UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Lucas Henrique Cavalcanti Santos

Improving Mobile Robot Navigation through Odometry Optimization using Particle Swarm Optimization at Kinematics Model

Recife
2023

Lucas Henrique Cavalcanti Santos

Improving Mobile Robot Navigation through Odometry Optimization using Particle Swarm Optimization at Kinematics Model

A M.Sc. Dissertation presented to the Centro de Informática of Universidade Federal de Pernambuco in partial fulfillment of the requirements for the degree of Master of Science in Computer Science. Concentration Area: Computer Engineering.

Advisor: *Edna Natividade da Silva Barros*

Recife
2023

Catalogação na fonte
Bibliotecária Nataly Soares Leite Moro, CRB4-1722

S237i Santos, Lucas Henrique Cavalcanti
Improving mobile robot navigation through odometry optimization using particle swarm optimization at kinematics model / Lucas Henrique Cavalcanti Santos – 2023.
92 f.: il., fig., tab.

Orientadora: Edna Natividade da Silva Barros.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2023.
Inclui referências.

1. Engenharia da computação. 2. Odometria. 3. Navegação autônoma. 4. Cinemática. 5. Robôs móveis. 6. PSO. I. Barros, Edna Natividade da Silva (orientadora). II. Título

621.39 CDD (23. ed.) UFPE - CCEN 2023 – 92

Lucas Henrique Cavalcanti Santos

“Improving Mobile Robot Navigation through Odometry Optimization using Particle Swarm Optimization at Kinematics Model”

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação. Área de Concentração: Engenharia da Computação

Aprovado em: 14/03/2023

BANCA EXAMINADORA

_____

Prof. Dr. Paulo Salgado Gomes de Mattos Neto
Centro de Informática / UFPE

_____

Prof. Dr. Flavio Tonidandel
Departamento de Ciência da Computação / Centro Universitário da FEI

_____

Profa. Dra. Edna Natividade da Silva Barros
Centro de Informática / UFPE
(**Orientadora**)

I dedicate this dissertation to all my family and friends.

ACKNOWLEDGEMENTS

I want to thank my wife, who supported, motivated, and stood by me in challenging times. With her assist this work became a reality. I also want to thank my whole family, that guided me through my life and gave me the best conditions they could, always valuing my study and knowledge. I cannot express how much their example and support since I was a kid changed my development.

I want to thank my advisor, Prof. Edna Barros, who guided me during my bachelor's degree, trusting and supporting me in all my projects and ideas, and now guided me during the master's degree. I'm lucky to have her comprehension and pressure whenever it is necessary.

I want to thank the Centro de Informática (CIn), and all the professors, for the best education, facilities, and infrastructure that leads the course and its students to excellence in computation. I thank Prof. Paulo Salgado for accepting the invitation to examine this work. And I also would like to express my gratitude to Prof. Flávio Tonidandel for this work examination and the robotics program's support which was fundamental to my education.

I cannot forget to thank my RobôCIn's colleagues and classmates, who shared their routine with me and taught me to become a better person. Also, I would like to thank the RobôCIn's young members, that motivate me daily and brought a purpose to my position on the team. Finally, I want to thank my friends Roberto Fernandes, Walber Macedo, Riei Joaquim, Victor Sabino, João Guilherme, Mariana Barros, and others who share their journeys, studies, worries and dreams with me.

"Insanity is doing the same thing over and over and expecting different results."

–Brown, R. (1984)

ABSTRACT

Autonomous navigation is crucial for mobile robots to move and interact with their surroundings. This requires the integration of intelligence, perception, and control in the robots. The first step in modelling the movement of robots is to create a kinematic model that explains how actuators influence their movement. The wheel velocity and the kinematic model are used to calculate the robot's velocity and then the path traveled by integrating velocity over time, known as odometry. Odometry is the foundation of robotics navigation, but due to systematic errors in the kinematic model, it may have translation and rotation errors that accumulate over time. This study introduces a method to improve odometry accuracy using Particle Swarm Optimization (PSO). The method employs wheel velocity data and an inertial sensor to optimize the robot's kinematic model. The technique involves experiments with the robot to record its velocity and position and to simulate the traveled path using the kinematic model. The simulation is evaluated using root-mean-square error compared to the ground-truth positions. The PSO method optimizes the kinematic parameters by minimizing the error between the simulation and the ground-truth positions. The proposed optimization technic improved odometry by 75%, from a mean squared error of 0.37 to 0.09. The result showed that the final position of a 6-meter path had an error of less than 5 cm, while previous methods achieved a minimum error of 10 cm. The optimization allows robots to navigate with greater autonomy without external information or additional sensors and is also efficient for low-power embedded computers.

Keywords: odometry; autonomous navigation; kinematics; mobile robots; PSO.

RESUMO

A navegação autônoma de robôs móveis é importante para a sua movimentação e interação com o ambiente. Isso exige inteligência, percepção e controle nos robôs. O primeiro passo para modelar o movimento dos robôs é criar um modelo cinemático que descreve como os atuadores afetam seu deslocamento. Usando a velocidade das rodas e o modelo cinemático, é possível calcular a velocidade do robô e determinar o caminho percorrido. Esse processo é chamado de odometria e é a base para a navegação autônoma de robôs. Mesmo se baseado na construção do robô, o modelo cinemático possui erros sistemáticos, que se acumulam no processo de integração no tempo. Sendo assim, a odometria também apresentará erros de translação e rotação. Este trabalho apresenta um método para melhorar a precisão da odometria baseado em *Particle Swarm Optimization* (PSO), o qual utiliza dados da velocidade das rodas e de um sensor inercial para otimizar o modelo cinemático do robô. A técnica proposta inclui experimentos com o robô, registrando sua velocidade e posição, e compara a simulação do caminho percorrido com as posições reais. O método de PSO é usado para otimizar os parâmetros cinemáticos para minimizar o erro entre a simulação e as posições reais. Com o processo de otimização, a odometria foi melhorada em 75%, de um erro médio quadrático de 0.37 para 0.09. O resultado mostrou que a posição final de um caminho de 6 metros tinha um erro menor que 5 cm, enquanto outros métodos alcançaram erro mínimo de 10 cm. A otimização permite que os robôs naveguem com maior autonomia sem precisar de informações externas ou sensores adicionais e também é eficiente para computadores embarcados de baixa potência.

Palavras-chave: odometria; navegação autônoma; cinemática; robôs móveis; PSO.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

CPS	Cyber-Physical System
IMU	Inertial Measurement Unit
PSO	Particle Swarm Optimization
QEI	Quadrature Encoder Interface
RMSE	Root Mean Square Error
SSL	Small Size League
WLAN	Wireless Local Area Network

LIST OF SYMBOLS

Δ	Interval
δ	Pose (position and orientation)
$\varepsilon_{max,\theta}$	Path maximum absolute angular error
$\varepsilon_{max,d}$	Path maximum error distance
ω	Angular Velocity
ϕ	Wheel Velocity
θ	Angle
ξ_I	Polar Velocity

LIST OF ALGORITHMS

CONTENTS

1 INTRODUCTION

In the past, computers and electronics revolutionized the industry. Since then, automated process usage has increased, with improved accuracy and manufacturing speed. However, such progress has remained the same manufacturing process in industries; changes only started with technological advancement and the creation of Industry 4.0 (HERMANN *et al.*, 2016). Industry 4.0 proposes a new concept to end conventional production line machines and introduce intelligent systems, seeking autonomy and customization to product manufacturing and leaving static and highly controlled manufacturing aside (WANG *et al.*, 2017).

As Industry 4.0 relies on intelligent systems, various pillars such as Cyber-Physical System (CPS) are necessary. According to authors, CPS are embedded intelligent systems that require network connectivity, sensing, acting and interacting with the environment. Combining these characteristics in intelligent systems requires artifacts to make decisions based on objectives and information.

According to Jay Lee (2015), there are significant challenges in creating systems that combine connectivity with real-time responses and decision-making. One of the areas that extensively explores the previously presented characteristics is robotics, which is one of the main applications of CPS. It is possible to see industries starting to apply mobile robots in factories through Kuka robots (KUKA, 2016) and other companies.

Despite major companies like Kuka betting on the power of robotics to reach our lives through our work environments, the challenges for easy and robust integration between robots and dynamic environments are significantly high. Recent research, such as Honig & Oron-Gilad (2022), evaluated the impacts and challenges of having autonomous vacuum cleaner robots inside people's houses. The vacuum cleaner robot is one of the first mobile applications to break barriers and live in houses. It has a single purpose to navigate and vacuum the entire place. However, it is an intelligent system that must deal with navigation in a dynamic environment. Therefore, it needs precision and quick response to accomplish its task safely.

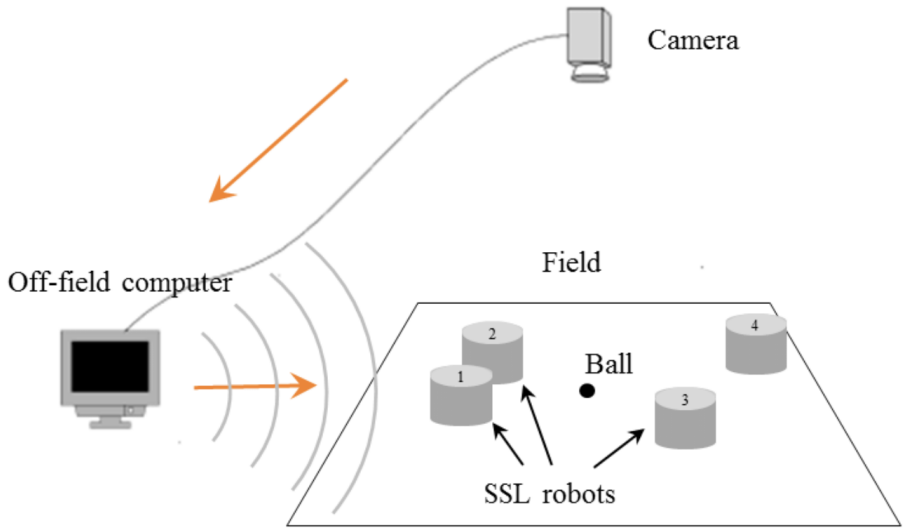
Developing accurate navigation in dynamic environments requires two opposite characteristics — a broad perception of the environment, considering its details, and a fast response to decisions and movements. Moreover, to develop autonomous robotics in dynamic environments, events were created to research and overcome challenges. One example is RoboCup which created the opportunity and motivated autonomous robots research and development since 1997 (KITANO *et al.*, 1997). The event challenges are based on robotics soccer because its complexity and dynamics expand mobile autonomous robot development barriers.

Within RoboCup, various categories were created, each attacking a subset of challenges in different areas of robotics. One of the most dynamic is the Small Size League (SSL), which provides an external localization system, the ssl-vision (ZICKLER *et al.*, 2010), that uses computer vision to track robots and ball. With the location system, each team must build its system composed of an off-field computer with artificial intelligent teams and 6 or 11 robots to

play against another team. Therefore, the teams must develop intelligent systems that cooperate between multiple robots and adapt themselves depending on the adversary without human interference.

Soccer is a platform that requires fast decision-making in a dynamic cooperative and adversarial environment. According to Siegwart *et al.* (2011), there are four pillars for a robotic system, perception, localization, cognition, and mobility. Therefore, in the SSL robots use the perception and localization provided by the competition, as shown in Figure 1, leaving for the teams the challenge to build valid cognition and mobility. The movement of the robots happens through radio frequency wireless commands that comes from the processing performed on off-field computer. Artificial intelligence chooses the robot's actions, and communication system deliver the controls to the robot, which move accordingly.

Figure 1 – SSL architecture overview, with robots being recognized from an external system, and controlled by an offboard computer.



Source: Yoon *et al.* (2016)

Although the teams build an autonomous system in the SSL using external vision and computer, the coordination and dynamic of the soccer skills are similar to the Industry 4.0 robots. Moreover, in recent years, the soccer robots are reaching more than three meters per second in their navigation; and makes crucial to have fast and precise navigation system into the robots. Therefore, the robotics embedded system requires its sensing and processing to become autonomous and interact with the environment.

Embedding movement in the robots requires developing their navigation, where localization is essential. The state-of-the-art embedded localization and navigation uses multiples sensor (KOLAR *et al.*, 2020), as information overlap enhances the perception quality. The sensors often used are cameras, lasers, or wireless signal triangulation. Although sensor's has improved in

past years, localization and navigation using the fusion of multiple sensors, such as cameras, are expensive and numerous times slow.

Tracking a robot's movement requires its kinematic model, which describes how its actuators contribute to its movement. The model, combined with the actuators' data, builds the odometry, which estimates the robot's movement and tracks its path over time. The critical part of monitoring a robot's movement is its model, as any modelling inconsistency creates a difference between the robot's actual movement and internal belief. Although crucial, the kinematic model often has multiple physical issues because the robot's construction does not entirely match the designed model. Moreover, even minor differences between model and mechanics accumulate path inaccuracies over time, as odometry is continuously summing a robot's movement to create a robot's path.

Whether the navigation system uses multiple sensors or only robot wheels' data, odometry is a base skill for navigation and localization. Therefore, it is essential even with the typical physical issues in the kinematic model. Because of its necessity and impact on navigation, researchers started proposing methods to calibrate and fix kinematic model mechanical inaccuracies, to deliver inexpensive and effective navigation based on odometry. The odometry improvement through the robot's kinematic model started many years ago. Borenstein & Feng (1996b) work proposed a closed-form equation to calibrate the parameters from a differential robot; to do so, it used a fixed path with a few manual measurements between the robot's calibrated and original odometry-based position. Recent work proposed by Lin *et al.* (2019) optimized the kinematic model parameters instead of the robot design. It also presented a technique with multiple distance measurements between optimized odometry and the robot's ground truth positions to evaluate the optimization quality. However, Lin *et al.* (2019) used an analytic optimization equation targeting a tricycle robot.

With the robotics evolution and the increase in the environment dynamism, many robots started to have more complex models as their requirements for fast and precise navigation increased. For example, omnidirectional robots move in any direction, increasing the complexity of the kinematic model. Moreover, applying these robots to soccer games or people's house requires precise navigation with fast reactions. More than ever, the navigation methods on top of odometry require a kinematic model that matches more complex and different robot designs. Therefore, Sousa *et al.* (2022) proposed work to optimize odometry through the kinematic model from the differential, tricycle and two types of omnidirectional robots. However, the optimization still requires analytical work to match the robot's design. Although Sousa *et al.* (2022) odometry-based path evaluation method also used multiple distance differences, the sampling is between every 0.5 meters and may cover odometry mistakes.

This work, unlike the previous, proposes a generic solution to optimize odometry through the kinematic model parameters. The methodology presented here uses optimization algorithms instead of closed-form equations to avoid analytic re-work to adapt the technique for different robots. Improving the odometry through the kinematic model without closed-form equations

requires experiments with robot navigating to collect data. The data, and the robot's kinematic model, enable the odometry-base path simulation, reproducing the kinematics model's mechanical inaccuracies. The data is also used to evaluate the odometry navigation quality from the kinematic model utilized. Consequently, simulation and evaluation compose the optimization that aims to improve navigation based on the robot's odometry through the kinematics model optimization. Moreover, this work seeks to answer the following challenges:

- How to simulate the robot's odometry containing systematic kinematic errors?
- How to evaluate the quality of a path between a robot's odometry path and its ground-truth position, generic for different robots' structures.
- How to optimize navigation from robot's odometry through the kinematic model, using the generic simulation and evaluation to fit different robot structures.

This work presents three main contributions to answer to solve these challenges:

- A method to simulate and evaluate robots' paths, which enable off-board parameter optimization. Therefore, it preserves the hardware and speed the calibration process.
- An odometry optimization method which optimize kinematic model parameters independent of the robot and improves path tracking.
- An optimized odometry parameters for an omnidirectional robot. The parameters improves the robot path tracking, despite the complex robot's drive.

The following chapters detail the background, existent solution, the method proposed to overcome the challenges above and its evaluation and comparison with previous works. The Chapter 2 presents the theoretical background, right after Chapter 3 presents the solutions in the literature. The proposed method is shown in Chapter 4, and Chapter 5 details implementation and use case experimentation. Finally, the Chapter 6 presents the odometry accuracy achieved and compares it for different experiment cases. The last chapter, Chapter 7, discusses the performance and innovation of this work, compare it to the literature, and proposes future works.

2 THEORETICAL BACKGROUND

Robots can have various applications, such as transportation, handling materials, searching objects, and watching spaces. Mobile navigation in static and dynamic environments is a common task required for performing these tasks. According to researchers (GUL *et al.*, 2019), mobile navigation is essential, and several technics aim to navigate robots freely in a static or dynamic environment. A robot that navigates is commonly called a mobile robot, as one of its skills aims to move between positions in the environment. Moreover, an autonomous robot is a robot that performs its tasks.

This chapter presents the background for this work, explaining the types and challenges of navigating robots. Then it describes how to properly model robots to control and take accountability for robot movements. Finally, it presents an inertial measurement sensor, which can contribute to the robot's perception of the environment. The last section of this chapter presents how to optimize functions using evolutionary algorithms, such as Particle Swarm Optimization, which is essential for fixing inconsistencies in models and the real world.

2.1 MOBILE ROBOT'S NAVIGATION

Robotic navigation refers to the ability of a robot to move around and navigate to different locations within that environment. Therefore, robots require perceiving and act in the environment to navigate. Studies show different sensors and algorithms to understand the robot's surroundings and determine the best path to a desired location (GUL *et al.*, 2019). It also categorizes robotics navigation into two types, global and local navigation.

Global navigation uses external references such as GPS (Global Positioning System) or other external landmarks to determine the robot's position and orientation within a larger environment. It may also involve using higher-level map representations to determine a longer-term path to a desired destination. Global navigation often appears in exploration tasks, mapping, or long-distance transportation; a typical example is car navigation assistance.

On the other hand, local navigation often involves using sensors such as lasers, sonar, or vision systems to perceive the robot's immediate surroundings and avoid obstacles. Processing sensory information can be a complex task, especially in a dynamic environment where the structure changes, as the robot, must be able to perceive its surroundings and make decisions based on that information. Although complex, it enables robots to perform tasks and interact with their environment efficiently, like domestic vacuum cleaning robots.

Robotics navigation has years of study and research, and since 1991 its importance was already presented by Borenstein & Koren (1991). In Borenstein & Koren (1991) work, researchers reviewed different techniques and approaches capable of performing robotics navigation, including using sensors, map-based approaches, and reactive control methods. The Borenstein & Koren (1991) work focused on local navigation due to its challenges, and years later Nguyen & Christensen (2008) did a review of mobile robot navigation technics and stated

new challenges. The complex and dynamic environments in which people live bring new requirements and challenges for autonomous navigation, such as quick adaptability, responsiveness, and performance of the algorithms.

While in streets, it is possible to use its map and external sensors like GPS, in indoor environments, it is necessary to use local sensors and create a minimum understanding of the surroundings to navigate. Building surrounding knowledge is essential to localize robots, as navigation moves them from point A to point B. Research also shows that understanding a robot's position is challenging (FANG *et al.*, 2018) because many applications need mapping or rely on any reference that robots find. In these cases, localization uses a statistical belief based on perceptions available from sensors to determine object position.

One of the fundamentals of statistical position tracking is movement tracking Shanavas *et al.* (2018) because the robot's external perception needs to be combined with the robot's movement to keep increasing the position belief Siegwart *et al.* (2011). Tracking a robot's movement is tracking its displacement around the environment, so it requires internal sensors that track the robot's actuation. The internal sensor also needs to be combined with the robot's structure to determine the robot's movement. With a proper combination of actuators and the robot's structures, the displacement will reflect the actual position change and reduce the robot's localization error. Then, a mobile autonomous robot requires a well-designed combination of perception and robot structure.

Describing the robot structure is accomplished by the Kinematics. Kinematics is the study of how an object moves, and it is typically concerned with position, velocity, acceleration, and the relationships between these quantities. Inside a robot, the kinematic describes how an element contributes to the robot's movement.

2.2 KINEMATICS MODEL

Kinematics is the study of how mechanical systems work. All these systems have mechanical systems, whether a robot manipulator, a mobile robot or a car. According to robotics researchers (SIEGWART *et al.*, 2011), manipulators and mobile robots share kinematics challenges because the active engagement of motors defines mobile robots and arm's controllability. Moreover, the workspace is where arms and mobile robots can reach, moving from pose to pose. The main difference reported is pose estimation. For arm position, the kinematic calculus of actuators' positions happens at any time from the actuator position readings. However, a mobile robot can freely move in its environment, and there is no instant way to measure position with actuator data. Instead, it is necessary to integrate the robot's movement over time to track its position. Motion estimation includes slippages, making the robot's position measurement extremely challenging, according to Siegwart *et al.* (2011).

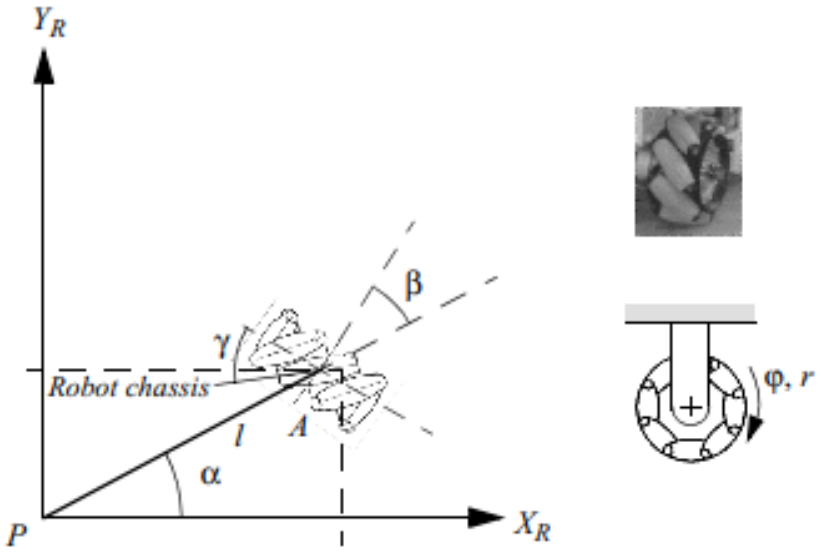
Tracking a position over time requires a Kinematic model, which converts actuators' movement into robot movement. It starts by describing the contribution of each actuator in

the robot body. In mobile robots, the description starts with the body's maneuverability and controllability. Maneuverability describes the robot's direction freedom, and controllability is the difficulty of controlling these movements. This description starts with the wheels and their types; a car wheel only goes forward and backward, creating a restriction in the lateral movement. Conversely, a castor wheel or omnidirectional wheel allows side movements.

One of the most complex types of a mobile robot is the omnidirectional (OLIVEIRA *et al.*, 2008), as it can move in any direction. Omnidirectional is a powerful drive that can reach every position in its workspace. This type of robot has high maneuverability and is hard to control. Moving in any direction makes the control complex; however, it can reach any place. The construction of an omnidirectional mobile robot may use castor wheels or omnidirectional wheels. A Castor wheel appears in shopping cars, and omnidirectional wheels have small angled pulleys in the wheel circumference to allow angled movement.

The omnidirectional wheels, also called the Swedish wheel, allow side movements. As shown in Figure 2, the omnidirectional wheel has rollers that allow side movements. It can be perpendicular to the wheel plane or have an angular difference, and it only needs to be accounted for in the robot's kinematics model.

Figure 2 – A holonomic wheel and its parameters.



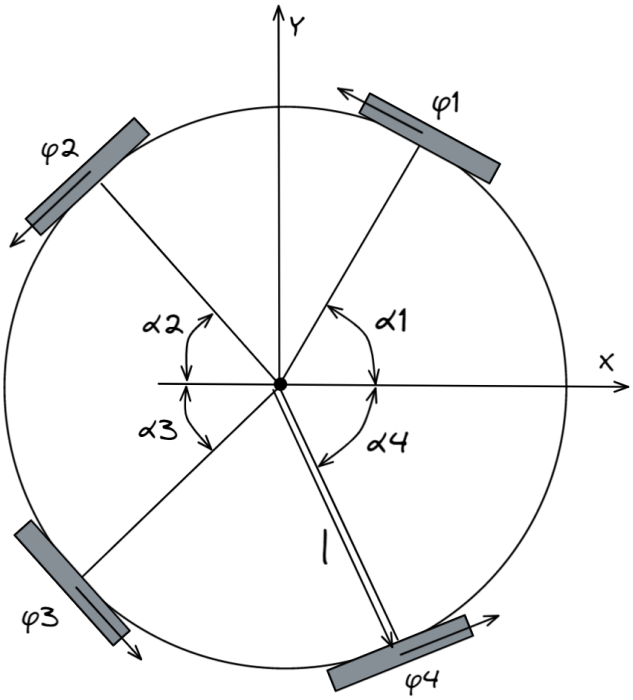
Source: Siegwart *et al.* (2011)

The complexity of the omnidirectional robot comes from the lack of wheel restrictions, which gives movement flexibility but requires precise control and kinematic modelling due to the dependency between the wheels and the robot's movement. An excellent example is cars; while front-wheel drive can push the car and change its direction, the rear wheels do not have side movements. However, in omnidirectional robots, all wheels can have side movement, which the model needs to account for the lack of side restriction.

