Développement d'algorithmes de traitement d'image pour la détection et la caractérisation des feuilles de betteraves sucrières

N. Sorel

INSTITUT MÉDITERRANÉEN D'ÉTUDES ET DE RECHERCHE EN INFORMATIQUE ET ROBOTIQUE

Développement d'algorithmes de traitement d'image pour la détection et la caractérisation des feuilles de betteraves sucrières

Mémoire de fin d'études

Diplôme de niveau I (BAC+5) reconnu par l'Etat de "Manager de systèmes informatique et robotique"

Réalisé par Nicolas SOREL
Année 2015-2018
Promotion Dessertine
Version 1

Maître de stage : Bernard BENET

Tuteur école : Blaise MADELINE

Validation par l'entreprise :

Résumé

Ce mémoire porte sur le développement d'algorithmes de perception par vision artificielle pour le guidage d'un bras robotisé évoluant en milieu agricole, et plus précisément dans des champs de betteraves sucrières. Pour un bras robotisé, la détection de son environnement et des objets qu'il doit analyser (dans ce cas, les feuilles de betterave) est importante afin de prendre de bonnes décisions au niveau de sa commande.

De nombreuses contraintes sont à prendre en compte lors de la conception de ces algorithmes, parmis lesquelles la variation de luminosité en milieu extérieur, les différences de forme des feuilles et plants de betterave, ou encore la mobilité du bras robotique. Les méthodes de vision artificielle présentées dans ce mémoire tiennent compte de ces contraintes.

Mots-clés : traitement d'image, perception active, bras manipulateur, phénotypage

Abstract

This report deals with the development of perception algorithms for the guidance of a robotic arm operating in an agricultural environment, and more precisely in sugar beet fields. For an automatic robotic arm, the detection of its environment and the objects it has to analyze (in this case, the beet leaves) is important in order to make good decisions at its command level.

Many constraints must be considered when designing these algorithms, among which the brightness variation in the outdoor environment, the differences in shape of leaves and beet plants, or the mobility of the robotic arm. The methods presented in this report make it possible to take into account these constraints.

Keywords : image processing, active perception, manipulator arm, phenotyping

Remerciements

Je tiens à remercier toutes les personnes m'ayant aidé lors de la réalisation de mon stage.

Tout d'abord un grand merci à Bernard BENET, mon maître de stage, pour son encadrement, ses conseils et ses réponses aux questions posées.

Je remercie ensuite l'équipe ROMEA ,et plus généralement l'ensemble du personnel de l'IRSTEA pour leur accueil chaleureux et leur participation à la bonne ambiance de l'entreprise.
Je remercie également toute l'équipe de l'IMERIR, et notamment mon tuteur école, M.MADELINE, pour le suivi effectué.

Enfin je tiens à remercier les personnes suivante pour le soutien particulier qu'elles m'ont apporté :
-Adrian COUVENT, pour les idées données et l'appui fourni
-Camille DUBOS, doctorante qui travaille également sur le projet
-Roland LENAIN, qui dirige l'équipe ROMEA
-Vincent ROUSSEAU, pour son aide et ses conseils précieux en informatique et sur ROS

Sommaire

Introduction

Présentation de l'entreprise

Présentation générale

Irstea (**I**nstitut **N**ational de **R**echerche en **S**ciences et **T**echnologies pour l'**E**nvironnement et l'**A**griculture) est un établissement public à caractère scientifique et technologique (EPST). Réparties en France dans 9 centres, ses activités de recherche et d'expertise sont tournées vers l'action et l'appui aux politiques publiques (figure 1). Ses principaux domaines de recherche concernent les enjeux majeurs d'une agriculture responsable et de l'aménagement durable des territoires, la gestion de l'eau et les risques associés, sécheresse, crues, inondations, l'étude des écosystèmes complexes et de la biodiversité dans leurs interrelations avec les activités humaines.

