# Final report group 30: AI-driven lawn robot

Cocchi Davide Erio, Di Biase Fabio, Novelli Giorgia

November 12, 2025

## Abstract

The primary goal of this project was to enhance a custom-designed autonomous lawn-robot system (Figure 1) initially developed by one of our team members. At its inception, the robot could navigate open environments using LiDAR to detect obstacles and randomly adjust its direction, but it didn't exploit the CMOS sensor fixed on top of the system. Recognizing its potential for improvement, we developed a more advanced software system utilizing computer vision techniques to better guide the robot and introduce new features. A critical improvement is in the new capability of detecting both the state of the grass in front of the robot and the presence of a nearby occlusion, allowing better conduced navigation. Additionally, we introduced the capability to distinguish familiar environments from new ones, enabling the robot to adapt its navigation strategy by constructing a new representation of the surroundings, daily optimizing times and energy.

## 1 Introduction

The algorithm pipeline involves three deep learning models and a two dimensional map to improve the navigation behavior of the lawn robot. This map is structured as a grid where each cell holds data acquired at regular intervals, due to static travel speed of the robot. Each cell stores an image, a binary tag indicating the grass state and a binary tag detecting the presence of imminent obstacles. Initially, the camera captures an image in JPG format, which is resized, and a feature map is extracted using a ResNet50 model. This feature map is compared with the those extracted from the images saved in the map grid. If there's an image similar enough among the stored ones, the map is maintained allowing to plan a navigation path aware of already detected obstacles and of the last recorded grass states. In absence of a match, or if it's the first time system is deployed, an empty map is instantiated and a random navigation starts, populating the grid. During this exploration each acquired image is stored in a map cell, then the image is preprocessed in two different ways and passed through both a custom CNN for the grass state estimation and to a depth estimation network, used to predict occlusions. Once the exploration process is complete, the generated map is stored, making it available for future use. This enables the lawn robot to leverage previously collected data, facilitating more efficient navigation and obstacle avoidance in future scenarios.

