LiDAR-only based navigation algorithm for an autonomous agricultural robot

Flavio Malavazi, Rémy Guyonneau, Jean-Baptiste Fasquel, Sébastien Lagrange, Franck Mercier

LiDAR-Only Based Navigation Algorithm for an Autonomous Agricultural Robot

Flavio B. P. MALAVAZI, Remy GUYONNEAU, Jean-Baptiste FASQUEL, Sebastien LAGRANGE, Franck MERCIER

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

Abstract

The purpose of the work presented in this paper is to develop a general and robust approach for autonomous robot navigation inside a crop using LiDAR (Light Detection And Ranging) data. To be as robust as possible, the robot navigation must not need any prior information about the crop (such as the size and width of the rows). The developed approach is based on line extractions from 2D point clouds using a PEARL based method. In this paper, additional filters and refinements of the PEARL algorithm are presented in the context of crop detection. A penalization of outliers, a model elimination step, a new model search and a geometric constraint are proposed to improve the crop detection. The approach has been tested over a simulator and compared with classical PEARL and RANSAC based approaches. It appears that adding those modification improved the crop detection and thus the robot navigation. Those results are presented and discussed in this paper. It can be noticed that even if this paper presents simulated results (to ease the comparison with other algorithms), the approach also has been successfully tested using an actual Oz weeding robot, developed by the French company Naio Technologies.

Keywords: Crop Navigation, LiDAR Measurements, Line extraction, PEARL, RANSAC

1. Introduction

The legislation about the use of chemical products for farming is getting increasingly severe. In France for instance, the Ecophyto 2018 program aims to drastically reduce the use of phytosanitary products [1]. As a result, some agricultural tasks that were ease (but still not easy) by the use of chemicals (weeding for instance) need alternative solutions to maintain the production efficiency. As a response to that need, the French company Naio Technologies[1] developed an autonomous weeding robot : the Oz robot. This robot is equipped with a LiDAR[2] sensor that is used to detect the crops, therefore allowing it to move autonomously inside the field without damaging the vegetables. For an efficient autonomous navigation, the robot (as provided by the company) needs some prior information about the length, the width and the number of the field crop rows. That is, the navigation behavior is directly dependent of the accuracy of those informations. The objective of the work presented here is to provide a new autonomous navigation algorithm that does not require any prior field information.

While expecting an autonomous robot navigation, the first results were obtained with camera based systems [2, 3]. But as pointed out in [4], the camera data are sensitive to light conditions and atmospheric effects, which can affect the robustness of the approach. An other approach is to consider a GPS[3]-based navigation [5, 6]. But unless improved precision is considered, RTK[4]-GPS for instance, classical sensors are not accurate enough for navigation purposes. Moreover RTK-GPS can be

*Remy GUYONNEAU

Email addresses: flavio.barrospimentelmalavazi@etud.univ-angers.fr (Flavio B. P. MALAVAZI), firstname.LASTNAME@univ-angers.fr (Remy GUYONNEAU, Jean-Baptiste FASQUEL, Sebastien LAGRANGE, Franck MERCIER)

[1] http://www.naio-technologies.com/en/

[2] Light Detection And Ranging

[3] Global Positioning System

[4] Real-Time Kinematics

expensive and are not adapted for an Oz robot size/price system.

LiDAR based approach appears to be an affordable alternative while being weakly sensitive to outdoor lighting, that is why it is considered in several commercial robots (the Oz robot, but also the new French robot PUMAgri[5] for instance). As mentioned before, in addition to the sensor data the current robots need some prior information about the crops (size, length...) and are dependent to the accuracy of those informations. That is, this paper focuses on processing LiDAR data to propose a robust autonomous navigation method that does not need those prior information.

To move autonomously in the field, the robot must detect the crop rows. This can be cast into a problem of model fitting: from a data set (LiDAR measurements), we have to be able to find a set of straight lines (the rows) that best fit the data into individual clusters (Figure 1).

In [4], an interesting LiDAR based autonomous navigation algorithm is presented. The main drawback of this approach is that it requires a "testing phase" in order to calibrate the algorithm. This can be assimilate to prior knowledge requirement, that we want to avoid for robustness purposes.

The considered approach in this paper is based on line detection (the crops) in a 2D point cloud (LiDAR measurements), as it can be done in [7]. Two famous approaches for line detection are RANSAC-based line fitting [8] and Hough transform [7, 9]. From [10] it appears that RANSAC-based approaches are generally more efficient than Hough transform to detect lines in a 2D point cloud.

The recently proposed PEARL Algorithm [11] appears to be more efficient than RANSAC. Nevertheless, this algorithm depicts limitations. In this paper, we propose a refined PEARL algorithm that overcome the limitations of the initial PEARL algorithm, using a penalization of outliers, a model elimination step, a new model search and a geometric constraint. Furthermore, a navigation algorithm based on this refined PEARL is proposed.

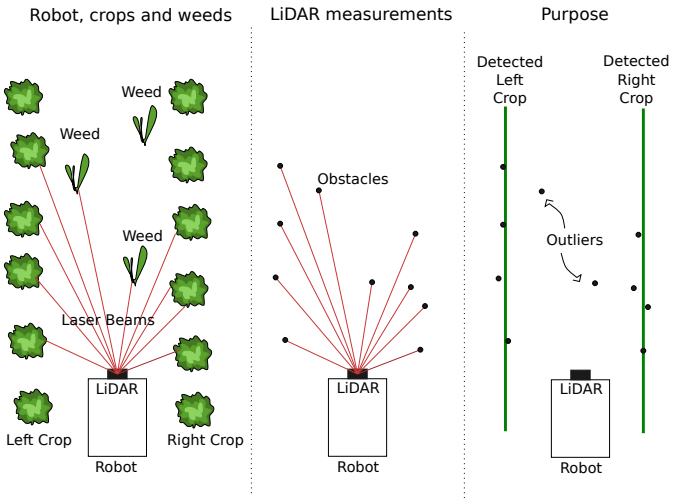


Figure 1: The considered line fitting problem.

