LiDAR-Only Crop Navigation for Symmetrical Robot

Gabriel Freitas Oliveira, Franck Mercier, Rémy Guyonneau

LiDAR-Only Crop Navigation for Symmetrical Robot

Gabriel FREITAS OLIVEIRA, Franck MERCIER, Remy GUYONNEAU

LARIS - System Engineering Research Laboratory of Angers - University of Angers, FRANCE

Abstract

This paper presents a robust navigation approach for autonomous agricultural robots based on LiDAR data. This navigation approach is divided into two parts: a find lines algorithm and a control algorithm. The paper proposes several find lines 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 those 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 based on ROS middle-ware and Gazebo (the source codes of that 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

1. Introduction

1.1. Context

The development of robotic tools for agriculture is a growing field. Developments are 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.

In order for a mobile robot, agricultural or not, to be autonomous it must be able to move in its environment. For an agricultural application, in market gardening for instance, 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 work considers GPS or camera data for navigation [5, 6]. But those sensors do not work in all conditions. Indeed, at the edge of a forest or in a greenhouse, the GPS receiver is degraded. Depending on the position of the sun, camera images may not allow robust navigation. In this context, we hypothesize that only the use of data from several sensor technologies can provide robust autonomy. This is why, without aiming to compete with the previously mentioned approaches, this paper proposes an approach based on LiDAR[1] data. This sensor technology has the advantage of being less dependent to ambient light (as opposed to a camera) and does not require satellite communication (as opposed to a GPS). Thus it can be used when camera or GPS fail to provide workable data. The work presented in this paper can be seen as a navigation brick which will be fused with others in future work.

A mentioned just above, the work presented in this paper aims to propose a robust approach for the navigation of a symmetrical robot based on LiDAR sensor data only. This work is a direct continuation of the work presented in [7]. The originalities presented here are mainly:

- The improvement of the existing find lines algorithm (Figure 2 introduces the find lines end). As we were adapting the find lines algorithm named Ruby in [7] we significantly improved the approach by offering new refine-

*Remy GUYONNEAU

Email address: `firstname.LASTNAME@univ-angers.fr` (Gabriel FREITAS OLIVEIRA, Franck MERCIER, Remy GUYONNEAU)

[1]Light Detection And Ranging

March 6, 2020

ments (detailed in Section 2). The interested reader can got to Section 4 for the algorithm behavior comparison;

- The setup of filters and a fuzzy controller to have a full navigation workflow. The previous work mainly focused on the find line algorithm. At the end, the robot were only able to move straight forward between two rows. The control algorithm presented in this paper (Section 3) allows the simulated robot robot to move autonomously in the entire field. Note that considering a symmetrical robot eases the row changing step;

- The development of a new simulation based on ROS and Gazebo. It has been noticed that the work presented in [7], mainly the simulator, was not easily reusable, as it was a proprietary software developed by the Naio Technology french company. Furthermore, as this simulator was not designed for academic experiments, the algorithm comparisons were done by processing screenshots of the simulated trajectory results. To overcome those limitations, it was decided to implement a new simulation (the ones presented in this paper) based on ROS[2] middle-ware and Gazebo (Section 4). Note that all the new simulator source codes and documentation are available in [8].

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 find lines algorithm and a control algorithm, as depicted in Figure 1.

To be able to navigate among the crops, it is needed to identify the crop rows from the LiDAR data. This step correspond to the find lines process of the approach and is depicted in Figure 2. The reader should note that it is assumed that the robot is equipped with 2D LiDAR sensor(s) and that the sensor(s) can detect the crop plants and the weeds (i.e. the LiDAR is low enough or the plants are high enough). In other words the plants have to be detected by the sensors otherwise this approach can not be applied...

[2]Robot Operating System

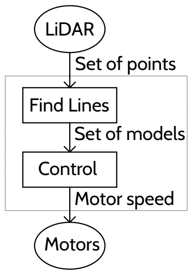


Figure 1: Overview of the approach: the LiDAR sensor provides points, from those points lines (models) are extracted and from those lines the motor speeds are deduced.

The find lines algorithm thus provides a set of models (i.e. a set of lines) that best fit the points detected by the sensor(s). 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 presents several find lines algorithms while Section 3 presents the considered control algorithm. Section 4 presents the designed simulation and the results obtained when testing the algorithms. Finally Section 5 concludes this paper.

1.3. Notations

To ease the reading of this paper, Table 1 resumes the considered notations.

2. Find lines algorithms

As explained in the introduction, this paper is based on the work depicted in [7] and more precisely on the Ruby algorithm. For a better understanding, this so called Ruby algorithm is again presented here (Section 2.1). Then, the novelties brought to this algorithm are presented in Section 2.2.

