Deep-Learning-Based Visual Odometry Models for Mobile Robotics

Frederico Luiz Martins de Sousa
Universidade Federal de Ouro Preto
Ouro Preto, Brasil
Email: fredericolsousa96@gmail.com

Natália F. de C. Meira
Universidade Federal de Ouro Preto
Ouro Preto, Brasil
Email: nataliaf.meira@gmail.com

Ricardo Augusto Rabelo Oliveira
Universidade Federal de Ouro Preto
Ouro Preto, Brasil
Email: rrabelo@gmail.com

Mateus Coelho Silva
Universidade Federal de Ouro Preto
Ouro Preto, Brasil
Email: mateuscoelho.ccom@gmail.com

***Abstract*—Odometry is a common problem in navigation systems where there is a need to estimate the position of the vehicle or carrier in the environment. To perform autonomous tasks, robotic or intelligent devices need to be aware of their position in the environment. There are many strategies to solve an odometry problem. This work explores a visual odometry solution with a deep neural network to infer the robotic vehicle's position in a known and mapped environment. The first robot, equipped with a LIDAR, IMU, and camera, maps the environment through a SLAM technique to perform this task. The data gathered by this first robot is used as ground truth to train the neural network, and later, other robots with only one camera can locate themselves in the environment. We also propose a validation and evaluation of the neural network.**

***Index Terms*—mobile robotics, odometry, edge-computing, deep neural networks, Robot Operating System**

I. Introduction

Odometry is a common problem in navigation systems where there is a need to estimate the position of the vehicle or carrier in the environment. To perform tasks such as autonomous navigation and obstacle avoidance, the platform needs to contain and preserve information about its position and orientation in space [1]. However, the odometry problem can be solved by many different strategies. The most common strategy to locate the vehicle position is through GPS (Global Positioning System) devices [2]. These devices rely on global satellites and radio waves to output a position on the earth's surface. Hence, GPS devices sometimes are not the best choice for self-localization because of low signal in forests, caves, or even underwater.

To the use mentioned above cases, alternative strategies can be used to determine the position and orientation of the vessel. Hereafter we present the most common strategies: visual, laser, wheel, and IMU odometry. The visual odometry consists of estimating the position from the difference between images and can be done with stereo cameras [3] or monocular cameras [4]. The laser odometry is based on the estimation of the robot position by finding environment features through planar or spatial LIDARs (Light Detection and Ranging) [5]. The wheel odometry is measured by wheel encoder sensors that sense how much and how fast the vehicle wheels are spinning, thus calculating the wheel revolutions such as early NASA's Mars rovers [6]. The IMU (Inertial Measurement Unit) odometry uses the IMU measurements to calculate the position. In order to perform this task, the IMU outputs the acceleration and gyroscope on each of the three axes [7].

All of the mentioned strategies contain limitations and an associated cost, both computational and financial. In this work, we explore deep neural networks to solve a visual odometry problem. Deep neural networks are artificial neural networks that mimic the behavior of biological neural networks and have been used to solve a vast number of computer vision problems, as shown by Voulodimos et al. [8].

We also explore a SLAM (Simultaneous Localization and Mapping) approach to gather the needed data to build our network ground truth. SLAM is a technique in which the vessel both maps the environment and locates itself in it, and it can be achieved in several ways, with multiple sensor types [9]. In this paper, we propose a different approach to the visual odometry problem. We trained a regression neural network to locate a vehicle/vessel in an already known and mapped environment. To perform this task, we used the map generated by the SLAM algorithm as our ground truth. This allows us to relate an image with a given position and orientation on the map. This strategy can be applied to a use case where there are two or more robots. The first robot explores the environment first and then creates the map and builds the dataset to train the network. This network enables other robots to extract a position from a monocular image.

To describe our approach, we present the theoretical references used to complete this work in Section II and III. In Section IV, we describe the robot's features and configurations used. We also present the training, evaluation, and dataset collection methodology for the regression deep neural network. In Section V, we show the results obtained by our approach. Finally, in Section VI we discuss the results achieved and future improvements.

