Using Machine Learning Approaches to Localization in an Embedded System on RobotAtFactory 4.0 Competition: A Case Study

Luan C. Klein[*][†], João Braun[†][‡][§][¶], Felipe N. Martins [††], Heinrich Wörtche [††][‡‡], Andre Schneider de Oliveira[*],
João Mendes[†][‡][||], Vítor H. Pinto[§][||], Paulo Costa[§][¶], José Lima[†][‡][¶]
[*] Universidade Tecnológica Federal do Paraná, Campus Curitiba, UTFPR/PR, Brasil
[†] Research Centre in Digitalization and Intelligent Robotics (CeDRI),
Instituto Politécnico de Braganca, Campus de Santa Apolonia, Portugal, Email: jllima@ipb.pt
[‡] Laboratory for Sustainability and Technology in Mountain Regions (SusTEC),
Instituto Politécnico de Bragança, Bragança, Portugal
[§] Faculty of Engineering of University of Porto, Portugal
[¶] INESC Technology and Science, Porto, Portugal
[||] SYSTEC (DIGI2), ARISE & ECE Dept.,
Fac. de Engenharia, Universidade do Porto - Rua Dr. Roberto Frias, 4200-465 Porto, Portugal
[**] ALGORITMI Center, University of Minho, Braga, Portugal
[††] Sensors and Smart Systems Group, Institute of Engineering, Hanze University of Applied Sciences,
The Netherlands
[‡‡] Department of Electrical Engineering, Eindhoven University of Technology, The Netherlands

***Abstract*—The use of machine learning in embedded systems is an interesting topic, especially with the growth in popularity of the Internet of Things (IoT). The capacity of a system, such as a robot, to self-localize, is a fundamental skill for its navigation and decision-making processes. This work focuses on the feasibility of using machine learning in a Raspberry Pi 4 Model B, solving the localization problem using images and fiducial markers (ArUco markers) in the context of the RobotAtFactory 4.0 competition. The approaches were validated using a realistically simulated scenario. Three algorithms were tested, and all were shown to be a good solution for a limited amount of data. Results also show that when the amount of data grows, only Multi-Layer Perception (MLP) is feasible for the embedded application due to the required training time and the resulting size of the model.**

***Index Terms*—Indoor Localization; Machine Learning; RobotAtFactory 4.0; Robotics Competitions; Embedded systems**

I. Introduction

An efficient Autonomous Mobile Robot (AMR) has a combination of several features, and each one of them has a different role. One of the most essential competencies is the capacity to locate itself in the environment because it is a prerequisite for making decisions about the future [1]. In the robotics context, localization means the estimate of the pose (composed of the position and orientation) according to a reference frame. There are some approaches to solving the localization problem in general contexts, such as the Global Positioning System (GPS). But due to limitations, it may not be available in indoor environments [2]. In this context, several approaches have been developed, and the use of artificial intelligence (AI) techniques is an example of this approach [3].

As previously mentioned, the use of localization is one of the most fundamental principles in several contexts, such as in robotic competitions. One of them is the RobotAtFactory 4.0 (RaF), where an AMR has to move boxes through a model of a warehouse, and is allowed to communicate only with the competition server. The goal is to move as many boxes as possible in the shortest time. This competition simulates an automated factory, where some materials have to be passed through several processes in different places inside the environment. At this competition, the AMR can use whichever means is necessary to locate itself as long as it complies with the competition rules.

One of the current approaches to solve the localization problem in this competition is using the markers placed on the environment (ArUcos markers). Using the stored data about the ArUcos' pose and their relative position with the camera's reference frame, the approach uses a mix of computer vision, analytical geometry, and stochastic filters to estimate the robot's pose [4]. Furthermore, with different estimations and aggregating these with a filter (such as an Extend Kalman Filter), the approach can obtain a good estimate of the robot's pose amongst noise and error. One issue with this approach is the dependency on the prior knowledge of the exact pose of the ArUcos because a small error in this data can lead to a huge estimate error.

Artificial Intelligence techniques can be applied to tackle the above-mentioned problem [5]. However, the implementation of AI, especially machine learning (ML), in embedded systems can be a challenge due to the limited computational resources normally available in embedded systems. An interesting work about the use of some ML techniques was done in [6], where some algorithms were tested on a Raspberry Pi 3 Model B and their performance was measured.

