Using Deep Learning for 2D Primitive Perception with a Noisy Robotic LiDAR

Conference Paper · November 2024

DOI: 10.1109/ROBOT61475.2024.10796946

6 authors, including:

Gonçalo Leão
University of Porto
13 PUBLICATIONS **54** CITATIONS
SEE PROFILE

Luís Paulo Reis
University of Porto
602 PUBLICATIONS **5,022** CITATIONS
SEE PROFILE

Armando Jorge Sousa
University of Porto
158 PUBLICATIONS **1,071** CITATIONS
SEE PROFILE

This is a pre-print of a contribution published in "Robot 2024: Seventh Iberian Robotics Conference" published by IEEE.

The final authenticated version is avalaible online at: https://doi.org/10.1109/ROBOT61475.2024.10796946

Using Deep Learning for 2D Primitive Perception with a Noisy Robotic LiDAR

Antónia Brito[1,2,4], Pedro Sousa[1,2], Ana Couto[1,2], Gonçalo Leão[1,3], Luís Paulo Reis[1,4], Armando Sousa[1,3]
[1] *FEUP - Faculty of Engineering, University of Porto, Portugal*
[2] *FCUP - Faculty of Sciences, University of Porto, Portugal*
[3] *INESC TEC - INESC Technology and Science, Portugal*
[4] *LIACC - Artificial Intelligence and Computer Science Laboratory, Portugal*
{up202107271}@edu.fe.up.pt,{up202108383, up200802466}@edu.fc.up.pt,{goncalo.leao,lpreis,asousa}@fe.up.pt

***Abstract*—Effective navigation in mobile robotics relies on precise environmental mapping, including the detection of complex objects as geometric primitives. This work introduces a deep learning model that determines the pose, type, and dimensions of 2D primitives using a mobile robot equipped with a noisy LiDAR sensor. Simulated experiments conducted in Webots involved randomly placed primitives, with the robot capturing point clouds which were used to progressively build a map of the environment. Two mapping techniques were considered, a deterministic and probabilistic (Bayesian) mapping, and different levels of noise for the LiDAR were compared. The maps were used as input to a YOLOv5 network that detected the position and type of the primitives. A cropped image of each primitive was then fed to a Convolutional Neural Network (CNN) that determined the dimensions and orientation of a given primitive. Results show that the primitive classification achieved an accuracy of 95% in low noise, dropping to 85% under higher noise conditions, while the prediction of the shapes' dimensions had error rates from 5% to 12%, as the noise increased. The probabilistic mapping approach improved accuracy by 10-15% compared to deterministic methods, showcasing robustness to noise levels up to 0.1. Therefore, these findings highlight the effectiveness of probabilistic mapping in enhancing detection accuracy for mobile robot perception in noisy environments.**

***Index Terms*—Convolutional Neural Network (CNN), Deep Learning, LiDAR, Mapping, Pose Estimation, Simulation, You Only Look Once (YOLO)**

I. Introduction

Mobile robotics rely on accurate environmental mapping to navigate efficiently, often using geometric primitives to describe objects. These primitives assist in mapping, object recognition, and scene understanding [1]. LiDAR sensors, which emit laser pulses to measure distances, generate point clouds that represent objects in a robot's surroundings. However, point clouds are computationally intensive to process in real-time, especially when resources are limited. Moreover, environmental factors like rain, fog, and dust introduce noise, further complicating the mapping process [2].

Recognizing and mapping geometric primitives is crucial for improving robotic navigation and decision-making, with real-world applications such as warehouse automation [3] and autonomous driving [4], [5].

This work aims to develop a machine learning model to recognize and determine the pose of 2D primitives within a robot's surroundings using LiDAR sensors in the Webots simulator. YOLOv5 [6], a state-of-the-art CNN-based model for object classification, was employed to identify primitive types, while custom CNNs were trained to estimate parameters like width and radius. Both models were trained on noisy and clean point clouds to improve robustness against imperfect LiDAR data.

A major challenge addressed in this research is managing noise in LiDAR data, which hinders geometric primitive recognition. The proposed deep learning approach contributes to the development of more robustness recognition techniques, ensuring reliable robot navigation in uncertain environments.

The paper is structured as follows: Section II reviews related work on shape recognition and mapping in robotics, Section III explains the methodology, including data collection and model evaluation, Section IV presents the results, and Section V summarizes the findings and outlines future research directions.