II. Theoretical References

In this section, we present the theoretical references for the presented proposal. Initially, we discuss the Edge AI perspective and its implications, as well as some implementations. Then, we present some discussions on mobile robotics and SLAM algorithms. These are the two key concepts behind the idea proposed in this work.

A. Edge AI

The information age, driven by the significant increase in computing and storage devices, is taking cloud computing services to the edge [10]. Artificial Intelligence (AI) applications through Deep Neural Networks (DNNs) highlighted the challenge of performing intensive tasks in the cloud and with limited computing resources and, currently, the benefits of artificial edge intelligence (edge AI) can be introduced in the Internet of Things (IoT), with more powerful edge devices [11]–[13].

The combination of edge computing and cloud computing applications brings benefits such as backbone network alleviation, reducing network overhead, increased response speed, and robust cloud backup when the edge cannot afford [10]. In addition, edge computing takes cloud computing power to the edge, bringing tasks and data closer to end devices and allows DNN-based applications to run responsively with real-time edge AI support [11], [13].

Shi et al. (2020) [13] presented the main challenges for the communication of edge AI systems and presented efficient communication techniques for training and inference tasks. For Lin et al. (2020) [14], the explosive growth of internet-connected mobile computing and IoT applications will generate zillions of bytes of data at the network edge pushing the boundaries of AI to the edge.

Mazzia et al. (2020) [15] implemented an embedded edge AI solution for real-time apple detection in orchards, with the YOLOv3-tiny algorithm on three embedded platforms. Klippel et al. (2020) [16] implemented an edge AI solution to detect failures in iron ore conveyor belts. Thus, work with edge AI applications increasingly contribute to the deployment of intelligent solutions on edge.

B. Mobile Robotics and SLAM

Mobile robots are mechanical systems capable of interact and move in the environment. However, regulate the communication between sensors, actuators, and algorithms can be a challenging task. To perform this task, a series of sensors, actuators, and a type of intelligence (through algorithms or routines) must be implemented in the robot [17].

The ROS (Robot Operating System) is an open-source meta operating system for robots. ROS has a series of services and other OS features for robots. It can facilitate the process of building a modular and cohesive robot due to its publisher/subscriber architecture [18]. ROS also comes with a powerful tool for 3D visualization and interaction with the robot's environmental sensing, called *RViz* [19]. *RViz* helps understanding how the robot is interpreting the sensed data, which is also helpful for debugging. Another advantage of ROS is the wide availability of open-source algorithms, from simple motor controllers to more complex SLAM techniques.

There are numerous SLAM algorithms available to use in ROS, and each one will fit a use case. In this work, we use a planar LIDAR for the SLAM process. Santos et al. [20] propose an evaluation for 2D planar LIDAR's SLAM algorithms available in ROS. According to their work, *gmapping* [21] and *HectorSlam* [22] show one of the smallest error accumulations in SLAM for 2D planar LIDARs. In Section IV we present the results of applying both of these algorithms in our robot.

Mobile robots are required more processing power given the growing features a robot needs to perform complex tasks. In this case, two approaches are viable: using powerful embedded hardware or distributing the computation through the network. Due to its publisher/subscriber architecture, ROS enables an easy way to distribute the computation across multiple devices [23] [24].

Silva et al. [25] shows a data quality test between multiple devices consuming data from a robot. Their work displays that the real-time quality factor decreases as more devices connect to the network. This information indicates that for real-time processing use cases that rely on multiple devices connected to the same network, it is interesting to have more powerful platforms embedded in the robot. This analysis outlines the edge computation paradigm [26].

III. Related Work

Odometry has a critical role in robotic devices, and as previously mentioned, there are different techniques to measure it. Our related works section focuses on papers that use deep learning, LIDARs, or visual odometry in mobile platforms. Then we outline the differences between our approach to theirs.