The paper is organized as followed. Section 2 details the methods considered in this paper, starting with the original PEARL approach then presenting our refined PEARL algorithm and finally introducing the navigation algorithm based on it. Section 3 focuses on experiments performed using the Oz robot simulator, including a comparison with other navigation algorithms. Finally, Section 4 discusses about this work and results while Section 5 concludes this paper and presents perspectives.

2. Method

This section presents the original PEARL method as detailed in [11], then our refined version of it to detect crops and finally a navigation algorithm based on this crop detection.

2.1. The Original PEARL Algorithm

2.1.1. The General Idea of the Approach

PEARL is an iterative approach (as RANSAC) that was found to be promising due to the fact that it usually converges in less iterations than RANSAC [11]. This is very interesting for real time applications such as the one at hand. It does so by using the knowledge of the last iteration when computing the new one while RANSAC starts over every time waiting for the residuals to be under a threshold.

[5] http://www.sitia.fr/innovation-robotique/plateforme-pumagri/

PEARL method aims at minimizing a function, called *energy* (Equation 1). This function represents a *score* for a set of models (in our case a model corresponds to a line) according to a data set of points. In other words, it allows to compare two sets of models for the same data points and thus to select the one that best fit the data. The energy E is defined by

$$E(\mathcal{L}_i) = \sum_p \|p - L(p)\| + \lambda \cdot \sum_{(p,q)\in\mathcal{N}} w_{pq} \cdot \delta(L(p) \neq L(q)), \quad (1)$$

where:

- $\mathcal{L}_i = \{L_j\} \bigcup L_\emptyset$ is the current set of models, $L_j : f_j(x) = a_j x + b_j$ is the j^{th} model (the j^{th} line) and $L_\emptyset$ the empty model. The empty model is used for points that are not associated to a line (thus considered as outliers, Figure 1). Figure 2 presents an illustration of points associated to models ;
- $p \in \mathbb{R}^2$ is a point extracted from the LiDAR sensor data and $L(p) \in \mathcal{L}_i$ is the model associated to the point p (i.e. $L : p \rightarrow L_j \in \mathcal{L}_i$);
- $\|p - L(p)\|$ is the euclidean distance between the point and its associated line ;
- $\mathcal{N}$ is the set of neighbor points and an element $(p, q) \in \mathcal{N}$ corresponds to two points p and q in the same neighborhood such that p is associated to the model $L(p)$ and q is associated to the model $L(q)$;
- $\delta(L(p) \neq L_q)$ equals 1 if $L(p)$ and $L(q)$ are not the same model, 0 otherwise ;
- $\lambda \cdot \sum_{(p,q)\in\mathcal{N}} w_{pq} \cdot \delta(L(p) \neq L(q))$ is a penalty for the placement of close points in different models. This penalty is weighted using

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

with $\|p - q\|$ being the euclidean distance between the points p and q ;

- ζ and λ are two constants chosen heuristically [11].

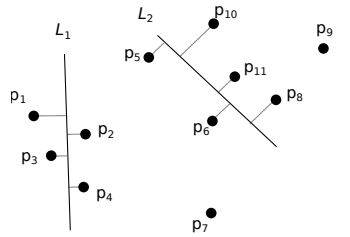


Figure 2: Labels example: $\{p_1, p_2, p_3, p_4\}$ are labeled with the model L_1, $\{p_5, p_6, p_8, p_{10}, p_{11}\}$ are labeled with the model L_2 and $\{p_7, p_9\}$ are labeled with the model $L_\emptyset$. For instance $L(p_2) = L_1$ and $L(p_8) = L_2$.

Note that outliers are points that are too far from any computed models according to an heuristically defined threshold. It corresponds to LiDAR points that are generated by an obstacle in the middle of the field (a weed for instance) and does not belong to any crop. That is, inliers are points associated with a model.

2.1.2. The Algorithm Steps

Here we present the original PEARL algorithm steps. This algorithm, detailed in Figure 3, searches for models (lines in our case) in a data set (LiDAR points).

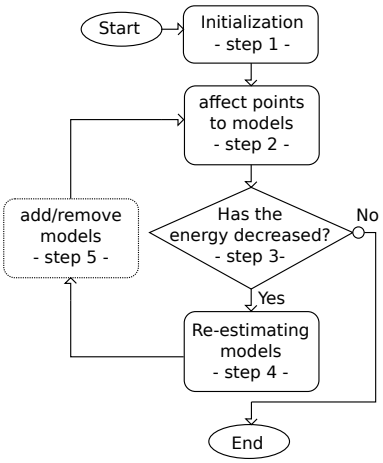


Figure 3: PEARL algorithm for model fitting.

The detailed algorithm steps:

1. At initialization, the algorithm randomly samples data to get $\mathcal{L}_0$, which is the first set of models, according to a defined number of initial models. It may also add the empty model $L_\emptyset$ for points that are considered as outliers ;

2. Run α-expansion [12] for the energy described by Equation 1 and $\alpha \in \mathcal{L}_i$. This step associate each data point to the closest model $L_j \in \mathcal{L}_i$ (in a euclidian distance point of view). If the closest model is further than a threshold the data point is associated to the outlier label $L_\emptyset$. Once the label of all the data points has been updated the energy of that iteration can be computed (Equation 1). Figure 4 presents an example of α-expansion result ;
3. If the computed energy from step 2 does not decrease with respect to the previous iteration, then the procedure ends ;
4. Else, one solves Equation 3 and obtains a new set of models $\mathcal{L}_i'$ by replacing each model $L_j \in \mathcal{L}_i$ by a model $L_j{}'$ that better fits the points of the model according to

$$L_j' = \arg\min_{L_j} \sum_{p \in P(L_j)} \|p - L_j\|, \tag{3}$$

where

- $P(L_j) = \{p | L(p) = L_j\}$ is the set of points p that are associated to the model L_j.