In this context, this work proposes a case study of the work done in [6], but focuses on the localization problem of RaF competition. The idea is to assess and validate the capacity for training and execution of ML techniques by a Raspberry Pi 4 Model B for the localization problem, considering response time, energy efficiency, and model size. Optimization of the algorithms is out of the scope of the present work. All data used in this work was obtained from a realistic simulator, based on a simulation of the real field of the competition [7], [8].

The remaining work is divided into four sections: Section II presents the related works; Section III describes how this work was developed; Section IV presents the results and the discussions; Finally, Section V presents the conclusion about the work.

II. Related Work

The localization problem is a common topic, which has been discussed for a long time. There are several approaches to solve this problem, and in outdoor scenarios, the most common approach is the Global Positioning System (GPS). But this approach fails indoors because of physical limitations [2], and in this way, other solutions are needed. Then, some approaches to solving indoor localization were developed.

One of these approaches is the Kalman Filter (KF), which is a mathematical approach that works by combining measurements of a system with predictions of how the system is expected to behave [9]. It works even if the measurements are noisy or incomplete. Kalman Filter is used in linear situations, however, most of the situations are not linear. In this way, some variants of the KF were developed to work in non-linear situations, such as the Extended Kalman Filter (EKF) [10].

Another interesting approach is the Markov Localization [11], which works as a probabilistic algorithm, which means the approach keeps a probability distribution over the space of all such hypotheses, and not only one hypothesis about the robot's localization. A further interesting approach is Monte Carlo Localization [12], which is a kind of Particle Filter, and it uses multiple samples (called particles) to represent a hypothesis of the interest variable, the robot's localization.

Other approaches for this problem are the map-matching algorithms, such as Perfect Match [13], Iterative Closest Point (ICP) [14], and Normal Distributions Transform (NDT) [15]. A comparison between these approaches was done in [16].

Furthermore, ML techniques were also used to try to solve this problem, using approaches like Random Forest (RF), Support Vector Machines (SVM), K-Nearest Neighbors (KNN), and Artificial Neural Networks (ANN) [3].

Another common technique used to help in localization is using landmarks in the environments, especially the fiducial markers [17]. There are several kinds of them, such as ARTag, AprilTag, ArUco, and STag [17]. They differ from each other in the way in which each of them is built. Figure 1 presents a comparison between these four types. On the RaF competition, the fiducial marker used is ArUco [4].

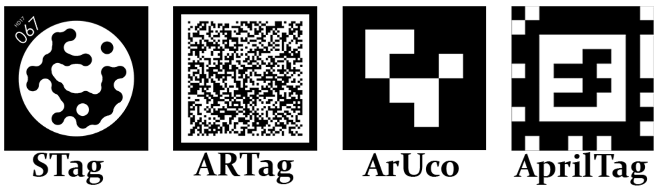

Fig. 1. Comparison between fiducial markers [1] [18].

The use of fiducial markers for localization is also used in the RaF competition. In the proposal done in [4], each ArUco identified in an image has the relative position with the camera calculated, and with 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. Furthermore, to aggregate all the estimations some stochastic filters are used, such as EKF and Mahalanobis Filter, aiming to improve the quality of the estimations.

The problem with such an analytical approach to the localization problem in this context is the necessity to know the exact pose of each ArUco, which can be a problem to obtain in hostile environments. In this way, a ML approach is proposed to solve the problem, where the previous knowledge of the ArUcos's pose is not relevant. This was explored in [5].

With the advancement of the Internet of Things (IoT), the use of AI in embedded systems is becoming a common approach. A study about the embedded ML was made in [19], exploring several platforms and ML libraries, and their efficiency.

The use of ML in a Raspberry PI 3 Model B was explored in [6], where the authors used three different approaches: RF, Multi-layer Perceptron (MLP), and SVM. These techniques were validated in regression and classification problems. The authors discussed the training and response time, the energy consumption, and a comparison between the approaches in terms of accuracy. In that, the authors conclude that all of the algorithms achieved a good accuracy (exceeded 80%), with inference time below one-millisecond and, according to the authors, moderately low energy consumption compared with other types of activities, such as browsing the web and watching videos. However, the authors didn't discuss the size of the models.