FIGURE 1 : *Implantation des centres Irstea en France*

Historique

L'irstea est née de la fusion de deux centres publics de recherche, le CTGREF (Centre Technique du Génie Rural et des Eaux et Forêts) et le CNEEMA (Centre National d'Etude et d'Expérimentation de Machinisme Agricole) en 1981. Il s'appelait alors Cemagref (Centre national du machinisme agricole, du génie rural et des eaux et forêts) mais a été renommé en 2012 pour mieux coïncider avec la réalité des recherches effectuées, qui ont évolué en 30 ans. Au premier janvier 2020, il est prévu de fusionner l'irstea avec l'INRA (Institut National de la Recherche Agronomique) car ils ont des thèmes de recherche proche.

Chiffres-clés

1533 collaborateurs

- 9 centres régionaux
- 3 départements de recherche : eaux, écotechnologie, territoires
- 14 unités de recherche, 5 unités mixtes
- 1129 chercheurs, ingénieurs, doctorants et post-doctorants

Les chiffres de la recherche

- 79 famille de brevets
- 146 entreprises partenaires
- 25 projets européens en cours
- 109,5 millions d'euros de budget annuel, dont 24% de ressource propre

L'unité de recherche TSCF

L'UR TSCF est composée de 3 équipes de recherche qui rassemblent 60 agents :

- Plateau de Recherche Technologique Pôle Épandage Environnement (PRT PEE)
- RObotique et Mobilité pour l'Environnement et l'Agriculture (ROMEA)
- Systèmes d'information agri-environnementaux communicants (Copain)

Elle mobilise les sciences pour l'ingénieur et les sciences et technologies de l'information et de la communication pour conduire des recherches sur les méthodes et outils pour une ingénierie des systèmes agro-environnementaux. Elle conduit également des activités de recherche, d'expertise et d'essai dans le domaine de la sécurité et des performances des agroéquipements pour contribuer à l'amélioration de la sécurité en agriculture et à la réduction des pollutions d'origine agricole.

L'équipe ROMEA

Au sein de l'unité de recherche TSCF, l'équipe ROMEA, basée à Aubière, conçoit des systèmes reconfigurables et à autonomie partagée, pour accroître les performances et la sécurité des engins œuvrant en milieux naturels, en particulier ceux rencontrés dans l'agriculture. Cette équipe concentre ses recherches sur trois axes :

- La conception de dispositifs de sécurité, aussi bien du point de vue de la conception mécanique que de la réalisation de dispositifs d'assistance.
- La perception de l'environnement et la modélisation du comportement d'un robot et de son interaction avec l'utilisateur.
- Le développement d'algorithmes avancés de commandes ou d'assistance, en prenant en compte les phénomènes perturbants et les incertitudes liées à l'évolution en milieu tout-terrain et dans différentes conditions.

Présentation du projet

Projet Phenaufol

Mon stage de fin d'études s'inscrit dans le cadre du projet Casdar Phenauphol. L'objectif de ce projet est d'automatiser la détection, l'identification et la quantification des maladies foliaires de la betterave (figure 2). Pour cela, il prévoit le développement d'un robot agricole dont le but est de réaliser de l'acquisition d'image sur des champs de betteraves sucrières. Les données images obtenues seront ensuite traitées afin de détecter des maladies foliaires et d'étudier leurs propagation dans les cultures. Les outils et méthodes créés à l'issue de ce projet permettront :

1. de développer d'avantage l'usage des variétés résistantes aux maladies foliaires
2. d'entrevoir de nouvelles pratiques agricoles plus efficientes et plus respectueuses de l'environnement.

Dans une deuxième phase de déploiement, il est prévu d'utiliser le système robotisé et la chaîne de traitement associée pour automatiser le déclenchement des traitements sur les exploitations agricoles.

Ce projet est organisé par l'Institut Technique de la Betterave (ITB). Sa mise en place est réalisée en partenariat avec l'UMR Agroécologie (Dijon) et Irstea (Clermont Ferrand). L'UMR Agroécologie s'occupe de programmer les algorithmes d'identification de maladie foliaire tandis que l'Irstea développe le robot agricole.

FIGURE 2 : Feuille de betterave malade (oïdium)

Objectif

Le but de mon travail de stage est de développer des algorithmes de vision artificielle pour faciliter le guidage du robot et de les implémenter.
Cette application de perception active consiste à détecter et localiser avec précision les plants de betterave, et leurs feuilles associées, à partir des images de caméra. Deux types de caméra sont envisagés pour cette étude : une caméra couleur RGB mais aussi une caméra hyperspectrale pour pouvoir détecter des maladies non détectables avec une caméra couleur classique. Les données géométriques et colorimétriques obtenues par traitement d'image sont utilisées dans la partie contrôle/commande du robot, pour positionner la caméra au dessus de chaques feuille détectée, ce qui permet alors de réaliser des acquisitions d'image en considérant différentes orientations autour de ces feuilles et répondre ainsi aux exigences de la détection de maladie foliaire.