2.1. 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

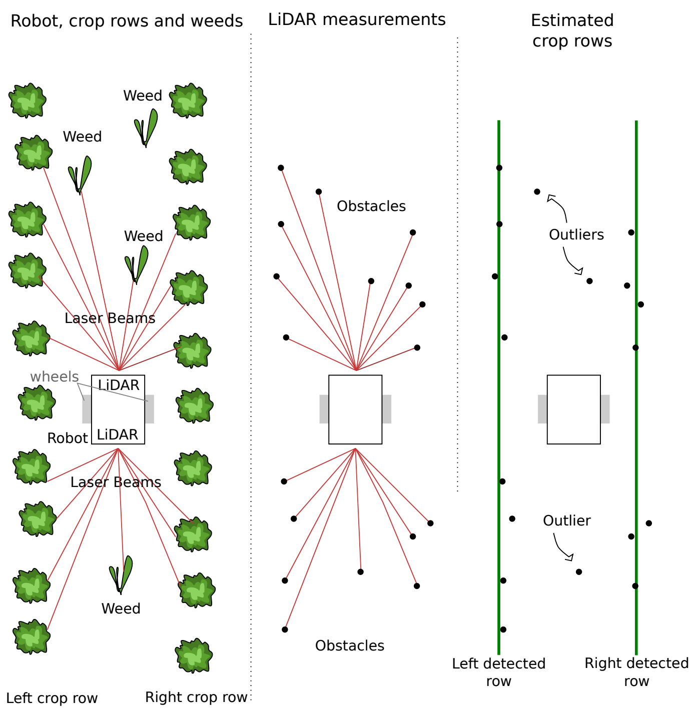


Figure 2: The find lines problem: The objective is to identify the crop rows from LiDAR data. The left part of the figure depicts a top view of the scene (a robot moving between two crop rows), the middle part depicts the data from the sensors (i.e. points corresponding to plants) and the right part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

$p \in \mathbb{R}^2$	A 2D point (provided by the LiDAR sensor for instance), $p = (x_p, y_p)$
L_j	A model (a line for the crop navigation case), $L_j : f_j(x) = a_j x + b_j$
$\mathcal{L}_i = \{L_j\}$	A set of models
$\mathcal{L}^{-1}$	A set of models from the previous call of find lines
$\mathcal{L}^n$	A set of filtered models, $\mathcal{L}^n = \{L^1, L^2, ..., L^n\}$
n	The number of filtered models
$L(p)$	The model associated to the point p
$\mathcal{E}(\mathcal{L}_i)$	The total energy of the model set $\mathcal{L}$
$\mathcal{E}_\mathcal{N}(\mathcal{L}_i)$	The penalty energy of associating close points to different models
$\mathcal{E}_\emptyset(\mathcal{L}_i)$	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 to the model L_j, $P(L_j) = \{p \in Z \| L(p) = L_j\}$
$Z = \{p\}$	the set of all the points
Z^*	a set of points based on Z, such that all the points that are close enough in Z are fused together
$\|\|X\|\|$	The euclidean distance of the X expression
ψ_j	Number of models parallel to model L_j
$\rho, \lambda, \zeta, \tau$	Constants, heuristically chosen
$\delta(\mathrm{X})$	A function that equals 1 if the X expression is true, 0 otherwise
$\mathcal{N}$	The set of neighbor points
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 crop
d	The distance between the rows (constant over the all crop)

Table 1: Notations

a function, called energy. In the later, a model $L_j : f_j(x) = a_j x + b_j$ corresponds to a line, and $\mathcal{L}_i$ depicts a set of models. A point p (an obstacle detected by the LiDARs) is associated to a model $L(p)$. Note that $L(p)$ could be the empty model $L_\emptyset$ (if p is an outliers for instance). All those notations are resumed in

Table 1.

2.1.1. *The energy function*

The considered energy function is described in Equation 1.

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

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

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

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

with $\sum_{p \in P(L_\emptyset)}$ the number of points in the current empty model $L_\emptyset \in \mathcal{L}_i$ and ρ a constant value chosen heuristically. 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, i.e. weeds).

The penalty energy aims at limited the fact that two close points (according to the euclidean distance) are associated to two different models. This energy is defined as

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

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

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

and

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

with $\|p - q\|$ the euclidean distance between the points p and q and ζ a constant chosen heuristically.

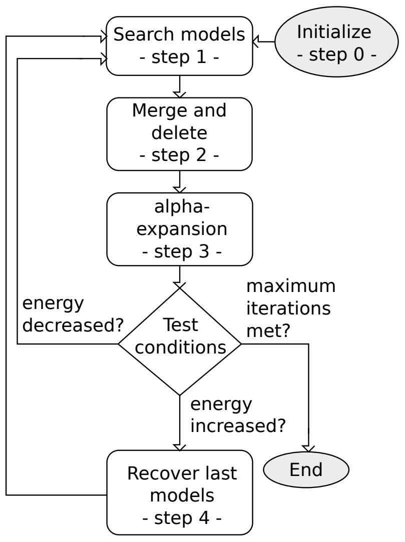


Figure 3: Overview of the Ruby method.

Finally, the energy of a model L_j is defined as

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

with $\|p - L_j\|$ the euclidean distance between a point p attached to the model L_j and that model.