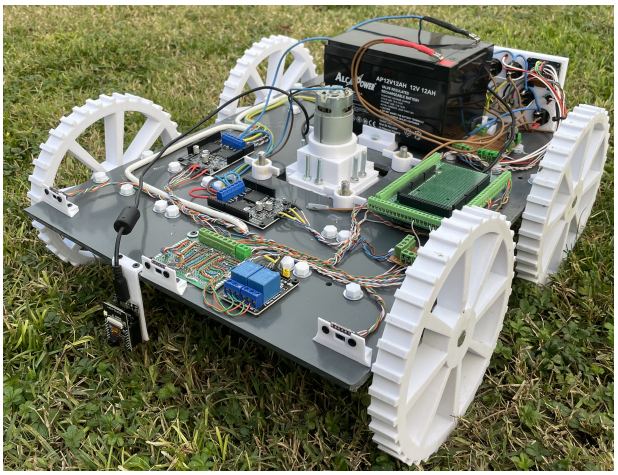


Figure 1: Picture of our lawn robot

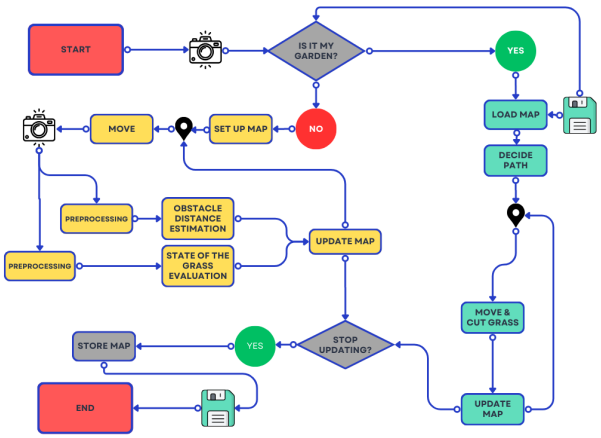



Figure 2: Complete algorithm pipeline

## 2 Related Work

Recent research presents innovative approaches in the field of visual navigation, where an agent must intelligently navigate a 3D environment to reach specific goals. Visual navigation has experienced significant advancements, driven by the development of large-scale datasets and simulation environments. While traditional navigation methods relied heavily on purely geometric representations such as SLAM, recent studies have highlighted the effectiveness of learned approaches that merge geometry with semantic understanding. One of the proposed concepts in this domain is occupancy anticipation[3]. Rather than waiting to observe distant or occluded areas of the 3D environment to determine their occupancy status, the proposed agent predicts the occupancy of unseen regions using RGB(D) inputs. This anticipatory capability allows the agent to make informed decisions about its environment without needing direct observation, enhancing its navigation strategy. It is essential to note that many models in this field are primarily trained on indoor environments, where datasets and simulation environments are more extensively developed. For example, the model of occupancy anticipation has been validated on datasets such as Gibson and Matterport3D, which include over 170 real-world spaces with a variety of obstacles and floor plans. Building on this approach, our study will employ RGB-D images to extract valuable information for map construction. Notably, depth projections will be generated by a model rather than physical devices, and these projections will be used to measure the camera's distance from obstacles.

## 3 Dataset acquisition

We start our approach collecting images directly from an OmniVision OV2640 (2.0 MegaPixel - 10bit CMOS color camera) installed on the front of our lawn robot, making it to explore few domestic outdoor environment in different weather conditions and in different times of the year, to enhance the robustness of the approach. We acquired around 2000 images per class: cut grass and uncut grass. This method was essential to maintain consistency as the images needed to be captured at the same height from the ground guaranteeing the effectiveness of an optimized standard cropping for the whole dataset. Also this standardization allowed to best capture the transition between grass and background in order to detect the state of the garden.

## 4 Image preprocessing

Starting from the grass state estimation network the preprocessing phases are the following (see also figure 4):

Figure 3: To the right, example of a class 0 image (cut grass) while to the left, example of a class 1 image (uncut grass)

- Grayscale conversion of the image from RGB color space to grayscale, to reduce the complex-

ity of the image by removing unnecessary color information, and to reduce the computational load of the model.

- Image cropping: we trimmed the image, to focus on the center of it and discard irrelevant details (sky, trees etc.);
- Blur filter: after cropping, is useful to apply this filter to reduce noise in the image by blurring the sharp transitions and edges in the image;
- Then, Canny Edge Detector algorithm is applied to identify the sharp edges of grass blades in the images. Through empirical testing on our personal dataset, we found that optimal threshold values (for our personal dataset) were 10 for the lower bound and 50 for the upper bound. These values effectively minimized noise and weak responses, which could otherwise introduce confusion into the model.
- Last, as good practice, a normalization of the input between 0 and 1

Moving to the obstacle detection task, the preprocessing phase involves generating a binary mask, which focuses on identifying tree trunks and plants, as these represent the primary obstacles in the dataset's images. The pipeline includes the following steps:

- Cropping the Image: The first step involves cropping the image to focus on the upper portion, where tree trunks and relevant obstacles are most likely to appear. This targeted area helps reduce noise from non-essential regions, enhancing the efficiency of subsequent processing. This framing captures the image vanishing point, allowing the depth estimation network to predict obstacles in the current moving direction.
- Grayscale Conversion: The cropped image is then converted to grayscale.
- Gaussian Filter Application: A Gaussian filter is applied to the grayscale image to smooth it and reduce noise.
- Thresholding for Binary Mask Creation: A thresholding technique is applied to generate a binary mask. A fixed threshold value is used, which was determined through visual testing of the results. This thresholding process highlights regions with intensity values matching those of tree trunks and other potential obstacles, converting them into black areas (foreground) while turning the rest of the image into white (background). This binary mask forms the basis for identifying obstacles within the processed scene.

Additionally, we performed preprocessing on images for input into an existing depth model. This ensures compliance with the model's input specifications. The process begins by cropping the image to retain only the upper half, focusing on the most relevant areas for depth detection. The cropped image is then resized to meet the model's expected input dimensions.