[1]STag: https://github.com/bbenligiray/stag, ARTag: https://shawnlehner.github.io/ARMaker/, ArUco: https://chev.me/arucogen/ and AprilTag: https://github.com/AprilRobotics/apriltag

III. Materials and Methods

A. Context Description

RobotAtFactory 4.0 is a competition in which autonomous robots move boxes from one place to another, similar to what happens in current factories. Robots that participate in this competition are subjected to several rules, such as being completely autonomous and cannot have any kind of communication with any external system that is not explicitly provided by the organization. Furthermore, the robot must fit into a cube of 30 × 30 × 30 cm. Some of the components of the robot are micro-controllers, an RGB camera (placed on the top front of the robot), motors, and other sensors and actuators [4].

Raspberry Pi is a single-board microcomputer available in several versions. In this work, version 4 model B was used, and Figure 2 presents a photograph of it. This version has 4 GB RAM, 40 pins, and Broadcom BCM2711, Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz[2]. The operating system used was the Raspberry PI OS[3] (previously called Raspbian), which is a 32-bits operational system, open-source software, based on Debian, and optimized for Raspberry Pi hardware.

Fig. 2. Raspberry 4 Pi Model B was used for the experiments.

A simulation scene was developed in the SimTwo simulator by the competition's organizer[4]. Thus, the simulator scenario follows the specifications of the official competition rules. The simulator has several features for the users, such as a code editor, which makes it possible to program a specific route by the robot. It works with rigid-body dynamics interactions and constraints [8]. Figure 3 presents the simulator that displays the virtual representation of the RaF competition field.

Furthermore, the robot in the simulator is based on a real robot and it has an onboard camera. Figure 4 presents an image captured by the robot's camera in the simulator. Each image collected by the simulator has an associated pose {x, y, θ}, where x and y are in meters and θ, which is the robot's

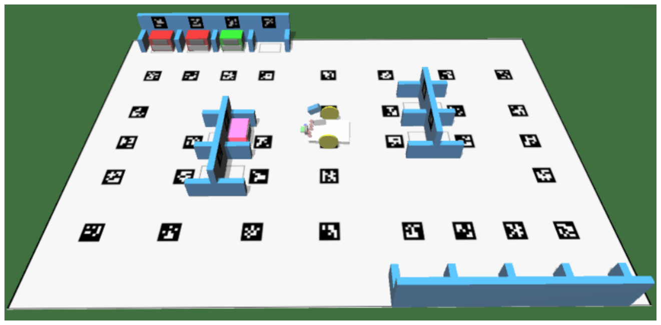

Fig. 3. Simulation scene that displays the robot and the competition's simulated environment.

orientation, in degrees. For instance, the pose {-0.008, 0.116, 125.65} is associated with Figure 4.

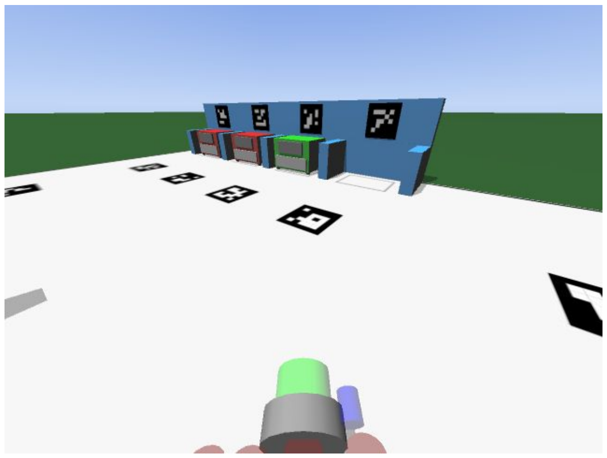

Fig. 4. Example of an image taken by the robot's camera in simulation.

B. Methodology

A sequence of four steps was followed to do the complete work. First, it was necessary to collect the data from the simulator, followed by data preprocessing, training models, and model evaluation. Figure 5 depicts the flowchart of the process, presenting the steps, the software used, and the place where it was executed (computer or Raspberry Pi). This process is similar to that done in [5], but here the process is executed in an embedded system, not in a computer. Besides this, it is important to highlight the Python version used was 3.9.2, the Pandas library was 1.5.3, the OpenCV library was 4.5.1, and the *scikit-learn* library was 1.2.1.