2.2.1 Modeling

There are the kinematic and inverse kinematic. At the same time, the kinematics model is composed of some equations that convert the wheel's movement into the robot's movements; the inverse kinematics converts the robot's movement into the wheels' movement. It is necessary to describe the robot structure to create its model. The structure description contains the wheels' distances from the robot's center and its angle from the x-axis of the robot; for example, the Figure 3 shows a 4-wheel circular omnidirectional robot.

Figure 3 – A model of one omnidirectional robot with four Sweden wheels, which allows perpendicular movements to the wheel plane. Each wheel has l distance from the robot's center, and the wheel's forward rotation, ϕ_1, ϕ_2, ϕ_3, ϕ_4 respectively, rotates the robot in counterclockwise. The last parameter is wheel's angle, wheels have α_1, α_2, α_3, α_4 angle from the robot's X axis respectively.



Source: The author

The wheels' movement is represented as ϕ and the robot's movement by ξ_I. The modelling describes how each wheel contributes to the robot's x, y and ω movement. The wheel spin contribution for the robot's movement is described by Equation (2.1) (SIEGWART *et al.*, 2011). The Equation (2.1) only considers one wheel. However, it is possible to describe it for all wheels contributions using matrix equations. All wheels equation, for the 4-wheel robot presented in Figure 3, is presented at Equation (2.2). The Equation (2.2) describes the inverse kinematics, calculating the robot's movement (ϕ) from the wheel's movement (ξ_I).

$$\begin{pmatrix} sin(\alpha+\beta+\gamma) & -cos(\alpha+\beta+\gamma) & (-l)cos(\beta+\gamma) \end{pmatrix} R(\theta)\dot{\xi}_I - r\dot{\phi}cos(\gamma) = 0 \quad (2.1)$$

$$\dot{\phi} = J_2^{-1} J_{1f} R(\theta)\dot{\xi}_I \quad (2.2)$$

The wheel's Equation (2.1), with the geometry parameters applied, builds the matrix J_{1f}, shown in Equation (2.3). In Equation (2.3), each line represents the wheel 1, 2, 3 and 4 equation, respectively. Each column represents the wheel contribution to x, y and ω movement, respectively, where x and y are the translation at the floor plane, and ω is the robot's rotation in its center. The radius in the wheel equation is aggregated in the Equation (2.4); however, for Equation (2.2), it is necessary its inverse, the J_2^{-1} matrix, as it is a diagonal matrix the inverse is the inverse of each value, as presented in Equation (2.5). The last necessary matrix is the rotational matrix, $R(\theta)$, which converts the robot's local coordinates to global ones in Equation (2.6). With J_{1f}, J_2^{-1}, $R(\theta)$ and desired robot speed, stated as $\dot{\xi}_I$ the wheels speed, represented by $\dot{\phi}$ is achieved using Equation (2.2).

$$J_{1f} = \begin{pmatrix} -sin(\alpha_1^\circ) & cos(\alpha_1^\circ) & l \\ -sin(\alpha_2^\circ) & -cos(\alpha_2^\circ) & l \\ sin(\alpha_3^\circ) & -cos(\alpha_3^\circ) & l \\ sin(\alpha_4^\circ) & cos(\alpha_4^\circ) & l \end{pmatrix} \quad (2.3)$$

$$J_2 = \begin{pmatrix} r & 0 & 0 & 0 \\ 0 & r & 0 & 0 \\ 0 & 0 & r & 0 \\ 0 & 0 & 0 & r \end{pmatrix} \quad (2.4)$$

$$J_2^{-1} = \begin{pmatrix} 1/r & 0 & 0 & 0 \\ 0 & 1/r & 0 & 0 \\ 0 & 0 & 1/r & 0 \\ 0 & 0 & 0 & 1/r \end{pmatrix} \quad (2.5)$$

$$R(\theta) = \begin{pmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (2.6)$$

The Equation (2.2) uses the robot's speed ($\dot{\xi}_I$) to calculate the wheel speed ($\dot{\phi}$), normally called as inverse kinematics. However, it is possible to isolate the robot's speed ($\dot{\xi}_I$) in the inverse kinematics equation. This calculus results in the kinematics equation, described at Equation (2.7), and it describes the robot's speed ($\dot{\xi}_I$) in the function of model matrixes and wheels' speed ($\dot{\phi}$).

The kinematics equation requires the inverse of J_{1f} the matrix, and as it is not a square matrix, it has no inverse. To find J_{1f}^{-1}, the generalization of the pseudo-inverse matrix method uses the method recommended by Adi Ben-Israel (2003).

Although kinematics and inverse kinematics convert from robots to wheel speed and vice versa, they can also convert movements, which consist of speed integration in time. In the literature, different authors may call inverse-kinematics kinematics and kinematics as inverse-kinematics (SIEGWART *et al.*, 2011), (ROJAS & GLOYE, 2005).

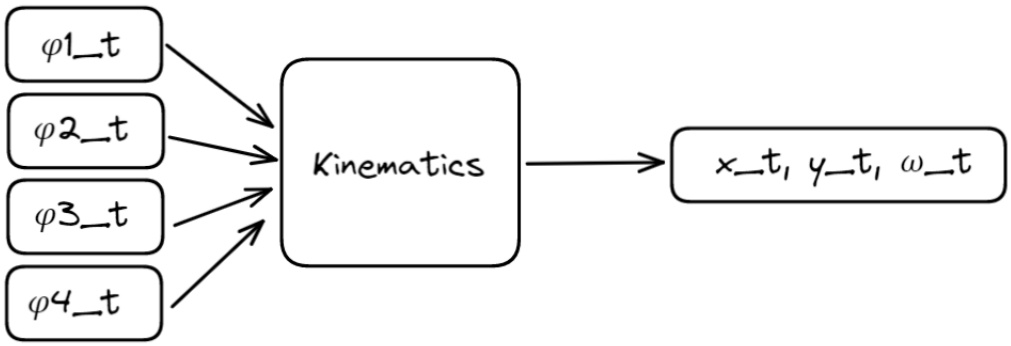
$$\dot{\xi}_I = R(\theta)^{-1} J_{1f}^{-1} J_2 \dot{\phi} \tag{2.7}$$

2.2.2 Odometry

Different navigation algorithms require robot movement tracking, as it is an important data source for statistical localization algorithms (SHANAVAS *et al.*, 2018) (ARVANITAKIS *et al.*, 2017). This tracking process is called odometry, and it is achieved by integrating the robot's speed in time, which results in robot movement. The direct kinematics model uses Equation 2.7 to calculate the odometry.

Movement tracking starts integrating each movement step that the robot does. A kinematic step can represent each movement step. Shown in Figure 4, a kinematic step converts the wheels' speeds, ϕ_1, ϕ_2, ϕ_3, ϕ_4, at any time instant, to the robot's speed, x, y, ω at the same time instant. Both wheels' and robot's speeds can be integrated in time to discover the wheels' or robot's movement, respectively.

Figure 4 – Diagram show how to calculate one step of the Kinematics. In diagram $\delta\phi 1$, $\delta\phi 2$, $\delta\phi 3$, $\delta\phi 4$ are the wheels speed and using kinematics model it is converted to robot's speed (x, y, ω). This step can be applied any time, and integrating by time, it converts speeds to movement.

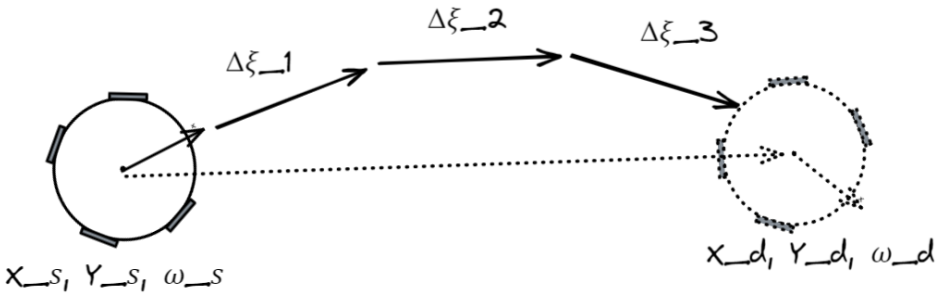


Source: The author

Integrating the robot's speed will produce the robot's movement, which is the odometry. The Figure 5 illustrates a situation in which the robot's position can be calculated using odometry. At Figure 5, the robot moves from position x_s, y_s, ω_s to position x_d, y_d, ω_d divided by three-time

steps, 1, 2 and 3. At each step, the robot does a movement, represented by $\delta\xi$, and the vector sum of source position x_s, y_s, ω_s, with movement $\delta\xi$ from all time steps, will achieve the destination position x_d, y_d, ω_d.

Figure 5 – Diagram of robot moving from x_s, y_s, ω_s position to x_d, y_d, ω_d position split by time instant 1, 2 and 3. In each timestamp, the robot does a $\Delta\xi_t$ movement, and the total movement is the integration of these intermediary movements. Then, x_d, y_d, ω_d position is x_s, y_s, ω_s position plus the integration of $\Delta\xi_t$ steps.



Source: The author

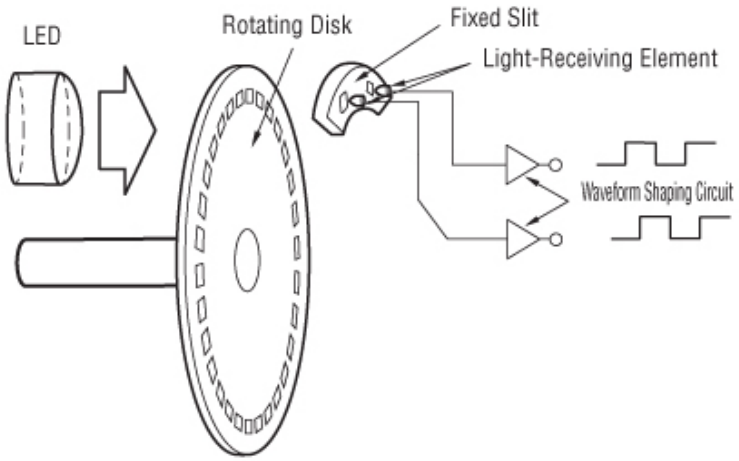
Another critical part is the source of the odometry, the wheel's speed. It comes from encoder sensors. Encoders monitor the wheel speeds by pulses that begin with angular movement from the wheel. Each pulse happens at a fixed angular difference from the wheel. In Figure 6, there is an internal diagram of an optical encoder, which creates pulses by the motor spinning encoder disk which alternates the light incidence in the optical sensor. There are other encoder techniques to generate pulses, like a magnetic sensor, while a disk has magnetic. Every encoder needs to generate two signals, with an angular difference, to create alternated pulses in time; the pulse difference determines the motor rotation direction on top of its speed. The direction of movement is critical because a kinematics model made from the J_{1f} matrix considers each motor's direction. It is necessary to correctly aggregate each wheel's contribution to the robot's movement.

To track a robot's position in the environment is crucial to record its movements. The tracking process brings the importance of odometry, made from kinematics models (GANGANATH & LEUNG, 2012A). However, the robot's wheels can slip on the floor, creating uncertainty in the movement tracking because the wheels can rotate without the robot changing its position. Slipping is a known problem typically solved with statistics algorithms and sensor fusion to create an information overlap.

2.3 INERTIAL MEASUREMENT UNIT

An Inertial Measurement Unit (IMU) has a combination of sensors that measure and track an object's motion. It typically includes accelerometers, gyroscopes, and sometimes a

Figure 6 – An internal diagram of an optical encoder sensor.



Source: Siegwart *et al.* (2011)

magnetometer (You, 2018).

Accelerometers measure linear acceleration, such as the force exerted on an object, and change its speed. Gyroscopes, however, measure angular velocity, also known as the rate at which an object rotates around its center. A magnetometer measures the strength and direction of the magnetic field surrounding an object; it usually points to the earth's magnetic field.

Many applications such as navigation, motion tracking, and control systems widely use IMU. It is common in mobile devices such as smartphones and tablets to track the device's orientation. IMU is also vital in robotics and drones for navigation and stabilization.

Researchers highlight that constructing an IMU brings important features for robotics. First, it measures an object's intrinsic movement, such as angular speed and acceleration. Unlike wheel encoders that measure wheel rotation, IMU does not need environmental interaction with the environment, such as the floor (Baghdadi *et al.*, 2018).

Intrinsic measures are essential to movement tracking because no matter the surrounding environment, one type of movement will result in a similar sensor reading. On the other hand, a wheel may slip, and then the same measure could occur from different movements. Another essential characteristic of IMU is the reading sample rate. As the sensors are electronic, they depend on external interactions and can reach over one thousand samples per second. According to research, different sensors measuring different object movement is crucial to improve tracking accuracy, and IMU's characteristics make it recurrently used for motion tracking (Borenstein & Feng, 1996a).

Even though IMU brings a high sample rate and intrinsic measures characteristics, it still has inherent limitations such as drift and bias, which can lead to measurement errors over time. Authors (Qureshi & Golnaraghi, 2017) suggest different calibration algorithms overcome these limitations, such as sensor fusion algorithms to combine data from multiple IMU sensors and other sensors, such as GPS, to produce more accurate and reliable results.

2.4 PARAMETER OPTIMIZATION

Parameters are an essential aspect of many problems; they define the characteristics of the problem and can be adjusted to find a better solution or optimize the performance. Solutions often involve parameters, whether a machine learning model or a physics problem; the correctness of parameters is relevant to the problem solution. For example, gravitational acceleration differences can change construction requirements. When it is impossible to calculate or obtain an accurate parameter, models will have physical inconsistencies.

One of the methods used to overcome parameter inaccuracy is parameter tuning; as long as it is possible to quantify the error, it is possible to search for better parameters. However, researchers say searching can be expensive because experimenting with all possible parameters in a real problem can be an infinite process (BERGSTRA *et al.*, 2011). Moreover, as the solution increases the number of parameters, the dimension of the search increases because one parameter may interfere with the other. Ultimately, the time to find a better combination of parameters grows exponentially.

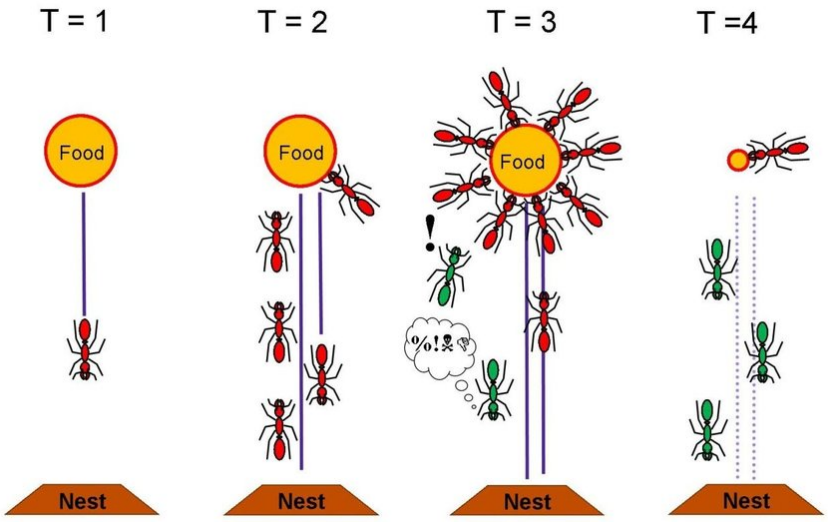
Searching for parameters is typically done through grid search, random search, genetic algorithms, evolutionary algorithms or gradient-based optimization methods. Although there are many techniques, the systematic review reports better algorithms or combinations between problems and optimizer (YU & ZHU, 2020). There is no correct technique; it varies according to the problem.

There are parameter optimizations based on natural evolution; they are genetic algorithms. It means they encode the optimization problem as an individual being and operate according to the nature (VIE *et al.*, 2021). For example, a genetic algorithm may encode the problem as a chromosome and realize combination and mutation. Unlike a grid search, where all possible solutions are tests, a genetic algorithm smartly searches for the best parameters to optimize the problem (NEUMANN *et al.*, 2022). There are two critical concepts in genetic algorithms; first, the particle, or an individual solution, means a set of parameters, also called a solution; another concept is the fitness, a quantity that measures how well an individual or solution is to the solution.

Besides genetic algorithms inspired by nature, there ate the swarm algorithms, but now multiple particles are involved. The main difference is the information exchange between particles. An ant colony is a swarm example, as shown in Figure 7. In this case, pheromones attract ants, and ants also release them in paths where ants find food. Therefore, whenever there is food, the path to it gets more pheromone concentrated, and more ants start going in that direction. On the opposite, when there is no food or food end, ants stop sending pheromones, and the path loses its attraction.

In swarm optimizations, the solution is not encoded in chromosomes, so, particles are not combined. To evolve the population smartly the swarms' algorithms exchange information such as the direction where the good solution was found. The information exchange in the ant

Figure 7 – Ant colony organization, in which pheromones attract ants. The pheromones are released also by the ants, whenever a favorable path to food is found. This guide more and more ants to food, on the other hand, when food is ending pheromone lose its strength and disperse the ants.



Source: Czaczkes (2012)

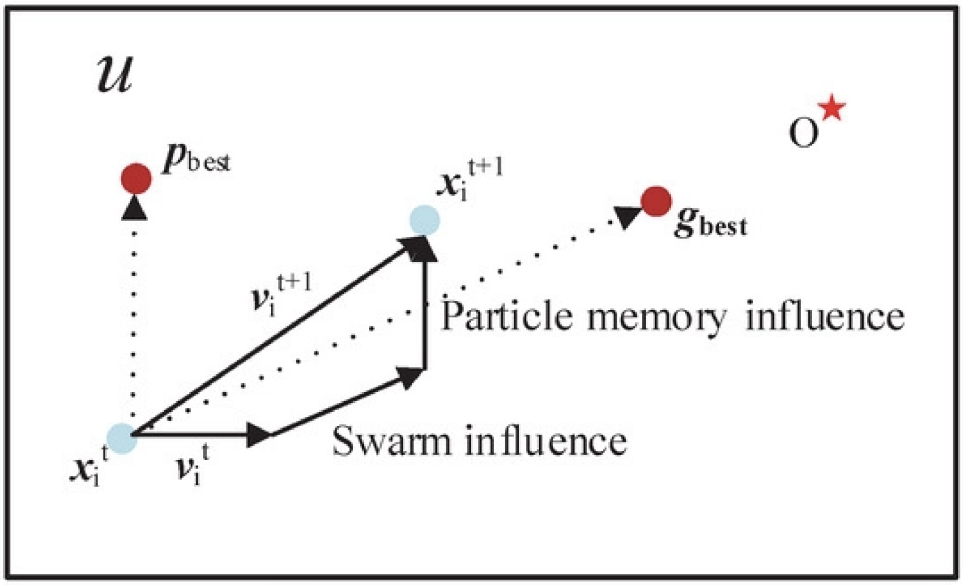
colony optimization algorithm (ACO) is based on the pheromones that guide particles to explore parameters with good solutions (DORIGO *et al.*, 2006). This concept clones the ant colonies' behaviour, where solutions are the ant's position and good fitness relates to food which releases pheromones and attracts other particles to explore the solution space in that direction. With more particles reaching high-quality solutions, pheromones increase around that set of parameters.

2.4.1 Particle Swarm Optimization

Another evolutionary algorithm using a swarming concept to find the best parameters is Particle Swarm Optimization (PSO). Research shows that PSO can produce better and faster solutions than genetic algorithms (KECSKÉS *et al.*, 2013). It is an optimization technique inspired by the behaviour of flocks of birds or schools of fish. Moreover, the movement of one particle influences the direction and strength of the movement from other particles, like flocks. Similar to flocks, PSO particle simulates the has its speed and moves according to flocks. As shown in Figure 8, the swarm comes from particle memories; one memory comes from the particle's best direction, and another is the global best solution direction. These two memories will be added to the current particle speed to produce the next particle speed, guiding particle movement across the solution space (ZHANG *et al.*, 2018).

The implementation of the particle parameters in the solution space follows the Equation (2.8), where each particle is represented by i. The velocity changes according to the Equation (2.9), where i is the particle and j is the iteration. The particle velocity equation considers its current velocity as $v_{ij}(t)$, the particle best solution as $r_{1j}(t) \times [y_{ij}(t) - x_{ij}(t)]$, and

Figure 8 – An iteration of Particle Swarm Optimization, updating particle speed according to the memory of the personal best solution and the global best solution.



Source: Zhang *et al.* (2018)

the population best solution as $r_{2j}(t) \times [y_j(t) - x_{ij}(t)]$, together with three hyperparameters C_1, C_2 and W.

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{2.8}$$

$$v_{ij}(t+1) = w \times v_{ij}(t) + c_1 \times r_{1j}(t) \times [y_{ij}(t) - x_{ij}(t)] + c_2 \times r_{2j}(t) \times [y_j(t) - x_{ij}(t)] \tag{2.9}$$

Systematic reviews show that PSO is a relatively simple and computationally efficient optimization method that applies to many optimization problems (KENNEDY & EBERHART, 1995). It is especially effective for problems with multiple parameters or when the optimization problem is multimodal, meaning that there are multiple possible solutions with similar optimal performance (GAD, 2022). PSO has been successfully used to optimize various neural network architectures. It showed competitive results with traditional optimization techniques such as gradient-based methods and other evolutionary algorithms (GUDISE & VENAYAGAMOORTHY, 2003).

However, it is essential to note that the performance of PSO depends on the specific problem and the quality of the implementation. Choosing an optimization method should be based on the problem's characteristics, the available data, and the computational resources available.

2.5 WIRELESS DATA COLLECTION

Mobile robots often require a power supply as it moves around. Mobile power brings challenges to monitoring and collecting data from its sensors, actuators and embedded systems. Some robots, such as marine and outer space inspection robots, exist to collect data (DAS *et al.*, 2015). However, domestic vacuum cleaning robots also require data collection to report their status or request maintenance. Collecting mobile robot data by wireless communication is applied; however, different applications will require different communication system technologies and topologies (HUANG *et al.*, 2019).

In general, researchers from Huang *et al.* (2019) point that wireless data collection is important for robotics for several reasons, such as:

- Flexibility: Wireless communication allows robots to move freely and collect data from various locations without being tethered to a device. It increases the robot's flexibility and versatility, allowing it to perform tasks in various environments.
- Real-time data: Wireless data collection allows robots to transmit data in real-time. Wireless communication enables the robot to receive commands and feedback from the computer in real time, which improves the robot's ability to perform tasks quickly and accurately.
- Remote control: Wireless communication allows a human operator to remotely control a robot, which is particularly useful in dangerous or inaccessible environments where a human cannot be present.
- Collaboration: Wireless data collection enables multiple robots to communicate and collaborate. Collaboration is vital in applications such as search and rescue, where multiple robots work together to locate and aid victims.
- Safety: Wireless data collection can be used to monitor the robot's status and detect if it has issues. The data allows actions to prevent accidents or failures that could harm the robot or its surroundings.
- Optimization: The data collection for mobile robots enables optimization, such as parameter optimization and reinforcement learning, that is later applied to the robot and improves performance.

Some studies compare wireless technologies such as Wi-Fi, WLAN, Bluetooth, and Mobile Data network (HASSAN, 2012). Moreover, it points to technology's advantages and disadvantages depending on the application. For robots that require access to the internet, Wi-Fi is the most recommended; however, in an outdoor environment, a Mobile Data network, such as 3G, may be applied. Wi-Fi may suffer from protocol decisions for real-time communication to avoid data transmission interference; therefore, Bluetooth is an alternative; For applications with

a wide range, Bluetooth suffers from data loss and may replace it Wireless Local Area Network (WLAN).

For indoor autonomous mobile robots, third-party protocols may be applicable. Research shows that Nordic nRF24 radios are efficient, appropriate for communication and feedback networks with rates over 60 packets per second, and are low-power (CAVALCANTI *et al.*, 2022). Therefore, appropriate mobile indoor robots rely on batteries.

