Intelligent sensorization system using ML applied to robotics

Luan Carlos Klein - 54160

Dissertation presented to the School of Technology and Management of Bragança to obtain the Master's Degree in Informatics within the scope of the double degree program with the Federal Technological University of Paraná

Work guided by:

Prof. José Lima and Prof. Felipe Nascimento Martins

Prof. André Schneider de Oliveira

Bragança

2022-2023

Intelligent sensorization system using ML applied to robotics

Luan Carlos Klein - 54160

Dissertation presented to the School of Technology and Management of Bragança to obtain the Master's Degree in Informatics within the scope of the double degree program with the Federal Technological University of Paraná

Work guided by:

Prof. José Lima and Prof. Felipe Nascimento Martins

Prof. André Schneider de Oliveira

Bragança

2022-2023

Dedication

I offer my dedication to God, whose infinite wisdom has guided my journey. Additionally, I dedicate this work to my parents, Méri and Leonor, without whose unwavering support, inspiration, and example, none of this would have been possible. Lastly, I dedicate this work to my brother, Jean, who has always been a source of inspiration for me.

Acknowledgments

Initially, I would like to express my gratitude to my family, who has been unwavering in their support throughout my academic journey, even during my time in Brazil and Portugal. Furthermore, I extend my appreciation to the professors at UTFPR-Curitiba and IPB for their valuable support and for imparting knowledge. Mainly, I am grateful to my advisors, José Lima, Felipe N. Martins, and André S. de Oliveira, for their guidance and mentorship throughout this project.

I also wish to thank João Braun and João Mendes for their assistance during the research and development phase of the project. Their guidance and willingness to share their knowledge were invaluable.

Resumo

A capacidade de se localizar com precisão é fundamental para robôs autônomos. Vários métodos voltados para a solução desse problema foram desenvolvidos ao longo do tempo, incluindo métodos clássicos, marcadores fiduciais e, mais recentemente técnicas de aprendizado de máquina (do inglês, machine learning, ML). Esse trabalho propõe diferentes técnicas de ML para abordar o problema da localização de robôs na competição RobotAtFactory 4.0. Esse estudo abrange desde testes das abordagens em sistemas embebidos, com foco na viabilidade da aplicação desses métodos, na exploração de diversos modelos e abordagens usando ML, até a aplicação de modelos treinados em simulação em ambientes reais. Os resultados experimentais mostraram que os modelos podem ser executados em sistemas embebidos, e diversas técnicas obtiveram resultados com precisão milimétrica. Além disso, a aplicação direta de modelos treinados em simulação para ambiente real se apresentou promisora. Uma das principais vantagens dos modelos de ML é o desvinculo com a dependência do conhecimento prévio da posição exata dos marcadores fiducias.

Palavras-chave: Localização de robôs; machine-learning; marcadores fiducias; Competição de robótica.

Abstract

The ability to accurately localize is a fundamental key for autonomous robots. Various methods to solve this problem have been developed, including classical methods, fiducial markers, and, more recently, machine learning (ML) techniques. This work proposes different ML techniques to address the issue of robot localization in the RobotAtFactory 4.0 competition. This study includes testing the approaches on embedded systems, focusing on the feasibility of applying these methods, exploring various ML models and approaches, and applying simulation-trained models in real environments. The experimental results demonstrated that the models could be executed on embedded systems, and several techniques achieved millimeter-level accuracy. Furthermore, the direct application of simulation-trained models to real environments showed promise. One of the main advantages of ML models is their independence from the need for prior knowledge of the exact positions of fiducial markers.

Keywords: Robot localization; machine learning; fiduciary markers; robotics competition.

Contents

List of Tables

List of Figures

Acronyms

AI Artificial Intelligence. 1, 5, 7, 15, 16

AMR Autonomous Mobile Robot. 1, 7

CNN Convolutional Neural Network. 15, 23, 24, 28, 31, 39, 41, 52, 58, 59, 66

DL Deep Learning. 1, 5

DoF Degrees of Freedom. 15, 16

EKF Extended Kalman Filter. 12

GB Gradient Boosting. 28, 39, 65

ICP Iterative Closest Point. 13

KNN K-nearest neighbors. 18

KNR K-nearest neighbors regression. 18, 28, 39, 65

LiDAR Light Detection and Ranging. xiii, 3, 15

MAE Mean Absolute Error. 31–33, 56, 57

ML Machine Learning. 1, 5, 6, 16, 17, 24, 29, 34, 35, 37, 41, 43, 49, 52–54, 56, 57, 65, 66

MLP Multilayer Perceptron. 20, 28, 37, 39, 43, 44, 46, 49–52, 59–62, 65, 66

MSE Mean Squared Error. 32

NDT Normal Distribution Transform. 14

NN Neural Network. 20, 22, 23

NRMSE Normalized Root Mean Squared Error. 31–33, 56, 58

RADAR Radio Detection and Ranging. 15

RaF RobotAtFactory 4.0. 1, 3, 5, 27, 38, 49, 56, 65, 66

RF Random Forest. 28, 37, 39, 49, 51, 52, 54, 56–58, 65, 66

RMSE Root Mean Squared Error. 31–33, 56–58

SVM Support Vector Machine. 37, 49–52

Chapter 1

Introduction

A necessary skill for agents in numerous circumstances, notably in robotics with Autonomous Mobile Robot (AMR), is the ability to localize themselves in an environment [1]. Several strategies have been developed in this area using sensor data and algorithms. To solve this issue, it has become increasingly popular to apply Artificial Intelligence (AI) techniques like Machine Learning (ML) and Deep Learning (DL) [2]. Furthermore, many robot competitions use localization as a core premise; an example is the RobotAtFactory 4.0 (RaF). In this competition, a robot has to move boxes from one place to another without external communication in the shortest possible time.

Several approaches have been developed over time, aiming to address the localization problem. One strategy using the data saved about ArUco's pose has been developed to address the localization issue at the RaF competition. Analytical geometry may be used to estimate the robot's position given the preliminary information on each ArUco's attitude and the relative pose between the robot's camera and the ArUcos [3]. This method requires prior knowledge about the precise ArUco's position, and even a tiny mistake can significantly influence the robot's pose estimation.

1.1 Context: RobotAtFactory 4.0 Competition

RobotAtFactory 4.0[1] is a robotics competition in which the robot has to move boxes through a warehouse, and the main goal is to move as many boxes as possible in the shortest possible time. There are three different levels: First, it is necessary to move the boxes from the incoming warehouse to the outcoming warehouse; Second, some boxes must go through some processing machines; and Third, some boxes must go through two processing machines. The robot can use whichever means to locate itself if it complies with the competition rules. One possibility is using the markers placed on the environment (ArUcos markers). The floor consists of a print on two A0 sheets and a flat layout. A top view of the field is shown in Figure 1.1. The origin of the reference frame is located at the center of the field, where the green arrow indicates the y axis, the red arrow indicates the x axis, and the numbers indicate the ID of each ArUco marker.

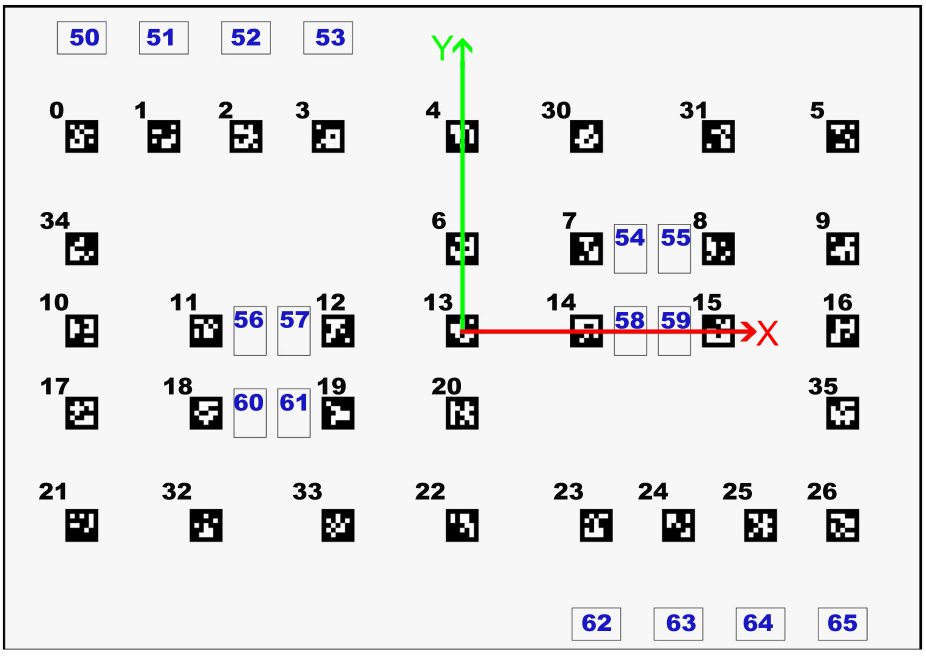

Figure 1.1: Field-top view of the RobotAtFactory 4.0 competition showing a representation of axes and ArUco's identification. Source: [4].

[1]Official page: `https://www.festivalnacionalrobotica.pt/2023/robotfactory-4-0/`.

1.1.1 Robot

The robot must comply with some restrictions to compete in the RaF competition, such as fitting within a 30 x 30 x 30 cm cube and being fully autonomous. It means that the robot cannot establish any communication with any external system that the organization does not explicitly provide.

Figure 1.2 shows the main components of one possible robot to be used in the competition: The Raspberry Pi deals with high-level control of the robot, controlling, for example, the RGB camera, the LiDAR sensor, the localization, navigation, and decision-making; The Arduino Uno manages the low-level control of the robot, such as motors, encoders, contact switch, and the electromagnet sensor. More details about the robot are available in [5] and [3].

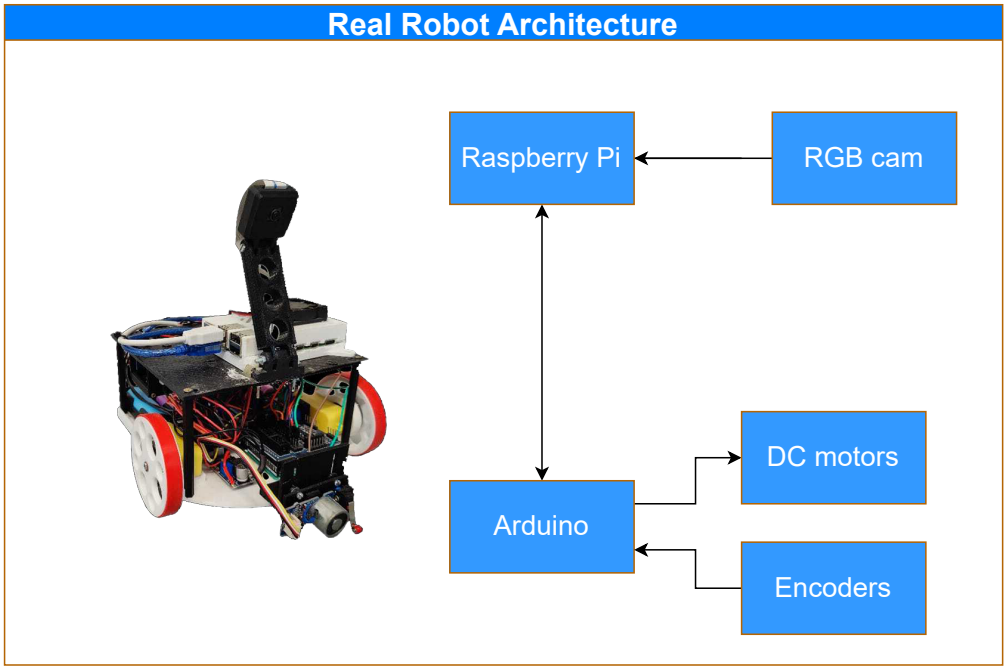

Figure 1.2: Robot architecture. The Raspberry Pi is responsible for the decision-making process and the control of the RGB camera and the LiDAR; The Arduino is responsible for the other physical parts, such as the motors and the electromagnet sensor. Source: [4].

1.1.2 Real Environment

Figure 1.3 presents a view of the real field, where it is possible to see the ArUcos markers, the boxes, and the walls of each warehouse/machine. According to the official rules, the field's dimensions are 1.7 × 1.2 m. Besides that, the competition organization provides the ID and the absolute pose (position and orientation relative to the global reference frame) for each ArUco marker.

All the ArUcos markers present in the field can be created online[2], using the settings dictionary 5x5 (50,100,250,1000) and a marker size of 60 mm. In each warehouse or machine, there will be markers with a size of 40 mm, while on the floor, the size is 50 mm.

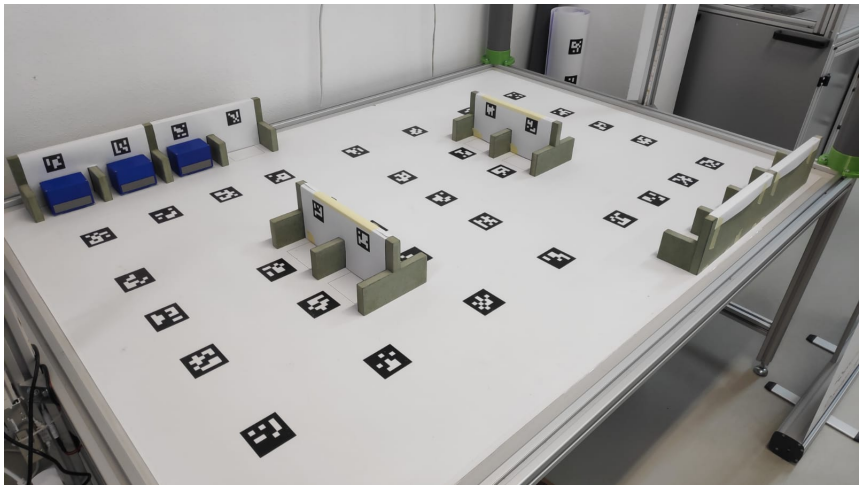

Figure 1.3: Real field competition.

1.1.3 Realistic Simulator

A simulation scene of the RobotAtFactory 4.0 competition was developed in the SimTwo simulator by the competition's organizers[3]. It works with rigid-body dynamics interactions and constraints [6]. The whole simulator, including the field and the robot, is based on the real environment. Figure 1.4 presents an image of the simulator that displays the

[2] `https://chev.me/arucogen/`.
[3] Available at `https://github.com/P33a/SimTwo`.

virtual representation of the official RaF competition field. The simulated scenario follows the specifications of the official competition rules too.

The simulator has several windows that the user can interact with. For instance, one of these windows is an XML editor, which enables the user to modify some definitions and configurations about the environment and the robot. Additionally, other features are available in the simulator, such as a code editor in Pascal language, which enables, for example, building an algorithm to define the robot's route [5]. The simulator is discussed in-depth in [6].

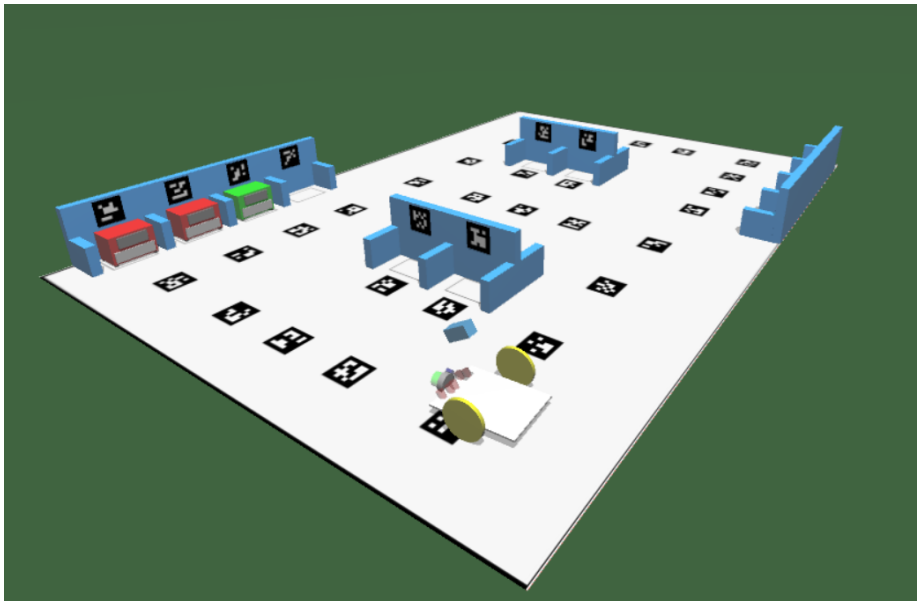

Figure 1.4: Simulator scene, which presents the robot in the competition field and the boxes.

1.2 Motivation and Objectives

This work proposes to study approaches to the localization problem, using the RaF competition as a scenario to validate and develop the approaches. The focus is to use images acquired from an onboard camera, process these images, and then use the data in AI approaches to estimate the robot's pose.

The main objective of this work is to develop algorithms of AI (including ML and DL)

focused on a robot's localization using images from one onboard camera. After building and validating the models, the objective is to do a final validation, putting the approaches in the real environment, i.e., executing the models in the physical robot. In addition, the proposed approaches can be compared with one of the current approaches used in the competition, an analytical approach proposed in [3].

The objectives follow the initial intent. The construction of a module containing the models was omitted, focusing exclusively on the validation and execution of the algorithms. However, the application of the algorithms is now specified and centered on robot localization using images. These algorithms have been expanded to consider Deep Learning techniques as well. Based on the results of this work, two papers were already published ([7] and [4]), and another one was already sent to a conference ([8]).

The methods presented in this thesis do not require explicit knowledge of ArUco's pose. The primary benefit of the strategies discussed in this study is this distinction. In circumstances where obtaining the precise pose of the markers is challenging or impossible (for instance, in hostile environments), images for training can be collected by a robot that is aware of its location using a different localization system (such as a Differential GPS-DGPS) to produce the dataset needed to train the model. Later on, while localization is underway, the DGPS is no longer required, and precise localization may be attained using only a camera [4].

The work is divided into five more chapters. Chapter 2 presents state of the art in localization and ML; Chapter 3 presents the methodology used in this work, the approaches proposed, and their implementations; Chapter 4 presents the results and discussions; Finally, Chapter 5 presents the conclusions and future works.