In Shan et al. [5], they introduce a lightweight and ground-optimized lidar-based odometry. Their work concerns keeping a lightweight application to improve performance on embedded devices like NVIDIA's Jetson family products. The critical part of their approach is that their odometry measurement relies on the LIDAR sensing the ground, making the use case of their strategy restricted to 3D lasers, which can increase the robotic platform's cost. Our approach focuses on having only one robot with a lower-cost planar lidar to map the environment and then using the created map to feed a neural network that enables other robots to traverse the mapped environment with only a monocular camera.

Both Zhan et al. [27] and Li et al. [28] propose monocular camera-based odometry through pixel motion prediction. These approaches also use deep learning to predict the movement between the pixel motion from frame t1 and t2. This data enables their network to extract depth information on the monocular image and predict the position and pose of the vehicle/robotic platform. They also outline their performance on KITTI dataset benchmarks but cannot be compared due to different metrics used to measure their error. In our work, we do not measure pixel variation. We only relate a frame with

a position on the mapped environment. Then we feed these positions with their respective images to a regression neural network to predict the vessel position within the mapped environment.

Yan et al. [29] fuses two popular visual odometry algorithms, LOAM (Lidar odometry and mapping) [30] and viso2 [4] to extract odometry information. The viso2 algorithm has a known error rate for monocular cameras. They propose and compare three fusion methods to benefit from both algorithms and thus strengthening the accuracy and decreasing the odometry drift error. In our approach, we fuse the *rf2o* ROS package with IMU information for a more accurate position ground truth training for the network. This data fusion increases odometry precision due to the robot's orientation given by the IMU which strengthens the measurements.

IV. Methodology

In this section, we introduce the robotic platform and the adopted training methodology for the neural network. The text splits into three subsections, one to describe the robotic platform, the other to outline the ground truth building process, and the last one to detail the neural network's training process.

A. Robotic Platform

The robotic platform used in this work uses a Raspberry Pi 4 as the central hardware unit. This unit represents our master node in ROS architecture, where most of the algorithms execute. The robot is equipped with four motors with mecanum wheels for omnidirectional locomotion in the environment. We also have an IMU connected to an Arduino Uno, which communicates to the master node through the serial ports. Our main SLAM sensor bases on the planar LIDAR X2 by Ydlidar. The X2 LIDAR is a low-cost laser with a scan frequency of 7hz and a range distance of 8 meters, which is enough for our indoor testing purposes. We use the company's ROS node to publish the laser information as a native ROS *LaserScan* message type. Regarding the robot's alimentation system, the motors are fed by a 12V Li-Po battery, and the rest of the system is powered by a regular 5V power bank.

We also created a URDF (Universal Robot Description File) with the purpose of better visualization of the robot's interaction with the map in *RViz* tool. The URDF is also needed to display the correct transformations from the robot's position in the SLAM map. Figure 1 shows the robotic platform and its URDF on *RViz*.

B. Ground Truth Collection Methodology

To collect our ground truth, as mentioned in Section II, we evaluated two SLAM methods. In the process of environment mapping through SLAM, we developed a teleoperation module. Due to its mecanum wheels, our robot can move freely in both x and y axes, so we developed an omnidirectional teleoperation script.

At first, we used *HectorSlam* with no specific odometry data. *HectorSlam* has a configuration mode where it only gets the odometry through the laser and other global optimizers.

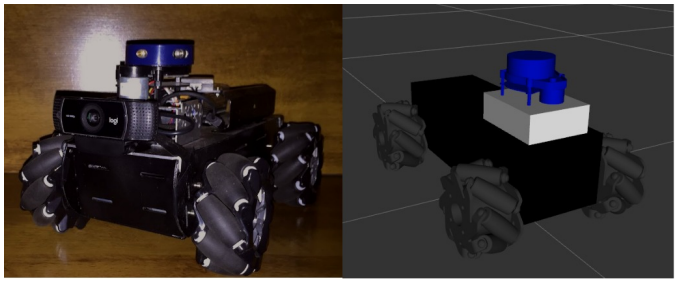

Fig. 1. Real robotic platform and its virtual representation in URDF.