5. Sample more models from the points that belong to $L_\emptyset$, or merge/split current models in $\mathcal{L}_i'$ if there are too closed (according to the euclidean distance). Go to step 2.

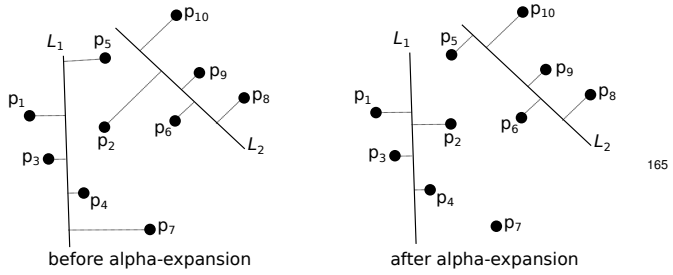


Figure 4: Example of α-expansion process: before the α-expansion, the points p_5 and p_7 are associated to the model L_1 and the point p_2 is associated to the model L_2. After the α-expansion, the point p_5 is associated to the model L_2 because it is closer to L_2 than L_1 (according to the euclidean distance). For the same reason, the point p_2 is associated to L_1 after the α-expansion. Note that the point p_7 is finally associated to the outliers model $L_\emptyset$ because it is considered too far from L_1.

2.1.3. Motivation for Implementing Step 5

It was observed that, because of the characteristics of cropping fields, the inclusion of step 5, described as optional in [11], was very likely to provide better results.

Merging models that are too close seems very reasonable, considering that we are treating 3-dimensional plants that do not represent a single point in space, but a small cluster. In order to extract the best possible model for a row, PEARL is expected to have a single model for a single row, this step prevents PEARL from assuming that there are two very close models if using different points of the same plant.

Furthermore, searching for new models in the outlier pool allows the algorithm to retrieve rows that may exist, but were not yet probed or had been discarded during previous iterations.

2.2. The Refined PEARL Algorithm

Using the standard PEARL method described in Section 2.1.2 as a starting point, we present here adjustments that would make it more adapted to the context of a crop navigation.

2.2.1. Penalizing the Energy Based on the Outlier Count

It was noticed, during the benchmark for PEARL, that a typical set of LiDAR measurements would contain a fair amount of points that would be automatically included within non empty models. This means that the outlier model $L_\emptyset$, would usually be small, compared to the whole initial data set. The reason for that is: if we have weeds distributed alongside the plants, the LiDAR does not provide enough information to rule them out from the data set, so they will end up falling into the model that describes the row that they are close to. If, on the other hand, the weeds are positioned in the middle of the row, they will very likely be ruled out, but even then, it is reasonable to assume that the amount of points provided by each row will be greater than the amount of points in the middle of them.

Due to those facts it is possible to penalize the energy (Equation 1) based on the number of outliers at the end of each α-

expansion. Thus the new energy computation is

$$E(\mathcal{L}_i) = \sum_p \|p - L(p)\| + \lambda \cdot \sum_{(p,q)\in\mathcal{N}} w_{pq} \cdot \delta(L(p) \neq L(q)) + \phi \cdot \|P(L_\varnothing)\| \tag{4}$$

where:

- $\phi \cdot \|P(L_\varnothing)\|$ is the inserted energy penalty based on the amount of outliers, ϕ represents the weight (heuristically chosen constant) given to each point that is within the empty model $L_\varnothing$.

2.2.2. Model Elimination After Each α-expansion

Another modification proposed for the PEARL procedure, is a run through all the models after each α-expansion in order to check if they have a number of inliers that is greater than the minimal number of points they have started with. If they do, they are maintained, but if not, we remove the points from the labels placing those into the outlier pool $L_\emptyset$. This allows to eliminate failed models even faster, and, by doing so, increases the chances of getting the right models in less iterations.

2.2.3. Searching for New Models After the α-expansion

With the model elimination, the algorithm is now eliminating models just after each α-expansion. Thus, the searching models from within $L_\emptyset$ step can be moved to be just after each alpha expansion. The models that are obtained through this search then join the others that were already known for the proceeding of the algorithm.

2.2.4. Adding a Geometric Constraint

In a field, the plants are generally organized as a set of parallel strait lines. This configuration allows the seeding, caring and harvesting of the plants to be faster and more efficient.

For this reason a parallelism constraint is proposed for the PEARL procedure. Models that PEARL is supposed to fit the data with must also be parallel and, by being so, having a sizable portion of inliers attributed to each one of them.

This is made by performing the following procedure:

1. For each model L_j of the current set of models $\mathcal{L}_i$, we calculate D_j the sum of the distance between all the points of the model and the model itself over the number of points attached to the model (model inertia).
$$D_j = \frac{\sum_{p\in P(L_j)} \|p - L_j\|}{\|P(L_j)\|}.$$
2. For each model $L_j \in \mathcal{L}_i$, we compute ψ_j that corresponds to the number of models that are parallel to L_j:
$$\psi_j = \sum_{L_k \in \mathcal{L}_i \setminus (L_j \bigcup L_\emptyset)} \delta(L_j // L_k).$$
The parallelism $\delta(L_j // L_k)$ is defined regards to a threshold (heuristically chosen constant):
$$\delta(L_j // L_k) = \begin{cases} 1 & \text{if } |a_j - a_k| < \text{threshold} \\ 0 & \text{otherwise} \end{cases}, \tag{5}$$
where a_j and a_k are slopes related to L_j and L_k lines.
3. We then compute the ratio R_j:
$$R_j = \omega \cdot \frac{D_j}{\psi_j},$$
with ω an heuristically chosen constant.

