



HAL
open science

LiDAR-Only Crop Navigation for Symmetrical Robot

Rémy Guyonneau, Franck Mercier, Gabriel Freitas Oliveira Freitas

► **To cite this version:**

Rémy Guyonneau, Franck Mercier, Gabriel Freitas Oliveira Freitas. LiDAR-Only Crop Navigation for Symmetrical Robot. *Sensors*, 2022, 22 (22), pp.8918. 10.3390/s22228918 . hal-03861646

HAL Id: hal-03861646

<https://univ-angers.hal.science/hal-03861646>

Submitted on 13 Feb 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.


L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

Article

LiDAR-Only Crop Navigation for Symmetrical Robot

Rémy Guyonneau , Franck Mercier  and Gabriel Freitas Oliveira

LARIS (Laboratoire Angevin de Recherche en Ingénierie des Systèmes), SFR MATHSTIC, University of Angers, F-49000 Angers, France

* Correspondence: remy.guyonneau@univ-angers.fr; Tel.: +33-244-687-595

Abstract: This paper presents a navigation approach for autonomous agricultural robots based on LiDAR data. This navigation approach is divided into two parts: a line finding algorithm and a control algorithm. The paper proposes several line finding algorithms (based on PEARL/Ruby approach) that extract lines from a LiDAR data set. Once the lines have been processed from the data set, a control algorithm filters these lines and, using a fuzzy controller, generates the wheel speed commands to move the robot among the crop rows. This navigation approach was tested using a simulator built on ROS middle-ware and Gazebo (the source codes of the simulation are available on GitHub). The results of the simulated experiments show that the proposed approach performs well for a large range of crop configurations (with or without considering weeds, with or without holes in the crop rows. . .).

Keywords: crop navigation; LiDAR measurements; line extraction; fuzzy controller; ROS/Gazebo



Citation: Guyonneau, R.; Mercier, F.; Oliveira Freitas, G. LiDAR-Only Crop Navigation for Symmetrical Robot. *Sensors* **2022**, *22*, 8918. <https://doi.org/10.3390/s22228918>

Academic Editors: Adam Ziębiński, Erik Kyrkjebø and Daniel Großmann

Received: 17 October 2022

Accepted: 16 November 2022

Published: 18 November 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

1.1. Context

The development of robotic tools for agriculture is a growing field. Current research is exploring a variety of tasks ranging from weeding robots [1,2] to harvesting robots [3,4]. Most of the platforms being developed aim to be autonomous.

One of the most fundamental problems for autonomous robots is autonomous navigation. In an agricultural application, such as market gardening, the robot must be able to follow the crop rows regardless of the weather conditions, the surrounding luminosity, or the configuration of the field.

Most recent works consider GPS or camera data for navigation [5,6]. However, these sensors do not provide reliable data in all conditions. Indeed, at the edge of a forest or in a greenhouse, the GPS signal is degraded. In these situations, range sensors are commonly used. This paper proposes an approach based on LiDAR (Light Detection and Ranging) data.

The work presented here aims to propose an approach for the navigation of a symmetrical robot based on LiDAR sensor data only. This paper presents a number of novelties, including:

- The improvement of the existing line finding algorithm (Figures 1 and 2 introduces the line finding idea). The interested reader can refer to Section 4 for the algorithm behavior comparison;
- The setup of filters and a fuzzy controller to have a full navigation stack. The control algorithm presented in this paper (Section 3) allows the simulated robot to move autonomously through the entire field. Considering a symmetrical robot eases the row changing maneuver;
- The development of a new simulation based on ROS and Gazebo. To ease the reusability of the work and to be able to test the algorithms for several crop configurations, a simulation has been developed based on ROS Robot Operating System middle-ware

and Gazebo (Section 4). Note that all the simulator source codes and documentation are available in Ref. [7].

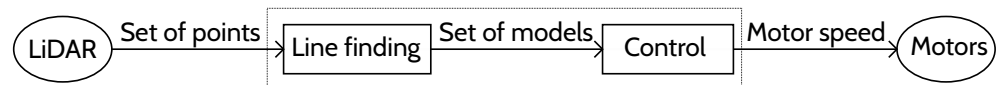


Figure 1. Overview of the approach: the LiDAR sensor provides points, and, from those points, lines (models) are extracted and from those lines the motor speeds are deduced.

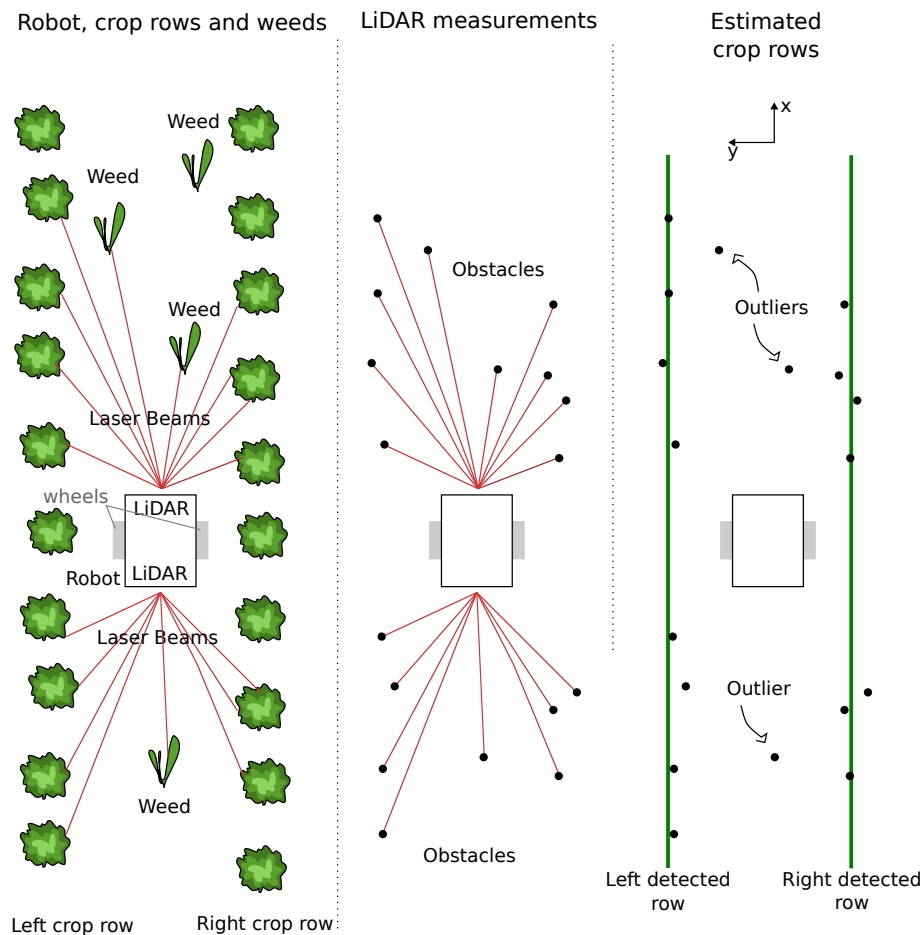


Figure 2. The line finding problem: The objective is to identify the crop rows from LiDAR data. The left part of the figure depicts a top view of the scene (a robot moving between two crop rows), the middle part depicts the data from the sensors (i.e., points corresponding to plants) and the right part shows an ideal result from the line finding algorithm (that is, the left and right crop rows have been identified from the LiDARs data set).

1.2. Overview of the Approach

Using LiDAR data, without prior knowledge of the field, the objective is for the robot to be able to autonomously navigate between the rows. The considered robot is a symmetrical two-wheeled differential robot. The navigation approach presented in this paper can be divided into two parts: a line finding algorithm and a control algorithm, as depicted in Figure 1.

To be able to navigate among the crops, it is necessary to identify the crop rows from the LiDAR data. This step corresponds to the line finding process of the approach and is depicted in Figure 2. One should note that it is assumed that the robot is equipped with 2D LiDAR sensors and that the sensors can detect the crop plants and the weeds (i.e., the LiDAR is low enough or the plants are high enough). In other words, plants have to

be detected by the sensors, otherwise this approach can not be applied. Nomenclature resumes the considered notations.

The line finding algorithm thus provides a set of models (i.e., a set of lines) that best fit the points detected by the sensors. Those models (lines) are then filtered in order to hopefully correspond to actual rows in the crop. This filtering is done by the control algorithm, which also provides a fuzzy controller to maintain the robot in the middle of the rows.

Section 2.1 presents several line finding algorithms while Section 3 describes the considered control algorithm. Section 4 presents the designed simulation and the results obtained when testing the algorithms. Finally, Section 5 concludes this paper.

2. Materials and Methods

2.1. Line Finding Algorithms

To ease the reading of this paper, the Ruby algorithm introduced in Ref. [8] is presented in Section 2.1.1. The novelties brought to this algorithm are then presented in Section 2.2.

2.1.1. The Ruby Algorithm

The Ruby algorithm is based on the Pearl method presented in [9]. Pearl, thus Ruby, is a method that aims at minimizing a function called energy. In the latter, a model $L_j : f_j(x) = a_jx + b_j$ corresponds to a line, and \mathcal{L}_i depicts a set of models. A point p (an obstacle detected by the LiDARs) is associated with a model $L(p)$. Note that $L(p)$ could be the empty model L_\emptyset (if p is an outlier for instance). These notations are listed in Nomenclature.