This configuration mode can be useful for robots or mobile devices that cannot retain more reliable odometry measurements. We achieved good mapping results with this configuration, but laser-only odometry resulted in a few mapping errors when the teleoperation needed to maneuver the robot to rotate in its axis. This condition is where the omnidirectional model of our robot was helpful to get a good map, but visually, it was still not accurate enough to use as a neural network's ground truth. Figure 2 shows the errors we got using *HectorSlam* with this configuration.

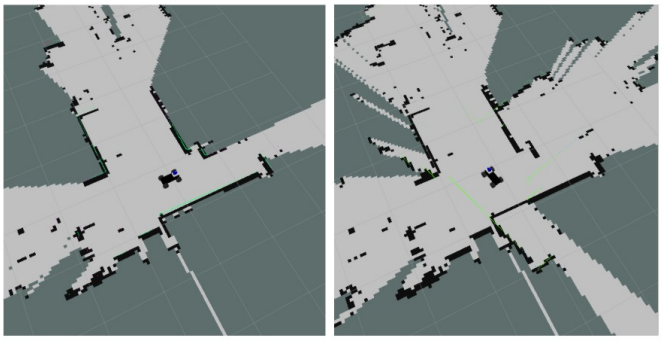

Fig. 2. HectorSlam rotation errors.

To improve our map, we configured odometry through ROS package *rf2o* [31] planar laser odometry. This package outputs the robot position in the three possible axes (x,y,z), and these values measurements are in meters. For better odometry data, we added an IMU and fused its orientation output with the position information from the laser, given by the *rf2o* package. We publish the fused data on the */odom* topic. From this odometry data, we validated the output laser position with a measuring tape, measuring the robot displacement in both x and y axis.

The *rf2o* package fused with orientation from IMU had better mapping results in *HectorSlam* when the robot spin on its axis. However, if we spin the robot too fast, the laser feature-based odometry would still be lost, and consequently, the map accuracy decreases. To improve this, we changed the SLAM algorithm to *Gmapping*. The results achieved by *Gmapping* were better, considering the robot's spin on its axis problem. Figure 3 shows the environment map, captured by *Gmapping*.

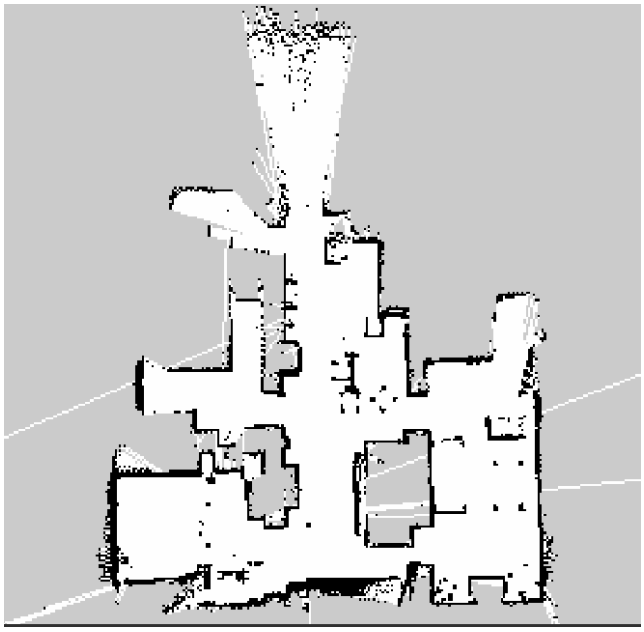

Fig. 3. Mapped robot environment through *Gmapping* algorithm.

Once we were able to get a good SLAM map, we built a script to relate an image to a specific position within the map. The script captures an image through a Logitech C922 USB camera and then subscribes to the */odom* topic to register the current robot's position and pose. Our ground truth annotation file consists of an image filename, the robot's current x and y position, and we also compute the robot's orientation through the yawn Euler angle value.