2.6 SIMULATION

The advent of computers has dramatically enhanced simulation potential as a powerful visualization, planning, and strategy tool in various research and development (ŽLAJPAH, 2008). Researchers surface that simulation is crucial in robotics because it overcomes many challenges. For example, analyzing the kinematics and dynamics of robotic manipulators, programming offline, designing control algorithms, creating mechanical structures for robots, and designing robotic cells and production lines.

Simulation plays a vital role for mobile robots because it allows testing and evaluating the robot's performance and control algorithms in a virtual environment before deploying it in the real world. As deploying robots to the real world is costly, simulation model accuracy is crucial to simulations, and authors propose different evaluation (JU *et al.*, 2014) (QUEIROZ & FERREIRA, 2022). Methodologies to build a simulation to custom robots and its applications (MAYER *et al.*, 2004).

One application in which simulation is crucial is surgery, and nowadays, many robots are performing surgery (WU *et al.*, 2020). Researchers also affirm that robot simulation needs to be developed, validated, and improved before any real-world test.

Learning and optimizing robotics performance depend on simulations; authors create simulators to enhance environment correctness and produce learning frameworks (MARTINS *et al.*, 2022). Not only for learning frameworks, such as reinforcement learning, that simulation has been used for; authors also rely on it to optimize robot models for driving and overall movements in real-world environments (OSIŃSKI *et al.*, 2020).

3 RELATED WORK

This chapter presents related research which aims to optimize odometry through a kinematics model to aid autonomous navigation in mobile robots. Navigation requires determining the robot's position within the environment where it is operating (MELO *et al.*, 2022). As discussed in Chapter2, there are two types of navigation, global or local, and its classification relies on the robot's knowledge of the environment. Both types are applied depending on the localization system, absolute and relative, which combines sensor fusion. Absolute localization typically relies on map matching and identification of active or passive landmarks or beacons, thus used in the global navigation. On the other hand, relative localization generally depends on the wheel, visual, or laser odometry and is usually used in local navigation (GANGANATH & LEUNG, 2012B) (LIU *et al.*, 2019).

Moving a mobile robot, no matter the type of navigation, requires the robot's kinematic model. Odometry construction needs the kinematic model and wheels' speed, and it is responsible for tracking the robot's movement, thus enabling position tracking for any autonomous navigation. It is essential to highlight that a kinematic model is based on the robot's structure, including the wheels' format and restrictions, while interacting with the environment. Furthermore, although parameters can be measured, research points to uncertainty in kinematics modelling because of mechanical issues in robot parameters (IVANJKO *et al.*, 2023).

Because of the kinematics parameters uncertainty, systematic error is a well-known disadvantage in using odometry. The accumulation happens because, with physical parameter issues, the wheel odometry will propagate this error. And with the motion tracking over time, the errors will accumulate.

In the literature, there are suggestions to overcome odometry limitations. Some authors, like Qin *et al.* (2019), suggest adding more sensors to create an overlap of information and combine it to reduce the path tracking error. Exploring this method Liu *et al.* (2019) suggested using cameras to build more robust local odometry. In contrast, (NEMEC *et al.*, 2019) adds inertial sensors alongside the camera.

Although cameras bring information and aid the robot's localization, it requires much processing. On the other hand, odometry path tracking from the wheels is inexpensive, easy to apply, and runs at a high sampling rate. Therefore, authors like Tomasi & Todt (2017), Sousa *et al.* (2022), and others proposed to fix odometry through the kinematics model to maintain path tracking high sampling rate and increase navigation accuracy.

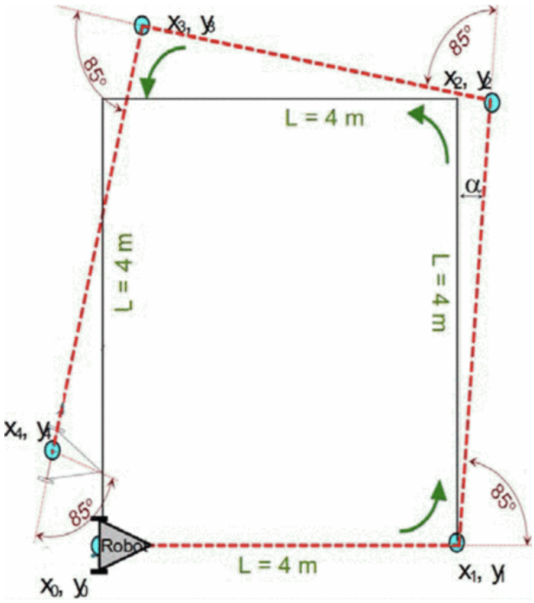
To reduce path-tracking errors in odometry, researchers proposed two approaches. Both methods require robot experimentation, but the first target is to find accurate robot parameters to reduce modelling issues using analytic methods. Differently, the second method focuses on the kinematic model values to minimize odometry path tracking error and uses optimization algorithms. In both approaches, there are crucial characteristics to understand the solution and its scope; these characteristics are the type of robots, if it calibrates the robot's parameters or

optimizes the kinematic model, which data was used, how it was evaluated, which algorithm was used, and the result. Therefore, the following sections analyzed calibration and optimization methods characteristics present in the literature, later table 1 summarizes the contribution and differences from each work.

3.1 ODOMETRY CALIBRATION

The calibration of the robot's measurements started long ago; researchers from 1996 have already begun to present calibration methods (BORENSTEIN & FENG, 1996B). Most of the calibration methods are based on the analytic solution of the path robot performed, compared to the path it was programmed. Consequently, analytical methods work to specific routes and robots, as shown in Figure 9. The analytics methods evaluate the robot's path error in particular points to fill the proposed calibration equations. One of the most recent works in the calibration method was presented by Tomasi & Todt (2017) and explored a circular experiments path to calibrate differential robot physical measurements. The Tomasi & Todt (2017) result achieved a path tracking error of around 0.2 m; it optimized wheels size and distance from the robot's center using five experiments and was manually measured.

Figure 9 – A calibration square path, where robot's performance error evaluation happens in each corner to fill analytics calibration method.



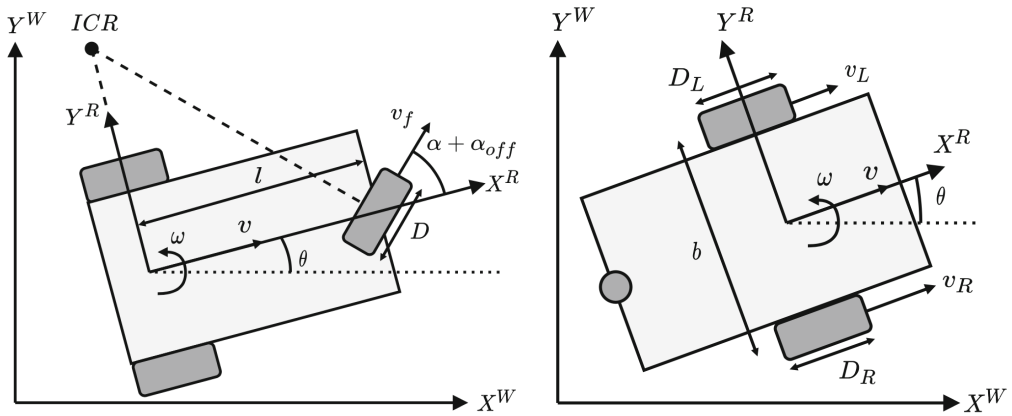
Source: Tomasi & Todt (2017)

Cecco (2002) proposed an analytic calibration included in the robot; therefore, the robot self-calibrated its parameters along the path. To understand its error, the robot used an inertial measurement sensor. Moreover, the method was deeply tied to the robot's analytic equation, and the same results in odometry quality were achieved by combining odometry with an inertial

sensor without optimization methods.

Using closed-form equations to estimate robot parameters narrow the scope of robots to which calibration methods are applied. Because of the complexity of forming analytic equations for different robots and paths, most calibration methods focus on differential and tricycle robots because their structure and movement restriction is simple and easily described as shown in Figure 10. The analytic methods also require specific experimental path format and length, as the path is also considered in the analytical equations. The initial works proposed a square to simplify translation and rotational movement analysis. In contrast, Tomasi & Todt (2017) suggested a rotational path and Maddahi *et al.* (2012) presents lines with rotation. The table 1 summarizes the calibration methods analyzed and their contribution.

Figure 10 – Model of a tricycle robot and its constructions parameters (a) and model of a differential robot and its structure parameters (b).



(a) Tricycle robot structure description, with wheels positions according to robot.

(b) Differential robot structure description, with wheels positions according to robot.

Source: Sousa *et al.* (2022)

The odometry calibration for omnidirectional robots is not explored in the literature because, with more wheels and less movement restriction, it is complex to build an analytic method to calibrate the robot's parameters. For example, in differential and tricycle robots, the length and angle of the path executed compared to the programmed length expose the wheels' size difference and issues. Additionally, the error in programmed rotations represents the wheels' positioning issue. For omnidirectional robots, there are more wheels, and it is possible to have side movements. Therefore, the error can come from a different source of the model issue; for example, it is impossible to directly convert rotation errors to wheels' model issues because they can come from the robot's side movements. Therefore, a different method to fix the kinematics modelling issue is required.

3.2 ODOMETRY OPTIMIZATION

To fix issues from more complex models, researchers proposed optimization methods. Unlike analytic calibration, the optimization methods present algorithms that reduce path-tracking errors by adjusting kinematic model parameter values. Different authors propose different algorithms and methodologies to perform experiments and optimize, and in this section, these works will be analyzed, compared and discussed.

3.2.1 Least-squares Optimization

Recent works proposed an odometry optimization using the least-squares algorithm to reduce an omnidirectional robot's final position and orientation through the kinematic parameters change (LIN *et al.*, 2019). The least-squares algorithm aims to minimize the sum of the squares of errors, and it is typically applied to regression analysis to approximate solutions. An omnidirectional robot has a multivariate linear equation between the wheels and the robot's movement because it has multiple wheels contributing to actions in X, Y, and θ. Therefore, Lin *et al.* (2019) proposed multiple linear regression using least-squares to optimize the odometry accuracy of a three-wheel omnidirectional robot.

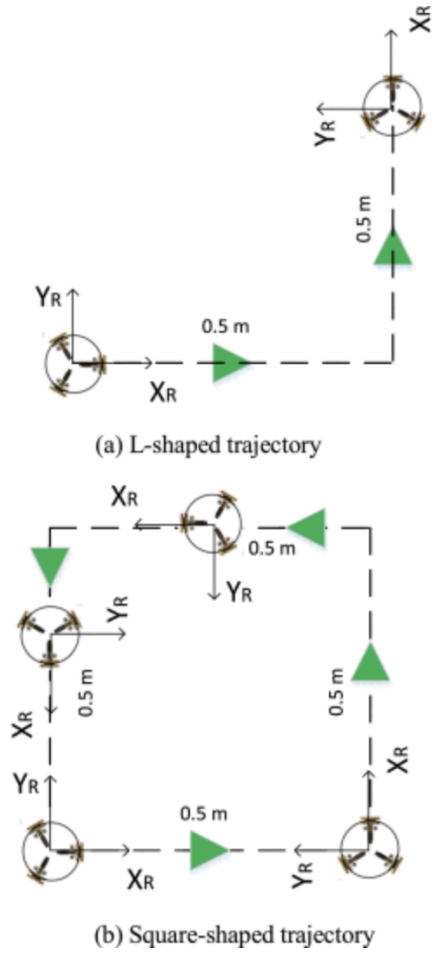
The Lin *et al.* (2019) work required ten experiments to use least-square error effectively, and the optimization was evaluated at a maximum speed of 0.8 m/s, which reduces the chance of slip errors due to low velocity. As shown in Figure 11, the experiments were conducted in paths with a maximum length of 2 m, and besides the low speed and short path, the calibration error was around 0.1 m.

The Lin *et al.* (2019) experiments used a camera in the robot to track visual landmarks in the environment and estimate the robot's ground truth position. At the same time, previous calibration methods used manual measurements between the expected and the robot's position.

There is a work proposed by Goronzy & Hellbrueck (2017) to use a non-linear weighted least-squares algorithm. The primary objective of Goronzy & Hellbrueck (2017) method is to calibrate quasi-online odometry. It requires the wheel's encoder samples and the robot's localization in the environment map. The work aims to optimize odometry for the differential robot, and the optimization reduces path tracking error to 0.2 m. The non-linear least-squares weights come from the quality of the robot's localization, which makes the work complex because it requires the robot to have a localization system.

A parameter estimation method was proposed by Kallasi *et al.* (2017). Formulated for industrial tricycle robots, the technique used a laser scanner to find the robot's position and, together with least-square linear regression, optimized the robot's odometry. The Tomasi & Todt (2017) work requires constant wheel speed; however, it does not need a fixed path, as movements come from external commands. The reported results from Kallasi *et al.* (2017) work were parameters precision and time to calibrate because the work's main goal was to estimate odometry parameters for industrial robots.

Figure 11 – Experimentation path and size proposed by Lin *et al.* (2019).



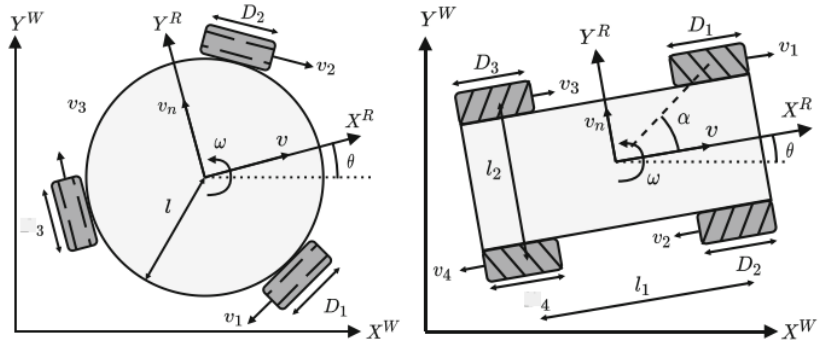
Source: Lin *et al.* (2019)

At the same time that least-square methods improved odometry's accuracy, it requires multiple linear regressions. Each linear regression outputs the best value for each coordinate and optimizes the parameters related to that coordinate. Therefore, it does not consider the correlation between parameters for different spatial coordinates. Although (Lin *et al.*, 2019) work targets an omnidirectional robot, it uses a three-wheel omnidirectional robot, which has an invertible kinematics model equation, and therefore facilitates splitting the matrices between different linear regressions.

3.2.2 OptiOdm Framework

The works analyzed previously in this section optimize or calibrate the odometry for one type of robot, and the methodology focused on the physical issue with the robot's structure. The OptiOdom is a framework proposed by Sousa *et al.* (2022) to do optimization algorithms for three different types of robots, the differential, and tricycle, as presented in Figure 10, a three-wheel omnidirectional robot, shown in Figure 12 (a), and a four Mecanum-wheel omnidirectional robot, presented in Figure 12 (b). Therefore, one of the contributions of the Sousa *et al.* (2022) work is a generic framework that optimizes different robots.

Figure 12 – Two different omnidirectional robots' odometry, with three-wheel and four Mecanum-wheel.



(a) Three Wheels (b) Four Wheels

Source: Sousa *et al.* (2022)

To optimize the robot's odometry, the OptiOdm framework uses the robot's odometry-predicted position and an external vision monitoring system, OptiTrack, to record the robot's ground truth position. With both sources of positions captured, Sousa *et al.* (2022), the methodology uses a Resilient Propagation algorithm (RProp) to optimize odometry. The Rprop is a back propagation iterative algorithm that uses the partial derivative of each kinematics model parameter to calculate its error direction. With the error direction calculated, the RProp adds a fixed value in the opposite direction to find better parameters. The RProp algorithm is known to train neural networks; however, it uses a known activation function in the neural network, while OptiOdm depends on the robot's kinematic model partial derivative.

The OptiOdm odometry optimization was evaluated across different paths, including square, circular and an arbitrary paths. The proposed procedure consists of the subsequent steps:

1. Measure the absolute position of the robot and initialize the odometry system with that position.
2. Run the robot through one of the paths. Measuring the robot's position every 0.5 m of displacement.
3. Repeat the first two steps to collect more data to reduce noise.
4. For omnidirectional robots, repeat the first steps with velocity in a different direction.
5. Perform optimization on top of odometry and visually recorded positions.
6. Replace the robot's parameters with the optimization result.

In the end, the OptiOdm reported a kinematic model improvement that achieved an odometry path accuracy of 0.1 m for differential and 0.2 m for omnidirectional robots. Although the accuracy is similar to previous works, the framework optimizes mechanical imperfections for four different robots' structures, while others focus on one type of robot. The optimization

method proposed by Sousa *et al.* (2022), the RProp, is also different from previous works, which used the least-square method. Although Sousa *et al.* (2022) framework already targets the robot's structure shown in Figure 10 and Figure 12, different robot's design requires to re-build the partial optimization derivatives from the different kinematic model.

3.2.3 Related Work Considerations

After studying the literature on Odometry calibration and optimization, the researchers propose fixing specific kinematic models, which apply to only a few robots. Therefore, any other robot's structure requires a new analytics equation based on previous works to achieve better odometry path-tracking. Besides the analytics equations, the previous research does not report the robot's velocity used, an essential variable for different applications. For example, SSL robots require navigation above 2 meters per second, which implies that odometry navigation still tracks movements with reduced inaccuracies.

The literature also differs in evaluating the odometry quality, and some researchers propose to use the difference between the path target and the odometry final position. Measuring only the last position ignores all the movement made between the initial and final position; it also creates a dependence on the path movements, as cyclic paths, such as squares, may help compensate for errors with mirrored paths.

The table 1 compares the Sousa *et al.* (2022) work with others presented in this chapter by the characteristics and results of relevant for odometry calibration and optimization through the kinematic model. The comparison contains the type of robot it covers, the experiment, the path tracking evaluation, the target of the optimization method, the optimization method, and the accuracy result reported.

Moreover, this work searches for a generic optimization method which is not dependent on the robot's structure and works with different robot speeds to support various odometry applications. This work also proposes a new evaluation method of the odometry path-tracking quality that accounts for every position reported and results in a precise quantification of the odometry quality.

Table 1 – The comparison between the most relevant odometry calibration and optimization works for the literature, analyzed by robot's type, experiment, data evaluation, the optimization target and method, and the accuracy reported.

Related Work	Robot	Experiment	Evaluation Method	Optimization Target	Optimization Method	Result
Borenstein & Feng (1996b)	Differential	Squares of 4x4 m at 0.2 m/s to avoid slips.	Difference from raw odometry final position	Wheels' mechanical parameters	Closed-formed analytical equation	Improvement of one order of maginitude between original and calibrated odoemtry
Tomasi & Todt (2017)	Differential	Rotation of 360° and 180°.	Difference from raw odometry final position	Wheels' mechanical parameters	Closed-formed analytical equations	Reduced final position error in 92%, from 285 mm
Goronzy & Hellbrueck (2017)	Differential	Simulation of line, squares and random paths.	Distance from raw odometry in multiple positions.	Kinematic model parameters.	Weighted nonlinear least squares	Simulation average error of 0.2 m, and mean position error of 0.15 m.
Kallasi *et al.* (2017)	Tricycle	Circular segments with increasing radius	Steering angle and relative motion	Comparison with human manual calibration	Least-squares with closed-form equations	Optmized in 12 minutes, previously process took hours. Parameter errors achieved less than 6 mm.
Lin *et al.* (2019)	Three-wheeled Omnidirectional	Random paths, however validated in L-shaped and square paths	Distance from raw odometry in multiple points	Kinematic model parameters	Least-squares optimization	Consistent path with postions erros around 0.1 m.
Sousa *et al.* (2022)	Differential, Tricycle and Omnidirectional	Squares, rotations and random paths	Distance from ground truth position in multiple points	Robot's mechanical parameters	RProp with partial derivatives	Maximum error of 0.1 m for differential and 0.2 m for omnidirectional.

Source: The author

4 THE ODOMETRY OPTIMIZATION METHODOLOGY

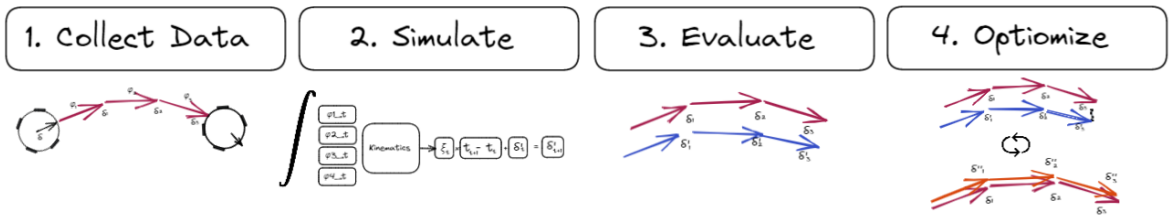
This chapter presents the proposed odometry optimization method, which aims to improve autonomous mobile robots' navigation, optimizing the odometry through the kinematics model. It proposes combining encoder wheels' velocity at a kinematic model with angular tracking from an inertial sensor to accomplish the best results. The proposed method is also computationally cheap because it maintains odometry tracking with a high sampling rate to serve as localization for autonomous navigation in short distances.

Unlike previous works, this work uses no close-formed analytic equation or partial derivatives from the specific kinematic models. Therefore, adapting it for other robots does not require re-work because this work proposes an odometry optimization based on the simulation of the robot's kinematic model and collected experimentation data to accomplish an effective generic odometry optimization. In other words, this work does not use closed-formed equations in the experimentation and optimization process. Previous studies have simulated only the robot's behaviour to perform optimization; however, this proposes a simulation that uses the robot's data to include the mechanical inaccuracies and movement issues in the generated path. Therefore, simulating different kinematic model parameters for the same robot without additional experiments is possible.

This work compares odometry against the ground truth positions, and different from previous work, it proposes to compare every odometry position generated against the robot's ground truth position. Additionally, while the previous works reported the maximum distance from odometry and ground truth position, this work presents a path-tracking metric that considers all position differences to better capture the odometry quality along the path. Because of the proposed evaluation metric and the optimization process through the original kinematic model, this work is not path specific, unlike most previous works.

The proposed methodology to optimize odometry for mobile robot navigation consists of four high-level steps. The steps diagram is presented in Figure 13, and consists of experimentation to collect the robot's data, including encoders, inertial sensor samples, and ground-truth positions. Then comes the proposed simulation that used the experiment data to simulate the odometry path from a given robot kinematic model. The simulation from the real robot's data is crucial because its input is the kinematic model parameters and experimentation data, and the output is the robot's odometry path. The third step is the evaluation metric, which calculates the quality between ground truth positions and the tracked position by odometry, simulated or not. Together, simulation and evaluation compose the odometry fitness for the optimization method that improves the kinematic model parameters based on the solution fitness.

Figure 13 – Diagram with overview steps of the method proposed. The initial step is collecting data from robots, the next step is simulating the odometry from data collected, then it is necessary to evaluate the quality of kinematics from simulate path. In the end the data collects, simulation and evaluation builds parts of the last step, the kinematics optimization.



Source: The author

Following the activities enumerated below is important to apply the proposed odometry optimization approach. Each activity is mapped to one of the high-level steps shown in Figure 13. The details of optimizing mobile robots' odometry are discussed later in this chapter.

1. Move the robot in any path, collecting wheels' speed, inertial sensor angles and ground-truth positions.

2. Build the simulation to receive the robot's kinematics and experiment data and output the robot's odometry path, including mechanical issues.

3. Build an odometry evaluator that compares odometry generated, from a robot or simulation, against the ground truth positions.

4. Execute optimization to improve kinematic model parameters, using the simulation from experiments data combined with evaluation method as solution fitness. The kinematics model parameters are the optimization target.

Sections below detail each step of the proposed optimization strategy. It is essential to highlight that this chapter will present the methodology, while Chapter 5 presents implementation details used to evaluate the method quality and the Chapter 6 will deliver the results to compare effectiveness with the previous works.

4.1 EXPERIMENTATION DATA COLLECTION

Data collection is the first and crucial step because kinematics errors will be present in the data, enabling optimization of the robot's kinematic model parameters according to its mechanical issues. This work proposes to use wheels and a IMU to evaluate robot movement and construct the robot's odometry path. Moreover, it is necessary to collect the wheels' speed and IMU readings used in the robot's odometry to correct, simulate and evaluate its path, including

model physical inaccuracies. As the data required from the IMU is the angular movement and the gyroscope is the relevant sensor used, this work references gyroscope and IMU interchangeably.

