13%	3	9.38%	6	18.75%
TOTAL	**32**	**100%**	**13**	**41%**	**19**	**59%**

Table 4.22: Users perception of how the green method (Kalman filter) could be improved, regarding its interference.

Answer	**All tests**		**Phase 1 (joystick)**		**Phase 2 (keyboard)**	
Interfere more	8	25.00%	1	3.13%	7	21.88%
Interfere less	16	50.00%	7	21.88%	9	28.13%
The interference is adequate	8	25.00%	5	15.63%	3	9.38%
TOTAL	**32**	**100%**	**13**	**41%**	**19**	**59%**

4.2.2 Trajectories - Quantitative validation

We will now start the quantitative analysis made with the data saved from the simulations made in ROS and Gazebo. From this data, it was possible to create graphs showing the trajectories made by users, as well as to analyse the second derivatives from said trajectories to define how smooth the driving was. Also, it is possible to analyse the average differences between user input and network input for each method and the difference between what the users intended as input and what they got as output from the system to the robot. Therefore, the following topics will deal with such analyses.

The first and most important result obtained with this analysis was that the Kalman filter did indeed give the smoothest trajectories, even if the qualitative analysis made in Subsection 4.2.1 did not show that explicitly. This deduction is due to the low value of the second derivative of the position found for the Kalman filter. Also, the red method (Network with 0.05 threshold) was, in general, the one with the most abrupt changes in the movement when considering the in-person

tests, as will be shown in 4.2.2.2.

4.2.2.1 Analysis of graphical representations of trajectories

The analysis of the graphical representation of the curves will be done for a couple of users only, since the overall analysis of the data done later on can give one more conclusive information about the tests as a whole, and the main objective will be to show the differences between the tests done using the joystick as input method (in person) and the ones with the keyboard (remote). It is essential to make this comparison because the neural network used was trained to work with an input signal from a joystick or at least with a similar profile, and the profile of the input signal generated by the keyboard, as explained earlier, is quite different. Also, as one will be able to see from the following graphs, the users had a significant increase in difficulty while using the keyboard.

The first thing to do is define how the trajectory graphs are supposed to be understood. All four trajectories from user 10, one for each method tested, can be seen in Figure 4.17. The X-axis represents the trajectory that was supposed to be made, and the Y-axis represents how far from the centre of the trajectory the user went. The beginning of each test always occurred on the left side of the graphs, with the robot's position around x=-10, and the centres of the oval shapes one can see were the traffic cones that indicated the desired path, as can be seen in Figure 3.5. Although one can see how each trajectory was in this figure, it is not clear the comparison between them, and one cannot tell which one was smoother. To solve that, all four curves will be plotted in the same graph, as shown in Figure 4.18.

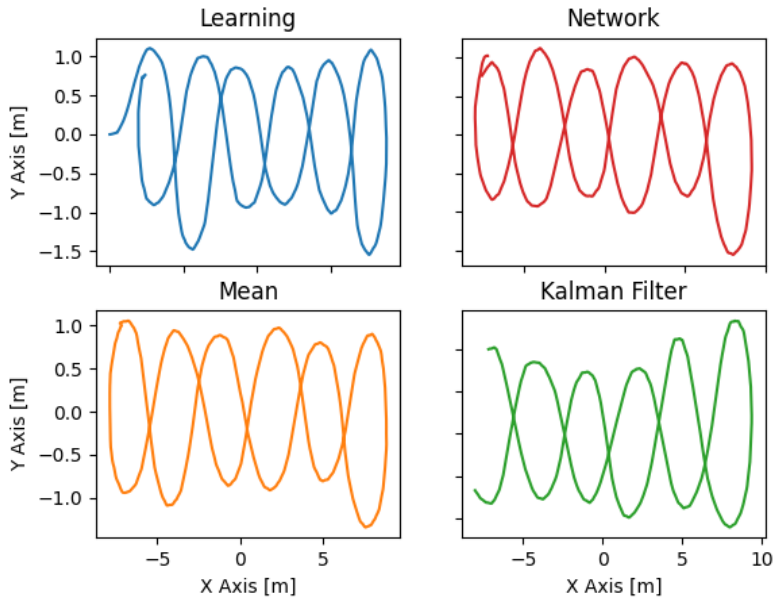

Figure 4.17: Trajectories done by user 10 using the joystick as input method, separated by method used.