Chapter 2

State of the Art

This chapter presents the state of the art, divided into two parts: Section 2.1 presents the state-of-the-art in localization, and Section 2.2 presents some of the concepts and state-of-the-art in AI.

2.1 Localization

Localization in robotics is intrinsically associated with the question "Where am I?" as performed by the robot. The capacity to localize itself with certain precision is a fundamental competency that an AMR requires. Knowledge of its position and orientation is essential to making reasonable decisions about future actions [1]. Localization can be understood as estimating a mobile system's position and orientation in some reference frame.

The information about the localization depends on the dimensions of the problem [9][1]. In scenarios with 2 dimensions (2D), there are 3 degrees of freedom: $(x,\ y,\ \theta)$, where x and y are the position about some reference and θ is the orientation; in scenarios with 3 dimensions (3D) there are 6 degrees of freedom: $(x,\ y,\ z,\ \alpha_{roll},\ \alpha_{pitch},\ \alpha_{yaw})$. With this, it is possible to define an important concept called **pose**, which is the set composed of the position and orientation. For example, in the 2D case, the pose is the set $\{x,\ y,\ \theta\}$.

[1]The complete course is available at `http://www.ipb.uni-bonn.de/msr1-2021/`.

Online and offline localization is a critical related concept. With offline localization, the data can be recorded and processed after the execution (an answer during the execution is not necessary), and an example of this is using localization just to update the environment map. On the other hand, online localization is when the system needs to know its localization at the moment of execution (as in the navigation of an autonomous vehicle) [9].

When a mobile robot has onboard sensors to track its motion, like wheel encoders, it can use this data to estimate its location concerning where it started if a mathematical motion model is available. This method is called odometry, also known as dead reckoning [1]. Another relevant definition is the type of localization, which can be divided into two categories: Global Localization and Tracking Pose [9]. In Global Localization, initially, the system can be anywhere in the world, and the initial position can be unknown. On the other hand, Tracking Pose initially knows where the system is located.

Simultaneous Localization and Mapping (SLAM) is an exciting localization issue. The question addressed in this topic is about the feasibility of a mobile robot being deployed in an uncharted environment at an unknown location and incrementally building a consistent environment map while simultaneously determining its location in this map [10]. In addition, one interesting issue in robotics localization is the *kidnapped robot problem*, which consists in the situation where the autonomous robot in operation is moved to an arbitrary position [11].

Furthermore, it is essential to highlight that in addition to estimating the pose, it is necessary to know the uncertainty associated with the estimation since there can be catastrophic consequences due to decisions made based on pose estimates that are assumed to be perfect [1].

The localization also can be divided into two types according to the environments: outdoor and indoor. One of the most famous approaches to the outdoors is the GPS (Global Positioning System). However, this approach may not be available indoors due to the limitations of blocked satellite signals or attenuated by structures like walls [12]. In this way, other alternatives are necessary to solve this problem, and some of them are

discussed in the following sections.

2.1.1 Markov Localization

The Markov Localization, also known as Grid Localization, is an approach to the Global Localization problem [13]. This approach aims to discretize the space of possible robot poses and manipulate discrete probability distributions [1]. It is presumed that the environment is static to simplify the explanation, but this approach can also be used in a dynamic environment [14]. Markov Localization is a probabilistic algorithm: instead of maintaining a single hypothesis of where a robot might be in the world, it maintains a probability distribution over the space of all such hypotheses [14]. For this, the approach uses a histogram for each degree of freedom.

An example of how Markov Localization works is presented in [15], considering a one-dimensional scenario where the robot can move horizontally. Consider a situation where a robot is placed in the environment but does not know its localization. So, the Markov approach represents this uncertainty by a uniform distribution over all positions, as shown in the first part of Figure 2.1. Suppose the robot uses its sensor to determine whether it is the neighbor. The strategy alters the probability distribution by increasing the likelihood near a landmark and decreasing it elsewhere. This is illustrated in the second part of Figure 2.1. Notice that places not next to the landmarks still possess non-zero probability because sensor readings are noisy, and a single sight of a door is typically insufficient to exclude the possibility of not being next to a door.

So, considering the robot has moved forward. The Markov Localization incorporates this information by shifting the belief distribution accordingly, as presented in the third part of Figure 2.1. Finally, assuming the robot senses a second time, it finds itself next to a landmark again. This observation is now multiplied into the present belief (non-uniform), which results in the belief depicted in the fourth part of Figure 2.1. At this point, most of the probability is centered around a single position, and the robot has more certain about its localization, as presented in the last part of Figure 2.1.

The Markov Localization approach has a disadvantage due to its use of a large quantity of memory because it must maintain a histogram for each degree of freedom. So, for instance, in a 3D scenario, six histograms are necessary.

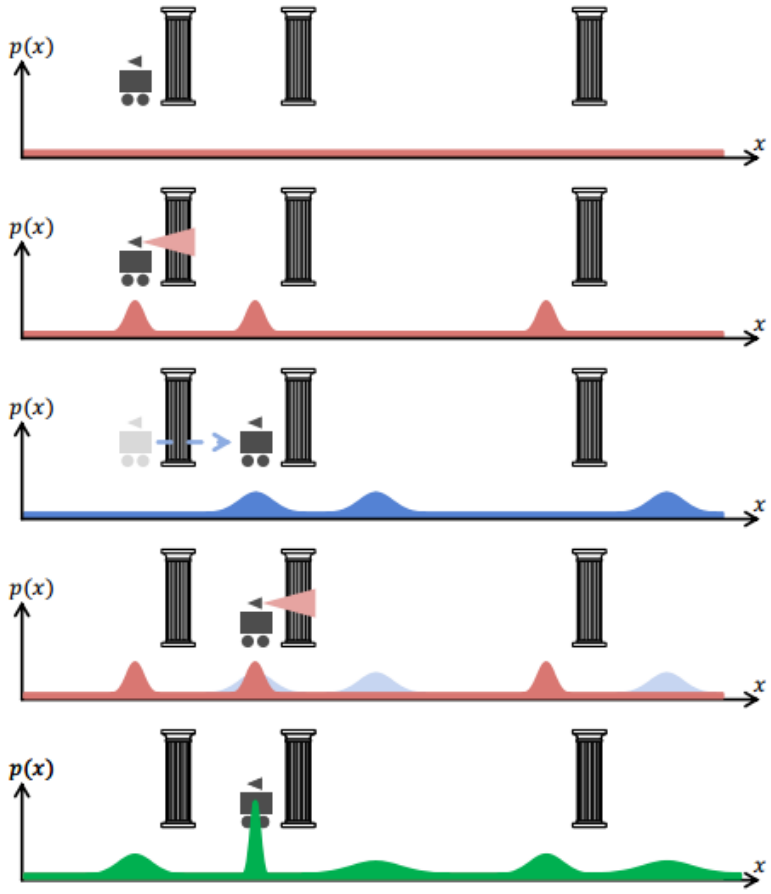

Figure 2.1: Example of Markov Localization in a 1D situation. In the first part, the robot knows nothing about its localization, and its probabilistic distribution is the same for all positions. In the second part, it identifies itself in front of a landmark and updates its probabilistic distribution. In the third part, the robot moves forward, identifies itself in front of the landmark again, and shifts its probabilistic distribution. The probabilistic distribution is multiplied in the fourth part, and the most probable localization is in a specific position, as presented in the last part. Source [15].

2.1.2 Monte Carlo Localization

The Monte Carlo estimation methods were first introduced in [16] as the bootstrap filter and further explained in [17]. This is another approach to the Global Localization problem

but can also be used for Pose Tracking. One of the main differences to Markov Localization is the amount of memory necessary. In the Monte Carlo approach, a sample rate is used instead of a histogram, and the scenario discretization is unnecessary, i.e., it does not make assumptions on linearity.

This algorithm is a type of *Particle Filter* [18], [19]. Multiple samples (particles) represent a hypothesis of the interest variable, i.e., the robot localization. Each of these hypotheses is associated with a weight representing the likelihood that the hypothetical estimate is true. A pose estimate can be obtained by the weighted sum of all samples[2]. The particle filter is recursive and operates in two stages: *predict* and *update*. The prediction happens after each action, and each particle is modified according to the existing model (*predict* stage). After that, all the weights are re-calculated based on the sensory information (*update* stage) [20].

2.1.3 Kalman Filter

Kalman Filter, first proposed in [21], is a mathematical approach that combines system measurements with predictions of how the system is expected to behave. It estimates a process using feedback control: the filter estimates the process state at some moment and then obtains feedback in the form of (noisy) measurements [22]–[24].

Figure 2.2 presents a simplified schematic of the Kalman Filter. There are two inputs: the initialization, which is performed once and inputs the initial state and the uncertainly about the initial state; The other input is the measurement performed in each cycle and inputs the measurement state and the associated uncertainly. The outputs are the predicted state and the uncertainty about that. The core of the filter is the update phase, which updates the internal parameters in each cycle and predicts the next state[3]. This cycle repeats at each time interval, which depends on each application.

The Kalman Filter is linear and is used in linear models. However, most systems are

[2]A tutorial for particle filters is available at `https://www.cim.mcgill.ca/~yiannis/particletutorial.pdf`.

[3]A tutorial with examples is available at `https://www.kalmanfilter.net/`.

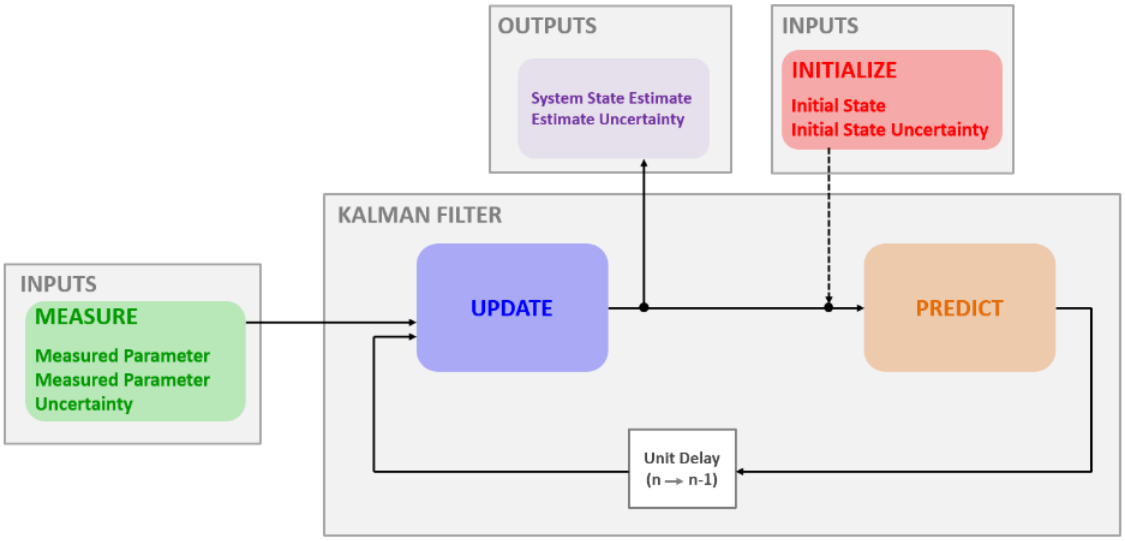

Figure 2.2: Kalman Filter example. There are two inputs: The ”initialize” step is executed once in the beginning and inputs the initial state, while the Measure input is the measurement for each system cycle. The core of the filter is composed of the ”update” and ”prediction” steps. The output is the system state estimate. Source [25].

nonlinear, and some variations of the Kalman Filter can be used in these situations, such as the Extended Kalman Filter (EKF) [22]. An important prerequisite for EKF-based localization is the ability to associate measurements obtained with specific landmarks present in the environment [1]. Data association is critical to the operation of this approach, and a catastrophic failure may result if data association decisions are incorrect.

2.1.4 Least Square

This approach is an offline approach, different from the others previously presented. That means the system needs all the data to be available beforehand. After calculating the route, it is possible to measure the difference between the calculated and the real values [9], [26]. Furthermore, this approach uses a Gaussian belief, such as the Kalman Filter. Often, this approach is used as a reference solution to other localization systems, such as the online approaches [9].

2.1.5 Slidding Window Least Squares

This approach is very similar to the Least Square. Still, instead of using all the data to calculate the localization, the sliding window uses only the most recent observations, for instance, the last 30 observations [9]. This approach aims to be better than the Kalman Filter but with a computational cost less than the Least Square [27]. This approach is currently widely used mainly for external locations, such as in autonomous cars [28].

2.1.6 Perfect Match

Perfect Match is a light algorithm, first proposed in [29]. This image-based approach was initially developed for robot localization in the Robocup Midsize league. This algorithm aims to minimize the matching error and fitting error between the data acquired and the environment map. In general, this algorithm could be divided into three main parts: (1) matching error and gradient computation; (2) optimization routine based on the Resilient Back-Propagation (RPROP); and (3) co-variance estimation using the second derivative [30].

2.1.7 Iterative Closest Point

The Iterative Closest Point (ICP) algorithm, introduced in [31], is a map-matching method. This approach minimizes the Euclidean distance between the input data and a reference model. This minimization in the localization problem corresponds to the sensor data and the map of the environment [30]. This approach is composed of two main parts: (1) The data association, that is, the matching stage, which matches each ”real” point to the closest point in the model; and (2) Transformation, which is based on the data association and tries to minimize the distance between the points. This is an iterative process, and it will converge to a result. Based on the ICP, other approaches and optimizations exist, such as the Point-to-Plane [32].

2.1.8 Normal Distribution Transform

The Normal Distribution Transform (NDT) approach, introduced in [33], is a method for 2D scan registration, and it was later extended to 3D [34], [35]. This approach is a map-matching-based algorithm, and it creates a smooth surface representation of the environment modeled by a set of local probability density functions. It uses a set of reference points grouped in fixed-sized cells forming a voxel grid to build this representation. Then, for each voxel grid cell that has at least a group of 6 points, it is computed the mean and covariance matrix [30].

2.1.9 Fiducial Markers approach

Fiducial Markers are objects used to provide a point of reference [36]. There are several types of fiducial makers, such as ARTag, AprilTag, ArUco, and STag [36]. These approaches are different from each other, such as how each of them is built. Figure 2.3 compares these four types of fiducial markers, and a deep study is present in [37]. One of the most common types is the binary square. An example of this is the ArUco Marker, which is present at OpenCV library[4] and was developed in [38].

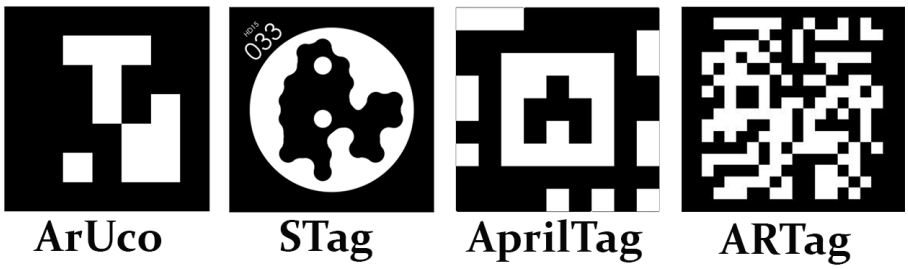

Figure 2.3: Fiducial Marker[5]comparison between ArUco, STag, ARTag. Source [4].

The RobotAtFactory 4.0 competition illustrates the use of fiducial markers for localization. One of the methods was developed in [3], where each ArUco marker identified

[4]`https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html`.

[5]Sources to generate: ArUco: `https://chev.me/arucogen/`, STag: `https://github.com/bbenligiray/stag`, AprilTag: `https://github.com/AprilRobotics/apriltag` and ARTag: `https://shawnlehner.github.io/ARMaker/`. Accessed on 12 February 2023.

in an image has its relative pose with the camera calculated. Using these values and the knowledge of ArUco's pose in a global reference frame, an estimation of the robot's pose is calculated using analytical geometry. To enhance the accuracy of the estimations, certain stochastic filters, such as the EKF and Mahalanobis Filter, are employed to aggregate all the estimations.

2.1.10 Artificial Intelligence approaches

Another interesting approach in robot localization is using AI approaches. An example of the AI application in this problem is using Deep Learning concepts, specifically the Convolutional Neural Network (CNN). This approach is discussed in the next section, with other machine learning approaches. The CNN can help locate the robot, using images from a robot camera and other sensor data [39], [40]. Interesting related work is present in [41], in which the authors used images to localization on a 6-Degrees of Freedom (DoF), using transfer learning of the GoogLeNet [42], and obtained good results, both in outdoor and indoor scenarios.

Additionally, an exciting survey on ML applications for localization was conducted in 2020 in [2]. That study investigates some methods that can be applied to feature extraction, feature selection, and regression in the context of localization. Random Forest (RF), Support Vector Machines (SVM), K-Nearest Neighbors (KNN), Artificial Neural Networks (ANN), and other techniques, including combinations of these methods, are some of the algorithms that are being presented. That study reveals that this field is developing and that further research is needed.

2.1.11 Alternatives approaches

There are a lot of approaches that were developed to solve the robot localization issue. Some of these approaches are using acoustic beacons to self-localization [43], infrared light [44], LiDAR and Radio Detection and Ranging (RADAR) with map-matching [45], image-based localization [46], map-based probabilistic visual [47], using landmarks [48]

and video tracking on 6 DoF [49].

2.2 Machine Learning

Machine learning is a sub-field inside the bigger area called AI. ML is a computer technique in which an algorithm learns from patterns in the data without explicit instructions. According to [24], three types of feedback determine the type of learning:

- **Unsupervised learning**: The algorithm identifies patterns to separate the data in groups, although no feedback is provided. An example of this is clustering.
- **Supervised learning**: From a set of input/output pairs, the algorithm tries to find a function representing the relation between them.
- **Reinforcement learning**: The algorithm learns based on a series of reinforcements (rewards and/or punishments) after taking actions in the world.

There are several techniques specific to each type of feedback. Due to the problem structure, only techniques for Supervised learning have been explored in this work. However, more details about the other two types and their appropriate techniques are available in [24], [50].

