



**Universidad de Valladolid**



**ESCUELA DE INGENIERÍAS  
INDUSTRIALES**

**UNIVERSIDAD DE VALLADOLID**

**ESCUELA DE INGENIERIAS INDUSTRIALES**

**Grado en Ingeniería en Electrónica Industrial y Automática**

# **Location of a Mobile Robot using Odometry in the DMF**

**Autor:**

**Pérez Alonso, Celia**

**Responsable de Intercambio en la Uva:**

**Fernando Martínez Rodrigo**

**Universidad de destino:**

**Fachhochschule Technikum-Wien**

Valladolid, Julio, 2023.

TFG REALIZADO EN PROGRAMA DE INTERCAMBIO

---

TÍTULO: Location of a Mobile Robot using Odometry in the DMF

ALUMNO: Celia Pérez Alonso

FECHA: 13 de Junio de 2023

CENTRO: Main location at Höchstädtplatz

UNIVERSIDAD: Fachhochschule Technikum-Wien (Austria)

TUTOR: Nikolaus Angel

# Resumen

La Universidad de ciencias aplicadas de Viena, cuenta con una fábrica digital en miniatura en la cual se puede realizar la investigación, desarrollo e implementación de las diferentes tecnologías de la industria 4.0. Esta tiene varias estaciones de trabajo y con un robot móvil que se mueve entre ellas para hacer llegar al cliente las piezas correspondientes del mosquetón pedidas por el mismo.

El tema principal de este trabajo fin de grado es el desarrollo de un procedimiento por el cual se pueda obtener la localización del robot calculando sus coordenadas y su ángulo; todo ello con el objetivo de integrarlo en la fábrica miniaturizada de la universidad. El método que se usará para conocer la posición y orientación del robot estará basado en la odometría de un robot diferencial.

El control del robot se realizará mediante el puerto serie de Arduino o mediante Thing Worx, enviando los comandos necesarios para su movimiento. La pose (posición en coordenadas y orientación) del robot será enviada al servidor central haciendo uso de la comunicación IoT, donde se podrán visualizar y hacer uso para otros trabajos.

**Palabras clave:** Mobile Robot, Position, Odometry, Arduino, Thing Worx

# Abstract

The University of Applied Sciences Technikum Wien has a digital miniature factory in which it can be done the research, development and implementation of different technologies related with the 4.0 industry. This miniature factory has several working stations and a mobile robot that moves between them in order to deliver the corresponding carabiner parts ordered by the supposed customer.

The main subject of this final bachelor project is the development of a procedure by which the localization of the mobile robot can be obtained by calculating its coordinates and angle; all with the aim of integrating it into the miniaturised factory of the university. The method to be used to know the position and orientation of the robot will be based on the wheel odometry of a differential robot.

The control of the robot is done through the serial port of the Arduino or through ThingWorx, sending the necessary commands to make it moves. The pose (position in coordinates and orientation) of the robot will be sent to the central server using IoT communication, where it can be visualised and used for other projects.

**Keywords:** Mobile Robot, Position, Odometry, Arduino, Thing Worx

# **BACHELOR PAPER**

Term paper submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Engineering at the University of Applied Sciences Technikum Wien - Degree Program Mechatronik/Robotik

## **Location of a Mobile Robot using Odometry in the DMF**

By: Celia Pérez Alonso  
Student Number: 2200330006

Supervisor 1: Nikolaus Angel

Vienna, 13/06/2023

## Declaration of Authenticity

“As author and creator of this work to hand, I confirm with my signature knowledge of the relevant copyright regulations governed by higher education acts (see Urheberrechtsgesetz/ Austrian copyright law as amended as well as the Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I hereby declare that I completed the present work independently and that any ideas, whether written by others or by myself, have been fully sourced and referenced. I am aware of any consequences I may face on the part of the degree program director if there should be evidence of missing autonomy and independence or evidence of any intent to fraudulently achieve a pass mark for this work (see Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I further declare that up to this date I have not published the work to hand nor have I presented it to another examination board in the same or similar form. I affirm that the version submitted matches the version in the upload tool.”

Firmado por PEREZ ALONSO CELIA  
- \*\*\*0683\*\* el día 13/06/2023  
con un certificado emitido por  
AC FNMT Usuarios

Vienna, 13/06/2023

---

Place, Date

---

Signature

## **Abstract**

The University of Applied Sciences Technikum Wien has a digital miniature factory in which it can be done the research, development and implementation of different technologies related with the 4.0 industry. This miniature factory has several working stations and a mobile robot that moves between them in order to deliver the corresponding carabiner parts ordered by the supposed customer.

The main subject of this final bachelor project is the development of a procedure by which the localization of the mobile robot can be obtained by calculating its coordinates and angle; all with the aim of integrating it into the miniaturised factory of the university. The method to be used to know the position and orientation of the robot will be based on the wheel odometry of a differential robot.

The control of the robot is done through the serial port of the Arduino or through ThingWorx, sending the necessary commands to make it moves. The pose (position in coordinates and orientation) of the robot will be sent to the central server using IoT communication, where it can be visualised and used for other projects.

**Keywords:** Mobile Robot, Position, Orientation, Odometry, Arduino, Thing Worx

## Resumen

La Universidad de ciencias aplicadas de Viena, cuenta con una fábrica digital en miniatura en la cual se puede realizar la investigación, desarrollo e implementación de las diferentes tecnologías de la industria 4.0. Esta tiene varias estaciones de trabajo y con un robot móvil que se mueve entre ellas para hacer llegar al cliente las piezas correspondientes del mosquetón pedidas por el mismo.

El tema principal de este trabajo fin de grado es el desarrollo de un procedimiento por el cual se pueda obtener la localización del robot calculando sus coordenadas y su ángulo; todo ello con el objetivo de integrarlo en la fábrica miniaturizada de la universidad. El método que se usará para conocer la posición y orientación del robot estará basado en la odometría de un robot diferencial.

El control del robot se realizará mediante el puerto serie de Arduino o mediante Thing Worx, enviando los comandos necesarios para su movimiento. La pose (posición en coordenadas y orientación) del robot será enviada al servidor central haciendo uso de la comunicación IoT, donde se podrán visualizar y hacer uso para otros trabajos.

**Palabras clave:** Mobile Robot, Position, Orientation, Odometry, Arduino, Thing Worx

## **Acknowledgements**

I would like to express my sincere gratitude to my supervisor Nikolaus Angel, for guiding and helping me through all the process of this bachelor thesis.

I would also like to thank my Erasmus friends for being part of this stage of my life, for having helped me and encouraged me when I needed it.

And of course, I want to thank my family and friends in Valladolid, because, although it has been at a distance, I have felt all their support in the same way.



# Table of Contents

1	Introduction .....	5
1.1	Motivation .....	5
1.2	Problem definition .....	6
1.3	State of research.....	7
2	Development of the mobile robot's odometry .....	10
2.1	Types of wheeled vehicles .....	11
2.2	Methods for the location of a Mobile Robot .....	14
2.3	Wheel Odometry .....	17
2.3.1	Equations and calculations .....	17
2.3.2	Calibration test.....	20
3	Implementation in the DMF .....	26
3.1	Arduino .....	26
3.2	Thing Worx .....	29
3.3	Tests and Results .....	31
4	Conclusion .....	37
5	Points for improvement or expansion in a future .....	38
	Bibliography .....	39
	List of Figures.....	41
	List of Tables.....	42
	List of Abbreviations .....	43
	A Appendix: Some Arduino functions. ....	44
	B Appendix: Station coordinates.....	46

# 1 Introduction

It is well known that nowadays we are living the fourth industrial revolution. Industry 4.0 signifies a new stage in the Industrial Revolution that places significant emphasis on connectivity, automation, machine learning, and real-time data. It encompasses concepts like the Industrial Internet of Things (IIoT) and smart manufacturing, combining physical production and operations with intelligent digital technology, machine learning, and extensive data analysis. This integration aims to establish a more comprehensive and interconnected ecosystem for manufacturing and supply chain-focused companies. Irrespective of their unique characteristics, all present-day companies and organizations share a common requirement for connectivity and access to up-to-date insights spanning processes, partners, products, and individuals [1].

The University of Applied Sciences Technikum Wien is developing a Digital Miniature Factory (DMF) in which it is being implemented several of the technologies related with the 4.0 industry, offering a good testing scenario in which students can try and develop their bachelor or master thesis.

## 1.1 Motivation

The utilization of mobile robots enables the development of adaptable and self-governing industrial automation that is crucial for establishing Smart Factories. These factories prioritize the exchange of information, facilitated by the integration of cutting-edge intelligent technologies into robotics, including the IoT, Artificial Intelligence (AI), and Big Data [2].

One of the most important things about the operation of this type of robots, is how to know accurately its localization during the navigation, because with this ability they could operate autonomously in several environments, being this crucial for their successful development in real world applications.

As a result, intelligent and autonomous mobile robots can be used, enabling the creation of highly efficient and optimal industrial processes that make better use of resources and ultimately lead to enhanced productivity in all aspects [2].

Indeed, employing a self-governing mobile robot equipped with a localization system in the DMF enables the transportation of necessary boxes at any given time without the involvement of a human operator to control the robot's movements. The robot will autonomously follow the lines marked on the ground and, using the method of wheel odometry, some protocols will be established to ensure its safety when approaching areas that demand specific attention or pose potential risks.

All this bachelor thesis will answer the following research question: Will it be possible to get precise coordinates of the mobile robot in the DMF using the wheel odometry method?

## 1.2 Problem definition

The DMF has several stations, each one of them can perform different processes individually. The sorting station is responsible for separating the different pieces by colours. The separating station guarantees the necessary number of nuts or bolts. The storage station stores complete or semi-complete kits of the final product. The pieces in boxes are transporting between stations by a mobile robot. The stations and the mobile robot are controlled by a central server, using an adequate protocol.

Previously, a bachelor thesis has already been developed on this mobile robot, being the author of this Martin Kritzl in 2020. He developed the mechanical part of the mobile robot and the software to control the robot, so that the robot could be able to transport the pieces in boxes between the different stations of the DMF, by receiving commands from the central server or from the serial port from Arduino. The path the robot should take to a station must be the shortest distance to reach it, to do this, an algorithm was implemented to find the correct path to follow.

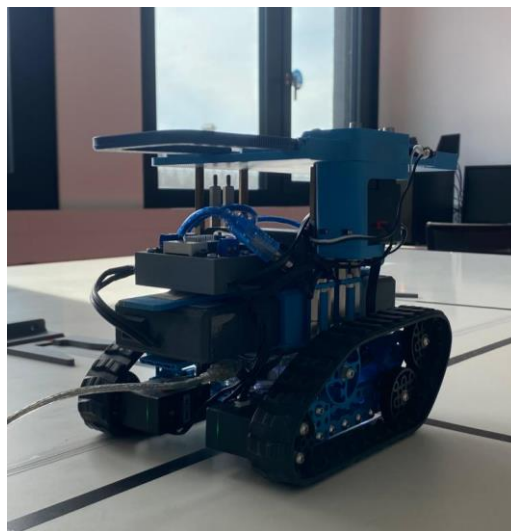


Figure 1: Mobile Robot in the DMF.

This bachelor thesis is also related with other thesis, which is being developed by Mrs. Claudia Lomas Blanco. She is doing a safety part, to avoid that the mobile robot could collide with the stations. She needs to know the localization of the robot in the factory; so, the main part of the present thesis is based on how calculate accurately the coordinates and orientation of the robot, knowed as its pose. This is one of the most critical aspects of controlling a mobile robot.