Contraintes

Quelques contraintes sont imposées pour la réalisation des objectifs. L'une de ces contraintes posées est que l'opération de traitement d'image doit être temps réel et s'exécuter dans une durée de l'ordre de la centaine de milliseconde. Une autre contrainte est que le seul capteur utilisé est la caméra couleur. Cette contrainte implique que les données pour le guidage du robot sont uniquement des images couleurs (en deux dimensions).

Des contraintes dues à l'environnement sont également à prendre en considération :

- La variation de lumière en milieu extérieur peut être importante (temps ensoleillé/sombre, passage de nuage, …). La variation de lumière doit être prise en compte car elle impacte directement la couleur des feuilles et des éléments de l'image en général.
- La morphologie des feuilles (forme, taille et orientation) est très différente d'une feuille à l'autre.

Cadre du projet

Environnement d’évolution de l’application

Environnement matériel

Afin de réussir les objectifs du projet, un robot, nommé Bettybot, développé par la société robotnik a été utilisé. Ce robot est composé d’un axe linéaire sur lequel coulisse un bras manipulateur UR5, de universal robot. Une caméra couleur est située à l’extrémité du bras UR5. Une armoire de commande électrique et un ordinateur central assurent la coordination de l’ensemble de ces éléments. Ce système robotisé a pour fonction de réaliser des acquisition d’image sur des plants de betterave. Le bras UR5 est un robot à six degrés de liberté possédant six axes de rotations. Il peut porter jusqu’à 5 kg et possède la particularité d’être “collaboratif”, c’est à dire qu’il possède des capteurs de pression intégrés qui lui permettent de s'arrêter automatiquement après avoir donné un coup.

Pour des expérimentations sur le terrain (champ de betteraves sucrières) le robot sera attelé sur un microtracteur (figure 3).

FIGURE 3 : Représentation du robot Bettybot sur un tracteur

La caméra utilisée est un modèle Baumer VLG40C (figure 4). Il s'agit d'une caméra Gigaethernet grand angle 1" avec une résolution native de 2044x2044 pixels. Un objectif de 6 mm est monté sur cette caméra.

FIGURE 4 : *Caméra Baumer*

Avant d'être utilisé dans les champs de betterave, le robot Bettybot est utilisé dans un hall technologique (figure 5), pour développer et tester les algorithmes de vision artificielle et de contrôle/commande des sept degrés de liberté du robot.

FIGURE 5 : Robot Bettybot dans le hall technologique de Irstea

Environnement logiciel

L'ordinateur du robot Bettybot est configuré avec un système d'exploitation Ubuntu. Au niveau logiciel, les différentes parties du robot (caméra, axe linéaire, bras UR5,...), ainsi que les programmes développés communiquent à travers ROS [1] (Robot Operating System). En ce qui concerne le fonctionnement du robot : les capteurs, ici des caméras, récupèrent les données sur l'environnement extérieur au robot. En condition normale d'utilisation, cet environnement est un champ de betterave. Les images sont envoyées sur un "topic" ROS où elles peuvent être récupérées par les autres programmes présents sur le robot (et utilisant ROS). Ces images sont ensuite traitées par les algorithmes qui seront présentés dans la partie Développement de ce mémoire, lesquels renvoient les informations sur la position des betteraves et de leurs feuilles sur un autre topic. Ces informations sont utilisées par les programmes de commande du robot afin de positionner les caméras de façon à obtenir des images exploitables pour pouvoir faire de la détection de maladie foliaire dessus.
En ce qui concerne le fonctionnement de ROS, tous les programmes, capteurs et moteurs du robot sont représentés par un ***nœud***. Chaque nœud peut souscrire ou publier des informations, appelés ***message***, sur des ***topics*** ou envoyer directement ces messages via des ***services***. Les topics sont un mode de communication asynchrone permettant l'échange entre plusieurs nœuds tandis que les services assurent la transmission de message entre deux nœuds de façon synchrone. L'intérêt d'utiliser ROS est de pouvoir faire fonctionner en continu et en parallèle de nombreux processus, sachant que ce système est conçu pour optimiser la gestion de la concurrence.