Figure 4: to the right: class 0 image cropped after grayscale conversion; to the center: the same image after blurring filter; to the left: image after canny algorithm

# 5 Methods and architectures

In this section there's a deeper dive in each of the main blocks of the pipeline. The first block, which verifies whether the acquired image belongs to a known garden, utilizes ResNet50, a well known, general-purpose architecture [1]. We removed both the softmax and classification layers to maintain the image feature map, enabling a direct comparison based on image features. To match the acquired image's feature map with those already stored in the

map, we applied the cosine similarity metric, particularly suitable for calculating similarity scores in high-dimensional spaces.

$$\text{Cosine Similarity} = \cos(\theta) = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\|\|\vec{B}\|}$$

For estimating the state of the grass, we developed a custom CNN, carefully balancing the number of parameters against the limited dataset. The result is a streamlined network that could be implemented directly on a lawn robot's microprocessor. To minimize the number of parameters, we kept the channel dimensions of the convolutional layers relatively small, encouraging the network to focus on learning a few meaningful representations, as only a limited amount of relevant detail is present after the preprocessing stages. The network architecture is illustrated in Figure 9b. To solve the problem of identifying obstacles

(a) sample (b) binary mask

(c) valid areas

Figure 5

and estimating the distance between them and the lawn mower, we developed a method that uses connected components and a pre-existing model for image depth estimation. After preprocessing, we obtain a binary mask that separates potential obstacles from the background, while also filtering out noise from elements such as grass blades and other possible environmental disturbances. Starting from this binary mask, regions that can be considered potential obstacles are identified using the connected components technique. This technique analyzes the mask to detect contiguous groups of pixels that share the same intensity, grouping them as individual components. Next, the identified regions are filtered based on an area threshold: any component with an area below the predefined threshold is discarded, as it is deemed too small to represent a significant information. An example of the output for each elaboration is shown in Fig. 5. The resulting mask of regions where obstacles have been identified is then compared with the image's depth information. Using this data, the average depth for each area is calculated. The area with the smallest average distance is then returned as the relevant data for the garden map, considering it as the obstacle closest to the lawn mower. The depth of the image is obtained using the ANSRGB model[3], which is a deep learning-based model trained to estimate depth from RGB images. The provided model architecture consists of three main parts:

- Feature extraction with pre-trained ResNet-18: This model serves as the base for extracting features from the input.
- Equivalent Fully Connected Layers Using 1x1 Convolutions: These layers replace traditional fully connected layers, enabling feature processing while maintaining the spatial dimensions.
- Upsampling: This section is designed to restore the spatial resolution of the image, with the goal of increasing the output dimensions to match the original resolution.

After gathering information about the explored garden area, including the presence of obstacles, cut and uncut grass zones, and saving them in a map, we implemented a navigation algorithm that is designed to enable the lawnmower robot to explore all the uncut grass cells in a grid, using a backtracking approach. It works by creating a navigation grid based on two input matrices: one that marks obstacles and

another that shows where the grass is either cut or uncut. At the start, the algorithm looks at each cell to figure out whether it's an obstacle, cut grass, or uncut grass. Obstacles are marked as blocked, cut grass is treated as passable, and uncut grass is considered open for exploration. The resulting navigation grid combines information about obstacles and the grass's state, creating a map for the algorithm to follow. Once the navigation grid is set up, the algorithm starts from a given starting point. It uses a stack to keep track of the current position and make use of the backtracking technique when necessary. The algorithm checks the neighboring cells in four directions. If it finds an unvisited neighbor with uncut grass, it moves to that cell, adds it to the path, and continues searching. If it reaches a point where no valid neighboring cells are available, the algorithm backtracks by removing the last visited position from the stack and retracing its steps. This process continues until all reachable uncut grass cells have been explored.

# 6 Experiments

For each component of the pipeline, we tested different solutions. the following subsections describe the main choices.

## 6.1 Retrieval Algorithm

To distinguish the owner's garden from an unknown one, the algorithm extracts feature maps from saved images using the ResNet50 model [1] and compares them with the feature map from the robot's first acquisition. To show its efficacy we choose an anchor image (Fig.6a) and a pair of samples, positive (Fig.6b) and negative (Fig.6c). The results are shown in (Tab 1). The threshold used to distinguish between known and unknown gardens is

$$\tau = 0.6$$

.

| Sample | Cosine similarity |
|---|---|
| **positive** | 0.71 |
| **negative** | 0.47 |