The considered energy function is described in Equation (1).

$$\mathcal{E}(\mathcal{L}) = \mathcal{E}_\emptyset(\mathcal{L}) + \mathcal{E}_\mathcal{N}(\mathcal{L}) + \sum_{L_j \in \mathcal{L} \setminus \{L_\emptyset\}} \mathcal{E}_L(L_j). \quad (1)$$

For a set of models \mathcal{L} , the energy $\mathcal{E}(\mathcal{L})$ is divided into three terms: the outlier energy $\mathcal{E}_\emptyset(\mathcal{L})$, Equation (2), the penalty energy $\mathcal{E}_\mathcal{N}(\mathcal{L})$, Equation (3), and the sum of all the model energies $\mathcal{E}_L(L_j)$, Equation (6).

The outlier energy $\mathcal{E}_\emptyset(\mathcal{L}_i)$ aims at taking into account the points that are not associated with a model (i.e., those that are associated with the empty model L_\emptyset). It is defined as

$$\mathcal{E}_\emptyset(\mathcal{L}_i) = \rho \cdot \sum_{p \in P(L_\emptyset)} 1, \quad (2)$$

where $\sum_{p \in P(L_\emptyset)} 1$ is the number of points in the current empty model $L_\emptyset \in \mathcal{L}_i$ and ρ is a constant value that is heuristically chosen. As it is assumed that the LiDAR will detect more crops than weeds, it is appropriate to penalize the points that are not attached to any model (i.e., outliers and weeds).

The penalty function tends to favor the association of two nearby points (according to the Euclidean distance) to the same model. This energy is defined as

$$\mathcal{E}_\mathcal{N}(\mathcal{L}_i) = \lambda \cdot \sum_{(p,q) \in \mathcal{N}} w_{pq} \cdot \delta_{\neq}(L(p), L(q)), \quad (3)$$

where \mathcal{N} is the set of neighboring points such that an element $(p, q) \in \mathcal{N}$ corresponds to two points p and q in the same neighborhood, with p associated with the model $L(p)$ and q associated with the model $L(q)$. λ is a constant heuristically chosen. $\delta_{\neq}(L(p), L(q))$ is defined as

$$\delta_{\neq}(L(p), L(q)) \begin{cases} 1 & \text{if } L(p) \neq L(q) \\ 0 & \text{otherwise} \end{cases}, \quad (4)$$

additionally

$$w_{pq} = \exp \frac{-\|p - q\|^2}{\zeta^2}, \quad (5)$$

where $\|p - q\|$ is the Euclidean distance between the points p and q , while ζ is a heuristically chosen constant.

Finally, the energy of a model L_j is defined as

$$\mathcal{E}_L(L_j) = \sum_{p \in P(L_j)} \|p - L_j\|, \quad (6)$$

where $\|p - L_j\|$ is the Euclidean distance between a point p attached to the model L_j and that model.

2.1.2. Details of the Algorithm

The Ruby algorithm is presented in Figure 3 and detailed in Algorithm 1. The different steps are explained in the latter.

1. Line 1 Algorithm 1, which corresponds to the initialization. The initial set of models \mathcal{L}_0 is initialized with the one from the previous find line computation \mathcal{L}_i^{-1} . Note that if it is the first ever call of line finding, the initial set of models is initialized with only an empty model: $\mathcal{L}_i^{-1} = \{L_\emptyset\}$ with $P(L_\emptyset) = \mathcal{Z}$, i.e., all the points are associated with the empty model, \mathcal{Z} being the set of all the points. This is relevant because in a crop navigation context, it is assumed that between two LiDAR scans, the robot will sense mostly the same crops and weeds: the rows will not be completely different from two consecutive find line calls.
2. Lines 5 to 10 Algorithm 1. Three random points (p, q, z) are picked in L_\emptyset and the best model is computed (linear regression) so that it fits those points. This creates a new model that is added to the model set. This process is then repeated until half of the outliers are attached to a model (or below a threshold).
3. Lines 11 to 16 Algorithm 1, the merging part. It merges all the models that are too close to each other.
4. Lines 17 to 20 Algorithm 1, the deleting part. It removes all the models that do not meet the constraint

$$\frac{\mathcal{E}_L(L_j)}{\psi_{\mathcal{L}_i}(L_j) \cdot \sum_{p \in P(L_j)} 1} - \alpha < 0, \quad (7)$$

where $\mathcal{E}_L(L_j)$ is the energy of the model L_j as described in Equation (6), α is the constant chosen heuristically and $\psi_{\mathcal{L}_i}(L_j)$ is the number of models of \mathcal{L}_i parallel to the model L_j . $\psi_{\mathcal{L}_i}(L_j)$ is defined as

$$\psi_{\mathcal{L}_i}(L_j) = \sum_{L_k \in \mathcal{L}_i \setminus \{L_j, L_\emptyset\}} \delta_{||}(L_j, L_k), \quad (8)$$

with

$$\delta_{||}(L_j, L_k) \begin{cases} 1 & \text{if } |a_j - a_k| - \beta < 0 \\ 0 & \text{otherwise} \end{cases}, \quad (9)$$

where a_j and a_k are the slopes of the models L_j and L_k , and β a constant chosen heuristically.

5. Lines 21 to 24 Algorithm 1. All the points are detached from their models and re-attached to the closest model. Then the new energy is computed for the complete set of models.
6. Lines 26 to 27 Algorithm 1. If the energy of the new set of models is worse (bigger) than the previous iteration, the previous set of models is kept instead of the new computed one.

Algorithm 1: Ruby algorithm.

```

1 Data:  $\mathcal{Z}, \mathcal{L}_i^{-1}$ 
2 Result:  $\mathcal{L}_i$ 
3  $\mathcal{L}_0 = \mathcal{L}_i^{-1}, \mathcal{E}(\mathcal{L}_0) = +\infty;$ 
4 for  $i = 1, \dots, \text{max iteration}$  do
5   while  $\sum_{p \in P(L_\emptyset)} 1 < \text{threshold}$  do
6      $(p, q, z) \in P(L_\emptyset);$ 
7      $P(L_\emptyset) = P(L_\emptyset) \setminus \{p, q, z\};$ 
8      $L_j = \arg \min_{L_j} \|p - L_j\| + \|q - L_j\| + \|z - L_j\|;$ 
9      $P(L_j) = \{p, q, z\};$ 
10     $\mathcal{L}_i = \mathcal{L}_i \cup L_j;$ 
11  end
12  for  $\forall L_j \in \mathcal{L}_i \setminus \{L_\emptyset\}$  do
13    for  $\forall L'_j \in \mathcal{L}_i \setminus \{L_\emptyset, L_j\}$  do
14      if  $\|L_j - L'_j\| < \text{threshold}$  then
15         $P(L_j) = P(L_j) \cup P(L'_j);$ 
16         $\mathcal{L}_i = \mathcal{L}_i \setminus \{L'_j\};$ 
17      end
18    end
19     $L_j = \arg \min_{L_j} \sum_{p \in P(L_j)} \|p - L_j\|;$ 
20  end
21  for  $\forall L_j \in \mathcal{L}_i \setminus \{L_\emptyset\}$  do
22    if  $\frac{\mathcal{E}_L(L_j)}{\psi_{\mathcal{L}_i}(L_j) \cdot \sum_{p \in P(L_j)} 1} - \alpha < 0$  then
23       $\forall p \in P(L_j), L(p) = L_\emptyset;$ 
24       $\mathcal{L}_i = \mathcal{L}_i \setminus \{L_j\};$ 
25    end
26  end
27  for  $\forall p \in \mathcal{Z}$  do
28     $L(p) = L_j | \forall L'_j \in \mathcal{L}_i \setminus \{L_\emptyset\}, \|p - L_j\| < \|p - L'_j\|;$ 
29    if  $\|p - L(p)\| > \text{threshold}$  then
30       $L(p) = L_\emptyset;$ 
31    end
32  end
33   $\mathcal{E}(\mathcal{L}_i) = \mathcal{E}_\emptyset(\mathcal{L}_i) + \mathcal{E}_N(\mathcal{L}_i) + \sum_{L_j \in \mathcal{L}_i \setminus \{L_\emptyset\}} \mathcal{E}_L(L_j);$ 
34  if  $\mathcal{E}(\mathcal{L}_i) > \mathcal{E}(\mathcal{L}_{i-1})$  then
35     $\mathcal{L}_i = \mathcal{L}_{i-1};$ 
36  end
37 end

```

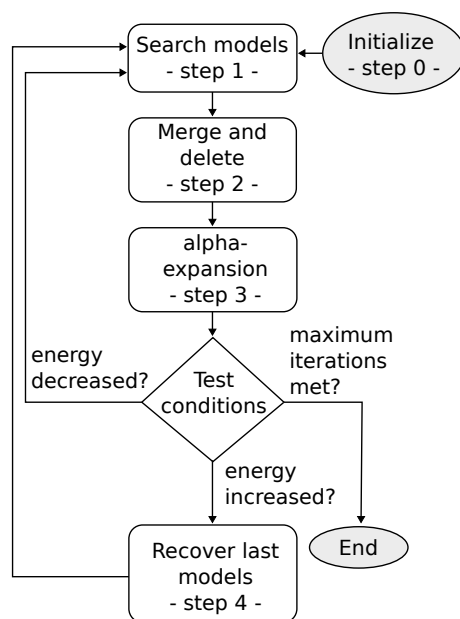


Figure 3. Overview of the Ruby method.