If the ratio R_j is greater than a threshold, the model L_j is excluded. If not it is maintained. This favors keeping compact parallel lines, meaning that each line depicts a weak scatter (i.e. low inertia D_j) of detected points around the center line. Note that if $\mathcal{L}_i = \{L_\emptyset\}$ (all the models are excluded) after step 4, we re-append the best model we had from step 3 (that is the model with the lowest ratio R_j) before starting the next iteration of PEARL. Otherwise the new set of models $\mathcal{L}_{i+1}$ will correspond to the ones that were maintained.

2.2.5. OPAL and RUBY : Refined PEARL Algorithms

We named our final refined PEARL algorithm *RUBY*. Based on PEARL, RUBY includes all the modifications presented before. This new algorithm, depicted in Figure 5, is detailed below.

1. At initialization, the algorithm randomly samples the data to get $\mathcal{L}_0$, which is the first set of models. It may also add the model $L_\emptyset$ for the points that are considered as outliers.

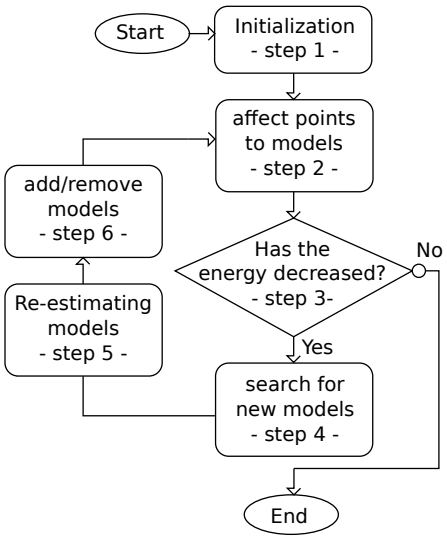


Figure 5: Refined PEAL - RUBY Algorithm

2. Run α-expansion [12] for energy described by Equation 1 and $\alpha \in \mathcal{L}_i$. This places each data point in the closest model L_j within $\mathcal{L}_i$, or, if it is still too far from all the models (according to a threshold), into $L_\emptyset$. The label of each data point gets altered, and the final configuration allows us to compute the energy for that iteration.
3. If the calculated energy from step 2 does not decrease (regards to the last iteration), stop and return the last set of models $\mathcal{L}_{i-1}$.
4. Sample more models using the points that are included in the $L_\emptyset$ model.
5. Solve Equation 3 to obtain a new set of models $\mathcal{L}_{i+1}$.
6. Eliminate the models that do not meet the geometric criteria (parallelism) and merge models that are too close to each other (according to a threshold) as a way of not having two different models for a single row.

In order to be able to evaluate the gain obtained by adding the parallel constraint, we named *OPAL* the refined PEARL algorithm that does not have it. In other words, the only difference between OPAL and RUBY is that RUBY considers the parallel constraint where OPAL does not.

2.3. Navigation of the Robot in the Crop

The RUBY algorithm depicted before allows to detect the crops around the LiDAR sensor (i.e. the robot). In this subsection we present an algorithm that uses this information to navigate among the crops.

Basically speaking, in our case, navigation consists in moving the robot along a line that is equidistant from the closest left and right parallel crops previously estimated (Figure 1). Thus, in addition to the RUBY algorithm, we propose filters to improve the robustness of the autonomous navigation of the robot.

The model filtering aims at selecting models that are coherent over the time, rejecting possible abnormal models that might be given by one isolated RUBY call.

Different additional filters are added to improve the navigation. One filter is used to enlarge the LiDAR's range (using results from the previous iterations) in order to improve the data points selection. An other filter is used to prevent the robot from taking wrong control decisions while standing non-parallel to the detected rows. Note that the navigation work-flow is depicted on the Figure 9 while the steps are detailed in the next subsections.

2.3.1. The Data Filtering

The idea of this filter is to use the previously computed models (the known rows) to filter the new LiDAR data set, removing points that are generated by obstacles between the rows. This may be seen as building region of interests (boxes) around the followed models, from which it is possible to take the data for the next reading. The boxes have infinite dimensions in length, being limited only in width, in order to eliminate outliers that are not part of the models, and by the LiDAR range that is approximately 4 meters for the Oz robot. The boxes are illustrated in Figure 6

2.3.2. The Model Filter

At this step, we take all the models that were found (by the refined PEARL algorithm for instance) and select the closest to the robot on the right and left sides. We then proceed to test the selected models in order to see whether they are parallel or not to the ones we had on the previous iteration. In other words: it compares the set of models that arrived with the two (or one) main models we had before (right and/or left) in order to come

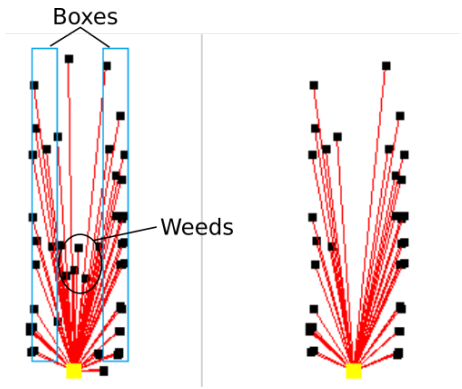


Figure 6: Boxes used as filters for the LiDAR measurements and resulting data points. It can be seen that points generated by obstacles between the rows are removed from the data set.

up with the best possible model for the robot to follow. If the new computed lines are not consistent with the previous ones, they are ignored. A safety mechanism is added that prevents the rejection of too many results in a row: it consists of a counter of permanence that erases the followed models if a number of successive results are rejected. After that amount of rejections (limit chosen heuristically), the filter becomes more flexible (as it goes into an initialization mode). This caution is important because in case of a fast correction, the new arriving models may diverge from the previous ones by more than the tolerance given by the strict filters, and in that case, the robot would never have a new model to follow, and consequently would run over the plants.