1) Data Collection: Since all images come from a simulated scenario, the simulator was executed in an external device to collect data and later, it was transferred to the Raspberry Pi. Two adjustments were made in the standard simulator settings, aiming to avoid problems that are not relevant to this work. The first modification was about the illumination, which was replaced to avoid dark parts on the field. The second was changing the color of the walls, changing from blue to white

[2]More details in: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/

[3]https://www.raspberrypi.com/software/

[4]Available at: https://github.com/P33a/SimTwo

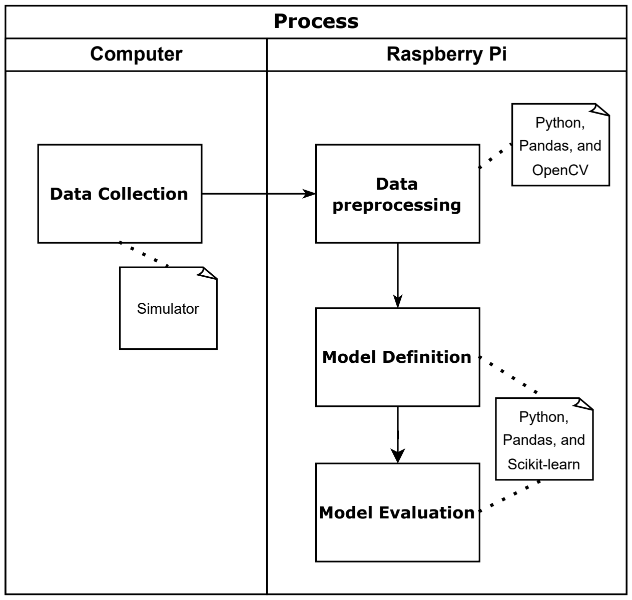

Fig. 5. Flowchart of the process. The first step is data collection, followed by data processing, model definition and training, and model evaluation.

(because the OpenCV library[5], used to identify the fiducial markers, works better when the edges are white).

To collect the data in the simulator, the field was discretized in a grid with a 1 cm resolution. Thus, the robot was placed in all possible positions in this grid, i.e., squares without obstacles. For each position, the robot performed a 360° turn and took around 60 pictures per loop, in order to create a database. This way, around 350 thousand images were collected, with each image having an associated pose. This is a large number of images, occupying about 7 GB of memory. Since the idea of this work is only to validate the feasibility of the ML in the localization problem at RaF competition, only images from a part of the field were used: a square with size 10x10 cm in the center of the field, totaling around 8 thousand images. This restriction aimed to reduce the number of images since one of the goals is training the models inside of the Raspberry Pi. Furthermore, the quantity of images depends on the grid's resolution and can vary drastically. Finally, the data collected was divided into 85% of the images for the training set and 15% for the test set.

2) Data Preprocessing: The preprocessing process consists of extracting the information about the ArUcos in each image and aggregating all these data to create the datasets that will be used for the ML models. To do this, the OpenCV library was used. Given an image, this library is able to recognize the present ArUcos and return the ID of the tag and arrays with the position and orientation, called *tvec* and *rvec* respectively. The *tvec* array is in the unit metric used in the camera's parameters (meters, centimeters, etc) and *rvec* is in axis-angle format. Each of these arrays contains 3 elements, where each one corresponds to one axis (*x, y, z*). So, to create the dataset, each tag identified (it can identify more than one tag per image) will be an observation in the dataset, with 7 features (id, *rvec* and *tvec*) and with 3 targets (*x, y* and θ, which are associated with the image that the ArUco was identified). This process was done for the two datasets, i.e., training and testing sets. Furthermore, it is important to highlight that the preprocessing does not necessarily have to be done in the Raspberry Pi, but could be executed in a more powerful computer. However, we chose to execute the training in the Raspberry Pi to investigate if the system could be trained and executed in the same hardware, which makes it independent. In addition, in [6] the authors realized the model training in the Raspberry Pi, which gave us an indication that the same procedure could be done for our case.

3) Model definition: Three algorithms were used, such as the work done in [6]: MLP, SVM, and Random Forest. The *scikit-learn* library was used to implement these models. The methods used were:

- For Random Forest: *RandomForestRegressor*[6]
- For MLP: *MLPRegressor*[7]
- For SVM: *SVR*[8]

Since the objective is to only validate the performance, no modification was made to the hyper-parameters, i.e., the standard values defined in the *scikit-learn* library were used. Besides this, since the target is composed of three independent values, one model was created for each target. This means, in total, 9 models were created: three Random Forest, one to x, one to y, and another to θ, and the same way to other algorithms. This division was made because once the three targets are uncorrelated between them, creating a specific model for each one can improve the quality of the predictions.