Indeed, other purposes of this thesis would be to enhance the control of the existing mobile robot which was developed before by Martin Kritzl, eliminating errors, improving its motion, in summary, making it more reliable and accurate.

Odometry is the most widely used technique for determining the instantaneous position of a mobile robot. In practical scenarios, it provides readily available real-time positional information between intermittent absolute position measurements. The frequency with which these (often costly and/or time-consuming) absolute measurements must be performed depends largely on the accuracy of the odometry system. However, the well-known drawback of odometry is its inaccuracy, which leads to the accumulation of unbounded errors. Typical odometry errors can grow to such an extent that the estimation of the robot's internal position becomes completely erroneous after even a short distance of as little as 10 metres [3], [4].

In spite of this, in this bachelor's thesis, the odometry-based control of the robot is going to be investigated with the aim of develop a reliable and accurate localization method for our mobile robot. Different odometry models are going to be studied as well as several options that can help to the motion of the robot. By the end of the research, it will be a better understanding of the opportunities and the potential of odometry methods, and the results could be aplicated in the fields of robotics and autonomous vehicles.

### **1.3 State of research**

The next text [5] discusses certain aspects to be taken into account in the localization of an autonomous robot. Mobile robots face a difficult challenge with navigation, the robot must go through various stages to carry out a good navigation, perception, localisation, cognition and motion control. There are three localisation problems based on the initial position: position trancking, global positioning and kidnapped robot problem. The first one, involves the robot's initial position being known, and it is based on continuously track the robot's location. Odometry and sensor data are used for tracking, but large uncertainty can make the localisation difficult. On the other hand, in Global localisation, the robot hasn't any knowledge of its initial position, so it needs to locate itself within the environment. The last problem is related to when the robots is taken to an unknown location and must recover its pose.

There exist several projects and researchs related with the localization of mobile robots that implement different solutions:

This paper [6] focuses on incorporating odometry and a localization system based on the DC magnetic field present in the indoor environment for the navigation of a mobile robot. The DC magnetic field, generated by factors such as building rebar and magnetic materials, is known to exist in the environment. In this study, a magnetic sensor is utilized to detect and create a

magnetic map of this field. The robot's localization is achieved by comparing the readings from the sensor with the stored DC magnetic field information in the magnetic map.

The following paper [7] aims to create a localization technique that utilizes affordable devices, specifically cameras, while still achieving satisfactory accuracy in tracking performance. Vision data is rich in information and has the potential for high tracking precision. However, due to its computational demands, visual-based localization is typically conducted at a low frequency. To improve the speed and accuracy of visual localization, it is integrated vision information with the robot's odometry using a Kalman-Filter. This combined approach allows for tracking performance with acceptable accuracy, with errors in the range of a few centimeters, at a frequency of approximately 35Hz.

In the next project [8] it is introduced the Fast Sampling Plane Filtering (FSPF) algorithm for processing depth camera data in real-time for indoor mobile robot localization and navigation. The algorithm reduces the volume of the 3D point cloud by sampling points and classifying them as "plane filtered" or "outlier" points. The authors also present a localization algorithm that projects the plane filtered points onto a 2D map and assigns correspondences to lines the map. The proposed approach operates in real-time, achieves high frame rates, and requires low CPU usage. Experimental results demonstrate its effectiveness for indoor mobile robot localization and navigation compared to alternative methods.

This paper [9] focuses on RFID-based localization for mobile robots, where RFID tags are distributed within the environment. To address this, the authors propose a novel algorithm that combines an RFID system with an ultrasonic sensor system to enhance localization accuracy. By incorporating distance data from ultrasonic sensors, the proposed system mitigates uncertainties associated with RFID systems. The authors introduce a global position estimation (GPE) process based on RFID and a local environment cognition (LEC) process utilizing ultrasonic sensors. To estimate the robot's position, a hierarchical localization algorithm is presented that integrates both GPE and LEC information.

In the following article [10], the importance of odometry in estimating the pose of wheeled vehicles is addressed, and is emphasized the need to minimize systematic and nonsystematic errors for accurate results. The focus is specifically on systematic error sources in car-like mobile robots (CLMRs), and a novel calibration method is proposed to figure out these issues. By performing a few test drives, the kinematic parameters of the CLMR can be calibrated, reducing deterministic errors. Additionally, the paper suggests enhancing odometry accuracy by fusing redundant odometry measurements using the extended Kalman filter (EKF), which effectively reduces nonsystematic or stochastic errors. The combination of calibration and odometry fusion results in improved pose estimation for CLMRs.

Using wheel encoders, odometry can provide good estimates of a vehicle's pose without prior knowledge of the environment. However, the main drawback of this method is the accumulation of errors. There exists two types of errors. Systematic errors can be eliminated or reduced by calibration kinematic parameters, we mean to errors produced by unequal wheels diameters or misalignment of them. The other type of error is known as nonsystematic error, these are stochastic and cannot be compensated directly. Wheel slippage is one of this type of error. Despite this, calibrating systematic errors can still improve the accuracy of odometry. [10]

In this bachelor thesis, the localization of the mobile robot during navigation is going to be studied by Odometry, so it is expected to develop a method with which the mobile robot could calculate its position indoor accurately.

## 2 Development of the mobile robot's odometry

The basis hardware of the mobile robot that is used in this project was the mBot Ranger from the Makeblock kit. This underwent some changes in the previous project dedicated to the same mobile robot, as some mechanical parts were added that were necessary to satisfy all its needs. In fact, several of the robot's component parts were printed on a 3D printer.

Another example is that the DC motors which supposed to be used to the proper movement of the robot were changed by the Smarts Servos MS-12A, that are developed also by Makeblock. It should be noted that these servos are going to be very important for this project, because due to them, we will acquire with the necessary information for executing our method in order to determining the robot's whereabouts. Subsequently, within this thesis, we will provide more details regarding these elements.

The robot can be controlled in two ways, one via the Arduino serial port, the other way is via Thing Worx. For this second manner, it was also necessary to include a raspberry pi, which allows such communication.

Regarding the software, the robot is controlled by the Auriga microcontroller also from the MakeBlock kit. Me Auriga's board is the updated version of Orion board and is equipped with multiple onboard sensors for temperature, sound, a gyroscope, a buzzer driver, wireless Bluetooth, etc. [11] Me Auriga is compatible with Arduino Mega 2560, so we used Arduino IDE to develop our code (based on C/C++); additionally, we needed to download and install the Makeblock library, enabling us to use all the robot's components such as the servos or the RGB sensor, by taking advantage of the pre-existing functions within it. Note that the Raspberry Pi and the serial port is connected to the Auriga's board via the USB interface of the microcontroller.

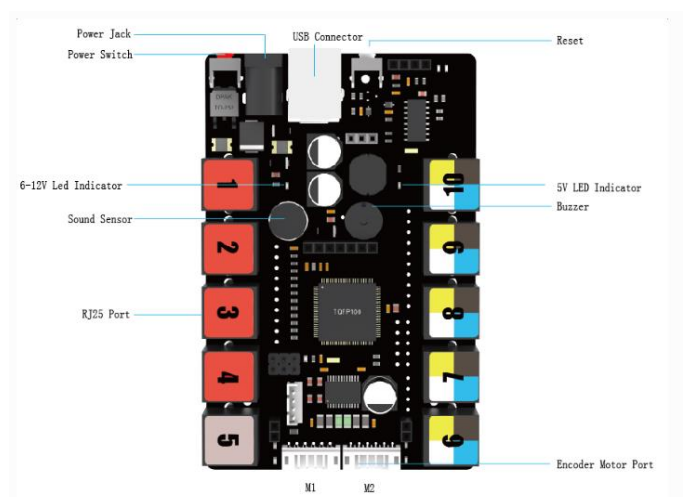


Figure 2. Components of the Me Auriga microcontroller (modified by [11]).

## 2.1 Types of wheeled vehicles

This section will give an overview of the different types of wheeled vehicles that exist according to their kinematic configuration.

The kinematic configuration of a mobile robot refers to the arrangement and types of joints and wheels used for its mobility. Hence, it is crucial to familiarize oneself with the different types of wheels available to gain a deeper comprehension of the subsequent explanations regarding kinematic configurations.

Driving wheels: Those that transmit traction to the ground and make our vehicle move.

Steering wheels: These wheels are responsible for inducing the car to change its direction either to the left or right.

Fixed wheels: The fixed wheels are not driving and therefore not controlled, and can only rotate around one axle.

Idler wheels: These wheels can rotate freely and are usually located at the bottom of the robot.

Moreover, it is important to know about the instantaneous centre of rotation (CIR), that is defined as "... the point fixed to a body undergoing planar movement that has zero velocity at a particular instant of time" [12]. This point is where all wheel axes intersect.

The different kinematics configuration are (translated from spanish, [13]):

- **Diferential configuration.**

This configuration uses two independently driven wheels or tracks on either side of the robot, each of them will be equipped with a motor. By varying the speeds and directions of the wheels, the robot can move forward, backward, and rotate in place. Differential drive is simple and widely used in small to medium-sized mobile robots.



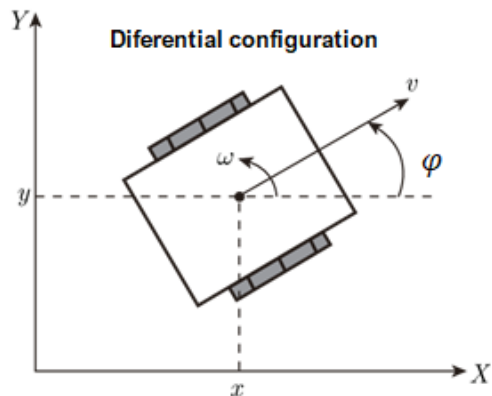


Figure 3. Differential configuration. (modified by [14])

- **Ackerman configuration.**

This configuration is commonly used in vehicles and incorporates two steered front wheels, together with a drive system for the two rear driving wheels. The front wheels are linked by a mechanism that allows them to turn at different angles, enabling the robot to follow curved paths and make smooth turns.

Because of its similarity to conventional vehicles, it is also called a car-type vehicle. [15]

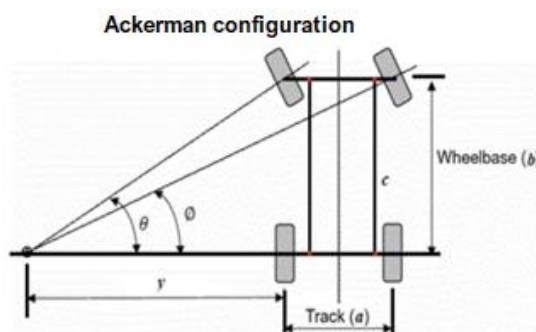


Figure 4. Ackerman configuration. (modified by [16])

- **Tricycle configuration.**

The setup of this system involves a front wheel that is both driven and steered, along with two rear wheels that are parallel and passive. Compared to the Ackerman configuration, this one offers increased maneuverability due to having only one steered wheel. However, it can still be prone to instability.

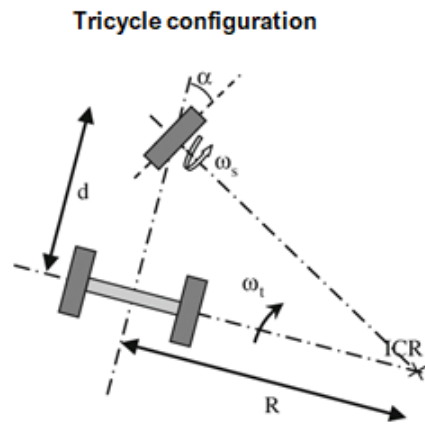


Figure 5. Tricycle configuration. (modified by [17])

- **Synchronous configuration.**

This configuration involves all wheels being actuated simultaneously and rotating in synchronization. Each wheel can be both driven and steered. The transmission is accomplished using ring gears or concentric belts.