To create the dataset, we teleoperated the robot through the environment capturing the images and relating these images to a particular position. We fixed a starting point in the environment to be the origin of our map. Thus, all of the collections needed to start at this same point so that the data could make sense. The created dataset contains 6203 images for training and other 1183 images for testing. The robot's environment considered for this work is a domestic scenario, where the robot does the mapping.

C. Training Process

To train the network, we tested three different neural network architectures. The proposed network architectures receive a 640x480 RGB image and output the x and y position in the map and the robot's orientation. Therefore, we trained a custom model architecture and other two with literature known backbones. Our custom model is a simple network with five convolutional layers, alternating with five 2D max-pooling layers. We also add two dense layers at the end for the output. We use ReLu as the activation function for all layers, except the last layer, which has a linear activation function. We also used MAE (Mean Absolute Error) as our loss function and Adam as the optimizer. The second network that we used is based on *VGG-18*, and the third uses a *ResNet50* backbone. We also added two dense layers at the end of each network to output the desired values. Using these pre-trained weights benefits the model to learn faster. All models were trained through 250 epochs.

V. Results

To evaluate the trained model's accuracy, we used the mean Euclidean distance as a metric. This metric calculates the distance in centimeters between the predicted robot's position and ground truth. The mean distance from predictions and ground truth and the standard deviation from each model are shown in Table I.

Model	Mean Error Distance (m)	Standard Deviation
ResNet50	0.4587	0.4924
VGG-18	0.7587	0.5752
Custom Model	0.6299	0.8774

TABLE I
MODEL COMPARISON WITH THE EUCLIDIAN DISTANCE METRIC.

Table I shows that the *ResNet50* model has the smallest mean error and deviation. We have to consider that this model has 50 layers, which is considerably larger than the others. It had a 45.87 centimeters mean error which corresponds to roughly 10.02% of the biggest distance in our mapped environment, which is 4.5758 meters. The *VGG-18* model had the largest mean error, which corresponded to 75.87 centimeters, a 16.58% variation from our biggest distance. Lastly, our custom architecture performed 62.99 centimeters mean error, which corresponds to 13.76% of the biggest distance. Although our model performed better than the *VGG-18* architecture, its deviation was considerably larger.

We also propose a visual form of evaluation of the models. Through a scatter plot, it shows the correspondence of the ground truth positions with the predicted ones. The scatter plot is useful to visualize whether the predicted points fit the ground truth since we use a regression model. Figure 4 shows our custom model scatter plot, where we can observe that there are still some errors in predicting the robot's position. While in Figure 5 we show the scatter plot of the *VGG-18* indicating that it also makes wrong predictions, as Table I shows. Figure 6 displays the scatter plot of the *ResNet50* model, and it was the model that best fitted the ground truth. These visual graphs agree with our metric of evaluation, displayed in Table I. Thus, we chose the *ResNet50* model for further evaluation.

For better evaluation of the *ResNet* model, we analyzed an error histogram. Figure 7 shows this error histogram performed by this model. It displays that most of the errors are within the mean error distance showed in Table I. We also built a boxplot to analyze the error dispersion. Figure 8 shows this boxplot. In this graph, we can see that the error is well distributed through the mean error showed in Table I.

VI. Conclusions and Future Work

This work presented a deep learning approach to the visual odometry problem applied to a mapped domestic environment. We collected the ground truth relating an image to a position

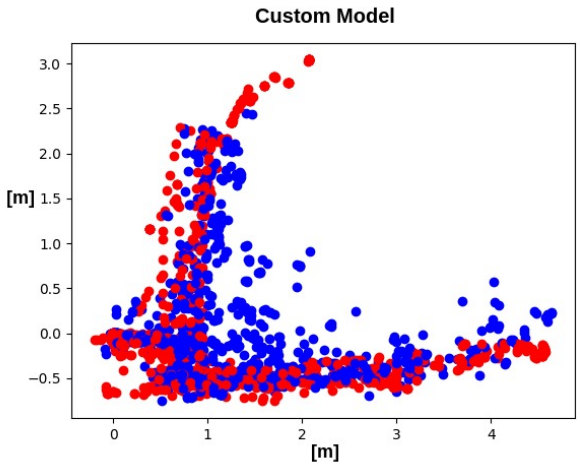

Fig. 4. Custom model prediction/ground truth scatter plot. Ground truth points in red, and prediction points in blue.

