Indoor Localization of Mobile Robots using Inertial Data and Deep Learning ⋆

Harumi Pino * **Ubaldo Ruiz** *,**

* *Centro de Investigación Científica y de Educación Superior de Ensenada, Ensenada, Baja California 22860, Mexico (e-mail: {hpino,uruiz}@cicese.edu.mx).*
** *Corresponding author*

Abstract: Localization is one of the fundamental problems in mobile robotics, being the basis of autonomous navigation and trajectory planning. While technologies such as GPS, trilateration, or triangulation are widely used outdoors, they are not applicable in indoor environments. The most common methods to address this problem are SLAM algorithms based on visual sensors, such as RGBD cameras or laser ranging sensors. However, these methods are susceptible to failures in environments with poor illumination and regions without texture or transparent walls. To avoid those issues, a novel alternative is to utilize machine learning methods that take inertial data to compute the position of a mobile robot as it moves in the environment. This work compares seven different deep neural network models to estimate the robot's displacements in the environment and reconstruct its current position: ResNet-18, ResNet-50, RestNet-Small, MobileNetV3-Small, MobileNetV3-Large, CNN + LSTM, and RBCN-Net. We performed experiments in the empty space and two different environments with obstacles. The best results for the empty space were obtained using the MobileNetV3 architecture, with a mean RMS error of 0.9794 through ten test paths. ResNet-18 obtained the best results for environments with obstacles, having a mean RMS error of 0.4571.

Keywords: Localization, Machine learning, Inertial sensor, Mobile robot

1. INTRODUCTION

Mobile robots play an increasingly important role in everyday life, being used in applications in different areas such as medical, manufacturing, mining, and education, among others (Ruan et al., 2019). Localization is one of the fundamental problems in the field of mobile robotics. A robust and accurate localization method is the foundation for autonomous navigation and robotic trajectory planning (Xiao et al., 2017).

The Global Positioning System, or GPS, is widely used for outdoor localization applications. However, satellite signals do not adequately penetrate complex indoor areas, so using this technology for indoor locations is not feasible. Other methods used outdoors, such as trilateration and triangulation, require an unobstructed line of sight, so their use is limited indoors. Those constraints have motivated the development of specific techniques for indoor location applications (Roy and Chowdhury, 2021).

The Simultaneous Localization and Mapping Problem, or SLAM for short, is the problem that simultaneously addresses the construction of a map of the environment and the robot's localization in that map. This methodology has been extensively studied over the past decades (Wen et al., 2020). Among the first methods that addressed this problem and are still among the most widely used are those based on the Extended Kalman Filter (EKF). Currently, various techniques are used to solve the SLAM problem based mainly on vision sensors or cameras. These include GraphSLAM (Thrun and Montemerlo, 2006), RGBD SLAM (Engelhard et al., 2011), RTAB-MAP (Labbe and Michaud, 2013) and ORB-SLAM3 (Campos et al., 2021). However, methods that use cameras have some troubles when performing indoor SLAM. Corridors with white walls, texture patterns, or repeated lights can cause confusion between similar locations, resulting in incorrectly created maps and incorrect localization (Hashemifar et al., 2019).

LiDAR (Laser Imaging Detection and Ranging) sensors, which can determine the distance to an object using a beam of laser light, are another of the most commonly used sensors in localization. However, methods using these types of sensors fail in environments with transparent or reflective walls or surfaces, bars, or dust in the air (Dobrev et al., 2018). In addition, cameras and LiDAR produce large volumes of data, making these methods computationally more demanding, especially on resource-limited devices.

The problems described above have led to the emergence of other localization methods, such as inertial odometry. The sensors used for inertial localization have become increasingly cheaper and more compact, making them easy to incorporate into any type of mobile robot or vehicle. However, since they are used cumulatively, inertial localization provides pose estimates with an increasing and unbounded error, commonly called drift error.

⋆ This work was supported by CONAHCYT under Grant A1-S-21934.

On the other hand, thanks to recent advances in artificial intelligence, different machine learning techniques have been successfully applied to mobile robot localization problems, such as data fusion from different sensors (Li et al., 2019), sensor data correction (Brossard and Bonnabel, 2019), missing data synthesis (Pfeiffer et al., 2018), among others.

Different machine-learning techniques have been employed to correct the drift error in inertial odometry. In particular, good results have been obtained when a deep neural network is used to estimate the position of a person walking in an environment from inertial data captured by a mobile device (Chen et al., 2018). This suggests that a similar approach can be used for estimating the position of a mobile robot moving in an indoor environment with obstacles. In this work, we study the use of deep neural networks to estimate the robot's position in such environments. The main contributions of this paper are as follows:

(1) A database containing inertial data and the ground truth position of a TurtleBot3 robot moving in the empty space and following 36 different simulated paths was generated. 26 of them correspond to the training set and 10 to the testing set.
(2) 12 paths in an indoor simulated environment with obstacles were also included in the database. 10 correspond to the training set and 2 to the testing set.
(3) Unlike previous works, MobileNet and some of its variants are suggested and compared to compute the robot's displacement in the environment.
(4) Seven different deep neural network models to estimate the robot's displacements in the environment and reconstruct its current position are compared: ResNet-18, ResNet-50, RestNet-Small, MobileNetV3 Small, MobileNetV3 Large, CNN + LSTM, and RBCN-Net.
(5) The results show that MobileNet and ResNet18 architectures can achieve better results considering only inertial data than GMapping and Karto. These two SLAM algorithms are included in the Turtlebot3 software and use odometry and range data from a laser sensor.

2. RELATED WORK

Recent advances in microelectromechanical systems (MEMs) have enabled the development of inertial measurement units (IMUs) that are small and inexpensive enough to be implemented in robots and other mobile devices. However, these low-cost sensors have high levels of noise that propagate during navigation, leading to rapidly growing errors during location estimation, known as drift error (Chen, 2023).

To address the previous problem, various machine learning techniques have been used to attempt to correct the errors that result from employing localization methods based on IMUs.

Chen et al. (2018) presented a method called Inertial Odometry Neural Network (IONet) to constrain the inevitable drift of inertial systems. This method implements a deep neural network (DNN) consisting of long short-term memory (LSTM) layers, which is trained using inertial measurements from an IMU divided into independent time windows. The model receives as input one of these windows and delivers as output the change in position and orientation of the device during this time.