2.3.3. The Pre-control Function

Once the models corresponding to the rows have been identified, some values are computed such as: the average of the a and b coefficients (note that models are lines such that $y = ax + b$), the average distance between rows (consequence of the b coefficient) and the current distance to the target (the center of two rows or, if it is a single one, a threshold of half the average distance between the past rows or even a default distance).

The computation of this last value also has a safety measure built in. If the followed models are too steep (which may happen if the robot turns within the row), it changes the way of calculating distance to target (i.e: the error used by the FIR filter and the PID controller detailed later). The new computation becomes the distance between the robot and the point where it would intercept the straight line defined by the model. This safety measure is very important since RUBY and filters may return two steep models with the robot at their exact center. In this case, it would be interpreted as a 0 error, so no correction would be made and plants would most certainly be lost. The Figure 7 shows two situations where the given models are steep, and how the robot may be positioned in relation to those. In the first case, the regular error calculation could be enough for driving the robot back to the center of the rows, but in the second scenario, the error coefficient would be close to 0 since both rows are equidistant to the robot (yellow square).

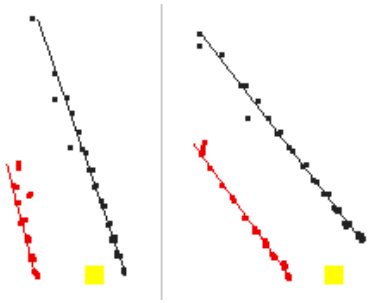


Figure 7: Two different scenarios for steep models (models that are not parallel to the robot direction) returned by RUBY and the filters: the small squares correspond to the data (LiDAR measurements), the big squares correspond to the robot estimated position and the lines correspond to the estimated crops.

2.3.4. Finite Impulse Response Filter (FIR)

The error value calculated and saved previously, is passed through a FIR (Finite Impulse Response Filter), in order to have a bit of the influence of past values of error in the new one. Equation 6 presents the filter computation.

$$e_f[k] = b_0 e[k] + b_1 e[k-1] + ... + b_N e[k-N], \qquad (6)$$

with

$e_f[k]$: the filtered error at time k (value that will be considered by the PID)

$e[k]$: the non filtered error at time k (direct estimated value)

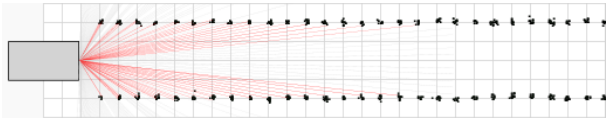
N : the filter order (constant chosen empirically)

$b_0, b_1 ... b_N$: the values of the impulse response

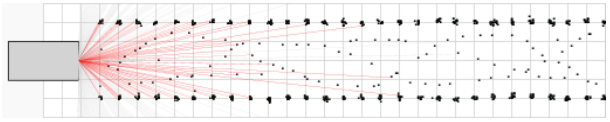
This is interesting because during movement, the robot does not alter its position instantly so, the errors are directly connected one to another if taken in sequence. The FIR has another important feature that is limiting the influence of an impulse over time, not allowing the error to build up indefinitely.

2.3.5. *Proportional Integrate Derivative controller*

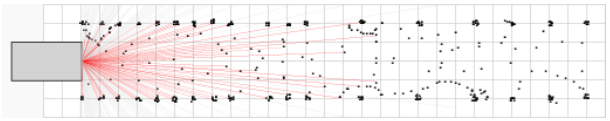
Finally, we arrive to the controller itself, for this part we use a standard PID controller and the error calculated by the FIR filter. The output of the PID controller is the correction that is applied to the wheels-speed, and by doing so, guides the robot through the field.



(a) Layout for level 1.



(b) Layout for level 2.



(c) Layout for level 3.

Figure 8: The three simulated crops. The gray rectangle on the left illustrates the robot, the dark shapes correspond to crops and weeds and the lines correspond to the LiDAR measurements.

3. Results

This section presents tests that have been done using the Oz simulator Version 1.0.0 provided by the Naio Technologies company. First the considered crop scenarios are presented, then the navigation procedure and the evaluation protocol are presented to finish with a presentation and discussion of the results.

3.1. *The Experimentations*

We have created 3 different simulated crops (Figure 8) for the robot to go through, and running eight different algorithms through those crops.

3.1.1. *Navigation Procedures*

The set of procedures introduced in Figure 9 has been defined to be able to evaluate the contribution, in terms of navigation efficiency, of each step of the proposed navigation system. Each flowchart represents an approach that has its results quantified in the Tables 1, 2 and 3. All the blocks presented in the flowcharts are described in the list bellow.

- **Raw LiDAR data:** Arrays of data points provided by the LiDAR sensor in front of the robot.
- **Data filtering:** Selecting data from the arrays that fits the lastly accepted model of each side, if it is not too steep. In case there are no accepted models or the model on that side is too steep, the filter takes the points that are in that side of the robot with the only limitation that they must be between 30 cm and 3 m from the robot.
- **CROP MODELING:** This block is either our RUBY algorithm as described in Section 2.2.5, the OPAL algorithm (the RUBY algorithm but without considering the geometric constraint), the original PEARL algorithm as described in 2.1.2, or the RANSAC-based method for model fitting as described in [8].
- **Right and left model filtering - 1:** Selection, from the models given by the previous step, of which will be the model on the right and on the left side of the robot. This filter takes into account the history of already accepted models by using: the average value of the distance between right and left models (thus, the distance between the rows of the crop), the average and the value of the last a and b coefficients (reminder: the models are lines described by the equation $L_j : f_j(x) = a_j x + b_j$).