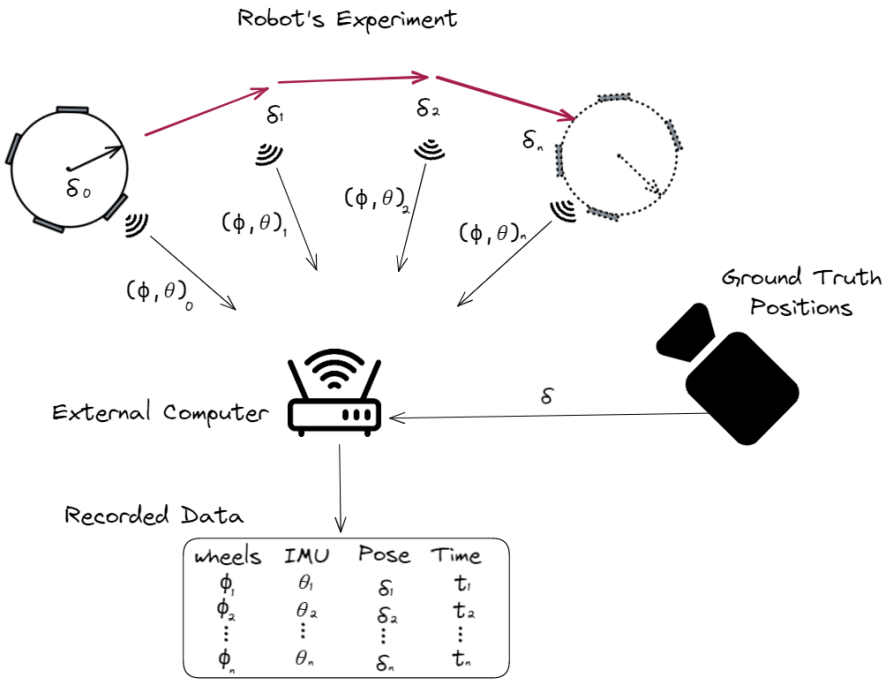
The proposed work targets autonomous mobile robots, which move freely around the environment. However, the experiments to collect data do not require long distances; they require ground truth position tracking to compare robot odometry positions. Therefore, indoor environments accommodate more options for position tracking, such as external vision. Besides position tracking, indoor scenarios also facilitate data collection. For high-rate sampling, as used in odometry, the Cavalcanti *et al.* (2022) work shows that WLAN is recommended to collect data wirelessly from the robot.

The Figure 14 presents the diagram to collect data in robot experiments. There are three main components: robot's movement, ground truth position capture, and an external computer with a radio receiver. While the robot moves in the environment, it transmits the wheels' speed (ϕ) and gyroscope angle movement (θ) to an external computer, where data will be recorded together with the robot's ground-truth (δ) position in the correct order and timestamp (t). At the end of the experiment, the computer produces the experiment dataset.

Moreover, data collection needs gyroscope-filtered samples from the robot. The gyroscope sensor readings output its body's rotation velocity; however, as described in Chapter 2, the sensor has a residual reading value, usually small, but similar to kinematic model issues, its accumulation leads to tracking errors. Therefore, the gyroscope-filtered readings are required for the proposed experiments in this work. The robot stationary can measure the noise to filter residual values subtracting it from every reading. A suggested implementation is presented at Chapter 5.

The robot's position ground truth can come from the robot's internal sensor (GANGANATH & LEUNG, 2012A), such as onboard cameras and laser scanners. However, most of the robots does not have these sensors and aim to optimize odometry to improve navigation quality without additional sensors. An external ground-truth position tracking method is recommended for these cases, such as computer vision pattern localization proposed by Zickler *et al.* (2010). An external position tracking system requires synchronizing the time the robot's data and ground-truth position are collected.

Figure 14 – Diagram with overview components to collect robot's data. First, it is necessary to run an experiment with robot moving, during the movement send robot's data at each odometry iteration with wheels' speed (ϕ) and angular movement from inertial sensor(θ). An external computer, with a data receiver, collects robot's data and aggregate with ground truth positions (δ), for each iteration (t).



Source: The author

Studies to optimize motor control using Nordic Radio Frequency modules proposed rates over 30 samples per second (ARAÚJO *et al.*, 2022). Moreover, the same communication technology is recommended to achieve a higher sampling rate, increasing the samples collected. The sampling rate also impacts the quality of the simulation and evaluation proposed, in result it improves the odometry optimization process.

Each wheel's encoder calculates the wheel speed in radians per second, and gyroscopes produces the robot's relative angle integrating angular velocity over time. Besides the wheels' speed and angular movement, the ground truth captured will quantify kinematic model quality and issues. Moreover, independently of the robot which the optimization is required, the data to collect are the wheels' speed, gyroscope angle, and ground truth positions.

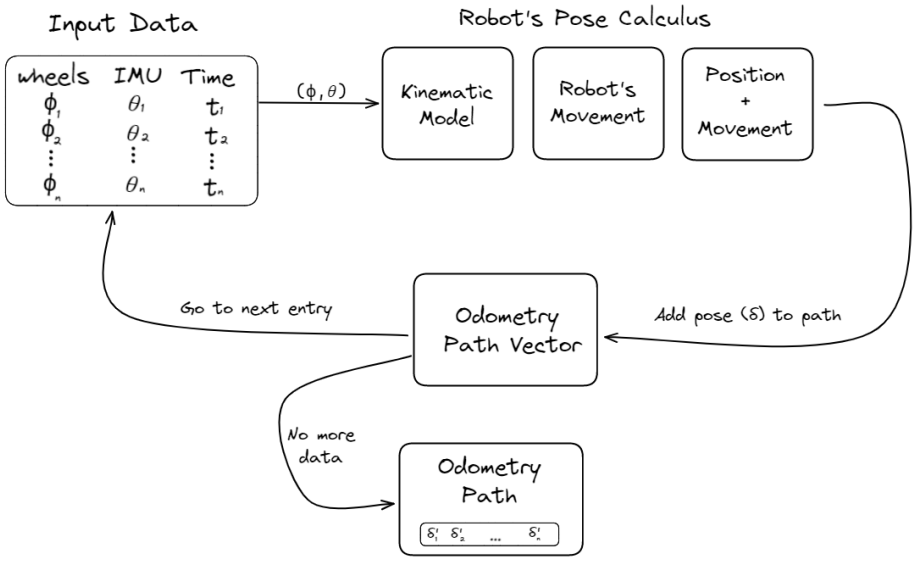
4.2 SIMULATING ODOMETRY PATH

After collecting the wheels' speed and robot's angular movement, it is possible to simulate the odometry-based path using the kinematic model to include its inaccuracies. This simulation is essential to evaluate different kinematic models, the original one with mechanical issues and others, such as optimized models.

The Figure 15 illustrates the process to simulate the robot's odometry path from the

collected data. First, the collected data is inputted, and then, for each experiment sample collected, the simulation will calculate the odometry position and orientation (δ'). The pose calculus involves applying a kinematic model to the wheels' speed (ϕ) and converting it to movement using elapsed time. The angular movement (θ) from the IMU gyroscope comes in the pose calculus to aid with robot's orientation tracking. The robot position and orientation movement from each data entry, is added to the previous position and results in the the odometry path.

Figure 15 – Diagram with overview steps of simulation process. The data collected is the input to create the simulated odometry path. To do so, each data entry has robot's data, which is converted to robot's movement. The movement is then added to the previous pose. All poses build the expected odometry path. In the figure, ϕ represents wheels' speed, θ represents IMU angular movement, δ' the robot's expected position and orientation, and t the timestamp.



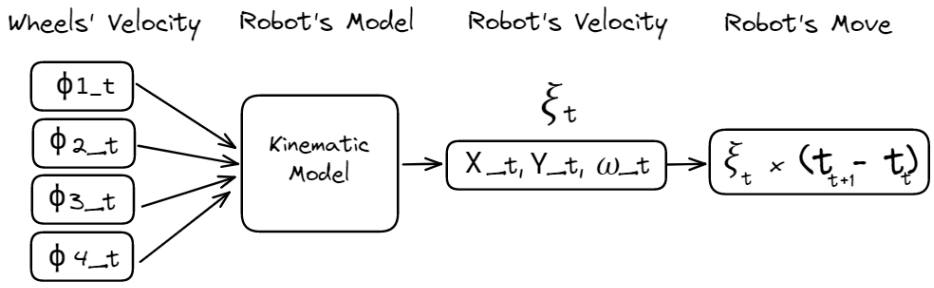
Source: The author

The simulation's core is the kinematic model's calculus from the robot's experiment data, which converts wheels' velocity into robot's velocity, as shown in Figure 16. After the robot's speed is calculated, it is possible to estimate its position, converting speed in displacement and adding the movement to its previous location.

The kinematics equation used in the simulation core is presented in Equation (4.1). For a four-wheeled omnidirectional robot the kinematics equation includes the matrices $R(\theta)$, as shown in Equation (4.2), the J_{1f}^{-1}, which is the inverse of J_{1f}, presented at Equation (4.3), the J_2, at Equation (4.4) and wheel's speed matrix, $\dot{\phi}$, shown in Equation (4.5).

The kinematic model matrices are designed for the robot to navigate, and same matrices are used in this method to optimize odometry though kinematic model parameters. As shown by the designed matrices, the model receives the wheel's speed and outputs the robot's speed; therefore, after the matrices multiplication, robot's speed is integrated over time to build odometry

Figure 16 – Diagram with simulation core, which from experiment collected data, calculates the robot's movement using wheels' velocity, kinematic model and elapsed time.



Source: The author

paths. This calculation is expected in robot's odometry modules and applied in the simulation process to mirror the issues which optimization aims to fix.

$$\dot{\xi}_I = R(\theta)^{-1} J_{1f}^{-1} J_2 \dot{\phi} \tag{4.1}$$

$$R(\theta) = \begin{pmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.2}$$

$$J_{1f} = \begin{pmatrix} -sin(\alpha_1^\circ) & cos(\alpha_1^\circ) & l \\ -sin(\alpha_2^\circ) & -cos(\alpha_2^\circ) & l \\ sin(\alpha_3^\circ) & -cos(\alpha_3^\circ) & l \\ sin(\alpha_4^\circ) & cos(\alpha_4^\circ) & l \end{pmatrix} \tag{4.3}$$

$$J_2 = \begin{pmatrix} r & 0 & 0 & 0 \\ 0 & r & 0 & 0 \\ 0 & 0 & r & 0 \\ 0 & 0 & 0 & r \end{pmatrix} \tag{4.4}$$

$$\dot{\phi} = \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \end{pmatrix} \tag{4.5}$$

The Algorithm 1 presents detailed steps to build the simulation. First, the simulation algorithm loads the experiment data, records the robot's initial position, and then iterates over the data entries. At each iteration, the kinematic model is multiplied by the velocity of the wheels

from the experiment entry to generate the robot's speed. After finding the robot's velocity, it is converted into its movement using the elapsed time from current and subsequent data entry. The difference between the current and the following inertial sensor sample defines the robot's rotation movement. The final step at the iteration is adding the calculated move to the current position and recording it at the path vector. At the end of the data entries iteration, the path vector has the odometry path from the robot according to the kinematic model used in the simulation.

It is essential to highlight that the odometry path outputted from the simulation considers the kinematic model issues and wheel slips from the robot, as the data collected reflects all the information from the robot's interaction with the surrounding environment.

Algorithm 1: Odometry simulation from data collected at experimentation.

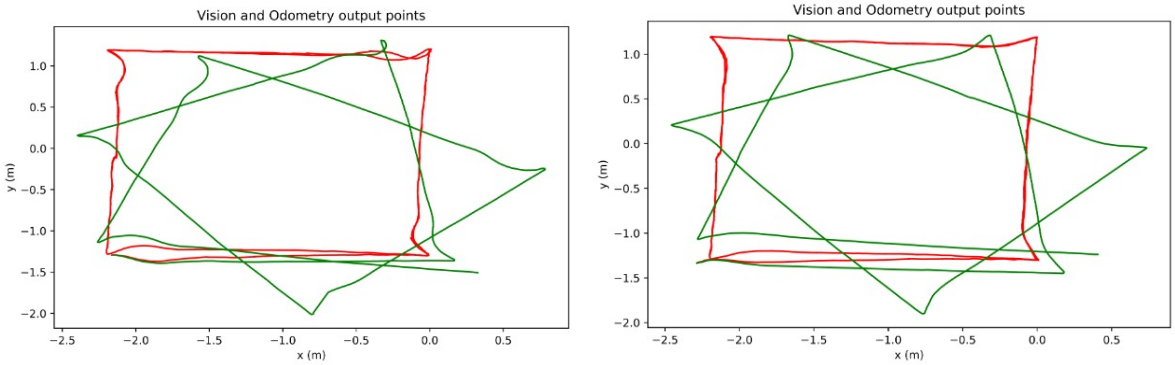
```
1  Load logs from data collection;
2  path[0] = first position of ground truth;
3  for i = 1; i < len(logs)-1; i++ do
4      //Convert wheels speed to robot speed;
5      wheels_speed = (logs[i].wheels+logs[i+1].wheels)/2;
6      robot_speed = Kinematics.convert(wheels_speed);
7      //Covert speed to movement, using logs elapsed time;
8      movement[x, y] = robot_speed * (logs[i+1].time - logs[i].time);
9      movement[ω] = (logs[i+1].imu.angle - logs[i].imu.angle);
10     //Add new point to the odometry path;
11     path[i+1] = path[i] + movement;
```

It is possible to see in the Algorithm 1 that movement calculus uses wheels velocity average from current and next data entries. This velocity average helps to capture the velocity between samples interval, and it is recommended to build an accurate odometry path Siegwart *et al.* (2011). This technique also serves to overcome data entry loss because wireless communication may lose samples, and using velocity average and time interval absolves loss. However, sample loss may harm generated path quality, and the high data collection rate minimizes impact.

4.3 EVALUATING ODOMETRY PATH

After the data collection and simulation, it is necessary to define how to evaluate the odometry-based path. As shown in Figure 17, it is possible to analyze ground-truth and odometry paths visually; however, it is impossible to quantify how good an odometry-based path is, compared to the path the robot executes. It is also impossible to compare different experiments, as shown in Figure 18a and Figure 18b, because it is not visually apparent which path is the best.

Figure 17 – Two examples of robot's path tracking with two tracking sources each. Although both examples perform the same path, it is not possible to visually qualify which example has better tracking..



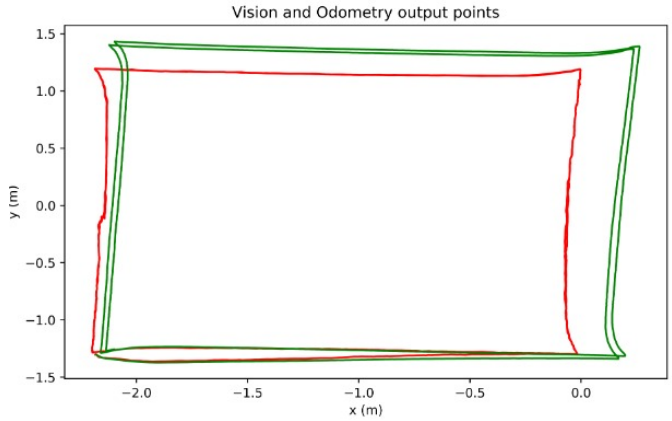
(a) Robot's path tracking from two different sources, in red tracking from an external camera and in green tracking from robot's odometry.

(b) Robot's path tracking from two different sources, in red tracking from an external camera and in green tracking from robot's odometry.

Source: The author

The first solution to measure the path quality was introduced by Borenstein & Feng (1996b). It is the difference between the robot's path end pose, which combines position and orientation, from the expected end pose. With a similar end position in the robot path and at the odometry path, the better the path tracking and its kinematic parameters were. Sousa *et al.* (2022), on the other hand, relied on the maximum difference between odometry and the robot's ground truth position; again, the quality was measured by one position difference.

Figure 18 – Experiment of robot's path tracking from two different sources, in red tracking from an external camera and in green tracking from robot's odometry.



Source: The author

It is possible to see a difference in the quality of the odometry path compared with the ground truth path, comparing Figure 17 with Figure 18. However, the final and maximum distance from these experimenters, as proposed by Borenstein & Feng (1996b) and Sousa *et al.*

(2022), does not capture all their differences.

This work proposes a measure that quantifies the path over all captured data points, which uncovers errors in the middle or throughout the odometry path. Therefore, this work suggests capturing the euclidean distance from the robot's odometry path and ground truth path and applying it to RMSE. As described in Equation (4.6), the RMSE takes accountability for all entries from the experiment and simulation. It also weights all location equality; therefore, every odometry error is considered, not only initial or maximum errors. In the Equation (4.6) n is the total number of samples, δ_i is the robot's ground truth position for the i^{th} sample, $\hat{\delta}_i$ is the robot's odometry predicted position for the i^{th} sample.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\delta_i - \hat{\delta}_i)^2} \tag{4.6}$$

Applying the RMSE to the robot's path examples presented previously, it is possible to compare their quality, as shown in Table 2. The robot's ground truth position as δ_i and the robot's odometry position as $\hat{\delta}_i$, revealed a RMSE for path example in Figure 18a of 1.24, and for Figure 18b example is resulted in 1.23 of RMSE. These results make it possible to compare both paths and conclude that the second experiment has better odometry than the first. Applying RMSE for the example in Figure 18 outputs 0.09, confirming the visual quality of the latest robot's path example and usability of the Root Mean Square Error (RMSE) metric.

Table 2 – RMSE comparison between examples robot's path.

Robot's Path Example	**Root Mean Square Error**
Figure 18a	1.24
Figure 18b	1.23
Figure 18	0.09

Source: The author

4.4 OPTIMIZING ODOMETRY KINEMATIC MODEL

The odometry calculates the robot's movement, integrating the wheels' speed over time, applied to the kinematic model and the gyroscope angular movement. The data come from samples collected at the experiment, where the robot moves around the environment. Although robots have their own embedded odometry, the wheel's speed and gyroscope samples are necessary to simulate the odometry with different parameters without using the robot. Therefore, the simulation allows the usage of different kinematic models without running experiments with the robot. The simulation mimics the robot's embedded odometry; however, it may use other kinematics parameters which achieve different odometry paths from the same data. As discussed before, the robot's kinematic model parameters have mechanical inaccuracies. Therefore, the optimization evaluates and optimizes the kinematic model parameters by comparing the simulated odometry path with the ground-truth path, as both came from the same robot's movement.

Optimization happens by changing the model parameters and producing a different simulated path compared to the ground-truth course.

This work proposes the optimization algorithm of the Particle Swarm Optimization (PSO) because it produces better and faster solutions according to previous research (KECSKÉS *et al.*, 2013). The work from Cao *et al.* (2022) also applied PSO for robotics parameter optimization and reported a successful optimization process, which gives more insights and background of the using PSO to optimize kinematic parameters.

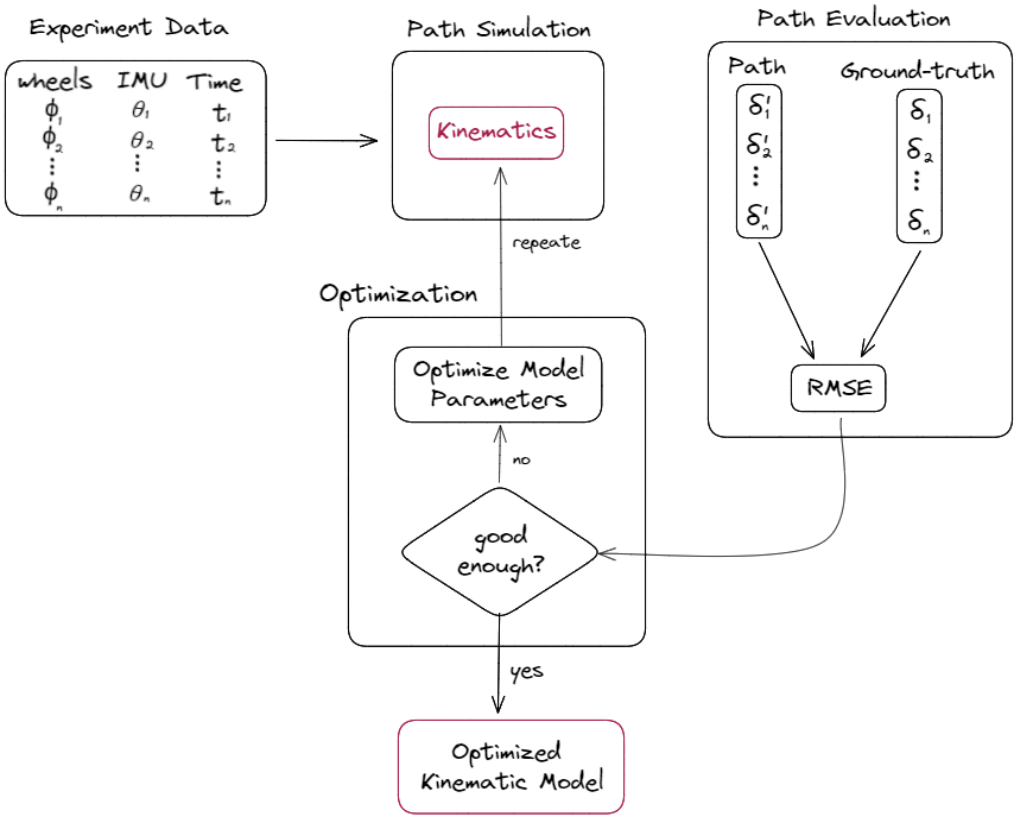
As shown in Figure 19, the optimization process relies on iterations of kinematic parameter enhancement, odometry simulation, and its evaluation. The optimization aims to change the kinematics parameters to achieve a better evaluation. In the case of the PSO algorithm proposed in this work, there is a population of solutions, where each key is called an individual and represents the kinematics parameters. The algorithm changes each solution parameter in the direction of its best set of parameters found and the best solution at the population, as presented in Chapter 2. Evaluation occurs again after the whole population changes, and if a solution is satisfactory, the optimized set of parameters that build the kinematics is found. The evaluation result is called fitness, and if fitness is not good, the algorithms iterate over the new population if no solution is satisfactory.

Therefore, it is necessary to define the set of parameters, to build the optimization and the evaluation that quantifies solution quality. In this work, the evaluation requires the experiment data, the simulation and the RMSE method. Building the PSO algorithm is possible with all these inputs.

The simulation presented in Algorithm 1 uses Equation (2.7). At the equation, $\dot{\xi}_I$ is the robot's speed, $R(\theta)^{-1}$ is responsible for covert the robot's local coordinates to global, J_{1f}^{-1} and J_2 considers robot's structure description, and $\dot{\phi}$ is the wheel speed. From all these matrices, only J_{1f}^{-1} and J_2 depends on robot's structure. Then, by changing these two matrices, the odometry path built by the robot or simulation will also change. Ultimately, these parameters are the optimization target as they come from the model and change the odometry result.

The J_{1f}^{-1} matrix comes from the pseudo inverse J_{1f} matrix. This happens because the kinematic matrix actuates in three axis, X, Y, and θ; however, some mobile autonomous robots have more than three actuators, which creates a non-square kinematic matrix like J_{1f} presented in Equation (4.3) (SIEGWART *et al.*, 2011). It brings two challenges, first eliminates most of the previous works that proposed analytic solutions; second, it requires finding the inverse matrix using the pseudo-inverse method as described by Adi Ben-Israel (2003). The J_{1f}^{-1} maintains the number of values from J_{1f}, which keeps the same number of parameters in the optimization process. The J_2 is presented at Equation (2.4) and depends on the wheel's radius. Therefore, for the robot presented at Figure 3, both matrix's values will result in the parameters to optimize. And it has 12 parameters from J_{1f}^{-1} and one from the four-wheel omnidirectional robot. In the end, the four-wheel omnidirectional robot presented in Figure 3 has 13 kinematic parameters to be optimized.

Figure 19 – Diagram with overview steps of the optimization proposed. The initial step is input experiment data in the simulation, where the kinematic model is used to predict odometry path. Then the RMSE of predict path and ground-truth is extracted. If predict path is not into an acceptable margin of error to the ground-truth, the kinematics is optimized and the process repeats. The margin of error is defined based on the robot's and application requirements.



Source: The author

This work proposes the PSO algorithm to optimize multiple real values because of its efficiency and low cost, presented at Chapter 2. The set of 13 parameters for the given robot represents an individual solution. A good solution means a group of 13 parameters that the simulated odometry and the robot's odometry path approximates the ground-truth way.

The proposed evaluator, presented in Algorithm 2, receives the 13 kinematics parameters, 12 from J_{1f}^{-1} and the wheel radius used at J_2, and one experiment file with the wheels' velocity and IMU data. The evaluator will use the 13 parameters to simulate the odometry path on top of the experiment data, as defined in Algorithm 2. The path is then used to calculate the RMSE error against the ground truth from the experiment file. The RMSE outputs the solution error and returns to the optimization as the quality of the set of parameters.