II. Related work

Shape recognition and mapping in mobile robotics have been central problems in computer vision research. Recent deep learning methods have proven highly effective for geometric shape classification in LiDAR point clouds due to their ability to process large-scale data, learn complex features and handle noise. Various approaches for point cloud classification have been developed, including graph-based, attention-based, and convolution-based methods [7]. Diab *et al.* [8] reviewed state-of-the-art deep learning models, highlighting CNN-based methods such as Dynamic Graph CNN and ConvPoint as particularly effective and lightweight for point cloud processing.

Point-by-point methods are also widely used for point cloud classification. A detailed survey on these methods is provided by Zhang *et al.* [7]. This work builds on this by improving generalization with multi-level noisy point cloud LiDAR scans.

PointNet [9] was an early breakthrough in point cloud methods, using a T-Net module to transform the input and capture global features through maximum pooling. However, its inability to capture relationships between points affects

fine-grained classification. PointNet++ addressed this by hierarchically processing point clouds with layers for sampling, grouping, and feature extraction [10]. PointNeXt improved on PointNet++ by incorporating separable MLPs and an inverted residual bottleneck for better performance [11].

GDANet [12] introduced the Geometry-Disentangle module, which decomposes point clouds into contour and plane components, enhancing local information representation. This method improved 3D semantic understanding, demonstrating strong performance and robustness in point cloud classification.

III. Methodological Approach

This work's pipeline consists of two main blocks: the data collection and dataset generation, detailed Sections in III-A and III-B, and the final predictive pipeline, described in Section III-C. Figure 1 illustrates the proposed methodological approach.

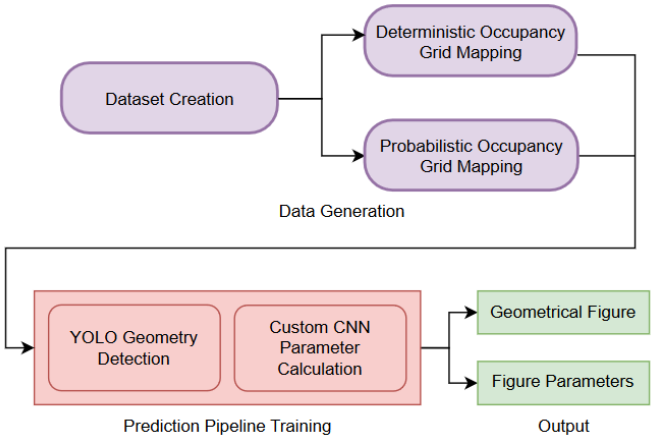


Fig. 1: Flowchart with the methodological approach

A. Occupancy Grid Mapping

The neural networks for primitive detection take occupancy grid maps as input, where space is divided into cells representing whether each is occupied by an obstacle or not. These binary images use black for occupied and white for free spaces. To create the grids, a robot equipped with LiDAR scans the environment from multiple locations, generating a point cloud that updates the grid. Two mapping approaches were used: deterministic and probabilistic.

On the one hand, with deterministic mapping, Bresenham's algorithm [13] is used to mark cells intersected by each LiDAR ray. If a ray reaches an obstacle, then the corresponding cell is marked as occupied, while the cells leading up to the obstacle, in the robot's direction, are marked as free. This method is sensitive to sensor noise, which may lead to misclassified cells.

On the other hand, probabilistic mapping accounts for sensor noise [14]. Each cell contains the log-odds l of the probability p of it being occupied, with equations 1 and 2 relating l and p, where $ln(x)$ is the natural logarithm of x.

$$l = ln(\frac{p}{1-p}) \tag{1}$$

$$p = 1 - \frac{1}{1 + exp(l)} \tag{2}$$

Initially, all cells have a log-odd of 0 (i.e. the log-odd of 0.5) since there is no information about the cell's state. If one of the rays from the t-th LiDAR scan intersects a cell i, its log-odd is updated according to equation 3 [14], based on Bayes' theorem, where $p(o_i|s_{1:t})$ represents the conditional probability that cell i is occupied, given the sensor data from the first scan to the t-th scan.

$$p(o_i|s_{1:t}) = l(o_i|s_t) - l(o_i|s_{1:t-1}) + l(o_i) \tag{3}$$

In the previous equation:

- $l(o_i|s_t)$ is the value given by the sensor's inverse sensor model, a function $p(d)$ that indicates the probability p that a given point of the map aligned with a LiDAR ray is occupied, where d is the signed difference (in m) between two distances: the distance from the robot to a cell's real (x,y) coordinate and the LiDAR's measured distance. The function used in this work, in equation 4, is a piecewise linear function defined empirically by trial-and-error.
- $l(o_i|s_{1:t-1})$ is the cell's previous log-odd value.
- $l(o_i)$ reflects the cell's default (apriori) probability of being occupied, and is thus set to 0.

$$p(d) = \begin{cases} 0.3 & d \leq -0.03 \\ 20d + 0.9 & -0.03 \leq d \leq 0 \\ -13.333x + 0.9 & 0 \leq d \leq 0.03 \\ 0.5 & d \geq 0.03 \end{cases} \tag{4}$$

After all scans are captured, the probability of each cell is rounded to either 0 or 1 using a maximum likelihood approach.

B. Dataset generation

Synthetic maps are created with images of various shapes—squares, rectangles, circles, triangles, and ellipses—varying in size and rotation (in mm and degrees, respectively). The shapes have specific parameter ranges: squares (side length with 80-300 mm, 0-360 degrees), rectangles (width and height with 60-320 mm, 0-360 degrees), circles (radius with 50-300 mm), triangles (side lengths with 100-300 mm, 0-360 degrees), and ellipses (both axes with 40-320 mm, 0-360 degrees). These maps are scanned to produce occupancy grids using both mapping algorithms. Different LiDAR noise levels are incorporated to evaluate model robustness. The final grids are converted into images for model training and testing.

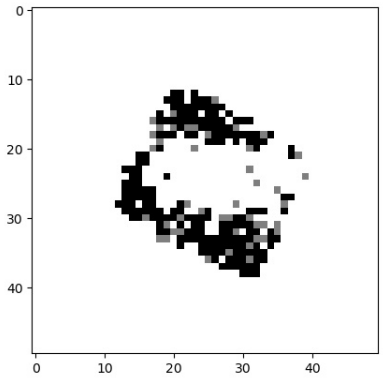
C. Predictive Pipeline

To be able to make the final prediction, two different models were used: YOLO, used in all images, and our custom CNN, used in each detected primitive.

Firstly, YOLO is used to predict the location and shape/class of each 2D primitive in an image. YOLO is a state-of-the-art, real-time object detection algorithm that is very fast and accurate [6]. In this framework, the proposed YOLO model is trained on top of the pre-trained YOLOv5[1].

Afterwards, each primitive parameter (such as a circle's radius) is determined using a CNN. To tackle this task, each figure detected by YOLO was isolated through a bounding box and its position was standardized. These new individualized, standardized figures (as exemplified in Figure 2) were then used to generate a new dataset to train the CNN. All images were thoroughly annotated with the corresponding parameters.

It should be noted that YOLO is a CNN-based model designed for object detection, whilst the custom CNNs are specifically tailored for primitive parameter estimation. Both are based on the network, and they serve distinct but complementary purposes within this workflow.



figure_id	0
filename	map0.png
class	1
angle	333.0
width	227.0
height	196.0
radius	NaN
xmin	572
ymin	1219
xmax	853
ymax	1514

Fig. 2: Example of a figure and its corresponding parameters

Figure 3 depicts the architecture that is common to all five CNNs. The network includes three convolutional blocks, each one followed by batch normalization and max-pooling. The extracted features are then flattened and passed through a series of fully connected layers with decreasing numbers of neurons, all with ReLU activation functions. The output layer is a dense fully connected layer with variable output neurons matching the number of parameters each primitive shape requires: 1 for circles (radius), 2 for squares, 3 for rectangles (angle, height and width), 3 for triangles and 3 for ellipses (angle, semi-major axis, semi-minor axis).

Lastly, to optimize the CNN training, the size of the images containing an isolated primitive was resized from 200 x 200 pixels to 50 x 50 pixels. In addition, to remove ambiguities in the data labels, the rotation angle was adjusted, for squares between 0 and 90º and for rectangles and ellipses between 0 and 180º. This prevents the network from being punished during training, for instance, when (correctly) predicting a 95ºs angle for squares when it is labeled as having a rotation of 5º.