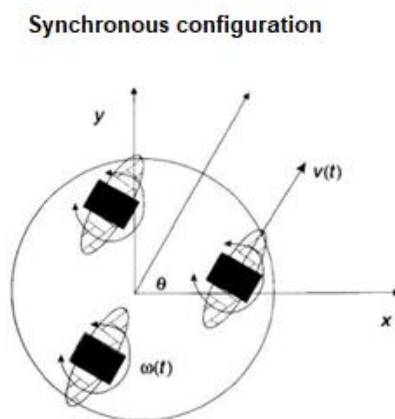


Figure 6. Synchronous configuration with 3 steering wheels. (modified [13])

- **Omnidirectional configuration.**

This configuration is built around three wheels that are both steered and driven. With three degrees of freedom, it possesses the ability to execute a wide range of movements and position itself in any desired location.

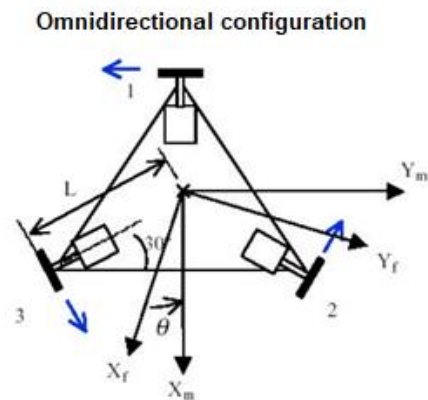


Figure 7. Omnidirectional configuration. (modified by [13])

## 2.2 Methods for the location of a Mobile Robot

The location of a mobile robot encompasses both its position in three-dimensional space, represented by  $x$ ,  $y$ , and  $z$  coordinates, as well as its orientation. This combination of position and orientation is commonly referred to as the robot's pose. Determining the mobile robot's pose, it would be able to answer to the fundamental question of "Where am I?".

There exist various odometric techniques for determining the pose of a mobile robot including:

- Wheel Odometry (WO).

Wheel odometry (WO) is widely used and particularly common in self-contained localization systems for skid-steering robots. It is a simple method utilized in vehicles where the right and left wheels can operate independently with varying speeds.

WO relies on wheel encoders to track the number of wheel revolutions, allowing for incremental pose estimation. The robot's current position is determined relative to its starting point.

However, this technique has several limitations. One of its drawbacks is position drift, where measurement errors accumulate over time, leading to decreased accuracy. Additionally, wheel slippage can occur on complex and uneven terrains, further affecting the reliability of the measurements.

While wheel odometry is a straightforward and cost-effective localization method, it is not suitable for platforms that require precise and long-term reliable localization systems. [18]

- Visual odometry (VO).

This method (VO) consists of estimating the motion using only the input of one or multiple cameras. VO is used in various applications, including robotics, wearable computing, and augmented reality.

Its functioning is similar to the wheel odometry, because VO incrementally estimates the pose by analysing the changes in motion reflected in the camera images. VO relies on having sufficient illumination and a static scene with enough texture for apparent motion estimation.

Unlike wheel odometry, VO is not affected by wheel slippage in uneven terrain or adverse conditions, making it advantageous. VO's relative position error ranges from 0.1 to 2%, making this method a supplement to wheel odometry and even to other systems like global positioning systems (GPS), inertial measurement units (IMUs) and laser odometry. [19]

- Inertial Odometry (IO).

The position, orientation, and linear velocity of the robot are determined in this system using measurements from an IMU sensor, relative to a specified starting point.

An IMU sensor is a micro-electro-mechanical system (MEMS) device equipped with a 3-axis accelerometer and a 3-axis gyroscope. The accelerometer measures acceleration not caused by gravity, while the gyroscope measures orientation by detecting gravity and magnetism. Due to their compact size and low power consumption, these MEMS-based sensors have become popular in resource-constrained systems like drones and micro-robots. Additionally, IMU-based navigation systems do not rely on external references to estimate platform position accurately.

However, these systems are prone to drifting issues caused by errors from various sources, such as consistent errors in gyroscope measurements and accelerometers. As a result, inertial odometry systems based on IMUs are not highly accurate and are not suitable for long-term localization applications. [18]

- Laser Odometry (LO) or LiDAR Odometry.

LiDAR (Light Detection and Ranging) is an approach used to estimate the position and orientation of a platform by tracking laser speckle patterns reflected from objects in the surrounding environment. One of the advantages of LiDAR is its insensitivity to ambient lighting conditions and its ability to operate effectively in low-texture environments.

The LiDAR-based sensing process typically involves two main stages: laser emission and optical observation. In the laser emission stage, a laser device emits coherent and spatial light into the environment. In the optical observation stage, the laser light interacts with objects, causing laser speckles to form on a 2D observation plane. These laser reflections are then monitored using optical detectors. By analyzing consecutive 2D images, a 3D representation of the environment can be reconstructed.

However, LiDAR odometry has some limitations. Implementing it on resource-constrained platforms can be challenging due to the computational demands of the iterative optical matching process required to establish correspondences between points in two sets. Furthermore, accurately scanning and correcting motion distortion caused by certain objects, such as glass, poses significant challenges and can result in decreased performance of the LiDAR system. [18]

- Radar Odometry (RO).

Radar odometry (RO) is a method used to estimate the relative motion of a platform by analysing scans obtained from an onboard radar sensor. Radar, derived from "radio detection and ranging," utilizes radio waves to measure the velocity, range, and angle of objects in the vicinity. There are two types of radar available: pulse radar and continuous-wave (CW) radar.

Pulse radar systems emit short yet powerful pulses and receive the echo signals during silent intervals. On the other hand, CW radar, particularly frequency modulated-continuous wave (FMCW) radar, continuously transmits modulated CW signals. The main distinction between these two types lies in the fact that CW radar, with its continuous signals, can generate high-resolution images from the reflected signals. However, pulse radar typically has a blind spot in front of the sensor, with a range of up to 50 meters.

CW radar, with its advantageous characteristics such as low sampling rate, low power consumption, and minimum target range, has garnered significant attention in localization and object avoidance applications. It can effectively generate high-resolution two-dimensional images while maintaining a physically compact antenna by combining FMCW and synthetic aperture radar (SAR) techniques.

Radar serves as a long-range active sensor that remains unaffected by adverse weather conditions and is capable of operating in environments with low texture. [18]

After conducting a thorough study and gathering relevant information, the decision was made to adopt the wheel odometry method for implementation. This choice was driven by its low cost and ease of implementation. While it is acknowledged that odometry tends to accumulate errors over distance, it is deemed suitable for our factory setting where the robot will only need to travel relatively short distances, mostly around 2 meters. Furthermore, since the robot will operate on a flat surface, any errors arising from uneven terrain can be avoided. Consequently, it is anticipated that the odometry method will yield satisfactory results.

## 2.3 Wheel Odometry

This chapter will discuss the equations needed to calculate the robot pose, as well as which calibration method will be used to adjust our parameters.

### 2.3.1 Equations and calculations

Our robot resembles a caterpillar, with 4 wheels on each side connected by a rubber chain. Out of the total 8 wheels, only two are actively driven by motors (the Smarts Servos MS-12A), while the others are fixed.

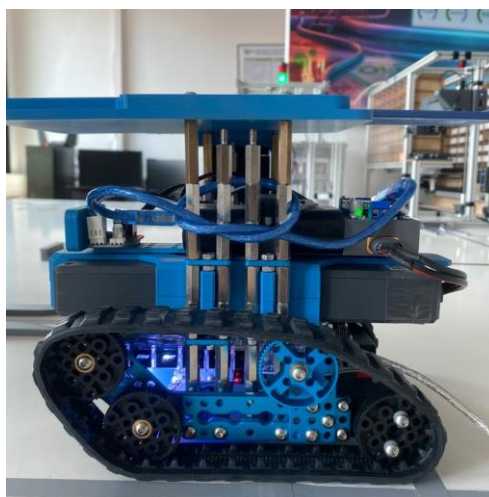


Figure 8. Profile view of our mobile robot. 4 wheels can be seen, the bottom right wheel is the drive wheel and the other 3 are fixed.

When designing a tracked mobile robot, it is important to consider various parameters, which result in complex mathematical models. This is primarily because this type of robot is commonly used in outdoor terrains, where factors like ground cohesion, density, sliding, and sinking need to be considered. (translated from Spanish, [20]).

Among the different kinematic configurations available, the most suitable one for our robot is the differential type with wheels. However, there is a notable distinction to keep in mind. In vehicles with wheels, the Instantaneous Centre of Rotation (ICR) remains constant and aligns with the wheel contact points on the ground. On the other hand, in tracked vehicles, the ICR varies and falls outside the track centres due to lateral sliding. This sliding is primarily caused by centrifugal force. Nevertheless, if our robot operates at low speeds or performs turns with a radius of 0, we can consider this lateral sliding insignificant. Another form of sliding that may occur is longitudinal sliding, influenced by the external ground factors.

In conclusion, we can approximate our robot to a differential configuration. It will not be operating at high speeds, all turns will have a radius of 0, and the terrain it traverses is smooth, eliminating the need to consider terrain disturbances. (translated from Spanish, [20]).

Now we will proceed to explain the development of the equations that model the differential kinematics of our robot based on (translated from Spanish, [21]).

The differential kinematics of translation and rotation depend on the individual motion of each drive wheel. The kinematic equations consistently calculate the coordinates relative to the centre point of the driving wheel axis. In the next figure, the location of a robot in the cartesian plane is shown:

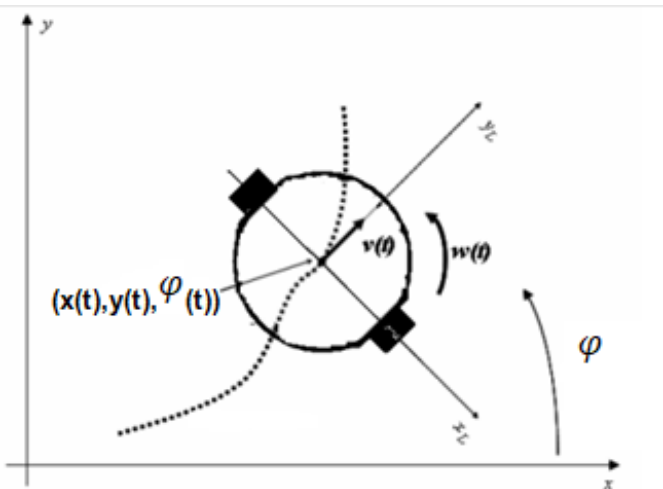


Figure 9. Location of the robot in the cartesian plane. (modified by [21]).