The evaluator is the core of the optimization and change from application to application because it determines whether a solution is good enough or needs to evolve. The quality of the solution also reveals the best solution in the population, which influences on parameter's

Algorithm 2: Path evaluator used in the optimization to quantify a better solution.

```
path_error(experiment, params):
    # Simualting odometry path
    iJ1 = [[params[0], params[1], params[2], params[3]],
           [params[4], params[5], params[6], params[7]],
           [params[8], params[9], params[10], params[11]]]
    wheelRadius = params[12]
    simulated_path = Odometry(experiment, iJ1 , wheelRadius).simulate_path()
    # Returing error of the solution path
    return Error(experiment.ground_truth_path(), simulated_path).rmse()
```

direction of update, and the best individual to return as an optimization result. The quality of each individual in the population changes based on its position and the velocity that points to personal and best solutions, as shown in Equation (4.7), where $x_i(t+1)$ is the next particle position based on its current position and velocity change.

The velocity is defined according to the Equation (4.8), where i refers to the particle and j to the algorithm interaction. The particle velocity considers current velocity as $v_{ij}(t)$, the module $(r_{1j}(t))$ and direction $([y_{ij}(t) - x_{ij}(t)])$ to the best particle solution, and the module $(r_{2j}(t))$ and direction $([y_j(t) - x_{ij}(t)])$ to the best solution in the population. The Equation (4.8) requires three hyperparameters to scale each component of the velocity equation, and these hyperparameters are C_1, C_2 and W. Moreover, these three hyperparameters, together with the population size, iterations limit and 13 dimensions, build the PSO optimization.

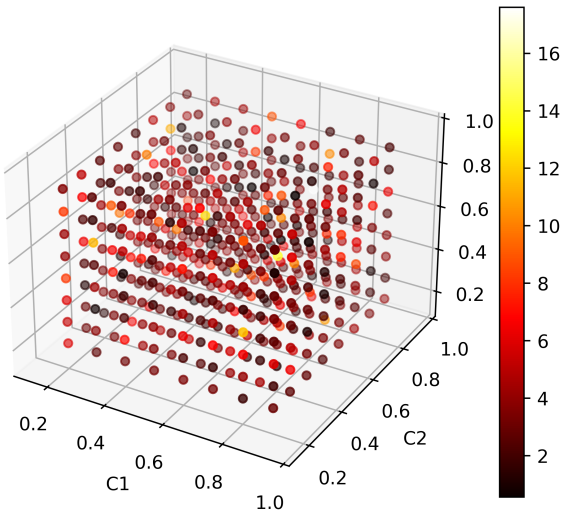
$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{4.7}$$

$$v_{ij}(t+1) = w \times v_{ij}(t) + c_1 \times r_{1j}(t) \times [y_{ij}(t) - x_{ij}(t)] + c_2 \times r_{2j}(t) \times [y_j(t) - x_{ij}(t)] \tag{4.8}$$

The C_1 controls how much a particle will weigh its best solution direction, the C_2 weights the best global solution direction, and W controls the particles' inertia; then, it weights the tendency to maintain the same vector orientation. A grid-search experiment was made to define the hyperparameters. It used robot experiment data and kinematic parameters and compared the set of hyperparameters that better optimized the kinematic for the given experiment. As shown in Figure 20, searching for these three parameters is not straightforward because no combination of hyperparameters surpasses others in the interval from 0.2 to 0.9. The population size for the grid search was 20 to optimize experiment time.

There is the size of the population to define and the number of iterations. The tests presented at Figure 21 converged to the best solution around 5,000 iterations, so iterating more was unnecessary. The size of the population helps to define how much the optimization will

Figure 20 – Grid-search of C_1, C_2 and W hyperparameters for PSO, using robot's experiment to optimize odometry evaluated by RMSE.



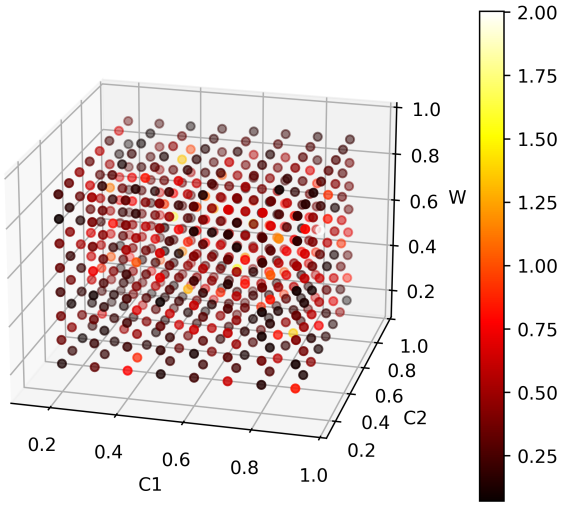
Source: The author

explore different solutions; however, for odometry optimization, there is already a candidate for the kinematic parameters, the original one. As shown in Figure 21, a new grid search experiment, with the initial population as the robot's actual kinematic parameters, varying with a limit of 0.2, improved significantly. Without the initial parameters' setup, the first grid search had a mean of 3.82. In contrast, with the initial parameters, the second experiment achieved a mean of 0.41, proving that the initialization is crucial for the overall solution, no matter the parameters.

This work proposes to use C_1, C_2, and W as 0.5, 0.3, and 0.9, respectively, as this combination was one of the best set of parameters for both experiments, presented in Figure 20 and Figure 21. The maximum number of iterations used was 5,000 (five thousand) to guarantee exploitation and reduce experiment time. For population size, this work suggests 20 because Piotrowski *et al.* (2020) work suggests it is the minimum population size for unimodal problems; therefore, it is sufficient to explore the solutions space and reduce optimization time. For each parameter's upper and lower limit, the recommended value is 0.2, as experiments show satisfactory results; additionally, parameters represent measuring in meters, like a wheel's radius. Therefore, 0.2 meter represents a broad modelling error. Suggested parameters are shown in Table 3; with these parameters, data, simulation, and evaluation, it is possible to run the PSO to optimize the robot's odometry.

Different experiments showed that hyperparameter variations might apply to other robots, such as reducing the initialization limit for small robots or increasing the number of dimensions if the kinematic model produces more than 13 parameters, in case the robot has more actuators and produces a bigger J_{1f}^{-1} and J_2 matrices. It is also possible to reduce the number of iterations with less complex robot's kinematic models, such as differential robots.

Figure 21 – Grid-search of C_1, C_2 and W hyperparameters for PSO, using robot's experiment to optimize odometry evaluated by RMSE, and initial population around the original kinematic parameters.



Source: The author

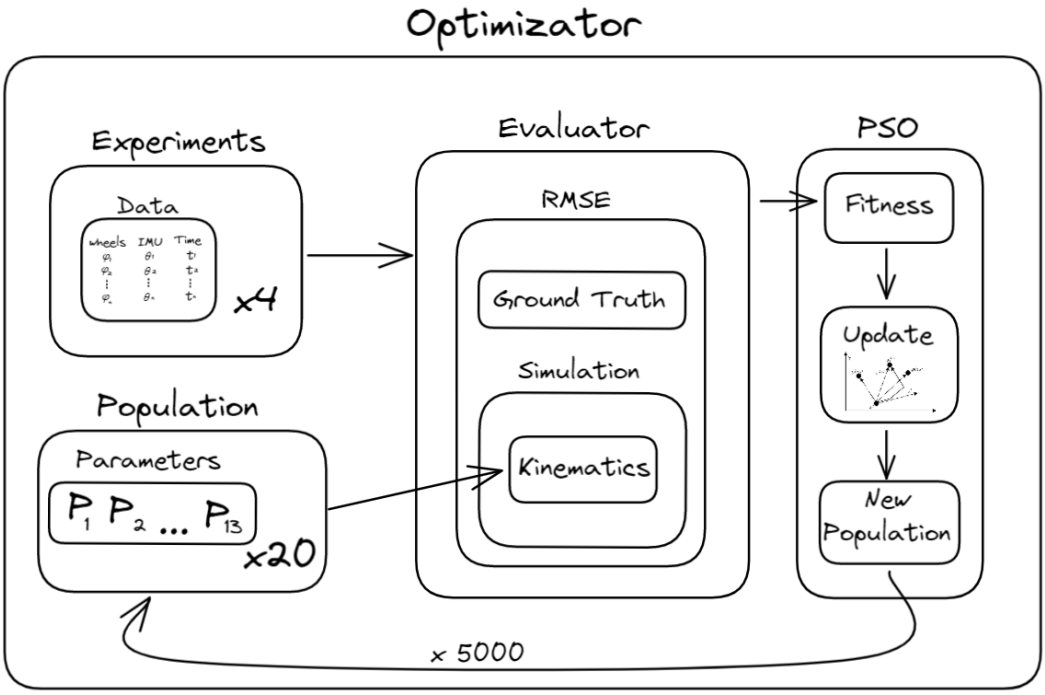
Table 3 – Proposed PSO parameters.

Parameter	**Value**
Problem dimension	13
Population size	20
Initial parameters	kinematic $\pm rand(0, 0.2)$
Iterations	5000
C_1	0.5
C_2	0.3
W	0.9

Source: The author

The complete block diagram of the optimization process is shown at Figure 22. It considers four different experiments to evaluate each solution. The population has 20 particles, each going into an odometry path simulation. The evaluator consumes the experiment wheels' speed to simulate a path using parameters from a particle. The evaluator also calculates the RMSE between the generated path and ground truth. With each particle evaluation, the PSO updates the population by changing each solution parameter in the vector sum direction between the current solution velocity, the particle's best solution, and the whole population's best solution. The new population goes again for evaluation and keeps iterating until it reaches 5,000 iterations. After all iterations, the best solution found represents the best kinematic model parameters for the robot's odometry path tracking.

Figure 22 – Diagram of the proposed optimizator using PSO, the evaluator and the simulator proposed.



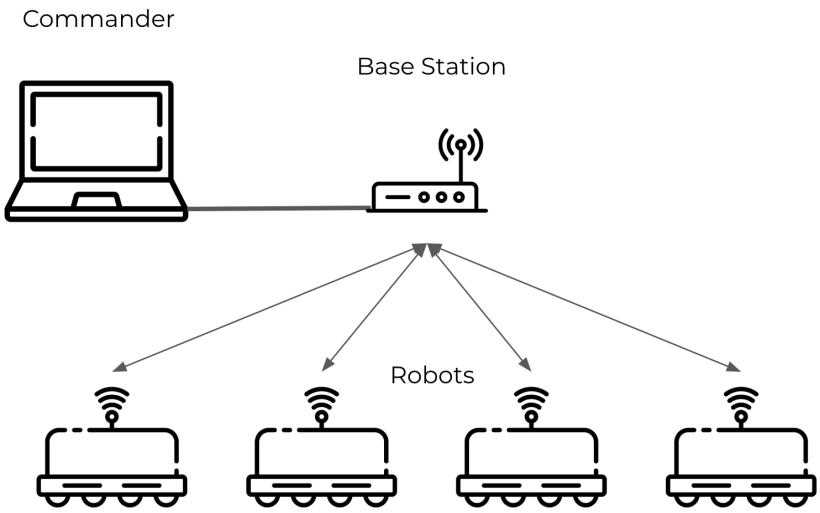
Source: The author

5 ODOMETRY OPTIMIZATION IMPLEMENTATION

The proposed method requires experimentation in the robot's environment; therefore, it was necessary to collect data, perform the optimization process and then evaluate the results in the real robot. This chapter details the robots in which odometry was optimized, the infrastructure required, experiments made, and the implementation details necessary.

The robotics environment chosen to optimize was mobile soccer robots, which require precision to navigate in a dynamic game where multiple robots and a ball are moving. The soccer robots used in this work's evaluation are from RoboCup soccer competitions Kitano *et al.* (1997), where there are different soccer categories, and the robot's navigation performance and skills matter to the result of the game. One of RoboCup's most dynamic soccer competitions is the SSL. Small Size League Technical Committee (2022) presents the category's rules, which the match happens with teams of 11 versus 11 robots or 6 versus 6. Each soccer robotics team is custom designed and made by each university, which requires students to build the whole robotics system.

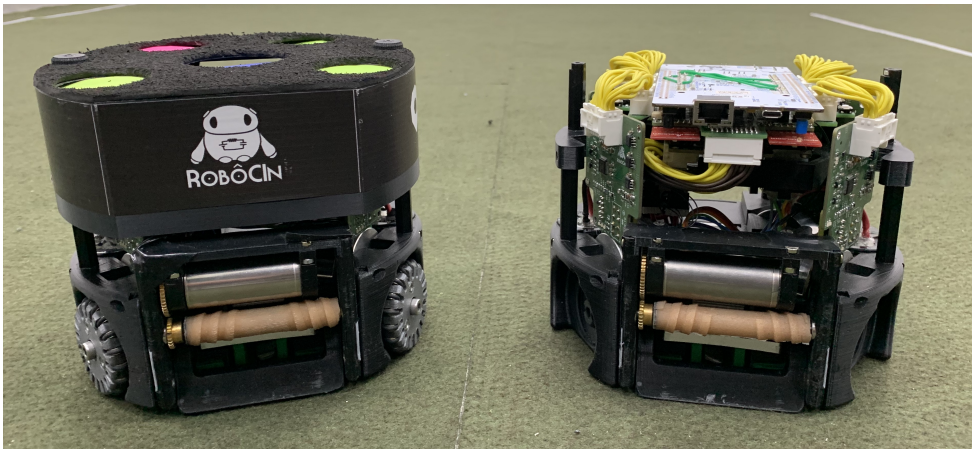
Figure 23 – An overview of Small Size League (SSL) robotics competition, where robot's plays in a filed, acting from external commands. Each team has its off-field computer, that uses cameras to localize robot's, compute the strategic movement for the team's robots and send commands to them.



Source: The author

As shown in the Figure 23, the SSL competition has an infrastructure to process the robot's position and requires wireless communication with robots to send commands. The dynamic movements, together with this infrastructure, turn SSL into an exciting environment to run the experiment, collect data and evaluate the kinematics optimization in the robot's gameplay.

Figure 24 – RobôCIn's Small Size League (SSL) robot



Source: The author

5.1 ROBOT

The robot used in the experiments, shown at Figure 24, is an omnidirectional robot with four wheels. On the left is the robot fully assembled, and on the right side is the robot without external protection and wheels.

The RobôCIn SSL robot is a cylinder with 180 cm of diameter and 15 cm of height. It has a front cut to facilitate the robot handling the ball, an orange golf ball with a 4.27 cm diameter. The ball is orange to facilitate its localization.

In the SSL soccer game, cameras are on top of the field to capture images and deliver them to a computer, in which the ssl-vision software tracks robots and the ball's position and position and orientation at the field (ZICKLER *et al.*, 2010). In Figure 26a it is possible to see that robot has a colour pattern on the top of the robot. Each pattern identifies one robot, and the colour in the center identifies if it is the blue or yellow team.

With object poses in the field, the next step is strategically controlling the robots to score the adversary's goal and defend its own goal. This task requires algorithms to plan each robot's task, build the path for each robot to its goal, and navigate it until the objective. To command the robots, teams send wireless commands. The RobôCIn robot uses a Nordic Radio Frequency module, which proved to have low power consumption and a high rate of data exchange (CAVALCANTI *et al.*, 2022).

The SSL robots are fast and can achieve over $3m/s$. Usually, teams send the robot's desired speed to control them; however, with the robot's speed increasing year over year, it started to require more precise and fast reaction navigation. Moreover, to achieve it, teams propose sending the robot's desired position and letting it navigate by its internal odometry. The internal odometry has a higher update rate, and lower delay, as it does not depend on image capture and processing to build each movement step. On the other hand, it requires odometry with a proper kinematics model.

5.1.1 Kinematics Parameters

The kinematics aims to convert the wheels' speed into the robot's speed, as shown in Equation (5.1). This equation requires some matrix, the $R(\theta)^{-1}$, which is the inverse of the Equation (2.6), presented at Equation (5.2). The J_{1f}^{-1} matrix comes from the inverse of the wheel's equation, the J_{1f}. As RobôCIn robot structure has four omnidirectional wheels at a cylinder robot, the robot's J_{1f} matrix has the same format as Equation (2.3), depending on the wheels' angle to the robot X axis and the distance from the wheel to the robot's center. The last matrix is the J_2, presented at Equation (2.4), it depends on the wheel's radius.

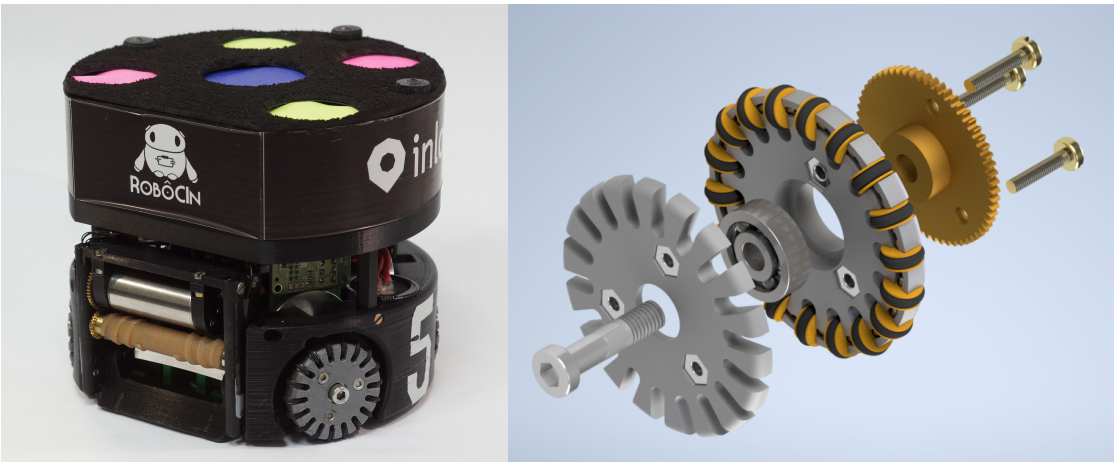
$$\dot{\xi}_I = R(\theta)^{-1} J_{1f}^{-1} J_2 \dot{\phi} \tag{5.1}$$

$$R(\theta)^{-1} = \begin{pmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5.2}$$

From RobôCIn robot 3D model, it is possible to acquire all required parameters to build matrix J_{1f} and J_2. As shown in Figure 33, the robot has omnidirectional wheels with rollers orthogonal to the wheels' rotation plane. The wheel RobôCIn, shown in Figure 33, has 18 rollers and 2.48cm of radius. Applying the wheel construction to the wheel equation, presented at Equation (2.1), it becomes Equation (5.3), because $\gamma = 0\circ$ and $\beta = 0\circ$. The signal of $sin(\alpha)$, $cos(\alpha)$, and l, may change according to the wheel contribution for the robot's movement. For example, if a wheel in forward rotation makes a robot walks backwards, the $sin(\alpha)$ is the opposite. If the same movement makes a robot walk positively in its Y axis, the $cos(\alpha)$ is positive. l depends on the robot's rotation; if the robot turns counterclockwise, the l is positive.

$$\begin{pmatrix} sin(\alpha) & -cos(\alpha) & (-l) \end{pmatrix} R(\theta)\dot{\xi}_I - r\dot{\phi} = 0 \tag{5.3}$$

Figure 25 – The first figure is the RobôCIn SSL robot with its lateral wheel in focus. The second is the exploded views of the omnidirectional wheel.



(a) Angular view of RobôCIn SSL omnidirectional robot.

(b) Exploded view of robot omnidirectional wheel assembly.

Source: The author

The kinematics structure definition involves the wheel's placement, movement direction, and wheel's type. As shown in Figure 26, the RobôCIn robot has two front wheels with 60◦ opening from the robot's X axis and 45◦ for the back wheels. The distance from the wheel to the center, the l, has $8.32cm$. Inserting these parameters into the J_{1f} matrix, the Equation (5.4) is found.

$$J_{1f} = \begin{pmatrix} -sin(60°) & cos(60°) & l \\ -sin(45°) & -cos(45°) & l \\ sin(45°) & -cos(45°) & l \\ sin(60°) & cos(60°) & l \end{pmatrix} \tag{5.4}$$

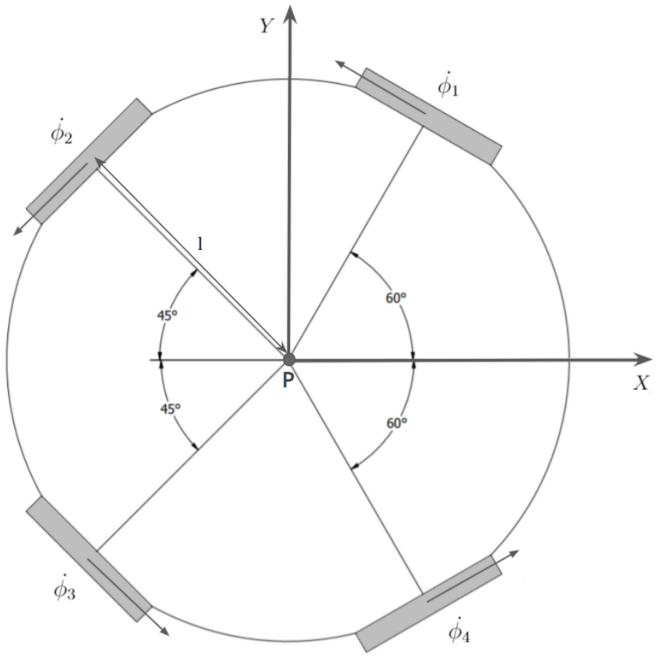
Calculating the inverse of Equation (5.4), using the pseudo-inverse method Adi Ben-Israel (2003), the results are Equation (5.5). Moreover, applying the wheel radius to Equation (2.4), the Equation (5.6) is defined.

$$J_{1f}^{-1} = \begin{pmatrix} -0,346410 & -0,282843 & 0,282843 & 0,346410 \\ 0,414214 & -0,414216 & -0,414214 & 0,414214 \\ 3,615966 & 2,556874 & 2,556874 & 3,615966 \end{pmatrix} \tag{5.5}$$

$$J_2 = \begin{pmatrix} 0.0248 & 0 & 0 & 0 \\ 0 & 0.0248 & 0 & 0 \\ 0 & 0 & 0.0248 & 0 \\ 0 & 0 & 0 & 0.0248 \end{pmatrix} \tag{5.6}$$

With the kinematics matrices defined for the RobôCIn robot, it is possible to calculate

Figure 26 – The description of RobôCIn omnidirectional kinematics structure. The robot has four Sweden wheels, and each wheel has l distance from the robot's center. The wheel's forward rotation, ϕ_1, ϕ_2, ϕ_3, ϕ_4 respectively, rotates the robot in counterclockwise. The last parameter is wheel's angle, wheels have α_3 degrees from X axis to front wheels' plane, and α_4 degrees at back wheels.



Source: The author

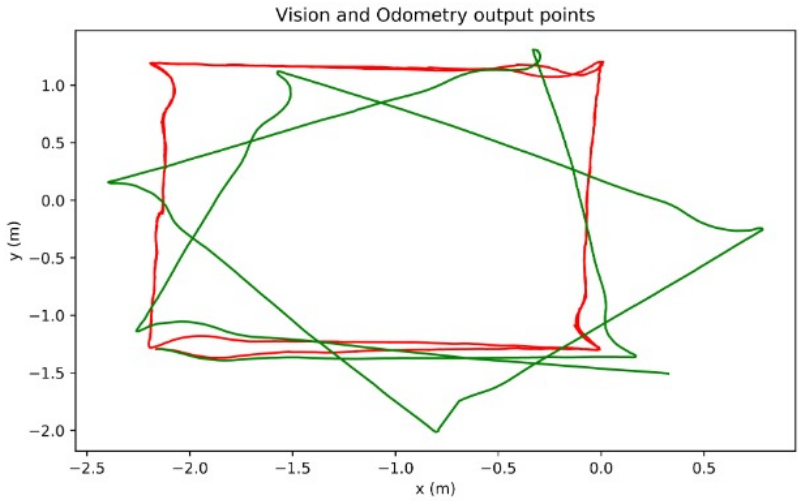
the odometry from Equation (5.1). Although kinematics considers all the robot's construction parameters, the odometry produces a path that differs from the actual course. Shown in Figure 27, the robot's odometry path in green tracks it in different positions and orientations from the ground truth, the red.

