Tree detection with low-cost 3D sensors for autonomous navigation in orchards

Adrien Durand-Petiteville, Emile Le Flecher, Viviane Cadenat, Thierry Sentenac, S. Vougioukas

Tree detection with low-cost 3D sensors for autonomous navigation in orchards

A. Durand-Petiteville[1], E. Le Flecher[2,3], V. Cadenat[2,3], T. Sentenac[2] and S. Vougioukas[1]

***Abstract*—This paper deals with autonomous farming and with the autonomous navigation of an agricultural robot in orchards. These latter are typical semi-structured environments where the dense canopy prevents from using GPS signal and embedded sensors are often preferred to localize the vehicle. To move safely in such environments, it is necessary to provide the robot the ability of detecting and localizing trees. This paper focuses on this problem. It presents a low cost but efficient vision-based system allowing to detect accurately, quickly and robustly the trees. It is made of four stereo cameras which provide a point cloud characterizing the environment. The key idea is to find the tree trunks by detecting their shadows which are materialized by concavities in the obtained point cloud. In this way, branches and leaves are not taken into account, improving the detection robustness and therefore the navigation strategy. The method has been implemented using ROS and validated using data sequences taken in several different orchards. The obtained results definitely validate the approach and its performances show that the processing time (around 1ms) is sufficiently short for the data to be used at the control level. A comparison with other approaches from the literature is also provided.**

I. INTRODUCTION

It is projected [1] that agricultural production will need to double by 2050 to cover the needs of an increasing population for food, feed, fiber and biofuels, and do so in a sustainable manner, despite increasing shortages in available farm labor. Advanced automation of field operations has been identified as a key technology to achieve this goal [2]. One of the main challenges in developing automated solutions for operations like mowing, spraying, pruning, thinning and harvesting in orchards is reliable and precise autonomous navigation. The most basic operation of an autonomous vehicle in an orchard is to drive from one end of a row to the other end (row traversal), and then navigate in the headland to turn and enter the next row. This operation is repeated to cover the area of interest (e.g., an orchard block). Unfortunately, precision GPS-based guidance is often unreliable in orchards. Tall tree foliage can introduce large errors in GPS positions due to multipath effects, and also cause intermittent or complete GPS signal outage. Furthermore, commercial orchards constitute semi-structured outdoor environments: trees are planted in parallel rows, but individual tree positions and trunk-canopy shapes and sizes vary in space and time and tree lines may also not be perfectly parallel or at exactly the same distance from each other. Finally, precise georeferenced up-to-date orchard maps are typically unavailable and can be costly to generate. For these reasons, orchard navigation based only on GPS has not proved - so far - to be reliable and accurate enough to become commercially available.

An alternative approach is to utilize on-board sensors that acquire data from the surrounding environment and develop navigation strategies that exploit the spatial structure of orchards. Since orchard structure is primarily defined by the tree rows, reliable detection and localization of trees is considered very important for navigation. Such a capability would make it possible to navigate locally through rows and headlands using feedback controllers based on exteroceptive sensing, as well as to detect transitions between an orchard row and its neighboring headland.

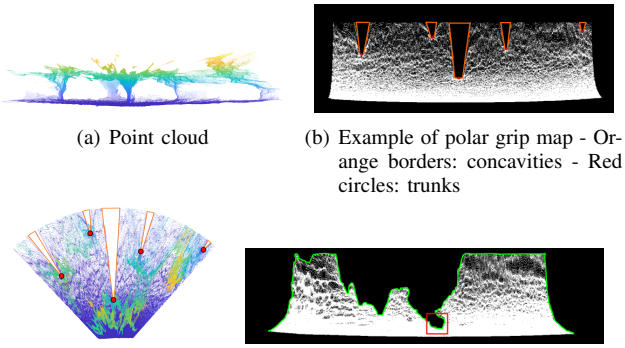
This navigation strategy appears to be very well adapted to orchards for several reasons. First, as a reactive control strategy is used, there is no need for building an orchard metric map or for localizing the robot within it. This is of great interest in our context. Indeed, embedded sensor-based localization can lead to inaccuracies with time and obtaining an up to date map of an orchard can be challenging because of its changing nature (branches, pruning, fruits, ...). Next, the position of the trees appears as a relevant and robust feature to navigate safely within the rows and headlands. From them, it is possible to deduce the lines the robot has to move along to efficiently follow a row. Moreover, in previous works [3], it has been shown that the last trees of the rows can be used as references to navigate in the headlands, i.e., to perform U-turn maneuvers allowing the robot to transition from one row to the next one. Moreover, in commercial orchards, it appears more suitable to use tree trunks as characteristic features for navigation rather than to compute the free space in front of the robot and use it to determine a path, as it is done for example in [4]. One reason is that this computed free space can be severely distorted when branches are present in the field of view of the sensors or the non-uniform distribution of the branches around the tree. Such a distortion can cause path deformation and affect navigation robustness. Another reason is that free space in the headlands does not provide any valuable information to find the next row and reach it. Thus, the knowledge of the trees positions appears to be a key feature in order to perform orchard navigation. It has then be decided to provide the robot the ability of visually detecting trees.