Environnement de développement de l'application

Présentation du poste de travail

Le développement et les tests sont réalisés sur un ordinateur portable possédant un système d'exploitation Ubuntu (version 16.04), 8 Gio de mémoire et un processeur Intel® Core™ i7 d'une fréquence de 4 x 2.10GHz.
Les environnements (IDE) de développement utilisés lors de ce projet sont Visual Studio Code (version 1.25) pour la partie en C++ et Pycharm Community (version 2018.2) pour la partie en python. L'utilisation de ces IDE pour développer présente de nombreux avantages comme l'autocomplétion, un débogueur ou encore un outil de contrôle de version.

Gestionnaire de version : git

Voici une définition donnée dans le livre Pro Git [2] :

gestionnaire de version : “Un gestionnaire de version est un système qui enregistre l'évolution d'un fichier ou d'un ensemble de fichiers au cours du temps de manière à ce qu'on puisse rappeler une version antérieure d'un fichier à tout moment.”[1]

Les gestionnaires de versions sont très utilisés en développement informatique pour revenir à une version antérieure d’un programme, ce qui permet de récupérer des fonctionnalités perdues après modification, ou pour fusionner des fichiers sources. Les fichiers constituant la référence sont appelés dépôt et sont généralement stockés sur un serveur accessible aux utilisateurs. Lorsque l’on utilise un gestionnaire de version, on doit d’abord récupérer les informations du dépôt sur lequel on travaille (adresse et contenu). Après avoir modifié des fichiers sur le poste de travail, il faut vérifier qu’ils n’ont pas été modifié sur le dépôt, on doit donc se mettre à jour. Si les fichiers ont subis des modifications depuis la dernière mise à jour, il est nécessaire de fusionner toutes les modifications effectuées avant d’envoyer une nouvelle version sur le serveur.

Lors de mon stage, j’ai utilisé *git* afin de versionner le code source produit. Le choix de *git* a été motivé principalement par le fait que Irstea utilise la plateforme Gitlab, qui (entre autre) gère les dépôts *git* et que le projet était déjà hébergé sur cette plateforme à mon arrivée. Git a été créé par Linus Torvalds et possède la particularité d’être décentralisé. Cela signifie que lorsque l’on récupère le projet sur le poste de travail, on récupère toute les informations du dépôt, notamment tout l’historique des modifications. L’un des avantages est qu’en cas de perte du serveur contenant le dépôt, n’importe quel développeur possède tout le projet, il n’y a donc pas de perte de l’historique du code.

Framework et bibliothèques utilisés

L’utilisation de bibliothèques ou de framework est fondamentale lors de développement d’applications. En effet, ces outils permettent d’éviter de recoder des parties du programme et de gagner beaucoup de temps. De plus les fonctions fournies sont souvent optimisées, donc leur utilisation augmente l'efficacité des programmes développés.

[1] CHACON, scott, et Ben STRAUB. *Pro Git*, New York, APress, 2009, p.9

Opencv [3] est une bibliothèque open source et gratuite pour le traitement d'image. Programmée en C/C++, elle donne accès à de nombreuses fonctions, pour différentes applications de vision artificielle.

Voici les principales fonctions utilisées lors de mon stage :

- ***cvtColor*** est une fonction qui permet de passer d'un espace de couleur à un autre. Avec Opencv, les images sont chargées par défaut dans l'espace de couleur BGR (Blue, Green, Red). On peut utiliser ***cvtColor*** pour passer d'une image couleur à une image en niveaux de gris, mais aussi pour passer en HSV (Hue, Saturation, Value). On peut alors appliquer des filtres dans ces différents espaces (figure 6).