Silva do Monte Lima et al. (2019) introduced a model based on a CNN, combined with two bidirectional LSTM layers, which receives as the input data coming from an IMU with six degrees of freedom, i.e., three values of linear acceleration and three values of angular velocity. The data is divided into windows of 200 measurements each, and the model outputs the change in position and orientation between two consecutive windows. Once the relative positions for all time windows are available, the agent's trajectory in the 3D space can be reconstructed.

To solve the problem of accumulated error during the use of inertial sensors, Esfahani et al. (2019) proposed a triple-channel Long Short-Term Memory (LSTM) deep network architecture based on the physical and mathematical models of IMUs, which they call AbolDeepIO. The model receives the measurements of acceleration, angular velocity, and the time interval between two consecutive measurements as input. In addition, the proposed method simulates the noise model in the training phase and becomes robust to noise during the testing phase.

Herath et al. (2021) presented a method called RoNIN (Robust Neural Inertial Navigation) to reconstruct the trajectory of users through inertial data from their cell phones. They tested three different neural network architectures (ResNet, LSTM, and TCN) as a basis for localization. Of these three architectures, LSTM and TCN performed slightly better than ResNet in estimating the user's position but took three to four times more training time.

Liu et al. (2020) proposed a method called TLIO (Tight Learned Inertial Odometry). This comprises a convolutional neural network, specifically the ResNet18 architecture presented by He et al. (2016), which calculates the 3D relative displacement and its covariance between two consecutive time instants. The output of this model is then sent to an extended Kalman filter (EKF) that estimates the agent's current state: 3D position, velocity, orientation, and IMU biases, in addition to a set of past poses.

Asraf et al. (2021) presented a method named PDRNet, which uses data from a cell phone's accelerometer and gyroscope to calculate the user's position. This method is divided into two steps: determining the cell phone's position on the user (in the hands or in the pocket, for example) and then computing the user's position in the environment. A single-layer convolutional network is implemented for the first step, while the ResNet-50 architecture is used for the second step, which predicts the change in the user's position and orientation.

Zhu et al. (2023), inspired by the work of Herath et al. (2021), created a new model called RBCN-Net for inferring the path followed by a user. They did this by taking the ResNet18 architecture as a base and combining it with bidirectional LSTM layers. They also added convolutional attention modules (CBAMs) ((Woo et al., 2018)) and normalization-based attention modules (NAMs) ((Liu

et al., 2021)). Including attention modules help the network distinguish the most important data from the input, while bidirectional LSTM layers allow it to learn temporal dependencies between data.

Wang et al. (2022) expanded on the work of Liu et al. (2020), arguing that the use of TLIO is unsuitable for mobile devices due to its high computational cost. Therefore, they proposed a lighter system called LLIO (Lightweight Learned Inertial Odometer), replacing the original TLIO ResNet18 network with a multilayer perceptron (MLP) network. They then proved that this new system performs similarly to TLIO, while its processing time is 1.9 to 12 times faster on mobile devices.

3. DATABASE GENERATION

This section describes the processes of generating and collecting the data that constitute the database. We used the open-source robotics 3D simulator *Gazebo* and the simulation package of *TurtleBot3* to emulate the mobile robot and the environments to be traversed. This package includes the physical and control model of the robot, as well as the models of the different sensors that are installed on it. The robot's model, as depicted in the simulator, is shown in Figure 1. One of the simulated sensors is the IMU, which provides, among other data, the linear acceleration (a_x, a_y, a_z) and the angular velocity $(\omega_x, \omega_y, \omega_z)$ of the robot. These values are reported at a frequency of 200 Hz and are the ones that will be stored and used to estimate the robot's location.

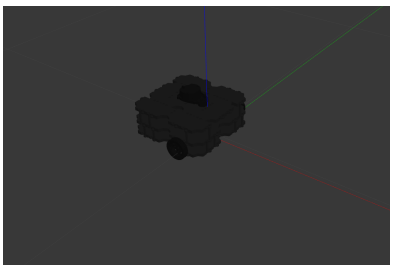

Fig. 1. Robot *TurtleBot3* used for the experiments.

The linear accelerations and angular velocities reported by the robot's IMU are represented in the local reference frame of the sensor, so the first step is to convert these measurements to the global reference frame. The robot's orientation reported by the IMU, represented by a quaternion $q = q_w + q_x i + q_y j + q_z k$, is employed to accomplish that. In practice, only the four coefficients are used to specify a quaternion of the form $q = (q_w, q_x, q_y, q_z)$. The linear accelerations can be rotated by the quaternion q using the following expression:

$$a' = q \otimes a \otimes q^{-1} \tag{1}$$

where $a = (0, a_x, a_y, a_z)$ is the local linear acceleration vector in quaternion form, a' is the rotated linear acceleration vector, $q^{-1} = (q_w, -q_x, -q_y, -q_z)$ is the conjugation of the rotation quaternion q, and $\otimes$ is the Hamilton product. a' corresponds to the linear acceleration in the global reference frame. Similarly, the angular velocities can be rotated to the global reference frame with the following expression:

$$\omega' = q \otimes \omega \otimes q^{-1} \tag{2}$$

where $\omega = (0, \omega_x, \omega_y, \omega_z)$ is the local angular velocity vector in quaternion form, and ω' is the rotated angular velocity vector.

To process the data converted to the global reference frame, we took as a basis what was done by Silva do Monte Lima et al. (2019). The total data is divided into time windows of 200 samples each (1 second of data), with a stride of 50 samples (0.25 seconds). For each of these windows, the corresponding robot's position change is calculated so that given a window of 200 samples and a jump size of 50, the change in position is calculated from sample #75 to sample #125. This is illustrated in Figure 2.

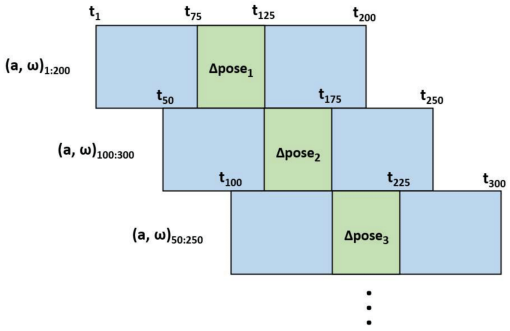

Fig. 2. Process to generate the data windows and compute the changes in the robot's position, which will be used as inputs and outputs of the model, respectively.