This paper presents the design of a tree detection algorithm aiming at detecting the trunks of the surrounding trees and providing their position with respect to the current robot frame. The algorithm processes the range component of point cloud data acquired by low-cost stereo vision systems in real time. The adoption of low-cost sensors makes it possible to eventually use several of them to provide a large field of view. Detecting tree trunks using only range data makes the approach more robust to changes and discontinuities in

[1]Departement of Biological and Agricultural Engineering, University of California, Davis, CA, 95616, USA

[2]CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France

[3]Univ de Toulouse, UPS, LAAS, F-31400, Toulouse, France



(a) Point cloud

(b) Example of polar grip map - Orange borders: concavities - Red circles: trunks

(c) Top view of the point cloud

(d) Example of polar grip map - Green line: ridge of the shape

Fig. 1. Example of point cloud in an orchard

illumination, which can be severe in orchards.

The proposed method operates by detecting the empty spaces ("shadows") in the point cloud created by the trees. Indeed, when processing data from a single view of a stereo vision system, it is not possible to compute 3D points behind the objects present in the scene. The empty spaces can then be seen as the shadows of the trees in the point cloud. Figure 1(c) presents an example of a single point cloud acquired in an orchard. The empty spaces are represented by the orange triangles, whereas the red circles represent the trunks of the trees. The proposed tree detection algorithm detects the tree shadows (orange rectangles) and their origins (red circles) in the point cloud, and can be briefly described as follows. First, the 3D points belonging to the ground are extracted from the point cloud. Next, these points are used to create 2D grid maps expressed in polar coordinates. As it can be seen in figure 1(b), the computed points form a concave shape, where the shadows of the trees correspond to the concavities[1]. The bottom point of a concavity corresponds to the trunk.

In the second part of the tree detection process the concavities and their bottom points are detected, and concavity areas are estimated. Figure 1(b) suggests that it may be possible to represent the top boundary of the polar grid map by a smooth function. If that were so, finding the bottom of a concavity would be equivalent to determining its local minimum by using a gradient descent. However, as it can be seen in figure 1(d) where the green line represents the boundary, branches and irregular trunk shapes in the camera field of view create shapes in the grid map with boundaries that cannot be represented by single-valued functions. For this reason, this paper presents a novel concavity detection algorithm relying on 2D grid maps at different resolutions that takes advantage of the upward orientation of the concavities. Moreover, the algorithm does not filter the shape in the grid map in order to avoid shape distortion that would result in inaccuracies in the estimation of the trunk position. Instead, it processes the non-smooth contours of the shape which are the result of noise introduced by the low-cost sensors. Finally, the computed bottom points of the concavities, *e.g.,* the trunks, are filtered using a density map to deal with the false detections. By combining the ground extraction process with the concavity detection algorithm, a robust and efficient method is obtained to estimate tree positions. Moreover, its implementation allows its use for real-time control, as the obtained computation times are smaller than 15 ms (see experimental section). Thus, the proposed method allows to robustly and efficiently provide the features which are required to perform exteroceptive feedback based navigation in orchards, both inside rows and in headlands.

The rest of the paper is organized as follows. Next section provides an overview of the related works. The third section presents the robot and its stereo cameras. The fourth section presents the point cloud processing pipeline and focuses on a novel concavities detection algorithm. The last section presents and discusses experimental results obtained in orchards.

[1]Concavities are the difference between the original shape and its convex hull.

II. RELATED WORKS

The proposed method addresses the detection of tree trunks by finding concavities. It is intended to provide features for navigation purposes. Thus the following section will review works related to these three domains, namely: orchard navigation, tree detection and concavities determination.

A. Orchard navigation

The existing orchard navigation systems mostly focus on the navigation through the rows. Two main strategies have been identified. The first one consists of matching the acquired data with a map to localize and control the robot [5], [6]. However, orchards are highly changing environments as the trees aspect greatly depends on seasons (presence of fruits or not, ...) or on agricultural treatments (pruning, ...). Thus, characterizing them with a metric map is not always easy and accurate enough.

The second approach relies on the extraction of lines representing the tree rows. In [7], [8], the lines are computed in the image space, whereas in [9], [10], [11] and [12] they are extracted from point clouds either by using a Hough transform, or by using a RANSAC algorithm coupled with an extended Kalman filter. In this second approach, the branches and the leaves are taken into account to compute the lines, which leads to inaccuracies when they are not homogeneously distributed around the trunk. For this reason, it seems mandatory to first extract the position of the trunks, then to compute the lines fitting the rows.

Moreover, the tree detection algorithm can be used to provide exteroceptive feedbacks during the headland navigation. In the previously mentioned works, the headland navigation is not addressed or simply performed using pure dead-reckoning. In [3], we have presented a navigation strategy and the related controllers allowing to make the robot switch from the current row to the next one. The strategy relies on the position of the trees in both the rows and the headlands.

B. Tree detection

The tree detection problem has been addressed in works related to orchards and other agricultural environments. The

work presented in [13] proposes to identify trees to catalog them. To do so, a trunk localization is performed thanks to the point clouds acquired with a LIDAR. However, the presented approach is not relevant to our case. Indeed, the trunk detection is performed off-line, with the whole orchards previously scanned, in order to easily extract the rows based on a simple distance criteria.

In [14] the authors present an algorithm able to generate a navigation path in an orchard for a harvesting robot based on machine vision. It dynamically recognizes the main tree trunk area from orchard images. However, it is shown that this algorithm is extremely affected by the presence of weeds in the orchards.

[15] focuses on the trunk detection of oil-palm using a Kinect camera. The data processing uses the Viola and Jones detector in the image space combined with a Hough transform to extract line from depth data. In this work, the trunk has to fill a major part of the field of view, making this approach suitable only for trees with tall trunks.

The work presented in [16] proposes to detect trees using a camera coupled with a laser scanner. It relies on stochastic models, obtained during a pre-navigation step, to detect trees in both Cartesian and image spaces. Despite the reported high accuracy, such a system has two issues. First, it relies on a planar scan of the environment, which might lead to errors when the trunks are occluded with leaves. Then, it only provides a narrow field of view, and increasing the number of the selected sensors might be an expensive solution.