5.1.2 Wheels' Speed

A magnetic encoder measures the wheel speed with 1024 pulses for every motor rotation. The robot uses a Quadrature Encoder Interface (QEI) present the F767ZI ARM embedded board (ST, 2019). The QEI does four readings for every pulse of the encoder because there are channels A and B for every 1024 pulses, and the interface counts each rise and fall of the two channels. The embedded system keeps this count in a register, and every two milliseconds, there is a routine to calculate the motor rotation frequency. The routines also convert the motor frequency to wheel velocity using 5.7, in which $MOTOR_{GEAR}$ has 18 teeth and $WHEEL_{GEAR}$ has 60.

$$WHEEL_{SPEED} = 2.0 * \pi * (MOTOR_{GEAR}/WHEEL_{GEAR}) * MOTOR_{FREQUENCY} \quad (5.7)$$

Figure 27 – Robot's path tracking from two different sources, in red tracking from an external camera and in green tracking from robot's odometry.



Source: The author

5.1.3 Inertial Measurement Unit

The Inertial Measurement Unit (IMU) reduces the orientation error as shown in Figure 27. IMU reports angular velocity, but it is possible to achieve the robot's relative orientation by integrating velocity over time. The relative orientation can be easily integrated at the odometry, replacing the kinematic equation angular rotation from wheels to the IMU.

As a result of tracking the robot's orientation movement using the IMU, the odometry went from Figure 27, with RMSE of 1.24, to the odometry path at Figure 18, with RMSE of 0.09.

IMU gyroscope used in RobôCIn robot is the MPU6050, and it has $1kHz$ of refresh rate, giving a high angular velocity precision (INVENSENSE, 2013). The RobôCIn robot integrates it every $2ms$, which results in the robot's relative angular movement. The only IMU complication is the systematic offset error in the readings, which increases angular orientation continuously even when the robot is steady. The solution is to apply a self-calibration method when the robot is not moving. Therefore, the work proposes Algorithm 3 to build the offset calibration of the IMU readings. The Algorithm 3 is based on an integrative control. The offset is the sum of the previous sensor samples multiplied by a factor of 0.05 to integrate noise samples until it reaches the correct offset partially. Moreover, the algorithm will reach the correct offset when all sensor readings minus the offset are less than the acceptable error proposed in this work, as 0.05 degrees per second.

It is important to note that the robot must be stationary so the algorithm correctly reaches the offset value. Therefore, this work recommends running at initialization of the robot's embedded software.

Algorithm 3: IMU offset calibration using an Integration control over the readings.

```
1  ε = Minimum IMU accepted error;
2  //Value to proportionally increase offset with reads;
3  k = 0.05;
4  offsetX = 0;
5  offsetY = 0;
6  offsetZ = 0;
7  while true do
8      //Read IMU angular velocity for each axis;
9      x = imu.read.x();
10     y = imu.read.y();
11     z = imu.read.z();
12     //Removes IMU offset calibration;
13     x -= offsetX;
14     y -= offsetY;
15     z -= offsetZ;
16     //Calculate error, and update it if necessary;
17     imuError = x + y + z;
18     if imuError < ε then
19         break;
20     else
21         offsetX += x * k;
22         offsetY += y * k;
23         offsetZ += z * k;
```

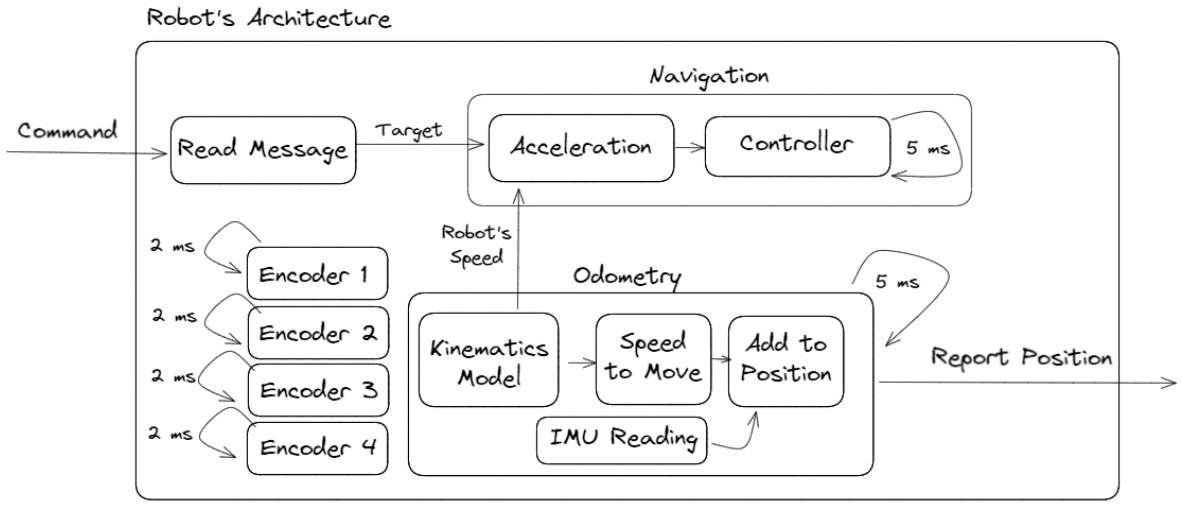
5.1.4 Robot's Odometry

The main goal of this work is to optimize the robot's odometry, and then it is essential to evaluate the robot's path tracking before and after the optimization process. The simulation aims to recreate the same odometry path as the robot's odometry; however, different from the simulation, the robot's odometry comes from robot-embedded kinematics. Moreover, another experiment is required to evaluate kinematics with other parameters.

The robot has a timed routine to build the odometry, similar to IMU and the encoder sensor. The routine is triggered every five milliseconds, and whenever triggered, it gets the wheels' speed and converts it to the robot's movement. It uses kinematic equations to convert from wheels to the robot's speed and the time to convert speed into motion. The routine also gets the IMU relative angle, then the displacement from the last five milliseconds is calculated.

The displacement, added to the last known position, transforms displacement into positions and forms a path. This process is shown in the robot's high-level diagram in Figure 28, and it starts with a timed routine of 5 milliseconds for the odometry calculus. The wheels' speed is acquired by four routines of 2 milliseconds each. Then, a second routine of 5 milliseconds runs to report via radio all robot information, including the current position estimated by the odometry. Together with the robot's odometry, the wheel's speed is reported to the off-field computer to source the simulation and validate it between the robot's internal path tracking and ground-truth positions.

Figure 28 – RobôCIn robot embedded software diagram which has 4 encoder routines running every 2 milliseconds, one navigation routine and another odometry routine, both running every 5 milliseconds. Besides of the routines, the robot receives external commands that together with its velocity information go through the acceleration and controller module. When the odometry routine runs, it gets encoders' readings, calculates the kinematic equation, and converts it to the robot's current position together with IMU angular movement.



Source: The author

In the routines and their connections, shown in Figure 28, it is possible to see that the kinematic model is used in the odometry routine. The odometry routines get the encoder's readings, stored from their individual two milliseconds routing, to find the robot's velocity and then convert it in movement and position, together with the angular change from IMU. Besides the odometry module, the navigation routine runs when a command is received and updates the movement controller, which updates the wheels' velocity every five milliseconds.

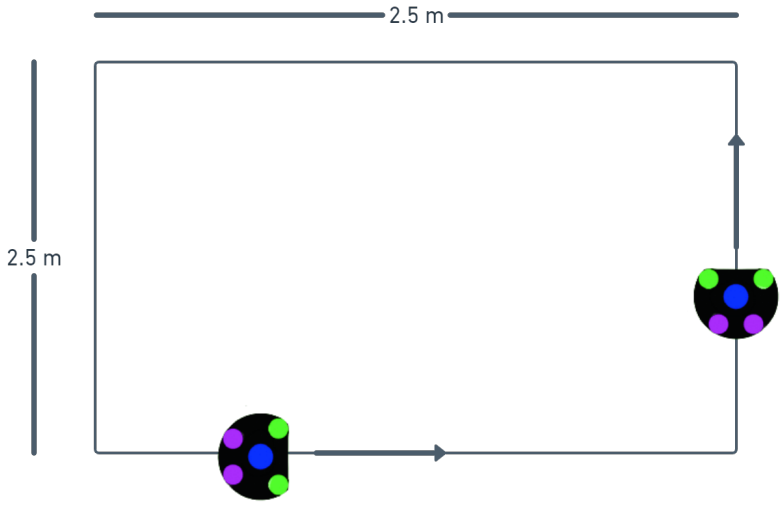
5.2 DATA COLLECTION

After setting the kinematics model to the robot, it is necessary to record data to optimize the modelling. It is essential to capture data with the robot that requires optimization and where it will work because data should include wheels' velocity from the robot, which model has mechanical inaccuracies. Therefore, simulation can reproduce inaccuracies in the simulation path, and later, optimization can fix them.

5.2.1 Experiment

The proposed method uses a quadrilateral path in the environment. As shown in Figure 29, the square has a 2.5 m side and is used to capture the robot's motion in different axes. In the experiment, the robot walks forward, has side movements, and rotates around the square. These variations evaluate the kinematics at all the pose components (x, y, ω) possible.

Figure 29 – A 2.5 m square experiment path, performed by the robot.



Source: Melo *et al.* (2022)

Although Sousa *et al.* (2022) proposes different paths and directions, such as doing the square in different directions, this work proposes optimizing odometry with a simple and flexible setup. The proposed optimization works because of the evaluation method that takes

accountability for all samples and not a few ones like Sousa *et al.* (2022) and Borenstein & Feng (1996b).

One experiment may run the robot multiple times in the square or only once. In the end, the number of samples and movement variation matters. The only detail is to update the odometry's first position to match the ground truth. This first point update is necessary because the odometry tracks relative and ground-truth absolute positions. Only giving the same starting point will equal referential for both tracking systems.

5.2.2 Wireless Communication

RobôCIn robot already used wireless communication by Nordic Radio (NORDIC, 2008). It has a base station, shown at Figure 30, in which there are two Nordic radios, one to send commands from a computer and another to receive the robot's data at the computer. These two radio system enables a high-rate duplex communication developed in a prior work Cavalcanti *et al.*, 2022.

Figure 30 – RobôCIn's base-station, responsible to transmits data from computer to robot, and to receive data from robot to computer.

Source: The author

Each radio only supports up to 32 bytes by packet, and to achieve a high-rate transmission of the sample, all required information should use the minimum number of bytes necessary. Although encoded samples may have a few bits, the optimization depends on the data's precision. The precision decrease happens because the number of bits determines the precision of the sample, and fewer bits may reduce simulation accuracy. On the other hand, increasing the number of bits or messages will reduce the rate of messages.

Although one integer may take 32 bytes, the robot has limits of velocity and sensor precision, and these limits serve as a tradeoff between accuracy and message rate. Then, the

sample encoding relies on readings limits to set the required number of bits.

Table 4 – Packet data sent from robot to computer in order to simulate and evaluate odometry path.

Bits Offset	Bits Size	Information
0-3	4	Message Type
4-7	4	Robot Id
8 - 23	16	x - Odometry position in X
24 - 39	16	y - Odometry position in Y
40 - 55	16	ω - Odometry orientation
56 - 70	15	Dribbler's Motor Speed
71 - 78	8	Kick's Capacitor Load
79	1	Ball on the Robot
80 - 87	8	Robot's Battery
88 - 103	16	ϕ_1 - Motor 1 Speed
103 - 119	16	ϕ_2 - Motor 2 Speed
120 - 135	16	ϕ_3 - Motor 3 Speed
136 - 151	16	ϕ_4 - Motor 4 Speed

Source: The author

As the robot has four wheels, there are four routines to calculate the speed of all revolutions. Each wheel's speed encode uses 16 bits; in the end, all four wheels' speed requires 8 bytes.

5.2.3 Ground Truth

The ground truth is that the robot's poses navigated during the experiment. It is essential to synchronize the robot information log with the ground truth position to compare and evaluate using RMSE later.

The Small Size League (SSL) uses a shared vision for all teams called ssl-vision proposed by Zickler *et al.* (2010). The ssl-vision sends the absolute position of the robot in a local Ethernet network, and then each team can consume it and process it. The ssl-vision collects the robot's ground-truth position in the field and becomes evaluation data for odometry optimization.

The off-field computer, which received the robot's information from the base station, also received the ssl-vision absolute position of the robot. Whenever a message arrives from the robot, the computer fetches a ground-truth position and logs that information in a comma-separated value format.

The comma-separated value file from an experiment has multiple lines; each line has the wheel's speed, the robot's odometry, and the ground truth position. This information makes optimizing odometry and evaluating the robot's odometry possible. For example, Figure 31 presents an example of the log file lines, with the robot's odometry at $ROBOT_X$, $ROBOT_Y$, and $ROBOT_W$ columns, the wheel's speeds are at $ROBOT_{M1}$, $ROBOT_{M2}$, $ROBOT_{M3}$, and $ROBOT_{M4}$ columns, and ground-truth as $VISION_X$, $VISION_Y$, and $VISION_W$ columns.

Figure 31 – A comma-separated value file, with the robot's pose from odometry, then there are the wheels' speed in radians per second. And later, there is the ground-truth pose, named as vision.

ID	ROBOT_X	ROBOT_Y	ROBOT_W	ROBOT_M1	ROBOT_M2	ROBOT_M3	ROBOT_M4	VISION_X	VISION_Y	VISION_W
0	0.569	-2.514	1.5999	0	0	0	0	0.57028	-2.51811	1.60207
0	0.569	-2.514	1.5999	0	0	0	0	0.570028	-2.51816	1.60207
0	0.569	-2.514	1.5999	0	0	0	0	0.570685	-2.51842	1.60289
0	0.569	-2.514	1.5999	0	0	0	0	0.570648	-2.51853	1.60289
0	0.569	-2.514	1.5999	0	0	0	0	0.570611	-2.51864	1.60289
0	0.569	-2.514	1.5999	0	0	0	0	0.571232	-2.5184	1.60488
0	0.569	-2.514	1.5999	0.23	0	0	0	0.571375	-2.51841	1.60488
0	0.569	-2.513	1.5996	1.38	0.92	-2.76	-1.15	0.57149	-2.51841	1.60488
0	0.569	-2.513	1.599	1.61	0.92	-2.3	-1.61	0.57154	-2.51811	1.60488
0	0.569	-2.513	1.5985	1.61	1.38	-2.07	-1.38	0.57111	-2.51784	1.61053
0	0.569	-2.512	1.5973	1.38	1.15	-1.38	-2.07	0.570769	-2.51756	1.61053

Source: The author

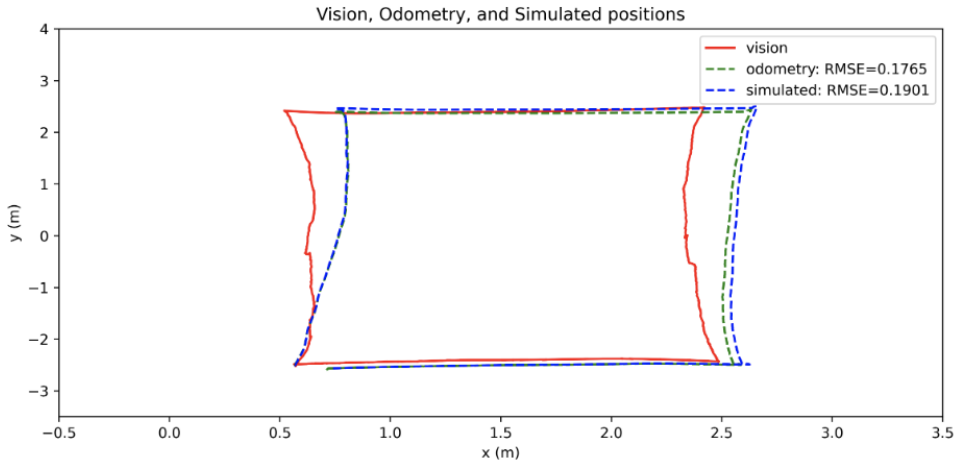
5.3 SIMULATION

The data collected from the robot, shown at Figure 31, facilitate the robot's path simulation, applying Algorithm 1 to the wheels' speed at 5.1, and IMU angle sent as robot odometry angle, and present in columns $ROBOT_W$.

Every log line at the comma-separated file will result in a simulation pose; the set of poses builds the simulated path, which evaluation goes against the ground-truth course.

A Python script, built to consume the log file, then simulate the odometry path using Equation (5.5) and Equation (5.6) at the Algorithm 1. The same script evaluates paths and plots the robot's odometry, the simulation and the ground-truth path, as shown in Figure 32.

Figure 32 – A graph with positions from robot's odometry in green, simulation, built from wheels' speed in blue, and ground-truth in red.



Source: The author

5.4 ODOMETRY OPTIMIZATION

The PSO algorithm was implemented in Python script using the parameters proposed in Chapter 4. The stop condition of five thousand took approximately 5 hours for four experiments, with parallelized evaluation for each of the 20 particles used.

The computer used to run the optimization had an Intel Xeon W-2235 CPU with eight threads at 3.8GHz. The high number of threads in the computer allowed the optimization to finish in a few hours. Without parallelization, the optimization took more than 16 hours.

The enormous optimization time happens because each optimization iteration will optimize 20 solutions across four experiments of approximately 600 samples, which sums 48 thousand runs of the kinematics equation. Over the 5 thousand iterations, this process will add up to 240 million calculations of the kinematics model; as the kinematics model has some matrix operations, the number of operations surpasses 1 billion. On top of that, each path goes over the evaluation against the ground truth.

This number of operations reinforces the need to optimize and evaluate the odometry outside the robot because the robot's battery lifetime cannot support the number of experiments multiplied by the population size and optimization iterations. The number of experiments may also degrade the robot's parts, generating expenses and failure to optimize if the robot malfunctions. Parallelization is also not possible to realize in real robots.

6 RESULTS

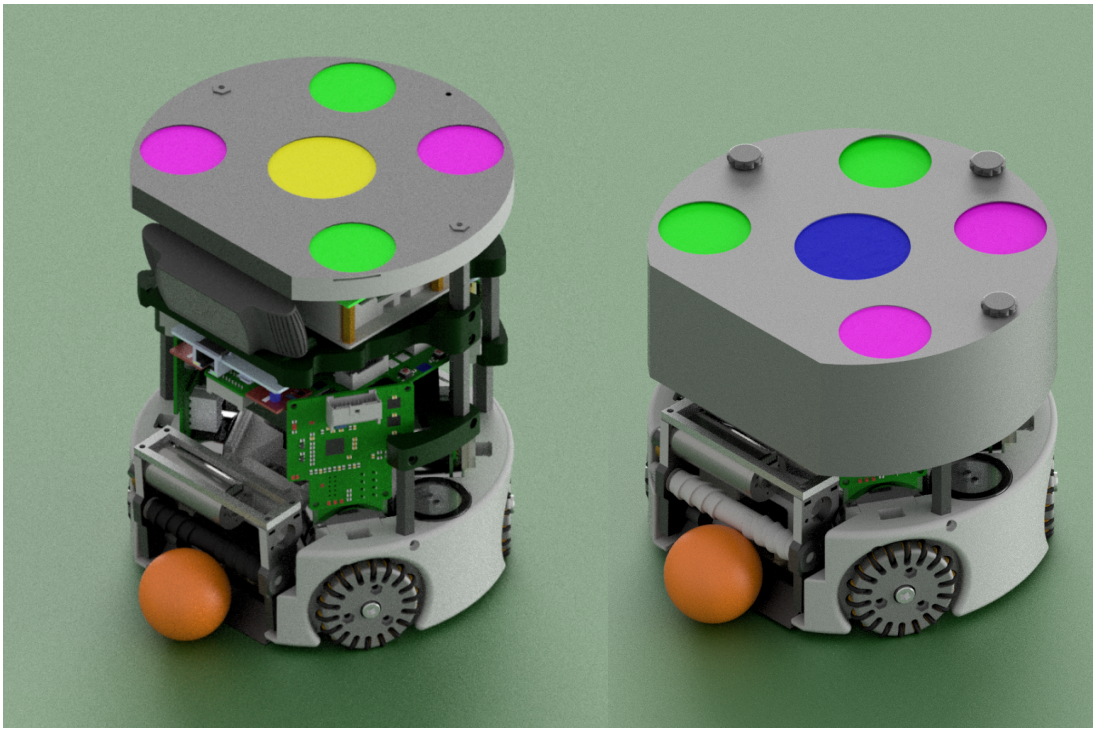
The optimization results for an omnidirectional robot using the wheel's speed data to simulate, evaluate and optimize kinematics are presented in this chapter. The main objective of this research was to propose a method to improve the accuracy of the robot's odometry to give the robot navigation autonomy. A series of experiments were necessary to accomplish optimization using a RobôCIn SSL robot designed for RoboCup soccer robotics competition.

The data collected from the experiments consisted of the robot's motion in the environment in which it plays soccer. The experiment movement path was squared, as shown in Figure 29, and it includes motion in field x, y and θ coordinates. The data collection used the robot's wireless communication and an external camera sensor to capture ground-truth positions.

The results of the experiments showed that the optimized kinematic model significantly improved the accuracy of the odometry estimates compared to the previous methods presented in Chapter 3 kinematic model. It was possible to reduce the odometry RMSE in simulation by 82%, and the real robot path tracking reduced error by 75%. These results demonstrate the proposed method's effectiveness in improving odometry accuracy.

Experiments used two robots, called robot 0 and robot 5, because of the id used in the vision system to identify them. Robot 0 and robot 5 have the same structure and kinematic model but are slightly different. Robot 0 is taller and heavier because it has an additional sensor and embedded system, shown in Figure 34a. On the other hand, robot 5 does not have it, to make the robot smaller and lighter as shown in Figure 34b.

Figure 33 – The first figure is the RobôCIn SSL robot with additional sensors, used for autonomous tasks. The second is the original RobôCIn SSL robot, used at the soccer games.



(a) RobôCIn SSL omnidirectional robot with aditional module, for autonomous tasks.

(b) Original RobôCIn SSL omnidirectional robot.

Source: The author

Experiments happened using both robots to validate the proposed optimization. The experiments conducted are listed below, which presents results from each experiment are in the sections below.

1. Optimization of robot 0 odometry.
2. Second optimization of robot 0, to enhance the first optimization.
3. Evaluation of robot 0 optimized kinematic parameters in robot 5.
4. Optimizing robot 5 odometry, with its data.

6.1 ROBOT 0 OPTIMIZATION

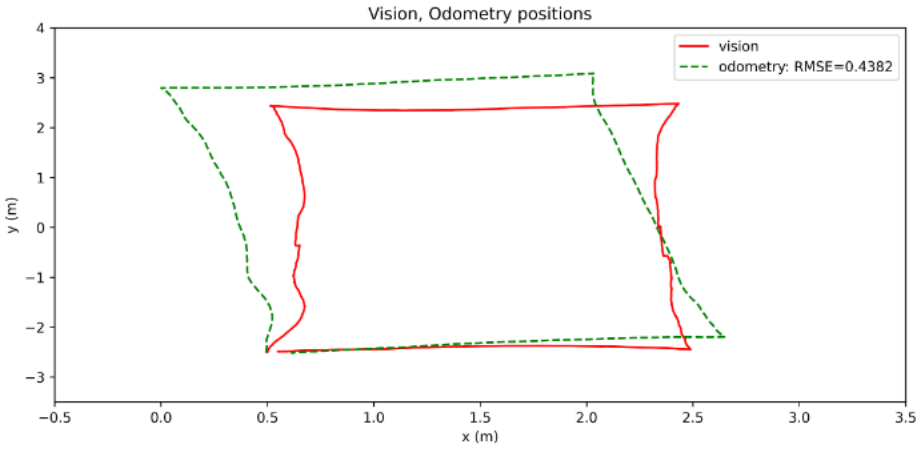
The first 4 square path movement experiments used robot 0. The data collected has the robot wheels' speed, ground-truth pose, and robot's odometry from original kinematic parameters, shown at Table 5. The robot's odometry and ground-truth position from vision have RMSE of 0.3749, as shown in Figure 34.

Table 5 – Original 13 kinematics parameters for RobôCIn SSL robot. Each ϑ is one parameter, the first 12 compose the J_1^{-1} matrix, and the last one is the wheel radius.

Parameter	Value
ϑ_1	0.34641
ϑ_2	0.282843
ϑ_3	-0.282843
ϑ_4	-0.34641
ϑ_5	0.414214
ϑ_6	-0.414216
ϑ_7	-0.414214
ϑ_8	0.414214
ϑ_9	3.616966
ϑ_{10}	2.556874
ϑ_{11}	2.556874
ϑ_{12}	3.616966
ϑ_{13}	0.02475

Source: The author

Figure 34 – Robot 0 experiment with ground-truth and robot's odometry position plot.