Two challenges in ML: *underfitting* and *overfitting*. Underfitting occurs when the model cannot obtain a sufficiently low error value on the training set. At the same time, overfitting occurs when the gap between the training and test errors is large, i.e., the model learns to specify the training dataset. Still, it cannot generalize to other observations [50].

Another essential concept in ML is called the *bias-variance* trade-off. The variance refers to the amount the prediction function would change if estimated using a different training dataset. On the other hand, bias refers to the error introduced by approximating a real-life problem, which may be highly complicated, by a much simpler model [51]. Ideally, the model aims to have a low bias and low variance, but, in reality, when one decreases, the other increases, and this is the trade-off between them.

2.2.1 Linear Regression

Linear regression is a statistical approach that aims to model the relationship between a scalar variable, the dependent variable or target, and one or more features, called independent variable [24]. The linear regression aims to find a mathematical relationship between these variables. An important aspect is that all features and the target needs to be scalar. Suppose one variable is not a scalar, such as a category. In that case, there are some ways to transform that into a scalar, like transforming the values into categories and assigning scalar values. An example of the linear regression function is:

$$f(x) = w_1 + w_0 x, \tag{2.1}$$

where w_1 and w_0 are the weights the algorithm calculates, x is the independent variable, and the function *f(x)* tries to represent the target value y.

A standard method to define the best function to fit the training data is to minimize the mean-square error [52]. In that, the distance estimated by the function and the real value for each point is calculated, and the difference, called residual, is calculated as the square and summed. The function that presents the least sum of the squares is the function that fits well in the data. In Figure 2.4, it is possible to see an example of linear regression with one independent variable, where the blue points are the training data and the red line represents the function found by the algorithm.

Finally, it is possible to use multiple variables to predict one. When it uses only one independent variable, the method is called Linear Regression, and when there is more than one variable, the method is called Multiple Regression [24].

2.2.2 Gradient Boosting

Gradient Boosting is an ensemble ML approach that is built in a sequential way [53]. This technique is based on the use of weak learners. The basic idea is to add a new model in each iteration to improve the previous models. It is important to highlight these models are not independent of each other the new model that will be added is based on

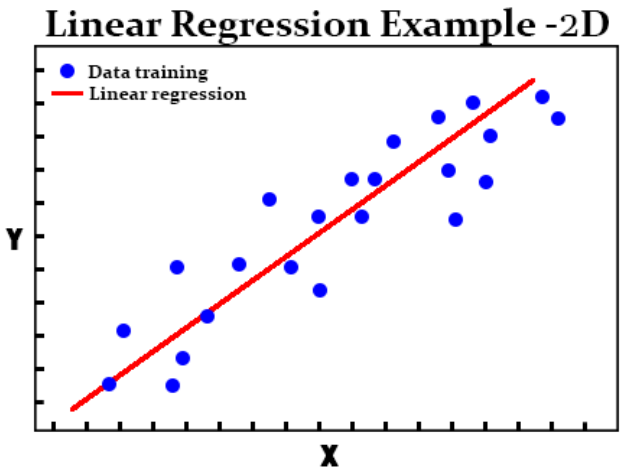

Figure 2.4: Example of linear regression. X is the independent variable, Y is the dependent variable, blue points are the examples in the training set, and the red line is the function found by the model.

the previously built base models. Each model is created based on the called *residual*, which is, in summary, the set that contains the error between the model prediction and the ground-truth value. It happens because these residuals encode the errors the previous base model makes.

Furthermore, Gradient Boosting uses a hyperparameter to control the influence of the models in the produced predictions, which is called the learning rate. This parameter is meant to prevent overfitting. A tutorial on Gradient Boosting is available in [54].

2.2.3 KNN

The algorithm K-nearest neighbors (KNN) is a probability algorithm used for classification problems. The central concept of this algorithm is that the objects related to the same concept are similar. Similar objects are concentrated in the same region of the data space, given a positive integer K and a test observation x_0, the KNN classifier first identifies the neighbors K points in the training data that are closest to x_0, represented by N_0. With this, the algorithm sees the predominant class in N_0 as this class. Because of this, the choice of K drastically affects the KNN classifier obtained [51].

The K-nearest neighbors regression (KNR) is the version for regression of the KNN.

Still, instead of selecting the most probable class in the K nearest point of x_0, the algorithm estimates the value of x_0 using the average of all the training responses in N_0 [51].

2.2.4 Decision Tree

A decision tree represents a function whose input is a vector of values and returns an output [55]. There are several algorithms to generate the decision tree, and each has particular ways of defining how to make the decision tree [56]. An example of a decision tree is given in the sequence[6]. Figure 2.5 presents a dataset, with features (blue), their values (green) and the target (yellow). When this data is used to train a decision tree, the result is something like the tree in Figure 2.6, where each internal node (blue) tests a feature, each branch (green) represents the value of the feature (tested by the node). The leaf node (yellow) defines the final classification.

Train Examples					
Day	Aspect	Temp	Humidity	Wind	Play Tenis
D1	Sun	Hot	High	Light	No
D2	Sun	Hot	High	Stong	No
D3	Clouds	Hot	High	Light	Yes
D4	Rain	Mild	High	Light	Yes
D5	Rain	Fresh	Normal	Light	Yes
D6	Rain	Fresh	Normal	Stong	No
D7	Clouds	Fresh	Normal	Light	Yes
D8	Sun	Mild	High	Light	No
D9	Sun	Fresh	Normal	Light	Yes
D10	Rain	Mild	Normal	Stong	Yes
D11	Sun	Mild	Normal	Stong	Yes
D12	Clouds	Mild	High	Stong	Yes
D13	Clouds	Hot	Normal	Light	Yes
D14	Rain	Mild	High	Stong	No

Figure 2.5: Example of a dataset used to create a decision tree.

Defining which feature is tested first is necessary to make the decision tree since the order in a tree is very important. So, to select the best feature, it is necessary to define a measure of their importance. The most used is the *entropy*, a measure of the uncertainty

[6]Based on: `http://web.tecnico.ulisboa.pt/ana.freitas/bioinformatics.ath.cx/bioinformatics.ath.cx/indexf23d.html`.

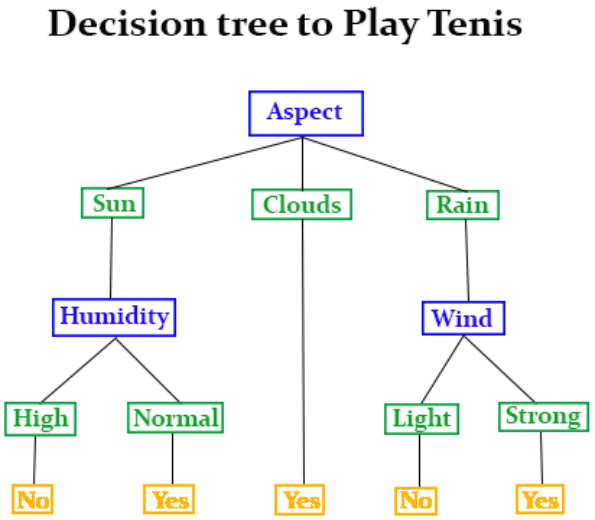

Figure 2.6: Example of a decision tree.

that represents the gain of information [57]. Furthermore, one way to avoid overfitting in trees is the pruning tree method, which limits the tree size [24].

An interesting use of the decision tree is the **Random Forest**, which combines independent trees to make a better decision [58]. This algorithm uses the concept of ensemble learning, and a visual example of a Random Forest can be seen in Figure 2.7.

2.2.5 Neural Networks

The Neural Network (NN) emerged as an attempt to simulate what happens in the human brain. This approach started from the principle that mental activity comprises the brain cells, called neurons, and connections between them, called synapses. In this way, a simple neuron model was developed in 1943 [59], as the model presented in Figure 2.8. The neuron model has several inputs; each input has its weight, and the input value is multiplied by the corresponding weight. Next, all the values are summed, and another value, bias, is added. So, the final sum goes to an activation function, and an output is generated. An implementation of the NN is the Multilayer Perceptron (MLP), which is a fully connected neural network and was presented in [60].

A NN is a set of these neurons, as presented in Figure 2.9. In this image, it is possible

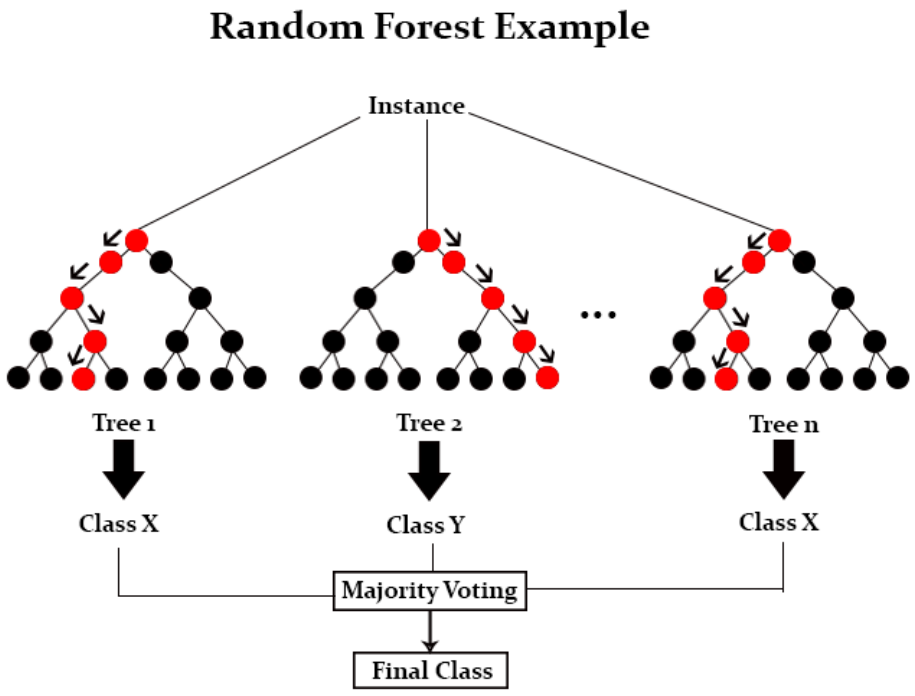

Figure 2.7: Example of random forest. The instance has n trees, and each tree makes its prediction independent (red nodes). In the end, these predictions are joined, and the class that gets more votes is the final prediction.

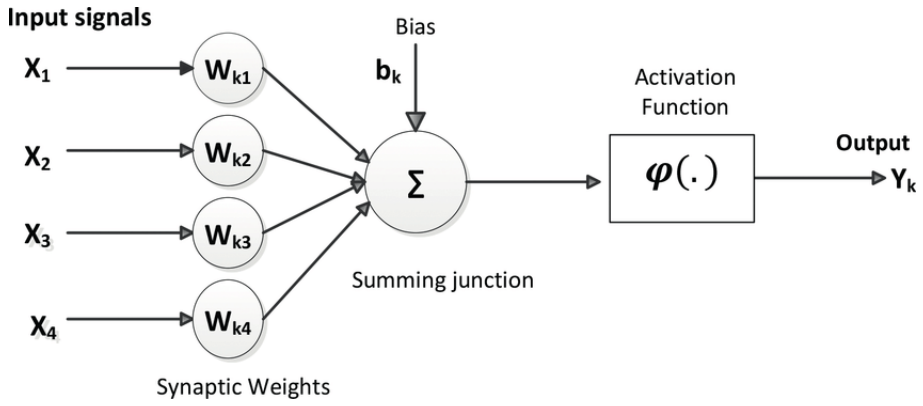

Figure 2.8: Neuron model. The model has several inputs, each one with a weight associated. Each input is multiplied by its weight, all the results are summed, and a special value called bias is summed together. Finally, the resulting value is put in an activation function, the neuron's output. Source: [59].

to see the inputs, the hidden layers (the picture has just two, but can be more layers on the hidden layers), and the outputs.

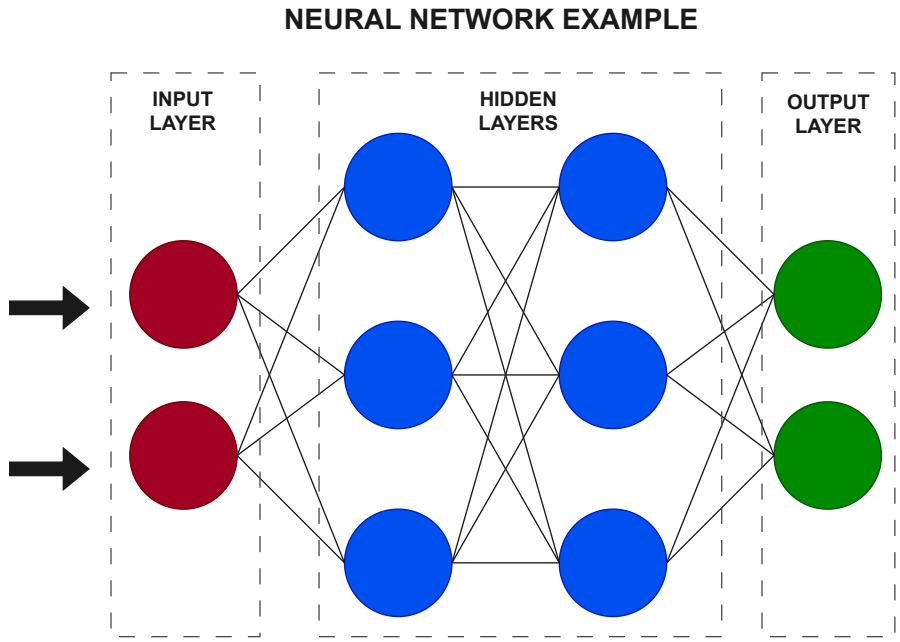

Figure 2.9: Neural Network representation with three main layers: Input Layer (yellow), Hidden Layers (blue), and the output layer (green). It is important to highlight the hidden layers can be composed of several layers, not only two, as in this example.

The sequence of putting the inputs in a NN and each neuron output is an input for the neurons in the next layer are called *feedfoward*. Furthermore, one of the possible methods to define the best weights and biases is the *back-propagation* [61]. An important parameter of this method is the *learning rate*, which defines how fast the network can adapt the weights and biases.

The activation function is used to get the result of the node and add non-linearity to the neural network. There are several activation functions, and the choice of which one to use depends on the context in which it is applied. The most used are: ReLu activation function, described by

$$f(x) = max(0, x) \tag{2.2}$$

, Sigmoid activation function, described by:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

, and Tanh (hyperbolic tangent) described by:

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.4}$$

, where x is the input of the function. Figure 2.10 presents a comparison between these functions [24].

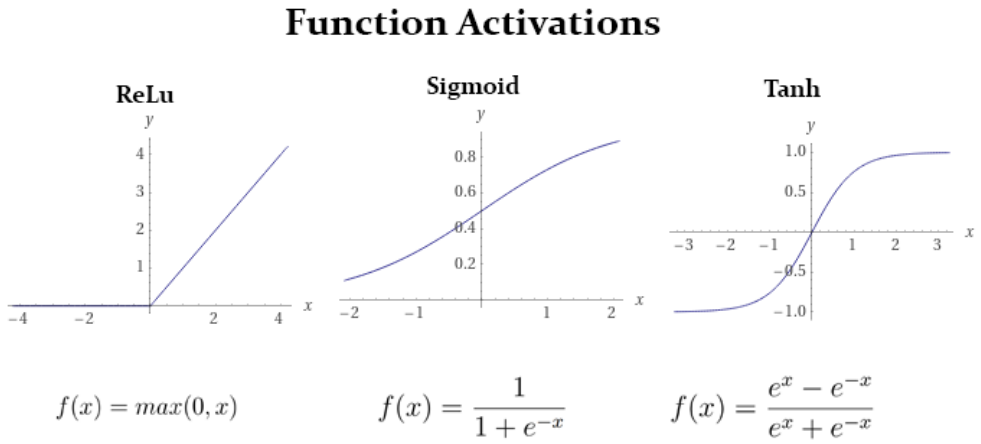

Figure 2.10: Compare between ReLu x Sigmoid x Tanh activation functions.

An interesting approach in the Neural Networks is the CNNs, a specialized neural network for processing data with a known grid-like topology [50]. This method involves applying filters to reduce the image size and finding patterns to identify the image's content. In general, CNN are NN that use convolution in place of general matrix multiplication in at least one of their layers [52].

Compared to simple NN, which involves several hidden layers, CNN consists of many layers, and this characteristic allows it to represent highly nonlinear functions compactly. It can learn complicated functions that represent the relations but require large architectures. CNN involve a lot of connections, and the architecture usually comprises various

layers, including convolution, pooling with thoroughly connected layers, and regularisation [39].

Simply, CNN comprises feature extraction and classification. The first one comprises convolutional layers, which apply filters, and Pooling layers, which help reduce the representation size, speed up the computation, and make the features more robust. The Fully connected layers make the classification (or regression). It is possible to represent the architecture of a CNN graphically. Figure 2.11 presents the architecture to the VGG16 model, introduced in [62].

According to the authors of [62], the structure of the VGG16 model was initially presented as a fixed input ConvNet (224x224 RGB image) requiring only normalization of the image pixel values as pre-processing. Compared to the other models presented, this structure exhibited an extension of 16 weight layers (13 convolutional layers and three fully-connected layers) with increasing depth and 3x3 filtering to extract features from the input picture successfully. A 2x2 max-pooling layer is used at the end of the filter stack to minimize the information volume. In the model trained using the ImageNet[7], the final max-pooling layer is linked to a fully connected layer that comprises 4096 neurons. The output of this layer is then passed to a softmax layer for 1000 classifications. Figure 2.11 presents the VGG16 architecture graphically.

Transfer Learning is a newly developing area in the ML context. In general, ML algorithms are addressed on only one specific task. In this way, Transfer Learning attempts to change this by developing methods to transfer knowledge learned in one or more source tasks and use it to improve learning in a related target task [64]. There are several models[8] that had been already pre-trained and can be used as on a Transfer Learning, for example, the GoogLeNet [42], used on [41], and VGG16, explained before.

[7]Details in `https://www.image-net.org/`.
[8]Some examples are available at `https://keras.io/api/applications/`.