In [17], the trunks of dwarf orange trees are detected and localized thanks to cameras coupled to ultrasonic sensors mounted on pan-platforms. The tree recognition is based on several features such as color, texture and contour used to train a support vector machine classifier. Moreover, for each detected tree, the ultrasonic sensor mounted on the same pan-platform as the camera, provides the distance to it. In this work, a pair of camera/ultrasonic sensors can only detect, localize and track one tree at the time.

In [18], the authors segment images of citrus trees to extract the areas corresponding to the fruits and trunks. They use elliptical regions in the YCrCb color space to extract the fruits and trunks areas from the background. However, in this approach, only one tree is considered at the time and the results are sensitive to illumination.

C. Concavity detection

In [19], the authors give an overview of the methods developed to detect inflection points: curvature based [20], skeleton based, chord based [21] and polygon approximation [22]. These methods require a smooth contour and poorly perform in presence of a noise in the extracted shape. However, because one aims at accurately computing the position of the trees, it is not suitable to remove noise with morphological operations. Indeed, these latter distort the original shape leading to inaccuracies in the tree position computation. Thus, these methods are not adapted to our case.

III. MATERIAL

To navigate across the orchards, a Toro workman MDE vehicle is used. It is a car-like robot equipped with ZED stereo cameras from Stereolabs (see figure 2). They represent a low-cost solution to acquire point clouds in an outdoor environment and on a long range, up to 20 meters. Moreover, the API computing the point cloud uses NVIDIA Graphics Processing Unit (GPU) to provide a frame rate from 15Hz to 60 Hz. For all these reasons, it has been decided to install four cameras on the robot. Two are mounted in the front and face towards, whereas the two other ones are installed on the left and right sides to provide a large field of view to the system. The acquired data are individually processed for each camera in order to detect the trees surrounding the robot. The obtained features allow to feed the controllers used to drive the robot in the row or in the headland. This paper only focuses on the point cloud processing algorithm. More details on control aspects can be found in [3].

Fig. 2. Toro workman MDE vehicle equipped with 4 stereo cameras

IV. TREE DETECTION SYSTEM

This section first presents the pipeline of methods applied to the acquired point cloud. As explained before, the goal is to detect tree trunks by computing the concavity inflection points. Most of the processes used in this pipeline belong to classical tools from point cloud and image processing. However, the concavity detection algorithm has been specifically developed for this project. Firstly, an overview of the whole pipeline is presented before focusing on our main contribution, the concavity detection algorithm.

A. The overall processing pipeline

The pipeline designed to compute the position of the trees is presented in figure 3. It is split into two parts, the first one being related to the ground extraction and the second one to the tree detection itself.

1) The ground detection: The first task consists of extracting the points representing the ground from the acquired point cloud XYZ. To do so, three steps are performed. First, to save computation time[2], XYZ is randomly sampled to generate

[2]It would have been possible to directly extract the points belonging to the ground from XYZ thanks to a least square regression using a RANSAC algorithm, as it is done in step 2 with XYZ_S. However, this approach is more time consuming than a process combining steps 1, 2 and 3.

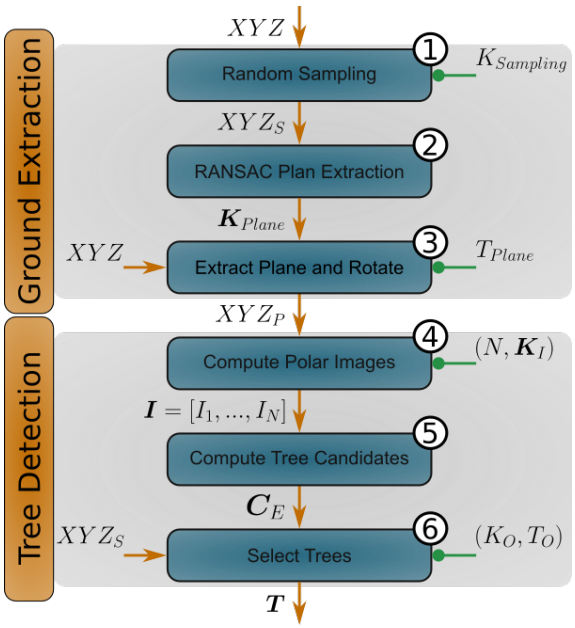
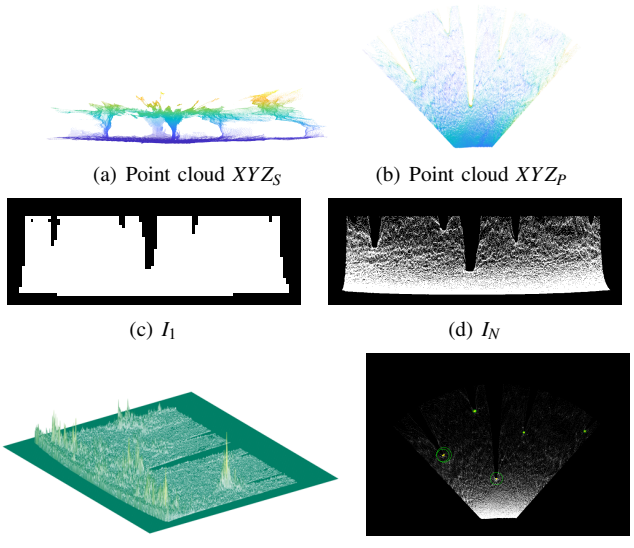


Fig. 3. Pipeline to detect the trees from point cloud XYZ

a point cloud XYZ_S (figure 4(a)) with a smaller dimension $K_{Sampling}$ (step 1). Next, the parameters $\boldsymbol{K}_{plane}$ of the plane characterizing the ground are computed thanks to a least square regression using a RANSAC algorithm (step 2). From this result, XYZ_P, the point cloud containing all the points from XYZ belonging to the plane, is computed (step 3). It is determined by including all the points from XYZ which are at an Euclidean distance smaller than T_{Plane} from the plane defined by $\boldsymbol{K}_{plane}$ (figure 4(b)). Finally, XYZ is rotated to align the ground and camera axis.