2.1.2. *The Ruby algorithm*

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

step 0: Line 1 of the algorithm. It corresponds to the initialization. The initial set of models $\mathcal{L}_0$ is initialized with the one found at the previous find line computation $\mathcal{L}_i^{-1}$. Note that if it is the first call ever of find lines, the initial set of models is initialized with only an empty model: $\mathcal{L}_i^{-1} = \{L_\emptyset\}$ with $P(L_\emptyset) = Z$. 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...

step 1: Lines 3 to 7 of the algorithm. It picks 3 random points (p, q, z) in $L_\emptyset$ and computes the best model (linear regression) that fits those points. This creates a new model that is added to the model set. This process is then repeated

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

Algorithm 1: Ruby algorithm

until half of the outliers are attached to a model (or below a threshold).

step 2.1: Lines 8 to 15 of the algorithm, the merge part. It merges all the models that are too close to each other.

step 2.2: Lines 16 to 20 of the algorithm, the delete part. It removes all the models that do not met the constraint

$$\frac{\mathcal{E}_L(L_j)}{\sum_{p \in P(L_j)} \cdot \psi_j} < threshold, \tag{7}$$

where $\mathcal{E}_L(L_j)$ is the energy of the model L_j as described in Equation 6 and ψ_j is the number of models parallel to the model L_j. ψ_j is defined as

$$\psi_j = \sum_{L_k \in \mathcal{L}_i \setminus \{L_j, L_\emptyset\}} \delta(L_j // L_k), \tag{8}$$

with

$$\delta(L_j // L_k) \begin{cases} 1 & \text{if } |a_j - a_k| < threshold \\ 0 & \text{otherwise} \end{cases}, \tag{9}$$

a_j and a_k being the slopes of the models L_j and L_k.

step 3: Lines 21 to 27 of the algorithm. All the points are detached from their models and reattached to the closest model. Then the new energy is computed for the complete set of models.

step 4: Lines 28 to 30 of the algorithm. If the energy of the new set of models is worst (bigger) than the previous iteration, the previous set of models is kept instead of the new computed one.

2.2. *Ruby suggested refinements*

In the later are presented refinements for the Ruby algorithm. Those 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*

The Pearl and Ruby algorithms are very similar to genetic algorithms. To enhance this resemblance, the Ruby genetic refinement proposes to modify the delete part of the step 2 (Figure

3). In the classical Ruby approach a threshold is used (Equation 7) to decide if a model has to be deleted or not. Ruby genetic proposes a fitness criteria instead, named φ_j and defined as

$$\varphi_j = \frac{b_j^2 + \tau \cdot \mathcal{E}_L(L_j)}{\psi_j \cdot \sum\limits_{p \in P(L_j)}}, \tag{10}$$

with $\mathcal{E}_L(L_j)$ defined in Equation 6, ψ_j defined in Equation 8 and τ an heuristically chosen constant. This criteria allows to sort all the models, and only the best ones are kept, the others are deleted.

2.2.2. *Ruby Genetic One Point*

The idea of this refinement is based on to assumption that one plant of the crop will generate several LiDAR readings. Thus, the data that are close to each other can be resumed into one merged point as they may belong to the same plant. This leads to the computation of a new point set named Z^*. Algorithm 2 describes how it is done.

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

Algorithm 2: The computation of Z^*

For this Ruby Genetic One Point refinement, the input data is not longer Z as defined in Algorithm 1, but Z^*. The rest of the algorithm remains the same as for the Ruby genetic.

2.2.3. *Ruby Genetic One Point Positive / Negative*

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

$$P_{left}(L_\emptyset) = \{p \in P(L_\emptyset)\}|y_p < 0 \tag{11}$$

$$P_{right}(L_\emptyset) = \{p \in P(L_\emptyset)\}|y_p > 0 \tag{12}$$

Then the research for models is done into those two subsets. Thus, the step 1 of the Ruby algorithm (Figure3), i.e. lines 3 to 7 of Algorithm 1, becomes what is detailed in Algorithm 3. An example of this new model research approach is depicted in Figure 4.

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

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

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 to 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

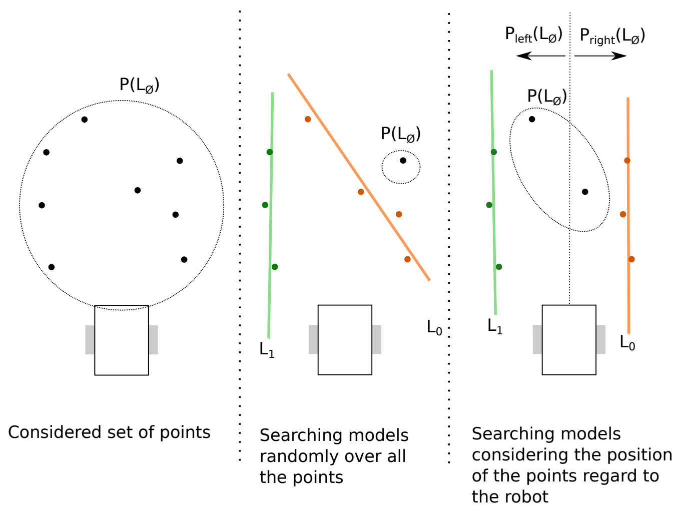

P(L∅)
P(L∅)
P(L∅)
Pleft(L∅)
Pright(L∅)
L1
L0
L1
L0
Considered set of points
Searching models randomly over all the points
Searching models considering the position of the points regard to the robot

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 is increased (right section of the illustration).

limitation this new refinement proposes to allows 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 (data set from the LiDAR), a control can to be processed based on those models. The control algorithm presented here can be divided into three main steps: an initialization step, a filtering step and a controller. Those steps are depicted in Figure 5.

3.1. Initialization

The idea of the overall approach is to allow the robot to autonomously navigation into a crop without any prior knowledge of the field. To overcome the lack of prior information (width and lenght of the rows), it is assumed that:

- The field is organized in straight lines, as it is the case most of the time [10, 11];

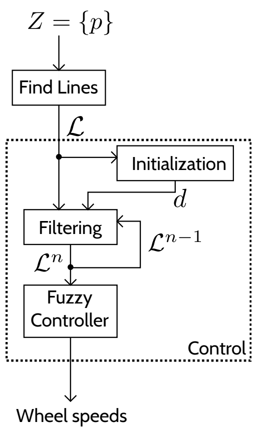


Figure 5: Details of the control approach.