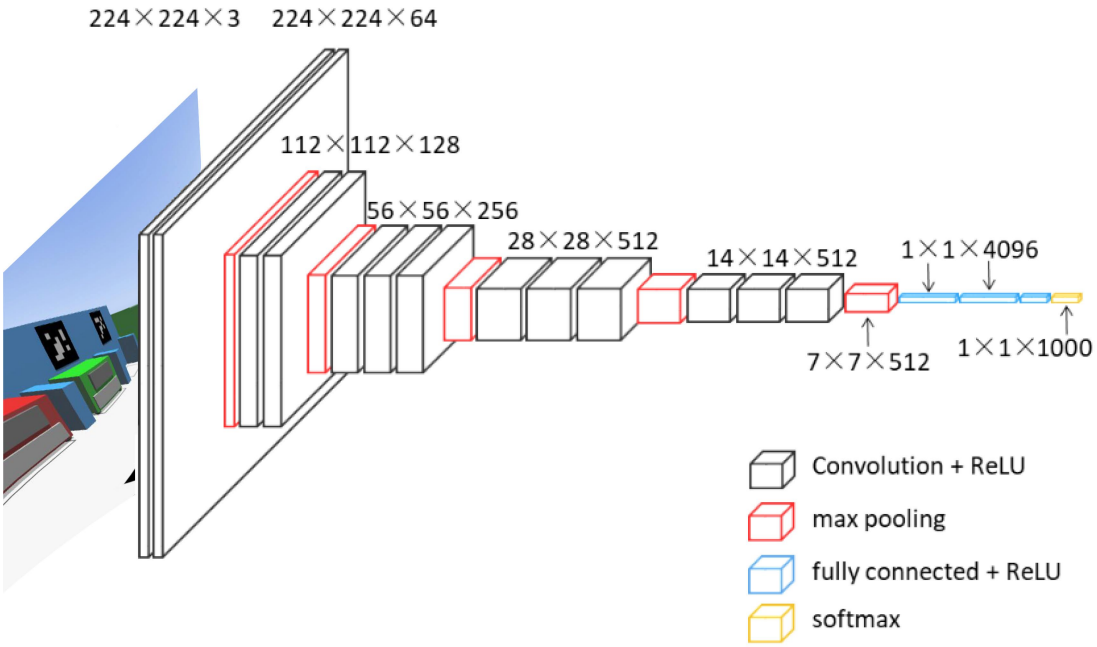

Figure 2.11: VGG16 architecture. Adapted from: [63].

Chapter 3

Methodology

This chapter is divided into four subsections, focusing on explaining the problem, the concept solutions, the metrics used for evaluation, and practical implementation. Section 3.1 presents a detailed problem description. Section 3.2 presents the approaches to the problem. Section 3.3 presents some metrics that compare the model's performance. Finally, Section 3.4 presents the implementation of the approaches.

3.1 Problem definition and Research Question

There are several robot competitions with different goals. This work focuses on the RaF, presented in detail in Section 1.1. The present work also focuses on mobile robot localization in the context of that competition. The robot has some onboard sensors, including a camera. Figure 3.1 presents an example of an image captured by the robot's camera. Three main research questions were proposed to guide the work's development. They are:

1. Can machine learning approaches effectively estimate the position and orientation of a robot solely based on camera images in a structured environment with the presence of fiducial markers?

2. Among different types of machine learning approaches, such as MLP, Random Forest (RF), Gradient Boosting (GB), KNR, and CNN, which approach leads to the smallest pose error in estimating the robot's position and orientation based solely on camera images in a structured environment with fiducial markers?

3. Considering the constraints of embedded devices with limited memory and computing power, which type of machine learning approach is better suited for achieving accurate pose estimation in a structured environment with fiducial markers?

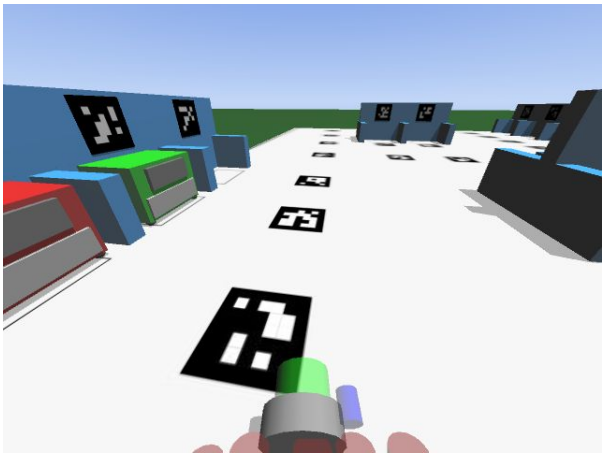

Figure 3.1: Example of a robot camera image.

3.2 Concept Solutions

To answer the research questions, three solutions to the localization problem will be implemented and investigated; two rely on the relative pose of identified ArUco markers in an image, while the other one is based on the direct use of the images. The following steps will be followed for each of the three solutions: gathering data from a simulator, training a model using this data, and then applying the trained model in a real environment. The following sections describe each concept solution in detail.

3.2.1 Concept 1: Using Machine Learning to Generate one Pose Estimation per Image with tags

The first concept solution involves providing a single pose estimation for an image captured from the environment. To achieve this, a unique pre-processing step is required. The OpenCV library is utilized to detect the ArUco markers present in the image. Subsequently, the same library returns each marker's ID and relative pose concerning the camera. Two arrays, corresponding to the rotation and translation, represent the relative pose, called *rvec* and *tvec*. Each array is composed of three elements; each one is an axis (x, y, z). The *tvec* values are in the units given by the camera's specifications, such as millimeters, while the *rvec* values, which is the rotation vector expressed in axis-angle format[1].

Next, all the relative poses of the ArUcos are collected and aggregated into a matrix comprising 49 rows and seven columns. Each row represents a distinct ArUco. The first row corresponds to the ArUco with ID 0, the second row to the ArUco with ID 1, and so on. The columns contain six elements from the *rvec* and *tvec* arrays, and the 7th element is a Boolean value indicating if that particular ArUco was identified or not. If an image does not capture a specific ArUco, the corresponding row in the matrix will be assigned a value of 0.

Therefore, the proposed solution consists of three distinct models, each one dedicated to estimating one variable (x, y, and θ). This separation was made to improve the quality of the estimations, given that these variables are uncorrelated. Thus, the overall process involves receiving an image, performing a pre-processing step to create a matrix containing the Aruco's relative pose information, and then using this matrix as input to the three independent ML models, which will generate the estimations for the robot's pose. Figure 3.2 shows a graphical representation of this flow.

[1] `https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html`.

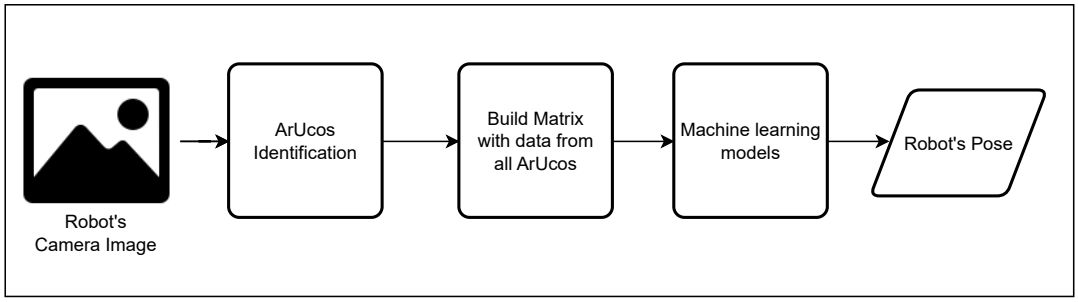

Figure 3.2: Diagram representation of concept 1.

3.2.2 Concept 2: Using Machine Learning to Generate one Pose Estimation per Tag Identified

In concept 2, the model will return one pose estimation for each ArUco identified in that image, i.e., if there are X markers, X different poses will be generated. The same OpenCV library used in Concept 1 is used here. The difference is that, instead of aggregating all the ArUco's relative poses in one matrix, the information is directly applied to the models resulting in one pose estimation per ArUco. Figure 3.3 presents this flow graphically. Again, three independent models are used in machine learning, one for each variable of the pose (x, y, θ).

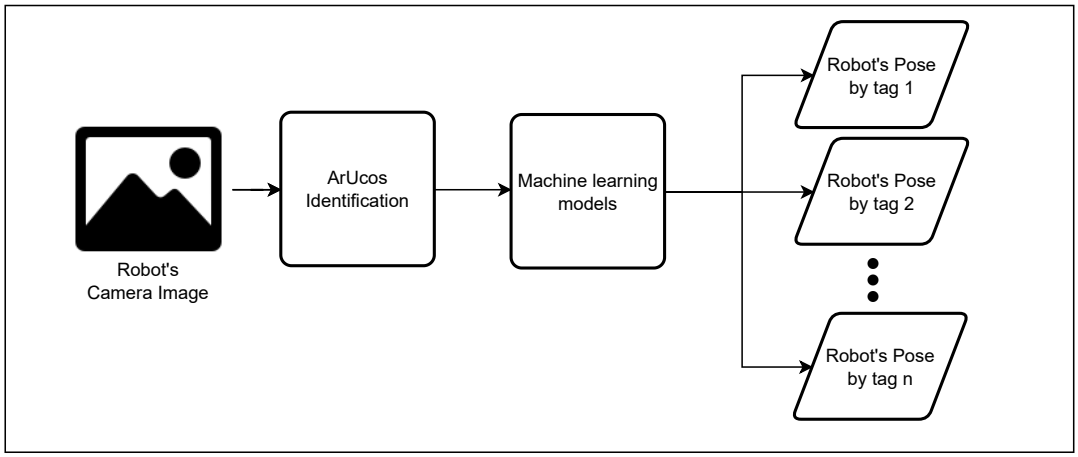

Figure 3.3: Diagram representation of Concept 2.

The resulting estimations can be aggregated through the use of filters, such as the Extended Kalman Filter, and improved with the use of odometry. Such aggregation was already developed in [3], and it is not the focus of this work.

3.2.3 Concept 3: Using CNN to Generate one Pose Estimation per Image without Tags Identification

The third concept proposed is based on the direct use of the images without the necessity of Aruco's identification. So, in this proposal, an image taken by the robot's camera is pre-processed (changing, for example, the image size), and it is used as input for a CNN model, which returns the pose. The flow of the process is presented in Figure 3.4. The CNN used is based on the transfer-learning concept. Besides, two models will be developed, but with the same architecture: One to estimate the position (x, y) and another to estimate the orientation (θ).

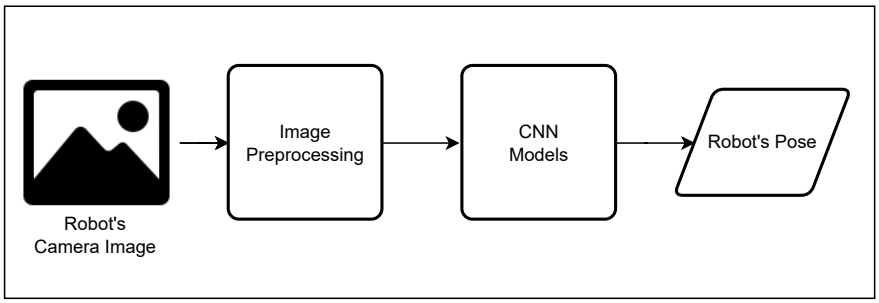

Figure 3.4: Diagram representation of concept 3.

3.3 Performance Evaluation

There are several metrics used to measure the quality of the predictions. It is essential to highlight the problem in this work is addressed as a regression and not a classification. So, there are specific metrics when the target is a numeric continuous value. In this work, the metrics used to verify the quality of the predictions are: Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), Normalized Root Mean Squared Error (NRMSE),

and R^2. The following sections describe these metrics.

3.3.1 Mean Absolute Error - MAE

The MAE of a model is associated with a test set and is the mean of the absolute values of the individual prediction errors on overall instances in the test set [65]. Each prediction has an associated error and is the difference between the predicted and true values. The MAE function is defined by:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{3.1}$$

, where y_i represents the true value and the $\hat{y}_i$ represents the predicted value for the instance i. The best value possible for MAE is 0, and the worst value is $+\infty$.

3.3.2 Root Mean Squared Error - RMSE

The RMSE metric is derived from another common metric called Mean Squared Error (MSE). The MSE of a model concerning a test set is the mean of the squared prediction errors over all instances in the test set [66]. The function for this error is defined by:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.2}$$

, where y_i represents the true value and the $\hat{y}_i$ represents the predicted value for the instance i. The RMSE, defined by:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3.3}$$

, consists in the root of the MSE. The best value possible for RMSE is 0, and the worst value is $+\infty$. Furthermore, another interesting metric derived from the RMSE is the NRMSE, which is defined by:

$$NRMSE = \frac{RMSE}{y_{max} - y_{min}} \tag{3.4}$$

, where y_{max} and y_{min} represent the maximum and minimum value, respectively. This metric is interesting because, with this, it is possible to compare values with different units.

3.3.3 R Squared - R^2

Another essential metric is called R^2, also known as the coefficient of determination. This metric can result in values in the range $(-\infty, 1]$ according to the mutual relation between the ground truth and the prediction model [67]. The R^2 function is defined by:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \overline{y}_i)^2} \tag{3.5}$$

, where y_i represents the true value, $\hat{y}_i$ is the predicted value, for instance, i and $\overline{y}_i$ is the average of the true values.

3.3.4 Baseline definition

To create a comparison parameter between the models that estimate the pose, it is necessary to define a baseline, the most basic possible model. This model calculates the average of the training set and uses it as the estimation for all cases, doing it for the three target values: *x, y* and θ. For instance, suppose the targets on the training set are T = {1,2,3,4,5}, and the targets at the test set are Ts = {1, 2, 6}. The average of the training set is 3, so all predictions for the test will be 3. Then, calculate the metrics discussed in Sections 3.3.1, 3.3.2, and 3.3.3, it is possible obtain: MAE = 2, RMSE = 2.16, NRMSE = 0.54, and $R^2 = 0$ (by definition). These values form the baseline dataset and are used to evaluate the models and determine if they are better than the simple average.

3.4 Implementation

The development of the work was divided into three parts. Each part is explained in detail in the following sub-sections. Section 3.4.1 presents the validation of the ML feasibility in embedded systems. Section 3.4.2 presents a comparison between some algorithms and resolutions in data collection, focusing on the quality of the estimations and the validation of solutions 1, 2, and 3 proposed in Sections 3.2.1, 3.2.2 and 3.2.3, respectively. Finally, Section 3.4.3 focuses on implementation in the real robot. It is essential to highlight the first two sections only using data from a simulated scenario since the focus is only on validating the approaches.

Some activities are the same for the three parts and will be discussed first. Then, each of the three parts mentioned above will be discussed individually. The first common step is the data collection in the simulator. Thus, the field was discretized in a grid, with each square having a size of 1 cm. So, the robot was positioned on this grid in the center of all available positions, i.e., without obstacles. In each position, the robot takes around 60 images while performing a 360° turn to create a database (due to delays in the simulator, the number of collected images in each turn can vary slightly), with around 350 thousand images in total.

Furthermore, to test other grids' resolutions, a small part of 10 x 10 cm in the center of the field was used to collect data. Figure 3.5 presents this limited part of the field, and the grid's sizes used were: 10 mm, 5 mm, 2.5 mm, and 1 mm. Finally, one last data collection was done in the simulator, considering a random route along the field, and around 500 images were collected.

Following the data collection, the initial preprocessing step involved removing duplicate images and those containing incorrect values resulting from delays in the simulator. Due to the collection of the data being performed in a Virtual Machine and the operation of the simulator (where the images collected are passed through a network before being saved), some images may be saved multiple times, resulting in subsequent images being taken out of sync with the camera's pose.

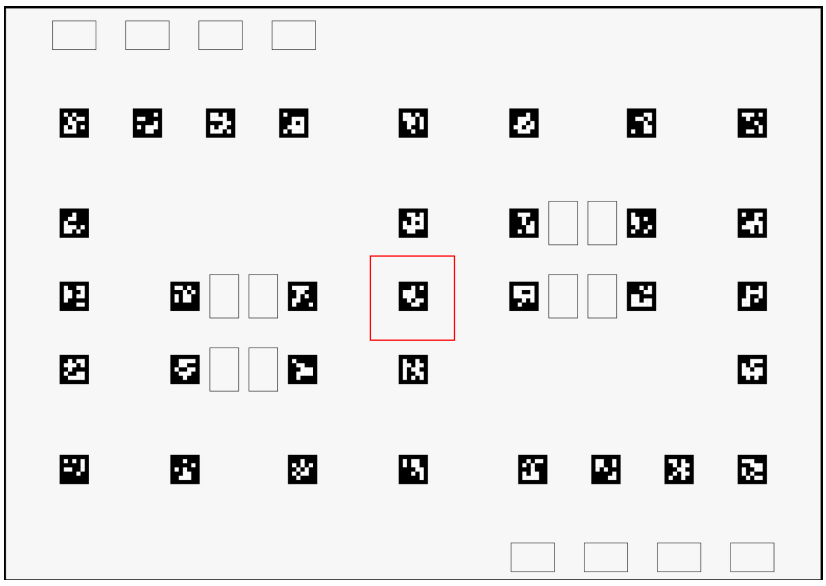

Figure 3.5: The red square in the center shows the part of the field used to collect images for the tests to show the relationship between the grid resolution and errors. Source: [4].

Furthermore, it is essential to highlight that only images containing at least one ArUco were used during training and testing. This is important to avoid problems with "ambiguous images", as presented in Figure 3.6. This image can be associated with any corner of the field. This phenomenon occurs mainly at the field's border when the robot has the camera pointing outside the field. This treatment is essential because these images do not contribute to the training and evaluation of the models. Moreover, it is necessary to emphasize that this problem can be solved with the complete localization system implementation, as presented in [3]. In this work, such implementation is not considered since the focus is only on validating the ML approaches.

To create the datasets, the images collected will be processed using the OpenCV library[2] to identify the ArUcos[3] and to estimate the pose of each ArUco relative to the camera's reference frame. The OpenCV library returns arrays for each marker with the position and the marker's orientation to the camera. As previously mentioned, these arrays are called *tvec* and *rvec*, respectively. More details about the ArUco identification

[2] `https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html`.
[3] In this work, the version used was 4.6.0.