(a) Point cloud XYZ_S (b) Point cloud XYZ_P

(c) I_1 (d) I_N

(e) Density map M_D (f) Selected trees

Fig. 4. Intermediate results of the pipeline

2) The tree detection: Once the points representing the ground have been extracted, it remains to detect the trees. This is done thanks to the concavity detection algorithm, detailed in section IV-B, which allows to compute the concavities inflection points. It relies on 2D polar representation of the point cloud at N different resolutions. Incrementally increasing the resolution allows to deal with the noise inherent to a point cloud, as well as to decrease the required computing time. Thus, the next step of the pipeline consists in computing N images of the points belonging to the plane expressed in polar coordinates, at different resolutions: $\boldsymbol{K}_I = [\mu_1, \theta_1, ..., \mu_N, \theta_N]$, where μ_i and θ_i, with $i \in [1, .., N]$, are respectively the length and angle represented by each pixel (pipeline step 4). In the computed images, the shadows now correspond to vertical concavities (see figures 4(c), 4(d)).

To detect these concavities, the algorithm (step 5) places coins at the top of the images and then drop them. The coins move down until they are ejected or stuck. This process is repeated for the N images. At the end, it provides, in the last image I_N, a vector $\boldsymbol{C}_E$ containing the N_C tree candidates positions.

However, due to uneven ground, tall grass or some left over equipments, it may happen that a shadow which does not represent a tree appear is detected. It is then necessary to add a tree selection process (pipeline step 6) which will reduce the number of false positive elements. It consists in checking if there are branches or leaves above each detected candidate point (which is supposed to be a trunk). Thus a candidate is considered to be a tree if the number of points above it is sufficiently important. To do so, a discrete map M_D whose resolution is similar to I_N is computed. Each cell of M_D gives the number of points of XYZ_S belonging to the area it represents (figure 4(e)). Finally, the values of the cells belonging to a square of dimension K_O and centered on the candidate are summed to compute an occupancy score S_O. If S_O happens to be higher than a predefined threshold T_O, the candidate point is added to the tree vector T which contains all the detected trunk positions.

B. Concavity Detection Algorithm

The concavity detection algorithm aims at detecting the concavities, computing their inflection points, and provide an estimate of their area. To do so, it uses N images representing the ground in polar coordinates at different resolutions. They are stored in an image vector $\boldsymbol{I} = [I_1, ..., I_n]$, where I_1 has the lowest resolution and I_N has the highest one. Algorithm 1 provides an overview of the method and presents the three steps successively applied to each image. First, the initial

Algorithm 1: trunkCandidatesComputation

input : $\boldsymbol{I}, N$
output: $\boldsymbol{C}_E$
1 **for** $i = 1 : N$ **:**
2 $\quad \boldsymbol{C}_S =$ `startingCoordinates` $(\boldsymbol{I}[i], \boldsymbol{C}_E, i)$
3 $\quad \boldsymbol{C}_C =$`detectConcavities` $(\boldsymbol{I}[i], \boldsymbol{C}_S)$
4 $\quad \boldsymbol{C}_E =$`exploreConcavities` $(\boldsymbol{I}[i], \boldsymbol{C}_C)$

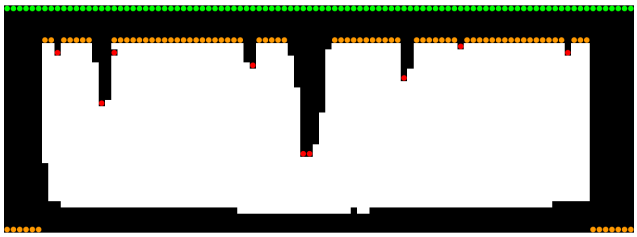
positions of the coins are computed. Second, the coins drop down until the entrance of a concavity is detected or they are ejected. Finally, the non-ejected coins explore the concavity until they reach its bottom. Once this process is over, the final

positions $\boldsymbol{C_E}$ of the N_C tree candidates obtained in the image I_N are provided. The three previous steps are detailed hereafter in the algorithms 2, 3 and 4.