These equations are defined as follows:

$$\begin{aligned} \dot{x} &= v(t)\cos(\varphi(t)) \\ \dot{y} &= v(t)\text{sen}(\varphi(t)) \\ \dot{\varphi} &= w(t) \end{aligned} \quad (1)$$

The position and orientation can be calculated by integrating the velocities over a period  $\Delta t$ .

$$\begin{aligned} x(t) &= x(t_0) + \int_{\Delta t} v(t)\cos(\varphi(t))dt \\ y(t) &= y(t_0) + \int_{\Delta t} v(t)\text{sen}(\varphi(t))dt \\ \varphi(t) &= \varphi(t_0) + \int_{\Delta t} w(t)dt \end{aligned} \quad (2)$$

However, when the observation period " $\Delta t$ " becomes extremely small, we can analyze infinitesimal changes ( $\Delta x$ ,  $\Delta y$ ,  $\Delta\varphi$ ) in order to eliminate the need for integration. In simpler terms, if we maintain a constant and high sampling frequency during the robot's odometry, we can assume that the angular velocity of each wheel remains constant. This allows us to estimate the position and orientation using the following set of difference equations [21].

$$\begin{aligned} x_k &= x_{k-1} + \Delta x_k \\ y_k &= y_{k-1} + \Delta y_k \\ \varphi_k &= \varphi_{k-1} + \Delta\varphi_k \end{aligned} \quad (3)$$

By knowing the distance travelled by the wheels, the coordinates where the robot is located can be known. In our case, because we are using the smart servos MS-12 to move the wheels, we are able to know the angle turned ( $\theta$ ) by the wheels in radians. The relationship between the rotated angle and the linear displacement of a wheel is as follows:

$$\begin{aligned} D_{right} &= R_{right} * \theta_{right} \\ D_{left} &= R_{left} * \theta_{left} \end{aligned} \quad (4)$$

The differential configuration considers the vehicle lineal travel and the turned angle at the mid-point as the mean values associated with the right and left wheel (translated from Spanish [22]).

$$\begin{aligned} D &= \frac{D_{right} + D_{left}}{2} \\ \varphi &= \frac{D_{right} - D_{left}}{L} \end{aligned} \quad (5)$$

Where "L" is the length of the axle connecting the two wheels.



The lineal displacement of the robot is divided in the x coordinate and the y:

$$\begin{aligned} Dx &= \frac{D_{right} + D_{left}}{2} \cos(\varphi) \\ Dy &= \frac{D_{right} + D_{left}}{2} \sin(\varphi) \end{aligned} \quad (6)$$

So finally, the position of the robot using the difference equations (3) and the lineal displacement (6) will be determined by the following equations:

$$\begin{aligned} x_k &= x_{k-1} + \frac{\Delta D_{right} + \Delta D_{left}}{2} \cos(\varphi_k) \\ y_k &= y_{k-1} + \frac{\Delta D_{right} + \Delta D_{left}}{2} \sin(\varphi_k) \\ \varphi_k &= \varphi_{k-1} + \frac{\Delta D_{right} - \Delta D_{left}}{L} \end{aligned} \quad (7)$$

### 2.3.2 Calibration test

As it was said before, Odometry counts with different types of errors that must consider if an accurate localisation is required. There are two categories of errors [3], [23]:

- A) Non-systematic Odometry Errors.
  - Movement on uneven ground.
  - Movement over unexpected objects on the ground.
  - Wheel slippage.
  
- B) Systematic Odometry Errors.
  - Unequal wheel diameters.
  - Uncertainty about the effective wheel base.
  - Discrete resolution of the encoder (non continuous).

In order to develop an accurate odometry, these errors must be examined. The methods used to model and overcome the problems created by the previous errors are classified in benchmarks and using multiple sensors. On which we are going to base and study in this thesis are the "Benchmark techniques". They are based on testing the mobile robot over some predefined paths. In each experiment, the real value is compared with the value obtained by the odometry equations. [23]

As we have seen in the study of the equations, for the estimation of the position, it is necessary to calibrate two parameters: the wheels ratio and the distance between them. With the calibration, these two parameters will be adjusted. The first one is related to the distance travelled by the wheels, and the second to the turns.

As it is said in the next text [23], this method offers a clear advantage in that it allows modelling and compensation of mechanical inaccuracies, such as the variations in wheel diameters. However, it is important to note that this method specifically addresses systematic errors and does not provide estimation for non-systematic errors.

The calibration process that it is going to be done in this project, consists in three steps, that should be done order, in first place the distance must be calibrated, then the deviations and for last the turns.

The following tests had been done before tried to implement the mobile robot in the whole factory. In fact, the robot did not follow any line to carry out its actions, as it does in the DMF. Instead, it was given straightforward commands to either move straight ahead or make a turn. The purpose of this approach was to test the reliability of the wheel odometry method for our robot. It's well-known that the robot's caterpillar mechanism (tracks) can cause inaccuracies and potential slippage, which can result in complications, so maybe this method would not have been the best.

1. First Test: Distance Calibration.

In this first test, the robot must be programmed to move in a straight line. The length of the line must be known. In our case, we have measured the straight line with the meter, as can be seen in the Figure 10.



Figure 10. Distance and Deviation Calibration.

With this test, the diameter of the wheels is adjusted, so that in combination with the servo angles readings we can achieve an accurate measurement for the robot's travel along the line that closely approximates the actual value.

Initially, we begin with a rough radius measurement of 2 centimetres.

The robot will move 61.1centimetres over the line, when it reaches the ending, we will check the value calculate by the odometry equations. If it exceeds the actual distance of the line, the radius will be reduced, and if the opposite is the case, the radius will be increased.

Table 1. Results of the first calibration test of the robot.

<i>RATIO (m)</i>	<i>REAL DISTANCE (m)</i>	<i>ODOMETRY (m)</i>
0.02	0.611m	0.5908966
0.021	0.611m	0.61598594
0.0205	0.611m	0.61506546
0.0206	0.611m	0.61634677
0.0204	0.611m	0.61125347
0.02045	0.611m	0.60999
0.02047	0.611m	0.6117707
	0.611m	0.61183227
	0.611m	0.61116539

2. Second Test: Deviation Calibration.

In this second test, we calibrate the deviation of the robot following a straight line. In certain cases, when we instruct the robot to move in a straight line, the calculated distance based on odometry may indicate movement in both coordinates, which is not logically possible. This discrepancy arises from slight variations in the wheels, even if their differences are only a few millimetres. As a result, the next course of action involves adjusting the spokes of each wheel to ensure that they are calibrated to move an equal distance.



Figure 11. Calibration Test.

If the robot determines a deviation towards the left, it will be necessary to reduce the radius of the right wheel. Similarly, if it detects a deviation towards the right, the radius of the left wheel should be decreased.

### 3. Third Test: Turns Calibration.

In this third test, two procedures were conducted. The first procedure involved ensuring the accurate adjustment of a 90-degree turn. The second procedure entailed performing a complete calibration square.

In the initial experiment, the robot was directed to move in a straight line for 68.6cm. Subsequently, it was instructed to make a 90-degree turn and continue moving forward for another 68.6cm. This sequence allowed the robot to cover the desired distance in both the x and y directions.

If the robot detects that it has turned less than 90 degrees, it indicates that the parameter L is greater than it should be. Conversely, if it detects that it has turned more than 90 degrees, it suggests that the distance between the axes is smaller than expected.

Table 2. Results of the third calibration test.

$L$ (m)	REAL DISTANCE (cm)	$x$ (cm)	$y$ (cm)	$\theta$ (degrees)
0.235	64.6	74.07	62.295639	81.4465
0.23	64.6	71.0255	62.37	84.1521
0.225	64.6	64.943	63.435	89.54
0.22	64.6	64.233	63.7824	91.233

The experiment is concluded once we observe that the turn made by the robot is 90 degrees or as close to 90 as possible. With  $L=0.225$ , it is obtained the closest value, a turn of 89.54 degrees.

To confirm that this  $L$  value is good, the calibration square test called “Uni-directional square path” will be conducted as we can see in [3]. This test involves directing the robot to follow a unidirectional path in the form of a square, whose sides measure 64.6cm.

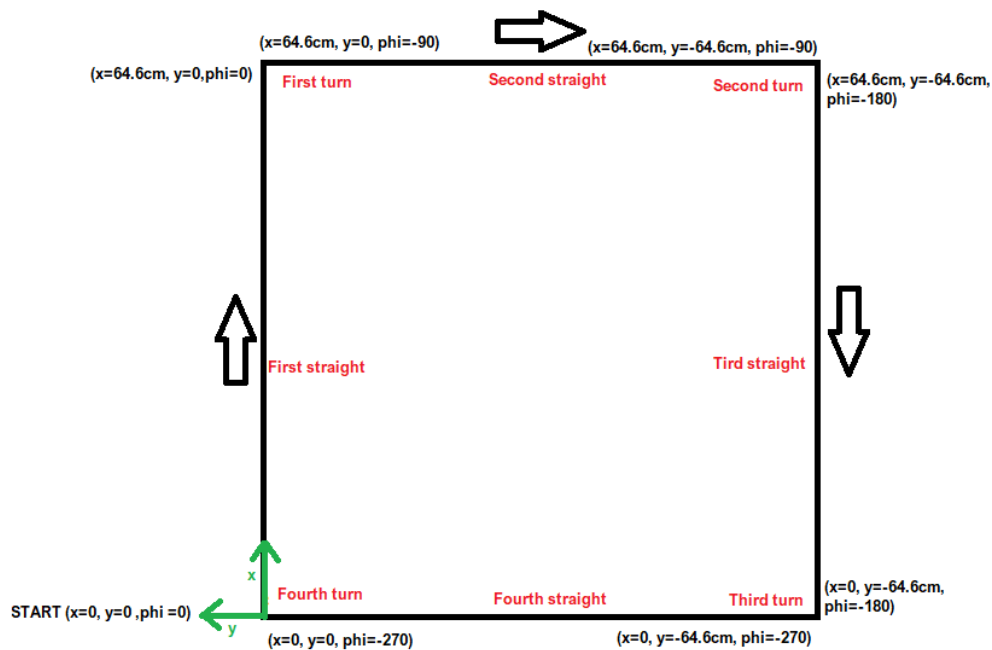


Figure 12. Unidirectional square path our robot must do.

Table 3. Results of the third calibration test of the robot.

	FIRST TRY			SECOND TRY		
	<i>x (cm)</i>	<i>y (cm)</i>	$\theta$ (degrees)	<i>x (cm)</i>	<i>y (cm)</i>	$\theta$ (degrees)
<i>First straight</i>	64.8	0.9699	-0.0959	64.39048	1.0186	0.3154
<i>First turn</i>	65.7532	0.025158	-91.5732	65.24	0.109377	-90.12411
<i>Second straight</i>	63.8818	64.3792	-91.8162	63.4338	-64.1658	-91.61666
<i>Second turn</i>	63.5354	-65.8057	-180.54	63.1587	-65.7450	-177.5026
<i>Third straight</i>	-1.0519	-65.3603	-180.331	-1.68946	-66.9762	-177.4706
<i>Third turn</i>	-2.4117	-63.9541	-265.7682	-2.96938	-66.9513	-261.6251
<i>Fourth straight</i>	-8.7919	-1.69519	-265.6892	-8.7031	-1.91513	-262.2178
<i>Fourth turn</i>	-7.5708	-0.12861	-349.2305	-8.612	-2.3862	-345.2011

As it can be seen in the experiment, initially, the odometry results closely matched the expected values. However, as the robot performed the square, the values began to deviate further and further away, reaching approximately -8 cm in x and -2 cm in y coordinates, whereas they should have been closer to 0.

A similar divergence was observed in the angle, which should have been 360 degrees at the end. It is evident that these discrepancies arise due to various disturbances, such as wheel slippage, and the errors accumulate progressively over time.

Despite these challenges, the obtained results are still considered reasonably good. Therefore, the utilization of this method to calculate the orientation and position of our robot is deemed valid.

### 3 Implementation in the DMF

This chapter will explain how the implementation has been carried out within the DMF. It will talk about how the odometric equations have been implemented in Arduino, which variables have been chosen, etc. It will also talk about how the coordinates are sent to thing Worx.