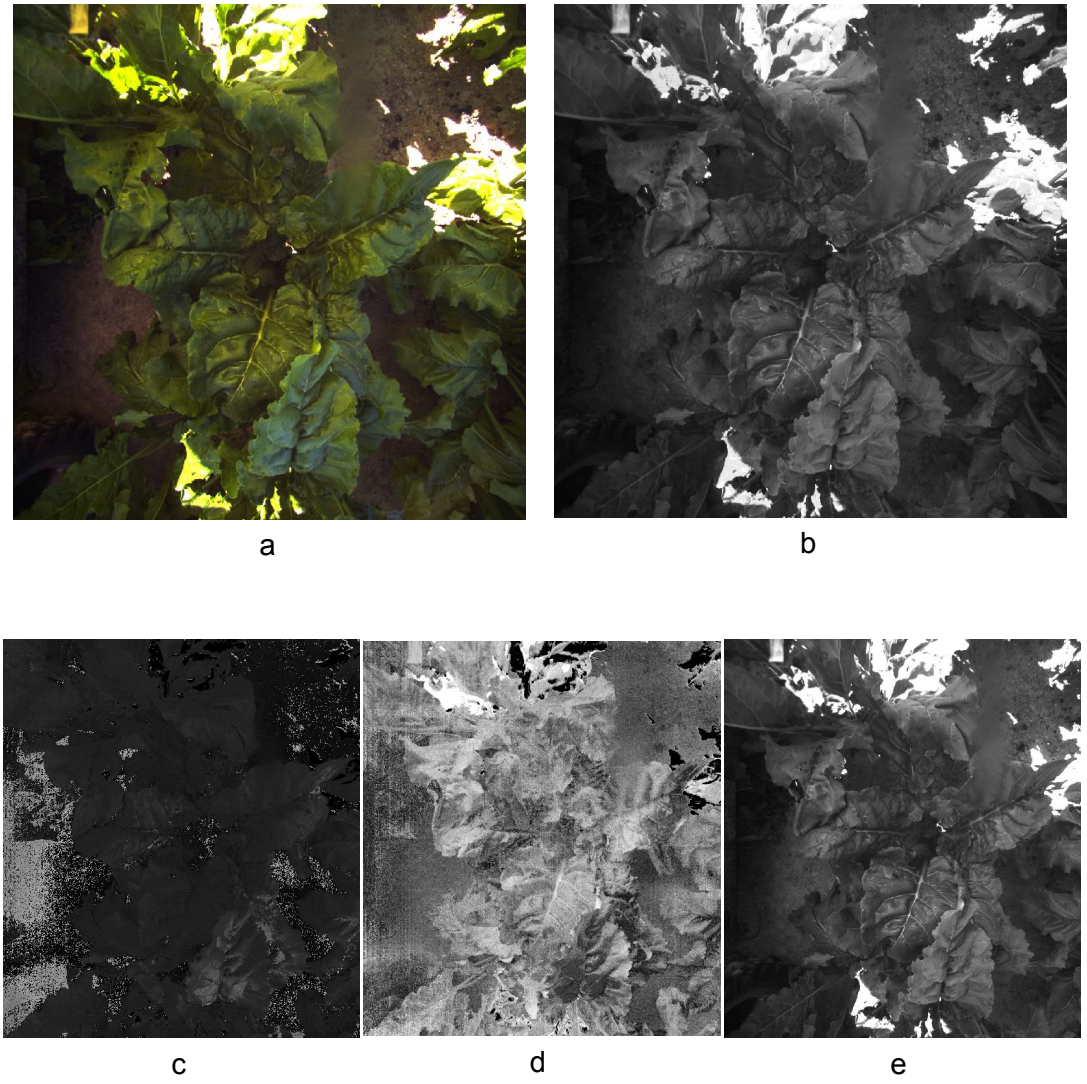


FIGURE 6 : *Application de **cvtColor**. Image originale (a), image en niveaux de gris (b), canal H de l'image (c), canal S de l'image (d) et canal V de l'image (e).*

- Les fonctions morphologiques de base (***dilate*** et ***erode****)* (figure 7) combinées sont utilisées pour supprimer du bruit ou mettre en valeur des éléments de l'image en fonction de leur taille. Ces fonctions peuvent déformer l'image et entraîner une perte d'information.