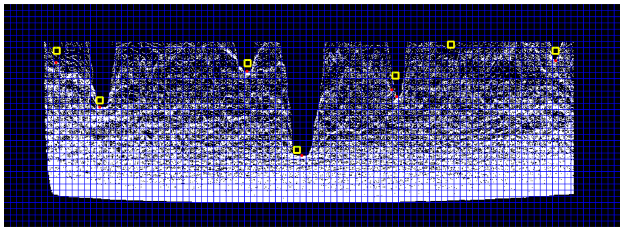
Algorithm 2: startingCoordinates

input : $I, \boldsymbol{C_E}, i$
output: $\boldsymbol{C_S}$

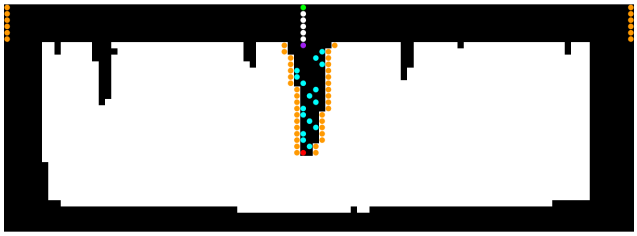
if $i = 1$ **:**
  for $j = 1 :$ `numberOfColumn` (I) **:**
    $\boldsymbol{C_S}[j] = (1, j)$
else:
  for $j = 1 :$ `numberOfColumn` $(\boldsymbol{C_E})$ **:**
    $(x, y) =$ `convertCoordinates` $(\boldsymbol{C_E}[j])$
    $\boldsymbol{C_S}[j] = (x, y)$



(a) Concavity detection at low resolution in I_1 - Green dots: intial positions - Orange dots: ejected coins - Red dots: stuck coins



(b) Concavity detection at high resolution in I_N - Blue lines represent the previous resolution - Yellow squares: initial positions computed from the final positions at the lowest resolution - Red dots: final positions



(c) Example of dropping one coin in I_1 - Green dot: intial position - White dots: dropping step - Purple dot: concavity detected - Cyan dots: concavity exploration - Red dot: final position - Orange dots: borders compupted for each row in this example

Fig. 5. Example of concavity detection at different resolutions

Algorithm 2 aims at performing the first step: the computation of starting coordinates. It provides the vector $\boldsymbol{C_S}$, which contains the initial coordinates of the coins for the i^{th} image $\boldsymbol{I}[i]$. For I_1, one coin per column is placed at the top row of the image (represented by the green dots in figure 5(a)). For the other images I_i, with $i \in [2, N]$, the initial coordinates are randomly selected within the area obtained by converting the final coordinates $\boldsymbol{C_E}$ in image I_{i-1} to the current resolution (represented by the yellow square in figure 5(b)).

Once the coordinates vector $\boldsymbol{C_S}$ has been computed, algorithm 3 allows to drop the coins to detect the entrance of the concavities in an image I. To do so, for each element present in $\boldsymbol{C_S}$, the closest pixels in the right and left directions, belonging to the same row, and with a non-null value are calculated. From now on, they are named left and right borders, and denoted B_l, B_r (orange spots in figure 5(c)). If B_l or B_r is one of the image border, then the coin has to move to the next row (white dots in figure 5(c)). If the next pixel has a non-null value, then the coin is ejected. Otherwise it is dropped to the next row at the same column. If both B_l and B_r represents pixels with non-null value, then the entrance of a concavity has been detected (purple dot in figure 5(c)). The current location of the coin is then added to $\boldsymbol{C_C}$, vector containing the position of the concavity entrances.

Algorithm 3: detectConcavities

input : $I, \boldsymbol{C_S}$
output: $\boldsymbol{C_C}$

$k = 1$
for $j = 1 :$ `numberOfColumn` $(\boldsymbol{C_S})$ **:**
  $(x, y) = \boldsymbol{C_S}[j]$
  $ejected =$ **FALSE**, $borders =$ **FALSE**
  while $!ejected$ *and* $!borders$ **:**
    $(B_l, B_r) =$ `findBorders` (I, x, y)
    $borders =$ `evaluateBorders` (B_l, B_r)
    if $borders =$ ***FALSE*** **:**
      $y = y + 1$
      $ejected =$ `evaluateNextRow` (I, x, y)
    else:
      $\boldsymbol{C_C}[k] = (x, y)$
      $k = k + 1$

Algorithm 4: exploreConcavities

input : $I, \boldsymbol{C_C}$
output: $\boldsymbol{C_E}$

$k = 0$ **for** $j = 1 :$ `numberOfColumn` $(\boldsymbol{C_C})$ **:**
  $(x, y) = \boldsymbol{C_C}[j]$
  $ejected =$ **FALSE**, $stuck =$ **FALSE**
  while $!ejected$ *and* $!stuck$ **:**
    $(x, y) =$ `dropCandidates` (I, x, y)
    $(B_l, B_r, ejected) =$ `findBorders` (I, x, y)
    if $!ejected$ **:**
      $\boldsymbol{C_N} =$ `nextCandidates` (I, x, y, B_l, B_r)
      $NOC =$ `numberOfColumn` $(\boldsymbol{C_N})$
      if $NOC > 0$ **:**
        $r =$ `random` (NOC)
        $(x, y) = \boldsymbol{C_N}[r]$
      else:
        $\boldsymbol{C_E}[k] = (x, y)$
        $k = k + 1$
        $stuck =$ **TRUE**

The concavities being detected, algorithm 4 aims at exploring them to find the point of inflection (cyan dots in figure 5(c)). To do so, each element of $\boldsymbol{C_C}$ is moved to the next row

and the borders B_l and B_r are computed. If at least one of the border corresponds to the image border, the coin is ejected. Otherwise, all the non-null pixels of the next row within the borders B_l and B_r are placed in the next candidate vector $\boldsymbol{C}_N$. Next, if $\boldsymbol{C}_N$ is not empty, the next position is randomly selected among the elements of $\boldsymbol{C}_N$. Otherwise, it means that the coin is stuck at the bottom of a concavity and is added to the final coordinates vector $\boldsymbol{C}_E$ (red dot in figure 5(c)).