[1]https://github.com/ultralytics/yolov5

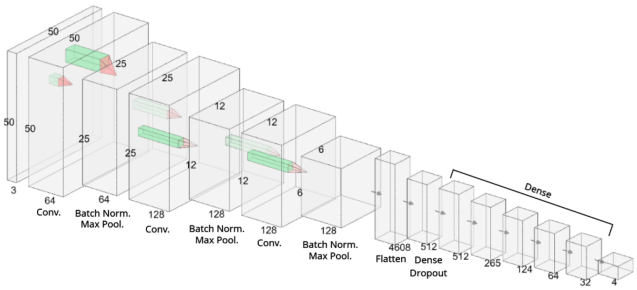


Fig. 3: Architecture of the CNNs designed to predict the primitives' parameters

IV. Experimental Setup and Evaluation

A. Simulator Setup

Webots[2] [15], an open-source robot simulator, was used to collect data and train/test the models. New 2000x2000 mm maps, featuring various primitive shapes in different sizes and orientations, were generated for the simulations. An e-puck robot equipped with LiDAR, GPS, and a compass was used to scan the maps. The LiDAR sensor had 500 rays, a field of view of 6.28 rad, and a range of 32 to 5000 mm. Additionally, different noise levels (0, 0.05, 0.075, and 0.1) were added to the LiDAR scans to test the models' robustness. The added noise followed a Gaussian distribution, in which a value of 1.0 represents noise with a standard deviation of 5000 mm. Using Webots' Supervisor functionality to warp the robot around, it scanned each map from 10x10 locations, collecting point clouds and creating 200x200 pixel images of the environment. A generated map and the robot are depicted in figure 4.

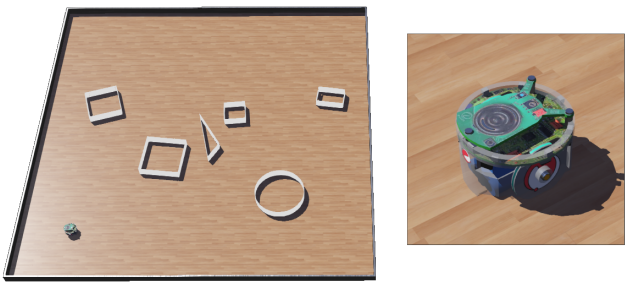


Fig. 4: Visualization of a simulated test scenario (left), with a close-up view of the robot (right)

B. Model Training

Three datasets, each containing 500 images with 0 to 2 figures per shape/class, were used for training. Two models were trained using a deterministic mapping approach with noise levels of 0 and 0.05. A single model was trained with a probabilistic mapping algorithm at a noise level of 0.05. Training was conducted on an Intel(R) Core(TM) i7-9750HF CPU at 2.60GHz, and without the aid of a GPU.

[2]https://cyberbotics.com

1) YOLO Training: For each dataset, a new network was trained. All YOLO models were built upon the existing YOLOv5 model for 32 epochs, using a batch size of 32, and employed Stochastic Gradient Descent with a learning rate of 0.01.

2) CNN Training: Following the standardization of individual primitives as described in Section III-C, the datasets were prepared for CNN training. Five different CNNs were trained per dataset, one for each primitive shape, totaling 15 CNN models. All models were tested for 20 epochs with a batch size of 128, utilizing the Adam optimizer with a learning rate of 0.001.

C. Model Testing

To evaluate the performance of deterministic and probabilistic mapping in object detection, experiments were conducted with various shapes and noise levels. The primary objective was to determine which mapping method—probabilistic or deterministic—provides better predictive performance and robustness.

The experiments tested YOLO and CNN models on datasets containing specific primitives or a mixture of geometric shapes, under different noise conditions labeled as n=0 (no noise), n=0.05, n=0.075, and n=0.1.

D. Results for the shape detection YOLO

Table I presents the predictive performance of the YOLO network, including metrics such as F1 score, Mean Average Precision (mAP) at IOU 0.5, mAP at IOU 0.5-0.95, Area Under Curve (AUC), and inference time (labeled as 'Inf. time', in seconds).

1) Shape-wise Analysis:

- Square: Probabilistic mapping consistently yielded higher F1 scores, when testing with some noise.
- Rectangle: Higher mAP and F1 scores were observed with probabilistic mapping.
- Triangle: Significant improvements in mAP at IOU 0.5 were noted for models trained on probabilistic maps.
- Circle: The probabilistic approach excelled in circle detection, achieving high mAP at both IOU 0.5 and IOU 0.5-0.95.
- Ellipse: Performance improvement was less pronounced, but probabilistic mapping still showed better or comparable results, particularly when comparing both mapping techniques when training and testing with 0.05 noise.