To begin with, a decision was made regarding the placement of the coordinate origin, which was determined to be at the sorting station. This choice was because the robot departs from this station at the beginning. Once this origin was established, all the other stations would be positioned using coordinates relative to this origin. In essence, all coordinates would be referenced in relation to this chosen origin. In the following picture it can be seen where is located:

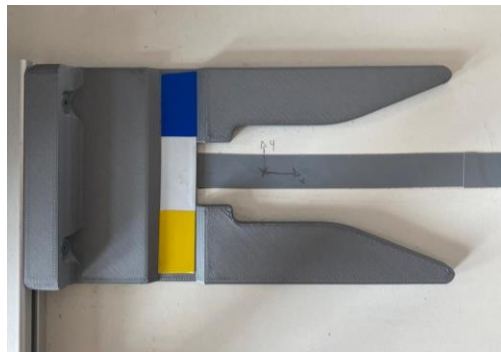


Figure 13. Origin of coordinates in the sorting station.

It coincides with where the centre of mass of the robot would be located if the robot is placement in the station.

#### 3.1 Arduino

Each time the robot detects which station it is at, it will obtain the corresponding values of the station coordinates, so that each time it must move from that station to another station, the odometry will start to be calculated from those coordinates. As we have said before, the other stations coordinates were measured from the origin (in the sorting station).

In the “int ExtendedLineFollower::readStationColour()” function, the robot determines in which station is at, so in this function it is observed what coordinates and angle values it obtains.

When the robot starts at the sorting station, the coordinates it will have would be:

$$x = 0, y = 0, phi = 180$$

The explanation why phi will be equal to 180, is due to the fact that when the robot has to leave that station it will go backwards, having to make a half turn, and as the direction of the x-axis is as shown in the Figure 13 it must be taken into account that after turning half a turn the degrees would be 0 if it were perfectly aligned with the black line to start following it.

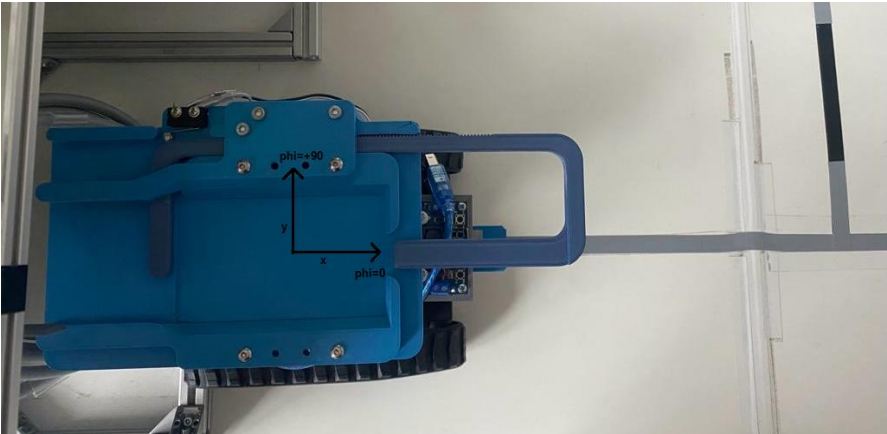


Figure 14. Position before leaving the sorting station.

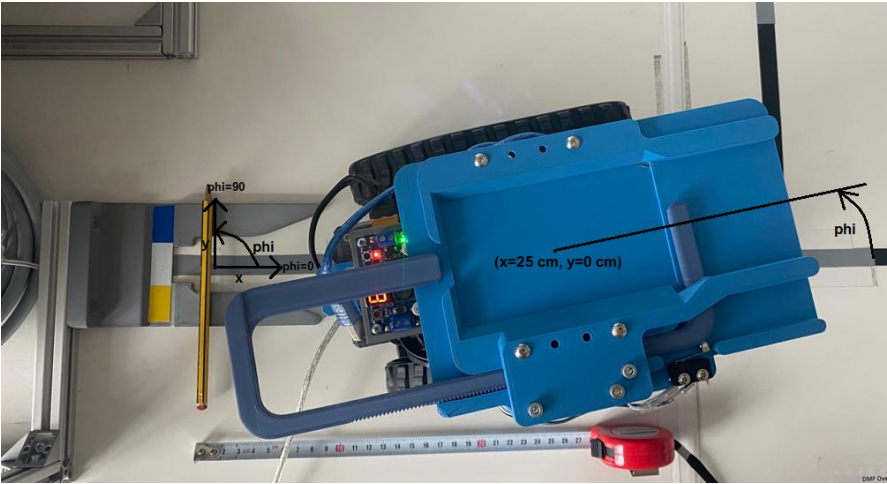


Figure 15. Position after leaving sorting station.

The odometry values calculated were the following:



```
16:52:37.119 -> LeaveStation_success
16:52:37.119 -> Odom_x= 24.68
16:52:37.160 ->
16:52:37.160 -> Odom_y= 0.01
16:52:37.160 ->
16:52:37.160 -> Odom_p= 12.42
16:52:37.160 ->
```

Figure 16. Odometry values after leaving sorting station.

The calculated coordinates are quite accurate.

As we have explained for the first station, in the others are done in the same way.

For example, for the quality check station, the coordinates are measured from the origin as it can be seen:

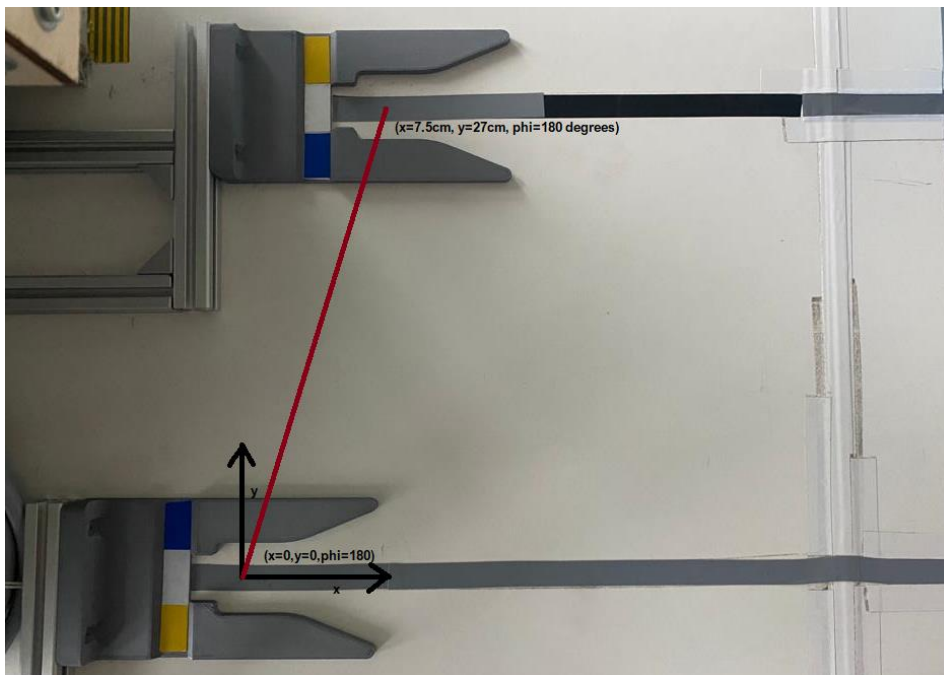


Figure 17. Quality check station coordinates.

The coordinates of the other stations can be seen in the Appendix B at the end of this thesis.

Inside this function, it has also been incorporated that the servos reset their angles and set them to 0, so that if the robot has just arrived from another station, it will not accumulate errors in the calculation of the odometry on its next journey.

The implementation of the odometric equations in the Arduino code is done as seen in section 2.3.1 but considering some specific details.

To begin with, it was taken into account that each time the program execution is restarted, the servos would have to be initialised so that they start at angle 0. In order to do this, the “mysmartservo.setZero(servold)” function of the make block library is called.

The odometry shall be calculated every certain period, which should be small in order to make the sampling as accurate as possible. Our sampling period is of 5000 microseconds.

So, when the time period elapses, the new servo angles will be obtained with the “mysmartservo.getAngleRequest(servold)” make block function. As we have already seen in the equation calculations section, the odometry is calculated from the small increments of the wheels, so the difference between the new angle obtained and the previous one is calculated also each time.

Then, the linear displacements of each wheel are calculated, by multiplying each wheel radio with its corresponding angle increment. We had to consider that the left servo (if we look at the robot from the front) is placed upside down, so if the robot moves forward the angles will be counted in negative, so we add a minus sign to its linear increment to be able to use it in the equations.

Finally, the equations can already be used to obtain the x and y coordinates, and the orientation phi.

## 3.2 Thing Worx

Thing Worx is used for IoT application development and deployment. It provides a platform for connecting, managing, and analysing data from devices and sensors. It utilizes modelling approach instead of traditional coding, allowing content developers to prioritize agility and application composition. [24]

In Arduino code, the coordinates are sent as follows:

```
currenttime = millis();

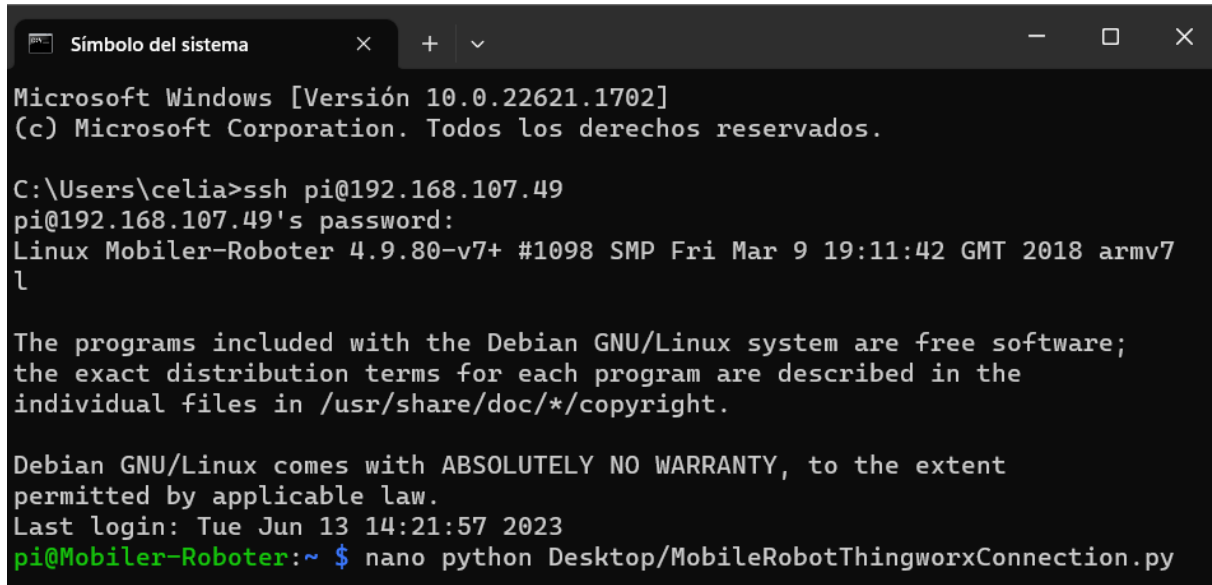
if(currenttime-previousime>=interval)
{
  previousime=currenttime;

  serialPrint("Odom_x= ");serialPrint(100*x);serialEnd();
  serialPrint("Odom_y= ");serialPrint(100*y);serialEnd();
  serialPrint("Odom_p= ");serialPrint(phi*180/3.14159);serialEnd();
}
```

Figure 18. Arduino code to send the coordinates.

