Journal of Mechatronics Engineering

Comparative analysis of localization methods for autonomous mobile robots using Robot Operating System 2

Antônio Sávio Silva Oliveira[1], Josias Guimarães Batista[1]

[1]Graduate Program in Telecommunications Engineering (PPGET),
Federal Institute of Ceará - IFCE, Treze de Maio Av., 2081, 60040-531, Fortaleza, Ceará, Brazil.
<savio.oliveira@ifce.edu.br>, <josiasbatista@ifce.edu.br>

DOI: 10.21439/jme.v8i1.122

Abstract. Localization is a fundamental requirement for mobile robots. In order to navigate autonomously and perform tasks, a robot must accurately estimate its position within the environment over time. This work aims to analyze and compare various mobile robot localization methods identified during the literature review, including Monte Carlo Localization (MCL), Adaptive Monte Carlo Localization (AMCL), Sensor Fusion, and a combined method of AMCL with Sensor Fusion, all implemented using the Robot Operating System (ROS 2). The study was carried out in the Webots simulator, with the algorithms developed in Python. Each method was evaluated in terms of efficiency (location accuracy) and performance (hardware resource consumption). The combined method of AMCL with Sensor Fusion achieved the best performance in terms of position accuracy, with a root mean squared error (RMSE) of 4–5 cm and an R² score ranging from 95.10% to 99.79% in Scenario 01, and from 68.20% to 89.65% in Scenario 02. The Sensor Fusion method ranked second, with an average error of 4–7 cm and an R² score of 94.22% to 99.69% in Scenario 01, and 52.27% to 81.42% in Scenario 02. Regarding hardware usage, Sensor Fusion showed the lowest resource consumption, using around 17% of CPU and 32 MB of RAM, followed by AMCL, which used 21% of CPU and 40 MB of RAM. The main contributions of this work include: the application and evaluation of different localization techniques in specific simulation scenarios, allowing for a comparative study; the use of ROS 2 and the public availability of the developed algorithms and results in a GitHub repository, supporting further studies in Webots simulation, ROS 2, and robot localization techniques.

Keywords: Robot Operating System. Location. Mobile robots. Autonomous robots.

1 Introduction

The constant modernization of the industry and its production techniques combined with industry 4.0 technologies[1] and mainly competitive issues show the need to adapt so-called "traditional" factories and means of production to reliable and more economical methods (ODEBRECHT et al., 2020). Given this, one can reflect on the advantages of replacing traditional equipment and robots that are large, fixed and dependent on an operator with modern mobile robots.

According to the paper by (FERREIRA, 2020) mobile robots are more versatile because they do not need to be fixed to work units and can be used for tasks that do not have geographic restrictions. Robots move using legs or wheels, they can be autonomous or not, it is possible to program them at a low or high level and they sense the environment around them through sensors.

In paper from (PIO; CASTRO; CASTRO JÚNIOR, 2006) a mobile robot can be defined as an autonomous agent capable of extracting information from the environment and using this knowledge of the world to move safely in a meaningful and intentional way, acting and executing tasks. For the robot to be able to perform these tasks, it is necessary to use sensors (devices that allow the robot to collect information from the environ-

[1]Industry 4.0 is a concept that represents industrial automation and the integration of different technologies such as artificial intelligence, robotics, internet of things and cloud computing with the aim of promoting the digitalization of industrial activities, improving processes and increasing productivity.

ment), an analogy can be made with the human being and say that the sensor for the robot is like one of the senses for humans (vision, hearing, smell, etc.), in other words, it is through the sensor that the robot can capture information from the environment.

Something essential for a mobile robot is the ability to locate itself in the environment. Whenever it is necessary to move from one point to another, the robot must be able to detect obstacles, avoid them and achieve its objective. If the environment is known, this problem can be summarized as trajectory planning, whereas if the environment is unknown, the robot must act reactively to what is detected through the sensors (MARCHI et al., 2001).

According to (COUSINS, 2010) due to the complexity of modern autonomous mobile robots, it is necessary to simplify systems integration and have a set of tools that can manage this complexity to facilitate robot development. ROS (Robot Operating System) provides functionality for hardware abstraction, device drivers, inter-process communication across multiple machines, testing and visualization tools, among others. Furthermore, a mobile robot application using ROS can be easily adapted to work with other types of robots.

Therefore, this paper aims to carry out a review of the existing literature on the methods used to localize mobile robots with ROS, implement some in a simulator to obtain results that can describe their performance in a quantitative and qualitative way and compare the results obtained to provide an overview for new researchers in the area and serve as auxiliary material in teaching mobile robotics.

1.1 Objectives and Contributions

The main objective of this work is to analyze and compare mobile robot localization methods using ROS, in addition to identifying the most used localization methods with ROS through a literature review and selecting 03 (three) of them for further study, combining two methods aiming to improve position accuracy, implement the selected techniques using ROS in a simulator, and analyze the results obtained when implementing these techniques.

Follow are highlight the contributions of this paper:

- The use of different localization techniques for mobile robots, applied in two different scenarios in order to enable a comparative study between them.
- The use of ROS, which is a widely used *middleware* for mobile robotics due to its modularity and ability to abstract a complex problem into smaller steps and carry out communication between the robot's different sensors.
- The availability of algorithms and results in a GitHub repository [2], which can be used for studies of the simulator used, ROS2 and techniques of localization in robotics.

1.2 Organization of the Paper

This paper is organized as follows. Section 2 presents the theoretical foundation. Section 3 presents the materials and methods for developing the research. Section 4 presents the results obtained with the implementation of localization methods for mobile robots using ROS 2 in the Webots simulator are presented and discussed. Section 5 presents discussions and comparisons of the implemented methods. Finally, Section 6 presents the research conclusions as well as suggestions for future work.

2 Theoretical Foundations

This section presents some theoretical foundations such as mobile robotics, ROS and main localization techniques used for mobile robots.

2.1 Mobile Robotics

An autonomous mobile robot is a system that operates in a partially unknown or unstructured environment. This means that the robot must be able to navigate even in the presence of noise and avoid possible obstacles (ALATISE; HANCKE, 2020). Robots can sense the environment and make decisions that allow them to perform their tasks efficiently, effectively and safely. Due to these characteristics, the deployment of these robots has occurred in various fields such as personal homes, industries, hospitals, schools, exploration of unknown lands, rescue operations and many other areas (NILOY et al., 2021).

Mobile robot architectures can be classified according to some characteristics such as structure (centralized and distributed), reasoning (reactive, deliberative and hybrid), decomposition and encapsulation (based on behavior and based on functional modules) (NAKHAEINIA et al., 2011).

In mobile robots with a centralized architecture, all decisions are made locally, that is, to make any decision, data from the robot's sensors will be analyzed; it

[2]<https://github.com/savio-dev/analise_localizacao_ros>

acts autonomously based solely and exclusively on the information collected during its foray into the environment. (TZAFESTAS, 2018).

According to (SILVA; ROCHE; KONDOZ, 2018), a typical autonomous mobile robot/vehicle is composed of three main technological components: a mapping system that is responsible for detecting and understanding objects in the surrounding environment; a location system with which the robot knows its current location at any time, and the third component responsible for decision making. Other authors such as (PANIGRAHI; BISOY, 2021) and (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011) consider four components: perception, localization, cognition and movement control.

In the perception module, the robot extracts significant data by interpreting its sensors. In the localization phase, the robot estimates its current location in the environment using information obtained from the sensors. In the cognition module, the robot must perform some type of calculation to plan the route it should take, and the chosen route needs to avoid obstacles along the way. The motion control phase allows the robot to achieve its desired trajectory by modifying its motor outputs. This model with four modules is represented in Fig. 1.

It can be seen from Fig. 1 that these four phases do not occur just once, but rather cyclically, while the robot is in operation, whenever it finishes its planned actions it returns to the phase of perception, in which you capture information from the environment and can use it to plan your next steps or check whether you have already achieved your goal. These two views can be considered equivalent, the only difference being the level of detail of each phase. It is clear that the mobile robot constantly needs to know its current location to be able to navigate accurately in the environment and carry out the planned route. According to (NILOY et al., 2021), navigation offers the most difficult challenges for mobile robots. This can be described in four steps:

1. Environmental Modeling (Mapping);
2. Location;
3. Path planning;
4. Avoid obstacles.

2.2 Location

Localization is a basic requirement of a mobile robot. To be able to navigate autonomously and carry out its activities, it must accurately understand its position in the environment over time (DOURADO JUNIOR et al., 2019). The localization challenge consists of estimating the position and orientation of the robot through information acquired by sensors and other systems (ALATISE; HANCKE, 2020).

There are many strategies to solve or even avoid this location problem. For example, according to (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011), using a pre-defined solution, you can make a robot go from point A to point B without the robot needing to know its current position. However, this type of solution is highly dependent on the environment, that is, it only applies to a certain environment and if it is necessary to move the robot, it is necessary to carry out all the programming again, which makes it unfeasible for most mobile robot applications.

In (CAMPBELL et al., 2020) some common approaches for robot localization were presented, among which the following can be highlighted:

1. GPS, which works through satellite navigation systems;
2. Odometry, which calculates the robot's position through the use of an encoder that measures the rotation of the wheels;
3. Inertial navigation system (INS), which works in a similar way to odometry, has the advantage of not requiring any initial reference point, however, it is not suitable for carrying out localization over long periods due to accumulation errors, and is often used to complement other localization methods such as vision systems or GPS;
4. Map based location.

According to (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011), if the GPS (Global Positioning System) system had an accuracy in the range of centimeters or millimeters, much of the location problem would be avoided. The GPS would tell the robot its exact position, inside and outside the house and the answer to the question "where am I?"would always be available. However, the accuracy is currently several meters and this is unfeasible for mobile robots, on a human scale or in miniature.

Additionally, GPS does not work indoors or in obstructed areas due to potential obstacles that could block the connection with satellites. But, if we look beyond these limitations, we can see that the robot's location

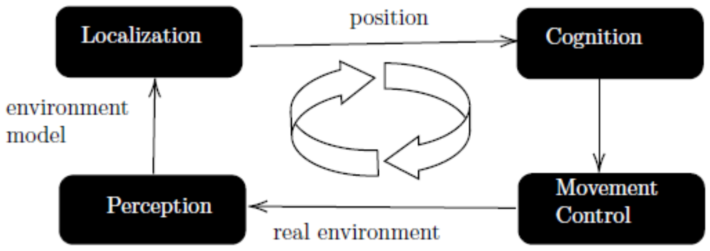

Figure 1: Diagram of component modules of an autonomous mobile robot.

implies more than just knowing someone's absolute position on Earth. Considering a robot that is interacting with humans, it would need to identify its absolute position, but its relative position to humans, other robots and objects, is equally important.