2) Overall Performance: Models trained and tested on noise-free data (n=0) performed well under the same conditions but dropped significantly with the introduction of even some slight noise (Table I, rows 1-2).

Probabilistic mapping consistently outperformed deterministic mapping across different shapes and noise levels, when deterministic mapping was tested with some noise. For instance, the F1 score of the YOLO model trained with n=0.05 and tested with n=0.1 using probabilistic mapping (Table I, row 8) was 1.6 times better than its deterministic counterpart (row 5).

TABLE I: Performance of YOLOs trained on 500 samples and tested with 125 samples

Map set	Shape	F1 score	mAP at IOU 0.5	mAP at IOU 0.5-0.95	AUC	Inf. time (in s)
Train n=0 Test n=0 det. map	All	0.800	0.821	0.554	0.500	1.181
	Square	0.692	0.653	0.467	0.653	0.224
	Rectangle	0.631	0.636	0.384	0.636	0.239
	Triangle	0.906	0.976	0.599	0.676	0.228
	Circle	0.901	0.946	0.766	0.946	0.236
	Ellipse	0.800	0.896	0.554	0.896	0.251
Train n=0 Test n=0.05 det. map	All	0.168	0.080	0.029	0.079	1.878
	Square	0.166	0.117	0.040	0.117	0.416
	Rectangle	0.140	0.072	0.024	0.077	0.393
	Triangle	0.094	0.089	0.024	0.089	0.429
	Circle	0.145	0.107	0.053	0.107	0.337
	Ellipse	0.017	0.013	0.004	0.113	0.304
Train n=0.05 Test n=0.05 det. map	All	0.475	0.454	0.236	0.500	0.432
	Square	0.437	0.462	0.287	0.462	0.106
	Rectangle	0.411	0.367	0.183	0.367	0.100
	Triangle	0.511	0.453	0.154	0.453	0.110
	Circle	0.617	0.749	0.472	0.750	0.086
	Ellipse	0.182	0.237	0.083	0.237	0.078
Train n=0.05 Test n=0.075 det. map	All	0.323	0.230	0.095	0.230	0.504
	Square	0.368	0.312	0.146	0.312	0.112
	Rectangle	0.250	0.159	0.057	0.159	0.106
	Triangle	0.244	0.140	0.031	0.140	0.115
	Circle	0.503	0.466	0.219	0.466	0.090
	Ellipse	0.117	0.075	0.021	0.075	0.082
Train n=0.05 Test n=0.1 det. map	All	0.226	0.156	0.059	0.156	0.578
	Square	0.332	0.250	0.106	0.250	0.128
	Rectangle	0.206	0.130	0.106	0.130	0.121
	Triangle	0.105	0.060	0.015	0.060	0.132
	Circle	0.270	0.305	0.012	0.305	0.104
	Ellipse	0.069	0.033	0.011	0.033	0.093
Train n=0.05 Test n=0.05 prob. map	All	0.657	0.664	0.446	0.312	0.681
	Square	0.610	0.599	0.430	0.599	0.142
	Rectangle	0.608	0.550	0.350	0.550	0.153
	Triangle	0.817	0.899	0.52	0.899	0.132
	Circle	0.787	0.919	0.732	0.919	0.147
	Ellipse	0.433	0.352	0.196	0.352	0.107
Train n=0.05 Test n=0.075 prob. map	All	0.545	0.572	0.342	0.572	0.248
	Square	0.563	0.572	0.359	0.575	0.055
	Rectangle	0.465	0.429	0.240	0.429	0.052
	Triangle	0.661	0.746	0.341	0.746	0.057
	Circle	0.669	0.797	0.612	0.797	0.044
	Ellipse	0.299	0.324	0.154	0.314	0.040
Train n=0.05 Test n=0.1 prob. map	All	0.331	0.312	0.148	0.312	0.225
	Squares	0.439	0.353	0.154	0.353	0.050
	Rectangle	0.338	0.257	0.094	0.257	0.047
	Triangle	0.210	0.192	0.052	0.192	0.051
	Circle	0.526	0.632	0.397	0.632	0.040
	Ellipse	0.116	0.125	0.045	0.125	0.036

E. Results for the parameter estimation CNNs

Table II depicts the predictive performance of the CNNs, with the following metrics:

- Average angle error (in degrees);
- Average width and height errors (in millimeters); This width and height error has different meanings depending on the shape, being the base width and height for triangles, the semi-major axis and the semi-minor axis for ellipses, and the radius on both for circles, respectively.
- Average inference time (in seconds).