Remark 1: The exploration of the concavity is randomly done in order to deal with noise. Indeed, in the case of a small concavity inside another concavity, due to the presence of noise, a deterministic approach could lead to stuck all the coins in the small concavity. A random approach allows to minimize the impact of noise in the exploration process.

Remark 2: The final coordinates vector $\boldsymbol{C}_E$ obtained for the last image I_N corresponds to the coordinates of the tree candidates. It is the output of the concavities detection algorithm.

V. EXPERIMENTS

(a) Orchard #1 winter

(b) Orchard #1 spring

(c) Orchard #2

(d) Orchard #3

Fig. 6. View of the orchards

The above point cloud processing pipeline has been implemented on our Toro Workman robot. It is equipped with ROS middleware which is running on a laptop equipped with an Intel Core i7-6700H, 16 GB RAM and a NVIDIA GeForce GTX 960M GPU. The vision system is composed of four ZED cameras, setup with a resolution of 720p. Each camera provides a point cloud of 777600 points in a 15m range. In order to evaluate the performances of the proposed approach, data from three different orchards were collected (see figure 6). The first one is a plum orchard and the data gathered during winter (Data #1) and spring (Data #2). The two other ones are walnut (Data #3) and almond (Data #4) orchards and the data collected during spring. In table I, the average row width, tree spacing and trunk diameter are given for the different data sets. Finally, the data were both acquired in the rows and the headlands.

The tree detection algorithm has been implemented using the C++ and CUDA languages, as well as the openCV and PCL libraries. Moreover, its parameters have been fixed as follows: size of the sample $K_{Sampling} = 350000$; distance to plane threshold $T_P = 0.25$ m; number of images $N = 2$ [3] with $\mu_1 = 0.5$ meter, $\theta_1 = 1$ degree, $\mu_2 = 0.05$ meter, and $\theta_2 = 0.1$ degree. Moreover, the dimension of the density square is $K_O = 11$ and the occupancy threshold $T_O = 200$. In table II, the performances of the proposed approach are given for the four data sets. It provides the precision and recall of the tree detection algorithm. The results are given by distance range: from 0 to 5 m, from 5 to 10 m and over 10 m. The position of the cameras was changed between winter and spring to make them face downward. Thus, the data acquired during spring do not allow to detect trees beyond 10 meters. The ZED cameras provide relevant high quality data up to 5 meters. Thus, for this range, the tree detection algorithm offers great performances. Indeed, for the four data sets, precision is above 97% and recall is above 96%. For the 5 to 10 meters range, the precision is still good with at least 97%. However, recall drops especially for the second and third data sets with respectively 72.4% and 83.1%. Two factors explain the drop of recall performances at this range. First, the point clouds computed by the stereo vision system contains more noise and the position of the points is less accurate. Second, especially for the spring data sets, the field of view might be blocked by leaves, making impossible for the camera to acquire data related to further trees. However, the performances obtained within the 10 m range is sufficient to navigate within a commercial orchard.

TABLE I
DESCRIPTION OF THE ORCHARDS - AVERAGE WIDTH, SPACING AND DIAMETER EXPRESSED IN METER

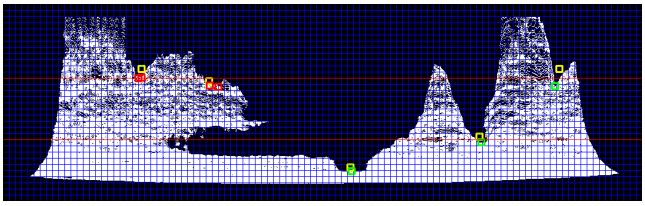
Data #	Season	Type	Width	Spacing	Trunk diameter
1	Winter	Plum	5	3	0.5
2	Spring	Plum	5	3	0.5
3	Spring	Walnut	4	3.5	0.35
4	Spring	Almond	6	2.5	0.35

Figure 7 shows four detailed results of tree detection in the last image I_2. The yellow squares represent the position of the tree candidates computed in image I_1, while the green squares are the selected candidates. Finally, the red squares correspond to the candidates rejected based on S_O and T_O. The first image 7(a) represents the tree detection for the front left camera in the first orchard. The three trees in the field of view of the camera are detected, while the false positive detections are filtered thanks to the density based criteria. Moreover, this case is an example of the necessity to randomly explore the concavities. Indeed the two left trees belong to the same large concavity with four points of inflection (but only two of them correspond to trees). Thus, if the coins are moved in a systematic way, by moving to the middle of the next row for example, then only one of the four inflection point is detected. In images 7(b) and 7(c) the results for the front and left sided sensors in orchards

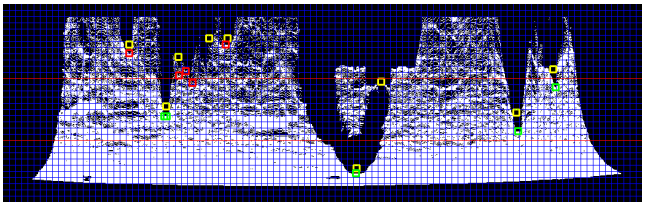
[3]For the current resolution only 2 images are required. Indeed, the low resolution of the first image I_1 allows to obtain a detailed representation of the scene while removing most of the noise. Thus, the results obtained in I_1 can be directly used in the high resolution image I_2 without risking false detections. In the case of a higher resolution of the image provided by the camera or a more noisy representation of the scene, it might be required to use a higher value for N.