It was necessary to use a timer, so that not all coordinates are sent to the server, as it is a large amount of data causing a long delay in the update on the Thing Worx server. Therefore, the coordinate values are only sent when a certain time interval elapses, which is 2.5 seconds.

In order to get the values of x, y and phi to Thing Worx, the code written in python, which is used to connect Thing Worx with the mobile robot, had to be modified.



```
Símbolo del sistema x + v - □ ×
Microsoft Windows [Versión 10.0.22621.1702]
(c) Microsoft Corporation. Todos los derechos reservados.

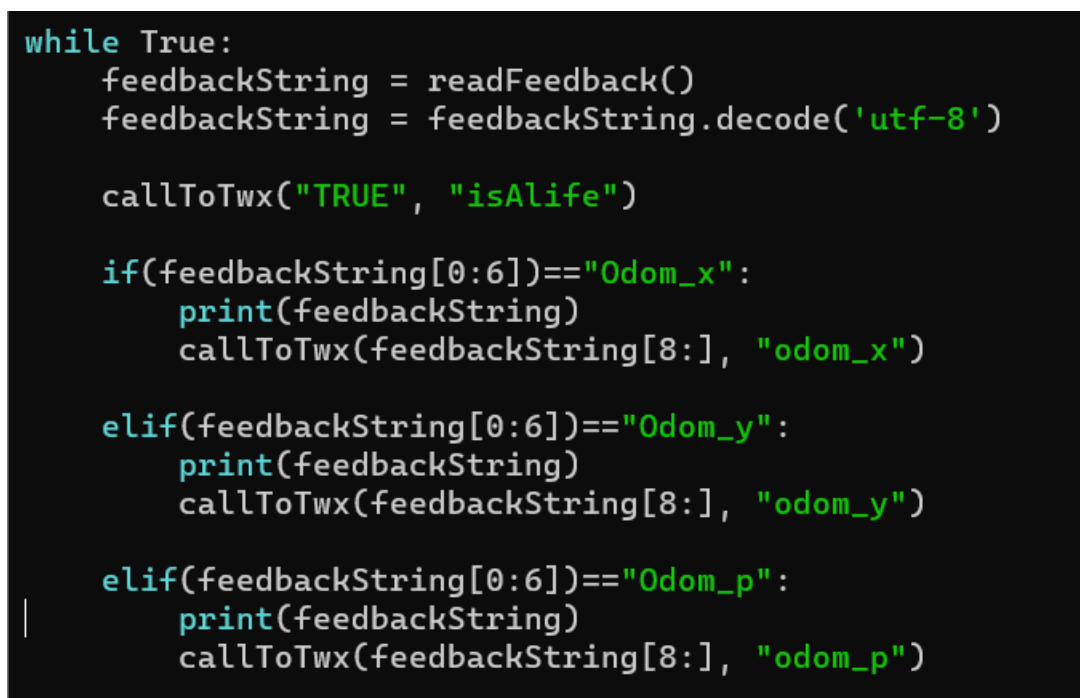
C:\Users\celia>ssh pi@192.168.107.49
pi@192.168.107.49's password:
Linux Mobiler-Roboter 4.9.80-v7+ #1098 SMP Fri Mar 9 19:11:42 GMT 2018 armv7
l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Jun 13 14:21:57 2023
pi@Mobiler-Roboter:~$ nano python Desktop/MobileRobotThingworxConnection.py
```

Figure 19. Connection to Raspberry Pi.

To the previous python code, is has been included:



```
while True:
    feedbackString = readFeedback()
    feedbackString = feedbackString.decode('utf-8')

    callToTwx("TRUE", "isALife")

    if(feedbackString[0:6])=="Odom_x":
        print(feedbackString)
        callToTwx(feedbackString[8:], "odom_x")

    elif(feedbackString[0:6])=="Odom_y":
        print(feedbackString)
        callToTwx(feedbackString[8:], "odom_y")

    elif(feedbackString[0:6])=="Odom_p":
        print(feedbackString)
        callToTwx(feedbackString[8:], "odom_p")
```

Figure 20. Python code for send the coordinates I.

```

elif len(feedbackString) > 5:
    print (feedbackString)
    callToTwx(feedbackString, "orderStatus")

order = readFromTwx()

callToTwx("TRUE", "isAlife")

if(order):
    print(order)
    arduino.write(bytes(str(order)+'\n', 'utf-8'))
    time.sleep(0.04)
    callToTwx("", "order")
    print("order set to NULL!")
callToTwx("TRUE", "isAlife")
time.sleep(1)

```

Figure 21. Python code for send the coordinates II.

The 3 variables corresponding to the coordinates were created:

<input type="checkbox"/> # odom_p		0	0	degrees
<input type="checkbox"/> # odom_x		0	0	cm
<input type="checkbox"/> # odom_y		0	0	cm

Figure 22. Coordinates variables in Thing Worx.

These variables are numeric and logged.

### 3.3 Tests and Results

During the initial testing phase within the miniature factory, an issue arose when attempting to command the robot to move between stations using the "p" command and following the shortest path. Although the path calculation was accurate, the line follower functionality stopped to work correctly. Efforts were made to rectify this error; however, due to time constraints, it was not feasible to fix the problem with the robot's function. As a result, the odometry implementation tests within the factory were conducted solely by instructing the robot to follow the lines, so when the robot encountered a branch with multiple paths to follow, as the route had not been calculated, it continued straight ahead. Therefore, it may appear that the only implemented route between stations is the one from the sorting area to the storage location. To do this, we create a new function in Arduino called "followBlackLine()" that

implements all the code of “follow()” function unless the part when the path is obtained. To call this function the command to introduce is “x”.

The tests were done through the serial port of Arduino because it is quicker than with Thing Worx, but as it could be seen in the previous chapter, the same coordinates that are sending to the Arduino port will be sending to Thing Worx as well.

To initiate our testing, we directed the robot to follow the longest line in the factory, specifically the one stretching from node 9 to the bad-parts station. The objective of this initial test was to observe and evaluate the odometry results obtained, which were calculated using the radius and L values determined during the robot calibration in section 2.3.2. We then compared these results with the expected outcomes.

```
13:45:09.609 -> Moveodometry_success
13:45:09.609 -> Odom_x= 151.36
13:45:09.609 ->
13:45:09.609 -> Odom_y= 14.17
13:45:09.609 ->
13:45:09.609 -> Odom_p= 8.21
```

Figure 23. Odometry results from node 9 to bad-parts I.

The results should be:  $x = 157\text{ cm}$ ,  $y = 0\text{ cm}$ ,  $\phi = 0\text{ degrees}$ .

As anticipated, the obtained results did not come as a surprise. We recognized that the initial calibration was performed without the robot following any line, and when it subsequently followed a line, the robot's movements became more abrupt, potentially affecting the outcomes of the equations. Therefore, it is necessary to readjust the ratios and the value of L directly within the factory setting.

We observe that the robot calculates a wrong displacement to the left, the radius of the right wheel (RL) must be decreased:

```
Moveodometry_success
Odom_x= 150.53

Odom_y= -0.97

Odom_p= -1.59
```

Figure 24. Odometry results from node 9 to bad-parts II.

The y coordinate has a better result when it is closer to 0, but we must increase both so that the value of x increases as well.

Doing several tests, and by modifying the ratios values little by little as explained in the calibration chapter 2.3.2, we finally obtain the following results with these ratios:

$$RR = 0.0199 m$$

$$RL = 0.0201 m$$

Moveodometry_success	Moveodometry_success
Odom_x= 156.49	Odom_x= 156.90
Odom_y= 1.98	Odom_y= 3.55
Odom_p= -2.88	Odom_p= -1.34

Figure 25. Odometry results from node 9 to bad parts III. Left was the first test and right the second.

The obtained results are remarkably close to the expected ones, indicating a good level of accuracy. Nonetheless, it is essential to consider the presence of random factors that can influence our robot's performance, which are beyond our control. As a result, there may be instances where, despite having identical ratios values, the outcomes might deviate more than anticipated from the theoretical expectations. The provided Figure 25 illustrates this point, where both simulations were conducted using the same values, the second simulation has a less favourable result in terms of the y-coordinate.

Now we test that it follows the straight line from the node 8 to the storage station, it will only have x coordinate, in fact the real values it would have:  $x = 117.8 cm, y = 0, phi = 0$

```

Moveodometry_success
Odom_x= 116.88

Odom_y= -0.23

Odom_p= 0.08

```

Figure 26. Odometry results from node 8 to storage.

The results are good as they are very close to the expected.

Then we calibrate the turn (corner) from the end of the grey bifurcation (node 9) to the storage station. It turns at node 8, and arrives at the storage station, starting at 0,0,0:

If we measure from the grey to where the station is, it is about 33 cm approximately. The results must be:  $x = 33 cm, y = 117.8 cm, phi = 90$ .

Moveodometry_success	Moveodometry_success
Odom_x= 35.48	Odom_x= 35.10
Odom_y= 117.09	Odom_y= 117.83
Odom_p= 85.73	Odom_p= 84.95

Figure 27. Odometry results from node 9 to storage.

The results obtained are very close to what we want, the L parameter and the ratios values calibrated for this are:

$$\begin{aligned}RR &= 0.0199 \text{ m} \\RL &= 0.0201 \text{ m} \\L &= 0.223 \text{ m}\end{aligned}$$

Even so, as the different tests are carried out, a small change of the values may be required, so that they give the best possible results.

- **Sorting station to storage station.**

The values the equations must calculate are:  $x = 82.6 \text{ cm}$ ,  $y = 117.8 \text{ cm}$ ,  $\phi = 90 \text{ degrees}$ .

LeaveStation_success	Moveodometry_success
Odom_x= 24.68	Odom_x= 83.54
Odom_y= 0.02	Odom_y= 117.61
Odom_p= -6.58	Odom_p= 88.54

Figure 28. Odometry results from sorting to storage. Test 1.

First, the coordinates obtained after leaving the station are displayed. It should be noted that the calculation of the coordinates in this movement is quite accurate, and they always give values very close to those required.

Then we obtain the values when the robot reaches the storage, we can also confirm that are quite close.

Other test was carried out to confirm the good results:

Moveodometry_success
Odom_x= 81.70
Odom_y= 119.25
Odom_p= 89.07

Figure 29. Odometry results from sorting to storage. Test 2.

- **Storage to sorting station.**

The values the equations must calculate are:  $x = 0 \text{ cm}$ ,  $y = 0 \text{ cm}$ ,  $\phi = -180 \text{ degrees}$ .

LeaveStation_success	Moveodometry_success
Odom_x= 82.71	Odom_x= -8.68
Odom_y= 92.63	Odom_y= 13.82
Odom_p= -126.10	Odom_p= -191.09

Figure 30. Odometry results from storage to sorting. Test 1.

In this particular test, we observed that the obtained results deviated from our expectations. Notably, there was an 8cm difference in the x-coordinate and a 13cm difference in the y-coordinate. Upon examining the final angle phi, we noticed that it had a value of -191, which exceeded the expected range of -180. This discrepancy in angle contributed significantly to the deviation in the coordinates. Consequently, we decided to adjust the value of L by increasing it to 0.224, aiming to reduce the magnitude of rotation calculated by the robot. This illustrates the need for occasional parameter modifications during testing in order to achieve results that align more closely with the desired outcomes.

$$L = 0.224 \text{ m}$$

```

· LeaveStation_success
· Odom_x= 82.48      Odom_x= 1.72
·
· Odom_y= 93.20     Odom_y= -1.91
·
· Odom_p= -116.99   Odom_p= -178.18

```

Figure 31. Odometry results from storage to sorting. Test 2.