- 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 parallel with the crop rows.

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

To that aim, 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 is found, the distance between the rows is computed as follow (Figure 6):

$$\cos(\theta) = \frac{d}{|b_1 - b_2|}$$
$$d = \cos(\theta) \cdot |b_1 - b_2|$$
$$d = \cos(\arctan(a_1)) \cdot |b_1 - b_2|$$
$$d = \cos(\arctan(a_2)) \cdot |b_1 - b_2|$$

with

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

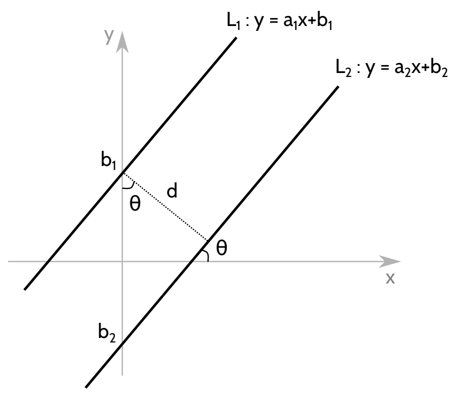


Figure 6: Computing the distance between two parallel models L_1 and L_2. Note that $a_1 = a_2$ for the models to be parallel.

it can be concluded that

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

To be more robust, this distance computation is done several times (according to several results of the find lines 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 and will be use for the robot's navigation.

3.2. Filtering

The find lines algorithm, as detailed in Section 2, provide models (lines) according to a point set (LiDAR data). But most of time some returned models do not correspond to effective rows in the field (Figure 7). It is then needed to filter the provided models in order to only keep the ones that are consistent with what it is known of the field (previously computed models and the distance between the rows). This filter is presented in Algorithm 4 and detailed in the later.

The required data are:

$\mathcal{L}$: the set of models provided by the find lines algorithm;

$\mathcal{L}^{n,-1}$: the filtered model set of the previous iteration. Note that $\mathcal{L}^{n,-1} = \{L^{1,-1}, L^{2,-1}, ..., L^{n,-1}\}$ 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);

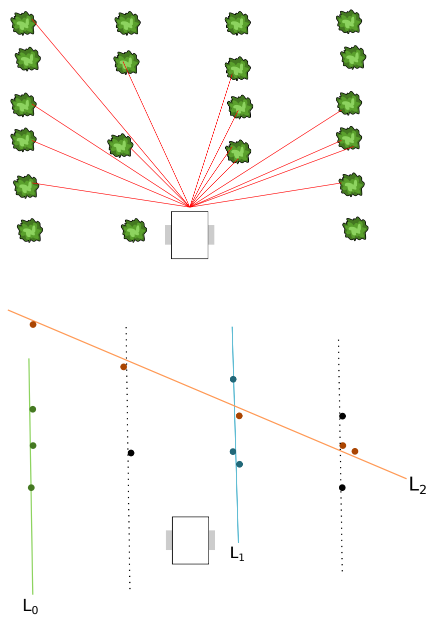


Figure 7: Assuming that from the top configuration, the find lines algorithm returns the models L_0, L_1 and L_2 (bottom part of the figure). The expected behavior of the filter is to keep the models L_0 and L_1 (removing the model L_2) and knowing the distance d between the rows, to compute the missing models (doted lines).

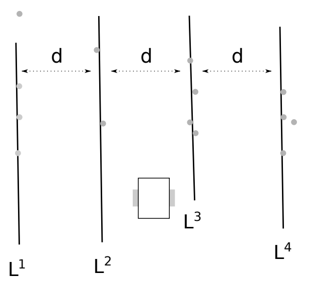


Figure 8: The filtered models $\mathcal{L}^n$, with $n = 4$, resulting of the configuration depicted in Figure 7. The models L^2 and L^4 are computed according to L_0, L_1 and d.

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

Algorithm 4: Model filtering

d: the distance between the rows, computed during the initialization step (Section 3.1).

The result of the filtering is a filtered model set $\mathcal{L}^n = \{L^1, ..., L^n\}$, 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 find lines 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.

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

The search for expected models, lines 2 to 9 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 than they were for the previous iteration. It means that the new models should be close to the previous ones. Thus, from the new model set $\mathcal{L}$ provided by the find lines 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 is kept ($\mathcal{L}^n = \mathcal{L}^{n,-1}$). If this keep happening 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 model L^i may not have found 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 10 to 14 of Algorithm 4. The computation of a missing model L^i is done as

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

with

$\overline{a}$ the average slope of the found filtered models

$$\overline{a} = \frac{\sum\limits_{L^i \in \mathcal{L}^n} a^i}{\sum\limits_{L^i \in \mathcal{L}^n}} \tag{16}$$

i_{best} the index of the best match

$$i_{best} = \arg\min_i ||L^i - L^{i,-1}|| \tag{17}$$

At the end of this filter step we have n models that should be representative of the crop and the robot configuration. Those 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 mobile robot control, and it can be considered when dealing with agricultural robot [11, 12]. The objective of the controller detailed in the later 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);