4) Model Evaluation: To evaluate the accuracy of the models, the metric used was the Mean Absolute Error (MAE) [20]. This metric is given by the following equation:

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

where y_i represents the true value, the $\hat{y}_i$ represents the predicted value for the instance i and n the number of predictions. The best value for MAE is 0, which indicates a perfect estimation. To evaluate the feasibility of the models, the metrics used were: training and execution time; quantity of energy used in Watt-hour (Wh), and the sizes of the models after training. To measure the energy, an Atorch USB Tester was used, and Figure 6 presents a picture of it.

IV. Results and Discussions

The first result to show is about the preprocess performance. To preprocess the data to train was necessary 157.49 seconds and consumed 180 Wh, while to preprocess the data to test, 28.62 seconds and 35 Wh.

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

[6]https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

[7]https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html

[8]https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html

Fig. 6. An Atorch USB Tester was used to measure the energy consumption of running ML algorithms on the Raspberry Pi 4 Model B.

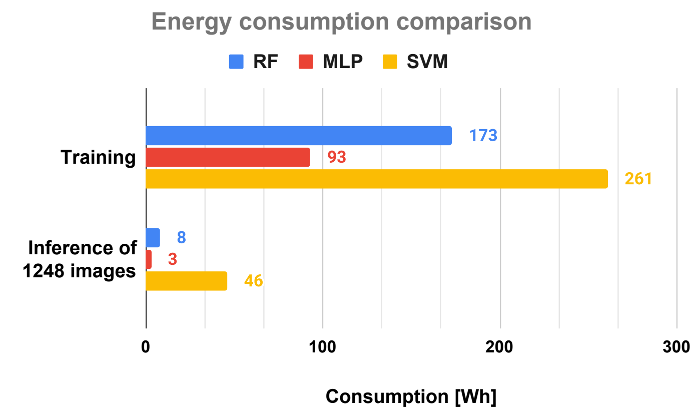

Fig. 8. Energy consumption comparison between the three approaches. The energy used to pre-process data was not included in these results.

The second result is about the time spent by the models to train and execute the tests. Figure 7 shows this comparison, where each color represents one model and the values are in seconds. Furthermore, Figure 8 presents a comparison between the energy consumption by each model in the training and execution process, where again each color represent one model and the values are in Wh. The execution process was done considering the pose estimation for 1248 images, so in the graph, the label used for the execution is *Inference of 1248 images*.

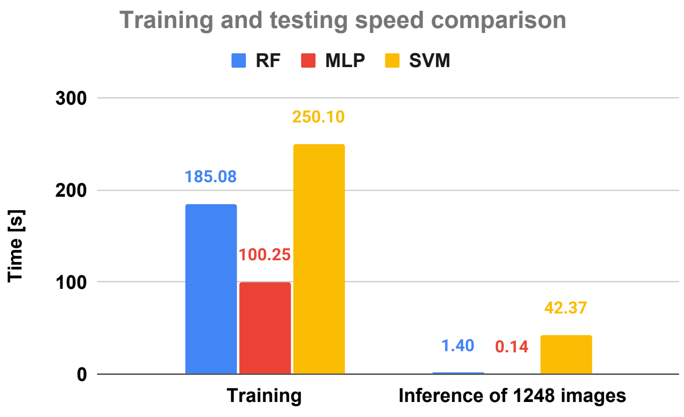

Fig. 7. Training and testing time for the three ML approaches. The time to pre-process data was not included in these results.

The next result is about the quality of the models, i.e., the errors between the estimated values and the ground truth, and their physical sizes. Table I presents these comparisons, where the MAE presents their respective errors with x and y in meters and θ is in degrees. The model size columns present the sizes of each model, measured in kB (kilobytes).

Analyzing Figure 7, it is possible to notice that the SVM algorithm used more time to train and execute than the others. MLP was the fastest approach, even in the training and in the execution. The same happens with energy consumption, where SVM used more energy when compared with the others, and MLP, again, was the best approach. The response time

TABLE I
COMPARISONS BETWEEN THE APPROACHES ON ESTIMATION ERROR (MAE) AND SIZE IN MEMORY.