2.2.1 Map-based location

A possible solution to the localization problem is the use of map-based techniques, in this approach the robot tries to locate itself using data collected from the sensor, which is used to update a belief (estimate) about its position in relation to a map. of the environment. According to (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011), the main advantages of map-based systems are the following:

- The explicit map-based concept of position makes the system's belief about position available transparent to human operators.
- The existence of the map itself represents a means of communication between humans and robot: the human can simply give the robot a new map if the robot goes to a new environment and if the robot creates a map it can also be used by humans.

The disadvantages of map-based positioning are the requirements for sensor accuracy and the need for there to be stationary, easily distinguishable features that can be used for comparison.

An example of the map-based architecture is shown in Fig. 2 it can be seen that the data obtained by the sensors are interpreted in the perception layer, and this information is used to build the map and locate the robot on this map, after these steps the robot is able to plan the route and move around the environment using its actuators.

A general localization scheme for mobile robots using maps is represented in Fig. 3, as previously stated, the localization problem consists of estimating the

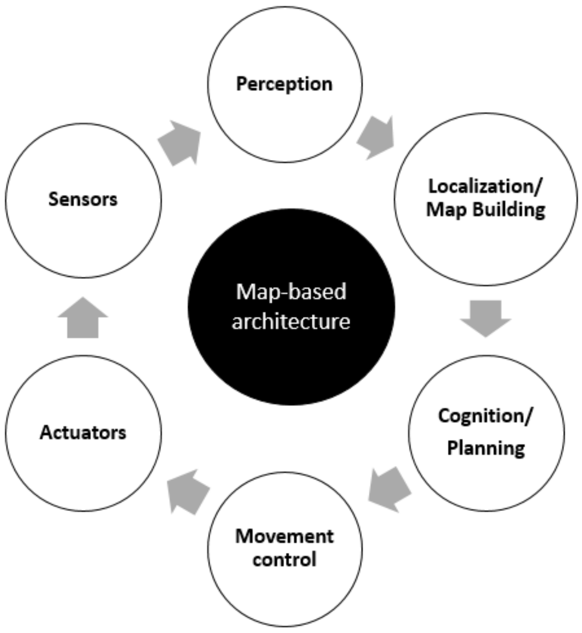

Figure 2: Map-based architecture.

position and orientation of the robot. To perform this task, the robot must have information about the map of the environment, use some technique, such as odometry, to predict the robot's position, obtain information from sensors that can assist in the process of verifying whether the estimated position matches with the characteristics observed in the environment and thus be able to update your position estimate.

You can define the position p of the robot as follows:

$$p = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}, \tag{1}$$

where x represents the horizontal axis, y represents the vertical axis and θ represents the angle.

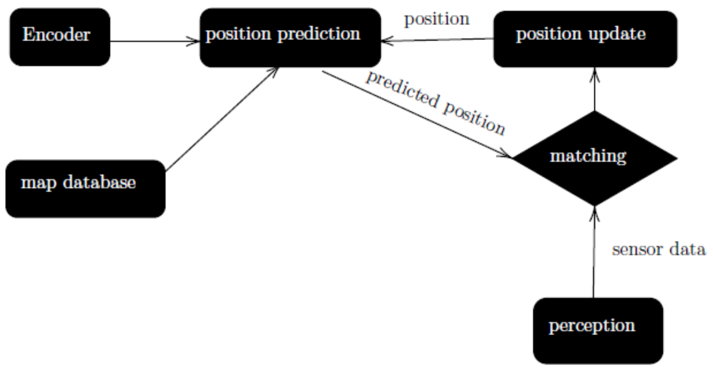

Figure 3: General localization scheme for mobile robots.

For a robot with differential wheels (such as the e-puck robot used in this work) the current position can be estimated from a known initial position, adding the incremental displacement distances:

$$\Delta x = \Delta s \cos(\theta + \frac{\Delta\theta}{2}), \tag{2}$$

$$\Delta y = \Delta s \sin(\theta + \frac{\Delta\theta}{2}), \tag{3}$$

$$\Delta\theta = \frac{(\Delta S_r - \Delta S_l)}{b}, \tag{4}$$

$$\Delta s = \frac{(\Delta S_r + \Delta S_l)}{2}, \tag{5}$$

where $(\Delta x; \Delta y; \Delta\theta)$ represent the path taken in the last sampling interval Δt, the distance traveled by the right wheel is represented by ΔS_r and the distance traveled by the left wheel is represented by ΔS_l and b is the distance between the wheels.

The updated position p' can be calculated as follows:

$$p' = \begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \Delta s \cos(\theta + \frac{\Delta\theta}{2}) \\ \Delta s \sin(\theta + \frac{\Delta\theta}{2}) \\ \Delta\theta \end{bmatrix} \tag{6}$$

Replacing $\Delta\theta$ and Δs, we have:

$$p' = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \frac{(\Delta S_r + \Delta S_l)}{2} \cos(\theta + \frac{(\Delta S_r - \Delta S_l)}{2b}) \\ \frac{(\Delta S_r + \Delta S_l)}{2} \sin(\theta + \frac{(\Delta S_r - \Delta S_l)}{2b}) \\ \frac{(\Delta S_r - \Delta S_l)}{b} \end{bmatrix} \tag{7}$$

The position error grows over time due to integration errors of the uncertainties of p and the motion errors that occurred during the motion increment $(\Delta S_r; \Delta S_l)$. (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

2.2.2 Belief representation

According to (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011), a fundamental issue that differentiates map-based localization systems is representation. There are two specific concepts that the robot must represent, the map and its belief (estimate) regarding its position on the map. To draw the map, it is necessary to question which aspects of the environment should be represented and the level of fidelity of the map in relation to it. To represent the belief in relation to its position on the map, it is necessary to decide whether the robot will identify a single position as the current one or whether it will describe its position in terms of a set of possible positions.

Probabilistic robotics represents beliefs through conditional probability distributions. A belief distribution assigns a probability (or density value) to each possible hypothesis regarding the true state. Belief distributions are posterior probabilities on state variables conditional on the available data. Let us denote belief about a variable state x_t by: $bel(x_t) = p(x_t|z_{1:t}, u_{1:t})$, where the values of z_t represent the data of measurement, obtained from camera images, laser ranging scans, etc. And the values of u_t represent control data obtained from information such as the robot's speed and odometers (THRUN; BURGARD; FOX, 2006).

According to (THRUN; BURGARD; FOX, 2006) the most general algorithm for calculating beliefs is given by the Bayes filter algorithm. This performs the calculation of a belief ($bel(x_t)$), generally made from measurements and control data. Furthermore, it is a recursive filter, which means that the belief at a given instant t is calculated based on the belief at a past instant $t-1$. As can be seen in the Algorithm 1 the Bayes filter is composed of two main steps, which are the prediction step: $\overline{bel}(x_t)$ and the correction step: $bel(x_t)$.

Algorithm 1 - Bayes filter

```
Function Filter_of_Bayes ($bel\ X_{t-1}$), $ut$, $zt$:
1: For all $x_t$ do
2: $\overline{bel(x_t)} = \int p(x_t|u_t, x_{t-1}) bel(x_{t-1}) dx$
3: $bel(x_t) = \eta p(z_t|x_t)\overline{bel}(x_t)$
4: end
return: $bel(x_t)$
```

Other examples of probabilistic localization are Markov Localization and Kalman Filter Localization, in both approaches the total probability theorem is applied in the update phase of the forecast. However, Bayes' rule is used to update perception (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

2.3 Robot Operating System

ROS (Robot Operating System) is an open source software for developing robotic applications, ROS offers a standardized software platform for developers in all industries that will take them from research and prototyping to deployment and production (MACENSKI et al., 2022).

Among the main characteristics of ROS, the fact that it has a global community of developers for more than 10 years stands out, it has reliability proven by several researches in robotics and it provides a set of tools, libraries and resources to accelerate the development of the robotic application allowing also integration with existing software.

The open source framework operates as a structured communications layer above the host operating system, making it possible for robots on a network to communicate and work together. The main goal is to provide an efficient approach to robotics research and development, as the workflow can vary greatly across different projects.

At the heart of any ROS system is the ROS graph. The ROS graph refers to the network of nodes in a ROS system and the connections through which they communicate.

2.3.1 Nodes

Each node in ROS must be a module responsible for a single purpose (e.g. a node to control wheel motors, a node to control a laser rangefinder, etc). Each node can send and receive data to other nodes through topics, services, actions or parameters. A complete robotic system is made up of several uniquely named and therefore identifiable nodes. This ensures better fault tolerance, as faults are isolated to individual nodes, which can be easily traced.

Topics are vital elements of the ROS graph that act as a bus for nodes to exchange messages. A node can publish data to any number of topics and simultaneously have subscriptions to any number of topics.

2.3.2 Services

Services are another method of communication for nodes in the ROS graph. Services are based on a request/-response model, versus the publisher-subscriber model of topics. While topics allow nodes to subscribe to data streams and obtain continuous updates, services only provide data when specifically called by a client.

2.3.3 Actions

Actions are one of the types of communication in ROS 2 and are intended for long-running tasks. They consist of three parts: a goal, feedback and a result.

Actions are built on topics and services. Their functionality is similar to services, except that actions are preemptable (you can cancel them while they are running). They also provide constant feedback, unlike services that return a single response.

Actions use the client-server model, an "action client" node sends an objective to an "action server" node that recognizes the objective and returns a stream of feedback and a result.

2.4 Localization Methods using ROS

With the aim of selecting the most used localization methods with ROS, a systematic review of scientific literature was carried out on the Web of Science, Science Direct and IEEE Xplore and Scopus platforms. The selected works were analyzed and compared following some criteria described in Table 1.

2.4.1 Selected Works

(BELKIN; ABRAMENKO; YUDIN, 2021) discuss the main features of the proposed modular approach for lidar-based localization and propose a new procedure for evaluating the robot's initial position based on the modified Monte Carlo localization method. The application of the proposed modular approach showed a

Table 1: Main Location Methods.