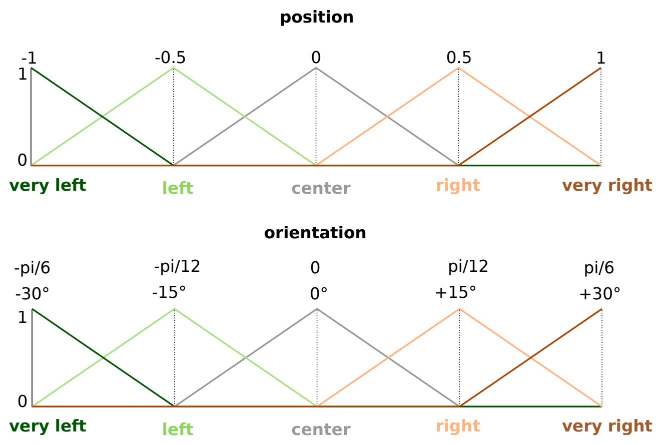


Figure 9: Input membership functions.

- 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 into 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 $L^{left} = L^3$.

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

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

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

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

Those 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.

3.3.2. The rules

Before defining the rules it is needed to define the fuzzy output membership functions. Those 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.

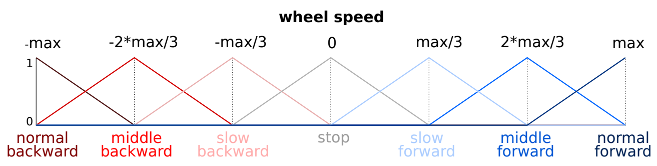


Figure 10: Output membership functions. Note that the left wheel speed and the right wheel speed have the same membership function, and that "max" means the maximal possible speed value.

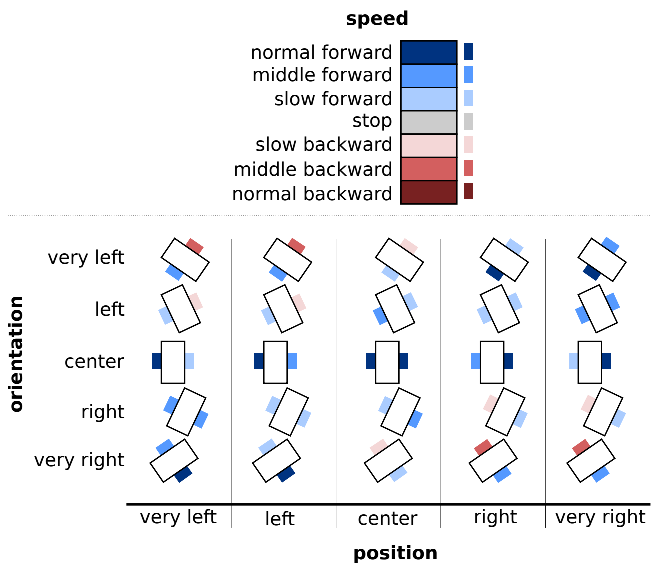


Figure 11: Rule chart. Here are detailed the rules for the left and right wheel speeds according to the position and orientation of the robot.

The last part of the fuzzy controller is to define its rules. Those 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 2 details them.

3.3.3. Operators summary

Here are listed the considered operators for the fuzzy controller:

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.

Left wheel speed						
		Position				
		VL	L	C	R	VR
Orientation	VL	MF	MF	SF	NF	NF
	L	SF	SF	MF	SF	MF
	C	NF	NF	NF	MF	SF
	R	MF	SF	SF	SB	SB
	VR	MF	SF	SB	MB	MB

Right wheel speed						
		Position				
		VL	L	C	R	VR
Orientation	VL	MB	MB	SB	SF	MF
	L	SB	SB	SF	SF	MF
	C	SF	MF	NF	NF	NF
	R	MF	SF	MF	SF	SF
	VR	NF	NF	SF	MF	MF

Table 2: Here is depicted a more classical representation of the controller rules. L: left, R: right, V: very, C: center, M: middle, S: slow, N: normal, F: forward, B: backward

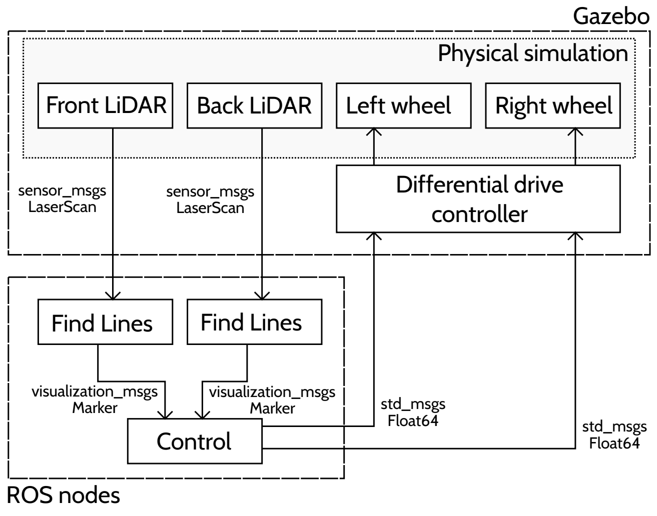


Figure 12: The simulator overview

4. Simulation and results

To test the algorithms in several controlled environments, a simulation has been designed based on ROS middle-ware and Gazebo robot simulation. Those tools are widely used in robotic community, and thus in agricultural robot [13, 14, 15]. It can be noticed that all the source code of this simulation can be download on github [8]. This section presents the developed simulation but also the methodology considered while testing the algorithms and the results of the conducted tests.