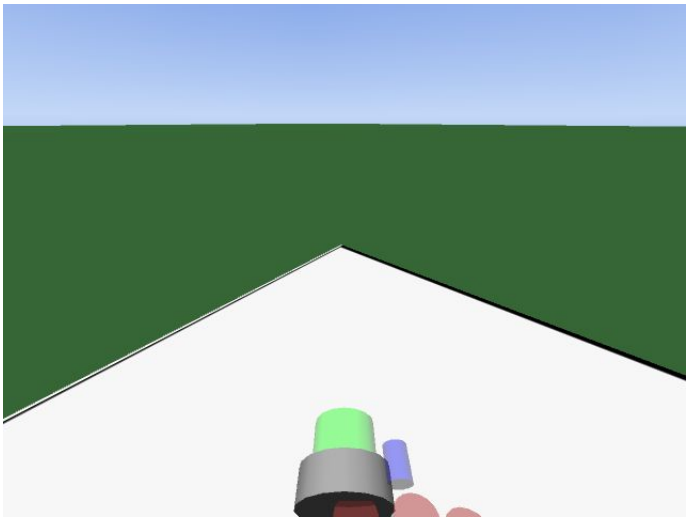

Figure 3.6: Example of ambiguous image.

can be seen in [3]. Considering the images collected, some different datasets have to be created. They are [4]:

- **Dataset A** - *Considering the whole field and combining observations' attributes in a matrix format*: To generate this dataset, the images used were collected across the entire field, considering the grid resolution equal to 1 cm. Each observation within the dataset represents a single image containing one feature and three targets. The feature is comprised of a matrix with 49 rows, each representing an ArUco marker, and seven columns that represent the *tvec* and *rvec* arrays, along with a boolean value indicating whether the marker was detected in the image. If a particular ArUco marker is absent from an image, the corresponding row in the matrix is filled with 0. The target variables are x, y, and θ, respectively.

- **Datasets B** - *Considering the collection in part of the field with different grid's resolutions and combining observations' attributes in a matrix format*: Dataset B is composed of four datasets. To generate each of these datasets, images taken at the center of the field (Figure 3.5) were used, varying the grid resolutions. Each dataset was composed of images from different resolutions: **B1** for 10 mm, **B2** for 5 mm, **B3** for 2.5 mm, and **B4** for 1 mm. Each dataset was produced using the

same procedure as Dataset A.

- **Dataset C** - *Considering the whole field in an ArUco's array*: To generate this dataset, images taken across the entire field were used, with a grid resolution of 1 cm. However, unlike datasets A and B, each observation in this dataset represents a detected ArUco marker rather than an entire image. As such, each observation consists of seven features: the marker's ID, *rvecs*, and *tvecs*, as well as three targets: x, y, and θ. In this dataset, the image information is no longer relevant, as the focus is solely on the relative pose of the ArUco markers.

- **Dataset D** - *Considering the collection in part of the field with different grid's resolutions and combining observations' attributes in ArUco's array*: Dataset D is composed of four datasets, as Dataset B. To generate these datasets, images taken at the center of the field (3.5) were used, varying grid resolutions. Each dataset was composed of images from different resolutions: **D1** for 10 mm, **D2** for 5 mm, **D3** for 2.5 mm, and **D4** for 1 mm. Each dataset was produced using the same procedure as Dataset C.

- **Dataset E** - *Considering a random route in an ArUco's array*: This dataset was created using images captured across the entire field, using a random path. The same process as Dataset C was used to generate this dataset.

3.4.1 Part 1: Feasibility of the ML in Embedded Systems

This part involves verifying and validating the feasibility of using ML in an embedded system to implement a vision-based localization system. To accomplish this, a Raspberry Pi 4 Model B was used for validation. A previous study in [68] investigated the feasibility of employing ML in a Raspberry Pi 3 Model B. In that study, the authors applied RF, MLP, and Support Vector Machine (SVM) to both regression and classification problems. The authors determined that all algorithms achieved high accuracy (exceeding 80%), with an inference time of less than one millisecond and notably lower energy consumption than

other activities, such as web browsing and video watching. However, the authors did not discuss the size of the models used.

Using that previous work as a reference, a case study explored the localization problem at RaF. The study employed the same three algorithms as [68], focusing on the localization problem and using the concept presented in Section 3.2.2. To validate the feasibility of the models in an embedded system, a Raspberry Pi 4 Model B[4] was used, and only images from the simulator and collected from a limited portion of the field with a grid resolution equal a 10 mm (dataset D1) were employed. The evaluation metrics included training and execution times, energy consumption (in mWh), and model size. An Atorch USB tester was used to measure energy consumption, as shown in Figure 3.7. The complete details regarding the feasibility work are available in [7].

A USB Tester is a device project to check the functionality and performance of USB connections, giving information about voltage, current and other parameters. The USB tester has a display that shows the relevant information during the execution.

Figure 3.7: Atorch USB Tester.

To perform the tests, the hardware utilized was utilized a Raspberry Pi version 4 Model B with a Broadcom BCM2711, Quad-core Cortex-A72 (ARM v8) 64-bit SoC @

[4]Further information available at:
https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/.

1.5GHz1, 4 GB RAM, and 40 pins. The operating system used was the Raspberry PI OS[5], also known as Raspbian, which is a 32-bit OS, open-source and based on Debian Linux. The image of the utilized Raspberry Pi can be seen in Figure 3.8.

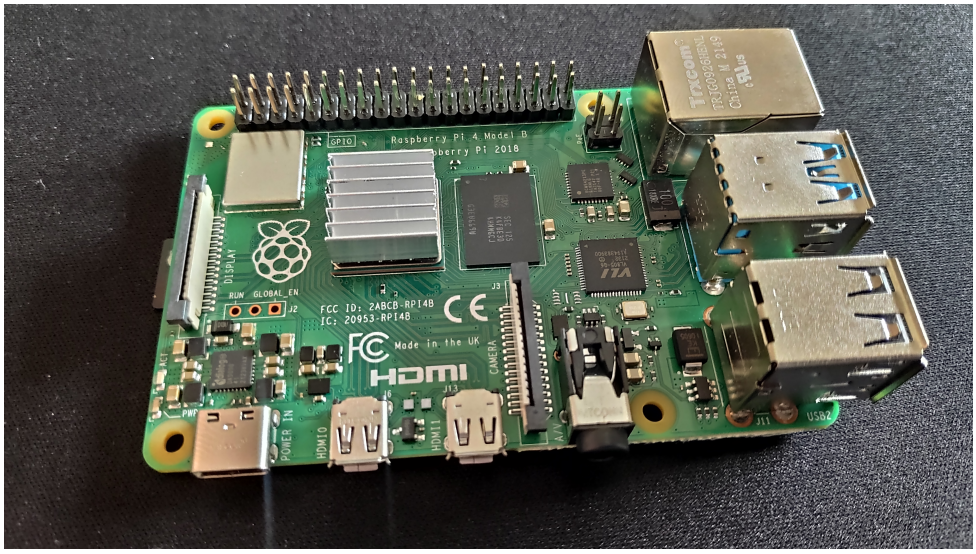

Figure 3.8: Raspberry Pi 4 Model B.

3.4.2 Part 2: Quality of ML models

The second part of this study was dedicated to evaluating the quality of the models, as documented in [4]. Similar to the first part, data from the simulator was used to conduct tests to establish a parameter for comparing the quality of the models. This part was divided into two: First, in Section 3.4.2, four algorithms, namely Multi-layer Perceptron (MLP), Random Forest (RF), KNR (KNN regressor), and Gradient Boosting (GB), were used to verifying the feasibly of the solutions proposed in 3.2.1 and 3.2.2 and the accuracy of the estimations. Second, in Section 3.4.2, a CNN model, based on the pre-trained VGG16 model, was used to validate the concept solution explained in 3.2.3.

[5]More at `https://www.raspberrypi.com/software/`.

Approach 1: ML techniques using fiducial markers

A CPU with an AMD EPYC 7351 16-Core Processor (2.40GHz) and 32 GB of RAM was used to perform the first tests using machine learning techniques. The metrics utilized to evaluate the performance of the models are described in Section 3.3. It is essential to highlight the hyper-parameters were not modified during the tests; all Scikit-learn (version 1.1.2) default definitions were used.

The initial examination in this study involves contrasting the various methods using dataset A. The proposed solution is outlined in Section 3.2.1, while Section 3.3.4 describes the definition of a baseline for comparison with the four techniques. To achieve this, the dataset was partitioned into two sections: 85% for training and 15% for validation. A visual depiction of this procedure is shown in Figure 3.9.

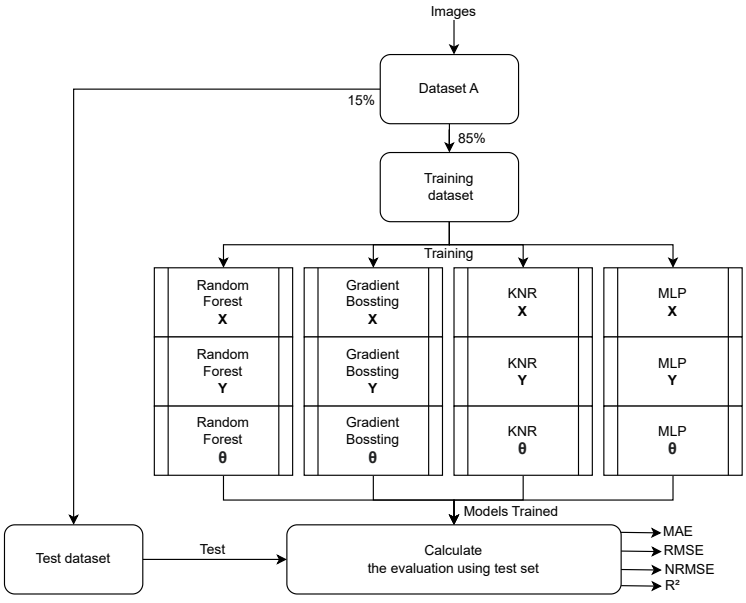

Figure 3.9: Flow process. Source: [4].

The subsequent evaluation in this investigation involves comparing the performance with different grid resolutions. The algorithm that generated the best results in the earlier analyses was used. Consequently, data from datasets B1, B2, B3, and B4 with corresponding resolutions of 10 mm, 5 mm, 2.5 mm, and 1 mm were used to train this

algorithm, and an analysis of its outcomes was conducted. Section 3.2.1 presents the solution employed in this analysis.

The concluding assessment in this study entailed comparing analytical and ML techniques. The algorithm that yielded the best results in the previous analysis was employed to achieve this. However, this time trained on dataset C and evaluated on dataset E. Section 3.2.2 presents the concept solution employed in this comparison. This comparison is significant because the analytical approach was introduced in [3] and provides a good benchmark. Notably, the results obtained in this phase are not the final pose estimations to be used in the robot, as the values need to be subjected to several filters to enhance the accuracy of the estimation, as outlined in [3].

Approach 2: CNN technique

The second part of the process of the validation approaches is using the CNN technique. The proposed model is based on the transfer learning concept, using the pre-trained VGG16 model. To adapt it to our problem, the last layers of the model (dense and softmax) were removed, and three dropouts (with factor 0.2) and three dense layers (2 with 4096 neurons and 1 with 1072) were added. Furthermore, an output layer was defined as a linear activation function with 1 or 2 outputs (with one output for the model to estimate θ and with two outputs to estimate x and y). It is essential to highlight that only the new layers were trained, and all the original weights of VGG16 were maintained, i.e., no fine-tuning was done. Figure 3.10 presents the adapted model.

Two tests were performed considering this model: first, data collected from the whole scenario (as Dataset A), and second, the data collected from the limited part of the scenario (as Datasets B1, B2, B3, and B4). However, the difference is the preprocessing applied to the data. While in the cited datasets, the preprocessing necessary was based on the ArUcos identification. This preprocessing does not make sense for the CNN technique because the model's input is directly the images. So, the preprocess applied exclusively to this part of the work was considered the preprocessing function specific to the VGG16

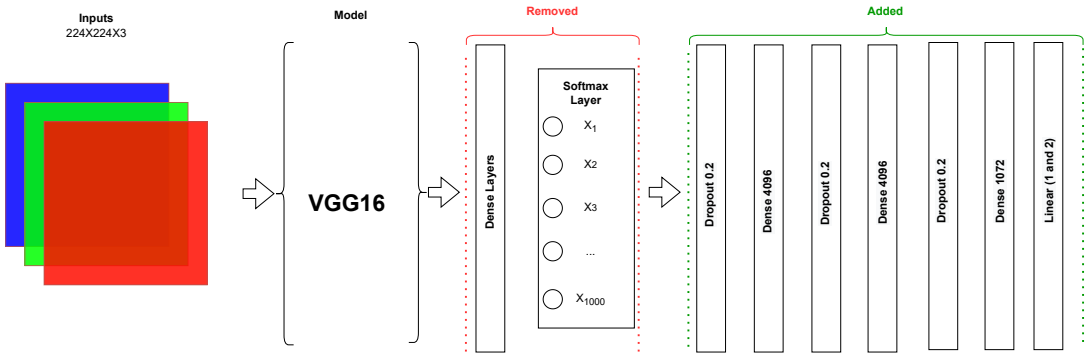

Figure 3.10: Architecture the proposed CNN model.

model[6]. Besides this, the data division was defined as 80% training, 10% validation, and 10% testing.

Besides this, the x and y values were multiplied by 1000 before the training (since the values were small to present relevant improvements in the training process). After the estimation, the values were divided by 1000. In addition, other settings in the CNN were defined, such as it was set 200 epochs for execution, with an adaptative Learning Rate[7] with the initial value of 0.0005 (and the metric used to monitoring was the MAE, with a reduction factor of 0.8, patience of three epochs and minimum values of 0^{-8}). Finally, an early stopping was defined as ten epochs.

All the tests in this part were performed using a GPU NVIDIA A100, with 16GB. The operating system used was Ubuntu 22.04.2 LTS (Jammy Jellyfish), the Python version used was 3.10.6, and the libraries used were: Pandas 2.0.0 and Tensorflow 2.12.0. The CUDA version used was 11.8, with CUDNN 8.9.1.

3.4.3 Part 3: Execution in the Real Scenario

The third and final part of the work is based on the implementation of models in a real scenario. This part aims to use the models trained based on the concept solution 1 and

[6]More detail at
`https://www.tensorflow.org/api_docs/python/tf/keras/applications/vgg16/preprocess_input`.

[7]`https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html`.

2, with data from the simulator in the real environment, and check the results. Since another team developed the real robot, and the whole localization system as well [3], the ML models must be adequate to run in the previously developed localization system. The approach selected to be used (due to the size of the trained model, as presented in 3.4.1) was the MLP, with and small optimization of the hyper-parameters and without the optimization. So, the first concept solution applied was presented in Section 3.2.2 (one pose estimation per ArUco identified), and the second one was presented in Section 3.2.1.

The optimization used to show the difference in the results with the standard and optimization models was not the best one and was done using the GridSearchCV function[8] based on the Dataset D1. The parameters were adjusted and tested, resulting in the following:

- $hidden\_layer\_sizes$ = (100,100,100,100,100)
  max_iter=500
  $learning\_rate$="adaptive"
  $learning\_rate\_init$=0.01
  $early\_stopping$=True

To perform a realistic test, the simulator was adjusted to simulate the real aspects of the real robot, especially the camera configurations. The first set was about the camera's position. It was moved to $[x = 0.133, y = 0.00, z = 0.075]$, with all values in meters relative to the robot's frame. In addition, it was necessary to perceive the angle rotation representation utilized in the simulator. To find this information and the values, several empirical tests were performed in the simulator, and it was found that the angles are represented in the Trait-Bryan notation [69] defined in the form of ZYX, which means $[yaw = 0.000, pitch = -21.093, roll = 2.800]$, respectively. Finally, to adjust the Field of View (FoV) of the robot's camera, the value was adjusted empirically, comparing the robot image with the image obtained in the simulator. The value set in the simulator

[8] `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html`.

was 70 in the focal view field, representing 110 degrees in FoV. Figure 3.11 presents an example of the image obtained in the simulator (left) and in the real robot (right) with the robot in the same position.

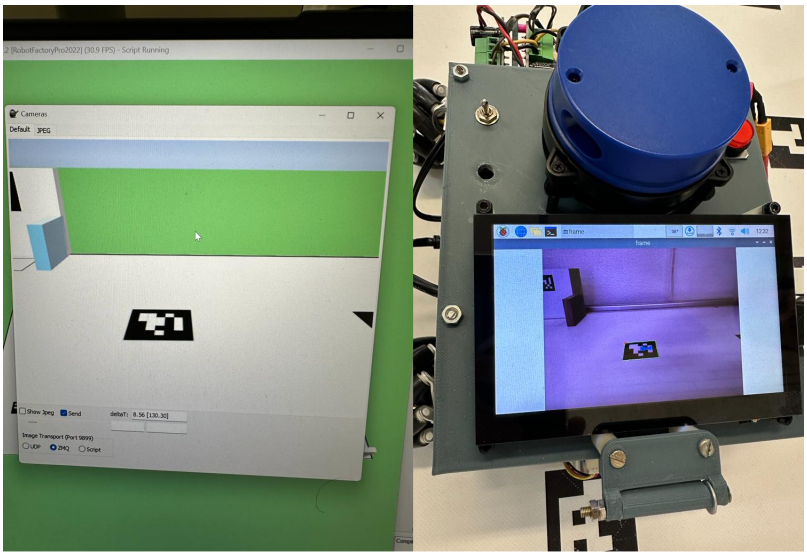

Figure 3.11: Example of the camera's image in the robot in the simulator (left) and in the real scenario (right).

So, after the adjustments, the data were recollected in the simulator, considering the whole environment. The data preprocessing was the same as done to Dataset C. So, the MLP models were trained with these new data, and the models were tested in the real robot simultaneously with the analytical approach.

An external system (already developed in [3]) was used to collect the ground-truth data, i.e., the real pose of the robot. This system consists of a camera placed on the top of the field, which can detect all of the tags of the environment and calculate their poses. So, using this and placing two tags on the robot, the system can state the real pose of the robot. In addition, this system presents a millimetric error in the x and y measure and a fractional error in orientation [3].