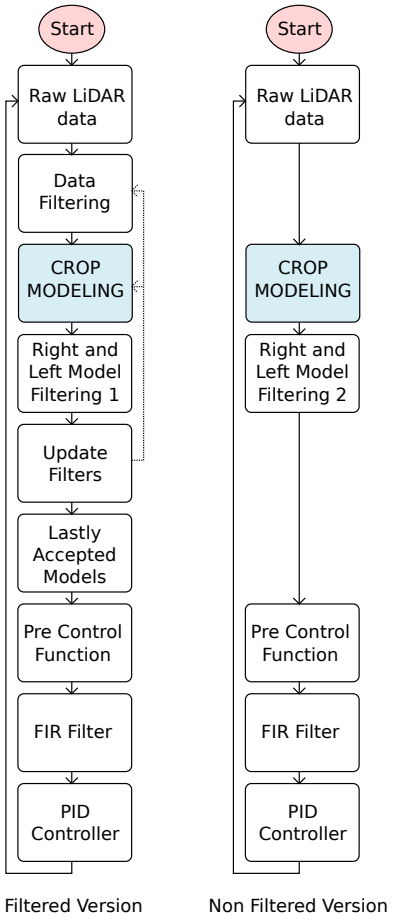


Figure 9: Routines considered for the tests. The crop modeling is either RUBY, OPAL, PEARL or RANSAC, depending on the tested algorithm.

- **Right and left model filtering - 2:** Simple filter, takes only the closest right and left models in order to place the robot between only two models instead of the unknown number that will emerge from the model fitting .
- **Update Filters:** Function that updates the model and data filters with the accepted model's values. If no model is accepted, it deactivates the data filtering and uses the average values of a and b for updating the right and left model filtering.
- **Pre-control function:** Calculates the average values of distances between rows, steepness of models, and the distance to the target. This will be used in the next step. The target distance is either the distance between the robot and the center of the two rows it is following (if the models are not too steep) or the distance between the robot and the model it is about to intersect (in case the lines are too steep).
- **FIR Filter:** Takes a number of past value errors and weight them using fixed coefficients to find the present error value.
- **PID Controller:** Classical PID controller that actuates if there are models to be followed. In the off case that there are not, the controller returns the command 0 and reduces the speed of the robot in order to give it time to find the models. Note that for experiments, navigation parameters (PID and FIR) have been empirically tuned.

3.1.2. Evaluation Protocol

As mentioned before, we use the simulator provided by Naio Technologies [6] version 1.0.0 and the tests are conducted like follows:

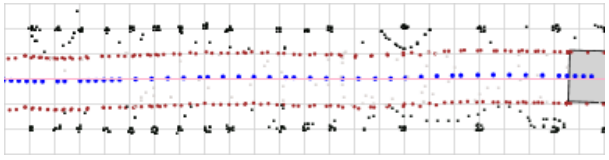
1. The robot went through every simulated field (Figure 8 : without weeds - 1, some weeds - 2 and a lot of weeds - 3) 10 times using each algorithm, which totals 240 paths creating images like in Figure 10a.
2. By remotely controlling the robot we create the ideal path that the robot is supposed to take (i.e. a straight line through the middle of the crop) as illustrated in Figure 10b.
3. To ease the algorithm comparison a line of distinctive color (pink) was drawn over the ideal path. This step is illustrated by Figure 11.
4. Navigation failure is defined over two scenarios:
   (a) The robot leaves the area shown in Figure 12a.
   (b) The robot stops for any reason before the end of the studied area.

   Figure 12b illustrates a navigation failure.
5. We count the number of blue spots (robot path) in each figure, and add their distances to the distinctive pink line

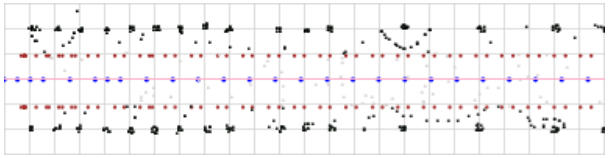
[6] http://www.naio-technologies.com/

representing the ideal path. This counting actually computes how much the robot has deviated from the ideal path. This error is normalized with the amount of blue dots in each figure (this number is directly connected to the robot speed).

6. The average, maximum and minimal values of the errors generated by the processed path regards to the ideal one are all calculated for each method at each level. The results are available in tables 1, 2 and 3. Note that the failures are not used and are just counted for each methods success rate calculations.
7. For every successful passage of each method, the number of times that at least one plant got destroyed (moved over) by each method is counted.



(a)



(b)

Figure 10: Illustrated paths taken by the robot in the simulator.

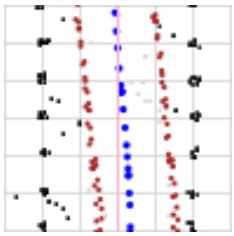
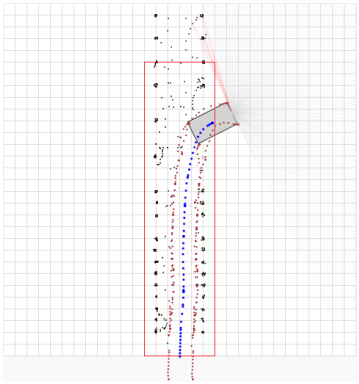


Figure 11: Line drawn in the middle of crop's rows, covering the ideal path.

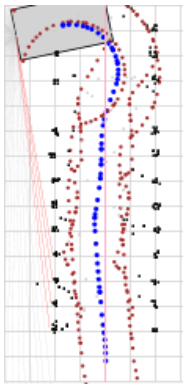
3.2. The Results

3.2.1. The Considered Metrics

Here are explanation about the table values:



(a) Box that sets the bounds for failure.



(b) Clear failure.