One can now begin to compare the trajectories done by different users. Both Figures 4.18 and 4.19 show trajectories done by user using the joystick as input method. User 7, shown in Figure 4.19, found the tests harder and even had to restart the tests twice because it lost control of the robot and went out of the path designated to the simulations. Although the graph shows only the successful attempts, one can see that the driving was not as smooth as the one done by user 10 in Figure 4.18 and was not as balanced either, having big differences in the trajectories done in each round.

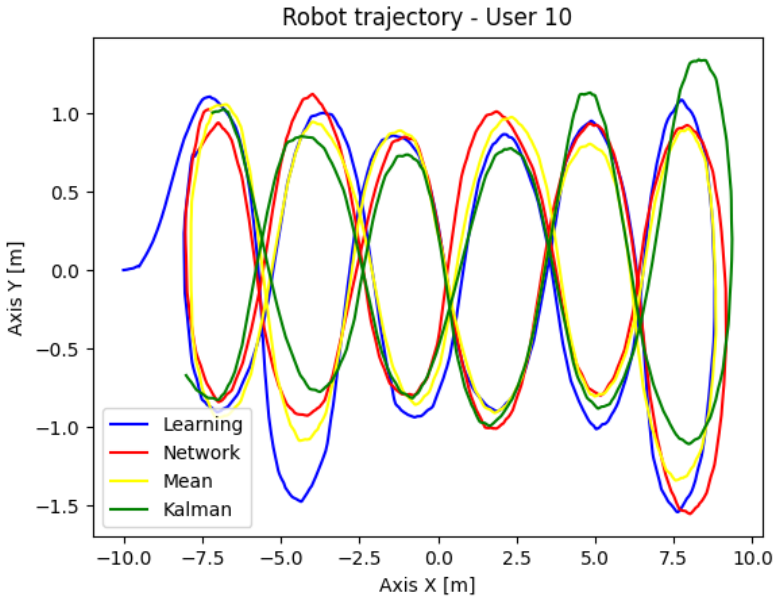

Figure 4.18: Trajectories done by user 10 (female, between 21 and 25 years old, with no familiarity with robots, very familiar with video games and very skilled with joystick) using the joystick as input method, with the four trajectories represented together.

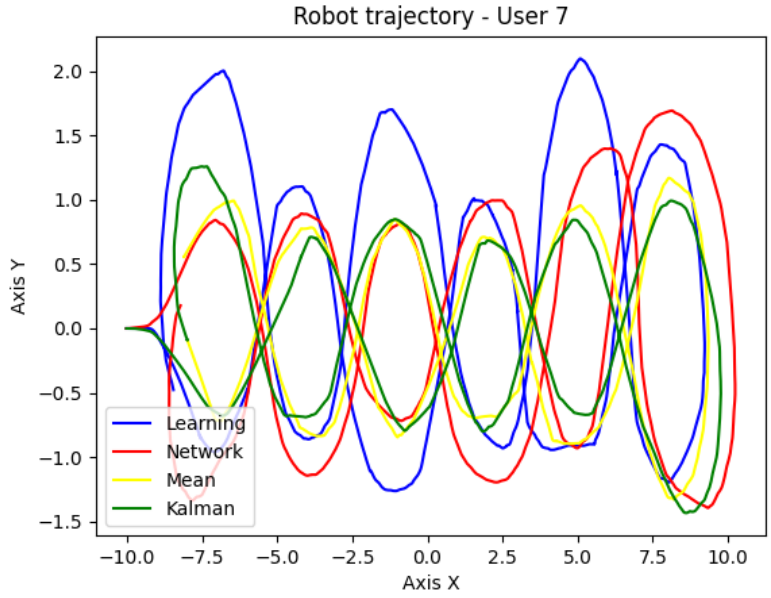

Figure 4.19: Trajectories done by user 7 (female, between 26 and 30 years old, with no familiarity with robots, not familiar with video games and not skilled with joystick) using the joystick as input method.