By processing all of the data collected during the paths, 162,995 windows or training samples were generated, each with a corresponding change in the robot's relative position.

The database contains inertial data and the ground truth position of the TurtleBot3 robot moving in the empty space and following 36 different simulated paths. 26 of them correspond to the training set and 10 to the testing set. Additionally, 12 paths in an indoor simulated environment with obstacles were included. 10 correspond to the training set and 2 to the testing set. For a more detailed description and to access the database files, we refer the reader to the following link Robot-Inertial-Localization-Dataset.

4. LOCALIZATION MODELS

Once the database was created, four different architectures were selected from the literature for implementation and comparison. A description of these architectures is presented below.

4.1 ResNet

First proposed by He et al. (2016), ResNet is a residual neural network architecture originally developed for image classification. ResNet is composed of blocks called *Basic Blocks*. They contain two 3x3 convolutional layers and a residual connection. For deeper networks, the *Bottleneck* block is used. It contains three convolutional layers of 1x1, 3x3, and 1x1, respectively, with a residual connection. For our work, these blocks are modified to process 1D data instead of 2D images by replacing the 2D convolutional

layers with 1D convolutional ones. The modified architecture is shown in Figure 3.

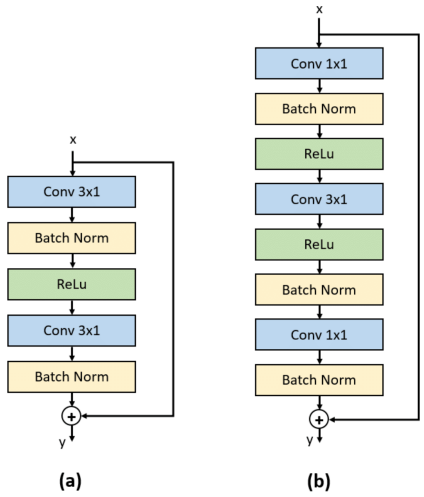

Fig. 3. Structure of the blocks (a) *Basic Block* and (b) *Bottleneck*, modified to work with 1D data.

ResNet sizes depend on the number of *Basic Blocks* or *Bottleneck* blocks used. For this paper, two of the sizes proposed in the original work are taken: ResNet-18, the smaller version of ResNet, counting eight *Basic Blocks*, and ResNet-50, a deeper version composed of 16 *Bottleneck* blocks. In addition, a smaller version than ResNet-18, consisting of four *Basic Block* blocks, is also implemented, which we called ResNet-Small. The three versions of ResNet used in the work are shown in Figure 4.

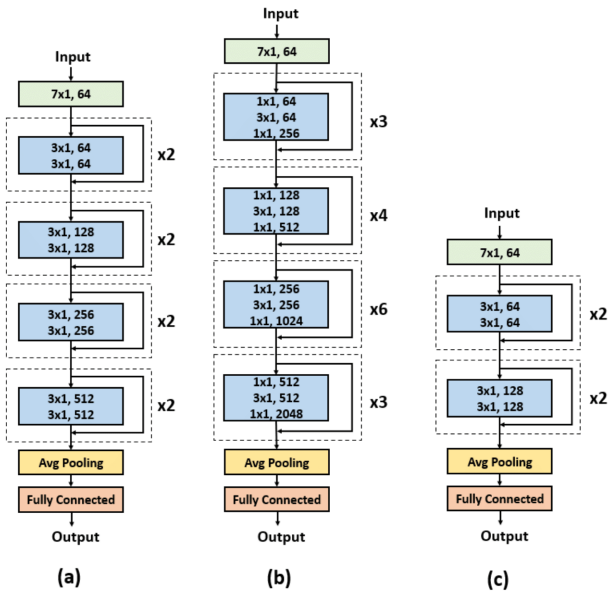

Fig. 4. Architectures of (a) ResNet-18, (b) ResNet-50 and (c) ResNet-Small.

4.2 MobileNetV3

Introduced by Howard et al. (2019), MobileNetV3 is the third generation of MobileNet, a convolutional neural network for image classification, detection, and segmentation problems, designed to run efficiently on resource-constrained mobile and embedded devices. This network has two variations, MobileNetV3-Small and MobileNetV3-Large, for low and high computational resource devices.

MobileNetV1 (Howard et al., 2017) introduced the concept of depth convolution to reduce the number of network parameters and improve performance on mobile devices. MobileNetV2 (Sandler et al., 2018) added inverted residual blocks, which have an expansion-filtering-compression structure, as opposed to regular residual blocks, which follow a compression-filtering-expansion structure. MobileNetV3 adds to the model squeeze and excitation layers (Hu et al., 2018), which help assign unequal weights to different input channels when creating feature maps, unlike a regular convolutional network that assigns equal weights. For this work, versions of MobileNetV3-Small and MobileNetV3-Large were implemented for 1-dimensional data. The general architecture of MobileNetV3 is shown in Figure 5.

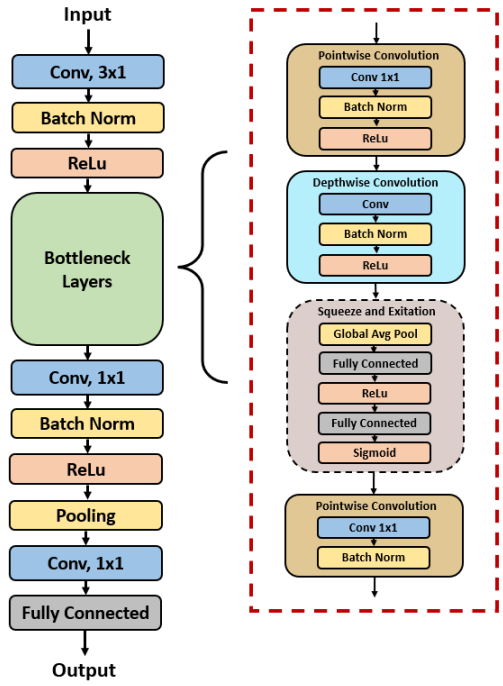

Fig. 5. General architecture of MobileNetV3. MobileNetV3-Small and MobileNetV3-Large differ in the number of *Bottleneck* layers used, with 11 and 15 layers, respectively.

4.3 CNN + LSTM Network

Silva do Monte Lima et al. (2019) proposed a neural network composed of four convolutional layers (two layers for the input linear acceleration and two for the input angular velocity) and two bidirectional LSTM layers. In the original work, this model is used to estimate the position and orientation of a mobile device in three-dimensional space. However, we are only interested in estimating the robot's position in the XY plane for this research, so we reduce the final fully connected layer from seven to two parameters.