2.2. Ruby Suggested Refinements

In this section we will present refinements for the Ruby algorithm. These refinements are part of the originality of the work presented in this paper. They lead to an improvement of the robot navigation results, as shown in Section 4.

2.2.1. Ruby Genetic

Pearl and Ruby algorithms are very similar to genetic algorithms [10]. To enhance this resemblance, the Ruby genetic refinement proposes to modify the deleting part of step 2 (Figure 3). In the classical Ruby approach, a constant α is used (Equation (7)) to decide if a model has to be removed or not. Ruby genetic proposes a fitness criterion instead, named φ_j and defined as

$$\varphi_j = \frac{b_j^2 + \tau \cdot \mathcal{E}_L(L_j)}{\psi_{\mathcal{L}_i}(L_j) \cdot \sum_{p \in P(L_j)} 1} \quad (10)$$

with $\mathcal{E}_L(L_j)$, defined in Equation (6), $\psi_{\mathcal{L}_i}(L_j)$ defined in Equation (8) and where τ is a heuristically chosen constant. By using this criterion, the best models are sorted, while the others are removed.

2.2.2. Ruby Genetic One Point

The idea of this refinement is based on the assumption that one plant of the crop will generate several LiDAR readings. Thus, the data that are close to each other can be merged together as they may belong to the same plant. This leads to the computation of a new point set named \mathcal{Z}^* . The process is described in Algorithm 2.

For this Ruby Genetic One Point refinement, the input data is no longer \mathcal{Z} , as defined in Algorithm 1, but \mathcal{Z}^* . The rest of the algorithm remains the same as for the Ruby genetic.

Algorithm 2: The computation of \mathcal{Z}^* .

```

1 Data:  $\mathcal{Z}$ 
2 Result:  $\mathcal{Z}^*$ 
3  $\mathcal{Z}^* = \emptyset$ ;
4 for  $\forall p \in \mathcal{Z}$  do
5   if first iteration then
6      $p^* = p$ ;
7   end
8   else
9     if  $\|p^* - p\| < \text{threshold}$  then
10       $p^* = \frac{p^* + p}{2}$ ;
11    end
12    else
13       $\mathcal{Z}^* = \mathcal{Z}^* \cup \{p^*\}$ ;
14    end
15  end
16 end

```

2.2.3. Ruby Genetic One Point Positive/Negative

This refinement proposes to change the way models are found (step 1 in Figure 3). For the classical Ruby approach, three points are randomly taken from the empty model L_\emptyset . However, assuming that the robot will most of the time be parallel to the crop rows, and not perpendicular to them, the points can be separated into two sets $P_{left}(L_\emptyset)$ and $P_{right}(L_\emptyset)$, defined as

$$P_{left}(L_\emptyset) = \{p \in P(L_\emptyset) \mid y_p < 0, \quad (11)$$

$$P_{right}(L_\emptyset) = \{p \in P(L_\emptyset) \mid y_p > 0. \quad (12)$$

Then the research for models is done into those two subsets. Thus, step 1 of the Ruby algorithm (Figure 3), i.e., lines 5 to 10 of Algorithm 1, becomes as detailed in Algorithm 3. An example of this new model research approach is depicted in Figure 4.

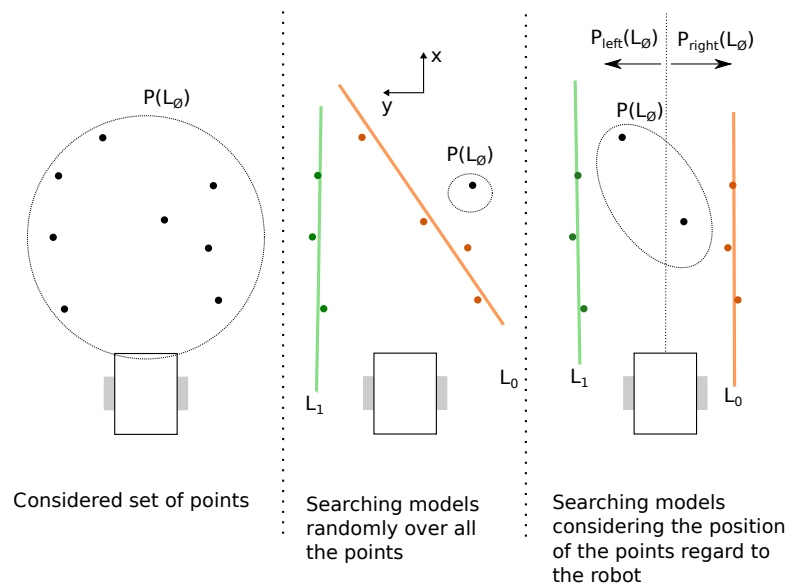


Figure 4. The idea behind considering $P_{left}(L_\emptyset)$ and $P_{right}(L_\emptyset)$. Without doing so, there is a chance that the model research considers improbable models regarding the orientation of the robot (middle part of the illustration), reducing the chance of finding the *best models*. On the other hand, while dividing the model research into left and right points, as the rows should be on the left or on the right, the chances of finding the correct models are increased (the right section of the illustration).

Algorithm 3: Ruby Genetic One Point Positive/Negative-Search models-step 1.

```

1 Data:  $\mathcal{L}_i$ 
2 Result:  $\mathcal{L}_i$ 
3 while  $\sum_{p \in P_{left}(L_\emptyset)} < threshold$  do
4    $(p, q, z) \in P_{left}(L_\emptyset)$ ;
5    $P_{left}(L_\emptyset) = P_{left}(L_\emptyset) \setminus \{p, q, z\}$ ;
6    $L_j = \arg \min_{L_j} \|p - L_j\| + \|q - L_j\| + \|z - L_j\|$ ;
7    $P(L_j) = \{p, q, z\}$ ;
8    $\mathcal{L}_i = \mathcal{L}_i \cup L_j$ ;
9 end
9 while  $\sum_{p \in P_{right}(L_\emptyset)} < threshold$  do
10   $(p, q, z) \in P_{right}(L_\emptyset)$ ;
11   $P_{right}(L_\emptyset) = P_{right}(L_\emptyset) \setminus \{p, q, z\}$ ;
12   $L_j = \arg \min_{L_j} \|p - L_j\| + \|q - L_j\| + \|z - L_j\|$ ;
13   $P(L_j) = \{p, q, z\}$ ;
14   $\mathcal{L}_i = \mathcal{L}_i \cup L_j$ ;
end

```

2.2.4. Ruby Genetic One Point Positive/Negative Infinity

The last considered refinement was proposed after noticing that as a point can only be associated with one model, a *bad model* can consume a point of a potentially *good model*, as it can be noticed in the middle part of Figure 4. To handle this limitation, this refinement proposes to allow a point to be attached to several models at the same time.

The main drawback of this approach is that it is time-consuming: all the points have to be checked all the time.

3. The Control Algorithm

Once models (lines) have been extracted from the points (the data set from the LiDAR), a control can be processed based on those models. The control algorithm presented here can be divided into three main steps: an initialization, filtering, and control step. The steps are depicted in Figure 5.

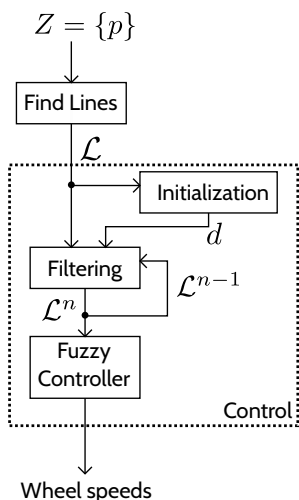


Figure 5. Details of the control approach.

3.1. Initialization

The idea of the overall approach is to allow the robot to autonomously navigate into a crop with the minimum of prior information (width and length of the rows for instance). The only assumptions about the field are that:

- The field is organized in straight lines, as this is the case most of the time [11,12];
- The distance between two rows does not change inside a crop;
- The initial position of the robot is somewhere between two crop rows;
- The initial orientation of the robot is roughly parallel with the crop rows.

It can be noticed that the two first assumptions (about the field configuration) do not reduce the application as it is done in most crop row detection approaches [13,14]. The last two assumptions (about the robot's initial position) may be more restricting, but as long as it is assumed that the robot can be placed in the crop, it is not an issue.

Based on these assumptions, an initialization step has been developed. The idea is to compute the distance between the rows from the models given by the line finding algorithm before starting to move the robot.