The results have improved.  
Another tries:

```

LeaveStation_success      Moveodometry_success
Odom_x= 82.66            Odom_x= -3.47

Odom_y= 92.84           Odom_y= -3.57

Odom_p= -117.34         Odom_p= -179.55

```

Figure 32. Odometry results from storage to sorting. Test 3.

The results are still better than the first time.

- **Bad parts to node 9.**

The values the equations must calculate are:  $x = 41.7 \text{ cm}$ ,  $y = 4 \text{ cm}$ ,  $\phi = -90 \text{ degrees}$ .

Doing the test with the previous L value:

```

LeaveStation_success      Moveodometry_success
Odom_x= 41.61            Odom_x= 34.43

Odom_y= 131.26          Odom_y= 2.49

Odom_p= -121.34         Odom_p= -95.83

```

Figure 33. Odometry results bad parts to node 9. Test 1.

The rotation exceeds the desired amount by a small margin of 5 degrees. Even though this difference may seem insignificant, if the excess rotation persists over an extended period, it results in the accumulation of errors. Consequently, the coordinates exhibit significant variations.



If we try to reduce that angle, by decreasing the L:

$$L = 0.225 \text{ m}$$

LeaveStation_success	Moveodometry_success
Odom_x= 41.25	Odom_x= 41.48
Odom_y= 132.00	Odom_y= 6.46
Odom_p= -137.41	Odom_p= -90.13

Figure 34. Odometry results bad parts to node 9. Test 2.

An improvement in the values obtained can be observed.

Now we are going to show what happens in **Thing Worx** when we order the robot to leave the bad parts station:

Name	Actions	Source	Default Value	Value	Alerts	Category	Additional Info
AccessibleStations				AccessibleStations (4)	0		StringArrayElement
currentPosition				Bad Parts Storage	0		
isAlive				TRUE	0		
moveFromTo_Goal				Set_value	0		
moveFromTo_Start				Set_value	0		
odom_e				90	0		degrees
odom_x				41.7	0		cm
odom_y				157	0		cm
order				Set_value	0		
orderStatus				DetermineStationNumber_success_6	0		
startUpRoutineActive					0		

Figure 35. Thing Worx coordinates in Bad-Parts station.

Name	Actions	Source	Default Value	Value	Alerts	Category	Additional Info
AccessibleStations				AccessibleStations (4)	0		StringArrayElement
currentPosition				Mobile Robot is leaving the station	0		
isAlive				TRUE	0		
moveFromTo_Goal				Set_value	0		
moveFromTo_Start				Set_value	0		
odom_e				-74.46	0		degrees
odom_x				41.69	0		cm
odom_y				132.56	0		cm
order				Set_value	0		
orderStatus				LeaveStation_success	0		
startUpRoutineActive					0		

Figure 36. Thing Worx coordinates after leaving Bad-Parts.

## 4 Conclusion

In conclusion, the utilization of wheel odometry proved to be a valuable method for estimating the robot's position and orientation. Through the calibration process and parameter adjustments, we were able to improve the accuracy of the odometry results in the DMF as can be seen in the previous chapter 3.3.

However, it is important to acknowledge the limitations of wheel odometry, particularly its susceptibility to systematic errors and the accumulation of inaccuracies over time. Examples of this can be seen in the tests performed at the DMF, the movement of the robot's leave station has always given acceptable results, because it was a short distance involved, but as the robot had to traverse longer distances, the accuracy of the odometry results deteriorated. In addition, the deviation in rotations, even by a slight amount, can lead to substantial discrepancies in the final coordinates due to error accumulation. Therefore, it is crucial to carefully monitor and calibrate the odometry system to minimize these errors and ensure reliable position estimation.

As far as non-systematic errors are concerned, it is possible that due to the use of this type of wheels, the robot suffers from slippage, which is a source of errors. The line follower also occasionally exhibits jerky movements, particularly when executing 90-degree turns around corners, making precise calculation more difficult too.

Additionally, it has been determined that the calibration of odometry must be periodically conducted. Optimal values for certain paths may not be suitable for others, necessitating adjustments. Similarly, as the robot's battery level decreases, modifications to the radius and L parameters become necessary. It is speculated that the power reaching the servos could be a contributing factor, because when the battery drains over time, the expected results are no longer obtained.

To summarize, precise results with minimal error margins of a few centimetres are achievable with well-calibrated radius and L values. However, to ensure the adequacy of these values in each simulation, recurring calibration is essential. In fact, it is recommended to perform relevant tests daily to validate the accuracy of the robot's calculated coordinates.

## **5 Points for improvement or expansion in a future**

In relation to odometry, future research could explore the integration of advanced calibration methods to reduce systematic errors. This could involve incorporating additional sensor or machine learning algorithms to refine the parameters.

Combining wheel odometry with other localization techniques, like visual odometry or GPS, helping mitigate the limitations and error accumulation of wheel odometry, to enhance overall accuracy and robustness.

Correct non-systematic errors, implementing error correction mechanisms such as filtering techniques, mitigate the impact of random noise and disturbances on odometry measurements.

Evaluate and study alternatives odometry methods, or other localization systems in general.

Furthermore, future considerations should encompass potential hardware enhancements, such as altering the wheel type, as well as addressing the issue that cause the robot's inability to function properly when employing the Dijkstra algorithm for inter-station movement.

## Bibliography

- [1] 'What is Industry 4.0?' <https://www.epicor.com/en/blog/what-is-industry-4-0/> (accessed Jun. 07, 2023).
- [2] Robotnik, 'Mobile Robots in Industry 4.0: automation & flexibility', *Robotnik*, Jan. 12, 2021. <https://robotnik.eu/mobile-robots-industry/> (accessed Jun. 08, 2023).
- [3] J. Borenstein and L. Feng, 'UMBmark: a benchmark test for measuring odometry errors in mobile robots', presented at the Photonics East '95, W. J. Wolfe and C. H. Kenyon, Eds., Philadelphia, PA, Dec. 1995, pp. 113–124. doi: 10.1117/12.228968.
- [4] M. O. A. Aqel, M. H. Marhaban, M. I. Saripan, and N. Bt. Ismail, 'Review of visual odometry: types, approaches, challenges, and applications', *SpringerPlus*, vol. 5, no. 1, p. 1897, Oct. 2016, doi: 10.1186/s40064-016-3573-7.
- [5] P. K. Panigrahi and S. K. Bisoy, 'Localization strategies for autonomous mobile robots: A review', *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 34, no. 8, pp. 6019–6039, Sep. 2022, doi: 10.1016/j.jksuci.2021.02.015.
- [6] S. A. Rahok and O. Koichi, 'Odometry correction with localization based on landmarkless magnetic map for navigation system of indoor mobile robot', in *2009 4th International Conference on Autonomous Robots and Agents*, Feb. 2009, pp. 572–577. doi: 10.1109/ICARA.2000.4803938.
- [7] B. Bischoff, D. Nguyen-Tuong, F. Streichert, M. Ewert, and A. Knoll, 'Fusing vision and odometry for accurate indoor robot localization', in *2012 12th International Conference on Control Automation Robotics & Vision (ICARCV)*, Dec. 2012, pp. 347–352. doi: 10.1109/ICARCV.2012.6485183.
- [8] J. Biswas and M. Veloso, 'Depth camera based indoor mobile robot localization and navigation', in *2012 IEEE International Conference on Robotics and Automation*, May 2012, pp. 1697–1702. doi: 10.1109/ICRA.2012.6224766.
- [9] B.-S. Choi, J.-W. Lee, J.-J. Lee, and K.-T. Park, 'A Hierarchical Algorithm for Indoor Mobile Robot Localization Using RFID Sensor Fusion', *IEEE Trans. Ind. Electron.*, vol. 58, no. 6, pp. 2226–2235, Jun. 2011, doi: 10.1109/TIE.2011.2109330.
- [10] K. Lee, W. Chung, and K. Yoo, 'Kinematic parameter calibration of a car-like mobile robot to improve odometry accuracy', *Mechatronics*, vol. 20, no. 5, p. 582, 2010.
- [11] 'Me Auriga · GitBook'. <http://docs.makeblock.com/diy-platform/en/electronic-modules/main-control-boards/me-auriga.html> (accessed May 15, 2023).
- [12] 'Instant centre of rotation', *Wikipedia*. Sep. 28, 2022. Accessed: May 21, 2023. [Online]. Available: [https://en.wikipedia.org/w/index.php?title=Instant\\_centre\\_of\\_rotation&oldid=1112781379](https://en.wikipedia.org/w/index.php?title=Instant_centre_of_rotation&oldid=1112781379)
- [13] A. B. Azcón and A. O. Pujol, 'Análisis Y Diseño Del Control De Posición De Un Robot Móvil Con Tracción Diferencial'.
- [14] 'Fig. 1: Kinematic model of mobile robot', *ResearchGate*. [https://www.researchgate.net/figure/Kinematic-model-of-mobile-robot\\_fig1\\_285739464](https://www.researchgate.net/figure/Kinematic-model-of-mobile-robot_fig1_285739464) (accessed May 21, 2023).
- [15] V. Leonov, 'Steering mechanism for a toy vehicle', US7938709B2, May 10, 2011 Accessed: Jun. 08, 2023. [Online]. Available: <https://patents.google.com/patent/US7938709B2/en>
- [16] 'Figure 2.1 Ackermann steering mechanism This is the diagram for the...', *ResearchGate*. [https://www.researchgate.net/figure/Ackermann-steering-mechanism-This-is-the-diagram-for-the-Ackermann-steering-mechanism\\_fig1\\_346561736](https://www.researchgate.net/figure/Ackermann-steering-mechanism-This-is-the-diagram-for-the-Ackermann-steering-mechanism_fig1_346561736) (accessed May 21, 2023).
- [17] 'Fig. 5 Tricycle configuration in which the front wheel is steered and...', *ResearchGate*. [https://www.researchgate.net/figure/Tricycle-configuration-in-which-the-front-wheel-is-steered-and-driven\\_fig18\\_264540298](https://www.researchgate.net/figure/Tricycle-configuration-in-which-the-front-wheel-is-steered-and-driven_fig18_264540298) (accessed May 21, 2023).