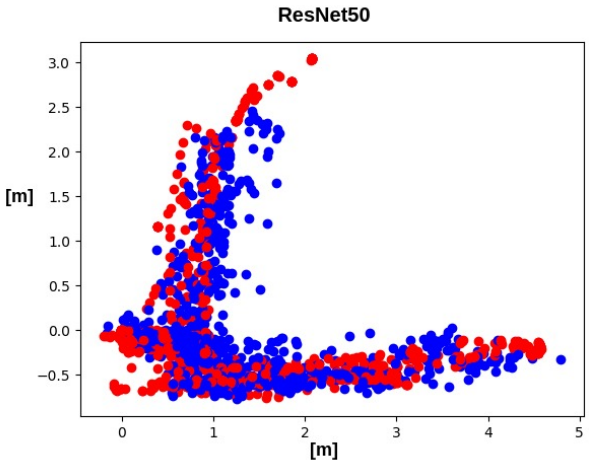

Fig. 6. *ResNet50* prediction/ground truth scatter plot. Ground truth points in red, and prediction points in blue.

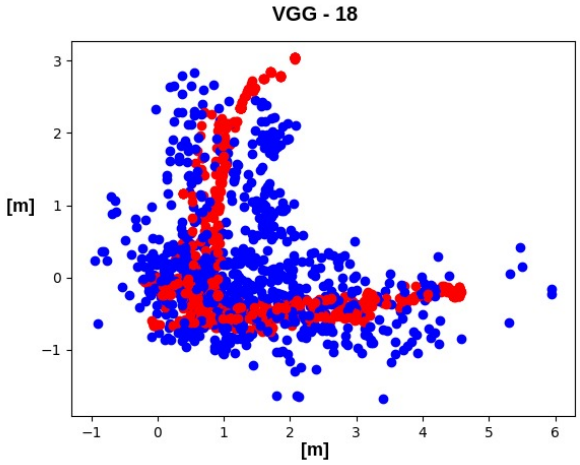

Fig. 5. *VGG-18* prediction/ground truth scatter plot. Ground truth points in red, and prediction points in blue.

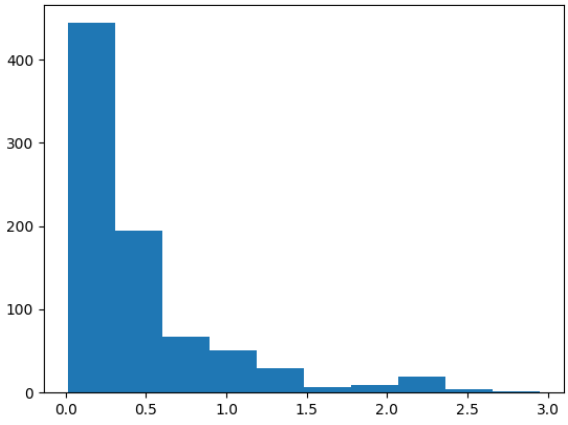

Fig. 7. *ResNet50* model error histogram.