4.1. The simulation

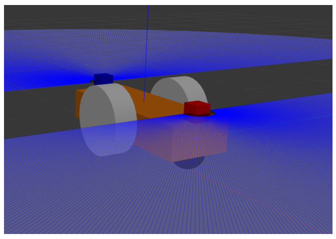
Figure 15 presents an overview of the simulation. Gazebo is used to simulate the physics of the system: it generates the LiDAR measurements from the environment and the robot's pose, and moves the simulated robot according to the wheel speed.

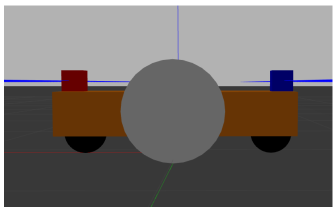
ROS nodes (i.e. programs) were developed to implement the find lines algorithms and the control algorithm.

To handle the fuzzy controller, the open source FuzzyLite C++ library was used [16].

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.


(a) General view


(b) Side view

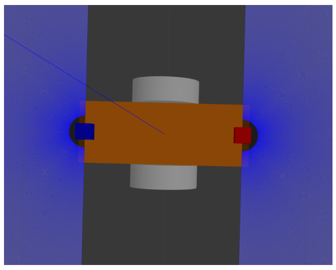

(c) Top view

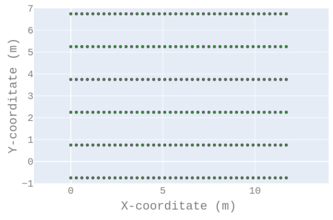
Figure 13: The simulated robot: the white wheels are the differential wheels, the black balls are the caster wheels and the blue/red boxes are the LiDARs. Note that the blue rays depict the LiDAR measurements.

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

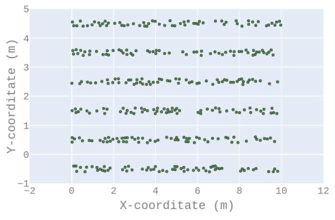
- update rate: 40 Hz
- number of measurements: 360
- minimum angle: $-\frac{\pi}{2}$
- maximum angle: $\frac{\pi}{2}$
- minimum range: 0.1 m
- maximum range : 4 m
- resolution : 0.01 m
- measurement noise: a Gaussian noise with a 0 mean and a 0.01 standard deviation is considered

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

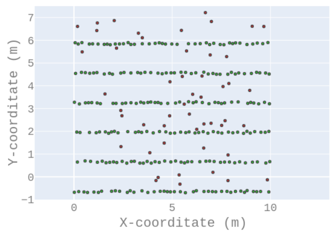
Two LiDARs are used so that the robot will have the same amount of information from its front than from its back. This symmetry helps to handle the change of row: indeed, the robot still detects the crops when going out of a row as the ”back” LiDAR still detects the plants behind it. Furthermore, the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



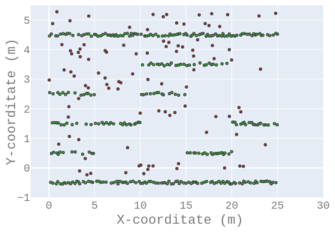
(a) Crop 1



(b) Crop 2



(c) Crop 3



(d) Crop 4

Figure 14: Simulated test crops.

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

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 find lines results
- Considering the presented controller, if the robot is able to autonomously navigate in the crops and how robust is this navigation

To do that, the same control approach (described in Section 3) has been tested with 6 different find lines algorithms: The two already published algorithms Pearl [9] and Ruby [7], and the 4 proposed Ruby refinements (Section 2.2) that 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).

Those 6 navigation algorithms (a navigation algorithm is the association of a find lines algorithm with the control approach) were tested in the four previously described environments (Figure 14).

A test run is done as follow: 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 has only three information:

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

Aside from that, it does not know the number of outliers, the distance between the rows, neither the length of the rows...

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

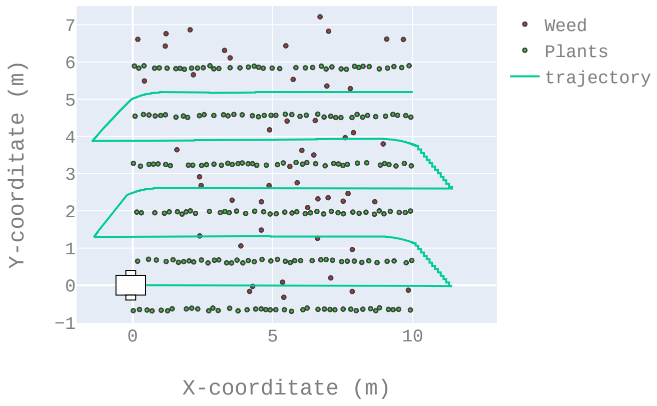


Figure 15: Example of trajectory: the robot starts at the position (0,0), parallel to the crops, as depicted in the figure. Then it has to reach the end of the fifth row without running over the plants. This figure depicts an example of successful trajectory.