The averages are calculated through the following formula:

$$\text{average} = \frac{1}{N}\sum_{i=1}^{N} |\text{true}_i - \text{predicted}_i|$$

with N being the quantity of test samples.

TABLE II: Performance of CNNs trained on 500 samples and tested with 125 samples

Map set	Shape	Avg. angle error	Avg. width error	Avg. height error	Inf. time (in s)
Train n=0 Test n=0 det. map	All	69.82	88.11	114.79	183.30
	Square	35.25	147.26	142.26	34.86
	Rectangle	72.00	111.02	183.14	37.20
	Triangle	104.59	26.10	21.47	35.44
	Circle	–	40.19	40.19	36.76
	Ellipse	74.06	49.95	95.24	39.02
Train n=0 Test n=0.05 det. map	All	67.34	111.70	132.07	40.60
	Square	31.16	148.67	145.65	8.99
	Rectangle	66.13	105.13	162.74	8.49
	Triangle	98.53	113.23	98.54	9.27
	Circle	–	49.34	49.34	7.27
	Ellipse	74.37	67.40	121.20	6.56
Train n=0.05 Test n=0.05 det. map	All	54.49	193.58	200.24	172.60
	Square	28.82	131.52	131.21	42.35
	Rectangle	50.09	58.27	127.28	39.95
	Triangle	99.85	560.25	481.32	43.94
	Circle	–	64.17	64.17	34.36
	Ellipse	45.06	53.54	82.74	31.16
Train n=0.05 Test n=0.075 det. map	All	55.32	224.64	225.08	49.50
	Square	27.83	131.63	131.62	10.96
	Rectangle	48.79	62.60	123.90	10.35
	Triangle	97.63	577.00	494.60	11.30
	Circle	–	67.16	67.16	8.87
	Ellipse	41.61	63.70	103.12	8.04
Train n=0.05 Test n=0.1 det. map	All	55.27	216.36	216.13	46.40
	Square	28.17	128.83	128.74	10.27
	Rectangle	48.38	57.14	108.81	9.71
	Triangle	98.23	554.69	474.65	10.59
	Circle	–	72.76	72.76	8.34
	Ellipse	40.62	64.12	109.28	7.54
Train n=0.05 Test n=0.05 prob. map	All	81.33	78.77	88.69	217.60
	Square	24.03	87.88	100.68	46.31
	Rectangle	31.95	85.97	100.60	48.29
	Triangle	61.36	90.58	77.19	41.38
	Circle	–	285.58	285.58	36.46
	Ellipse	215.86	51.06	71.61	45.16
Train n=0.05 Test n=0.075 prob. map	All	63.03	81.36	75.64	47.30
	Square	21.71	89.21	80.99	10.48
	Rectangle	56.12	72.96	101.66	9.91
	Triangle	91.03	74.43	68.09	10.87
	Circle	–	380.95	380.95	8.39
	Ellipse	89.00	91.27	45.33	7.65
Train n=0.05 Test n=0.1 prob. map	All	64.23	78.98	81.19	46.40
	Square	23.27	101.43	92.07	10.31
	Rectangle	56.30	66.07	91.41	9.69
	Triangle	95.65	82.50	75.87	10.51
	Circle	–	472.30	472.30	8.24
	Ellipse	86.19	59.98	60.60	7.42

1) Angle Error: CNNs trained and tested with no noise (n=0) generally show higher angle errors compared to those tested with noise. Additionally, the probabilistic mapping does not show a clear advantage in reducing angle errors, with some of the highest errors recorded under probabilistic conditions.

2) Width and Height Errors: CNNs trained on no noise (n=0) and tested on a dataset with no noise performed relatively better in terms of width and height errors, but still exhibited significant variability across shapes.

Due to testing on images with increasingly higher levels of noise, the width and height errors of the CNNs also increase significantly, revealing CNNs' sensitivity to noisy input data.