Author	Envir.	Method	Sensor	Platform
(CHIKURTEV et al., 2021)	Outdoor	MC Local Adap.	Lidar	Nvidia AGX Xavier
(CHIKURTEV et al., 2021)	Indoor	AMCL	Lidar and GPS	Not identified
(CHUNG; LIN, 2022)	Indoor	AMCL adapted	Lidar	K1 4Core, 2G RAM
(CHAME et al., 2018)	Outdoor	ANN SF	DGPS, SONAR	Ubuntu 16.04
(FRANCHI et al., 2021)	Outdoor	MAP	USBL and GPS	i7-16 GB RAM
(GUAN et al., 2020)	Indoor	VLP	Camera CMOS	Raspb Pi 3, 1 GB RAM
(GUNAS et al., 2018)	Indoor	AMCL	Laser	Dell laptop
(HOUSEIN et al., 2022)	Indoor	SF EKF	Odometry, LIDAR	Not identified
(JIANG et al., 2018)	Indoor	EKF adapted	Kinect and Laser	Intel 847 Dual Core
(JONASSON et al., 2021)	Indoor	ALOAM	LIDAR, RADAR	Nvidia Jetson Nano
(KAYHANI et al., 2022)	Indoor	SF EKF	Camera and IMU	Not identified
(KING et al., 2021)	Outdoor	Sound local	Microfone, LIDAR	Not identified
(LI et al., 2021)	Indoor	SLAM	Camera	3.3 GHz 16 GB RAM
(OğUZ-EKIM, 2020)	Indoor	SRD-LS	Lidar	Xeon 4-Core 8 GB RAM
(OISHI et al., 2019)	Outdoor	SeqSLAM++	3D LIDAR, Camera	Not identified
(PEEL et al., 2018)	Outdoor	AMCL, SLAM	Lidar	Raspberry Pi
(RIZZO et al., 2021)	Outdoor	RF	RF transmitter and receiver	Not identified
(SAEEDVAND et al., 2018)	Outdoor	LHOL	IMU	Intel Core i5, 8 GB RAM
(SULLIVAN et al., 2018)	Indoor	Localization EKF	3D Lidar	Not identified
(XIAO et al., 2019)	In/Outdoor	Dynamic-SLAM	Camera	Intel Core i5, 8 GB RAM
(YILMAZ et al., 2019)	Indoor	SA-MCL	Lidar	Not identified

position error of 10 cm and an orientation error of 0.7 degrees.

(CHIKURTEV et al., 2021) They present a method that uses data from Lidar and GPS sensors to perform localization with the AMCL and achieved position accuracy of +/- 0.93 m and orientation of 2.53 degrees.

(CHUNG; LIN, 2022) propose an improvement in AMCL and uses 2D laser information and rangefinder to perform the localization task. From the experimental results, the authors claim that the improved AMCL algorithm can improve the positioning accuracy of the robot compared to the original AMCL.

(CHAME; SANTOS; BOTELHO, 2018) propose a neural network model called B-PR-F that heuristically performs adaptive information fusion, based on the principle of contextual anticipation of the location signal with ordered neighborhood processing. A simulator experiment shows that the model outperforms the Kalman Filter and Augmented Monte Carlo Localization algorithms in the localization task.

(FRANCHI et al., 2021) present a MAP estimator (*Maximum A Posteriori*) adapted for localizing AUVs (Autonomous Underwater Vehicles) in the presence of USBL measurements (*Ultra-Short BaseLine*), the solution presented showed an error maximum horizontal of 2.5 m.

(GUAN et al., 2020) present for the first time a visible light positioning (VLP) localization system. A prototype system was implemented on a Turtlebot3. Experimental results show that the proposed system can provide position accuracy in the range of 1 cm and an average computational time of only 0.08 s.

(HOUSEIN et al., 2022) use the Extended Kalman Filter (EKF) to perform LIDAR sensor fusion and Odometry. Experimental results on the “turtlebot” robot using ROS show a significant improvement in the robot pose compared to Odometry.

(JIANG et al., 2018) developed a cooperative human localization system in indoor environments using a mobile robot and *smartphones*. An EKF-based dynamic localization algorithm was developed to fuse distance measurements from the Kinect sensor and smartphone-based acoustic range so that target positions can be estimated iteratively. Experiments showed that the positioning algorithm was able to locate and track moving human targets in different indoor environments. The accuracy of the median estimate ranges from 0.43 m to 1.12 m.

(KAYHANI et al., 2022) propose a *tags*-based visual inertial localization method for common UAVs with only one camera and one inertial measurement unit (IMU). the proposed method estimates the global pose of the UAV by fusing inertial data and tag measurements using EKF. The root mean square error (RMSE) ranges from 2 to 5 cm.

(KING et al., 2021) examine the feasibility of an audiovisual methodology for locating hidden sound sources. A four-channel microphone array is used in conjunction with LiDAR and 2D/3D mapping to merge estimated angles of arrival with room parameters for sound source localization. Tests were carried out in controlled and uncontrolled environments and the method was able to find the hidden source with an accuracy of 0.5 m.

(LI et al., 2021) propose a multi-robot visual SLAM partial computing strategy, causing part of the processing to be carried out in the cloud instead of locally. The best discharge point is given to reduce the energy consumption and time cost of the entire visual SLAM system.

(OğUZ-EKIM, 2020) propose a new algorithm for robot localization based on the least squares method. The approach describes the robot's location in polar coordinates, leading to a non-convex algorithm whose solution can be found efficiently. The root mean square error (RMSE) ranges from 2 cm to 2.2 m.

(OISHI et al., 2019) present a new approach to vision-based localization and navigation for outdoor environments. The new method called SeqSLAM++, is an extension of the conventional SeqSLAM so that in addition to robustly estimating the position of the robot by comparing sequences of images, it can also deal with changes in the direction and speed of a robot, as well as visualize changes using angular and a Markov localization scheme. Reliability has been demonstrated in an outdoor environment and achieved a 95.8% localization success rate.

(PEEL et al., 2018) use Adaptive Monte Carlo Localization on a known map and this method provided results comparable to Hector-SLAM, with error rates lower than a defined threshold of 10 cm.

(RIZZO et al., 2021) estimate the location of a robot along a pipe with an alternative Radio Frequency (RF) approach. The method is capable of locating a robot along a pipe by generating and detecting a periodic signal fading pattern, without the need for a previously known map of the scene or any substantial modification of existing infrastructure.

(SAEEDVAND; AGHDASI; BALTES, 2018) present a robust learning method to localize a humanoid robot called *Lightweight Humanoid robot Odometric Learning method* (LHOL). The basic learning of the method is based on the artificial neural network (ANN) that uses kinematic calculations, IMU (rotation and tilt data) and the load data present internally from the robot actuators as input data. The average error rate presented was less than 9 cm.

(XIAO et al., 2019) present a *framework* SLAM called DynamicSLAM, which is a monocular visual semantic simultaneous localization and mapping system using deep learning to improve performance in dynamic environment. The localization accuracy of DynamicSLAM is higher 7.48% 62.33% than the state-of-the-art ORBSLAM2 system and the operation performance is improved by about 10%.

(YILMAZ; TEMELTAS, 2019) propose a new ellipse-based energy model for the SA-MCL method (*Self Adaptive Monte Carlo Localization*) to eliminate the uniform sensor positioning constraint of the standard SA-MCL and make the method suitable for AGVs (*Autonomous Guided Vehicles*) with 2D or 3D LIDARs. It is observed that the algorithm estimated the position of the AGV with error rates of 6.69 cm and 4.74 cm RMSE in the x and y directions, respectively.

As can be seen in Table 1, the Monte Carlo and sensor fusion methods with EKF are the most used for localizing mobile robots with ROS and for this reason these methods are addressed in more depth in this work and are implemented in the Webots simulator using ROS 2.

3 Materials and Methods

This work seeks to study and implement mobile robot localization techniques in a simulated environment and compare them. For this purpose, the Webots simulation platform was used. This simulator allows controllers to be programmed in C/C++ (LANCHEROS; SANABRIA; CASTILLO, 2016), Python (FISHER, 2022), Java (FARLEY; WANG; MARSHALL, 2022), Matlab (MA; LIANG; TIAN, 2020) or ROS (DOBROKVASHINA et al., 2022). In this work we use the API (*Applications Protocol Interface*) for ROS (Robot Operating System), in relation to the ROS version, the most recent version 2 LTS (Long-Term Support) was used, codenamed (Foxy Fitzroy), in in conjunction with Python version 3.8 to implement localization methods in the simulator. Regarding the test environment, all techniques were evaluated in the same scenarios, the environment was prepared in such a way that it did not interfere negatively with any technique and the efficiency of each of these localization techniques could be analyzed and the results obtained with in relation to lines of code, processing time, CPU consumption, memory consumption and implementation difficulty.

The entire process was carried out on a notebook with a 2.4 GHz Intel Core i3 processor, 4 GB of DDR3 RAM, 240 GB of SSD and the Ubuntu Linux — Focal Fossa (20.04) 64-bit operating system.

3.1 Mobile Robot Used

The robot used in the experiments was the E-puck Mobile Robot (Fig. 4). The E-puck is a miniature mobile robot originally developed at EPFL (Swiss Federal Institute of Technology in Lausanne) for teaching purposes by the designers of the successful Khepera robot. The e-puck hardware and software is completely open source, providing low-level access to all electronic devices and offering extensive extension possibilities. The e-puck robot is 7.4 cm in diameter, 4.5 cm high and weighs 150 g.

This section presents the navigation methods used in this work. It should be noted that all methods were used with noise, simulating industrial environments.

The webots_ros2 package provides the standard URDF (Unified Robot Description Format) modeling of the E-puck robot, requiring changes only to include the new sensors that were added in the extension, such as GPS and IMU.

The webots_ros2_driver package automatically creates a ROS 2 interface for almost all Webots devices (except the IMU, as it is a combination of several devices).

The IMU is made up of three devices in Webots:

- An inertial unit device (In the simulator the inertial unit returns only one orientation).
- A gyroscope device for measuring angular velocities.
- An accelerometer device for measuring accelerations.