Table 1: Retrieval algorithm results

(a) anchor sample

(b) positive sample

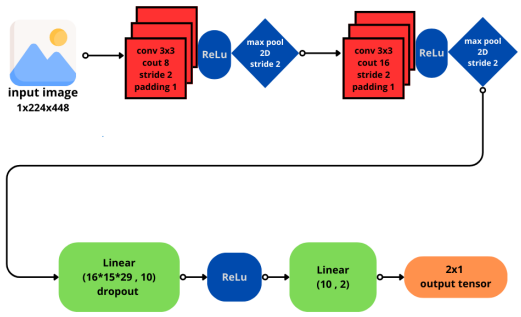
(c) negative sample

Figure 6: samples used to measure the cosine similarity

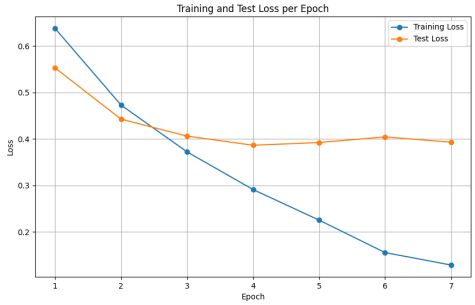
## 6.2 Custom CNN

Given the nature of this problem and the simplifications introduced by preprocessing, we opted to design

an ad-hoc network with an emphasis on optimizing the parameter count. Framing it as a binary classification task, we consistently employed cross-entropy loss across all experiments. Our initial approach focused on reducing feature map dimensions by leveraging stride-2 convolutions and stride-2 max pooling, effectively minimizing the parameter count in the final fully connected layer. Despite its simplicity, this model showed significant signs of overfitting after only four epochs (Fig.7b), and it was challenging to balance it with regularization techniques due to the limited number of neurons showing an oscillating behavior between converging too fast and don't learning at all.




(a) first trial: architecture




(b) first trial: learning curve

Figure 7

In the second attempt, we maintained the same architecture but replaced the strided convolution with a non strided convolution, using only the max pooling layer for downsampling. This allowed us to maintain a larger feature map before the fully connected layers, possibly increasing the model's learning capacity and allowing a stronger regularization. As shown in Fig.(8), this approach resulted in a more stable training curve and yielded a slight improvement in accuracy.

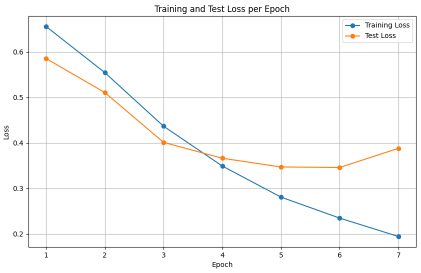


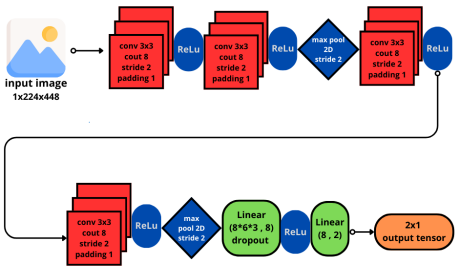

Figure 8: second trial learning curve

The final architecture introduces significant changes: the network depth was increased while maintaining constant channel dimensions, and the receptive field was expanded. Each layer retains a stride of 2 to generate a more compressed feature map. This design ensures the flattened feature map has a very low parameter count, effectively reducing the risk of overfitting. In this case, less proves to be better, as the accuracy further improved.

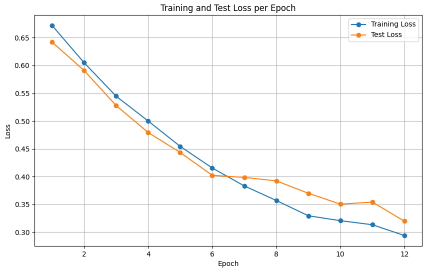
| Architecture | Accuracy |
|---|---|
| **First trial** | 79.14% |
| **Second trial** | 83.48% |
| **Third trial** | 86.71% |

Table 2: Comparison between best results of each of the three architectures

We also tested with multiple combination of hyperparameters in the last architecture, observing an aligned behavior with theory. For instance the best accuracy is obtained using a small batch size, which leads to a more explorative behavior of the gradient during training.