Figure 3.12 presents the whole scenario with the field, the robot with the tags, and the camera placed on the top. Besides, Figure 3.13 presents the software with the image from the system's camera and the robot's pose.

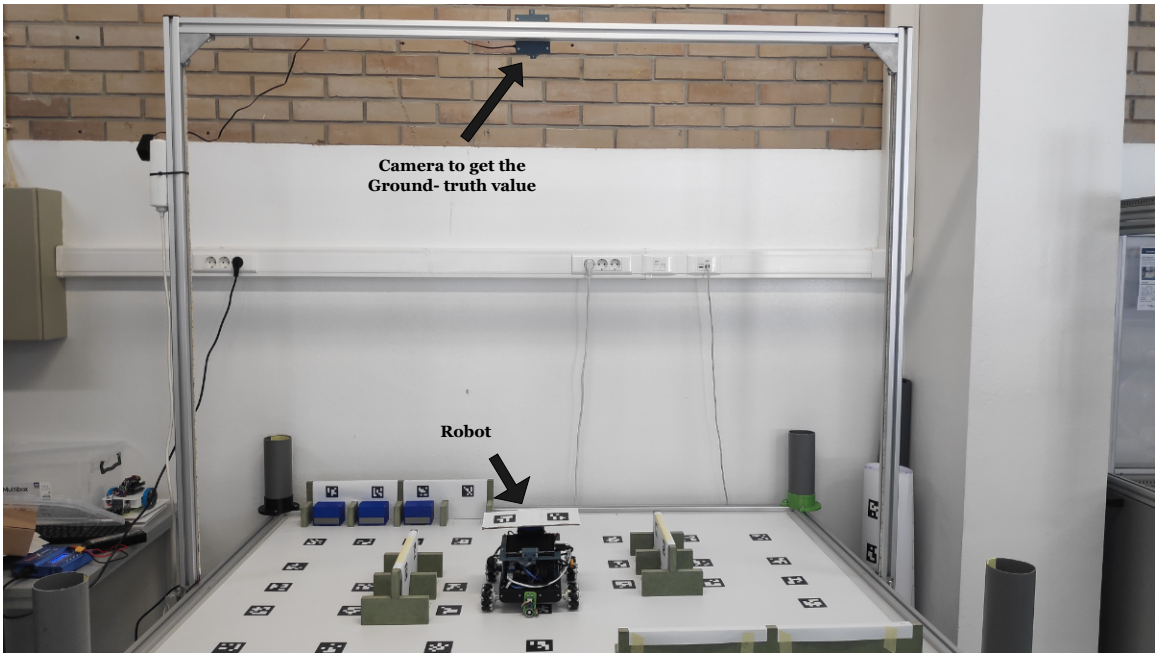

Figure 3.12: Real scenario where the tests were performed, indicating the system's camera used to collect the ground truth data and the robot.

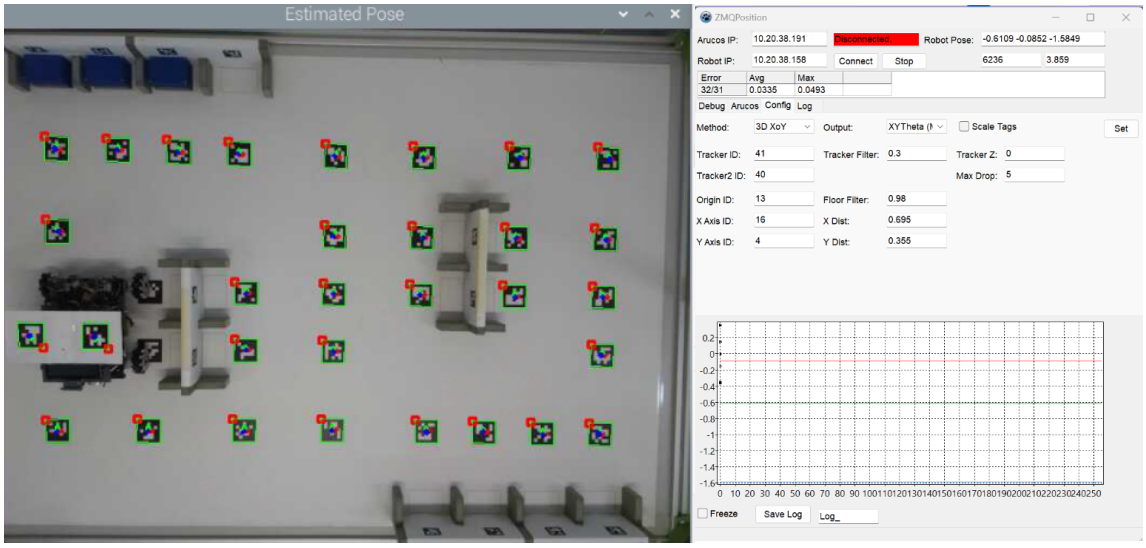

Figure 3.13: Ground truth system utilized.

So, the ground-truth system was integrated with the real robot (present in Figure 3.14) via sockets UDP, and the MLP models were integrated with the analytical approach to estimate the positions. So, all these data were saved during the execution of the robot, following the route presented in Figure 3.15, and after, the data were analyzed.

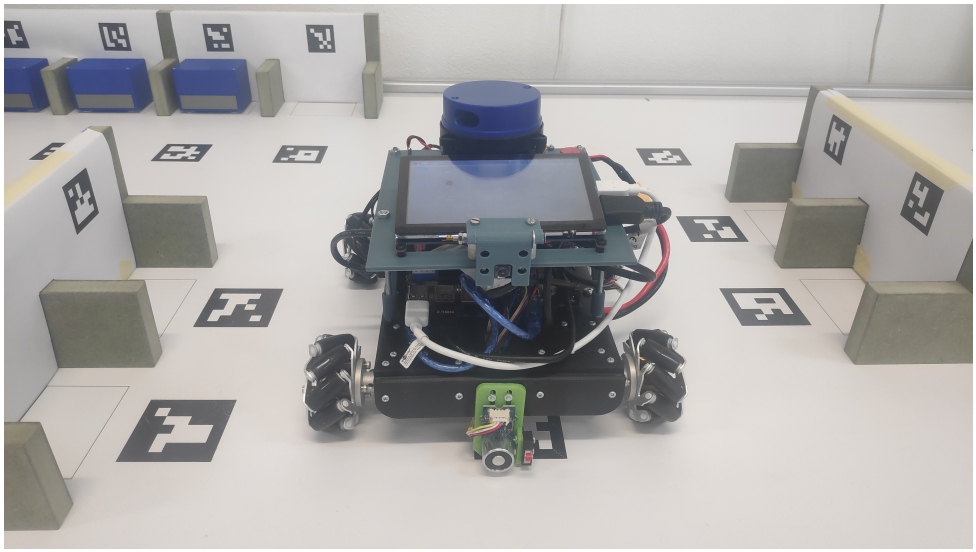

Figure 3.14: Real robot developed for RobotAtFactory 4.0 Competition.

After the data collection, the four MLP models were trained: Considering Concept Solution 1, with and without the optimization, and; Considering Concept Solution 2, with and without the optimization.

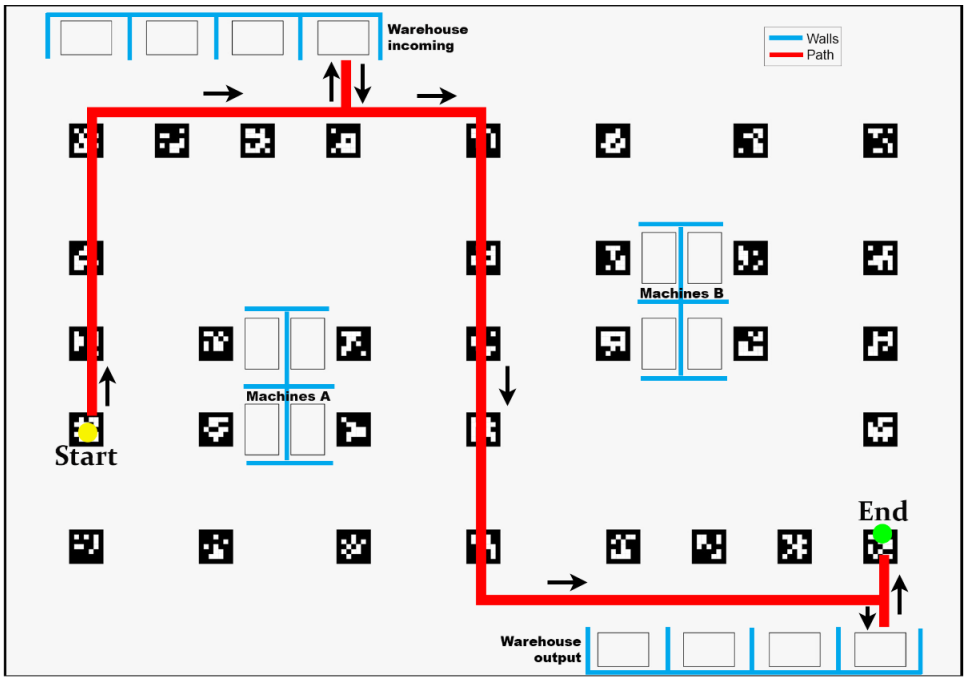

Figure 3.15: Route performed by the real robot.

Chapter 4

Results and Discussions

As presented in Chapter 3, this work is divided into three parts, with different perspectives and approaches. So, the results and discussions are presented in three different sections, one for each part. Section 4.1 presents the results to part 1 (Section 3.4.1), Section 4.2 to part 2 (Section 3.4.2), and finally Section 4.3 to part 3 (Section 3.4.3).

4.1 Results and Discussions Part 1: Feasibility of the ML in Embedded Systems

This part of the work involves training and deploying machine learning models on the Raspberry Pi to address the localization problem in RaF. The objective is to evaluate the feasibility of implementing ML in an embedded system, considering factors such as training and inference times, energy consumption, and the models' size. The models studied in this part were RF, MLP, and SVM. The dataset D1 was used for testing purposes. The anticipated outcome is that all three models will execute efficiently on the given data, each with a distinct size. While estimation quality was not a primary focus, the evaluation metric will confirm that the models executed and produced results, although they will not be analyzed or discussed. All the results presented here were previously published in [7].

The dataset was split into two parts, with 85% used for training and 15% for testing (this split occurred before the preprocessing). Preprocessing the training data took 157.49 seconds and consumed 180 mWh while preprocessing the testing data took 28.62 seconds and 35 mWh. The left graph in Figure 4.1 illustrates the training and execution times for each model, with each color representing a different model with values expressed in seconds. The right graph in Figure 4.1 compares the energy consumption of each model during the training and execution stages, with values presented in mWh. It is worth noting that the execution process involved pose estimation for 1248 images, and as such, the label used for execution in the graph is *Inference of 1248 images*.

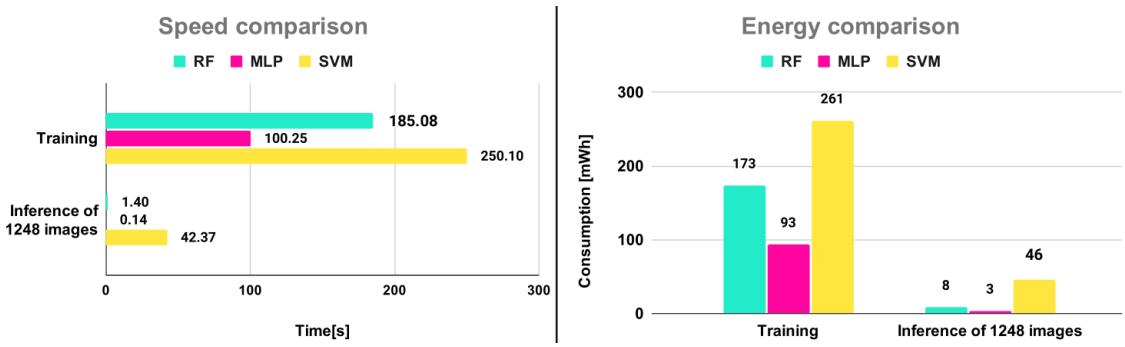

Figure 4.1: Time spent (left) and energy consumption (right) by three ML methods in training and testing. These statistics do not account for the time required to preprocess the data. Adapted from: [7].

An additional outcome concerns both the accuracy and physical size of the models. Table 4.1 provides a comparison of the models, with the Mean Absolute Error (MAE) indicating their respective estimation errors for x and y coordinates in meters and θ in degrees. The model size columns present the sizes of each model in memory, measured in kB (kilobytes).

The analysis of the left graph in Figure 4.1 reveals that the SVM algorithm took longer to train and execute than the other algorithms. On the other hand, the MLP algorithm was the fastest in training and execution. As shown in the right graph in Figure 4.1, the energy consumption pattern follows a similar trend, with SVM being the most costly and MLP being the most efficient. In terms of execution time, all approaches were able to

Table 4.1: Comparisons of the methods regarding memory size and estimating error (MAE). Source: [7].

		RF	**MLP**	**SVM**
Mean Absolute Error (MAE)	**x[m]**	0.009	0.029	0.030
	y[m]	0.012	0.046	0.027
	θ[°]	3.46	28.47	47.64
Size of the models in memory	**x[kB]**	138400	26	1
	y[kB]	147500	26	1
	θ**[kB]**	180000	30	2700

respond to each image within no more than a few milliseconds.

Regarding the quality of the models, the estimation errors obtained were on a centimeter scale, with RF being the best approach. It is important to remember that the models are not optimized, i.e., the standard hyper-parameters of the libraries were used.

The model's size, presented in Table 4.1, is a crucial aspect of the results. The SVM approach produced models for the x and y targets that were very compact, occupying only 1 kB, while the model size for θ was much larger, at 2.7 MB. In contrast, the three MLP models consistently had small sizes, ranging between 26 kB and 30 kB. Finally, the RF approach generated models with the largest size, approximately four orders of magnitude greater than the other models, with each model requiring more than 100 MB.

Analyzing the size of the models trained with data from a limited part of the scenario, it is clear that this is not a problem in the current situation. But it is crucial to emphasize how drastically different the models are. This implies that more data (images) would be required if the entire field is used, increasing the models' size and becoming a potential problem.

Due to this indication, a simple experiment was conducted on a computer, where the same algorithms were trained using data from the entire field (dataset C). The results showed that the MLP models remained small, with each model occupying no more than 1 MB. In contrast, each RF model required between 3 and 5 GB, representing a significant difference. The computational power required to store and execute models of this size is greater than the capacity of the Raspberry Pi. Finally, the SVM models could not be

trained due to the time required. For instance, training the SVM model for the x target would have taken more than 72 hours.

The results of this section indicate that MLP (neural networks) are the best option for embedding an ML model for use in the localization system considering a tradeoff between required memory, computing power, and energy consumption, even the other approaches, such as RF appears to have more promising results.

4.2 Results and Discussion Part 2: Quality of ML models

The work's second part involves presenting a solution to the localization problem by utilizing the approaches discussed in Section 3.2.1. To achieve this, multiple algorithms were examined, considering this methodology. The initial assessment was carried out on dataset A, and the most effective algorithm was identified. Subsequently, this algorithm was trained and tested on datasets B1, B2, B3, and B4 to evaluate the trade-off between the grid's resolution and the estimates' accuracy. Next, the proposed approach in Section 3.2.2 was compared to both analytical (presented in [3]) and ML methods. This part is completed and more detailed in [4]. These results are the outcomes from Section 3.4.2 and are presented in Section 4.2.1.

Finally, a CNN based on VGG16 transfer-learning was trained with the images from Dataset A, and after training with images from Dataset B1, B2, B3, and B4, the results were analyzed. These results are the outcomes from Section 3.4.2 and are presented in Section 4.2.2.

It is worth emphasizing that, for concept solutions 1 and 2, each dataset was split into two parts: 85% for training and 15% for testing purposes. Additionally, all hyperparameters used in the algorithms were kept unchanged, meaning the standard values from Scikit-learn (version 1.1.2) were utilized. On the other hand, the dataset division to solution 3 was 80% for training, 10% for testing, and 10% for validation. Only the

hyperparameters in the dense layers were trained, with the other values maintained as the original values.

4.2.1 Results and Discussions of Approach 1: ML techniques using fiducial markers

Table 4.2 presents a comparison of the ML techniques, considering the complete scenario (using dataset A and the concept solution 1). The first technique listed is the baseline, followed by the ML techniques. The MAE and RMSE sections present the respective errors, with x and y values in meters and θ in degrees. The R^2 column displays the coefficient of determination for each target value in each approach. The final two sections indicate the average time required to train the models and the response time for each image.

Table 4.2: Results of the proposed techniques. Source: [4].

Technique		**Baseline**	**GB**	**KNR**	**MLP**	**RF**
MAE	**x [m]**	0.336	0.099	0.025	0.015	**0.007**
	y [m]	0.200	0.084	0.024	0.014	**0.006**
	θ [°]	92.41	26.92	6.55	9.34	**3.05**
RMSE	**x [m]**	0.418	0.133	0.041	0.022	**0.014**
	y [m]	0.226	0.109	0.041	0.021	**0.015**
	θ [°]	105.99	37.85	19.40	21.80	**11.24**
NRMSE	**x**	0.303	0.096	0.030	0.016	**0.010**
	y	0.310	0.149	0.056	0.030	**0.019**
	θ	0.294	0.105	0.054	0.061	**0.031**
R^2	**x**	0.00	0.90	0.99	0.99	**0.99**
	y	0.00	0.77	0.97	0.99	**0.99**
	θ	0.00	0.87	0.97	0.96	**0.99**
Training time [s]		-	3302	**28**	1812	15433
Response time avg [ms]		-	**0.08**	4.16	0.12	0.46

The comparison between the techniques is presented graphically in Figure 4.2, where the left plot illustrates the MAE error for axes x and y in meters, and the right plot shows the MAE error for θ in degrees.