As the interface for the IMU device is not created automatically, for ROS to be able to recognize the device correctly, it was necessary to add the following plugin to the URDF file:

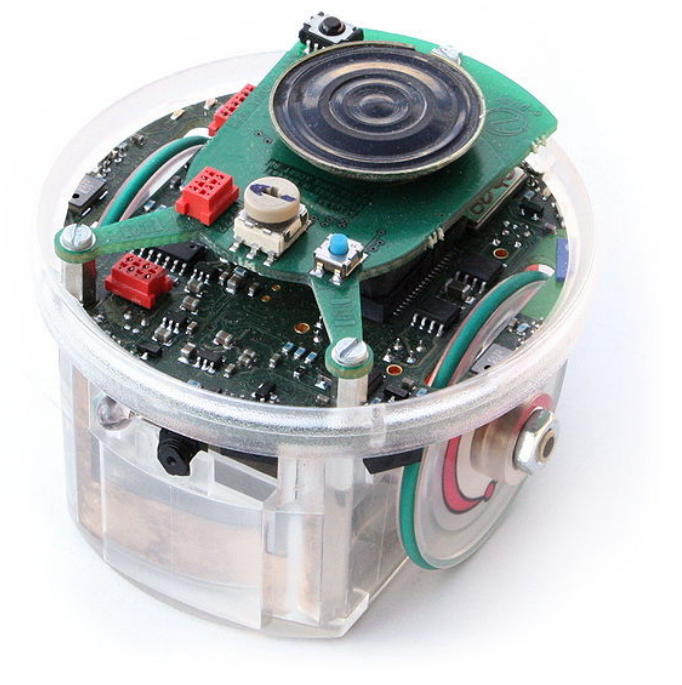

Figure 4: Epuck Mobile Robot.

```
<robot name="Epuck Webots">
<webots>
<plugin type="webots_ros2_driver::Ros2IMU">
    <enabled>true</enabled>
    <updateRate>20</updateRate>
    <topicName>/e_puck/imu</topicName>
    <alwaysOn>false</alwaysOn>
    <frameName>imu_link</frameName>
    <inertialUnitName>inertial_unit</inertialUnit
    <gyroName>gyro</gyroName>
     <accelerometerName>accelerometer</accelerome
</plugin>
</webots>
</robot>
```

3.2 Test Environment

To carry out the tests, scenario 01 was created in the Webots Simulator, as can be seen in Fig. 5, which is an environment 1.5 meters long and 1.5 meters wide, has four walls and a set of obstacles (wooden boxes). From the created environment, a map was generated to serve as a reference for localization techniques. It can be seen that the map shown in Fig. 6 has fewer details of the environment compared to the scenario shown in Fig. 5, this happens because the map should only have the most relevant information for the location, for example the location of walls and obstacles, other information such as the texture or color of the floor and walls are not present on the map because are not relevant to the location of the mobile robot.

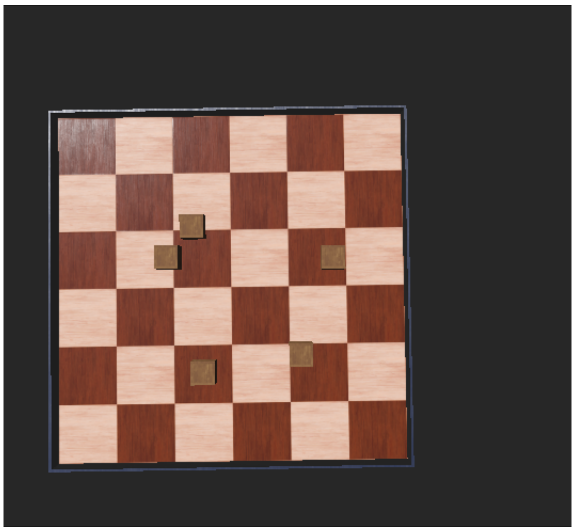

Figure 5: Scenario 01.

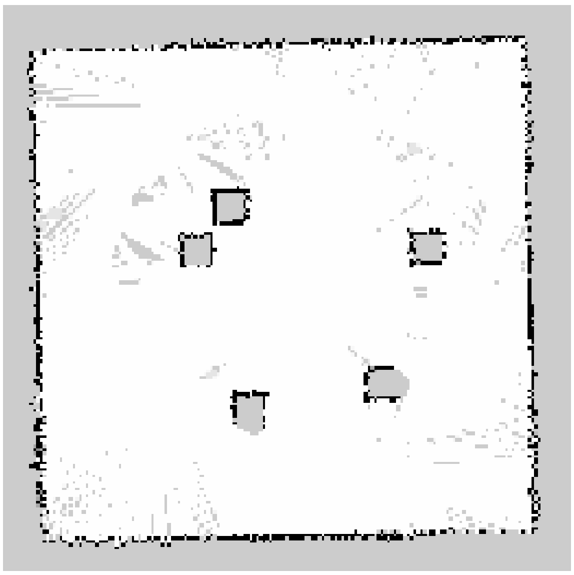

Figure 6: Map relating to scenario 01.

For purposes of comparison with scenario 01, scenario 02 was created, which is an environment similar to a house plan, with walls that separate each room, as can be seen in Fig. 7. Scenario 02 is 0.8 meters long and 0.8 meters wide, so it is considerably smaller than Scenario 01. The map generated for this Scenario is shown in Fig. 8, note that in the scenario 02, the main obstacles are the walls and in scenario 01, the main obstacles are the boxes, although there are also walls.

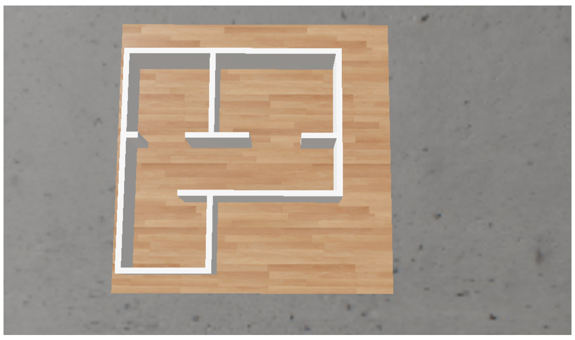

Figure 7: Scenario 02.

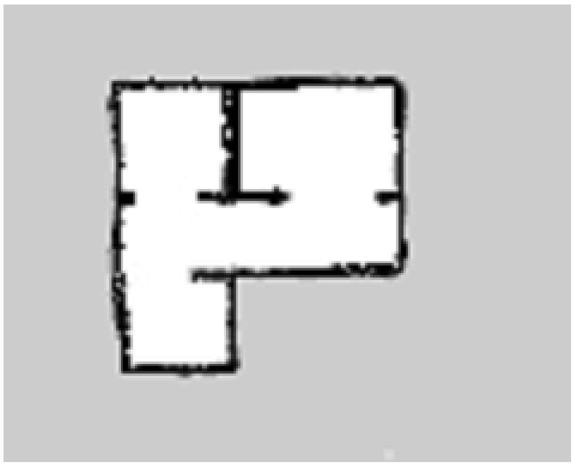

Figure 8: Map relating to scenario 02.

3.3 Location of Monte Carlo

Monte Carlo localization or also called particle filter is derived from the Bayes filter. The key idea is to use random samples drawn from the posterior probability $bel(xt)$. To implement the location of the particle filter, you need the following components:

- Movement model for forecast update;
- Measurement model for measurement update;
- Particle filter for all recursive updating.

3.3.1 Motion Model

Most robots are equipped with a wheel encoder to determine how far the wheels have turned. By using this information the robot can predict where it is positioned after moving for a period of time.

The developed odometry model uses relative motion information, which is the delta between the previous and current poses, and is calculated as follows

$$\delta_{rot1} = atan2(\overline{y'} - \overline{y}, \overline{x'} - \overline{x}) - \overline{\theta}, \tag{8}$$

$$\delta_{trans} = \sqrt{(\overline{x} - \overline{x'})^2 + (\overline{y} - \overline{y'})^2}, \tag{9}$$

$$\delta_{rot2} = \overline{\theta'} - \overline{\theta} - \delta_{rot1}, \tag{10}$$

where δ_{rot1} represents the variation in rotation 1, δ_{trans} is the variation in translation, δ_{rot2} is the variation in rotation 2, x represents the horizontal axis and y represents the vertical axis.

One can predict the current pose, adding some uncertainties to the process to model odometry errors. We model our noise as zero-mean Gaussian as (THRUN; BURGARD; FOX, 2006)

$$N(\mu = 0, \sigma^2), \tag{11}$$

where σ^2 is the variance and μ is the distribution mean. The noise of the movement model is calculated by the following functions, which have error parameters α_1 to α_4 (robot-specific errors and determine the error accumulated with the movement)

$$\delta_{rot1} = \delta_{rot1} - noise(\alpha_1 \delta_{rot1}^2 + \alpha_2 \delta_{trans}^2), \tag{12}$$

$$\begin{aligned}\delta_{trans} = \delta_{trans} - noise(\alpha_3 \delta_{trans}^2 + \\ + \alpha_4 \delta_{rot1}^2 + \alpha_4 \delta_{rot2}^2),\end{aligned} \tag{13}$$

$$\delta_{rot2} = \delta_{rot2} - noise(\alpha_1 \delta_{rot2}^2 + \alpha_2 \delta_{trans}^2), \tag{14}$$

where $\alpha_1, \alpha_2, \alpha_3$ and α_4 have the following values respectively for this specific model (0.001; 0.001; 0.01; 0.001).

The predicted state is calculated as follows

$$x' = x + \hat{\delta}_{trans} \cos(\theta + \hat{\delta}_{rot1}), \tag{15}$$

$$y' = y + \hat{\delta}_{trans} \sin(\theta + \hat{\delta}_{rot1}), \tag{16}$$

$$\theta' = \theta + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}. \tag{17}$$

3.3.2 Measurement Model

Measurement models describe the formative process by which sensor measurements are generated in the physical world. Formally, it is defined as a conditional probability distribution: $p(z_t|x_t, m)$, where x_t is the position of the robot, z_t is the measurement at time t, and m is the map of the environment.

When you know where the endpoints are on the map, you can calculate the probability of them being true. When there are no obstacles, information has no meaning and is therefore discarded.

To calculate the end points on the map, the following equation was used

$$\begin{aligned}\begin{pmatrix} x_{z_t^k} \\ y_{z_t^k} \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \\ + \begin{pmatrix} \cos\theta - \sin\theta \\ \sin\theta \cos\theta \end{pmatrix} + \begin{pmatrix} x_k, sens \\ y_k, sens \end{pmatrix} + \\ + z_t^k \begin{pmatrix} \cos(\theta + \theta_k, sens) \\ \sin(\theta + \theta_k, sens) \end{pmatrix},\end{aligned} \tag{18}$$

where z_t^k is the sensor measurement, $(x_k, sens$ $y_k, sens)$ is the relative location of the sensor with respect to the robot and $(\theta_k, sens)$ is the orientation of the sensor in relation to the robot's direction.