	MAE			Model size		
	x [m]	**y[m]**	**θ[°]**	**x[kB]**	**y[kB]**	**θ[kB]**
RF	0.009	0.012	3.46	138400	147500	180000
MLP	0.029	0.046	28.47	26	26	30
SVM	0.030	0.027	47.64	1	1	2700

during the execution is in order of milliseconds for each image for MLP and RF, indicating it is possible to use this kind of approach to make predictions during the competition because the results indicate that to predict the pose for each image, it was necessary no more than a couple of milliseconds.

Regarding the quality of the models, the predictions were not good, obtaining errors in a centimeter scale. In this aspect, the best approach was RF. But it is important to highlight these results will be used as observations in stochastic/position/data fusion filters (such as Extended Kalman and Mahalanobis Filters), which means that they are not the final results. Furthermore, the models are not optimized, and only the standard hyper-parameters were used, which means there is a margin to improve their quality.

A relevant aspect is the size of the models, presented by Table I. The SVM presents the models to x and y targets in a very small size (1kB), but for θ the size was considerably bigger (2.7MB). On the other hand, the sizes of the MLP models were consistently small (between 26 and 30kB). RF, which presented the best results, had the biggest size, around 4 orders of magnitude when compared to the others, with models achieving 180MB.

The size of the models in the reduced part of the field was not a problem but indicates that using the whole field, where more images are necessary, the models will increase, and consequently, they will become a problem. To prove this point, using a computer, the same algorithms were trained using the data from the whole environment. MLP models continue in a small size, achieving, each one, no more than 1MB. RF models achieve between 3 and 5GB. This fact indicates that it is necessary to have at least 9GB of RAM to execute the complete prediction, which is a considerable quantity when compared with the Raspberry Pi used in this work, that only has 4GB available. Finally, the SVM models were not able to train, because it would take a lot of time. For example, to train the SVM model to *x* target, it would be necessary more than 72 hours.

V. Conclusions

This work aimed to validate the feasibility of ML algorithms in an embedded system focused on the localization problem of the RobotAtFactory 4.0 competition. A simulated scenario was used to execute the tests and collect the results. The focus of this work was only on the feasibility of the models rather than the quality of the estimations by models. The results from Section IV showed that the three approaches were feasible to solve the localization problem in a Raspberry Pi 4 model B if the problem is considerably small, i.e., the data is manageable. Therefore, there is a practical limit for the number of images used in training, resulting in a limitation on the grid's resolution, which impacts estimation accuracy. However, if the grid's resolution is significant, such as 1cm or less, only the MLP can be executed, and the training should not be executed in the Raspberry Pi.

Another interesting point is about the energy consumption, where MLP was the algorithm that had the lowest consumption, while the SVM had an order of magnitude higher than the MLP. Despite the position estimation error being in the order of centimeters, there is still a margin to improve the quality of the models by optimizing their hyper-parameters. Furthermore, after this improvement, it is possible to use filters to aggregate different predictions and improve the final pose prediction. Since the focus of this paper is only on the execution performance and the feasibility of the models, the improvement of the accuracy and the precision were not treated.

In future work, we plan to optimize the model's quality, i.e., equalize the errors before the performance comparison and test other ML techniques. In addition, we intend to implement the whole localization system, with the filters, and implement this system in a real environment (doing the necessary adjustments, such as retraining the models), where the robot has to take the pictures, process, and make the prediction while performing other tasks.

Acknowledgment

This work has been supported by FCT - Fundação para a Ciência e Tecnologia within the Project Scope: UIDB/05757/2020 and UIDP/05757/2020 and SusTEC (LA/P/0007/2021). The project that gave rise to these results received the support of a fellowship from "la Caixa" Foundation (ID 100010434). The fellowship code is LCF/BQ/DI20/11780028. João Braun is PhD student at Faculty of Engineering of University of Porto.

References

[1] Huang, S., & Dissanayake, G. (1999). Robot localization: An introduction. Wiley Encyclopedia of Electrical and Electronics Engineering, 1-10.

[2] Grewal, M. S., Weill, L. R., & Andrews, A. P. (2007). Global positioning systems, inertial navigation, and integration. John Wiley & Sons.