The following comparison to be made is between the trajectories done using the joystick and the ones using the keyboard, shown in Figures 4.20 and 4.21. User 28 was the one that obtained the smoothest curves and most well-rounded trajectories among the ones that used the keyboard. In its graph, one can see that although this user had a good result with all four rounds, the one was done using the Kalman filter was smoother and more balanced throughout the whole path. However, when asked which method was the most comfortable one to drive the robot with, this user said the red (network with 0.05 threshold) one was better.

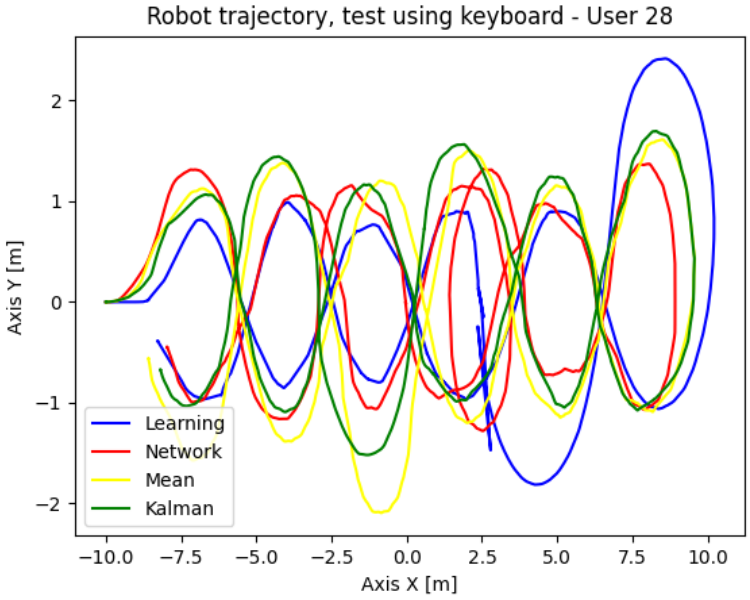

Figure 4.20: Trajectories done by user 28 (male, between 21 and 25 years old, with no familiarity with robots, very familiar with video games and not skilled with the keyboard) using the keyboard as input method.

In contrast, user 18 obtained the worst set of trajectories. This user had great difficulties using the keyboard as the input method and, as one can see from Figure 4.21, could not go around all cones properly. Moreover, even in that extreme case, the Kalman filter method was the smoothest, preventing any harsh curves like the ones seen on the learning round from happening. That is, it accomplished its objective nicely.