3.3.3 Particle Filter

The particle filter works by estimating the true pose using a set of particles/samples. The denser the area, the more likely it is to be close to the true pose. The particle filter algorithm used is presented below in Algorithm 2, in which basically two main steps occur, predicting the movement and updating the sensor measurement.

Algorithm 2 - Particles filter

```
Function Particles_Filter ($X_{t-1}$, $ut$, $zt$, $m$):
1: $X_t = 0$
2: For $m = 1 \to M$ do
3:     $x_t^m = movement\_update(ut, x_t^m)$
4:     $w_t^m = sensor\_update(zt, x_t^m, m)$
5: end
return: $X_t$
```

3.4 Sensor Fusion

To perform sensor fusion with ROS, the robot_localization package was used. This package contains state estimation nodes such as EKF_Localization

(Extended Kalman filter) and UKF_Localization (Unscented Kalman Filter). Furthermore, it allows the fusion of an arbitrary number of sensors and it is possible to configure the customization of input data per sensor, that is, if a given sensor message contains data that you do not want to include in your state estimation, the state estimation in robot_localization allow you to exclude this data per sensor. To use this package and perform sensor fusion, the first step is to check whether the sensor data complies with ROS standards. The two most important REPs (ROS Enhancement Proposals) to be considered are

- REP-103 (Standard Units of Measurement and Coordinate Conventions);
- REP-105(Coordinate frame conventions).

The sensors are configured in the ekf.yaml file, where you need to define whether you are working in 2D or 3D. The output frequency after the fusion, among other configurations that can highlight the configuration of the coordinate frames: map_frame, odom_frame, base_link_frame and world_frame; The association of sensors to the respective ROS topics and the individual configuration of each sensor according to

```
odom0_config:   [true, true, true,
                 false, false, false,
                 true,  true,  true,
                 false, false, true,
                 false, false, false]
```

The order of Boolean values is [x, y, z, roll, pitch, yaw, vx, vy, vz, vroll, vpitch, vyaw, ax, ay, az], where x, y and z represent the position of the robot, roll, pitch and yaw represent orientation, vx, vy and vz represent linear velocity, vroll, vpitch and vyaw represent angular velocity and ax, ay and az represent linear acceleration. The value obtained from each sensor will only be considered in the fusion if it is set to true, this way it is possible to customize the configuration for each sensor and obtain a better position estimate because when we obtain two or more sources of orientation data it is possible to merge only the sensor data more accurate.

3.5 Nav2 AMCL

The ROS 2 Navigation Stack (Also known as Nav2) is a free and open source navigation framework that aids in the quest to find a safe way to get a mobile robot to move from point A to point B. In other words , using Nav2 it is possible to move the robot without using the teleop_twist_keyboard package (package included with ROS that moves the robot based on the key pressed) the advantage of Nav2 is that the user informs the destination point and the robot can Draw the route and move towards it.

AMCL (Adaptive Monte Carlo Localization) is provided together with Nav2, its operation is very similar to the Monte Carlo implementation, but the number of particles is variable.

The AMCL receives a map, laser scans, and transformation messages, and its output represents the position estimate. At startup, your particle filter starts according to the parameters provided. Note that due to the defaults, if no parameters are set, the initial state of the filter is a cloud of moderately sized particles centered at (0,0,0).

To use AMCL it is necessary to install Nav2 and configure an initializer providing information about the map, configuring the robot's initial position and the parameters used.

3.6 Combined Method of AMCL with EKF Sensor Fusion

The implementation of the combined method of AMCL with sensor fusion was carried out using the position estimation output by AMCL in the topic /amcl_pose, as input for the sensor fusion method. AMCL position estimation and GPS and IMU sensors were combined with the aim of improving accuracy and decreasing the rate of position errors compared to isolated methods.

3.7 Performance Analysis

The techniques chosen for analysis are part of a subset of those found during the literature review, aiming to deepen the study of these techniques and take advantage of the time available, only some methods will be implemented and analyzed during this research. The technical analysis points will be listed below

- Quantitative Analysis:
  - Efficiency
  - CPU consumption
  - RAM memory consumption
  - Average response time
- Qualitative Analysis:
  - Ease or difficulty of implementation

– Main problems encountered

The visualization of the robot while moving around the map was carried out with RViz (ROS 3D Robot Visualizer) which is already included in the standard ROS installation.

To analyze the RAM and CPU consumption of each experiment, the following script was used, which uses the Linux top command to capture information, after which it converts to CSV format, which is compatible with Excel, and finally the graphs were generated in Excel. from the data obtained (see Algorithm 3).

Algorithm 3 - Obtaining performance data

```
while $Psleep$ 0.1;
1:  do top -b -n 3 | ...
2:  sed $-n'8, 12s/^*//; s/ * //; s/ * /; /gp;; 12q'$
3:  » output.csv;
done
```

To check position errors in relation to the real position of the robot, RMSE(oot mean squared error) was used, calculated as follows

$$RMSE = \sqrt{1/n * \sum_{i=1}^{n} (real_i - predicted_i)^2} \quad (19)$$

The performance of the methods was also evaluated using R2 Square (Coefficient of determination)

$$R2 = 1 - \frac{(\sum_{i=1}^{n} (Yi - \widehat{Yi})^2)}{\sum_{i=1}^{n} (Yi - \overline{Y})}, \overline{Y} = \frac{1}{N} \sum_{i=1}^{n} Yi, \quad (20)$$

where Yi represents the actual position, $\widehat{Yi}$ represents the predicted position, and $\overline{Y}$ represents the average of the actual position values.

To visualize the error graphs, PlotJuggler was used, which is one of the best tools for visualizing time series with ROS.

4 Results

In this section, the results obtained with the implementation of localization methods for mobile robots using ROS 2 in the Webots simulator are presented and discussed. The visualization of the position belief in relation to the map was extracted from RViz, all localization methods implemented are compared with each other.

After installing Webots and ROS2, the webots_ros2 package was used. This package provides the necessary interfaces to simulate a robot in the Webots simulator and integrate with ROS2 using messages, services and actions.

The ROS Graph with the application architecture and the relationship of ROS nodes is representeded in Fig. 9. The e_puck node is responsible for accessing the robot's hardware and all sensors, the monte_carlo_localizer node receives data from the odometry and laser scan sensors and based on this information, the robot's position is calculated at the node particle_cloud, which was developed according to the algorithm shown in the methodology and updates its position reference in relation to the map. The odom_path and mcl_path nodes are responsible for showing the path taken by the robot according to its odometry location history and the Monte Carlo algorithm respectively.

4.1 Location of Monte Carlo

MCL works by estimating the true pose using a set of particles or samples. The denser the area, the more likely it is to be close to the true pose. See the example in Fig. 10, the small red arrows represent the particles and the large red arrow is the odometry pose. Note that because MCL is a probabilistic algorithm there is no single position belief, but a set of possible positions and the best position probability is based on the number of particles with similar value.

To perform the location estimation through MCL, the initial pose of the robot is known, but not precise, in addition, noise was added to the odometry to make it more similar to what is found in development using real robots. To initialize the particles, the Gaussian distribution was used, defining the initial number of 1000 particles and as we used a non-adaptive particle filter, the number of particles remains the same throughout the execution.

In Fig. 11 the odometry and MCL location data are compared. The blue line is the Particle Filter path (based on the best particle information) and the red line is the Odometry path. Note that at some points the Particle Filter adjusts the robot's pose, most of the time in order to improve information about the location, however, there were situations in which the best particle was not in accordance with the robot's position, as can - observe the blue line crossing an obstacle, which in this case was impossible for the robot to do, that is, we have

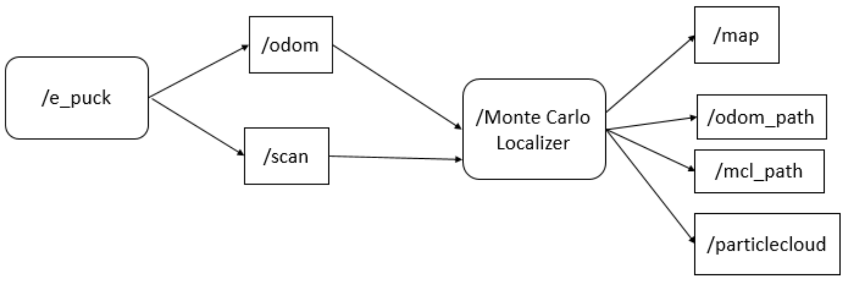

Figure 9: Application ROS graph.

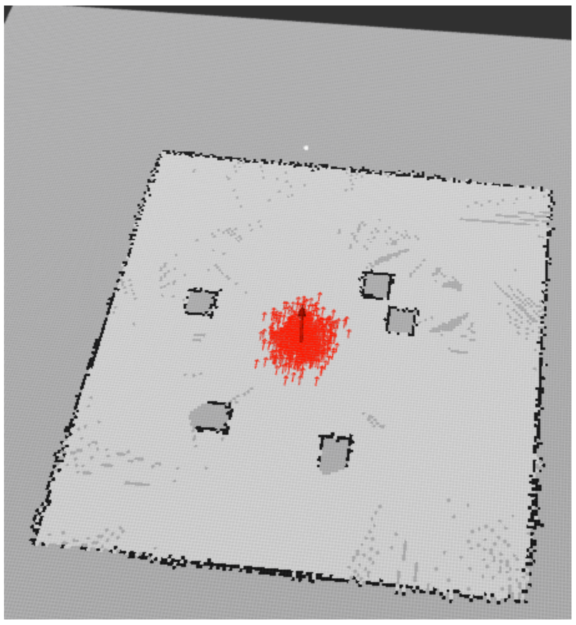

Figure 10: Monte Carlo Scenario 01.

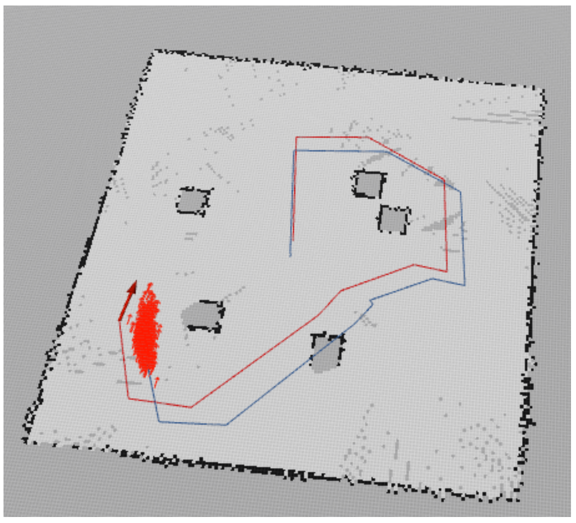

Figure 11: Comparison between Monte Carlo Location and Odometry.

a position error in that location.