The observed errors in the height and width parameters are relatively high, primarily due to the significant reduction in image size: the input for the YOLO is 200x200, whereas the one for the parameter estimation CNNs is 50x50. This reduction to 25% of the original image size results in an effective increase in the amount of information represented per pixel by a factor of 1.6. This increase in pixel information introduces greater ambiguity during both training and classification phases, ultimately leading to increased errors in these specific parameters and a degradation in the performance of the parameter analysis CNN.

3) Shape-wise Analysis:

- Square: the average errors for squares were generally lower compared to other shapes, with probabilistic mapping showing better robustness when testing with varying levels of noise, having an average drop of accuracy of 17%, whilst deterministic mapping has an average accuracy drop of 30%.
- Rectangle: The accuracy drops between deterministic and probabilistic mappings follow a similar trend as the square error, with a small 4% in average accuracy.
- Triangle: triangles exhibit the highest variability in errors, particularly in height and width, suggesting that this shape poses a significant challenge for CNNs.
- Circle: Circles demonstrate high robustness in classification on all noise levels, although width and height errors are still present and quite significant, especially under probabilistic mapping.
- Ellipse: ellipses also show high variability, with deterministic mapping generally performing significantly better than probabilistic. The introduction of noise led to the general misclassification of the figure, being constantly misinterpreted as a triangle.

4) Overall Performance: CNNs trained on no noise (n=0) perform relatively well when tested on the same set but struggle significantly with increasing noise levels (Table II, rows 1-2). In addition, probabilistic mapping increases the errors in most cases. The authors speculate this is due to the way the probabilistic mapping works, where each obstacle is represented as being slightly bigger than it really is, to account for the sensor's inaccuracy. This leads to shapes with thicker borders, and thus to errors when estimating their pose and, more significantly, their size. Ergo, this shows an interesting trade-off between deterministic and probabilistic mapping.

Figure 5 portrays an example of the full predictive pipeline outcome with each mapping technique and distinct noise levels. It is visually clear that the probabilistic mapping algorithm outperforms its deterministic counterpart when classifying primitive shapes with noisy scans.

Additional results are available in a Github repository[3].

V. Conclusions and Future Work

In this study, a machine learning-based approach was developed and evaluated to enhance the mapping capabilities of mobile robots using LiDAR point clouds. The method

[3]https://github.com/GoncaloLeao/Scientific-Research

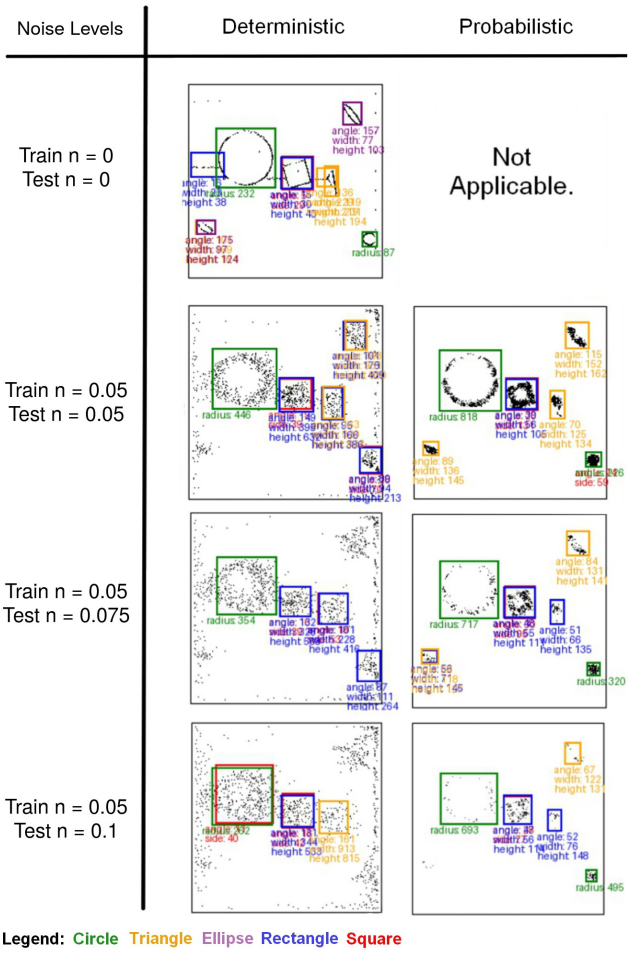


Fig. 5: Example of the YOLO followed by the CNN predictions, in different mapping techniques and noise levels

focuses on the recognition and parameter prediction of geometric primitive shapes, utilizing YOLOv5 for classification and a custom CNN for parameter prediction. The models were trained and tested on both clean and noisy datasets with both a deterministic and probabilistic mapping technique. The probabilistic technique proved more robust to noise for shape recognition, but occasionally had more difficulty than the deterministic one for shape parameter prediction due to thicker borders. Overall, the point cloud processing pipeline's performance suggests it can be of practical significance in real-world applications.