To that end, the initialization process extracts from the models the closest pair of models that are equidistant to the robot (the robot should be in the middle of two rows for its initial position) and have a slope close to 0 (the robot should be parallel to the crop rows). Once a pair of models $\{L_1, L_2\}$ is found, the distance between the rows is computed as follows (see Figure 6).

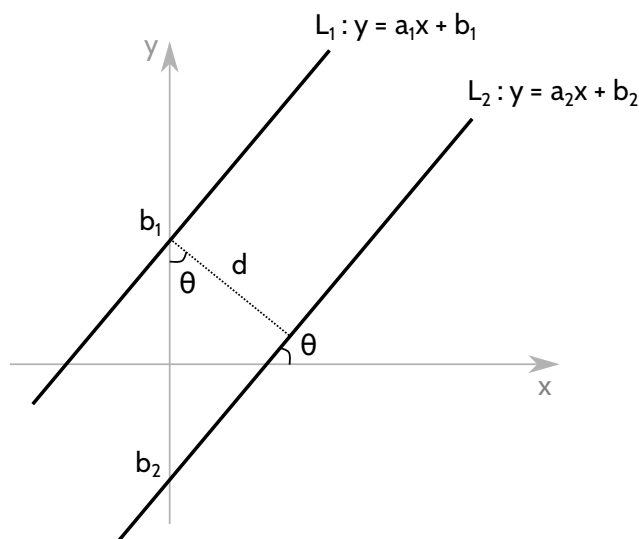


Figure 6. Computing the distance between two parallel models L_1 and L_2 . Note that $a_1 = a_2$ for the models to be parallel.

$$\begin{aligned}\cos(\theta_r) &= \frac{d}{|b_1 - b_2|}, \\ d &= \cos(\theta_r) \cdot |b_1 - b_2|, \\ d &= \cos(\arctan(a_1)) \cdot |b_1 - b_2|, \\ d &= \cos(\arctan(a_2)) \cdot |b_1 - b_2|,\end{aligned}$$

With θ_r the orientation of the robot, d the distance between two rows in the field and (a_i, b_i) the parameters of the model L_i . As

$$\cos(\arctan(a)) = \frac{1}{\sqrt{a^2 + 1}} \quad (13)$$

It can be concluded that

$$d = \frac{|b_1 - b_2|}{\sqrt{a_1^2 + 1}} = \frac{|b_1 - b_2|}{\sqrt{a_2^2 + 1}} \quad (14)$$

To have a more stable distance, this distance computation is done several times (according to several results of the line finding algorithm) and an average of all the computed distances is done to get an estimation of the distance between two rows. This estimation is then stored for use by the robot's navigation.

3.2. Filtering

The line finding algorithm, as detailed in Section 2.1, provides models (lines) according to a point set (LiDAR data). However, most of the time some returned models do not correspond to effective rows in the field (Figure 7). Filtering is then required to remove models that do not meet the field geometry assumptions (distance d between the rows and previously computed and verified models). This filter is presented in Algorithm 4 and is detailed in the latter.

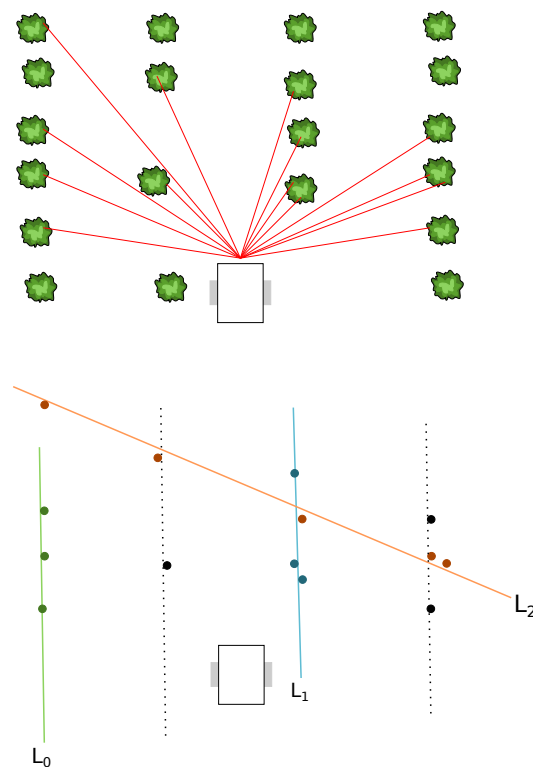


Figure 7. Assuming that from the top configuration, the line finding algorithm returns the models L_0 , L_1 , and L_2 (bottom part of the figure). The expected behavior of the filter is to keep the models L_0 and L_1 (removing the model L_2) and knowing the distance d between the rows, to compute the missing models (dotted lines). Note that for the bottom part of the figure, the colors of the dots correspond to the models the points are associated to.

The required data are:

- \mathcal{L} : the set of models provided by the line finding algorithm;
- $\mathcal{L}^{n,-1} = \{L^{1,-1}, L^{2,-1}, \dots, L^{i,-1}, \dots, L^{n,-1}\}$: the filtered model set of the previous iteration with n a defined number of models that is constant during the robot navigation. The i index represents the position of the row identified by the model (from left to right).
- d : the distance between the rows, computed during the initialization step (Section 3.1).

Algorithm 4: Model filtering.

```

1 Data:  $\mathcal{L}, \mathcal{L}^{n,-1}, d$ 
2 Result:  $\mathcal{L}^n$ 
3  $\mathcal{L}^n = \emptyset;$ 
4 for  $\forall L_j \in \mathcal{L}$  do
5   for  $i = 1, \dots, n$  do
6     if  $\|L_j - L^{i,-1}\| < \text{threshold}$  then
7        $L^i = L_j;$ 
8        $\mathcal{L}^n = \mathcal{L}^n \cup \{L^i\};$ 
9     end
10  end
11 for  $i = 1, \dots, n$  do
12   if  $L^i \notin \mathcal{L}^n$  then
13      $L^i : f(x) = \bar{a}x + b^{i_{best}} + d \cdot (i_{best} - i) \sqrt{\bar{a}^2 + 1};$ 
14      $\mathcal{L}^n = \mathcal{L}^n \cup \{L^i\};$ 
15   end
16 end

```

The result of the filtering is a filtered model set $\mathcal{L}^n = \{L^1, \dots, L^i, \dots, L^n\}$, with n a fixed number of models. Note that a filtered model L^i may be extracted from the models \mathcal{L} provided by the line finding algorithm or may be computed according to other filtered models L^j and the row distance d . Figure 8 depicts an expected filtered result according to the situation described in Figure 7.

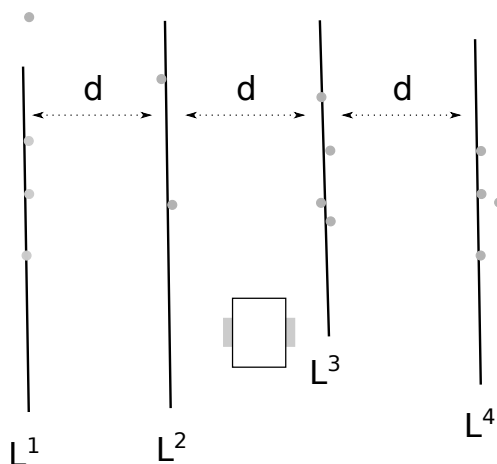


Figure 8. The filtered models \mathcal{L}^n , with $n = 4$, resulting in the configuration depicted in Figure 7. The models L^2 and L^4 are computed according to L_0, L_1 , and d .

The filtering algorithm can be divided into two parts: searching for expected models and computing missing models.

The search for the expected models, lines 4 to 8 of Algorithm 4, is based on the assumption that between two iterations the robot will not move significantly, i.e., the rows are mostly at the same place as they were for the prior iteration. It means that the new models should be similar to the previous ones. Thus, from the new model set \mathcal{L} provided by the line finding algorithm, we search all the models that are close enough (according to a threshold) to the previous filtered ones $\mathcal{L}^{n,-1}$. If a match is found, this new model is added to the new filtered set \mathcal{L}^n . Note that if during this step no match is found, the previous filtered models are kept ($\mathcal{L}^n = \mathcal{L}^{n,-1}$). If this happens for several iterations then the robot is considered lost and stops navigating.

Once all the models from \mathcal{L} have been tested, some expected filtered models L^i may not have a match in the model set \mathcal{L} (at most $n - 1$). That is, they have to be computed regarding the found filtered models and the row distance d , lines 9 to 12 of Algorithm 4. The computation of a missing model L^i is done as

$$L^i : f(x) = \bar{a}x + b^{i_{best}} + d \cdot (i_{best} - i) \cdot \sqrt{\bar{a}^2 + 1}, \quad (15)$$

where \bar{a} is the average slope of the found filtered models

$$\bar{a} = \frac{\sum_{L^i \in \mathcal{L}^n} a^i}{\sum_{L^i \in \mathcal{L}^n} 1}, \quad (16)$$

and where i_{best} is the index of the best match

$$i_{best} = \arg \min_i \|L^i - L^{i-1}\|. \quad (17)$$

At the end of this filter step we have n models $\mathcal{L}^n = \{L^1, \dots, L^n\}$ that should be consistent with the crop and the robot configuration. These are the models that are considered by the controller presented in the next section.