To move the robot around the map and observe the behavior of the localization algorithm, the ROS 2 teleop_twist_keyboard package was used, which allows controlling the robot remotely using commands sent via the keyboard. When changing the robot's position, the position of the particles on the map was changed according to the new position, which indicates that the algorithm can update the probability of the robot's position according to the information from the sensors.

In Scenario 02, shown in Fig. 12, due to it being a more closed and smaller space, a greater uncertainty in the position of the robot was observed in relation to scenario 01, which is demonstrated by the number of particles isolated and a higher number in relation to the maximum error in Table 2. Despite the greater degree of uncertainty, the algorithm managed to perform the localization adequately and the average error was in the range of 14 cm to 16 cm.

The position errors in relation to the real position of the robot were calculated using RMSE (Root Mean Square Error) and are shown in Figures 13 and 14, which respectively represent the position errors in scenarios 01 and 02. Furthermore, the position errors are shown in a more summarized form in Table 2. It can be seen that the accuracy of the algorithm is very high since even in the worst case, the average position errors only reached 21 cm, which demonstrates its efficiency.

In order to verify the influence of the number of par-

Table 2: RMSE Position Errors - MCL Method.

Scenario	Axis	Minimum Error (m)	Maximum Error (m)	Mean Error (m)
01	x	0.000687	0.273623	0.163671
01	y	0.000824	0.377224	0.211272
02	x	0.000228	0.362798	0.145356
02	y	0.000059	0.247096	0.160695

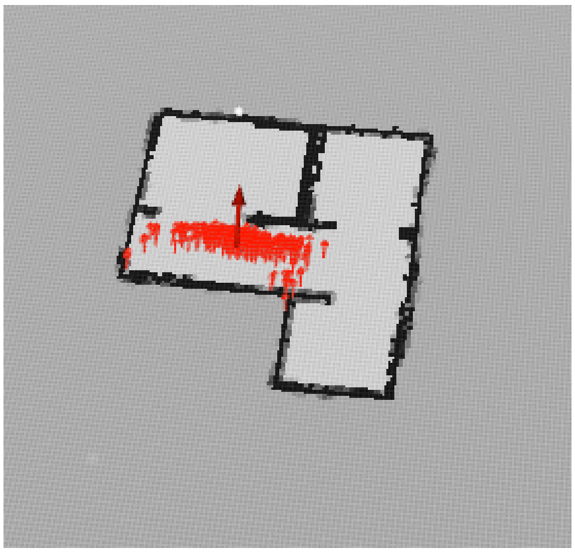

Figure 12: Monte Carlo Scenario 02.

ticles on the accuracy of the MCL location, the number of particles was set to 2000 (twice the previous experiment) and the results are shown in Table 3. It can be seen that there was no great improvement in the accuracy of the robot's localization, because despite the average error in x in scenario 02 and y in scenario 01 decreasing, the average error in x in scenario 01 and y in scenario 02 increased. Furthermore, there was an increase in computational cost and during the simulation there was greater slowness and small crashes due to this high number of particles. This slowness is due to the increase in CPU consumption, which was close to 100% utilization. So it can be seen that increasing the number of particles does not necessarily mean increasing localization accuracy.

The CPU and RAM consumption of the implementation of the Monte Carlo method (with 1000 particles) is represented in Fig. 15, which shows the percentage of use in relation to time

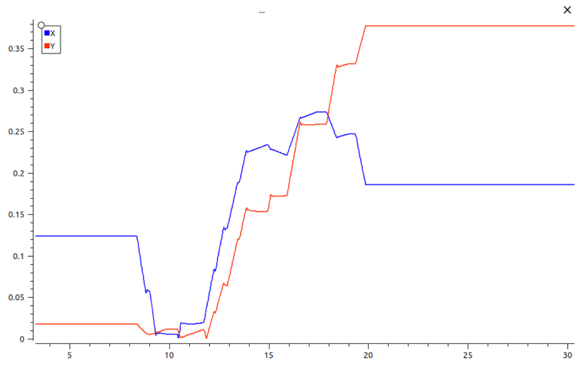

Figure 13: Monte Carlo Location Position Errors Scenario 01.

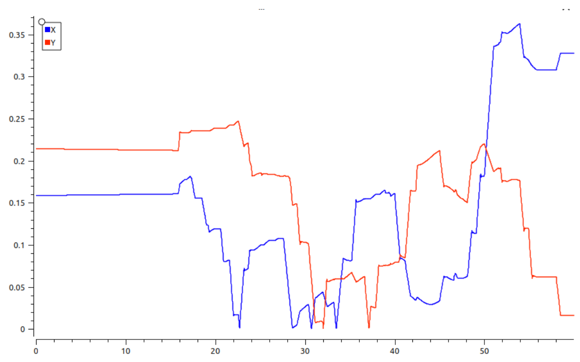

Figure 14: Monte Carlo Location Position Errors Scenario 02.

4.2 Sensor Fusion

The sensor fusion method was implemented by combining information from GPS sensors, IMU and odometry information. Position errors are shown in Figures 16 and 17 and in Table 4. It can be seen that this method also has good precision, with the average error between 1.8 cm and 6.7 cm and the maximum error of 10.9 cm.

The Memory and CPU consumption of this method is shown in Fig. 18, it can be seen that there is a lower consumption of hardware in relation to MCL. Therefore, this method can be considered a great option to be used in a low-cost autonomous mobile robot, as in addition to having good precision, it was the one with the

Table 3: RMSE position errors - MCL method with 2000 particles.

Scenario	Axis	Minimum Error (m)	Maximum Error (m)	Mean Error (m)
01	x	0.000070	0.522438	0.180753
01	y	0.000014	0.397050	0.126784
02	x	0.000063	0.421328	0.135680
02	y	0.000006	1.005860	0.484136

Table 4: RMSE position errors - Sensor Fusion Method with EKF.

Scenario	Axis	Minimum Error (m)	Maximum Error (m)	Mean Error (m)
01	x	0.014792	0.067875	0.048411
01	y	0.000001	0.029779	0.009210
02	x	0.000032	0.067991	0.018865
02	y	0.043279	0.109811	0.067829

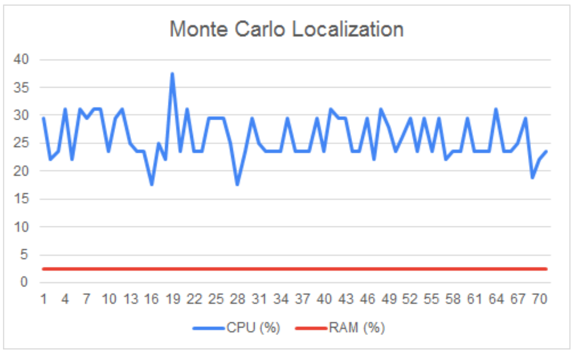

Figure 15: Memory and CPU Consumption - Monte Carlo Localization.

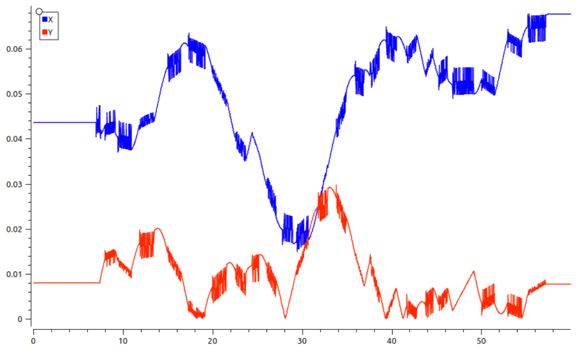

Figure 16: Sensor fusion method position errors in Scenario 01.

lowest consumption.

4.3 AMCL

To implement the AMCL, the minimum number of particles was defined as 60 and the maximum as 2000 particles. Regarding the accuracy of position estimation, this method presented results similar to the Monte Carlo localization method that was implemented following the equations, as can be seen in Figures 19 and 20 and in Table 5, the values are close to the implementation of the Monte Carlo method using 2000 particles of fixed shape. However, AMCL proved to be more efficient than normal MCL due to its variable number of particles, which directly influences performance and means it consumes less *hardware* without greatly increasing position errors.

Regarding Memory and CPU consumption (Fig. 21) although AMCL consumes less than the implemented MCL algorithm, it still consumes a little more than the sensor fusion method.

4.4 Combined Method of AMCL with Sensor Fusion

The position errors are shown in Figures 22 and 23 which represent scenarios 01 and 02 respectively. Furthermore, it is also presented in summary form in Table 6. It can be seen that the combination of these two methods was able to improve localization accuracy, as it showed much smaller errors compared to AMCL and EKF alone. Furthermore, this improvement in precision did not generate a large increase in processing, maintaining good performance.

To implement this method, the AMCL position estimate was used as input for the sensor fusion method with EKF and then this position information was combined with GPS, IMU and odometry. The *hardware* consumption of this method is shown in Fig. 24. It can

Table 5: RMSE position errors - AMCL method.

Scenario	Axis	Minimum Error (m)	Maximum Error (m)	Mean Error (m)
01	x	0.000381	0.423892	0.115521
01	y	0.000201	0.255508	0.066626
02	x	0.000000	0.163444	0.055063
02	y	0.027483	0.194471	0.081509

Table 6: RMSE position errors - Combined method of AMCL with Sensor Fusion.

Scenario	Axis	Minimum Error (m)	Maximum Error (m)	Mean Error (m)
01	x	0.011004	0.062481	0.041003
01	y	0.000021	0.023302	0.009547
02	x	0.000466	0.086590	0.031224
02	y	0.020877	0.077292	0.050768

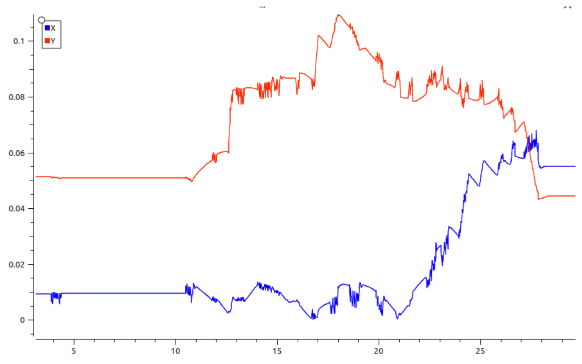

Figure 17: Sensor fusion method position errors in Scenario 02.