- [18] S. A. S. Mohamed, M.-H. Haghbayan, T. Westerlund, J. Heikkonen, H. Tenhunen, and J. Plosila, 'A Survey on Odometry for Autonomous Navigation Systems', *IEEE Access*, vol. 7, pp. 97466–97486, 2019, doi: 10.1109/ACCESS.2019.2929133.
- [19] D. Scaramuzza and F. Fraundorfer, 'Visual Odometry [Tutorial]', *IEEE Robot. Autom. Mag.*, vol. 18, no. 4, pp. 80–92, Dec. 2011, doi: 10.1109/MRA.2011.943233.
- [20] R. González, F. Rodríguez, and J. L. Guzmán, 'Robots Móviles con Orugas Historia, Modelado, Localización y Control', *Rev. Iberoam. Automática E Informática Ind. RIAI*, vol. 12, no. 1, pp. 3–12, Jan. 2015, doi: 10.1016/j.riai.2014.11.001.
- [21] J. Valencia, 'Modelo Cinemático De Un Robot Móvil Tipo Diferencial y Navegación a Partir De La Estimación Odométrica', *Scientia*, Accessed: May 25, 2023. [Online]. Available: [https://www.academia.edu/55496400/Modelo\\_Cinem%C3%A1tico\\_De\\_Un\\_Robot\\_M%C3%B3vil\\_Tipo\\_Diferencial\\_y\\_Navegaci%C3%B3n\\_a\\_Partir\\_De\\_La\\_Estimaci%C3%B3n\\_Odom%C3%A9trica](https://www.academia.edu/55496400/Modelo_Cinem%C3%A1tico_De_Un_Robot_M%C3%B3vil_Tipo_Diferencial_y_Navegaci%C3%B3n_a_Partir_De_La_Estimaci%C3%B3n_Odom%C3%A9trica)
- [22] C. Cuánticos, 'Robótica: Estimación de posición por odometría', *Cuentos Cuánticos*, Dec. 15, 2011. <https://cuentoscuanticos.com/2011/12/15/robotica-estimacion-de-posicion-por-odometria/> (accessed Jun. 13, 2023).
- [23] F. Azizi and N. Houshangi, 'Mobile Robot Position Determination', in *Recent Advances in Mobile Robotics*, A. Topalov, Ed., InTech, 2011. doi: 10.5772/26820.
- [24] 'Getting Started with ThingWorx'. [https://support.ptc.com/help/thingworx\\_hc/thingworx\\_8\\_hc/en/index.html#page/ThingWorx/Help/Getting\\_Started/GettingStarted.html](https://support.ptc.com/help/thingworx_hc/thingworx_8_hc/en/index.html#page/ThingWorx/Help/Getting_Started/GettingStarted.html) (accessed Jun. 11, 2023).
- [25] 'Smart Servo MS-12A · GitBook'. <http://docs.makeblock.com/diy-platform/en/electronic-modules/motors/smart-servo-ms-12a.html> (accessed May 15, 2023).
- [26] Makeblock, 'Makeblock Library v3.27'. Apr. 22, 2023. Accessed: May 16, 2023. [Online]. Available: <https://github.com/Makeblock-official/Makeblock-Libraries>

## List of Figures

Figure 1: Mobile Robot in the DMF.	6
Figure 2. Components of the Me Auriga microcontroller (modified by [11]).	10
Figure 3. Diferential configuration. (modified by [14])	12
Figure 4. Ackerman configuration. (modified by [16])	12
Figure 5. Tricycle configuration. (modified by [17])	13
Figure 6. Synchronous configuration with 3 steering wheels. (modified [13])	13
Figure 7. Omnidirectional configuration. (modified by [13])	14
Figure 8. Profile view of our mobile robot. 4 wheels can be seen, the bottom right wheel is the drive wheel and the other 3 are fixed.	17
Figure 9. Location of the robot in the cartesian plane. (modified by [21]).	18
Figure 10. Distance and Deviation Calibration.	21
Figure 11. Calibration Test.	23
Figure 12. Unidirectional square path our robot must do.	24
Figure 13. Origin of coordinates in the sorting station.	26
Figure 14. Position before leaving the sorting station.	27
Figure 15. Position after leaving sorting station.	27
Figure 16. Odometry values after leaving sorting station.	28
Figure 17. Quality check station coordinates.	28
Figure 18. Arduino code to send the coordinates.	29
Figure 19. Connection to Raspberry Pi.	30
Figure 20. Python code for send the coordinates I.	30
Figure 21. Python code for send the coordinates II.	31
Figure 22. Coordinates variables in Thing Worx.	31
Figure 23. Odometry results from node 9 to bad-parts I.	32
Figure 24. Odometry results from node 9 to bad-parts II.	32
Figure 25. Odometry results from node 9 to bad parts III. Left was the first test and right the second.	33
Figure 26. Odometry results from node 8 to storage.	33
Figure 27. Odometry results from node 9 to storage.	33
Figure 28. Odometry results from sorting to storage. Test 1.	34
Figure 29. Odometry results from sorting to storage. Test 2.	34
Figure 30. Odometry results from storage to sorting. Test 1.	34
Figure 31. Odometry results from storage to sorting. Test 2.	35
Figure 32. Odometry results from storage to sorting. Test 3.	35
Figure 33. Odometry results bad parts to node 9. Test 1.	35
Figure 34. Odometry results bad parts to node 9. Test 2.	36
Figure 35. Thing Worx coordinates in Bad-Parts station.	36
Figure 36. Thing Worx coordinates after leaving Bad-Parts.	36
Figure 37. Station coordinates.	46

## List of Tables

Table 1. Results of the first calibration test of the robot.	22
Table 2. Results of the third calibration test.	24
Table 3. Results of the third calibration test of the robot.	25

## List of Abbreviations

DMF	Digital Miniature Factory
IoT	Internet of Things
AI	Artificial Intelligence
ICR	Instantaneous Centre of Rotation
WO	Wheel Odometry
VO	Visual Odometry
LO	Laser Odometry
IO	Inertial Odometry
RO	Radar Odometry
MEMS	Micro-Electro-Mechanical System
IMUs	Inertial Measurement Units
CW	Continuous Wave
FMCW	Frequency Modulated-Continuous Wave



## A Appendix: Some Arduino functions.

Some arduino functions created and others that were modified from existing ones are shown.

```
void ExtendedLineFollower::period(){

    now = micros();//devuelve microseg
    if(zero==true)//each time we reset, the servos will set their angles at zero
    {
        this->mysmartservo.setZero(servoIdLeft);
        this->mysmartservo.setZero(servoIdRight);

        lastAngL=0;
        lastAngR=0;

        zero=false;
    }

    if(now>lastTime+samplingPeriod)
    {
        dt=now-lastTime;//diferential of time

        //current angles in grades
        angL = this->mysmartservo.getAngleRequest(this->servoIdLeft);
        angR = this->mysmartservo.getAngleRequest(this->servoIdRight);

        //differential of angles in radians
        dif_angL = (angL-lastAngL)*3.14159/180;
        dif_angR = (angR-lastAngR)*3.14159/180;

        lastTime = now;
        lastAngL = angL;
        lastAngR = angR;

        odometry();
    }
}

void ExtendedLineFollower::odometry(){

    deltaR= RR*dif_angR;
    deltaL= -(RL*dif_angL);

    phi= phiprev + ((deltaL-deltaR)/L);

    x= xprev + cos(phi)*((deltaL + deltaR)/2);
    y= yprev + sin(phi)*((deltaL + deltaR)/2);

    phiprev= phi;
    xprev = x;
    yprev = y;

}
```

---

```

bool ExtendedLineFollower::moveStrict(int angle, int speedLeft, int speedRight) {
    //Wird beim ersten ausfuehren aufgerufen

    if (!this->strictMovement) {
        this->debugStart();this->debug("Set new angle");debugEnd();
        //this->mysmartservo.setZero(this->servoIdLeft);

        int init_angle_left = this->mysmartservo.getAngleRequest(this->servoIdLeft);
        int init_angle_right= this->mysmartservo.getAngleRequest(this->servoIdRight);
        this->result_angle_left = angle+init_angle_left;
        this->result_angle_right = angle+init_angle_right;

        this->strictMovement=true;
    }
    int current_angle_left = this->mysmartservo.getAngleRequest(this->servoIdLeft);
    this->debugStart();this->debug("Result Angle Left: ");debug(this->result_angle_left);debugEnd();
    this->debugStart();this->debug("Actual Angle Left: ");debug(current_angle_left);debugEnd();
    this->pwmMove(speedLeft, speedRight);
    //Ueberpruefung ob der Winkel erreicht wurde
    //checking if the angle has been reached, when it occurs, false is returned
    if (angle<0 && current_angle_left<result_angle_left|| angle>0 && current_angle_left>result_angle_left) {
        this->stopServos();
        this->strictMovement=false;
        return false;
    }
    return true;
}

bool ExtendedLineFollower::areAllGray(LineFollowerState state){//CHANGED FUNCTION
    bool allGray = false;

    if((LF_SENSOR_LEFT)&&(state.colours[0] == COLOUR_GRAY)){
        // Serial.print("gray1");
        if((LF_SENSOR_LEFT_MIDDLE)&&(state.colours[1] == COLOUR_GRAY)){

            // Serial.print("gray2");
            allGray=true;
        }
    }
    if((LF_SENSOR_RIGHT)&&(state.colours[3] == COLOUR_GRAY)){
        //Serial.print("gray4");
        if((LF_SENSOR_RIGHT_MIDDLE)&&(state.colours[2] == COLOUR_GRAY)){
            //Serial.print("gray3");
            allGray=true;
        }
    }

    return allGray;//if it is true means that the robot is in a branch
}

```

# B Appendix: Station coordinates.

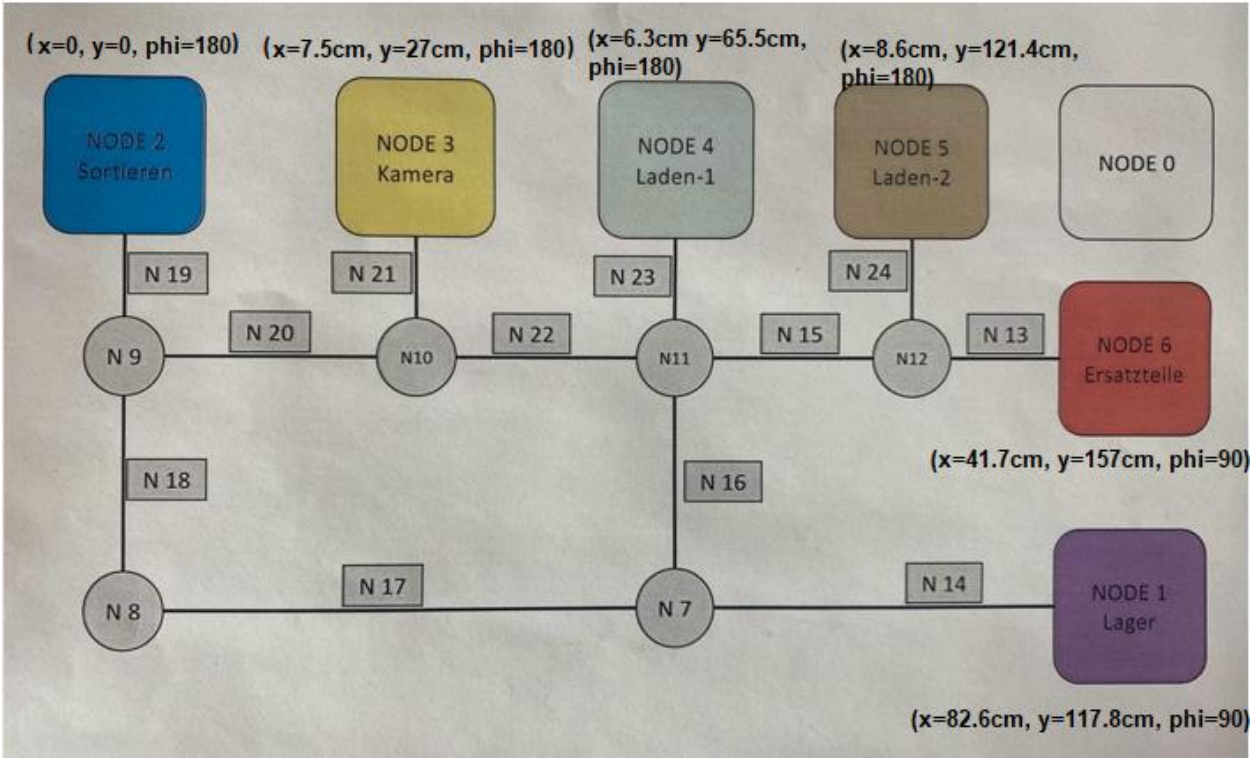


Figure 37. Station coordinates.