3.3. Fuzzy Controller

A fuzzy controller is a classical approach when dealing with a mobile robot control that can be applied to agricultural robots [12,15]. The objective of the controller detailed in the latter is, according to the filtered models \mathcal{L}^n , to compute the needed wheel speeds for the robot to move between the rows. The fuzzy controller presented here has three steps:

- Fuzzification: transforms the inputs into fuzzy inputs (Section 3.3.1);
- Rule evaluation: defines how the inputs impact the outputs (Section 3.3.2);
- Defuzzification: from the fuzzy outputs, generated by the fuzzy inputs and the rules, defines a non-fuzzy output (Section 3.3.3).

3.3.1. Inputs of the Controller

From the filtered models \mathcal{L}^n we are only interested in the model L^{left} that is directly to the left of the robot and the model L^{right} that is directly to the right of the robot. For instance, in the example depicted in Figure 8, $L^{left} = L^2$ and $L^{right} = L^3$.

The orientation of the robot in the field is computed as:

$$\theta_r = \frac{\sum_{L^i \in \mathcal{L}^n} \arctan(a^i)}{n} \quad (18)$$

The position of the robot between the models L^{left} and L^{right} is defined as

$$x_r = \begin{cases} \frac{|b^{left}|}{|b^{right}|} - 1 & \text{if } |b^{left}| < |b^{right}| \\ 1 - \frac{|b^{left}|}{|b^{right}|} & \text{otherwise} \end{cases}, \quad (19)$$

These are the inputs of the fuzzy controller (x_r and θ_r). To proceed to the fuzzification, the membership functions depicted in Figure 9 are considered. For instance, a position of -0.25 will be considered as 0.5 left and 0.5 center.

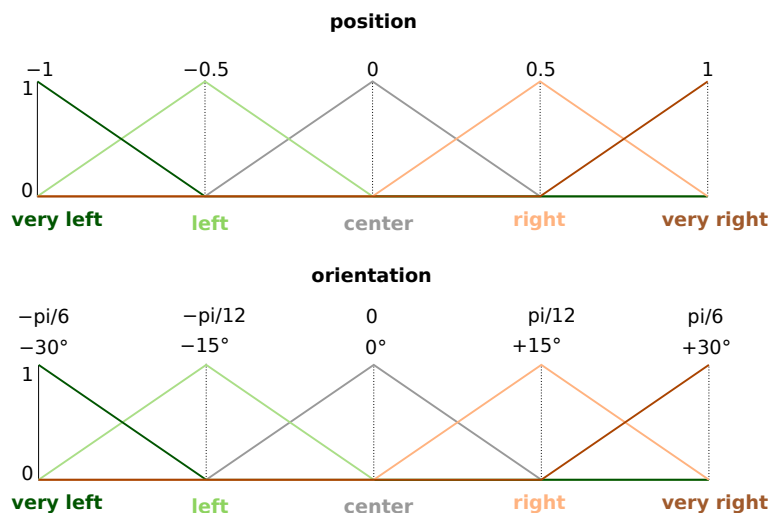


Figure 9. Input membership functions.

3.3.2. The Rules

Before defining the rules, it is necessary to define the fuzzy output membership functions. These functions are the same for the left and the right wheels (note that the considered robot is a two-wheeled differential robot) and are depicted in Figure 10.

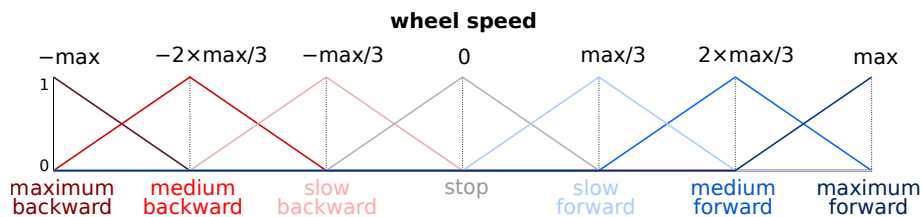


Figure 10. Output membership functions. Note that the left wheel speed and the right wheel speed have the same membership function, and that “max” means the maximal possible speed value.

A fuzzy controller’s last step is to define its rules. These rules aim to keep the robot at the center of two rows, and parallel to them. Figure 11 shows a graphical representation of the defined rules and Table 1 details them.

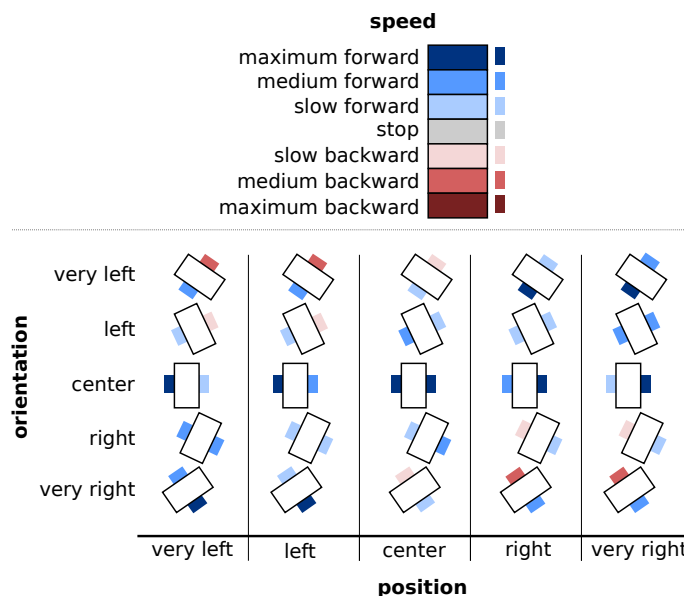


Figure 11. Rule chart. The detailed rules for the left and right wheel speeds according to the position and orientation of the robot.

Table 1. A more classical representation of the controller rules. L: left, R: right, V: very, C: center, 1: slow, 2: medium, 3: maximum, F: forward, B: backward.

Left Wheel Speed						
		Position				
		VL	L	C	R	VR
Orientation	VL	2 F	2 F	1 F	3 F	3 F
	L	1 F	1 F	2 F	1 F	2 F
	C	3 F	3 F	3 F	2 F	1 F
	R	2 F	1 F	1 F	1 B	1 B
	VR	2 F	1 F	1 B	2 B	2 B
Right Wheel Speed						
		Position				
		VL	L	C	R	VR
Orientation	VL	2 B	2 B	1 B	1 F	2 F
	L	1 B	1 B	1 F	1 F	2 F
	C	1 F	2 F	3 F	3 F	3 F
	R	2 F	1 F	2 F	1 F	1 F
	VR	3 F	3 F	1 F	2 F	2 F

3.3.3. Operators Summary

The considered operators for the fuzzy controller are as follows:

- AND operator: minimum;
- OR operator: maximum;
- Implication method: Algebraic product;
- Aggregation method: maximum;
- Defuzzification method: the centroid method. This corresponds to the “center of mass” of the results.

4. Simulation and Results

To test the algorithms under several controlled environments, a simulation has been designed based on ROS middle-ware and Gazebo robot simulation. These tools are widely used in the robotics community, and thus in agricultural robotics [16–18]. It can be noticed that all the source code of this simulation can be downloaded from GitHub [7]. This section presents the developed simulation as well as the methodology used to test the algorithms and the results of the conducted tests.

4.1. The Simulation

Figure 12 presents an overview of the simulation. Gazebo is used to simulate the physics of the system: it generates the LiDAR measurements from the environment and the robot’s pose, and moves the simulated robot according to the wheel speed.

ROS nodes (i.e., programs) were developed to implement the line finding algorithms and the control algorithm.

To handle the fuzzy controller, the open source FuzzyLite C++ library was used [19].

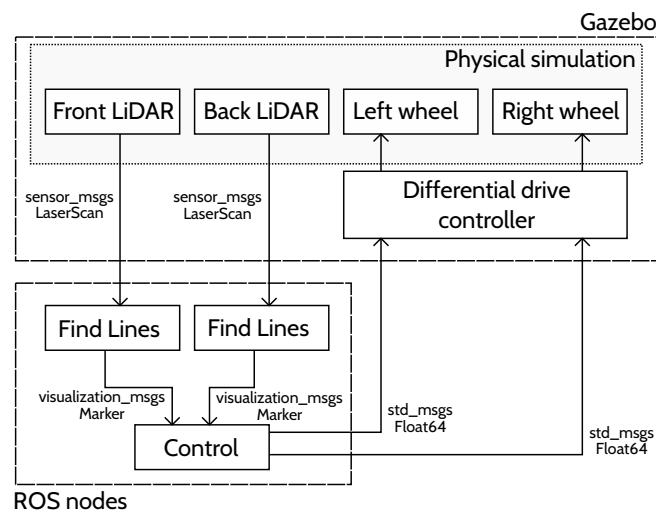


Figure 12. The simulator overview.

4.1.1. The Robot

The simulated robot is a two-wheeled differential robot, with two caster wheels for stability reasons. Figure 13 depicts the considered robot.