TABLE II
TREE DETECTION PERFORMANCE - RESULTS ARE GIVEN FOR 0 TO 5 M (0-5), 5 TO 10 M (M) AND OVER 10 M (10+)

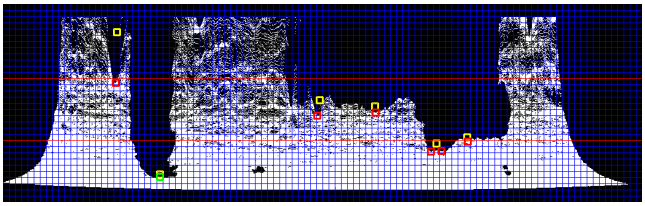
	0-5 m	5-10 m	10+ m	Total
Data #1				
Number of tree	38	67	49	154
Precision	100%	98.5%	64.7%	90.2%
Recall	97.3%	91%	44.8%	77.9%
Data #2				
Number of tree	198	213	N/A	411
Precision	97.5 %	98.7%	N/A	98%
Recall	98.9%	72.4%	N/A	85.1%
Data #3				
Number of tree	89	101	N/A	190
Precision	97.7%	97.6%	N/A	97.7%
Recall	96.6%	83.1%	N/A	89.4%
Data #4				
Number of tree	107	60	N/A	167
Precision	97.1%	98.2%	N/A	97.5%
Recall	96.2%	90%	N/A	91.5%



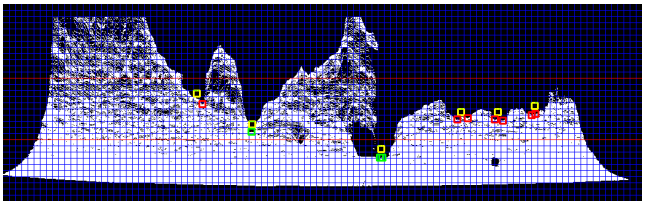
(a) Front left sensor from data #2



(b) Front left sensor from data #3



(c) Side left sensor from data #3



(d) Front right sensor from data #4

Fig. 7. Example of tree detection in image I_2 - Yellow squares: tree candidates computed in I_1 - Green squares: selected candidates - Red squares: rejected candidates - Red lines: 5 and 10 m limits

TABLE III
AVERAGE PROCESSING TIME MEASURED WITH 100 POINT CLOUDS (MS)

1: Random sampling	2.5 ms	2: Ransac	4.7 ms
3: Extract plane and rotate	1.8 ms	4: Polar images	1 ms
5: Concavity detection	1 ms	6: Select tree	0.1 ms

#2 can be seen. Once again all the trees in the field of view of the cameras are detected while the false positive detections are filtered. Moreover, the tree in the 0-5 meter range is a V-shape tree. It was then possible to perceive the ground between the two main branches, creating a non-empty concavity in the image. That fully explains why it has been decided to detect the trees by developing a new concavity detection algorithm, instead of using a function-based method such as the gradient descent one. Finally, the image 7(d) represents the result of the tree detection for the right front camera in the last orchard. In this case, only two of the three trees present in the field of view are detected. The inflection point of the non-detected one, the left red square in the figure, was detected by the concavity detection algorithm. However, the density score was not high enough to be selected as a tree.

The tree detection algorithm presented in this paper seems to perform at least as well as the ones presented in the literature. However, it is difficult to compare the obtained performances because of the different sensors and platform used, and types of orchards considered. Moreover, the listed results do not provide any information regarding the distance between the sensors and the detected trees. Here, the performances reported in previous works are summarized. The method presented in [14] performs in apple orchards and allows to obtain a correct recognition rate of 91.7% with a single camera. In [15], the authors report a detection rate of 97.8% and a false-positive rate of 15.2% for an oil-palm tree detection performed with a stereo camera. The algorithm presented in [16] allows to detect persimmon trees thanks to a camera coupled with a laser scanner. Precision rates ranging from 96.77% to 98.94% and recall rates from 93.75% to 96.88% are reported. In [17], orange trees are detected using cameras and ultrasonic sensors. The proposed algorithm allows to obtain a recall ratio of 92.14% and an accuracy ratio of 95.49%. Finally, the citrus tree detection using a camera presented in [18], performs with a true-positive ratio of 90.8% and a false positive-ratio of 95.49%.

In table III, the time required by our platform to compute the different steps of the pipeline are given. It is shown that the average total time to detect the trees is about 10 ms with peaks up to 15 ms. It should be noticed that a previous implementation not using CUDA was requiring an average processing time of 150 ms to process the entire pipeline.

VI. CONCLUSION

In this article, a robust, quick and accurate perception system dedicated to tree detection in orchards was developed. It proposes a vision-based system made of four low cost stereo cameras which provide a dense representation of the agricultural environment around the vehicle. In our method,

the point cloud is processed to detect trunks instead of the whole trees to avoid considering leaves or branches which might reduce the result accuracy. To do so, the key idea is to find the tree trunks by detecting their shadows which are materialized by concavities in the obtained point cloud. The designed algorithm is made of two steps, respectively consisting in extracting the ground and detecting the trees. This latter objective is fulfilled thanks to an original algorithm whose execution time is around 1ms, allowing its use in a control context. The pipeline was implemented on our robot using ROS middleware and validated using various sequences of data taken in different orchards. The obtained results have been proven to be quite satisfying. A comparison with other approaches has also been proposed. Our future works will be mainly orientated towards the coupling of our perception system to our control strategies. Indeed, the positions of the trees are now available with a sufficient accuracy to compute the control laws. Then, it will be necessary to build a topological map of the environment. Finally, it will remain to sequence the row following and U-turn controllers proposed in [3] depending on the robot position in the orchard to complete the navigation task.