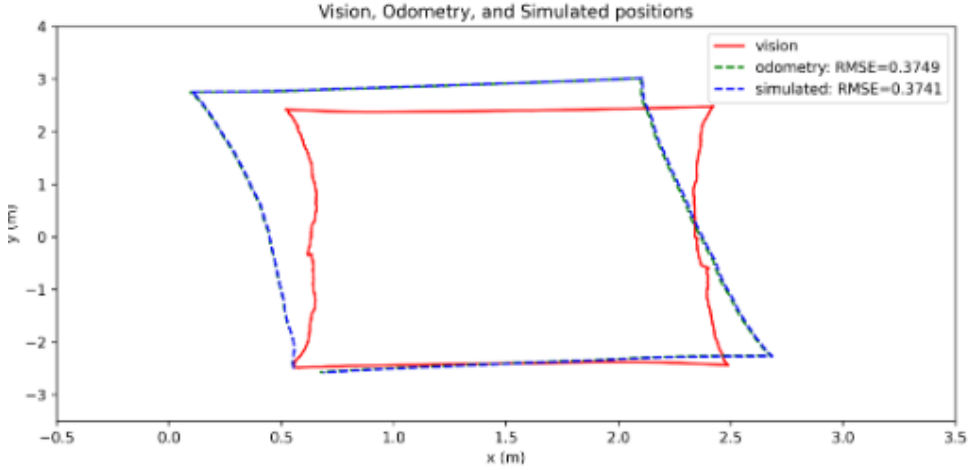


Source: The author

Applying the simulation in the data collected, as detailed in Algorithm 1, the RMSE was 0.3741. The simulation and robot's odometry match at the experiment path, shown in Figure 35, has the evaluation difference of 0.008.

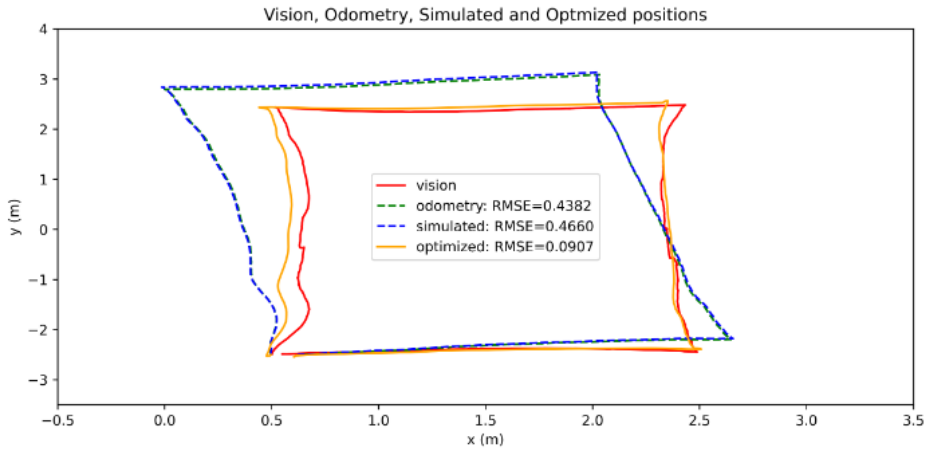
After validating the simulation, the PSO algorithm was executed with the simulation embedded in the evaluator, as detailed at Algorithm 2. As presented in Figure 36, the optimization achieved a set of kinematic parameters, presented in Table 6, which simulated path evaluation was 0.0907. Both the original parameters' and optimized simulation used the same dataset of wheels' speed. However, the original kinematic parameters' error was 0.3741, and the optimized parameters decreased to 0.0907 RMSE.

Figure 35 – Plot of Robot 0 experiment simulated positions, from wheel's speed, together with ground-truth and robot's odometry position.



Source: The author

Figure 36 – Plot of Robot 0 optimization result with optimized odometry positions, from wheel's speed, together original odometry simulation, ground-truth position and robot's odometry positions.



Source: The author

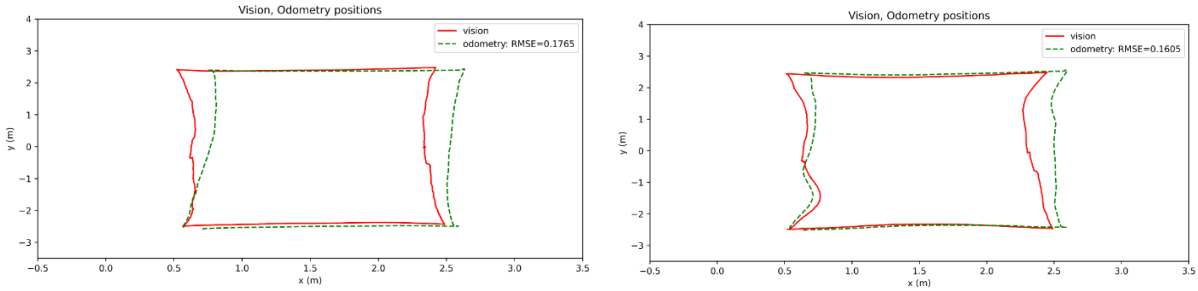
The final validation of the optimization was applying the optimized kinematic parameters in the real robot. The experiment was to move the robot in the same square path again to compare the robot's embedded odometry with the ground truth. The experiments results are shown at Figure 37, and it is possible to see that experiments resulted in an RMSE of 0.1765 and 0.1605.

Table 6 – Robot 0 optimized 13 kinematics parameters for RobôCIn SSL robot. Each ϑ is one parameter, the first 12 compose the J_1^{-1} matrix, and the last one is the wheel radius.

Parameter	Value
ϑ_1	0.383146590857885
ϑ_2	0.2601908088939393
ϑ_3	-0.29432191782356004
ϑ_4	-0.2947331116512286
ϑ_5	0.2942690569083855
ϑ_6	-0.5298154893258492
ϑ_7	-0.4135609185578648
ϑ_8	0.3062489555481053
ϑ_9	3.590359801850708
ϑ_{10}	2.5770948138832246
ϑ_{11}	2.572861189728561
ϑ_{12}	3.6344615627386694
ϑ_{13}	0.023262252867368625

Source: The author

Figure 37 – The first figure is the RobôCIn SSL robot optimized odometry validation with 1 meter per second, while the second is the validation with 2 meters per second.



(a) Plot of Robot 0 odometry validation using optimized kinematics parameters compared with ground-truth positions.

(b) Plot of Robot 0 odometry validation using optimized kinematics parameters, navigating at 2 meters per seconds, compared with ground-truth positions.

Source: The author

In Figure 38a, there is the experiment with the robot navigating at 1 meter per second, while in Figure 38b the navigation move robot with 2 meters per second. The real odometry optimization went from 0.3741, with original parameters, to 0.1765, with optimized parameters, reducing the path error by 53%.

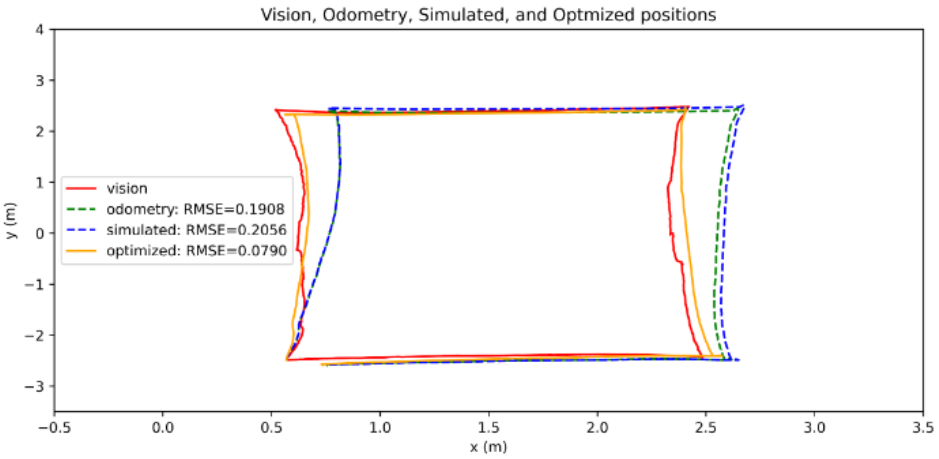
6.2 ROBOT 0 OPTIMIZATION OVER OPTIMIZATION

Although the error decreased by half, it is possible to see in both validation paths, at Figure 37, the robot's odometry is shifting the tracked path to the right side, the positive side of environment X axis. The experiment to validate the optimization also records the wheel's speed,

and then it is possible to optimize.

It was possible to simulate the robot's odometry path using the validation experiment data and the optimized parameters, shown at Table 6. The simulation was again used in the evolution process of the PSO to find even better kinematic parameters. As shown in Figure 38, the robot's odometry and simulation had a difference of only 0.0148 RMSE for the optimized parameters.

Figure 38 – Plot of Robot 0 second optimization result with optimized odometry positions, from wheel's speed, together original odometry simulation, ground-truth position and robot's odometry positions.



Source: The author

In Figure 38, the optimization process reduced the simulated odometry error from 0.2056 to 0.0790 with the optimization algorithm. This optimization found a new set of parameters, shown in Table 7

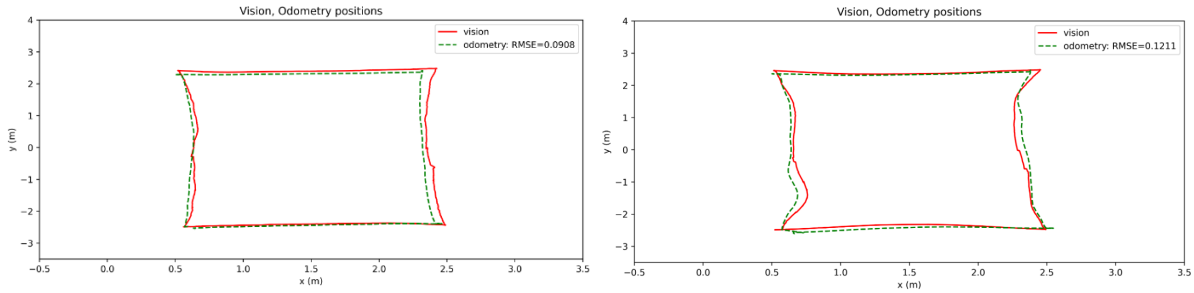
Again, this new set of parameters, found at the optimization over the first optimization process, was inserted in the robot to validate the kinematic in the robot's embedded odometry in the real environment. Shown in Figure 39, the robot produced an odometry tracking path with RMSE of 0.0908 for 1 m/s navigation and 0.1211 for 2 m/s navigation. The first kinematic optimization reached 0.1765 for 1 m/s navigation, and the second, with 0.0908, improved the odometry by 48%.

Table 7 – Robot 0 second optimized 13 kinematics parameters for RobôCIn SSL robot. The ϑ parameters compose J_1^{-1} matrix and wheel radius.

Parameter	Value
ϑ_1	0.42447140713655457
ϑ_2	0.35482242703016686
ϑ_3	-0.1823307411279881
ϑ_4	-0.274779656546075
ϑ_5	0.25745996970744844
ϑ_6	-0.4438060201135028
ϑ_7	-0.35102689244950486
ϑ_8	0.23426282528626358
ϑ_9	3.611082557939859
ϑ_{10}	2.6806232596725357
ϑ_{11}	2.5552490512551853
ϑ_{12}	3.5933353127553165
ϑ_{13}	0.022848272302565545

Source: The author

Figure 39 – The first figure is the RobôCIn SSL robot second kinematics optimization, validated with navigation at 1 meter per second, while the second is the validation with 2 meters per second.



(a) Plot of Robot 0 second optmized odometry validation using optimized kinematics parameters, navigating at 1 meter per second, compared with ground-truth positions.

(b) Plot of Robot 0 second optimized odometry validation using optimized kinematics parameters, navigating at 2 meters per seconds, compared with ground-truth positions.

Source: The author

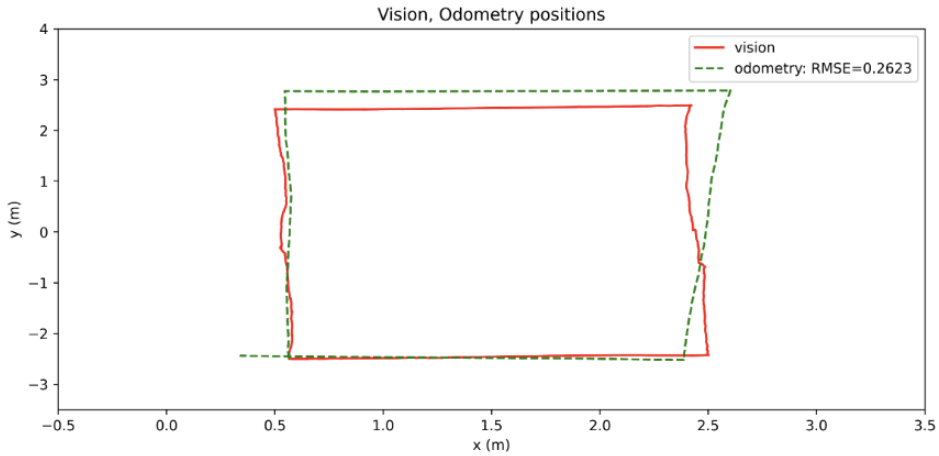
To summarize, the original kinematic model in the robot's odometry had an RMSE of 0.3749; after optimizing the error was reduced to 0.1765, and the set of parameters found are in Table 6. The validation experiment had the robot's odometry and wheels' speed; then, it enabled a second round of optimization using optimized parameters. The second optimization process reached 0.0908 of RMSE and found the set of parameters in Table 7. In the end, the error went from 0.3749 to 0.0908, reducing by 75%.

6.3 ROBOT 0 PARAMETERS IN ROBOT 5

With an improvement of 75% in path tracking using the robot's odometry optimization, testing another robot to validate parameters was possible. The experiment verifies if optimized kinematic parameters for one robot still optimize the odometry path for a different robot with the same model. The robot already optimized is called robot 0, shown at Figure 34a, and its optimized parameters, inputted in robot 5, shown at Figure 34b

Applying in robot 5 the original kinematic model, presented in Table 5, the robot's odometry path RMSE was 0.2623 as shown in Figure 40. As the RMSE error was higher than robot 0 optimization, applying robot 0 optimization can improve the path tracking.

Figure 40 – Robot 5 experiment with ground-truth and robot's odometry position plot.



Source: The author

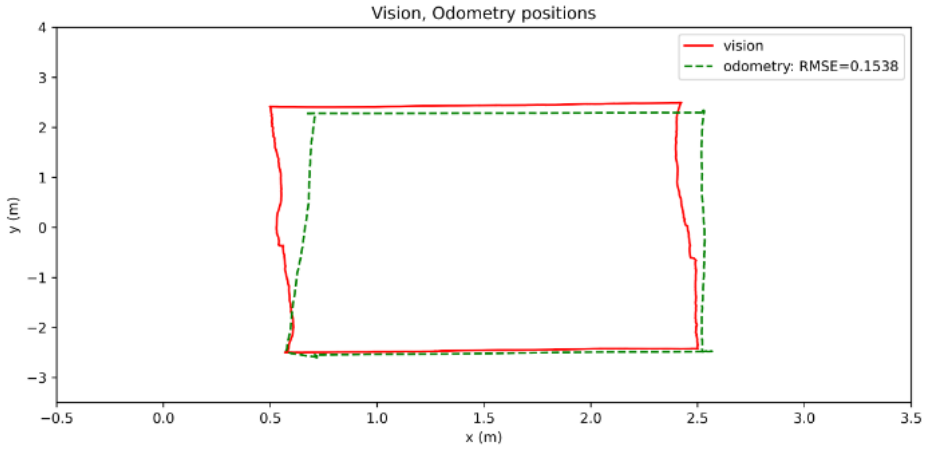
With robot 0 best-found parameters presented in Table 7, the robot 5 odometry tracking experiment, and the ground truth path is presented in Figure 41. The optimized parameters for robot 0, produced a path in robot 5 with RMSE of 0.1538. The robot 0 kinematic parameters improved robot 5 odometry tracking by 41%, as path error went from 0.2623 to 0.1538. Although improvement was significant, it did not reach an error close to the 0.0908 found when kinematic was evaluated in robot 0.

This result shows that using robot 0 kinematic parameters in robot 5, solved some systematic errors in the modelling. However, it was impossible to fix systematic errors due to the robot's construction and environmental interaction.

6.4 ROBOT 5 OPTIMIZATION

As the robot 0 optimized kinematic parameters applied to robot 5 did not reach a satisfactory error, compared to the robot 0 optimization, the robot 5 optimization was performed. In Figure 42, it is possible to see that the robot's odometry path RMSE was 0.2623, and the

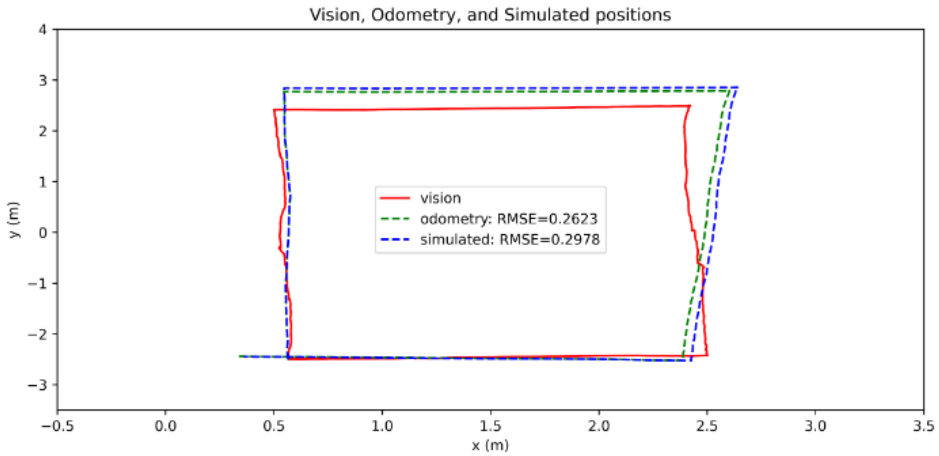
Figure 41 – Plot of Robot 5 odometry validation using optimized kinematics parameters from 0 optimization, compared with ground-truth positions.



Source: The author

simulation using the collected wheels' speed was 0.2978, both compared to the ground-truth positions.

Figure 42 – Plot of Robot 5 experiment simulated positions, from wheel's speed, together with ground-truth and robot's odometry position.



Source: The author

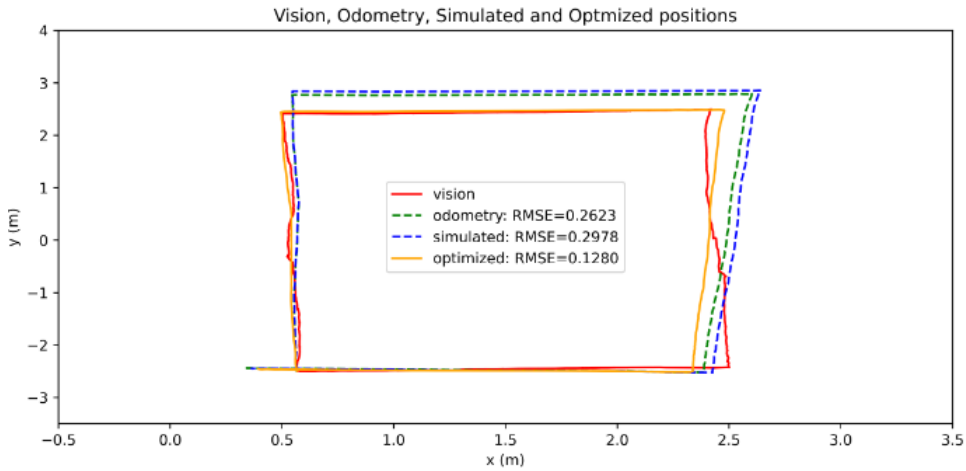
After playing the PSO algorithm to optimize the original odometry parameters to the best set of parameters for the robot, it was found an optimization with the error of 0.1280. The best parameters found by the optimization are shown in Table 8.

Table 8 – Robot 5 first optimized 13 kinematics parameters for RobôCIn SSL robot. The ϑ parameters compose J_1^{-1} matrix and wheel radius.

Parameter	Value
ϑ_1	0.3107303336519129
ϑ_2	0.16293553130406085
ϑ_3	-0.38569987170572684
ϑ_4	-0.4218453648289997
ϑ_5	0.38324707121540846
ϑ_6	-0.45977057928309134
ϑ_7	-0.43382527524942377
ϑ_8	0.35013037288291204
ϑ_9	3.5516509471783704
ϑ_{10}	2.563520283893822
ϑ_{11}	2.5794991635632774
ϑ_{12}	3.600614201817168
ϑ_{13}	0.022261783590247084

Source: The author

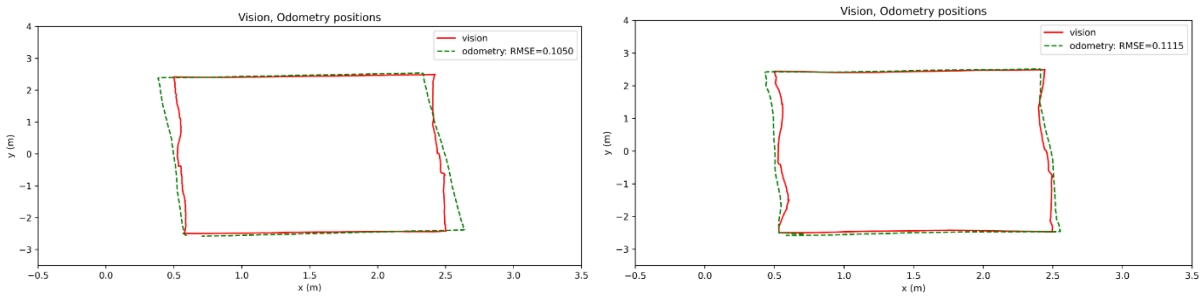
Figure 43 – Plot of Robot 5 optimization result with optimized odometry positions, from wheel's speed, together original odometry simulation, ground-truth position and robot's odometry positions.



Source: The author

Updating robot 5 with the new set of parameters found and realizing a new experiment, the odometry tracking returned the paths shown in Figure 44. In the Figure 45a, the robot navigated with 1 meter per second, and its odometry path error was 0.1050. The Figure 45b presents the path for robot 5 navigating with 2 meters per second, and in that experiment, the robot's odometry RMSE was 0.1115.

Figure 44 – The first figure is the RobôCIn SSL robot 5 optimized odometry validation with 1 meter per second, while the second is the validation with 2 meters per second.



(a) Plot of Robot 5 odometry validation using its optimized kinematics parameters compared with ground-truth positions.

(b) Plot of Robot 5 odometry validation using its optimized kinematics parameters, navigating at 2 meters per seconds, compared with ground-truth positions.

Source: The author

From the original odometry error at robot 5, with 0.2623 of error, to the optimized kinematic, there is a 60% decrease of the RMSE. Comparing with robot 5 using the robot 0 optimized parameters, shown at Table 7, and the robot 5 optimized one, shown in Table 8, the robot 5 error decreased from 0.1538 to 0.1050. This 31% error reduction proved that optimizing for the target robot potential the systematic error fixes, as data considers all construction errors.‘

6.5 CONSOLIDATE RESULTS

This section consolidates the results from robot experiments and optimization. The experiment results are presented in the Table 9, and besides the Root Mean Square Error (RMSE), there is the robot and its speed used during the experiment. For each configuration, four experiments were conducted, and the RMSE presented is the average between the experiments.

For the eight experiment cases evaluated and presented in Table 9, the first two cases are the robot 0 odometry path using its original kinematic parameters with speeds of 1 m/s and 2m/s speeds. Then, there is robot 5 with its original kinematics parameters for both speeds. The fifth and sixth results were from robot 0 optimized odometry, navigating at 1 m/s and 2 m/s, respectively. Right after, there is the robot 0 with its second optimization results. The later 4 rows are from the robot’s 5 experiments, first using the robot’s 0 kinematic optimizations at both speeds, and the last two rows are the results from robot 5 with its optimized kinematic parameters, navigating at 1 m/s and 2 m/s.

From Table 9, it is possible to see that the robot 0 odometry error went from 0.37486 to 0.17649 and, in a second optimization iteration, reached 0.09076. The overall improvement was 75% and was the best result for robot 0 path tracking. However, robot 5 has the same structure as robot 0, the best kinematic parameters for robot 0, were not the best for robot 5 odometry. The

Table 9 – Consolidate results for all experiments by kinematics and robot used to validate it. The table reports results from simulation and robots' embedded odometry RMSE, using each kinematics parameter, original and optimized.