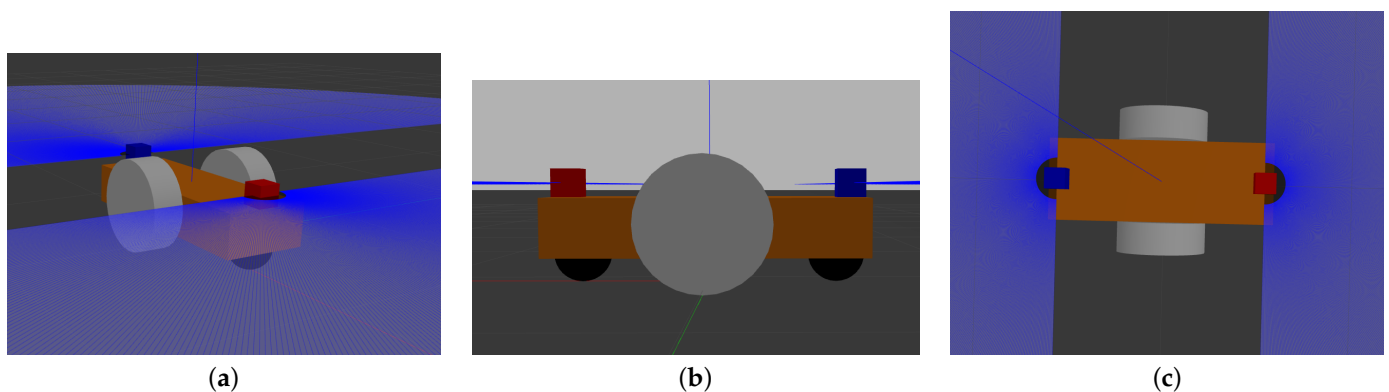


Figure 13. The simulated robot: the white wheels are the differential wheels, the black balls are the caster wheels and the blue/red boxes are the LiDARs. Note that the blue rays depict the LiDAR measurements. (a) General view; (b) Side view; (c) Top view.

The robot is equipped with two LiDAR sensors, one at the front and one at the back of the robot. Both LiDAR characteristics are:

- Update rate: 40 Hz;
- Number of measurements: 360;
- Minimum angle: $-\frac{\pi}{2}$;
- Maximum angle: $\frac{\pi}{2}$;
- Minimum range: 0.1 m;
- Maximum range: 4 m;
- Resolution: 0.01 m;
- Measurement noise: a Gaussian noise with a 0 mean and a 0.01 standard deviation is considered

$$f_{noise}(x) = \frac{1}{0.01\sqrt{2\pi}} e^{-\frac{1}{0.0002}x^2}. \quad (20)$$

Two LiDAR sensors are used so that the robot will have the same amount of information from its front as from its back. This symmetry helps to handle the change of row: indeed, the robot still detects the crops when moving out of a row as the *back* LiDAR sensor still detects the plants behind it. Furthermore, the robot does not have to turn around when changing rows, it just has to go *backwards*.

4.1.2. The Simulated Environments

Four fields were designed to test the algorithms. From the *easiest* to the *hardest*, the set-ups are:

- Crop 1, Figure 14a. This corresponds to the easiest configuration, which is a five-row crop, with equal rows, proper plant positioning, and without any weed;
- Crop 2, Figure 14b. In this configuration weeds are still not considered. Nevertheless, the plants are randomly spaced among the rows;
- Crop 3, Figure 14c. This is the first configuration with weeds (depicted as 50 red dots on the Figure 14);
- Crop 4, Figure 14d. This corresponds to the hardest configuration, which is 5 uneven rows with holes and weeds (100 in this case).

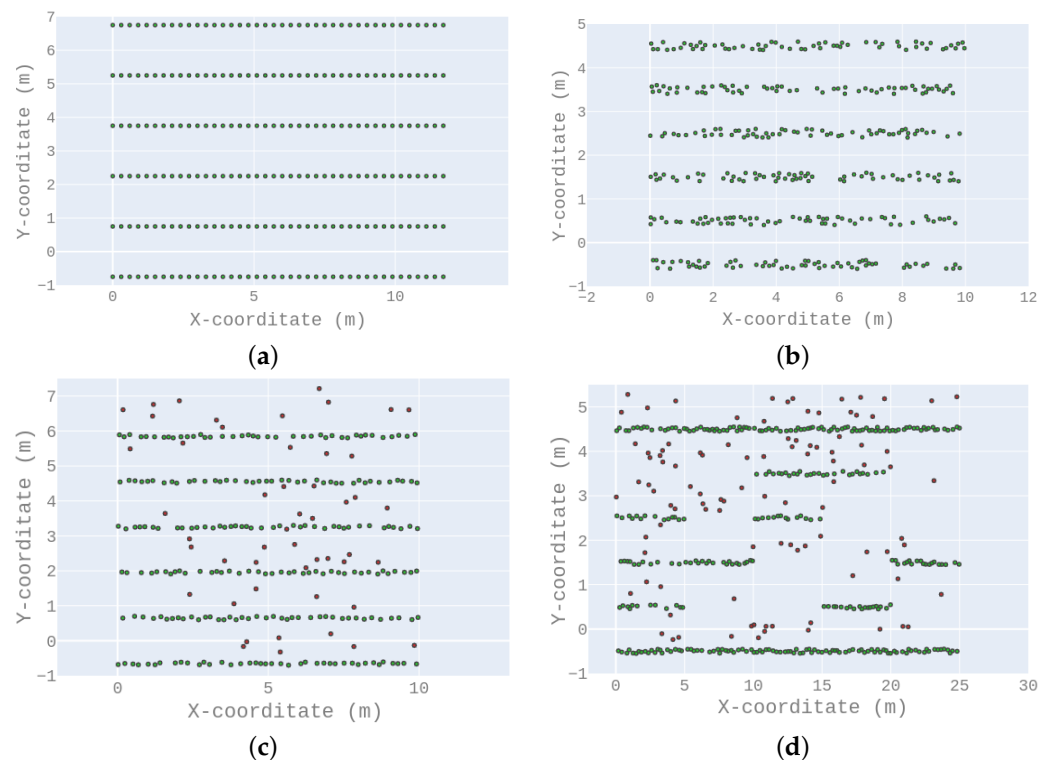


Figure 14. Simulated test crops. The green dots represent crops while the red dots represent weeds. (a) Crop 1; (b) Crop 2; (c) Crop 3; (d) Crop 4.

It can be noticed that the distances between the rows are not the same for all the simulated crops, but remain the same inside a crop.

4.2. Methodology

The following experiments were designed to test two things:

- How the presented refinements affect the line finding results;
- Considering the presented navigation algorithm: if the robot is able to autonomously navigate in the fields and how reliable the navigation is.

To do that, the same control approach (described in Section 3) has been tested with six different line finding algorithms: The two published algorithms Pearl [9] and Ruby [8], and the four proposed Ruby refinements (Section 2.2), which are Ruby Genetic (RG), Ruby Genetic One Point (RGOP), Ruby Genetic One Point Positive/Negative (RGOPPN), and Ruby Genetic One Point Positive Negative Infinity (RGOPPNI).

These six navigation algorithms (a navigation algorithm is the association of a line finding algorithm with the control approach) were tested in the four previously described environments (Figure 14).

A test run was done as follows: given a crop and a navigation algorithm, the robot is initially placed at the (0, 0) position of the environment and oriented according to the rows (as shown in Figure 15). Then the robot has to autonomously move through the rows until it reaches the end of the fifth row. Figure 15 presents an example of successful trajectory in the crop 3 environment. Before each run, the robot only has three pieces of information:

- It is at the beginning of the first row between two plant rows;
- It has to navigate through five rows;
- The first new row will be on the left.

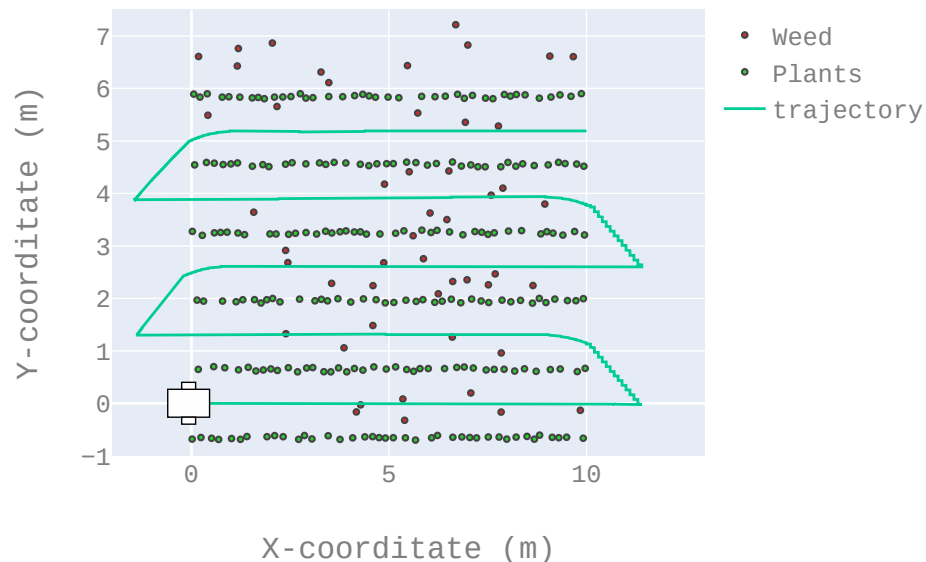


Figure 15. Example of trajectory: the robot starts at the position (0, 0), parallel to the crops, as depicted in the figure. Then it has to reach the end of the fifth row without running over the plants. This figure depicts an example of a successful trajectory.