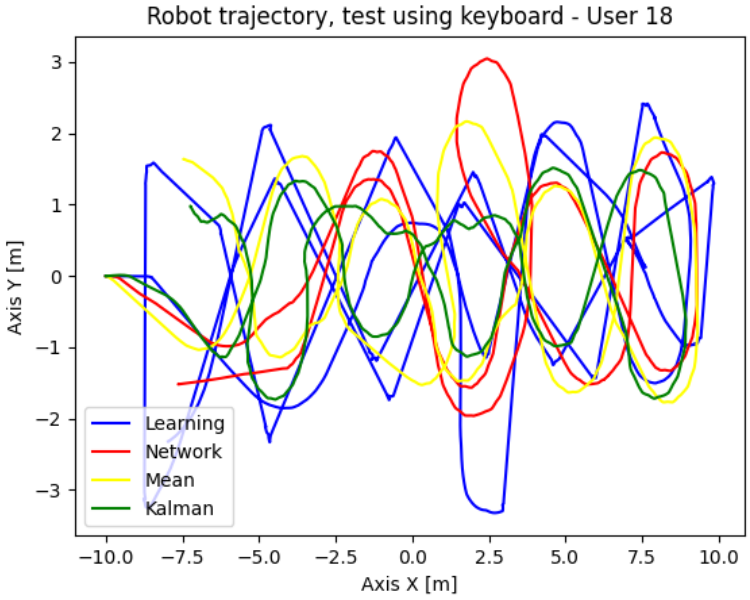

Figure 4.21: Trajectories done by user 18 (male, between 26 and 30 years old, with no familiarity with robots, very familiar with video games and very skilled with the keyboard) using the keyboard as input method.

4.2.2.2 Overall analysis

The first analysis will be of the second-order variation analysis, shown in Tables 4.23 and 4.24. This data represents the average of the second derivative of the robot's position in each test made. The analysis here consists of finding the method that provided the smoother average trajectory, so the smaller the value of the second-order variation, the smoother the trajectory was since those values represent the average spikes in the positions of the robot.

The graphs shown in Figures 4.22 and 4.23 are the graphical representation of the data shown in Tables 4.23 and 4.24, respectively. The type of graph used is called boxplot because it allows the analysis of several important data characteristics in one plot [19]. The box represents half of the data, showing the distance between the first and the third quartiles. The line inside the box represents the median, and the lines outside the boxes represent the standard deviation. Finally, the dots outside the boxes represent the outlier values present in the dataset.

In both phases, the standard deviations are high for two main reasons:

- There were few users, that is, few sets of data to analyse, so a high standard deviation is to be expected;
- The users had different driving profiles, meaning that some of them preferred to perform the curves more open, some of them tried to stay closer to the centre of the route, and also the level of difficulty faced was different from one another. This caused several differences in the trajectories made.

Table 4.23: Second-order variation analysis, for all users - Remote tests (keyboard used as input method).

Direction		Learning		Network	
		Average	**Standard deviation**	**Average**	**Standard deviation**
Linear	**Forward**	48.2402	31.1142	20.9681	14.1806
	Backward	59.5282	27.6504	29.6090	19.6379
Angular	**Left**	43.9538	22.2744	18.7693	10.8822
	Right	50.2897	22.5679	23.7479	14.8770

Direction		Mean		Kalman filter	
		Average	**Standard deviation**	**Average**	**Standard deviation**
Linear	**Forward**	19.1200	11.8489	6.9395	3.7628
	Backward	26.1463	19.3831	8.6685	4.3352
Angular	**Left**	18.4636	12.6435	6.2961	3.1427
	Right	21.7727	14.9672	7.0006	3.4443

From Figure 4.22, one can see that the Kalman filter did provide smoother movements in all four directions, with its averages of the second-order variation of the position being considerably more petite than the other methods and with the smallest standard deviations. Also, one can see that the network and mean method presented similar values, meaning that when the keyboard was used as the input method, there was not much difference in smoothness between those methods.

Finally, the behaviour seen for the learning round was expected since most users had great difficulties driving with the keyboard, as can be seen from the example given in Figure 4.21. Also, the users tended to increase the velocity too fast by mistake and then abruptly stopped the robot to correct the path. Also, most users were not able to make smooth curves on their own, as can be seen in the example given in Figure 4.20, where even the best user was not able to make smooth curves without help.

This shows that even in an unfavourable situation, where the user is provided with an input method different from what the system was originally made to use and is unable to carry out the route in a satisfactory way, the aid methods were able to fulfil their objective of correcting the trajectory and providing smooth curves. This is especially true for the Kalman filter method, which guaranteed the smoothest curves among the three evaluated methods.

Figure 4.22: Average of the second order variation, per direction, for all users - Remote test (keyboard).