Figure 12: Illustrating failures.

- Success Rate: percentage of successful runs (that did not failed as defined in the evaluation protocol).
- Success Rate without crushing plants: percentage of successful runs that did not destroyed any vegetable.
- Errors: due to irregular sampling rate[7], we did not choose an error value based on mean square values (difficulty of pairing irregularly spaced points). Instead we computed an error based on the total distance traveled by the robot during the runs regards to the perfect one (strait line regarding the optimal path):

$$\text{error} = (\text{traveled distance})/(\text{perfect run distance}) \quad (7)$$

The simulator considers $20 \times 20cm^2$ patches (see grids on figures), and the optimal path has a length of 480cm.

[7] In the considered simulation the sampling rate is directly dependent of the robot speed and thus of its accelerations and decelerations.

For instance, an error of 1.15 means that the robot path was 1.15 longer than the optimal path. As the optimal path is 480 cm long, an error of 1.15 means that the robot traveled 552 cm, so deviate from the path of 72 cm all along the row.

  - Average error: the mean error of successful runs regards to the ideal one (straight line with constant speed). The higher the value the more the robot deviated from the optimal path.
  - Maximal error: the worst successful run (longest traveled distance).
  - Minimal error: the best run (shortest traveled distance).

3.2.2. Analysis of the Tables

By looking at the tables 1, 2 and 3, it is possible to see an evolution pattern between the four PEARL based approaches that we have proposed for model fitting. Here is an analysis for each column and a general conclusion about the study.

- **Algorithm 1 - RUBY Filtered:** The best PEARL-based approach that we obtained, using the RUBY refinement. It combines all the features previously described for a method that proved itself robust, and quite close to the ideal path that we would like the robot to follow. It destroyed no plants at all, and it's deviation from the ideal path was the smallest one, giving us the best encountered solution for the problem to date.
- **Algorithm 2 - RUBY Only:** Having the parallelism constraint in its favor, by using RUBY, this method encountered quite a meaningful success by itself. It performed better than the OPAL only, and even the filtered OPAL in level 1. It destroyed less plants than both OPAL approaches, when comparing all levels, while maintaining a higher success rate as well. The distances increased significantly when weeds were added to the field, showing that it is not enough to use RUBY only for our goal.
- **Algorithm 3 - OPAL Filtered:** The usage of the filtering, combined with OPAL (PEARL variation without the parallelism constraint), provided us with a result that was somewhere between our best method, and the standalone PEARL, without filtering of any kind. For noisy scenarios (levels 2 and 3), it presented a superior behavior to the unfiltered approaches. At level 1, it performed slightly worse than the unfiltered RUBY, which shows that the parallel constraint has quite a deep impact at the method's performance.
- **Algorithm 4 - OPAL Only:** Stripped of the parallelism constraint and the filters, OPAL, performed fairly well for an ideal scenario, but had an extremely high destruction rate and trajectory deviance increased when faced with weeds in the field. It consisted on the first step towards development and it helped us to understand what would be necessary for reaching the *"RUBY - filtered"* successful approach.
- **Algorithm 5 - Original PEARL Filtered:** Using the original PEARL approach, alongside with the filters created during the development, produced results that were better than the RANSAC ones, and even better than the intermediate modified algorithms, although, when compared against the fully developed RUBY approach, the original PEARL crushed more plants and went through larger distances in two out of the three levels that were tested and, in the third one, had a success rate that was 20% lower than RUBY did, on the same scenario.
- **Algorithm 6 - Original PEARL Unfiltered:** The pure PEARL unfiltered approach was the one that had the most setbacks. Being the responsible for the bigger distances that the robot faced on levels 1 and 2, it failed completely on level 3, not providing any result that could be accounted as a success. This was addressed to during the development, by modifying the order of the optional steps, and adding the filtering and calculations that led to the RUBY method.

Algorithms	RUBY filtered	RUBY only	OPAL filtered	OPAL only	PEARL filtered	PEARL only	RANSAC filtered	RANSAC only
Success Rate	100 %	100 %	100 %	100 %	100 %	80 %	90 %	100 %
Success Rate without crushing plants	100 %	100 %	100 %	100 %	100 %	80 %	90 %	100 %
Average Error	1.15	1.38	1.49	1.65	1.33	3.53	1.10	1.09
Maximal Error	1.56	3.13	3.94	2.21	3.17	4.71	1.15	1.14
Minimal Error	1.04	1.00	1.02	1.03	1.00	2.18	1.05	1.05

Table 1: Level 1 - simulated perfect scenario

Algorithms	RUBY filtered	RUBY only	OPAL filtered	OPAL only	PEARL filtered	PEARL only	RANSAC filtered	RANSAC only
Success Rate	100 %	100 %	80 %	60 %	90 %	20 %	50 %	100 %
Success rate without crushing plants	100 %	100 %	70 %	50 %	90 %	10 %	20 %	100 %
Average Error	1.10	3.75	2.64	5.24	2.10	7.26	9.68	3.27
Maximal Error	1.29	4.74	6.06	9.06	4.09	9.41	11.28	4.14
Minimal Error	1.00	2.01	1.26	2.84	1.08	5.10	8.01	2.42

Table 2: Level 2 - simulated scenario with some outliers

- **Algorithm 7 - RANSAC Filtered:** - Changing from PEARL based methods to RANSAC, while keeping the same filters developed for RUBY proved itself to be a bad combination, having a success rate that dropped to 30% with numerous weeds, destroying plants at a rate and number that superseded almost all other approaches.

- **Algorithm 8 - RANSAC Only:** The pure RANSAC approach showed that it is possible to use our control functions (without any other filters than the FIR and the pre-control) to control the robot with a performance that was equivalent to *RUBY Filtered* at the ideal scenario, but had a serious degradation in performance after the addition of the smallest number of weeds.