4.4 RBCN-Net

Proposed by Zhu et al. (2023), RBCN-Net is a neural network architecture for localizing a mobile device using inertial data. They take the ResNet-18 architecture as a basis, and NAM attention modules (Liu et al., 2021) are added to each of the eight *Basic Blocks* of the network, and a CBAM attention block (Woo et al., 2018) at the beginning of the network. These attention blocks are included to learn to identify the most important features of the data. Also, two bidirectional LSTM layers are added to the network's output to learn the temporal differences between the data. Since the code of the original implementation of this architecture is not publicly available, a custom version was implemented following what is described in the article as closely as possible.

5. EXPERIMENTAL SETUP AND METRICS

In total, seven different models were trained and tested:

- ResNet-18
- ResNet-50
- ResNet-Small
- MobileNetV3-Small
- MobileNetV3-Large
- CNN + LSTM
- RBCN-Net

All these models were trained for 1000 epochs using the 162,995 training examples. A learning rate of 0.0001 was utilized, which is reduced by a factor of 0.1 if the validation loss does not improve after 20 epochs. The mean absolute error (MAE) is used as the loss function, the Adam optimizer, and a batch size of 32.

For an individual data window, the model outputs the change in the relative position of the robot during that time period. We can reconstruct the estimated robot path by adding each relative position change, assuming the starting point is known.

The paths reconstructed with the model outputs are compared to the actual paths using four different metrics: root mean square error ($RMSE$), mean positioning accuracy (RTE), and mean absolute error in X and Y (MAE_x and MAE_y).

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}[(x_i - \tilde{x}_i)^2 + (y_i - \tilde{y}_i)^2]} \tag{3}$$

$$RTE = \frac{\sum_{i=1}^{n}\sqrt{[(x_i - \tilde{x}_i)^2 + (y_i - \tilde{y}_i)^2]}}{n * path\_length} \tag{4}$$

$$MAE_x = \frac{1}{n}\sum_{i=1}^{n}|x_i - \tilde{x}_i| \tag{5}$$

$$MAE_y = \frac{1}{n}\sum_{i=1}^{n}|y_i - \tilde{y}_i| \tag{6}$$

According to these metrics, the two best models are selected and compared with two SLAM methods, *GMapping* (Grisetti et al., 2007) and *Karto*, using two test paths in an environment simulating the interior of a building (Figure 6).

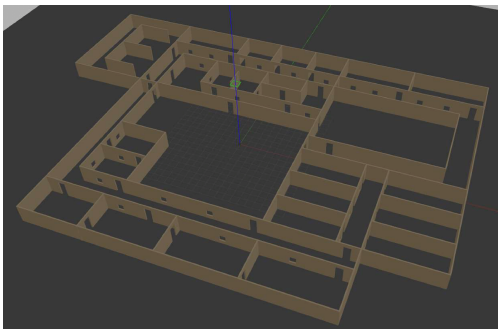

Fig. 6. Simulated environment used for comparison tests with SLAM algorithms.

6. RESULTS

This section compares the different trained models by evaluating them on different test paths. The results for the 10 paths in the testing set, considering only an empty space, are shown in Tables 1, 2, 3, 4 and 5. Table 6 presents the averages resulting from all tested paths. The green cells in the tables indicate the model performing best in each metric.

To illustrate the difference between the results of the distinct models, Figure 7 shows the paths reconstructed by the seven trained models for the test path #5.

From Table 6, it can be seen that the best results, in general, were obtained by MobileNetV3-Small, which presents the lowest average error in most of the metrics, being surpassed only by MobileNetV3-Large in one of the error metrics. MobileNetV3-Large is ranked in second place, and ResNet-18 in third place. The model that showed the worst performance was RBCN-Net, with about five times more error for all metrics than the second worst model, ResNet-50.

Based on the previous results, two trained models were selected for comparison with two SLAM algorithms: MobileNetV3-Small, which was the model that presented the best average results, and ResNet-18, which was the model of the ResNet family that showed the best performance.

As SLAM algorithms, we selected GMapping (Grisetti et al., 2007) and Karto, being the most popular and the default ones used for the robotic platform *TurtleBot3*, in addition to the fact that both make use of two different sources of information: odometry calculated by encoders on the robot's wheels, and a laser range sensor or Lidar.

The results for MobileNetV3-Small, ResNet-18, GMapping, and Karto on two different test paths considering obstacles are shown in Tables 7 and 8. The paths reconstructed by the SLAM algorithms for both test paths are shown in Figures 8 and 10, while those reconstructed by the trained models are shown in Figures 9 and 11.

As can be noticed from the previous tables and figures, the performance of the trained models in the environment with obstacles was not as good as in the test paths in

	Test path #1				Test path #2			
	RMSE	**RTE**	**MAEx**	**MAEy**	**RMSE**	**RTE**	**MAEx**	**MAEy**
ResNet-18	**0.6416**	**0.1156**	0.4277	0.3122	1.0616	0.2038	0.6930	0.6583
ResNet-50	1.5088	0.2738	1.162	0.6475	1.4269	0.2365	0.5519	0.8928
ResNet-Small	1.4648	0.2570	1.2894	0.2551	1.0456	0.1898	0.4576	0.7572
MobileNetV3-Small	0.8059	0.1402	**0.2014**	0.6795	0.8884	0.1512	0.5847	0.4437
MobileNetV3-Large	0.8916	0.1593	0.7857	**0.1968**	**0.6187**	**0.1184**	**0.3744**	**0.3715**
CNN + LSTM	1.4959	0.2592	0.931	0.8195	2.0668	0.3762	0.5243	1.7495
RBCN-Net	11.1637	2.0186	6.5044	6.8806	9.971	1.8577	4.1994	7.7963

Table 1. Results for test paths #1 and #2.