Now analysing the tests carried out in person, using the joystick as the input method, one can see from Figure 4.23 that the learning rounds averages are much closer to the red and yellow methods than in the remote tests. This was expected because the users were able to make smoother trajectories even without the help of the methods, as can be seen in the examples shown in Figures 4.18 and 4.19. Also, one can see that in this case, the Kalman filter performed even better, being even further from the other methods, presenting smaller averages when compared with the network and mean methods.

Lastly, one may notice that the overall values seen in the tests done with the joystick as input method were much higher than those seen in the keyboard tests. This can be explained by the users' driving profile in each method due to the behaviour imposed by the input method.

- The keyboard imposed a smaller and more constant velocity due to two main reasons: the user did not have to hold any key to maintain the velocity, so it was easier to use a constant velocity than not, and since it was hard to perform the tests, most users felt more comfortable driving slower.
- The joystick gave more freedom of movement since it was easier to control the robot, so the users felt more comfortable driving faster. Indeed, many users said that they would prefer if the robot went faster, comments that did not happen in the remote tests.

Table 4.24: Second order variation analysis, for all users - In person tests (joystick used as input method).

Direction		Learning		Network	
		Average	Standard deviation	Average	Standard deviation
Linear	**Forward**	239.0497	79.3228	233.4334	46.5733
	Backward	222.2069	82.1677	243.8541	57.4500
Angular	**Left**	194.5987	65.2137	197.6361	34.4480
	Right	195.5957	65.5269	208.2130	44.1867

Direction		Mean		Kalman filter	
		Average	Standard deviation	Average	Standard deviation
Linear	**Forward**	208.2290	83.8692	101.7654	34.2651
	Backward	216.9606	88.6800	93.5195	31.1897
Angular	**Left**	176.1674	68.1713	83.1563	28.0715
	Right	178.8295	70.2188	76.5865	24.8862

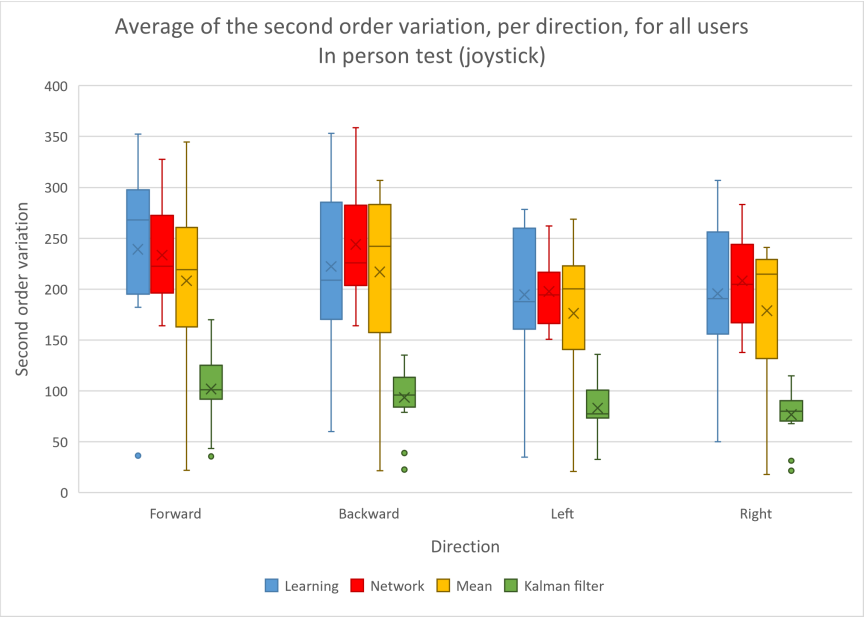

Figure 4.23: Average of the second order variation, per direction, for all users - In person test (joystick).

The next analysis is based on Tables 4.26 and 4.25, which show the differences between the user input, network input and system output to the robot. Here, network input stands for the output of the network that acts as input to the method tested.