After weighting the results seen in the tables, it can be concluded that the *RUBY Filtered* approach is indeed the most robust one, having the smallest deviation from the ideal path at levels 1 and 2, while keeping its efficiency at 100% on all levels, the pure PEARL algorithm, that has indeed obtained a smaller average distance on the hardest level, did not however manage to clear the field in all its iterations on level 3, hence demonstrating that the RUBY approach was indeed an improvement over PEARL for this scenario.

4. Discussion

The previously presented results show that our modified PEARL based approach can provide an efficient method to detect crops using LiDAR data. Those results were obtained using a simulator in order to easily obtain a ground truth for evaluation and comparison purposes. As this work is meant to be used with actual agricultural robots the results can be discussed for real application. Preliminary experiments have been performed

Algorithms	RUBY filtered	RUBY only	OPAL filtered	OPAL only	PEARL filtered	PEARL only	RANSAC filtered	RANSAC only
Success Rate	100 %	90 %	70 %	60 %	80 %	0 %	30 %	100 %
Success rate without crushing plants	100 %	100 %	70 %	50 %	80 %	0 %	0 %	100 %
Average Error	2.33	7.20	3.98	6.26	1.41	NaN	10.82	4.51
Maximal Error	3.39	10.94	7.39	8.67	3.17	NaN	13.44	5.59
Minimal Error	1.28	3.49	1.59	2.23	1.04	NaN	6.65	2.73

Table 3: Level 3 - simulated scenario with a lot of outliers

using an actual Oz robot and results are promising[8] (subjective evaluation of the robot navigation). It appears that the new approach allows the robot to move between the crops, even if the crops are non regular and people are moving around. The reader should note that those results were obtained without any modification of the algorithm tuned in our laboratory (the constants were heuristically defined in an indoor environment with plastic cones instead of plants). This depicts how robust the approach can be. Next steps will concerns the quantitative evaluation using these real data, with the underlying difficulty of building of a ground truth.

The proposed approach depicts some limitations, mostly regarding the considered sensor and robot. As the LiDAR is a 2 dimensional sensor, it has been noticed that its height is a critical parameter:

- if the LiDAR is too low according to the weeds that are between the crops, the sensor can be "blinded" by the weeds and in this case the approach can not be used to detect the crops,
- on the other hand if the sensor is too height according to the crops, the sensor does not detect them and thus the approach can obviously not be used.

To overcome this limitation, we plan to use 3 dimensional sensor in our future work.

[8] https://youtu.be/qQr68RLNs9o, https://youtu.be/bMudwhPALcM

An other limitation according to the LiDAR is that it is nearly impossible according to the sensor data to identify the type of obstacle in front of the robot: in other words a weed in front of the robot looks like a rock for the sensor, and the other way around. This leads to safety issue while the robot is moving. In the future, we plan to use camera sensors in front of the robot to overcome this issue.

Finally the turn around (to change row for instance) is quite difficult with only one LiDAR sensor in front of the robot. Indeed, when the robot is at the end of a row, it does not have any information about what is behind it. It thus has to use odometry data and a dead reckoning approach to change row. Due to slippery ground, holes, mud... the approach is not reliable and it often mistakes the row to go in. In the future, we plan to have symmetrical data to overcome this problem.

5. Conclusion

This paper presents a new approach to extract lines from a point cloud with application to agricultural robot autonomous navigation in a GPS denied environment. Simulated results are presented to demonstrate the interest of the approach compared to existing methods (RANSAC, PEARL). The approach also has been experimented with an actual Oz robot in an actual field. Experimental results are promising and the method seems to be robust even when the crops are not perfect. Some limitation remains and some work still have to be done in order to be able to use this approach in most of the actual agricultural

configurations (mainly due to the considered sensor and robot).

6. Acknowledgement

This work was supported by the Brazilian Capes Agency through the Brafitec program.

References

[1] The ecophyto 2018 plan, `http://agriculture.gouv.fr/ecophyto-english`, accessed: 2017-06-01.

[2] J. B. Gerrish, T. Surbrook, Mobile robots in agriculture.

[3] J. Reid, S. Searcy, Vision-based guidance of an agriculture tractor, IEEE Control Systems Magazine 7 (2) (1987) 39–43.

[4] S. A. Hiremath, G. W. Van Der Heijden, F. K. Van Evert, A. Stein, C. J. Ter Braak, Laser range finder model for autonomous navigation of a robot in a maize field using a particle filter, Computers and Electronics in Agriculture 100 (2014) 41–50.

[5] T. Bell, Automatic tractor guidance using carrier-phase differential {GPS}, Computers and Electronics in Agriculture 25 (12) (2000) 53 – 66.

[6] M. Prez-Ruiz, D. Slaughter, C. Gliever, S. Upadhyaya, Automatic gps-based intra-row weed knife control system for transplanted row crops, Computers and Electronics in Agriculture 80 (2012) 41 – 49.

[7] O. C. B. Jr., A. Mizushima, K. Ishii, N. Noguchi, Development of an autonomous navigation system using a two-dimensional laser scanner in an orchard application, Biosystems Engineering 96 (2) (2007) 139 – 149.

[8] M. A. Fischler, R. C. Bolles, Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography, Communications of the ACM 24 (6) (1981) 381–395.

[9] H. P. VC, Method and means for recognizing complex patterns, uS Patent 3,069,654 (1962).

[10] L. Jacobs, J. Weiss, D. Dolan, Object tracking in noisy radar data: Comparison of hough transform and ransac, in: IEEE International Conference on Electro-Information Technology , EIT 2013, 2013, pp. 1–6.

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

[12] A. Delong, A. Osokin, H. N. Isack, Y. Boykov, Fast approximate energy minimization with label costs, in: Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on, IEEE, 2010, pp. 2173–2180.