	Test path #3				Test path #4			
	RMSE	**RTE**	**MAEx**	**MAEy**	**RMSE**	**RTE**	**MAEx**	**MAEy**
ResNet-18	2.5598	0.7157	0.8580	1.7486	0.7967	0.2513	0.6453	**0.3078**
ResNet-50	4.4705	1.4035	1.3058	3.6815	1.1056	0.3563	0.7001	0.5551
ResNet-Small	3.1833	1.0376	1.0376	2.6340	**0.6510**	**0.2017**	0.4389	0.3619
MobileNetV3-Small	3.0186	0.9947	1.9684	1.7777	0.6697	0.2133	**0.2599**	0.5375
MobileNetV3-Large	**2.2497**	**0.4470**	1.1300	**0.5670**	0.7016	0.2266	0.4126	0.4997
CNN + LSTM	2.7388	0.9317	**0.8105**	2.409	0.9696	0.2934	0.7562	0.331
RBCN-Net	5.3366	1.6357	2.9768	2.9473	13.1708	4.1104	10.5325	4.9518

Table 2. Results for test paths #3 and #4.

	Test path #5				Test path #6			
	RMSE	**RTE**	**MAEx**	**MAEy**	**RMSE**	**RTE**	**MAEx**	**MAEy**
ResNet-18	**0.9943**	**0.1676**	**0.6128**	**0.4632**	0.5015	**0.1722**	0.3449	**0.2145**
ResNet-50	3.5502	0.6912	1.8084	2.8012	1.3791	0.4892	1.0649	0.6131
ResNet-Small	2.0609	0.4051	0.6733	1.7891	0.6577	0.2273	0.1917	0.5267
MobileNetV3-Small	1.1743	0.2331	0.9140	0.5714	**0.4729**	0.1761	**0.1604**	0.4194
MobileNetV3-Large	2.1416	0.4204	1.9637	0.4736	1.2467	0.4674	0.7556	0.8869
CNN + LSTM	2.9443	0.5853	1.3255	2.4366	1.8732	0.7121	0.5501	1.6823
RBCN-Net	12.2355	2.1376	4.4572	8.9675	5.0156	1.6492	2.6558	2.9395

Table 3. Results for test paths #5 and #6.

	Test path #7				Test path #8			
	RMSE	**RTE**	**MAEx**	**MAEy**	**RMSE**	**RTE**	**MAEx**	**MAEy**
ResNet-18	1.3603	0.3199	1.0573	0.7357	2.5739	0.6137	2.2411	0.7573
ResNet-50	2.2473	0.5263	1.1677	1.8315	4.6289	1.1107	3.5679	2.2722
ResNet-Small	0.8365	0.1843	0.3061	0.6749	3.684	0.8926	**0.8594**	3.3474
MobileNetV3-Small	0.6116	0.129	**0.2358**	0.4318	**1.1048**	**0.2684**	0.9618	**0.3080**
MobileNetV3-Large	**0.5006**	**0.1068**	0.3027	**0.3013**	1.4392	0.3225	1.1478	0.5043
CNN + LSTM	0.8574	0.1955	0.5782	0.4804	4.2898	1.0605	3.654	1.7334
RBCN-Net	14.5396	3.3319	12.0094	5.6612	10.8696	2.3811	4.452	7.5131

Table 4. Results for test paths #7 and #8.

	Test path #9				Test path #10			
	RMSE	**RTE**	**MAEx**	**MAEy**	**RMSE**	**RTE**	**MAEx**	**MAEy**
ResNet-18	0.906	0.2493	0.7312	**0.2637**	**0.3359**	**0.0955**	**0.1536**	0.23
ResNet-50	1.2365	0.3287	**0.2157**	1.1055	1.4244	0.3894	0.298	1.1425
ResNet-Small	1.0886	0.3056	0.2711	1.0111	0.9686	0.2547	0.3906	0.6742
MobileNetV3-Small	0.5970	0.1434	0.3865	0.2889	0.4505	0.1291	0.2401	0.2674
MobileNetV3-Large	0.5394	0.1393	0.3340	0.2958	0.7285	0.2019	0.5372	**0.1723**
CNN + LSTM	**0.4593**	**0.1218**	0.2498	0.3049	1.7452	0.4917	1.4627	0.3492
RBCN-Net	8.4454	2.2423	6.3783	3.8656	9.637	2.9076	4.6708	6.5769

Table 5. Results for test paths #9 and #10.

the empty space. For example, ResNet-18 presented a maximum RMSE of 2.5739 for all test paths in empty space, while in the environment with obstacles, the minimum RMSE was 4.0318. We conjectured that this may be because, regardless of the type of environment, the only data used is inertial information; in an environment with obstacles, the robot's movements tend to be more abrupt, presenting more frequent speed changes to avoid the obstacles, something that was probably not present in the data used for training.

Therefore, to improve the performance of the models, a second environment with obstacles was created, in which 10 different paths were executed to collect data and increase the training database, seeking to include the abrupt speed changes mentioned above. This environment is shown in Figure 12. As a result of this process, 66,319 new training examples were generated, which, added to the existing training data in the empty space, resulted in a total of 229,314 training examples.

	Total averages			
	RMSE	**RTE**	**MAEx**	**MAEy**
ResNet-18	1.1732	0.2905	0.7765	0.5691
ResNet-50	2.2978	0.5806	1.1842	1.5543
ResNet-Small	1.5641	0.3956	0.5916	1.2032
MobileNetV3-Small	**0.9794**	**0.2579**	**0.5913**	0.5725
MobileNetV3-Large	1.1058	0.2610	0.7744	**0.4269**
CNN+LSTM	1.9440	0.5027	1.0842	1.2296
RBCN-Net	10.0385	2.4272	5.8837	5.8100

Table 6. Average results for all tested paths.