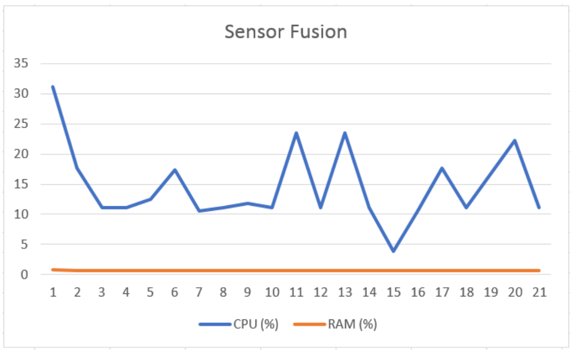

Figure 18: Memory and CPU Consumption - Sensor Fusion with EK.

be seen that it has low consumption, even though it is the result of the combination of two different methods.

5 Discussion

Table 7 shows the performance of the methods for R2 Square. It can be seen that the combined method of Sensor Fusion with AMCL presents the best performance in both scenarios (considering that for this metric the best possible score is 100%). Regarding the MCL and AMCL methods, although they have a low R2 Score value, this does not mean that they are bad localization methods, as despite the R2 Square value, both have position errors of less than 22 cm RMSE. It is understood that these low percentages only inform that the relationship between the actual value and the predicted value is not as close as the sensor fusion method or the combined AMCL fusion method.

In Table 8 we have a summary of the performance of the methods analyzed in Scenarios 01 and 02. The method that obtained the best performance in relation to position errors was the combined method of sensor fusion with AMCL and secondly the AMCL method. Regarding the consumption of *hardware* and average response time (TMR), the method that presented the best performance was sensor fusion.

Comparing the results obtained with the implementation of the methods and those found in the literature review, it can be observed that the MCL method, although it had the worst result among the methods analyzed with a maximum error of 21 cm, presented better precision than (CHIKURTEV et al., 2021) which presented an error of +/- 0.93 m and (FRANCHI et al., 2021) which presented an error of 2.5 m. The AMCL method demonstrated similar accuracy to (BELKIN; ABRAMENKO; YUDIN, 2021) which presented a position error of 10 cm and the sensor fusion method presented similar results to (KAYHANI et al., 2022) which presented errors of 2 to 5 cm and (YILMAZ; TEMEL-

Table 7: Performance of localization methods in R2 Square.

Method	Scenario	Axis	R2 Square
MCL	01	x	-11.51%
MCL	01	y	-76.07%
AMCL	01	x	-19.10%
AMCL	01	y	60.14%
Sensor fusion	01	x	94.22%
Sensor fusion	01	y	99.69%
Fusion combination with AMCL	01	x	95.10%
Fusion combination with AMCL	01	y	99.79%
MCL	02	x	-600.00%
MCL	02	y	38.14%
AMCL	02	x	95.56%
AMCL	02	y	8.89%
Sensor fusion	02	x	81.42%
Sensor fusion	02	y	52.27%
Fusion combination with AMCL	02	x	89.65%
Fusion combination with AMCL	02	y	68.20%

Table 8: Summary location methods.

Node	CPU (%)	RAM (MB)	TMR (s)	Spindle	Average Error S1 (m)	Average Error S2 (m)
Monte Carlo Location	30	102.40	0.10	x	0.163671	0.145356
				y	0.211272	0.160695
Sensor Fusion (EKF)	17	32.76	0.05	x	0.048411	0.018865
				y	0.009210	0.067829
AMCL (Nav2)	21	40.96	0.06	x	0.115521	0.055063
				y	0.066626	0.081509
AMCL/Sensor Fusion	33	73.72	0.07	x	0.041003	0.031224
				y	0.009547	0.050768

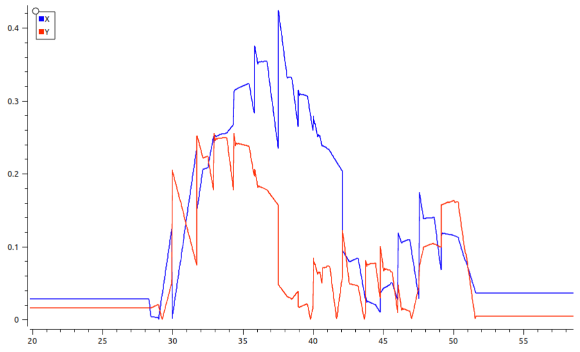

Figure 19: AMCL method position errors in Scenario 01.

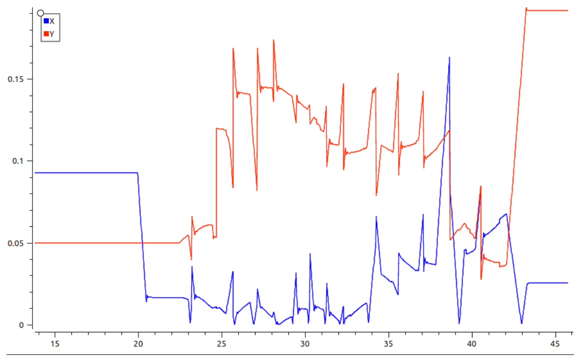

Figure 20: AMCL method position errors in Scenario 02.

TAS, 2019) which had error rates of 4.74 cm to 6.69 cm. Furthermore, sensor fusion showed a better result than (OğUZ-EKIM, 2020), where errors range from 2 cm to 2.2 m. The combined method of sensor fusion and AMCL was the one that came closest in terms of performance to (GUAN et al., 2020), which presented errors in the range of 1 cm.

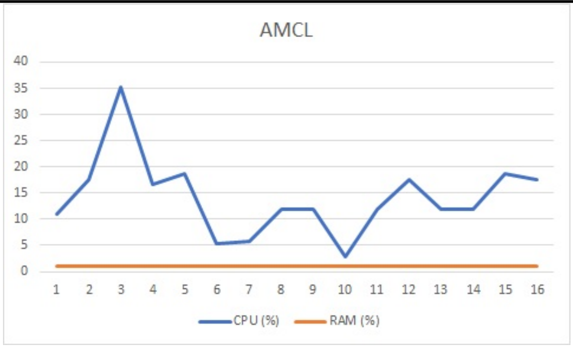

Figure 21: Memory and CPU Consumption - AMCL.

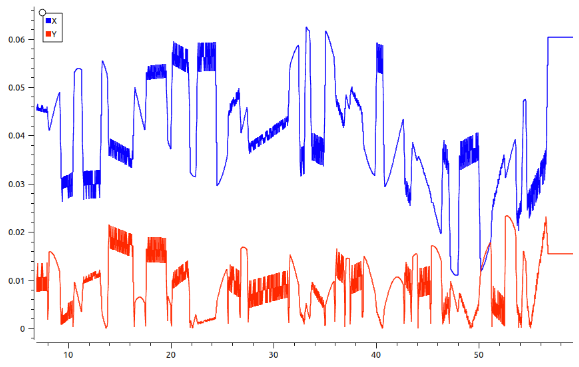

Figure 22: Position errors of the combined AMCL and sensor fusion method in Scenario 01.

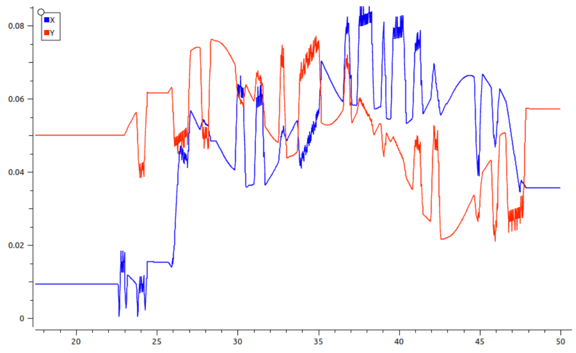

Figure 23: Position errors of the combined AMCL and sensor fusion method in Scenario 02.

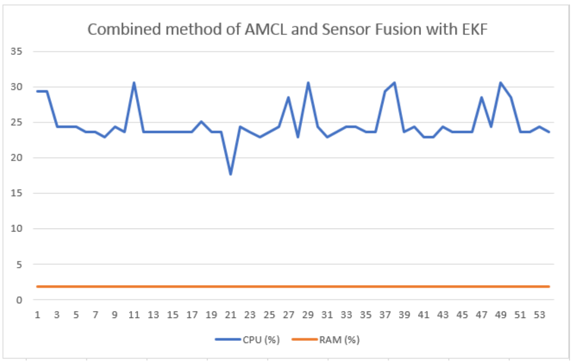

Figure 24: Memory and CPU Consumption - Combined method of AMCL and Sensor Fusion with EKF.

6 Conclusions

For an autonomous mobile robot to be able to perform its tasks properly, it is essential that this robot has the ability to locate itself in the environment. Faced with this problem, this paper addressed different localization methods applied to mobile robots, reviewing the literature on the most used methods, in addition, some were selected to be implemented and compared using the Webots simulator, the e-puck robot and ROS 2 The methods chosen were: Monte Carlo localization (MCL), Sensor Fusion (EKF), Adaptive Monte Carlo Localization (AMCL) and the combined method of AMCL with sensor fusion. The method that achieved the best performance regarding position errors was the combined sensor fusion method with AMCL with an average RMSE error of 4 to 5 cm and a score of up to 99.79% in R2 Square, and in second place the sensor fusion method with an average error of 4 to 7 cm and a score of up to 99.69% on R2 Square. Regarding hardware consumption, the best method was sensor fusion, consuming around 17% of CPU and 32 MB of RAM and in second place was AMCL with 21% of CPU and 40 MB of RAM.

The main contributions of this work were the use of different localization techniques for mobile robots, applied in 02 (two) different scenarios and in order to enable a comparative study between them. The use of ROS 2 and the availability of algorithms and results in a GitHub repository, which can be used for studies of the Webots simulator, the ROS2 and localization techniques in robotics and the literature review carried out.

Future work suggests analyzing the performance of different SLAM methods applied to ROS; analyze the implementation in the simulator in relation to robots in a real scenario; combine the techniques used with new localization methods to improve efficiency; and analyze the smallest value of particles necessary to estimate the position in each scenario.

Acknowledgements

Auzuir R. Alexandria thanks CNPq - National Council for Scientific and Technological Development (Grant #442182/2023-6).

Antônio S. S. Oliveira thanks PPGET/IFCE for funding.

References

ALATISE, M. B.; HANCKE, G. P. A review on challenges of autonomous mobile robot and sensor fusion methods. **IEEE Access**, IEEE, v. 8, p. 39830–39846, 2020.