From the data shown, one can see that in both test phases, the red method presented a more significant difference between the network input and the user input. This pattern repeats itself when one looks at the differences between the user input (that is, what the user wanted the robot to do) and the system output (that the system actually made the robot do). When looking at

Table 4.25: Percentile differences between the user input, network input and system output - In person tests (joystick used as input method).

		Red (Network)	Yellow (Mean)	Green (Kalman filter)
Network input vs. User input	**X**	7.6165%	2.9213%	3.0013%
	Y	9.2246%	3.0330%	3.4155%
User input vs. System output	**X**	5.5145%	1.4364%	3.0006%
	Y	7.6236%	1.6383%	3.4153%

the keyboard tests, shown in Table 4.26, this difference can go up to 26.46%. Comparing the green and yellow methods, one can see that the values are much smaller and closer to each other, especially in the in-person tests. This explains why most users felt more comfortable and preferred those two methods over the red since they did have more control over what the robot was doing and did not have to "fight" the system as much to impose their own will.

Table 4.26: Percentile differences between the user input, network input and system output - Remote tests (keyboard used as input method).

		Red (Network)	Yellow (Mean)	Green (Kalman filter)
Network input vs. User input	**X**	26.2873%	10.1288%	10.0613%
	Y	26.4592%	8.5214%	8.4914%
User input vs. System output	**X**	24.3907%	9.0385%	10.0619%
	Y	25.8554%	7.8768%	8.4935%

The higher values seen in Table 4.26 were expected because of the input method used. Since the system was originally made to work with an input signal coming from the Xbox controller, as explained in 3.7, some opposition from the system to the movement should be expected, especially because the driving profile of the users also changed a lot form one input method to the other, as one can see comparing the Figures 4.18, 4.19, 4.20 and 4.21.

Last but not least, one can analyse the overall average interference of the neural network in each method tests in both test phases, shown in Table 4.27. This percentage shows the fraction of the movement decisions that the neural network acted on. That said, the Kalman filter method is expected to have values close to 100%, since this method does not have a threshold. However, for the neural network and mean methods, the values can bring some insights into the proposed system's performance. As expected, in both phases, the red method had a significantly higher percentage of network interference when compared with the yellow method. That means that the user more freedom when driving with the yellow method, justifying once again the fact that few users preferred the red method, as shown in Subsection 4.2.1. Comparing the tests using the joystick with the ones using the keyboard, the data are shown in Table 4.27 also supports the conclusion that in the remote tests, the system had a different behaviour, interfering even more in the movement and giving less freedom to the user.

Table 4.27: Analysis of the average percentage of network interference in each of the methods.

Test phase	**Red (Network)**	**Yellow (mean)**	**Green (Kalman filter)**
In person (joystick)	23.02%	8.42%	99.82%
Remote (keyboard)	81.23%	61.24%	98.12%

Chapter 5

Conclusions

A problem found in the robotics field is the high cost of training for operators of teleoperated robots. One way to reduce these costs is to implement a system that gives some autonomy to the robots so that even an inexperienced operator can guide them without significant difficulties. This work consisted of proposing and testing a signal fusion method that can be used to implement such an autonomy system.

The main objectives were:

1. Propose a Kalman filter algorithm to fuse the two input signals;
2. Validate the filter in simulations using the database;
3. Validate the filter in the robot, quantitatively, testing with several users;
4. Validate the filter qualitatively, analysing the perception of those users.

The modelling of the system and the proposed algorithm for the Kalman filter can be seen in the Methodology chapter (Chapter 3), and it was validated in four steps. The first one was done with the preexisting database, with the results shown in Section 4.1. All the criteria defined for this validation in the Sections 3.2 and 3.4 were matched, having an average error for both trajectories testes (building and eight) smaller than 5%, so the system passed the first stage validation.

Following that, the system was validated with different users via simulation, this time divided into two phases: one executed in person, using an Xbox controller as the input method, and another one carried out remotely, using a keyboard as the input method. Although the user perception analysis on its own was inconclusive since 37.50% of the users preferred the Kalman filter and 37.50% preferred the mean method, when the quantitative analysis is made, there is no doubt that the Kalman filter is a superior method, providing smoother trajectories overall. This results can be seen in Sections 4.2.1 and 4.2.2, respectively.