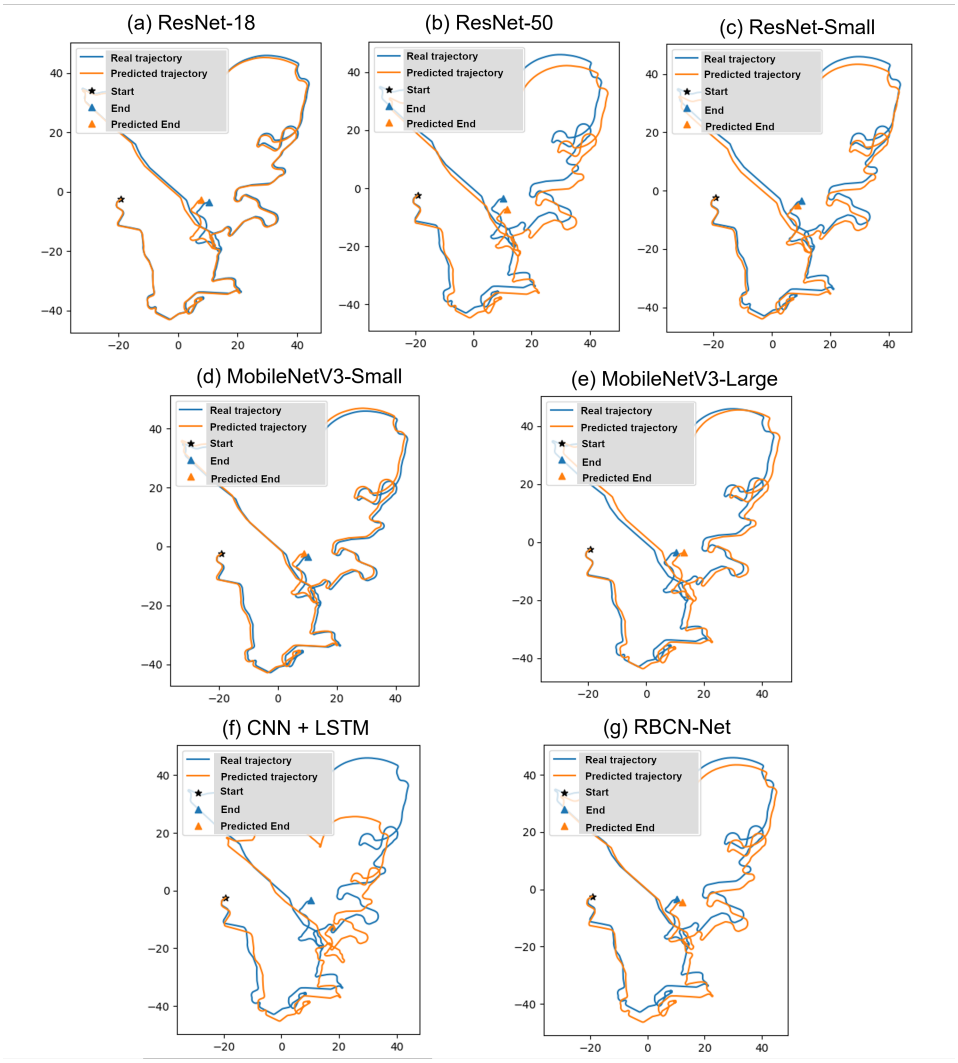

Fig. 7. Reconstructed paths for all models for the test path #5.

	Test SLAM #1				Test SLAM #2			
	RMSE	**RTE**	**MAEx**	**MAEy**	**RMSE**	**RTE**	**MAEx**	**MAEy**
ResNet-18	4.0318	1.8263	**0.5514**	3.5839	11.5302	4.0571	5.0153	7.4983
MobileNetV3-Small	3.0658	1.3117	0.6668	2.5114	7.1285	2.4943	2.5105	4.8837
GMapping	7.7348	3.5691	6.2504	2.6399	5.0749	1.9010	3.6389	1.4521
Karto	**1.3999**	**0.6139**	0.8695	**0.7986**	**1.2305**	**0.4071**	**0.7510**	**0.4479**

Table 7. Results for comparison with SLAM algorithms in test paths #1 and #2.

With this data, the models compared to the SLAM methods were retrained, and tests were performed on the two test paths with obstacles. The results of these new models are shown in Table 8, and the paths reconstructed by them in Figures 13 and 14.

From the table and figures mentioned in the previous paragraph, it can be concluded that training the models again with additional data generated in an environment with obstacles improved their performance considerably, outperforming both SLAM methods.

	Test SLAM #1				Test SLAM #2			
	RMSE	**RTE**	**MAEx**	**MAEy**	**RMSE**	**RTE**	**MAEx**	**MAEy**
ResNet-18	**0.3794**	**0.1743**	**0.2063**	0.2611	**0.5347**	**0.1883**	**0.2279**	0.3251
MobileNetV3-Small	0.5389	0.2367	0.3755	**0.2254**	0.7691	0.2966	0.5934	**0.2233**

Table 8. Results for comparison with improved models in paths #1 and #2

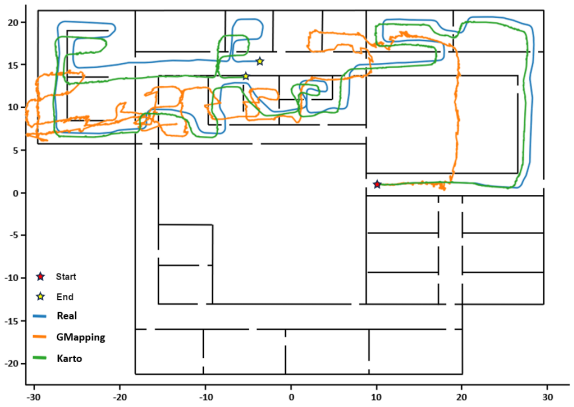

Fig. 8. Test path #1, real and estimated paths by SLAM algorithms.

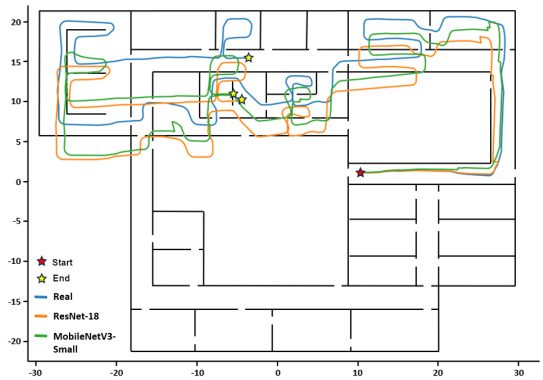

Fig. 9. Test path #1, real and estimated paths by trained models.