Aside from this, it does not know the number of outliers, the distance between the rows, nor the length of the rows.

It can be noticed that for all the runs, the algorithms constants and thresholds do not change, even when changing the field environment. Furthermore, when starting a new run, all the information gathered during the previous run (e.g., the distance d), is removed.

The full experiment is presented in Algorithm 5: all the algorithms are tested five times over all the crops.

Algorithm 5: Test plan.

```

1 Data: control_algo, constants
2 find_lines_algos = {Ruby, RG, RGOP, RGOPPN, RGOPPNi}_constants;
3 crops = {crop 1, crop 2, crop 3, crop 4};
4 for all find_lines  $\in$  find_lines_algos do
5   for all crop  $\in$  crops do
6     for  $i = 1, \dots, 5$  do
7       Run a test with find_lines and control_algo in crop;
8       Save the results ;
     end
   end
end

```

4.3. Experiment Results

Several criteria were considered to compare these algorithms.

- The ratio of successful runs, computed as the number of successful runs divided by the total number of runs (i.e., 5). A run is considered successful when the robot managed to reach the end of the fifth row without running over the “green” plants, i.e., the crop. Figure 15 presents an example of a successful run while Figure 16 presents an example of a failed run. Table 2 details the results according to this criterion.
- The mean square error when the robot navigated between two rows (not including the changing row maneuver). In a perfect situation, the robot should be exactly equidistant to the direct left and right rows at any time (once again, not including the changing row maneuver). Table 3 details the results according to this criterion.
- At the end of a row, the LiDAR sensor will have fewer points than it would have between two rows (because it detects fewer plants as it is at the end of the crop). That is, in order to have a fair representation of the following criteria, they were obtained over the 500 first iterations only (that is, before reaching the end of the first row).
 - The average execution time of the line finding algorithm. As the control algorithms are the same for all the tested navigation approaches, it is only relevant to test the line finding execution time (the only part that differs from one navigation algorithm to another). Table 4 details the results according to this criterion.
 - The average number of points processed by the line finding algorithms. While some algorithms consider the raw point data set \mathcal{Z} , others use the filtered one \mathcal{Z}^* (Section 2.2.2). That is, the average number of points, presented in Table 5, allows us to verify that for the 500 first iterations the line finding algorithms considered the same number of points (all the \mathcal{Z} and all the \mathcal{Z}^* have consistent sizes).
 - The average execution time per 100 points. Since the line finding algorithm does not consider the same number of points (\mathcal{Z} versus \mathcal{Z}^*), the execution time presented in Table 4 may not be comparable regarding only the line finding processes. That is, Table 6 provides an execution time normalized on the processing of 100 points.

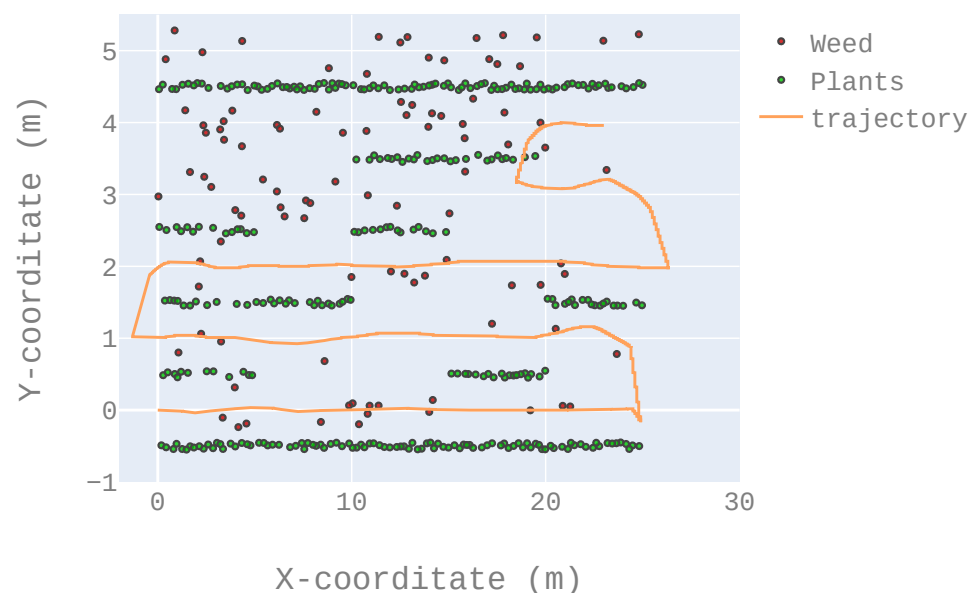


Figure 16. An example of a failed trajectory: the robot reaches the end of the fifth row but runs over the crop plants.

Table 2. The ratio of successful runs regarding the total number of tries (five, in this case). A result of 1 means that the algorithm never failed, a result of 0 means that the robot never reached the end of the field without crushing crop plants.

	Successful Runs (%)				
	Crop 1	Crop 2	Crop 3	Crop 4	Mean
Ruby	0.8	0.8	1	0.2	0.75
RG	0.8	0.8	0.4	0	0.5
RGOP	1	1	1	1	1
RGOPPN	1	1	1	0.8	0.95
RGOPPNI	1	1	1	0	0.75

Table 3. The positioning error that the robot made when moving between two crop rows. In a perfect situation, the robot should be exactly in the middle of the two crop rows, and thus should have an error of 0.

	Mean Squared Error (mm ²)				
	Crop 1	Crop 2	Crop 3	Crop 4	Mean
Ruby	2900	360	120	5500	2200
RG	8700	3100	89	9640	5300
RGOP	65	430	67	750	320
RGOPPN	300	700	140	2240	840
RGOPPNI	67	310	260	60,000	15,000

Table 4. The average execution time for the line finding algorithms (LiDAR data to models) during the 500 first iterations of each run.

	Average Execution Time (ms)				
	Crop 1	Crop 2	Crop 3	Crop 4	Mean
Ruby	4.14	21.15	9.39	9.61	11.07
RG	5.25	8.12	6.56	6.11	6.51
RGOP	2.35	3.44	2.9	2.7	2.84
RGOPPN	1.58	3.57	2.8	2.37	2.58
RGOPPNI	3.3	6.37	4.42	4.05	4.5

Table 5. The average number of points considered by the line finding algorithms during the 500 first iterations of each run.

	Average Number of Points				
	Crop 1	Crop 2	Crop 3	Crop 4	Mean
Ruby	91	179.3	122.54	120.36	128.3
RG	91.16	178.1	124.53	120	128.4
RGOP	40.1	73.12	54.18	47.16	53.64
RGOPPN	40	73.15	54.19	47.12	53.61
RGOPPNI	40.08	73.96	54.19	47.17	53.85

Table 6. The average execution time per 100 points for the line finding algorithms (LiDAR data to models) during the 500 first iterations of each run.

	Execution Time /100 Points (ms)				
	Crop 1	Crop 2	Crop 3	Crop 4	Mean
Ruby	4.54	11.79	7.66	7.98	7.99
RG	5.75	4.55	5.26	5.09	5.11
RGOP	5.86	4.7	5.35	5.7	5.4
RGOPPN	3.95	4.88	5.16	5.02	4.75
RGOPPNI	8.2	8.61	8.15	8.58	8.38

Regarding the data presented in the tables, it appears that filtering the input data with a genetic approach (Ruby Genetic One Point) increases the robustness and improves the precision and the computation time. As a matter of fact, combining the Ruby Genetic One Point line finding algorithm with the control algorithm provides a 100% success rate over the five experimental crops (Table 2). It also appears to be the most stable approach (Table 3). However, adding the positive/negative filtering and allowing a point to be associated with several models at the same time results in worse performance than RGOP (Table 2).

The number of processed points is consistent with what is expected: the first algorithms named Ruby and RG consider the same number of points, while the algorithms RGOP, RGOPPN, and RGOPPNI use fewer points (due to the filter point set \mathcal{Z}^*). This can be seen in Table 5.

The considered sensors have a rate of 40 Hz (Section 4.1.1), thus they provide a data set every 25 ms. According to Table 4, the RGOP approach needs less than 3 ms to process a data set. That is, it can be used in real time with this sensor, as can all the other algorithms.

It must be emphasized that the crops (more precisely the weed positions) were randomly generated (the source code for the generation of the simulated crops is available in Ref. [7]). This suggests that this navigation approach will provide reliable results in a wide range of crop configurations. Even if the approach needs to be tested in real conditions, the simulation results are more than encouraging.

5. Conclusions

In this paper a novel approach to extract crop rows from LiDAR data is presented. This approach has been coupled with a fuzzy controller to propose a complete navigation algorithm.