Kinematics Parameters	**Robot**	**Speed**	**Odometry RMSE**
Original Kinematic	$robot_0$	1m/s	0.37486
Original Kinematic	$robot_0$	2m/s	0.43823
Original Kinematic	$robot_5$	1m/s	0.26232
Original Kinematic	$robot_5$	2m/s	0.29521
$Robot_0$ 1st Optimized	$robot_0$	1m/s	0.17649
$Robot_0$ 1st Optimized	$robot_0$	2m/s	0.16051
$Robot_0$ 2nd Optimized	$robot_0$	1m/s	**0.09076**
$Robot_0$ 2nd Optimized	$robot_0$	2m/s	0.12111
$Robot_0$ 2nd Optimized	$robot_5$	1m/s	0.15380
$Robot_0$ 2nd Optimized	$robot_5$	2m/s	0.16205
$Robot_5$ 1st Optimized	$robot_5$	1m/s	**0.10501**
$Robot_5$ 1st Optimized	$robot_5$	2m/s	0.11152

Source: The author

odometry kinematic parameters for robot 0, applied to robot 5 had an error of 0.15380, while the optimization for robot 5 had an error of 0.10501. Therefore, the robot 5 path tracking RMSE improved by 60%, from original to robot 5 optimized parameters.

As expected, the path tracking error is higher for faster navigation velocity. However, Table 9 shows that 2 m/s navigation at robot 0 had 72% of RMSE improvement, going from 0.43823 to 0.1211. Moreover, robot 5 improved from 0.29521 to 0.11152, representing a 62% of error improvement.

One of the most recent works in odometry optimization is from Sousa *et al.* (2022). It proposes to evaluate odometry quality by the maximum distance error from the robot's path tracking to the ground truth positions for each sample, taken every 0.5 meters of robot motion. The Sousa *et al.* (2022) best result reported to a 4-wheel mecanum robot was 0.20 m and 20° of error. The distance error result from this work's proposed optimization is presented in Table 10, where, similar to OptiOdm metrics, there is the maximum distance and orientation error from each experiment performed. It is possible to see that all optimizations for the 4-wheel omnidirectional robot reached less than 0.1 m and 1° of error. Compared to Sousa *et al.* (2022), the result from this work doubled path precision for omnidirectional robots, and the best result, of 0.028 m of error, is more than seven times better than previous work. This result shows the great potential of this work's proposed optimization; however, this comparison uses two different robots with different wheels, which impacts the result. Although both robots have 4 wheels and move in any direction, their kinematics model is different, and the wheels model highly impacts the relationship between the actuator and the robot's movement. For a better comparison same robot should be used.

Also using a different robot model, the work proposed by Lin *et al.* (2019) and their method achieved 0.1 m of maximum distance error between the optimized and original path. An

Table 10 – The table presents the maximum error distance ($\varepsilon_{max,d}$) and absolute angle difference ($\varepsilon_{max,|\theta|}$) between odometry and ground-truth positions for all experiments.

Kinematics Parameters	**Robot**	**Speed**	$\varepsilon_{max,d}$	$\varepsilon_{max,\|\theta\|}$
Original Kinematic	$robot_0$	1 m/s	0.39459	0.36506
Original Kinematic	$robot_0$	2 m/s	0.52734	0.39237
Original Kinematic	$robot_5$	1 m/s	0.14103	0.44390
Original Kinematic	$robot_5$	2 m/s	0.22434	0.47111
$Robot_0$ 1st Optimized	$robot_0$	1 m/s	0.07858	**0.32835**
$Robot_0$ 1st Optimized	$robot_0$	2 m/s	0.10135	0.33875
$Robot_0$ 2nd Optimized	$robot_0$	1 m/s	**0.02822**	0.43840
$Robot_0$ 2nd Optimized	$robot_0$	2 m/s	0.09866	0.39686
$Robot_0$ 2nd Optimized	$robot_5$	1 m/s	0.06099	0.33901
$Robot_0$ 2nd Optimized	$robot_5$	2 m/s	0.10308	0.31638
$Robot_5$ 1st Optimized	$robot_5$	1 m/s	**0.03070**	**0.31431**
$Robot_5$ 1st Optimized	$robot_5$	2 m/s	0.08777	0.33296

Source: The author

Table 11 – Average and standard deviation of 10 validation at each robot, using its best optimization odometry.

Kinematics Parameters	**Robot**	**Speed**	**Average Odometry RMSE**
$Robot_0$ 2nd Optimized	$robot_0$	1m/s	0.12 ± 0.05
$Robot_5$ 1st Optimized	$robot_5$	1m/s	0.15 ± 0.08

Source: The author

error of 0.1 m in the distance is similar to the overall experiment results presented in Table 10. However, the average and median distance error from the proposed optimized experiments is 0.07298 and 0.08318, respectively, which improves (Lin *et al.*, 2019) results. Besides the distance error improvement, the Lin *et al.* (2019) work was proposed and evaluated only in tricycle robots, as optimization used analytic equations. On the other hand, this work proposes an optimization without restriction in the robot's design, which evaluation is from an omnidirectional robot.

The result highlights that the kinematic model has systematic errors, and their reduction is possible with the proposed optimization method. It also shows that robots with the same structure may have similar mechanical issues in the kinematic model, as optimization from one robot increases the odometry of another with the same design. However, it is possible to reduce path tracking error using similar robot odometry optimization, and for better results, the proposed methodology should be applied to the kinematic model for each robot individually. After evaluating the optimization 10x for each robot, the accuracy reported was consistent and the aggregated RMSE is presented at Table 11. The RobôCIn's embedded software computational time was measured before and after the odometry algorithm, and the computation average and standard deviation time at one thousand loops went from 126.01 ± 0.44 microseconds without odometry, to 156.24 ± 0.47 microseconds with odometry. The increase in 25.03 microseconds does not impact the robot's computational time, as external commands are produced with 15 milliseconds of internal time.

7 CONCLUSION AND FUTURE WORK

7.1 CONCLUSION

In conclusion, this work presents a method to improve mobile robot navigation by improving odometry accuracy through its kinematic model. The methodology used robot-collected data to simulate an odometry-based path containing the model's mechanical inaccuracies to improve the kinematic model parameters for a four-wheel omnidirectional robot. The robot's simulation and collected data went to an evaluator that quantified the kinematic parameters quality based on the robot's odometry and navigated path. The evaluation method became crucial the population fitness calculus of the Particle Swarm Optimization (PSO). The accuracy improvement in this work reduced path Root Mean Square Error (RMSE) in one order of magnitude, aiding the mobile robot navigation through odometry to achieve a precision of fewer than 10 centimetres. At the same time, the improvement maintained odometry's operation and only changed the kinematic model parameters.

While previous authors target specific robot designs, the optimization proposed in this work is not coupled to any robot's model. A generic optimization was successfully made using the robot's data, independent of its design, an external simulation based on the existent kinematic model, and an evaluator built with RMSE equation. These three components evaluated and guided the parameters population at the PSO. One set of parameters represented a kinematic model, and the whole population evolved as a swarm, where each individual changed at each algorithm iteration. The optimization result returned the improved kinematic parameters, representing an optimized kinematic model for the robot odometry path tracking. The enhancement process has limitations; it requires dedicated experiments and multiple hours to optimize the robot's kinematic parameters in an off-board computer through simulation, evaluation and iterative optimization. On the other hand, the odometry does not change its computational cost between the non-optimized and optimized kinematic model; therefore, after optimization, navigation will gain precision without any additional computational cost.

Although the optimization process is computationally costly, the results of the experiments showed that path tracking errors improved its accuracy by 75%. The strategy to optimize odometry resulted in a final position error of less than 5 cm for a 10-meter path. The best result was reported at a second round of optimization for robot 0, and achieved less than 3 cm error for the 10-meter path. This enhanced accuracy allows better autonomous navigation, changing kinematics model parameters without external information or additional sensors fusion, such as cameras and lasers. Differently from the odometry process, additional sensors bring high computational costs to navigation strategies and interfere with the reaction time.

Although previous work uses different robots, resulting in different odometry accuracy, the proposed method does not require changing the optimization and evaluation for different robots. It requires experimentation and original kinematic model parameters available for any robot's design. Therefore, this work achieved a promising solution for enhancing the navigation

of different mobile robots. In a 10-meter path, the parameters found reduced the tracking error to less than 0.1 m error, with an average of 0.07 m for the eight experiments with odometry optimized. Previous work reported optimized odometry with a distance error of 0.1 for the differential robot and 0.2 m for omnidirectional robots, and they only support a few robot's kinematics models. Moreover, this work achieved half the odometry error from previous works and supports a broader range of robot models.

This work's accuracy improvement in path tracking gives robots more autonomy and time to navigate, facilitating processing tasks and aggregating more sensors without losing path tracking quality. Besides using a different robot, method of evaluation and optimization, this work also implemented the ground-truth position recording through the ssl-vision software (ZICKLER *et al.*, 2010). And the accuracy of the position recording contributes to the quality of the optimized odometry, as it better captures the robot's actual movements. On the other hand, the robot's odometry navigation was reported with 1 and 2 meters per second of navigation speed, demonstrating accuracy, even with different navigation speeds.

7.2 FUTURE WORK

The main focus of the work was odometry optimization for mobile robots; however, every robot with an actuator needs a kinematic to interact with the environment. Therefore, the optimization method proposed in this work may be applied to different robot structures, such as manipulators, which consist of multiple joint movements modelled by the kinematic. Unlike mobile robots, manipulators do not suffer from wheel slip, and joints are hardly attached to motors. On the other hand, similar to mobile robots, manipulators require model precision to reach the correct position without danger. One good example is arm robots to assemble cars; although they are in a controlled environment, they require precision to perform assembly in a specific position, synchronized with factory tasks. The result reported opens several avenues for future research that could be pursued, some of them are:

1. Extend the study to evaluate the proposed method on different types of robots with different kinematic models. To evaluate the method's robustness in generic robots.

2. Integrate the proposed optimized odometry with different navigation techniques to evaluate the combination results in more accurate navigation.

3. Evaluate the performance of the proposed method in different real-world environments, such as uneven terrain. To evaluate more sources of errors in the odometry track.

4. Automatize the optimization to optimize the robot's odometry in different environments quickly.

The items proposed above are the future steps and can uncover different applications and robot designs to apply the methodology proposed. However, the results enhanced autonomous robotics navigation, such as vision-based and laser-based navigation, because odometry is a fundamental task that reduces localization uncertainty. Therefore, more precise odometry reduces path tracking variance, and besides the additional navigation precision, navigation may reduce dependency on additional sensors. Moreover, the proposed work already built an accurate odometry path tracking based on the kinematic parameters' optimization, which presented results that overcame previous works. Therefore, it should contribute to applications based on odometry.

REFERENCES

ADI BEN-ISRAEL, T. N. E. G. (2003). *Generalized Inverses*. Springer New York, NY, 2nd edition.

ARAÚJO, V., MARTINS, F., FERNANDES, R., & BARROS, E. (2022). A telemetry-based pi tuning strategy for low-level control of an omnidirectional mobile robot. In ALAMI, R., BISWAS, J., CAKMAK, M., & OBST, O., editors, *RoboCup 2021: Robot World Cup XXIV*, 189–201.

ARVANITAKIS, I., TZES, A., & GIANNOUSAKIS, K. (2017). Mobile robot navigation under pose uncertainty in unknown environments. *IFAC-PapersOnLine*, 50(1):12710–12714. 20th IFAC World Congress.

BAGHDADI, A., CAVUOTO, L. A., & CRASSIDIS, J. L. (2018). Hip and trunk kinematics estimation in gait through kalman filter using imu data at the ankle. *IEEE Sensors Journal*, 18(10):4253–4260.

BERGSTRA, J., BARDENET, R., BENGIO, Y., & KÉGL, B. (2011). Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24.

BORENSTEIN, J. & FENG, L. (1996a). Gyrodometry: a new method for combining data from gyros and odometry in mobile robots. In *Proceedings of IEEE International Conference on Robotics and Automation*, 1:423–428 vol.1.

BORENSTEIN, J. & FENG, L. (1996b). Measurement and correction of systematic odometry errors in mobile robots. *IEEE Transactions on Robotics and Automation*, 12(6):869–880.

BORENSTEIN, J. & KOREN, Y. (1991). Local navigation for autonomous mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288.

BROWN, R. (1984). *Sudden Death: A Novel*. Random House Publishing Group.

CAO, Y., MA, S., XIE, Y., HAO, Y., ZHANG, D., HE, Y., & CAO, Y. (2022). Parameter optimization of cpg network based on pso for manta ray robot. In WU, M., NIU, Y., GU, M., & CHENG, J., editors, *Proceedings of 2021 International Conference on Autonomous Unmanned Systems (ICAUS 2021)*, 3062–3072.

CAVALCANTI, L., JOAQUIM, R., & BARROS, E. (2022). Optimized wireless control and telemetry network for mobile soccer robots. In ALAMI, R., BISWAS, J., CAKMAK, M., & OBST, O., editors, *RoboCup 2021: Robot World Cup XXIV*, 177–188.

CECCO, M. (2002). Self-calibration of agv inertial-odometric navigation using absolute-reference measurements. In *IMTC/2002. Proceedings of the 19th IEEE Instrumentation and Measurement Technology Conference (IEEE Cat. No.00CH37276)*, 2:1513–1518 vol.2.

CZACZKES, T. (2012). *Organisation of Ant Foraging*. PhD thesis.

DAS, J., PY, F., HARVEY, J. B., RYAN, J. P., GELLENE, A., GRAHAM, R., CARON, D. A., RAJAN, K., & SUKHATME, G. S. (2015). Data-driven robotic sampling for marine ecosystem monitoring. *The International Journal of Robotics Research*, 34(12):1435–1452.

DORIGO, M., BIRATTARI, M., & STUTZLE, T. (2006). Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39.

FANG, J., WANG, Z., ZHANG, H., & ZONG, W. (2018). Self-localization of intelligent vehicles based on environmental contours. In *2018 3rd International Conference on Advanced Robotics and Mechatronics (ICARM)*, 624–629.

GAD, A. G. (2022). Particle swarm optimization algorithm and its applications: A systematic review. *Arch Computat Methods Eng*, 29:2531–2561.

GANGANATH, N. & LEUNG, H. (2012a). Mobile robot localization using odometry and kinect sensor. In *2012 IEEE International Conference on Emerging Signal Processing Applications*, 91–94.

GANGANATH, N. & LEUNG, H. (2012b). Mobile robot localization using odometry and kinect sensor. In *2012 IEEE International Conference on Emerging Signal Processing Applications*, 91–94.

GORONZY, G. & HELLBRUECK, H. (2017). Weighted online calibration for odometry of mobile robots. In *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*, 1036–1042.

GUDISE, V. & VENAYAGAMOORTHY, G. (2003). Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No.03EX706)*, 110–117.

GUL, F., RAHIMAN, W., & ALHADY, S. S. N. (2019). A comprehensive study for robot navigation techniques. *Cogent Engineering*, 6(1):1632046.

HASSAN, M. A. A. (2012). A review of wireless technology usage for mobile robot controller. In *Proceeding of the International Conference on System Engineering and Modeling (ICSEM 2012)*, 7–12.

Hermann, M., Pentek, T., & Otto, B. (2016). Design principles for industrie 4.0 scenarios. In *2016 49th Hawaii International Conference on System Sciences (HICSS)*, 3928–3937.

HONIG, S. & ORON-GILAD, T. (2022). How user presence impacts perceptions and operation routines of robotic vacuum cleaners – a 'stay at home' experiment. In BLACK, N. L., NEUMANN, W. P., & NOY, I., editors, *Proceedings of the 21st Congress of the International Ergonomics Association (IEA 2021)*, 282–290.

HUANG, H., SAVKIN, A. V., DING, M., & HUANG, C. (2019). Mobile robots in wireless sensor networks: A survey on tasks. *Computer Networks*, 148:1–19.

INVENSENSE (2013). Mpu-6000 and mpu-6050 product specification revision 3.4r. `https://product.tdk.com/system/files/dam/doc/product/sensor/mortion-inertial/imu/data_sheet/mpu-6000-datasheet1.pdf`. [Online; accessed 15-January-2023].

IVANJKO, E., KOMšI, I., & PETROVIC, I. (2023). Simple off-line odometry calibration of differential drive mobile robots.

JAY LEE, BEHRAD BAGHERI, H.-A. K. (2015). A cyber-physical systems architecture for industry 4.0-based manufacturing systems. *Manufacturing letters*, 3:18–23.

JU, Z., YANG, C., & MA, H. (2014). Kinematics modeling and experimental verification of baxter robot. In *Proceedings of the 33rd Chinese Control Conference*, 8518–8523.

KALLASI, F., Lodi Rizzini, D., OLEARI, F., MAGNANI, M., & CASELLI, S. (2017). A novel calibration method for industrial agvs. *Robotics and Autonomous Systems*, 94:75–88.

KECSKéS, I., SZéKáCS, L., FODOR, J. C., & ODRY, P. (2013). Pso and ga optimization methods comparison on simulation model of a real hexapod robot. In *2013 IEEE 9th International Conference on Computational Cybernetics (ICCC)*, 125–130.

KENNEDY, J. & EBERHART, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, 4:1942–1948 vol.4.

KITANO, H., ASADA, M., KUNIYOSHI, Y., NODA, I., & OSAWA, E. (1997). Robocup: The robot world cup initiative. In *Proceedings of the First International Conference on Autonomous Agents*, 340–347.

KOLAR, P., BENAVIDEZ, P., & JAMSHIDI, M. (2020). Survey of datafusion techniques for laser and vision based sensor integration for autonomous navigation. *Sensors*, 20(8).

KUKA (2016). Hello industry 4.0 we go digital. *KUKA Aktiengesellschaft*.

LIN, P., LIU, D., YANG, D., ZOU, Q., DU, Y., & CONG, M. (2019). Calibration for odometry of omnidirectional mobile robots based on kinematic correction. In *2019 14th International Conference on Computer Science and Education (ICCSE)*, 139–144.

LIU, J., GAO, W., & HU, Z. (2019). Visual-inertial odometry tightly coupled with wheel encoder adopting robust initialization and online extrinsic calibration. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 5391–5397.

MADDAHI, Y., SEPEHRI, N., MADDAHI, A., & ABDOLMOHAMMADI, M. (2012). Calibration of wheeled mobile robots with differential drive mechanisms: an experimental approach. *Robotica*, 30(6):1029–1039.

MARTINS, F. B., MACHADO, M. G., BASSANI, H. F., BRAGA, P. H. M., & BARROS, E. S. (2022). rsoccer: A framework for studying reinforcement learning in small and very small size robot soccer. In ALAMI, R., BISWAS, J., CAKMAK, M., & OBST, O., editors, *RoboCup 2021: Robot World Cup XXIV*, 165–176.

MAYER, H., NAGY, I., & KNOLL, A. (2004). Kinematics and modelling of a system for robotic surgery. In LENARČIČ, J. & GALLETTI, C., editors, *On Advances in Robot Kinematics*, 181–190.

MELO, J. G. R., MARTINS, F., CAVALCANTI, L., FERNANDES, R., ARAÚJO, V., JOAQUIM, R., MONTEIRO, J. G., & BARROS, E. N. S. (2022). Towards an autonomous robocup small size league robot. *2022 Latin American Robotics Symposium (LARS), 2022 Brazilian Symposium on Robotics (SBR), and 2022 Workshop on Robotics in Education (WRE)*, 1–6.

NEMEC, D., ŠIMáK, V., JANOTA, A., HRUBOš, M., & BUBENíKOVá, E. (2019). Precise localization of the mobile wheeled robot using sensor fusion of odometry, visual artificial landmarks and inertial sensors. *Robotics and Autonomous Systems*, 112:168–177.

NEUMANN, A., HAJJI, A., REKIK, M., & PELLERIN, R. (2022). A didactic review on genetic algorithms for industrial planning and scheduling problems*. *IFAC-PapersOnLine*, 55(10):2593–2598. 10th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2022.

NGUYEN, D. H. & CHRISTENSEN, H. I. (2008). A review of mobile robot navigation techniques. *Robotics and Autonomous Systems*, 56(10):931–944.

NORDIC (2008). nrf24l01+ single chip 2.4ghz transceiver. `https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Pluss_Preliminary_Product_Specification_v1_0.pdf`. [Online; accessed 15-July-2022].

OLIVEIRA, H., SOUSA, A., MOREIRA, A., & COSTA, P. (2008). Dynamical models for omni-directional robots with 3 and 4 wheels. 1.

OSIńSKI, B., JAKUBOWSKI, A., ZIęCINA, P., MIłOś, P., GALIAS, C., HOMOCEANU, S., & MICHALEWSKI, H. (2020). Simulation-based reinforcement learning for real-world autonomous driving. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 6411–6418.

PIOTROWSKI, A. P., NAPIORKOWSKI, J. J., & PIOTROWSKA, A. E. (2020). Population size in particle swarm optimization. *Swarm and Evolutionary Computation*, 58:100718.

QIN, T., PAN, J., CAO, S., & SHEN, S. (2019). A general optimization-based framework for local odometry estimation with multiple sensors. *CoRR*, abs/1901.03638.

QUEIROZ, B. C. & FERREIRA, F. (2022). Soccer robots modeling project based on robocupjunior: Simulation environment for physical robot improvement. In ALAMI, R., BISWAS, J., CAKMAK, M., & OBST, O., editors, *RoboCup 2021: Robot World Cup XXIV*, 115–126.

QURESHI, U. & GOLNARAGHI, F. (2017). An algorithm for the in-field calibration of a mems imu. *IEEE Sensors Journal*, 17(22):7479–7486.

ROJAS, R. & GLOYE, A. (2005). Holonomic control of a robot with an omni- directional drive.

SHANAVAS, H., AHMED, S. A., & SAFWAT HUSSAIN, M. H. (2018). Design of an autonomous surveillance robot using simultaneous localization and mapping. In *2018 International Conference on Design Innovations for 3Cs Compute Communicate Control (ICDI3C)*, 64–68.

SIEGWART, R., NOURBAKHSH, I. R., & SCARAMUZZA, D. (2011). *Introduction to Autonomous Mobile Robots*. The MIT Press, 2nd edition.

Small Size League Technical Committee (2022). Rules of the robocup small size league 2022. `https://robocup-ssl.github.io/ssl-rules/sslrules.pdf`, last checked on 2022-12-15.

SOUSA, R., PETRY, M., & COSTA, P. (2022). Optiodom: a generic approach for odometry calibration of wheeled mobile robots. *J Intell Robot Syst*, 105:39.

ST (2019). Nucleo-f767zi. `https://www.st.com/en/evaluation-tools/nucleo-f767zi.html`. [Online; accessed 01-September-2022].

TOMASI, D. L. & TODT, E. (2017). Rotational odometry calibration for differential robot platforms. In *2017 Latin American Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR)*, 1–6.

VIE, A., KLEINNIJENHUIS, A. M., & FARMER, D. J. (2021). Qualities, challenges and future of genetic algorithms: a literature review.

WANG, Y., MA, H.-S., YANG, J.-H., & WANG, K.-S. (2017). Industry 4.0: a way from mass customization to mass personalization production. *Advances in Manufacturing*, 5.

WU, J., KAZANZIDES, P., & UNBERATH, M. (2020). Leveraging vision and kinematics data to improve realism of biomechanic soft tissue simulation for robotic surgery. *Int J CARS*, 15:811–818.

YOON, M., BEKKER, J., & KROON, S. (2016). New reinforcement learning algorithm for robot soccer. *ORiON*, 33.

YOU, Z. (2018). Chapter 7 - miniature inertial measurement unit. In *Space Microsystems and Micro/nano Satellites*, Micro and Nano Technologies, 233–293. Butterworth-Heinemann.

YU, T. & ZHU, H. (2020). Hyper-parameter optimization: A review of algorithms and applications.

ZHANG, X., ZOU, D., & SHEN, X. (2018). A novel simple particle swarm optimization algorithm for global optimization. *Mathematics*, 6(12).

ZICKLER, S., LAUE, T., BIRBACH, O., WONGPHATI, M., & VELOSO, M. (2010). Ssl-vision: The shared vision system for the robocup small size league. In BALTES, J., LAGOUDAKIS, M. G., NARUSE, T., & GHIDARY, S. S., editors, *RoboCup 2009: Robot Soccer World Cup XIII*, 425–436.

ŽLAJPAH, L. (2008). Simulation in robotics. *Mathematics and Computers in Simulation*, 79(4):879–897. 5th Vienna International Conference on Mathematical Modelling/Workshop on Scientific Computing in Electronic Engineering of the 2006 International Conference on Computational Science/Structural Dynamical Systems: Computational Aspects.