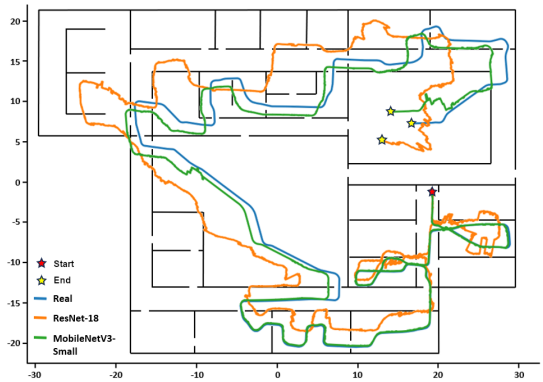

Fig. 10. Test path #2, real and estimated paths by SLAM algorithms

For ResNet-18, there was a 90.5898% reduction in RMS error for the first path and 95.3626% for the second path. For MobileNetV3-Small, the RMS error was reduced by 82.4222% for the first path and 89.2109% for the second path. ResNet-18 performed best in all metrics, except for

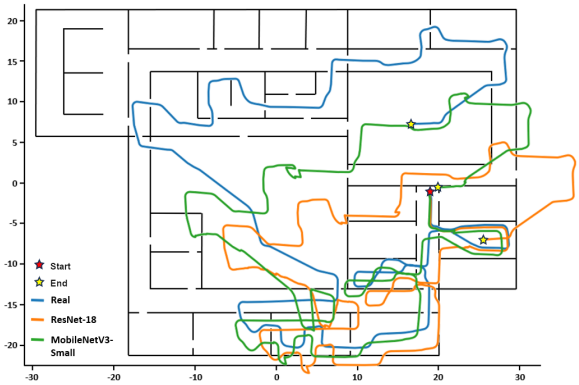

Fig. 11. Test path #2, real and estimated paths by trained models.

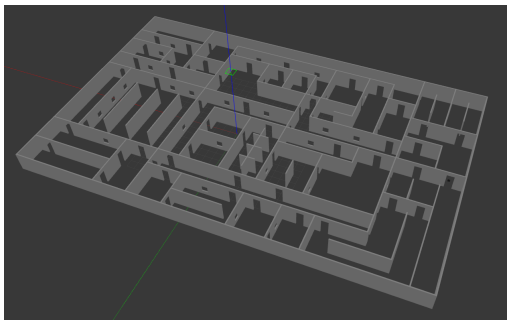

Fig. 12. Environment with obstacles used to generate additional training data.

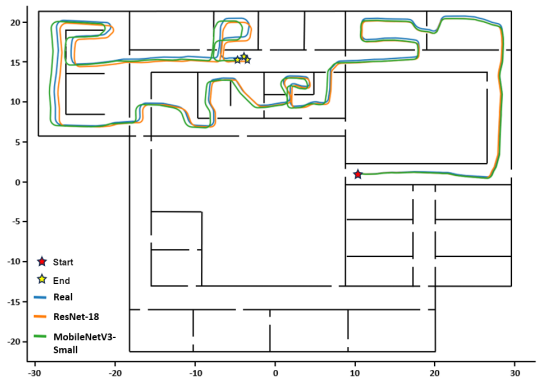

Fig. 13. Test path #1, real and estimated paths by models trained with additional data.

the absolute error in Y (MAEy), where MobileNetV3-Small showed better results.

Compared with the SLAM methods, Karto initially showed better performance than the two models tested in both paths. After training the models again, ResNet-18 presented an RMS error 72.8980% lower than that of Karto for test path #1 and 56.5461% lower for test path #2. On the other hand, MobileNetV3-Small showed an RMS error

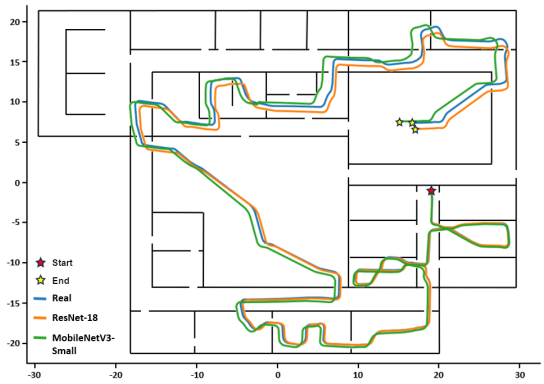

Fig. 14. Test path #2, real and estimated paths by models trained with additional data.

61.5043% lower than that of Karto for test path #1, and 37.4969% lower for test path #2.

6.1 Discussion

During the tests, a minimum RMS error of 0.38 was achieved for the obstacle experiments using ResNet-18, showing better performance than the two SLAM algorithms evaluated on the same paths. This is a remarkable outcome, given that the results obtained with the trained models use data from a single sensor, the IMU mounted on the robot. In contrast, the SLAM algorithms use at least two different types of sensors: encoders and laser range sensors. However, more modern SLAM algorithms use cameras, which can be more accurate than the ones tested in this work.

It is important to note that this result was obtained after training the models with additional data captured in a simulated environment with obstacles, even though one would think that the accelerations and angular velocities of the robot would behave the same way in an environment with or without obstacles. This highlights the importance of capturing the data for training the model in scenarios as close as possible to those the robot will face during normal operation.

Using only the robot's inertial data has the advantage over other types of data that it applies to any type of environment, whereas algorithms using cameras, for example, may fail when there is inadequate lighting, or those using laser range sensors may fail when encountering large open spaces or transparent obstacles.

The localization models trained in this work apply only to the robot used during the research. If they were to be used with another type of robot, they should be trained with inertial data captured from that specific robot since each robot has different motion, acceleration, and deceleration characteristics. An interesting research problem is to evaluate if the analyzed architectures have similar performance with other types of robots, such as car-like robots.

Another important point is that all experiments and results were obtained in simulation. In a real environment, different factors that were not taken into account in the simulation can influence the localization error, for example, vibrations due to the robot's motors or the texture of the ground on which it moves, and the effects that these would have on the acceleration data and the localization estimation.

7. CONCLUSION AND FUTURE WORK

This work explored the use of robot inertial data to solve the indoor mobile robot localization problem. Seven models based on deep neural networks were trained to estimate the robot's localization by taking its inertial data as input. Of all the trained models, ResNet-18 and MobileNetV3-Small were selected for comparison with two SLAM algorithms, GMapping and Karto. After training both models with additional data generated in an environment with obstacles, they showed better results than the SLAM algorithms evaluated.

In general, using only inertial data to estimate the location of a mobile robot in an indoor environment proved to be a good solution, showing better results than the SLAM algorithms included as part of the TurtleBot3 robotic platform.

Since all the experiments presented in this work were performed entirely in simulation, as future work, we would seek to implement the models and methods presented here, and any improvements that could be developed, in a real robot and real environments, comparing the results obtained with those shown in this work for experiments in simulation. One important issue that we need to overcome in real environments is how to obtain the ground truth information in large locations, which is required to train the models since, in this case, the use of motion caption systems that can estimate the robot's position is not feasible and GPS-based devices are known to not work well in indoor conditions. Thus, alternative approaches must be devised to capture this information.