This work has several interesting lines of future work. First of all, other networks can be used for primitive shape recognition, namely Faster R-CNN. Secondly, the inverse sensor model used in the probabilistic mapping technique can be calibrated to better represent the LiDAR's noise. Another possible path worth exploring would be to create a custom implementation of YOLO to not only detect and classify the primitive shapes, but also directly predict their geometric parameters, such as width and radius. This approach could potentially simplify the pipeline by utilizing a single network to handle both tasks and allow for real-time usage. Thirdly, one could experiment using a single (grayscale) input channel for the shape prediction CNNs, rather than three channels, to potentially reduce the model's learning time. Finally, the whole pipeline used in this work can be extended to 3D primitives, by using multiple layers of LiDAR scans, and then adapted to a real-world mobile robot.

Acknowledgments

This work was financed within the project UIDB/00027/2020 (DOI 10.54499/UIDB/50014/2020, https://doi.org/10.54499/UIDB/50014/2020) of the Artificial Intelligence and Computer Science Laboratory (LIACC), funded by national funds through the FCT/MCTES (PIDDAC).

References

[1] S. Xia, D. Chen, R. Wang, J. Li, and X. Zhang, "Geometric primitives in lidar point clouds: A review," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 13, pp. 685–707, 2020.

[2] S. Zheng and M. Castellani, "Primitive shape recognition from real-life scenes using the pointnet deep neural network," *The International Journal of Advanced Manufacturing Technology*, vol. 124, no. 9, pp. 3067–3082, 2023.

[3] E. O. Sodiya, U. J. Umoga, O. O. Amoo, and A. Atadoga, "Ai-driven warehouse automation: A comprehensive review of systems," *GSC Advanced Research and Reviews*, vol. 18, no. 2, pp. 272–282, 2024.

[4] Y.-H. Tseng and S.-S. Jan, "Combination of computer vision detection and segmentation for autonomous driving," in *2018 IEEE/ION Position, Location and Navigation Symposium (PLANS)*, pp. 1047–1052, IEEE, 2018.

[5] J. Janai, F. Güney, A. Behl, A. Geiger, *et al.*, "Computer vision for autonomous vehicles: Problems, datasets and state of the art," *Foundations and Trends® in Computer Graphics and Vision*, vol. 12, no. 1–3, pp. 1–308, 2020.

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[7] H. Zhang, C. Wang, S. Tian, B. Lu, L. Zhang, X. Ning, and X. Bai, "Deep learning-based 3d point cloud classification: A systematic survey and outlook," *Displays*, vol. 79, p. 102456, 2023.

[8] A. Diab, R. Kashef, and A. Shaker, "Deep learning for lidar point cloud classification in remote sensing," *Sensors*, vol. 22, no. 20, 2022.

[9] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660, 2017.

[10] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[11] G. Qian, Y. Li, H. Peng, J. Mai, H. Hammoud, M. Elhoseiny, and B. Ghanem, "Pointnext: Revisiting pointnet++ with improved training and scaling strategies," *Advances in Neural Information Processing Systems*, vol. 35, pp. 23192–23204, 2022.

[12] M. Xu, J. Zhang, Z. Zhou, M. Xu, X. Qi, and Y. Qiao, "Learning geometry-disentangled representation for complementary understanding of 3d object point cloud," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, pp. 3056–3064, 2021.

[13] J. E. Bresenham, "Algorithm for computer control of a digital plotter," in *Seminal graphics: pioneering efforts that shaped the field*, pp. 1–6, ACM, 1998.

[14] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*, ch. 9. MIT Press, 2005.

[15] O. Michel, "Cyberbotics ltd. webots™: professional mobile robot simulation," *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, p. 5, 2004.