Table 4.3 compares the quality of the estimations considering different grid resolutions

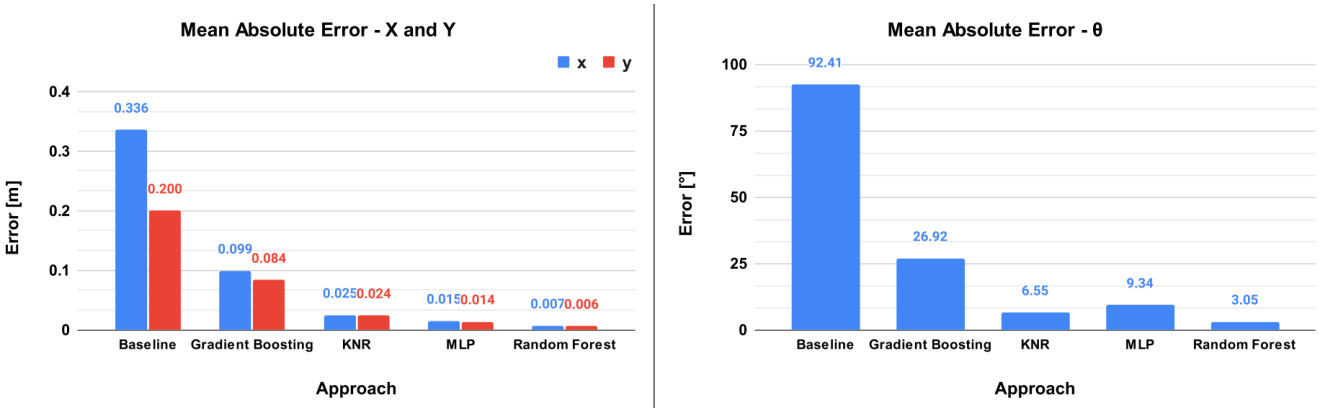

Figure 4.2: Image on the left displays the MAE for the x and y axes in meters, and the image on the right displays the MAE for the θ in degrees. Source: [4].

during the data collection. The datasets used were B1, B2, B3, and B4, with the best ML technique found, which was RF, as identified in the previous results (again, considering the concept solution 1). The table presents the number of collected images and quality metrics.

Table 4.3: Results using different grid resolutions. The technique used to obtain these results is the Random Forest Regressor. Source: [4].

Grid's resolution		**10 mm**	**5 mm**	**2.5 mm**	**1 mm**
Quantity of images		8306	33281	113596	655130
MAE	**x [m]**	0.005	0.003	0.002	0.001
	y [m]	0.005	0.003	0.002	0.001
	θ [°]	2.04	2.35	2.01	2.21
RMSE	**x [m]**	0.008	0.005	0.003	0.002
	y [m]	0.007	0.005	0.003	0.002
	θ [°]	8.98	12.46	10.35	10.60
NRMSE	**x**	0,073	0,045	0,030	0,018
	y	0,064	0,043	0,027	0,018
	θ	0,025	0,035	0,029	0,029
R^2	**x**	0.95	0.97	0.99	0.99
	y	0.95	0.98	0.99	0.99
	θ	0.99	0.99	0.99	0.99

The behavior of the MAE with the increases in the grid's resolution is presented graphically in Figure 4.3. The MAE for the x and y axes are the same and are in centimeters, presented by the left curve on the picture. The MAE for θ is in degrees,

shown by the curve on the right.

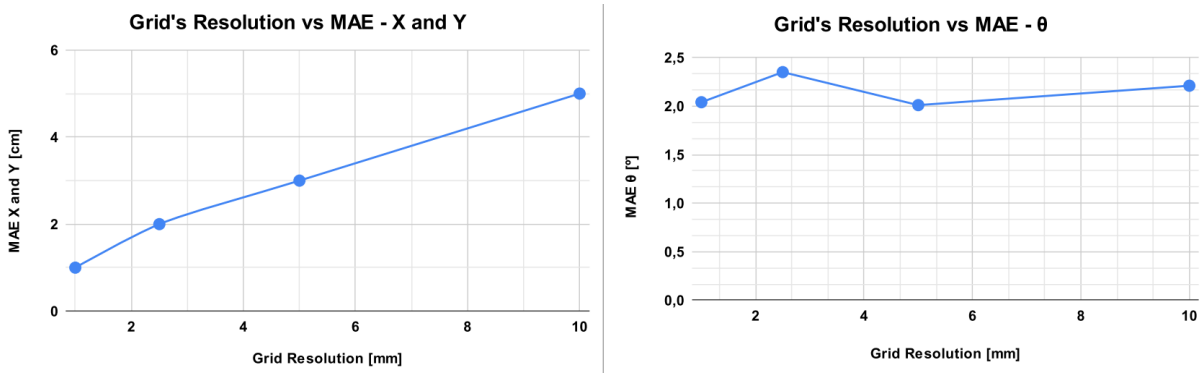

Figure 4.3: Comparison of the error obtained against the decrease of the grid's resolution. The graph on the left displays the MAE for the x and y axes in centimeters (the same curve is for both values), while the image on the right displays the MAE for the θ in degrees. Source: [4].

The last result in this part compares the ML and analytical approaches, presented in Table 4.4. This comparison was made considering the training with dataset C and the test with dataset E (considering here concept solution 2). To prevent division by 0 in the percentage relative error function, it was excluded predictions where the ground truth value was 0. This was necessary to calculate the relative percentage error accurately.

Table 4.4: Comparison of the two approaches: analytical and ML (Random Forest). Source: [4].

Approach		**Analytical**	**ML (Random Forest)**
MAE	**x [m]**	0.028	**0.022**
	y [m]	0.031	**0.023**
	θ [°]	**6.00**	7.27
RMSE	**x [m]**	**0.034**	0.037
	y [m]	**0.037**	0.043
	θ [°]	**8.89**	14.91
NRMSE	**x**	**0.023**	0.025
	y	**0.044**	0.051
	θ	**0.025**	0.041
Avg percentage Relative error	**x**	29.6	**27.6**
	y	35.9	**22.2**
	θ	**17.2**	17.3

Analyzing Table 4.2, it is possible to notice that the baseline has the highest error,

as expected, since it is the simplest prediction model available. However, all ML models produced errors lower than 10 centimeters. The RF approach had the best performance, with an error of only a millimeter in position (x and y) and 3.05 degrees in orientation. For all methods, RMSE was more significant than the MAE, indicating that the errors in the estimations had varying magnitudes, with some predictions exhibiting minor errors while others had more significant errors. Considering the NRMSE, it was possible to compare the errors in the x and y dimensions with the error in θ and then observe that they were of the same order of magnitude. The coefficient of determination (R^2) for all metrics was close to its maximum value, indicating that the estimations were accurate.

Another noteworthy aspect was the training time, which varied considerably across the models. The RF method required training times between 1 and 4 orders of magnitude higher than the other approaches. However, more crucial than training time is the response time, which was satisfactory across all models. The times reported in Table 4.2 did not include preprocessing time, which was approximately 13 milliseconds for each image.

Upon analyzing the results presented in Table 4.3, a correlation between the grid resolution and the estimation error in position can be observed. As the resolution of the grid increases, so does the error in the position estimation. This fact presents an exciting trade-off between improving the accuracy at the cost of acquiring more images. It is essential to highlight that these results were obtained considering a limited part of the field and may not represent the entire scenario. Figure 4.3 shows the grid resolution and error relationship. However, this correlation does not apply to the orientation aspect of the data collection, as the data is collected uniformly at a rate of 360 degrees for all grid resolutions.

So, improving the grid's resolution should be analyzed in each case since only the position error is improved. In systems that use odometry (as the robot used in RaF), the error in orientation is more important than the position error. So, due to this, in some cases, the improvement of the grid's resolution will not have a significant impact on the system due to the necessary extra computational effort, and the impact in the models,

for example, in the RF, the size of the model tends to increase.

Finally, the comparison presented in Table 4.4 reveals a similarity between the results obtained by the analytical and ML techniques. While the ML techniques outperform the analytical approach regarding MAE error in x and y, the orientation error is slightly higher. On the other hand, the analytical approach presents better RMSE. Overall, the ML techniques yield similar results to the analytical approach. It is important to note that only ArUcos placed on the floor were used, as in [3]. Additionally, it should be emphasized that these results are not the final prediction but inputs for a sensor fusion filter that will enhance the quality of estimations.

4.2.2 Results and Discussions Approach 2: CNN technique

Table 4.5 presents the results obtained with data from the whole environment, using 1 cm as a grid's resolution. Each column in the table presents a metric for each one of the three components of the pose. Each value averages the results collected in the three executions and their standard deviation. More details are available in [8].

Table 4.5: Results of the CNN approach obtained considering the whole field. Source: [8].

	MAE			**RMSE**			**NRMSE**			**R2**		
	x[m]	**y[m]**	θ[°]	**x[m]**	**y[m]**	θ[°]	**x**	**y**	θ	**x**	**y**	θ
Avg	0.0100	0.0070	6.05	0.0211	0.0142	11.79	0.015	0.019	0.033	0.997	0.996	0.985
Std Dev	0.0029	0.0017	2.72	0.0091	0.0028	5.05	0.007	0.004	0.014	0.002	0.002	0.012

The average training time for the position (x, y) was 10,047.15 seconds (standard deviation of 3,412.12 seconds), while the orientation (θ) required 5,078.67 seconds (standard deviation of 1,334.86 seconds). During the inference phase, the estimation time for position coordinates was 1.9 milliseconds (standard deviation of 0.7 ms), and for θ, it was also 1.9 milliseconds (standard deviation of 0.5 ms). Thus, the total time to estimate the complete pose was approximately 3.8 milliseconds, excluding the preprocessing time for the image.

Table 4.6 presents the results for the tests performed in a limited part of the environment with different grid resolutions. The columns represent the resolutions and the lines

Table 4.6: Results obtained considering the limited part of the field, using different grid's resolution, with *Avg.* columns indicating the average and *Std. Dev.* indicating the standard deviation. Source: [8].

Grid's Resolution		**10 mm**		**5 mm**		**2.5 mm**		**1 mm**	
Quantity of images		8306		33,291		113,596		655,130	
		Avg	**Std. Dev.**	**Avg**	**Std. Dev.**	**Avg**	**Std. Dev.**	**Avg**	**Std. Dev.**
MAE	**x[m]**	0.0026	0.0001	0.0026	0.0004	0.0023	0.0002	0.0023	0.0006
	y[m]	0.0026	0.0000	0.0027	0.0004	0.0023	0.0004	0.0023	0.0007
	θ[°]	2.97	0.29	2.76	0.14	1.58	0.10	4.51	3.82
RMSE	**x[m]**	0.0034	0.0002	0.0033	0.0005	0.0029	0.0002	0.0029	0.0006
	y[m]	0.0038	0.0006	0.0042	0.0002	0.0032	0.0003	0.0030	0.0007
	θ	6.03	1.50	7.09	0.76	4.06	0.71	6.99	3.80
NRMSE	**x**	0.0307	0.0015	0.0302	0.0043	0.0293	0.0026	0.0262	0.0059
	y	0.0345	0.0053	0.0368	0.0021	0.0292	0.0031	0.0279	0.0062
	θ[°]	0.02	0.00	0.02	0.00	0.01	0.00	0.02	0.01
R2	**x**	0.990	0.001	0.990	0.003	0.990	0.002	0.989	0.005
	y	0.985	0.005	0.984	0.002	0.990	0.002	0.990	0.004
	θ	0.996	0.002	0.995	0.001	0.998	0.001	0.994	0.006
Training time (Position) [s]		678.63	80.85	898.40	153.75	4505.48	647.51	8281.08	233.89
Training time (Orientation) [s]		592.89	80.17	1437.00	401.92	17369.99	3399.39	18509.41	8368.63
Inference time (Position) [ms]		2.95	0.74	2.02	0.07	2.44	0.35	2.06	0.15
Inference time (Orientation) [ms]		2.46	0.12	2.04	0.05	2.22	0.19	1.96	0.40

of the metrics for each pose component. All the values (except for the training time) are averages and the respective standard deviation of the three executions.

An interesting parallel can be drawn between the results presented in Table 4.5 and Table 4.2. The best ML technique found previously was the RF, and it is possible to notice the similarity with the results obtained with the CNN approach. In the measurements, there was a slight disparity of 3 mm and 1 mm in the values of x and y, respectively, while the orientation difference θ amounted to 3.0 degrees. Moreover, both the RMSE and the NRMSE exhibited similarity between the CNN and the RF. Additionally, all the outcomes displayed low standard deviation across the three iterations, reflecting the reliability and satisfactory performance of the models in the cross-validation process.

The models' consistent size across all employed picture quantities is another remarkable outcome of this part of the study. Each model is 512 MB, up to 1024 MB for the entire posture estimation. This is important because specific ML techniques, like Random Forest, can significantly increase the size, quote [7], since the model's structure can change, i.e., the depth of the trees, depending on the problem.

Table 4.6's analysis of the results reveals that the errors were essentially constant across all resolutions. This behavior differs from that found in Table 4.3, where the position estimation quality improved with the grid's resolution increase. As a result, it shows that the trade-off discussed in Section 4.2.1 does not apply to the CNN approach. However, for the resolutions of 10 mm, 5 mm, and 2.5 mm, the CNN results were better or on comparable levels with those shown by RF, and only for 1 mm were the CNN results worst.

CNN showed to be an exciting approach that presents the necessary stable size in memory, a low response time (in order of milliseconds), and high precision (in order of millimeters). The other side of this method is the generalization from the simulator to the real scenario is impossible since the CNN is based on the whole image and not on the tags' pose. So, to execute this in a real environment, it is necessary to collect the images in the scenario that the system will be used, and some cares are necessary, such as the luminosity of the environment.

4.3 Results and Discussions Part 3: Implementation in the Real Scenario

This part of the work involves the application of the model in the real robot. The objective is to evaluate the quality of the model by comparing it with the analytical approach and the ground truth and check the difference in the results obtained in the simulator and the real robot. To do this, as explained before in Section 3.4.3, the simulator had to be adjusted to be similar to the real robot, and all the necessary parameters (such as position, orientation, and field of view of the camera) were collected and applied in the simulator and were adjusted empirically until the image of the camera in the simulator looks like the image in the camera in the real robot. Next, the data were recollected, and the models retrained.

Two MLP approaches were used in the first test, which considers the concept solution

2. First, an original model considers all the default hyper-parameters from the Scikit-learn library, and another, an optimized model, considers the optimization of the hyper-parameters, as previously presented in Section 3.4.3. Table 4.7 presents the first result from the simulation, considering the training data as the whole environment and an arbitrary route to test.

Table 4.7: Results obtained in the simulator with two different MLP architectures, considering the concept solution 2.

	MLP Original			**MLP Optimized**		
	x[m]	**y[m]**	θ[°]	**x[m]**	**y[m]**	θ[°]
MAE	0.110	0.041	5.49	0.046	0.033	5.06
RMSE	0.165	0.063	8.07	0.074	0.044	6.90

Table 4.8 presents the results obtained in the execution in the real scenario, considering the route presented in Figure 3.15. In the first part of the test, the ground truth was not precise because only one of the tags positioned over the robot was available, making the ground-truth system return a wrong pose. So, these values were not considered, and only after the two tags were completely visible were the values considered. The results were from 189 pose estimations.

Table 4.8: Results of the errors obtained in the real scenario, comparing with the data collected in the ground-truth, considering the concept solution 2.

	MLP Original			**MLP Optimized**			**Analytical**		
	x[m]	**y[m]**	θ[°]	**x[m]**	**y[m]**	θ[°]	**x[m]**	**y[m]**	θ[°]
MAE	0.166	0.054	17.94	0.078	0.040	16.16	0.037	0.031	8.36
RMSE	0.184	0.064	34.45	0.104	0.057	16.69	0.060	0.044	20.62

Table 4.9 presents the MAE and RMSE between the MLP and analytical models' estimations. A video containing all of the executions of the analytical approach and the MLP approaches (considering the concept solutions 1 and 2) was recorded[1].

Comparing the results of the original MLP technique with the optimized one, there is a significant improvement, considering, especially in the x axis, improving from an MAE of 11 cm to 4.6 cm (representing an improvement of almost 42%), as presented in Table 4.7.

[1]Video of the comparison: `https://www.youtube.com/watch?v=-5ZmVpJobg0`.

Table 4.9: Results of differences obtained in the real scenario, considering the estimations with the MLP with the analytical method, considering the concept solution 2.

	MLP Original			**MLP Optimized**		
	x[m]	**y[m]**	θ[°]	**x[m]**	**y[m]**	θ[°]
MAE	0.145	0.0611	10.05	0.057	0.039	11.55
RMSE	0.166	0.066	13.04	0.078	0.048	14.77

Another interesting result is the comparison between Table 4.7 and Table 4.8. It is possible to see that the results presented a difference, for instance, of 3.2 cm in x, 7 mm in y, and around 11° in θ. Even with the differences, it is possible to notice the model's behavior in the simulation was amplified in the real world (i.e., all the errors obtained by the models in the simulator were smaller than that obtained in the real scenario by the same models), and slight differences, such as the parameters of the camera, can be influenced drastically by the results.

Table 4.8 presents the difference between the estimations of the MLP models and the analytical with the ground-truth value. It is possible to see a vast difference between the best MLP (optimized) and analytical, especially in x and θ, with MLP almost achieving double the values estimated by analytical. Observing this behavior during the tests was possible, as in the previously cited video. When comparing the executions using the analytical approach with the MLP-optimized models, the analytical approach was better. However, considering the test with the MLP-optimized, the robot presented a better precision than with the non-optimized mode, in y than the x (where the robot thought it was necessary to fix the position and did not move forward straight), and in θ, since the moves the robot used to be not well aligned.

Analyzing Table 4.9 can notice a big difference between the analytical and the MLP models. This fact reinforces the difference in the quality of the models. Based on the three tables, it is possible to see that the error in the y was the lower error, followed by θ, while the error in x was the worst (comparing the range of the possible values).

So, due to this, it is possible to see that this proposed model is not ready to be implemented entirely in the robot and be used in the competition. However, the proposal

was promising since, even with all empirical adjustments in the simulator to represent the real scenario, the robot work is reasonably plausible considering the y axis and the θ, even with the behavior in x being the worst. On the other side, it emphasizes that the small divergences between the simulator and the real will result in high differences.