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

```
Data: control_algo, constants
1  find_lines_algos = {Ruby, RG, RGOP, RGOPPN,
     RGOPPNI}_constants;
2  crops = {crop 1, crop 2, crop 3, crop 4};
3  for ∀ find_lines ∈ find_lines_algos do
4      for ∀ crop ∈ crops do
5          for $i = 1, .., 5$ do
6              Run a test with find_lines and control_algo
                 in crop;
7              Save the results ;
8          end
9      end
10 end
```

Algorithm 5: Test plan

4.3. Experiment results

Several criteria were considered to compare those algorithms.

- The ratio of successful runs. It is computed as the number of successful runs divided by the total number of runs (i.e. 5). A run is considered as 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

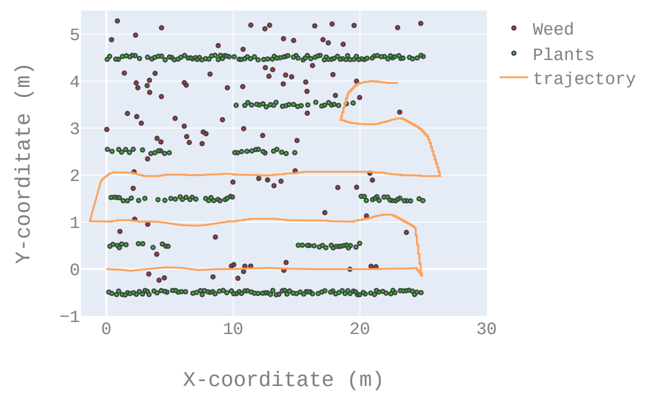


Figure 16: Example of a failed trajectory: the robot starts reached the end of the fifth row but run over crop plants.

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 ratio of successful runs regarding the total number of tries (5 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.

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	60000	15000

Table 4: 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 a 0 error.

an example of successful run while Figure 16 presents an example of failed run. Table 3 details the results according to this criteria.

- The mean squared error when the robot navigated between two rows (not considering 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 considering the changing row maneuver). Table 4 details the results according to this criteria.

- At the end of a row, the LiDAR sensor will have fewer points than it would have between two rows (because it detects less plants due to 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 find lines algorithm. As the control algorithms are the same for all the tested navigation approaches, it is only relevant to test the find lines execution time (the only part that differs from one navigation algorithm to an other). Table 5 details the results according to this criteria.

  - The average number of points processed by the find lines algorithms. Some find lines algorithms consider the raw point data set Z when some consider the filtered one Z^* (Section 2.2.2). That is, the average number of points, presented in Table 6, allows to verify that for the 500 first iterations the find lines algorithms considered the same amount of points (all the Z and all the Z^* have consistent sizes).

  - The average execution time per 100 points. As the find lines algorithm do not consider the same amount of points (Z versus Z^*), the execution time presented in Table 5 may be considered as not comparable regarding only the find lines processes. That is, Table 7 presents an execution time normalized on the processing of 100 points.

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 execution time for the find lines algorithms (LiDAR data to models) 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 number of points considered by the find lines algorithms during the 500 first iterations of each run.

5. Conclusion

5.1. Result interpretations

Regarding the data presented before, 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 mater of fact, combining the Ruby Genetic One Point find lines algorithm with the control algorithm provides a 100% success rate over the five experimental crops (Table 3). It also appears to be the most stable approach (Table 4). However, adding the positive/negative filtering and allowing a point to be associated to several models at the same time seem to perform worse than RGOP (Table 3).

The number of processed points is consistent with what is expected: the first algorithms named Ruby and RG are considering the same amount of points, when the algorithms RGOP, RGOPPN and RGOPPNI are using less points (due to the filter point set Z^*). This can be noticed in Table 6.

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

Table 7: The average execution time per 100 points for the find lines algorithms (LiDAR data to models) during the 500 first iterations of each run.

The considered sensors have a rate of 40 Hz (Section 4.1.1), thus they provide a data set every 25ms. According to Table 5, the RGOP approach needs less then 3ms 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 stressed out that the crops (more precisely the weed positions) were randomly generated (the source code for the generation of the simulated crops are available here [8]). Which suggests that this navigation approach will provide good results in a fair range of crop configurations. Even if the approach has still to be tested in real conditions, the simulation results are more than encouraging.

5.2. Discussion and future work

The presented find lines algorithm named Ruby Genetic One Point associated to the control algorithm described in this paper appears to be quite robust. Regarding the previous work, the find lines algorithm has been improved and a complete control algorithm has been detailed and validated using a ROS/Gazebo simulation.

Currently, we are developing an experimental platform (Figure 17). This robot will be a symmetrical straddle robot equipped with several LiDAR sensors. The navigation approach presented in this paper will then be adapted to this robot in order to do experiments in actual fields. This adaptation will especially have to manage the fact that the robot moves in two different rows at the same time (while the left wheels are in a row the right ones are in the next row). Furthermore, after some tries

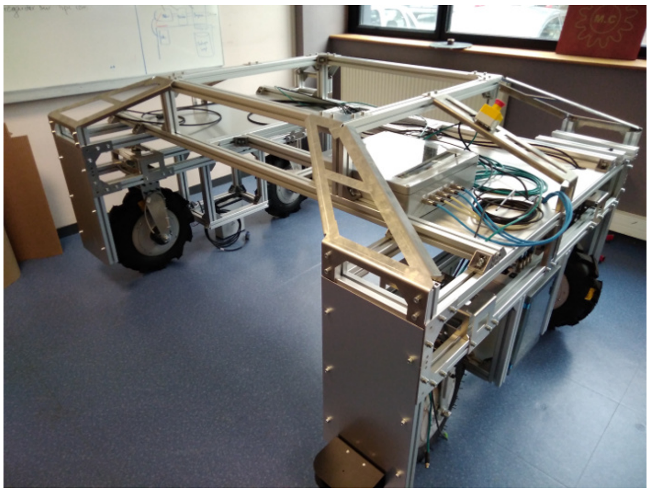


Figure 17: The currently developed platform.

designing a differential straddle robot, it has been decided to use steering wheels instead. That is, the fuzzy controller will have to be modified to fit a steering wheel robot model.