BELKIN, I.; ABRAMENKO, A.; YUDIN, D. Real-time lidar-based localization of mobile ground robot. **Procedia Computer Science**, v. 186, p. 440–448, 2021. ISSN 1877-0509. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877050921009996>.

CAMPBELL, S.; O'MAHONY, N.; CARVALHO, A.; KRPALKOVA, L.; RIORDAN, D.; WALSH, J. Where am i? localization techniques for mobile robots a review. In: IEEE. **2020 6th International Conference on Mechatronics and Robotics Engineering (ICMRE)**. [S.l.], 2020. p. 43–47.

CHAME, H. F.; SANTOS, M. M. dos; BOTELHO, S. S. da C. Neural network for black-box fusion of underwater robot localization under unmodeled noise. **Robotics and Autonomous Systems**, v. 110, p. 57–72, 2018. ISSN 0921-8890. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0921889018302926>.

CHIKURTEV, D.; CHIVAROV, N.; CHIVAROV, S.; CHIKURTEVA, A. Mobile robot localization and navigation using lidar and indoor gps. **IFAC-PapersOnLine**, v. 54, n. 13, p. 351–356, 2021. ISSN 2405-8963. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2405896321019091>.

CHUNG, M. A.; LIN, C. W. An improved localization of mobile robotic system based on amcl algorithm. **IEEE Sensors Journal**, v. 22, n. 1, p. 900–908, 2022. ISSN 1558-1748. Disponível em: <https://ieeexplore.ieee.org/document/9606907/>.

COUSINS, S. Welcome to ros topics [ros topics]. **IEEE Robotics & Automation Magazine**, v. 17, n. 1, p. 13–14, 2010.

DOBROKVASHINA, A.; LAVRENOV, R.; BAI, Y.; SVININ, M.; MAGID, E. Sensors modelling for servosila engineer crawler robot in webots simulator. In: IEEE. **2022 Moscow Workshop on Electronic and Networking Technologies (MWENT)**. [S.l.], 2022. p. 1–5.

DOURADO JUNIOR, C. M.; SILVA, S. P. da; NOBREGA, R. V. da; BARROS, A. C.; SANGAIAH, A. K.; FILHO, P. P. R.; ALBUQUERQUE, V. H. C. de. A new approach for mobile robot localization based on an online iot system. **Future Generation Computer Systems**, Elsevier, v. 100, p. 859–881, 2019.

FARLEY, A.; WANG, J.; MARSHALL, J. A. How to pick a mobile robot simulator: A quantitative comparison of coppeliasim, gazebo, morse and webots with a focus on accuracy of motion. **Simulation Modelling Practice and Theory**, Elsevier, v. 120, p. 102629, 2022.

FERREIRA, J. A. B. Redes neurais artificiais aplicadas em aprendizagem de trajetória em robótica móvel. 2020.

FISHER, J. C. A new python api for webots robotics simulations. 2022.

FRANCHI, M.; BUCCI, A.; ZACCHINI, L.; RIDOLFI, A.; BRESCIANI, M.; PERALTA, G.; COSTANZI, R. Maximum a posteriori estimation for auv localization with usbl measurements∗∗the research leading to these results has been partially supported by the european project eumarinerobots (eumr), which received funding from the european unions horizon 2020 research and innovation program under grant agreement no 731103. **IFAC-PapersOnLine**, v. 54, n. 16, p. 307–313, 2021. ISSN 2405-8963. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2405896321015111>.

GUAN, W.; CHEN, S.; WEN, S.; TAN, Z.; SONG, H.; HOU, W. High-accuracy robot indoor localization scheme based on robot operating system using visible light positioning. **IEEE Photonics Journal**, v. 12, n. 2, p. 1–16, 2020. ISSN 1943-0655. Disponível em: <https://ieeexplore.ieee.org/ielx7/4563994/9018355/09040589.pdf?tp=&arnumber=9040589&isnumber=9018355&ref=>.

HOUSEIN, A. A.; GAO, X. Y.; LI, W. M.; HUANG, Y. Extended kalman filter sensor fusion in practice for mobile robot localization. **INTERNATIONAL**

JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS, v. 13, n. 2, p. 33–38, 2022. ISSN 2158-107X 2156-5570 J9 - INT J ADV COMPUT SC.

JIANG, C.; FAHAD, M.; GUO, Y.; CHEN, Y. Robot-assisted smartphone localization for human indoor tracking. **Robotics and Autonomous Systems**, v. 106, p. 82–94, 2018. ISSN 0921-8890. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0921889016304201>.

KAYHANI, N.; ZHAO, W.; MCCABE, B.; SCHOELLIG, A. P. Tag-based visual-inertial localization of unmanned aerial vehicles in indoor construction environments using an on-manifold extended kalman filter. **Automation in Construction**, v. 135, p. 104112, 2022. ISSN 0926-5805. Disponível em: <https://www.sciencedirect.com/science/article/pii/S092658052100563X>.

KING, E. A.; TATOGLU, A.; IGLESIAS, D.; MATRISS, A. Audio-visual based non-line-of-sight sound source localization: A feasibility study. **Applied Acoustics**, v. 171, p. 107674, 2021. ISSN 0003-682X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0003682X20307787>.

LANCHEROS, P. N.; SANABRIA, L. B.; CASTILLO, R. A. Simulation of modular robotic system mecabot in caterpillar and snake configurations using webots software. In: IEEE. **2016 IEEE Colombian Conference on Robotics and Automation (CCRA)**. [S.l.], 2016. p. 1–6.

LI, B.; MI, Z.; GUO, Y.; YANG, Y.; OBAIDAT, M. S. A high efficient multi-robot simultaneous localization and mapping system using partial computing offloading assisted cloud point registration strategy. **Journal of Parallel and Distributed Computing**, v. 149, p. 89–102, 2021. ISSN 0743-7315. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0743731520304019>.

MA, Z.; LIANG, Y.; TIAN, H. Research on gait planning algorithm of quadruped robot based on central pattern generator. In: IEEE. **2020 39th Chinese Control Conference (CCC)**. [S.l.], 2020. p. 3948–3953.

MACENSKI, S.; FOOTE, T.; GERKEY, B.; LALANCETTE, C.; WOODALL, W. Robot operating system 2: Design, architecture, and uses in the wild. **Science Robotics**, v. 7, n. 66, p. eabm6074, 2022. Disponível em: <https://www.science.org/doi/abs/10.1126/scirobotics.abm6074>.

MARCHI, J. et al. Navegação de robôs móveis autônomos: estudo implementação de abordagens. Florianópolis, SC, 2001.

NAKHAEINIA, D.; TANG, S. H.; NOOR, S. M.; MOTLAGH, O. A review of control architectures for autonomous navigation of mobile robots. **International Journal of the Physical Sciences**, v. 6, n. 2, p. 169–174, 2011.

NILOY, A.; SHAMA, A.; CHAKRABORTTY, R. K.; RYAN, M. J.; BADAL, F. R.; TASNEEM, Z.; AHAMED, M. H.; MOYEEN, S. I.; DAS, S. K.; ALI, M. F. et al. Critical design and control issues of indoor autonomous mobile robots: A review. **IEEE Access**, IEEE, 2021.

ODEBRECHT, V. A. et al. Otimização de usinagem robótica através de análises estatísticas. Florianópolis, SC., 2020.

OISHI, S.; INOUE, Y.; MIURA, J.; TANAKA, S. Seqslam++: View-based robot localization and navigation. **Robotics and Autonomous Systems**, v. 112, p. 13–21, 2019. ISSN 0921-8890. Disponível em: <https://www.sciencedirect.com/science/article/pii/S092188901730684X>.

OğUZ-EKIM, P. Tdoa based localization and its application to the initialization of lidar based autonomous robots. **Robotics and Autonomous Systems**, v. 131, p. 103590, 2020. ISSN 0921-8890. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0921889020304309>.

PANIGRAHI, P. K.; BISOY, S. K. Localization strategies for autonomous mobile robots: A review. **Journal of King Saud University-Computer and Information Sciences**, Elsevier, 2021.

PEEL, H.; LUO, S.; COHN, A. G.; FUENTES, R. Localisation of a mobile robot for bridge bearing inspection. **Automation in Construction**, v. 94, p. 244–256, 2018. ISSN 0926-5805. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0926580517303916>.

PIO, J. L. de S.; CASTRO, T. H. C. de; CASTRO JÚNIOR, A. N. de. A robótica móvel como instrumento de apoio à aprendizagem de computação. In: **Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)**. [S.l.: s.n.], 2006. v. 1, n. 1, p. 497–506.

RIZZO, C.; SECO, T.; ESPELOSíN, J.; LERA, F.; VILLARROEL, J. L. An alternative approach for robot localization inside pipes using rf spatial fadings. **Robotics and Autonomous Systems**, v. 136, p. 103702, 2021. ISSN 0921-8890. Disponível em: <https://www.sciencedirect.com/science/article/pii/S092188902030542X>.

SAEEDVAND, S.; AGHDASI, H. S.; BALTES, J. Novel lightweight odometric learning method for humanoid robot localization. **Mechatronics**, v. 55, p. 38–53, 2018. ISSN 0957-4158. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0957415818301338>.

SIEGWART, R.; NOURBAKHSH, I. R.; SCARAMUZZA, D. **Introduction to autonomous mobile robots**. [S.l.]: MIT press, 2011.

SILVA, V. D.; ROCHE, J.; KONDOZ, A. Robust fusion of lidar and wide-angle camera data for autonomous mobile robots. **Sensors**, Multidisciplinary Digital Publishing Institute, v. 18, n. 8, p. 2730, 2018.

THRUN, S.; BURGARD, W.; FOX, D. Probalistic robotics. **Kybernetes**, Emerald Group Publishing Limited, 2006.

TZAFESTAS, S. G. Mobile robot control and navigation: A global overview. **Journal of Intelligent & Robotic Systems**, Springer, v. 91, n. 1, p. 35–58, 2018.

XIAO, L.; WANG, J.; QIU, X.; RONG, Z.; ZOU, X. Dynamic-slam: Semantic monocular visual localization and mapping based on deep learning in dynamic environment. **Robotics and Autonomous Systems**, v. 117, p. 1–16, 2019. ISSN 0921-8890. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0921889018308029>.

YILMAZ, A.; TEMELTAS, H. Self-adaptive monte carlo method for indoor localization of smart agvs using lidar data. **Robotics and Autonomous Systems**, v. 122, p. 103285, 2019. ISSN 0921-8890. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0921889019302106>.