ACKNOWLEDGEMENT

This work was supported by USDA-NIFA under awards: 2016-67021-24532 (AFRI) and 2016-67021-24535 (National Robotics Initiative).

REFERENCES

[1] J. A. Foley, N. Ramankutty, K. A. Brauman, E. S. Cassidy, J. S. Gerber, M. Johnston, N. D. Mueller, C. O'Connell, D. K. Ray, P. C. West, C. Balzer, E. M. Bennett, S. R. Carpenter, J. Hill, C. Monfreda, S. Polasky, J. Rockström, J. Sheehan, S. Siebert, D. Tilman, and D. P. M. Zaks, "Solutions for a cultivated planet," *Nature*, vol. 478, pp. 337–342, Oct. 2011.

[2] J. F. Reid, "The impact of mechanization on agriculture," *Bridge*, vol. 41, no. 3, pp. 22–29, 2011.

[3] A. Durand-Petiteville, E. Le Flecher, V. Cadenat, T. Sentenac, and S. Vougioukas, "Design of a sensor-based controller performing u-turn to navigate in orchards," *International Conference on Informatics in Control, Automation and Robotics*, vol. 2, pp. 172–181, 2017.

[4] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*, 2005.

[5] J. C. Andersen, O. Ravn, and N. A. Andersen, "Autonomous rule-based robot navigation in orchards," *IFAC Proceedings Volumes*, vol. 43, no. 16, pp. 43–48, 2010.

[6] S. J. Moorehead, C. K. Wellington, B. J. Gilmore, and C. Vallespi, "Automating orchards: A system of autonomous tractors for orchard maintenance," in *International Conference on Intelligent Robots and Systems Workshop on Agricultural Robotics*. IEEE/RSJ, 2012.

[7] M. Sharifi and X. Chen, "A novel vision based row guidance approach for navigation of agricultural mobile robots in orchards," in *Automation, Robotics and Applications (ICARA), 2015 6th International Conference on*. IEEE, 2015, pp. 251–255.

[8] V. Subramanian, T. F. Burks, and A. Arroyo, "Development of machine vision and laser radar based autonomous vehicle guidance systems for citrus grove navigation," *Computers and Electronics in Agriculture*, vol. 53, no. 2, pp. 130 – 143, 2006.

[9] O. C. Barawid Jr, A. Mizushima, K. Ishii, and N. Noguchi, "Development of an autonomous navigation system using a two-dimensional laser scanner in an orchard application," *Biosystems Engineering*, vol. 96, no. 2, pp. 139–149, 2007.

[10] B. Hamner, M. Bergerman, and S. Singh, "Autonomous orchard vehicles for specialty crops production," in *2011 Louisville, Kentucky, August 7-10, 2011*. American Society of Agricultural and Biological Engineers, 2011, p. 1.

[11] M. Bergerman, S. Singh, and B. Hamner, "Results with autonomous vehicles operating in specialty crops," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 1829–1835.

[12] J. Zhang, A. Chambers, S. Maeta, M. Bergerman, and S. Singh, "3d perception for accurate row following: Methodology and results," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 5306–5313.

[13] B. Suchet, U. J. P., N. J. I., and S. Salah, "A pipeline for trunk detection in trellis structured apple orchards," *Journal of Field Robotics*, vol. 32, no. 8, pp. 1075–1094.

[14] B. He, G. Liu, Y. Ji, Y. Si, and R. Gao, "Auto recognition of navigation path for harvest robot based on machine vision," in *Computer and Computing Technologies in Agriculture IV*, D. Li, Y. Liu, and Y. Chen, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 138–148.

[15] M. A. Juman, Y. W. Wong, R. K. Rajkumar, and L. J. Goh, "A novel tree trunk detection method for oil-palm plantation navigation," *Computers and Electronics in Agriculture*, vol. 128, pp. 172 – 180, 2016.

[16] N. Shalal, T. Low, C. McCarthy, and N. Hancock, "Orchard mapping and mobile robot localisation using on-board camera and laser scanner data fusion part a: Tree detection," *Computers and Electronics in Agriculture*, vol. 119, pp. 254 – 266, 2015.

[17] X. Chen, S. Wang, B. Zhang, and L. Luo, "Multi-feature fusion tree trunk detection and orchard mobile robot localization using camera/ultrasonic sensors," *Computers and Electronics in Agriculture*, vol. 147, pp. 91 – 108, 2018.

[18] T.-H. Liu, R. Ehsani, A. Toudeshki, X.-J. Zou, and H.-J. Wang, "Detection of citrus fruit and tree trunks in natural environments using a multi-elliptical boundary model," *Computers in Industry*, vol. 99, pp. 9 – 16, 2018.

[19] S. Zafari, T. Eerola, J. Sampo, H. Kälviäinen, and H. Haario, "Comparison of concave point detection methods for overlapping convex objects segmentation," in *Image Analysis*, P. Sharma and F. M. Bianchi, Eds. Cham: Springer International Publishing, 2017, pp. 245–256.

[20] ——, "Segmentation of partially overlapping nanoparticles using concave points," in *International Symposium on Visual Computing*. Springer, 2015, pp. 187–197.

[21] S. Kumar, S. H. Ong, S. Ranganath, T. C. Ong, and F. T. Chew, "A rule-based approach for robust clump splitting," *Pattern Recognition*, vol. 39, no. 6, pp. 1088–1098, 2006.

[22] W.-H. Zhang, X. Jiang, and Y.-M. Liu, "A method for recognizing overlapping elliptical bubbles in bubble image," *Pattern Recognition Letters*, vol. 33, no. 12, pp. 1543–1548, 2012.