Finally, even if the results are encouraging, it seems that we are reaching the limits of LiDAR data only navigation. To have a fully robust navigation approach fusing data from several types of sensors seems to be needed. The LiDAR data can not, for instance, differentiate a weed from an animal...

That is why we will next focus our work on adding new sensors to the robot (camera, global positioning system...) and developing a framework to add those sensors into our navigation algorithm.

6. Acknowledgment

This work was supported by the French "Pays de la Loire" region through the Atlanstic 2020 program.

References

[1] Y. Xiong, Y. Ge, Y. Liang, S. Blackmore, Development of a prototype robot and fast path-planning algorithm for static laser weeding, Computers and Electronics in Agriculture 142 (2017) 494–503.

[2] Q. Zhang, M. S. Chen, B. Li, A visual navigation algorithm for paddy field weeding robot based on image understanding, Computers and Electronics in Agriculture 143 (2017) 66–78.

[3] Y. Yu, K. Zhang, L. Yang, D. Zhang, Fruit detection for strawberry harvesting robot in non-structural environment based on mask-rcnn, Computers and Electronics in Agriculture 163 (2019) 104846.

[4] B. Arad, J. Balendonck, R. Barth, O. Ben-Shahar, Y. Edan, T. Hellström, J. Hemming, P. Kurtser, O. Ringdahl, T. Tielen, et al., Development of a sweet pepper harvesting robot, Journal of Field Robotics.

[5] An overview of autonomous crop row navigation strategies for unmanned ground vehicles, Engineering in Agriculture, Environment and Food 12 (1) (2019) 24 – 31.

[6] S. Kanagasingham, M. Ekpanyapong, R. Chaihan, Integrating machine vision-based row guidance with gps and compass-based routing to achieve autonomous navigation for a rice field weeding robot, Precision Agriculture (2019) 1–25.

[7] F. B. Malavazi, R. Guyonneau, J.-B. Fasquel, S. Lagrange, F. Mercier, Lidar-only based navigation algorithm for an autonomous agricultural robot, Computers and electronics in agriculture 154 (2018) 71–79.

[8] G. F. Oliveira, R. Guyonneau, Simulation and control of an agricultural robot through an unknown field using lidar sensors, `https://github.com/PolytechAngersMecatroniqueClub/istiaENGRAIS`, accessed: 2020-02-12.

[9] H. Isack, Y. Boykov, Energy-based geometric multi-model fitting, International journal of computer vision 97 (2) (2012) 123–147.

[10] K. H. Choi, S. K. Han, S. H. Han, K.-H. Park, K.-S. Kim, S. Kim, Morphology-based guidance line extraction for an autonomous weeding robot in paddy fields, Computers and Electronics in Agriculture 113 (2015) 266–274.

[11] J. Xue, L. Zhang, T. E. Grift, Variable field-of-view machine vision based row guidance of an agricultural robot, Computers and Electronics in Agriculture 84 (2012) 85 – 91. `doi:https://doi.org/10.1016/j.compag.2012.02.009`.
URL `http://www.sciencedirect.com/science/article/pii/S016816991200049X`

[12] Q. Meng, R. Qiu, J. He, M. Zhang, X. Ma, G. Liu, Development of agricultural implement system based on machine vision and fuzzy control, Computers and Electronics in Agriculture 112 (2015) 128 – 138, precision Agriculture. `doi:https://doi.org/10.1016/j.compag.2014.11.006`.
URL `http://www.sciencedirect.com/science/article/pii/S0168169914002877`

[13] F. B. N. d. Santos, H. M. P. Sobreira, D. F. B. Campos, R. M. P. M. d. Santos, A. P. G. M. Moreira, O. M. S. Contente, Towards a reliable monitoring robot for mountain vineyards, in: 2015 IEEE International Conference on Autonomous Robot Systems and Competitions, 2015, pp. 37–43. `doi:10.1109/ICARSC.2015.21`.

[14] N. Habibie, A. M. Nugraha, A. Z. Anshori, M. A. Ma'sum, W. Jatmiko, Fruit mapping mobile robot on simulated agricultural area in gazebo simulator using simultaneous localization and mapping (slam), in: 2017 International Symposium on Micro-NanoMechatronics and Human Science (MHS), 2017, pp. 1–7. `doi:10.1109/MHS.2017.8305235`.

[15] T. Linner, A. Shrikathiresan, M. Vetrenko, B. Ellmann, T. Bock, Modeling and operating robotic environent using gazebo/ros, in: Proceedings of the

28th international symposium on automation and robotics in construction (ISARC2011), 2011, pp. 957–962.

[16] J. Rada-Vilela, Fuzzylite 6.0: A fuzzy logic control library in c++, `https://fuzzylite.com/cpp/`, accessed: 2020-02-12.