5.1 Future perspectives

To improve the results obtained with the Kalman filter, some modification in its implementation could be made. One of them is upgrading the estimation of the process noise using a denoising autoencoder (an artificial neural network able to estimate the original data) [20]. Variations in the input matrices (B_1 and B_2) could be made, distributing the weight of the input signals differently.

Also, testing the system with more users, preferably using both input methods, would allow a more in-depth statistical analysis, mainly in users' perception. Furthermore, different tests could be performed, such as using a fixed velocity and allowing the user to change direction only.

REFERENCES

[1] SICILIANO, B. et al. *Robotics: modelling, planning and control*. [S.l.]: Springer Science & Business Media, 2010.

[2] SANTOS, T. N. S. Aprendizado automático utilizando um modelo lstm aplicado como auxiliar no controle de orientação e velocidade de robô móvel. v. 0, p. 0–84, jul. 2019.

[3] THRUN, S.; BURGARD, W.; FOX, D. *Probabilistic robotics*. [S.l.]: MIT press, 2005.

[4] GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep learning*. [S.l.]: MIT press, 2016.

[5] MURPHY, K. P. *Machine learning: a probabilistic perspective*. [S.l.]: MIT press, 2012.

[6] SANTOS, T. N. S. et al. Driving mobile robots using a deep lstm architecture: An experimental approach. *IEEE Latin America Transactions*, v. 100, n. 1e, 2020.

[7] MYTTENAERE, A. D. et al. Mean absolute percentage error for regression models. *Neurocomputing*, Elsevier, v. 192, p. 38–48, 2016.

[8] DATTALO, A. *ROS Introduction*. maio 2021. `http://wiki.ros.org/ROS/Introduction`.

[9] FOUNDATION, O. S. R. *Gazebo - Robot simulation made easy*. maio 2021. `http://gazebosim.org/`.

[10] CHUI, C. K.; CHEN, G. et al. *Kalman filtering*. [S.l.]: Springer, 1999.

[11] ROS. *Pioneer 3-AT*. maio 2021. `https://robots.ros.org/pioneer-3-at/`.

[12] MICHEL, O. *Adept MobileRobots Pioneer and Pioneer-compatible platforms*. maio 2021. `http://wiki.ros.org/Robots/AMR_Pioneer_Compatible`.

[13] ARAUJO, G. F. P. *P3at tutorial package*. maio 2021. `https://github.com/Gastd/p3at_tutorial`.

[14] OGATA, K. *Discrete-time control systems*. [S.l.]: Prentice-Hall, Inc., 1995.

[15] OGATA, K. *Modern control engineering*. [S.l.]: Prentice hall, 2010.

[16] TOCCI, R. J.; WIDMER, N. S.; MOSS, G. *Sistemas Digitais, princípios e aplicações. 10a edição*. [S.l.]: Editora Pearson Prentice-Hall, 805p, 2008.

[17] BOXER, B. *Parsec Fact Sheeet*. maio 2021. https://drive.google.com/drive/folders/1nlF8PFMrtb8z2fq94WG9oQwrRxAhwsaa/.

[18] MEYER, W. et al. Avaliação de operadores e técnicos de manutenção de máquinas agrícolas no setor canavieiro. *Multi-Science Journal (ISSN 2359-6902)*, v. 1, n. 3, p. 64–68, 2015.

[19] MINITAB. *Interpretar os principais resultados para Boxplot*. maio 2021. https://support.minitab.com/pt-br/minitab/18/help-and-how-to/graphs/how-to/boxplot/interpret-the-results/key-results/.

[20] PARK, S. et al. Measurement noise recommendation for efficient kalman filtering over a large amount of sensor data. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 19, n. 5, p. 1168, 2019.