A simulator was developed to test this approach in various field situations, and the results are positive: according to the simulated environments, the approach appears to perform well even when the crop has uneven rows with holes and weeds.

Future work will be focused on implementing this approach into an experimental platform and performing navigation in actual fields. Furthermore, cameras will be added to the loop in order to limit the current assumptions about the field.

Author Contributions: Conceptualization, R.G. and F.M.; methodology, R.G. and G.F.O.; software, G.F.O.; validation, R.G. and F.M.; formal analysis, G.F.O.; investigation, G.F.O.; resources, R.G. and F.M.; data curation, G.F.O.; writing original draft preparation, G.F.O. and F.M.; writing review and editing, R.G.; visualization, R.G.; supervision, R.G. and F.M.; project administration, R.G. and F.M.; funding acquisition, R.G. and F.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the French “Pays de la Loire” region through the Atlantic 2020 program, the convention number is 2015-08722 for the ENGRAIS project.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The simulator and the experimental environments are available in the Github repository: <https://github.com/PolytechAngersMecatronicClub/istiaENGRAIS> (accessed on 15 November 2022).

Conflicts of Interest: The authors declare no conflict of interest.

Nomenclature

$p \in \mathbb{R}^2$	A 2D point (provided by the LiDAR sensor for instance), $p = (x_p, y_p)$
\mathcal{N}	The set of neighboring points such that an element $(p, q) \in \mathcal{N}$ corresponds to two points p and q in the same neighborhood
L_j	A model (a line for the crop navigation case), $L_j : f(x) = a_jx + b_j$
L^j	The j^{th} filtered model, $L^j : f(x) = a^jx + b^j$
L^{left}	The filtered model closest to the left of the robot, $L^{\text{left}} : f(x) = a^{\text{left}}x + b^{\text{left}}$
L^{right}	The filtered model closest to the right of the robot, $L^{\text{right}} : f(x) = a^{\text{right}}x + b^{\text{right}}$
L^{j-1}	The j^{th} filtered model of the previous iteration
L_\emptyset	The empty model (for the outliers)
$\mathcal{L}_i = \{L_j\}$	A set of models
\mathcal{L}^{-1}	A set of models from the previous call of line finding
\mathcal{L}^n	A set of n filtered models, $\mathcal{L}^n = \{L^1, L^2, \dots, L^n\}$
$\mathcal{L}^{n,-1}$	The set of n filtered models of the previous iteration
$L(p)$	The model associated with the point p
$\mathcal{E}(\mathcal{L})$	The total energy of the model set \mathcal{L}
$\mathcal{E}_{\mathcal{N}}(\mathcal{L})$	The penalty energy of associating close points to different models
$\mathcal{E}_\emptyset(\mathcal{L})$	The outlier energy of the model set \mathcal{L}
$\mathcal{E}_L(L_j)$	The energy of the model L_j
$P(L_j)$	The set of points that are associated with the model L_j , $P(L_j) = \{p \in \mathcal{Z} L(p) = L_j\}$
$\mathcal{Z} = \{p\}$	The set of all the points
\mathcal{Z}^*	A set of points based on \mathcal{Z} , such that all the points that are close enough in \mathcal{Z} are fused together
$\ X\ $	The Euclidean distance of the X expression
ψ_j	Number of models parallel to model L_j
$\rho, \lambda, \zeta, \tau, \alpha, \beta$	Constants, heuristically chosen
$\delta_{\neq}(L(p), L(q))$	A function that equals 1 if the point p is not associated with the same model as the point q , 0 otherwise
w_{pq}	A weight associated with $\delta_{\neq}(L(p), L(q))$ in the computation of $\mathcal{E}_{\mathcal{N}}(\mathcal{L})$
$\delta_{\parallel}(L_j, L_k)$	A function that equals 1 if the models are parallel, 0 otherwise
φ_j	Fitness criteria for Ruby Genetic
x_r	The position of the robot between two rows
θ_r	The orientation of the robot in the field
d	The distance between the rows (constant over the all crop)
i_{best}	The index of the previous filter that best match the current expected i^{th} one
b^{best}	The b value of the previous filter that best match the current expected i^{th} one

References

- Xiong, Y.; Ge, Y.; Liang, Y.; Blackmore, S. Development of a prototype robot and fast path-planning algorithm for static laser weeding. *Comput. Electron. Agric.* **2017**, *142*, 494–503. [CrossRef]
- Zhang, Q.; Chen, M.S.; Li, B. A visual navigation algorithm for paddy field weeding robot based on image understanding. *Comput. Electron. Agric.* **2017**, *143*, 66–78. [CrossRef]
- Yu, Y.; Zhang, K.; Yang, L.; Zhang, D. Fruit detection for strawberry harvesting robot in non-structural environment based on Mask-RCNN. *Comput. Electron. Agric.* **2019**, *163*, 104846. [CrossRef]
- Arad, B.; Balendonck, J.; Barth, R.; Ben-Shahar, O.; Edan, Y.; Hellström, T.; Hemming, J.; Kurtser, P.; Ringdahl, O.; Tielen, T.; et al. Development of a sweet pepper harvesting robot. *J. Field Robot.* **2020**, *37*, 13. [CrossRef]
- Bonadies, S.; Gadsden, S.A. An overview of autonomous crop row navigation strategies for unmanned ground vehicles. *Eng. Agric. Environ. Food* **2019**, *12*, 24–31. [CrossRef]
- Kanagasingham, S.; Ekpanyapong, M.; Chaihan, R. Integrating machine vision-based row guidance with GPS and compass-based routing to achieve autonomous navigation for a rice field weeding robot. *Precis. Agric.* **2019**, *21*, 1–25. [CrossRef]
- Oliveira, G.F.; Guyonneau, R. Simulation and Control of an Agricultural Robot through an Unknown Field Using LIDAR Sensors. Available online: <https://github.com/PolytechAngersMecatroniqueClub/istiaENGRAIS> (accessed on 12 December 2020).
- Malavazi, F.B.; Guyonneau, R.; Fasquel, J.B.; Lagrange, S.; Mercier, F. LiDAR-only based navigation algorithm for an autonomous agricultural robot. *Comput. Electron. Agric.* **2018**, *154*, 71–79. [CrossRef]
- Isack, H.; Boykov, Y. Energy-based geometric multi-model fitting. *Int. J. Comput. Vis.* **2012**, *97*, 123–147. [CrossRef]
- Davis, L. *Handbook of Genetic Algorithms*; Thomson Publishing Group: Washington, DC, USA, 1991.
- Choi, K.H.; Han, S.K.; Han, S.H.; Park, K.H.; Kim, K.S.; Kim, S. Morphology-based guidance line extraction for an autonomous weeding robot in paddy fields. *Comput. Electron. Agric.* **2015**, *113*, 266–274. [CrossRef]

12. Xue, J.; Zhang, L.; Grift, T.E. Variable field-of-view machine vision based row guidance of an agricultural robot. *Comput. Electron. Agric.* **2012**, *84*, 85–91. [[CrossRef](#)]
13. Åstrand, B.; Baerveldt, A.J. A vision based row-following system for agricultural field machinery. *Mechatronics* **2005**, *15*, 251–269. [[CrossRef](#)]
14. Winterhalter, W.; Fleckenstein, F.V.; Dornhege, C.; Burgard, W. Crop row detection on tiny plants with the pattern hough transform. *IEEE Robot. Autom. Lett.* **2018**, *3*, 3394–3401. [[CrossRef](#)]
15. Meng, Q.; Qiu, R.; He, J.; Zhang, M.; Ma, X.; Liu, G. Development of agricultural implement system based on machine vision and fuzzy control. *Precision Agriculture Comput. Electron. Agric.* **2015**, *112*, 128–138. [[CrossRef](#)]
16. de Santos, F.B.N.; Sobreira, H.M.P.; Campos, D.F.B.; de Santos, R.M.P.M.; Moreira, A.P.G.M.; Contente, O.M.S. Towards a Reliable Monitoring Robot for Mountain Vineyards. In Proceedings of the 2015 IEEE International Conference on Autonomous Robot Systems and Competitions, Vila Real, Portugal, 8–10 April 2015; pp. 37–43. [[CrossRef](#)]
17. Habibie, N.; Nugraha, A.M.; Anshori, A.Z.; Ma'sum, M.A.; Jatmiko, W. Fruit mapping mobile robot on simulated agricultural area in Gazebo simulator using simultaneous localization and mapping (SLAM). In Proceedings of the 2017 International Symposium on Micro-NanoMechatronics and Human Science (MHS), Nagoya, Japan, 3–6 December 2017; pp. 1–7. [[CrossRef](#)]
18. Linner, T.; Shrikathiresan, A.; Vetrenko, M.; Ellmann, B.; Bock, T. Modeling and operating robotic environment using Gazebo/ROS. In Proceedings of the 28th International Symposium on Automation and Robotics in Construction (ISARC2011), Seoul, Korea, 29 June–2 July 2011; pp. 957–962.
19. Rada-Vilela, J. FuzzyLite 6.0: A Fuzzy Logic Control Library in C++. Available online: <https://fuzzylite.com/cpp/> (accessed on 12 December 2020).