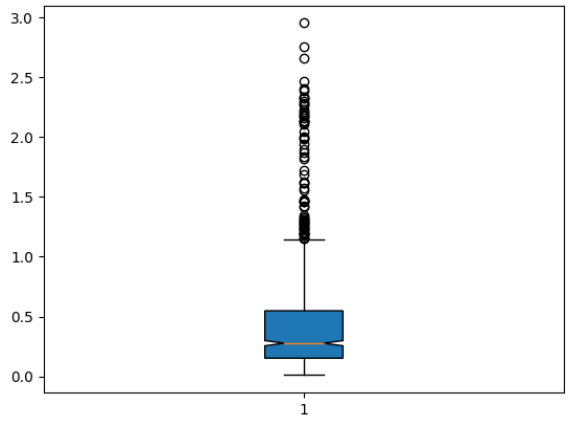

Fig. 8. *ResNet50* model error boxplot.

and orientation in the map. We also evaluated three neural network architectures to predict the robot's position and pose.

For this matter, we proposed a methodology using a Robotic Platform containing a raspberry pi 4, four motors with omnidirectional mecanum wheels, and an Inertial Measurement Unit (IMU). Furthermore, the platform also uses an Arduino Uno to handle some tasks. There is also a virtual correspondent model integrated using the Robotics Operational System (ROS). The robot performed planar laser odometry with the help of the IMU and related the positions with images captured from a camera.

To obtain a model to approach the visual odometry, we collected 6203 images using the process described above. Then, we proposed the usage of three different CNN architectures. At first, we tried to use a custom CNN model with five convolutional layers with 2x2 max-pooling layers afterward.

We experimented with the VGG-18 layer as the backbone, changing the final layers to the proposed function. Finally, we also tried using ResNet50 as the backbone.

The evaluation process showed that a *ResNet50* model was able to learn more of the environment's features. This model displayed the lowest error rate between the three evaluated models. The mean error of 45.87 centimeters is still high, considering the environment's total size of 4.5758 meters. To improve this, we can collect more data and analyze it through our defined metrics.

In future improvements of this work, we will evaluate a metric to validate the orientation information of the network. This evaluation method was not found in the literature within a scope that would make sense to our application. Another future improvement is a method of validating the LIDAR's measurements through a computer vision algorithm.

ACKNOWLEDGMENT

The authors would like to thank UFOP, CAPES e CNPq for supporting this work. This work was partially financed by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) - Finance Code 001, and by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) - Finance code 308219/2020-1.

REFERENCES

[1] S. A. Mohamed, M.-H. Haghbayan, T. Westerlund, J. Heikkonen, H. Tenhunen, and J. Plosila, "A survey on odometry for autonomous navigation systems," *IEEE Access*, vol. 7, pp. 97 466–97 486, 2019.

[2] D. Bender, W. Koch, and D. Cremers, "Map-based drone homing using shortcuts," in *2017 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*. IEEE, 2017, pp. 505–511.

[3] R. Giubilato, S. Chiodini, M. Pertile, and S. Debei, "An evaluation of ros-compatible stereo visual slam methods on a nvidia jetson tx2," *Measurement*, vol. 140, pp. 161–170, 2019.

[4] A. Geiger, J. Ziegler, and C. Stiller, "Stereoscan: Dense 3d reconstruction in real-time," in *2011 IEEE intelligent vehicles symposium (IV)*. Ieee, 2011, pp. 963–968.

[5] T. Shan and B. Englot, "Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 4758–4765.

[6] Y. Cheng, M. Maimone, and L. Matthies, "Visual odometry on the mars exploration rovers," in *2005 IEEE International Conference on Systems, Man and Cybernetics*, vol. 1. IEEE, 2005, pp. 903–910.

[7] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, "Keyframe-based visual–inertial odometry using nonlinear optimization," *The International Journal of Robotics Research*, vol. 34, no. 3, pp. 314–334, 2015.

[8] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, "Deep learning for computer vision: A brief review," *Computational intelligence and neuroscience*, vol. 2018, 2018.

[9] T. Taketomi, H. Uchiyama, and S. Ikeda, "Visual slam algorithms: a survey from 2010 to 2016," *IPSJ Transactions on Computer Vision and Applications*, vol. 9, no. 1, pp. 1–11, 2017.

[10] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 869–904, 2020.