The second test in the real robot used concept solution 1, where each image is processed, and a matrix with all the relative poses of the identified ArUcos is generated. This data is used in the ML models. The same as the previous tests, two types of MLP models were used: a model with all the default hyper-parameters of the scikit-learn library and another with an optimization of the parameters. The considered optimization was the same as previously presented. The test considered 1236 estimations, and the video of the executions was recorded[2].

Table 4.10: Results of the errors obtained in the real scenario, showing the error in the estimations with the MLP models based on concept solution 1.

	MLP Original			**MLP Optimized**		
	x[m]	**y[m]**	θ[°]	**x[m]**	**y[m]**	θ[°]
MAE	0.047	0.060	10.73	0.089	0.056	37.44
RMSE	0.077	0.082	17.68	0.091	0.062	38.79

The left part of Table 4.10 presents the results of Solution 1 without optimization. It is possible to see the MAE was on a centimeter scale, with 4.7 cm in x and 6.00 cm in y. The error in orientation was 10.7 degrees. This error is considerably better when comparing the models from concept solution 2 but worse than the analytical approach. However, the tests performed in the real scenario demonstrated the possibility of using this model in the real competition because the robot can collect and drop the box in the right places. The robot demonstrated no stability all the time, which can be observed in Table 4.10, in the RMSE considerably higher than MAE.

The right part of Table 4.10 presents the results of concept solution 1 but with optimized MLP models. The MAE results were considerably higher than those presented in the models without optimization. The results have shown an increase of MAE in 88.7% in

[2]Video of the comparison: `https://www.youtube.com/watch?v=-5ZmVpJobg0`.

x, -6.3% in y, and 248.9% in θ. These results show a considerable decrease of estimations quality in x and θ but a small increase in y. This can be explained due to the difference in the approaches, and the optimization for one approach can not be good for others. Due to this, the robot was not able to localize with precision during the test and was not able to be used in the robot competition. However, the interesting result in these results was the small difference between MAE and RMSE, which presented a difference of 2.7 mm in x, 5.1 mm in y, and 1.35º in θ. This fact represents a low variance in the error, indicating that this error can be systematic and adjusted manually to obtain good estimations. Due to this, even with the initial not being perfect result, this model has also shown to be promising.

Chapter 5

Conclusions

This work aimed to provide solutions to the localization problem in the context of the RobotAtFactory 4.0 competition using machine learning. Some solutions are already developed for the general problem of indoor localization (such as presented in Section 2.1). In addition, a specific approach to localization in RaF was developed in [3], using analytical geometry and the known pose of all tags. Three concept solutions were developed in this work: one that estimates one pose per image frame using the fiducial markers, another that estimates one pose per ArUco identified in an image frame, and another that estimates one pose per image without using ArUcos. All the proposals use machine learning approaches.

The first research question of this work (Can machine learning approaches effectively estimate the position and orientation of a robot solely based on camera images in a structured environment with the presence of fiducial markers?) that can be answered based on the results, with a yes, the ML approaches, even without tuning can be an exciting solution for localization, resulting in errors in millimeter-scale in position, and less than 10^{o} degree in orientation. In addition, by tuning in the models and adapting the simulator to the real scenario, it is possible to train the model only with data in the simulator and apply it to the real robot.

The second research question of this work (Among different types of machine learning approaches, such as MLP, Random Forest (RF), Gradient Boosting (GB), KNR, and

CNN, which approach leads to the smallest pose error in estimating the robot's position and orientation based solely on camera images in a structured environment with fiducial markers?), can be answered, based on the results, that RF and CNN were the approaches that presented the best results, presenting errors in millimeter scale, and response time of milliseconds.

The third research question of this work (Considering the constraints of embedded devices with limited memory and computing power, which type of machine learning approach is better suited for achieving accurate pose estimation in a structured environment with fiducial markers?) can be answered, based on the results, that the MLP and CNN are the best approaches. This is possible due to the controlled size of the resulting models and their response time in milliseconds.

So, due to this, it is possible to conclude that the work was successful since all the research questions were solved. The ML approaches were validated in terms of feasibility [7], and many possible machine-learning techniques were explored [4][8]. The models were also applied in the real robot, with concept solution 2 does not present good results but indicating a promising way to be explored; Concept solution 1, on the other hand, showed interesting results and could be used in the competition, even not presenting strong stability in the estimations, but indicating an interesting approach to be explored.

In addition, it is essential to emphasize that the ML concept solutions suggested in this study offer a notable advantage over the analytical approach, which relies on the precise knowledge of the ArUco marker locations for robot localization. This advantage is particularly relevant when obtaining accurate marker poses is difficult or unfeasible, such as in hostile environments. The proposed ML approaches do not depend on explicit marker information, eliminating the need for such precision [4]. Another interesting result is the possibility of completing training and preparing the models based on simulation and applying it in real scenarios, which was validated in the RaF competition scenario. Still, it can be extended to other scenarios with a similar structure.

Future works can explore more about the ML models, studying other methods and trying to improve the estimations by tuning the hyperparameters. In addition, it can

explore more the application of the model trained in simulation in the real world, trying to improve the quality and the adaptation from one to another, focusing on the robustness of the models. Another possible future work is to adaptable the other approaches, such as the CNN (considering the concept solution 3) and other techniques (such as concept solutions 1 and 2), tested in simulation in a whole localization system, and test and evaluates them in a real scenario.

Bibliography

[1] S. Huang and G. Dissanayake, "Robot localization: An introduction," in *Wiley Encyclopedia of Electrical and Electronics Engineering*. Hoboken, NJ, USA: John Wiley Sons, Ltd, 2016, pp. 1–10, ISBN: 9780471346081. DOI: https://doi.org/10.1002/047134608X.W8318. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/047134608X.W8318. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/047134608X.W8318.

[2] A. Nessa, B. Adhikari, F. Hussain, and X. N. Fernando, "A survey of machine learning for indoor positioning," *IEEE Access*, vol. 8, pp. 214 945–214 965, 2020. DOI: 10.1109/ACCESS.2020.3039271.

[3] J. Braun, A. Oliveira Júnior, G. Berger, *et al.*, "A robot localization proposal for the robotatfactory 4.0: A novel robotics competition within the industry 4.0 concept," *Frontiers in Robotics and AI*, vol. 9, Nov. 2022. DOI: 10.3389/frobt.2022.1023590.

[4] L. C. Klein, J. Braun, J. Mendes, *et al.*, "A machine learning approach to robot localization using fiducial markers in robotatfactory 4.0 competition," *Sensors*, vol. 23, no. 6, 2023, ISSN: 1424-8220. DOI: 10.3390/s23063128. [Online]. Available: https://www.mdpi.com/1424-8220/23/6/3128.

[5] J. Braun, A. O. Júnior, G. S. Berger, J. Lima, A. I. Pereira, and P. Costa, "Robotatfactory 4.0: A ros framework for the simtwo simulator," in *2022 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, 2022, pp. 205–210. DOI: 10.1109/ICARSC55462.2022.9784794.

[6] P. Costa, J. Gonçalves, J. Lima, and P. Malheiros, "Simtwo realistic simulator: A tool for the development and validation of robot software," *Theory and Applications of Mathematics Computer Science*, vol. 1, pp. 17–33, Apr. 2011.

[7] L. C. Klein, J. Braun, F. N. Martins, *et al.*, "Using machine learning approaches to localization in an embedded system on robotatfactory 4.0 competition: A case study," in *2023 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, 2023, pp. 69–74. DOI: 10.1109/ICARSC58346.2023.10129619.

[8] L. C. Klein, J. Mendes, J. Braun, *et al.*, "Deep learning-based localization approach for autonomous robots in the robotatfactory 4.0 competition," Manuscript submitted for publication, 2023.

[9] C. Stachniss. "Robot localization - an overview," Youtube. (2021), [Online]. Available: https://www.youtube.com/watch?v=8VJ-A9OlhAE&ab_channel=CyrillStachniss.

[10] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: Part i," *IEEE robotics & automation magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[11] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, and W. Burgard, *Principles of robot motion: theory, algorithms, and implementations*. MIT press, 2005.

[12] M. S. Grewal, L. R. Weill, and A. P. Andrews, *Global positioning systems, inertial navigation, and integration*. Hoboken, NJ, USA: John Wiley & Sons, 2007, ISBN: 9780470041901.

[13] D. Fox, "Markov localization-a probabilistic framework for mobile robot localization and navigation.," Ph.D. dissertation, Universität Bonn, 1998.

[14] D. Fox, W. Burgard, and S. Thrun, "Markov localization for mobile robots in dynamic environments," *Journal of artificial intelligence research*, vol. 11, pp. 391–427, 1999.

[15] R. Siegwart, M. Chli, and N. Lawrance, *Autonomous mobile robots MOOC*, https://www.edx.org/course/autonomous-mobile-robots-ethx-amrx-1, 2014.

[16] N. J. Gordon, D. J. Salmond, and A. F. Smith, "Novel approach to nonlinear/non-gaussian bayesian state estimation," in *IEE proceedings F (radar and signal processing)*, IET, vol. 140, 1993, pp. 107–113.

[17] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking," *IEEE Transactions on signal processing*, vol. 50, no. 2, pp. 174–188, 2002.

[18] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte carlo localization: Efficient position estimation for mobile robots," *AAAI/IAAI*, vol. 1999, no. 343-349, pp. 2–2, 1999.

[19] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," in *Proceedings 1999 IEEE international conference on robotics and automation (Cat. No. 99CH36288C)*, IEEE, vol. 2, 1999, pp. 1322–1328.

[20] I. Rekleitis, G. Dudek, and E. Milios, "Probabilistic cooperative localization and mapping in practice," in *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, IEEE, vol. 2, 2003, pp. 1907–1912.

[21] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, Mar. 1960, ISSN: 0021-9223. DOI: 10.1115/1.3662552. eprint: https://asmedigitalcollection.asme.org/fluidsengineering/article-pdf/82/1/35/5518977/35_1.pdf. [Online]. Available: https://doi.org/10.1115/1.3662552.

[22] G. Welch and G. Bishop, "An introduction to the kalman filter," Chapel Hill, NC 27599-3175, Tech. Rep., Apr. 2004.

[23] P. S. Maybeck, "The kalman filter: An introduction to concepts," in *Autonomous robot vehicles*, Springer, 1990, pp. 194–204.

[24] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall, 2010.

[25] A. Becker, *Online kalman filter tutorial*, 2022. [Online]. Available: `https://www.kalmanfilter.net/background.html`.

[26] Y. Huang, J. Benesty, G. W. Elko, and R. M. Mersereati, "Real-time passive source localization: A practical linear-correction least-squares approach," *IEEE transactions on Speech and Audio Processing*, vol. 9, no. 8, pp. 943–956, 2001.

[27] Q. Zhang, "Some implementation aspects of sliding window least squares algorithms," *IFAC Proceedings Volumes*, vol. 33, no. 15, pp. 763–768, 2000.

[28] D. Wilbers, C. Merfels, and C. Stachniss, "Localization with Sliding Window Factor Graphs on Third-Party Maps for Automated Driving," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2019.

[29] M. Lauer, S. Lange, and M. Riedmiller, "Calculating the perfect match: An efficient and accurate approach for robot self-localization," in *RoboCup 2005: Robot Soccer World Cup IX*, A. Bredenfeld, A. Jacoff, I. Noda, and Y. Takahashi, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 142–153, ISBN: 978-3-540-35438-3.

[30] H. Sobreira, C. Costa, I. Sousa, *et al.*, "Map-matching algorithms for robot self-localization: A comparison between perfect match, iterative closest point and normal distributions transform," *Journal of Intelligent Robotic Systems*, vol. 93, Mar. 2019. DOI: `10.1007/s10846-017-0765-5`.

[31] P. J. Best, "A method for registration of 3-d shapes," *IEEE Trans Pattern Anal Mach Vision*, vol. 14, pp. 239–256, 1992.

[32] S. Rusinkiewicz and M. Levoy, "Efficient variants of the icp algorithm," in *Proceedings third international conference on 3-D digital imaging and modeling*, IEEE, 2001, pp. 145–152.

[33] P. Biber and W. Straßer, "The normal distributions transform: A new approach to laser scan matching," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, IEEE, vol. 3, 2003, pp. 2743–2748.

[34] M. Magnusson, A. Nuchter, C. Lorken, A. J. Lilienthal, and J. Hertzberg, "Evaluation of 3d registration reliability and speed-a comparison of icp and ndt," in *2009 IEEE International Conference on Robotics and Automation*, IEEE, 2009, pp. 3907–3912.

[35] M. Magnusson, "The three-dimensional normal-distributions transform: An efficient representation for registration, surface analysis, and loop detection," Ph.D. dissertation, Örebro universitet, 2009.

[36] M. Kalaitzakis, B. Cain, S. Carroll, A. Ambrosi, C. Whitehead, and N. Vitzilaios, "Fiducial markers for pose estimation," *Journal of Intelligent & Robotic Systems*, vol. 101, no. 4, pp. 1–26, 2021. DOI: 10.1007/s10846-020-01307-9.

[37] M. Kalaitzakis, S. Carroll, A. Ambrosi, C. Whitehead, and N. Vitzilaios, "Experimental comparison of fiducial markers for pose estimation," in *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2020, pp. 781–789. DOI: 10.1109/ICUAS48674.2020.9213977.

[38] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas, and M. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, vol. 47, no. 6, pp. 2280–2292, 2014, ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2014.01.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320314000235.

[39] S. Sadeghi Esfahlani, A. Sanaei, M. Ghorabian, and H. Shirvani, "The deep convolutional neural network role in the autonomous navigation of mobile robots (srobo)," *Remote Sensing*, vol. 14, no. 14, p. 3324, 2022.

[40] A. Atanasyan and J. Roßmann, "Improving self-localization using cnn-based monocular landmark detection and distance estimation in virtual testbeds," in *Tagungsband des 4. Kongresses Montage Handhabung Industrieroboter*, Springer, 2019, pp. 249–258.

[41] A. Kendall, M. Grimes, and R. Cipolla, "Posenet: A convolutional network for real-time 6-dof camera relocalization," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2938–2946.

[42] C. Szegedy, W. Liu, Y. Jia, *et al.*, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9. DOI: 10.1109/CVPR.2015.7298594.

[43] S. Ogiso, T. Kawagishi, K. Mizutani, N. Wakatsuki, and K. Zempo, "Self-localization method for mobile robot using acoustic beacons," *ROBOMECH Journal*, vol. 2, no. 1, pp. 1–12, 2015.

[44] J. Wang and Y. Takahashi, "Indoor mobile robot self-localization based on a low-cost light system with a novel emitter arrangement," *ROBOMECH Journal*, vol. 5, no. 1, pp. 1–17, 2018.

[45] R. Yanase, D. Hirano, M. Aldibaja, K. Yoneda, and N. Suganuma, "Lidar-and radar-based robust vehicle localization with confidence estimation of matching results," *Sensors*, vol. 22, no. 9, p. 3545, 2022.

[46] T. Sattler, B. Leibe, and L. Kobbelt, "Fast image-based localization using direct 2d-to-3d matching," in *2011 International Conference on Computer Vision*, IEEE, 2011, pp. 667–674.

[47] M. A. Brubaker, A. Geiger, and R. Urtasun, "Map-based probabilistic visual self-localization," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 4, pp. 652–665, 2015.

[48] M. Avgeris, D. Spatharakis, N. Athanasopoulos, D. Dechouniotis, and S. Papavassiliou, "Single vision-based self-localization for autonomous robotic agents," in *2019 7th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, IEEE, 2019, pp. 123–129.

[49] N. F. L. Nebeker, "External robot localization in 6dof," M.S. thesis, Faculdade de Engenharia da Universidade do Porto (FEUP), Oct. 2015. [Online]. Available: https://repositorio-aberto.up.pt/handle/10216/80454.

[50] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[51] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014, ISBN: 1461471370.

[52] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[53] J. H. Friedman, "Greedy function approximation: A gradient boosting machine.," *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001. DOI: 10.1214/aos/1013203451. [Online]. Available: https://doi.org/10.1214/aos/1013203451.

[54] A. Natekin and A. Knoll, "Gradient boosting machines, a tutorial," *Frontiers in Neurorobotics*, vol. 7, 2013, ISSN: 1662-5218. DOI: 10.3389/fnbot.2013.00021. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnbot.2013.00021.

[55] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[56] S. B. Kotsiantis, "Decision trees: A recent overview," *Artificial Intelligence Review*, vol. 39, no. 4, pp. 261–283, 2013.

[57] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[58] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[59] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–137, 1943.

[60] F. Murtagh, "Multilayer perceptrons for classification and regression," *Neurocomputing*, vol. 2, no. 5, pp. 183–197, 1991, ISSN: 0925-2312. DOI: https://doi.org/10.1016/0925-2312(91)90023-5. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0925231291900235.

[61] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[62] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," Computational and Biological Learning Society, 2015, pp. 1–14.

[63] Y. Chen, R. Chen, M. Liu, A. Xiao, D. Wu, and S. Zhao, "Indoor visual positioning aided by cnn-based image retrieval: Training-free, 3d modeling-free," *Sensors*, vol. 18, no. 8, 2018, ISSN: 1424-8220. DOI: 10.3390/s18082692. [Online]. Available: https://www.mdpi.com/1424-8220/18/8/2692.

[64] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, IGI global, 2010, pp. 242–264.

[65] "Mean absolute error," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2010, pp. 652–652, ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_525. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_525.

[66] "Mean squared error," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2010, pp. 653–653, ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_528. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_528.

[67] D. Chicco, M. J. Warrens, and G. Jurman, "The coefficient of determination r-squared is more informative than smape, mae, mape, mse and rmse in regression analysis evaluation," *PeerJ Computer Science*, vol. 7, e623, 2021.

[68] M. T. Yazici, S. Basurra, and M. M. Gaber, "Edge machine learning: Enabling smart internet of things applications," *Big Data and Cognitive Computing*, vol. 2, no. 3, 2018, ISSN: 2504-2289. DOI: 10.3390/bdcc2030026. [Online]. Available: https://www.mdpi.com/2504-2289/2/3/26.

[69] "Robot modeling and control, second edition [bookshelf]," *IEEE Control Systems Magazine*, vol. 42, no. 1, pp. 126–128, 2022. DOI: 10.1109/MCS.2021.3122271.