[3] Nessa, A., Adhikari, B., Hussain, F., & Fernando, X. N. (2020). A survey of machine learning for indoor positioning. IEEE access, 8, 214945-214965.

[4] Braun, J., Júnior, A. O., Berger, G., Pinto, V. H., Soares, I. N., Pereira, A. I., Lima, J., & Costa, P. (2022). A robot localization proposal for the RobotAtFactory 4.0: A novel robotics competition within the Industry 4.0 concept. Frontiers in Robotics and AI, 9. https://doi.org/10.3389/frobt.2022.1023590

[5] Klein, L. C., Braun, J., Mendes, J., Pinto, V. H., Martins, F. N., de Oliveira, A. S., Wörtche, H., Costa, P., & Lima, J. (2023). A Machine Learning Approach to Robot Localization Using Fiducial Markers in RobotAtFactory 4.0 Competition. Sensors, 23(6), 3128. https://doi.org/10.3390/s23063128

[6] Yazici, M. T., Basurra, S., & Gaber, M. M. (2018). Edge machine learning: Enabling smart internet of things applications. Big data and cognitive computing, 2(3), 26.

[7] Braun, J., Júnior, A. O., Berger, G. S., Lima, J., Pereira, A. I., & Costa, P. (2022, April). RobotAtFactory 4.0: a ROS framework for the SimTwo simulator. In 2022 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC) (pp. 205-210). IEEE.

[8] Costa, P., Gonçalves, J., Lima, J., & Malheiros, P. (2011). Simtwo realistic simulator: A tool for the development and validation of robot software. Theory and Applications of Mathematics & Computer Science, 1(1), 17-33.

[9] Kalman, R. E. (1960). A New Approach to Linear Filtering and Prediction Problems. Journal of Basic Engineering, 82(1), 35–45. https://doi.org/10.1115/1.3662552

[10] Welch, G., & Bishop, G. A. (1995). An Introduction to the Kalman Filter. New York EBooks, 1(4), 1–16. http://academic.csuohio.edu/simond/reduce/ijar.pdf

[11] Fox, D., Burgard, W., & Thrun, S. (1999). Markov localization for mobile robots in dynamic environments. Journal of artificial intelligence research, 11, 391-427.

[12] Gordon, N. J., Salmond, D. J., & Smith, A. F. (1993, April). Novel approach to nonlinear/non-Gaussian Bayesian state estimation. In IEE proceedings F (radar and signal processing) (Vol. 140, No. 2, pp. 107-113). IET Digital Library.

[13] Lauer, M., Lange, S., & Riedmiller, M. (2006). Calculating the perfect match: an efficient and accurate approach for robot self-localization. In RoboCup 2005: Robot Soccer World Cup IX 9 (pp. 142-153). Springer Berlin Heidelberg.

[14] Besl, P. J., & McKay, N. D. (1992, April). Method for registration of 3-D shapes. In Sensor fusion IV: control paradigms and data structures (Vol. 1611, pp. 586-606). Spie.

[15] Biber, P., & Straßer, W. (2003, October). 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) (Vol. 3, pp. 2743-2748). IEEE.

[16] Sobreira, H., Costa, C. M., Sousa, I., Rocha, L., Lima, J., Farias, P. C. M. A., & Moreira, A. P. (2019). Map-matching algorithms for robot self-localization: a comparison between perfect match, iterative closest point and normal distributions transform. Journal of Intelligent Robotic Systems, 93, 533-546.

[17] Michail, K., Cain, B., Carroll, S., Anand, A., Camden, W., & Nikolaos, V. (2021). Fiducial Markers for Pose Estimation. Journal of Intelligent Robotic Systems, 101(4).

[18] Kalaitzakis, M., Carroll, S., Ambrosi, A., Whitehead, C., & Vitzilaios, N. (2020, September). Experimental comparison of fiducial markers for pose estimation. In 2020 International Conference on Unmanned Aircraft Systems (ICUAS) (pp. 781-789). IEEE.

[19] Lopes, M. Â. L. (2021). Benchmark de Sistemas Embebidos para Machine Learning em Visão Computacional (Doctoral dissertation).

[20] Sammut, C., & Webb, G. I. (Eds.). (2011). Encyclopedia of machine learning. Springer Science Business Media.