[11] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge ai: On-demand accelerating deep neural network inference via edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 447–457, 2019.

[12] X. Lin, J. Li, J. Wu, H. Liang, and W. Yang, "Making knowledge tradable in edge-ai enabled iot: A consortium blockchain-based efficient and incentive approach," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 12, pp. 6367–6378, 2019.

[13] Y. Shi, K. Yang, T. Jiang, J. Zhang, and K. B. Letaief, "Communication-efficient edge ai: Algorithms and systems," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 4, pp. 2167–2191, 2020.

[14] S. Lin, Z. Zhou, Z. Zhang, X. Chen, and J. Zhang, "Edge intelligence in the making: Optimization, deep learning, and applications," *Synthesis Lectures on Learning, Networks, and Algorithms*, vol. 1, no. 2, pp. 1–233, 2020.

[15] V. Mazzia, A. Khaliq, F. Salvetti, and M. Chiaberge, "Real-time apple detection system using embedded systems with hardware accelerators: An edge ai application," *IEEE Access*, vol. 8, pp. 9102–9114, 2020.

[16] E. Klippel, R. Oliveira, D. Maslov, A. Bianchi, S. E. Silva, and C. Garrocho, "Towards to an embedded edge ai implementation for longitudinal rip detection in conveyor belt," in *Anais Estendidos do X Simpósio Brasileiro de Engenharia de Sistemas Computacionais*. SBC, 2020, pp. 97–102.

[17] L. Jaulin, *Mobile robotics*. John Wiley & Sons, 2019.

[18] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[19] A. Cid, M. Nazário, M. Sathler, F. Martins, J. Domingues, M. Delunardo, P. Alves, R. Teotônio, L. G. Barros, A. Rezende *et al.*, "A simulated environment for the development and validation of an inspection robot for confined spaces," in *2020 Latin American Robotics Symposium (LARS), 2020 Brazilian Symposium on Robotics (SBR) and 2020 Workshop on Robotics in Education (WRE)*. IEEE, 2020, pp. 1–6.

[20] J. M. Santos, D. Portugal, and R. P. Rocha, "An evaluation of 2d slam techniques available in robot operating system," in *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE, 2013, pp. 1–6.

[21] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007.

[22] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," in *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, November 2011.

[23] E. Munera, J.-L. Poza-Lujan, J.-L. Posadas-Yague, J. Simo, and J. F. B. Noguera, "Distributed real-time control architecture for ros-based modular robots," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 11 233–11 238, 2017.

[24] M. Liu, J. Niu, and X. Wang, "An autopilot system based on ros distributed architecture and deep learning," in *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*. IEEE, 2017, pp. 1229–1234.

[25] M. C. Silva, F. L. M. de Sousa, D. L. M. Barbosa, and R. A. R. Oliveira, "Constraints and challenges in designing applications for industry 4.0: A functional approach." in *ICEIS (1)*, 2020, pp. 767–774.

[26] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[27] H. Zhan, C. S. Weerasekera, J.-W. Bian, and I. Reid, "Visual odometry revisited: What should be learnt?" in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 4203–4210.

[28] R. Li, S. Wang, Z. Long, and D. Gu, "Undeepvo: Monocular visual odometry through unsupervised deep learning," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 7286–7291.

[29] M. Yan, J. Wang, J. Li, and C. Zhang, "Loose coupling visual-lidar odometry by combining viso2 and loam," in *2017 36th Chinese Control Conference (CCC)*. IEEE, 2017, pp. 6841–6846.

[30] J. Zhang and S. Singh, "Loam: Lidar odometry and mapping in real-time." in *Robotics: Science and Systems*, vol. 2, no. 9, 2014.

[31] M. Jaimez, J. G. Monroy, and J. González-Jiménez, "Planar odometry from a radial laser scanner. a range flow-based approach," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 4479–4485. [Online]. Available: http://mapir.isa.uma.es/mapirwebsite/index.php/mapir-downloads/papers/217