In addition to the deep learning architectures shown here, other types of neural networks could be analyzed, for example, transformer networks initially used for natural language processing. New variants of these networks can process data from time series, such as those provided by inertial measurement units.

Another aspect not studied in this research is the effect of the robot's speed on the localization error. For that, another type of robot that can move at a higher speed could be used since the robot utilized in this work has a maximum speed of only 0.26 m/s.

One of the ways to improve localization estimation using inertial data is to perform fusion with other types of sensors. In general, it was observed that the error in the estimated position increases as the robot traverses the environment and progresses over time. This error could be corrected with the help of a more accurate sensor, which typically has a much lower sampling rate than the inertial sensor. A more accurate sensor can be a camera or a laser range sensor, and even if these fail in specific situations, the location estimation using only the inertial data is good enough to ignore the data from these sensors in the scenarios where they tend to fail.

REFERENCES

Asraf, O., Shama, F., and Klein, I. (2021). Pdrnet: A deep-learning pedestrian dead reckoning framework. *IEEE Sensors Journal*, 22(6), 4932–4939.

Brossard, M. and Bonnabel, S. (2019). Learning wheel odometry and imu errors for localization. In *2019 International Conference on Robotics and Automation (ICRA)*, 291–297. IEEE.

Campos, C., Elvira, R., Rodríguez, J.J.G., Montiel, J.M., and Tardós, J.D. (2021). Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. *IEEE Transactions on Robotics*, 37(6), 1874–1890.

Chen, C. (2023). Deep learning for inertial positioning: A survey. *arXiv preprint arXiv:2303.03757*.

Chen, C., Lu, X., Markham, A., and Trigoni, N. (2018). Ionet: Learning to cure the curse of drift in inertial odometry. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Dobrev, Y., Gulden, P., and Vossiek, M. (2018). An indoor positioning system based on wireless range and angle measurements assisted by multi-modal sensor fusion for service robot applications. *IEEE Access*, 6, 69036–69052.

Engelhard, N., Endres, F., Hess, J., Sturm, J., and Burgard, W. (2011). Real-time 3d visual slam with a hand-held rgb-d camera. In *Proc. of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum, Vasteras, Sweden*, volume 180, 1–15.

Esfahani, M.A., Wang, H., Wu, K., and Yuan, S. (2019). Aboldeepio: A novel deep inertial odometry network for autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 21(5), 1941–1950.

Grisetti, G., Stachniss, C., and Burgard, W. (2007). Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1), 34–46.

Hashemifar, Z.S., Adhivarahan, C., Balakrishnan, A., and Dantu, K. (2019). Augmenting visual slam with wi-fi sensing for indoor applications. *Autonomous Robots*, 43, 2245–2260.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Herath, S., Irandoust, S., Chen, B., Qian, Y., Kim, P., and Furukawa, Y. (2021). Fusion-dhl: Wifi, imu, and floorplan fusion for dense history of locations in indoor environments. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 5677–5683. IEEE.

Howard, A., Sandler, M., Chu, G., Chen, L.C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., et al. (2019). Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, 1314–1324.

Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Hu, J., Shen, L., and Sun, G. (2018). Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 7132–7141.

Labbe, M. and Michaud, F. (2013). Appearance-based loop closure detection for online large-scale and long-term operation. *IEEE Transactions on Robotics*, 29(3), 734–745.

Li, C., Wang, S., Zhuang, Y., and Yan, F. (2019). Deep sensor fusion between 2d laser scanner and imu for mobile robot localization. *IEEE Sensors Journal*, 21(6), 8501–8509.

Liu, W., Caruso, D., Ilg, E., Dong, J., Mourikis, A.I., Daniilidis, K., Kumar, V., and Engel, J. (2020). Tlio: Tight learned inertial odometry. *IEEE Robotics and Automation Letters*, 5(4), 5653–5660.

Liu, Y., Shao, Z., Teng, Y., and Hoffmann, N. (2021). Nam: Normalization-based attention module. *arXiv preprint arXiv:2111.12419*.

Pfeiffer, M., Shukla, S., Turchetta, M., Cadena, C., Krause, A., Siegwart, R., and Nieto, J. (2018). Reinforced imitation: Sample efficient deep reinforcement learning for mapless navigation by leveraging prior demonstrations. *IEEE Robotics and Automation Letters*, 3(4), 4423–4430.

Roy, P. and Chowdhury, C. (2021). A survey of machine learning techniques for indoor localization and navigation systems. *Journal of Intelligent & Robotic Systems*, 101(3), 63.

Ruan, X., Ren, D., Zhu, X., and Huang, J. (2019). Mobile robot navigation based on deep reinforcement learning. In *2019 Chinese control and decision conference (CCDC)*, 6174–6178. IEEE.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4510–4520.

Silva do Monte Lima, J.P., Uchiyama, H., and Taniguchi, R.i. (2019). End-to-end learning framework for imu-based 6-dof odometry. *Sensors*, 19(17), 3777.

Thrun, S. and Montemerlo, M. (2006). The graph slam algorithm with applications to large-scale mapping of urban structures. *The International Journal of Robotics Research*, 25(5-6), 403–429.

Wang, Y., Kuang, J., Niu, X., and Liu, J. (2022). Llio: Lightweight learned inertial odometer. *IEEE Internet of Things Journal*, 10(3), 2508–2518.

Wen, S., Zhao, Y., Yuan, X., Wang, Z., Zhang, D., and Manfredi, L. (2020). Path planning for active slam based on deep reinforcement learning under unknown environments. *Intelligent Service Robotics*, 13, 263–272.

Woo, S., Park, J., Lee, J.Y., and Kweon, I.S. (2018). Cbam: Convolutional block attention module. In *Proceedings of the European conference on computer vision (ECCV)*, 3–19.

Xiao, Y., Ou, Y., and Feng, W. (2017). Localization of indoor robot based on particle filter with ekf proposal distribution. In *2017 IEEE international conference on cybernetics and intelligent systems (CIS) and IEEE conference on robotics, automation and mechatronics (RAM)*, 568–571. IEEE.

Zhu, Y., Zhang, J., Zhu, Y., Zhang, B., and Ma, W. (2023). Rbcn-net: A data-driven inertial navigation algorithm for pedestrians. *Applied Sciences*, 13(5), 2969.