(a) Third trial: chosen architecture




(b) Third trial: learning curve

Figure 9

| num epochs | learning rate | batch size | dropout | accuracy |
|---|---|---|---|---|
| 10 | 0.001 | 8 | 0.1 | 84% |
| 15 | 0.0005 | 8 | 0.2 | 78% |
| 9 | 0.0015 | 8 | 0.2 | 80% |
| 12 | 0.001 | 4 | 0.2 | 79% |
| 16 | 0.0005 | 4 | 0.1 | 84% |
| 12 | 0.001 | 4 | 0.1 | 87% |

Table 3: Various experiments on the definitive network hyperparameters, the first column represents the epoch at which the model has obtained the highest accuracy

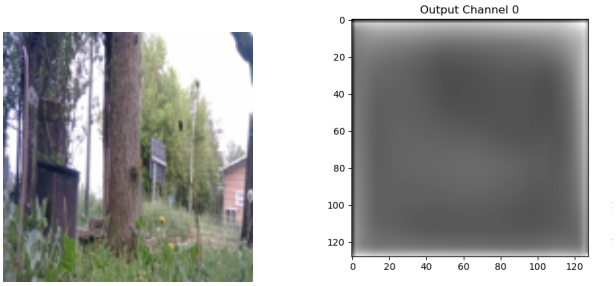
## 6.3 Obstacle detection

The best results for the obstacle detection process were achieved by applying a threshold value of 100 to the binary mask obtained at the end of the preprocessing pipeline. To highlight significant areas, the connected components technique was used. A minimum area threshold of 600 was set, filtering out components smaller than this threshold as they were considered too minor to pose a genuine obstruction. These specific threshold values were chosen after testing with a sample set of images, where they consistently provided acceptable results.

(a) (b) binary mask

Figure 10

In the example shown in Fig.10, we have a specific case of uncut grass where, visually from the image, it can be observed that no obstacles are present. The binary mask returns a small portion of the image in black (potential obstacle), which, however, will be ignored due to the area threshold constraint after applying the connected components technique. Fig.11, shows an example of depth prediction. The result of calculating the distance between the obstacle and the camera is not precise, as the model was trained on indoor environments. Therefore, we can only obtain an approximate estimate of the value.




(a) sample (b) depth prediction

Figure 11: depth estimation map

### 6.4 Navigation

The backtracking algorithm used for navigation purposes ensures that every reachable uncut grass area is explored. However, this method can be slow and computationally expensive when applied to large grids, as it explores every potential path. It does not always select the most optimal route, often leading to unnecessary revisits to cells and redundant backtracking over already explored areas. The use of a stack to track visited positions, along with the backtracking process itself, can consume a significant amount of memory, especially in large or complex grids. In our tested example, Fig.(12) shows the result of the exploration of a 5x5 matrix with some obstacles.

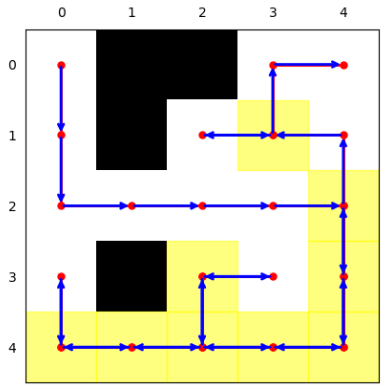



Figure 12: obstacles are represented by black cells. The yellow cells indicate that the robot traversed them multiple times, highlighting backtracking operations performed to locate remaining cells to visit.

## 7 Conclusion

In conclusion, we have made significant improvements to the lawn robot navigation path and method of work, enabling it to map the owner's garden and at the same time we've drastically reduced the time to reach a standard daily work, distinguish between work and living or danger zones and identifying obstacles, storing all this information and data for daily uses. Additionally, the robot is now capable of adapting to a new environment when moved, learning its layout dynamically, recreating its own perimeter of work each time. However, given the academic scope of this project, there remain areas for further refinement. These include optimizing the custom network for better performances, enhancing the depth estimation system originally designed for indoor environments and advancing the navigation system to handle more complex scenarios.

# 8 bibliography

[1]https://pytorch.org/vision/0.18/models/generated/torchvision.models.resnet50.html
[2]$https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html$
[3] https://arxiv.org/abs/2008.09285
[4]$https://docs.opencv.org/4.x/d7/da8/tutorial_table_of_content_imgproc.html$
[5]$https://pytorch.org/tutorials/beginner/basics/data_tutorial.html$