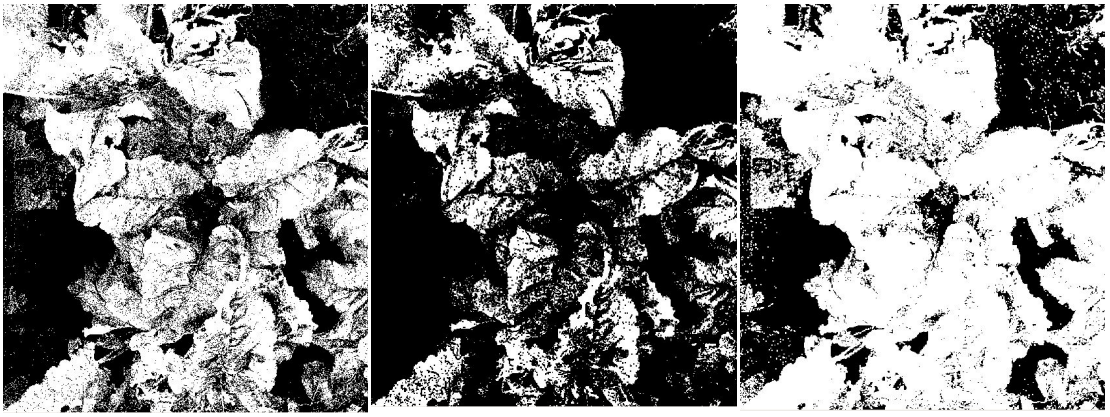


FIGURE 7 : *Exemple d'utilisation de* ***erode*** *(milieu) et* ***dilate*** *(droite) sur une image de betterave binaire (gauche)*

- La fonction ***threshold*** permet de seuiller une image en niveau de gris. Le résultat du seuillage dépend de la valeur du seuil et du type de seuillage. Les principales utilisations de la fonction ***threshold*** sont la binarisation classique (les pixels dont le niveau de gris est inférieur au seuil sont mis à zéro, les autres sont mis à 255), la binarisation inverse (les pixels dont le niveau de gris est supérieur au seuil sont mis à zéro, les autres sont mis à 255) et le seuillage à zéro (les pixels dont le niveau de gris est inférieur au seuil sont mis à zéro, les autres conservent leur valeur) (figure 8).

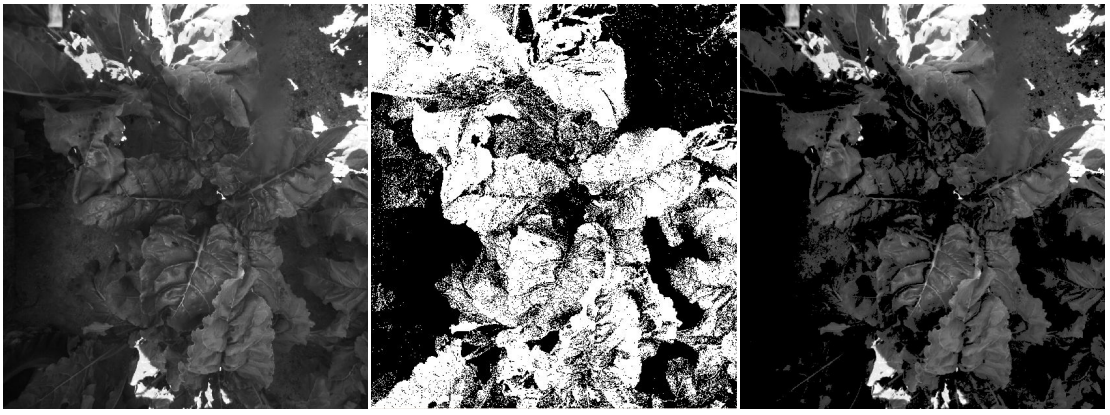


FIGURE 8 : *Utilisation de* ***threshold*** *pour binariser (milieu) ou seuiller à zéro (droite) une image en niveaux de gris (gauche)*

- Les fonctions de détection de contour, parmis lesquelles le filtre de ***Canny*** *(figure 9)*, le filtre de ***Sobel*** et la fonction ***Laplacian*** sont efficaces pour identifier des éléments sur une image. Leur utilisation est très sensible aux variations de luminosité et au floutage de l'image. Elles s'utilisent généralement après les fonctions ***GaussianBlur*** ou ***blur*** qui modifient le floutage de l'image et réduisent le bruit.

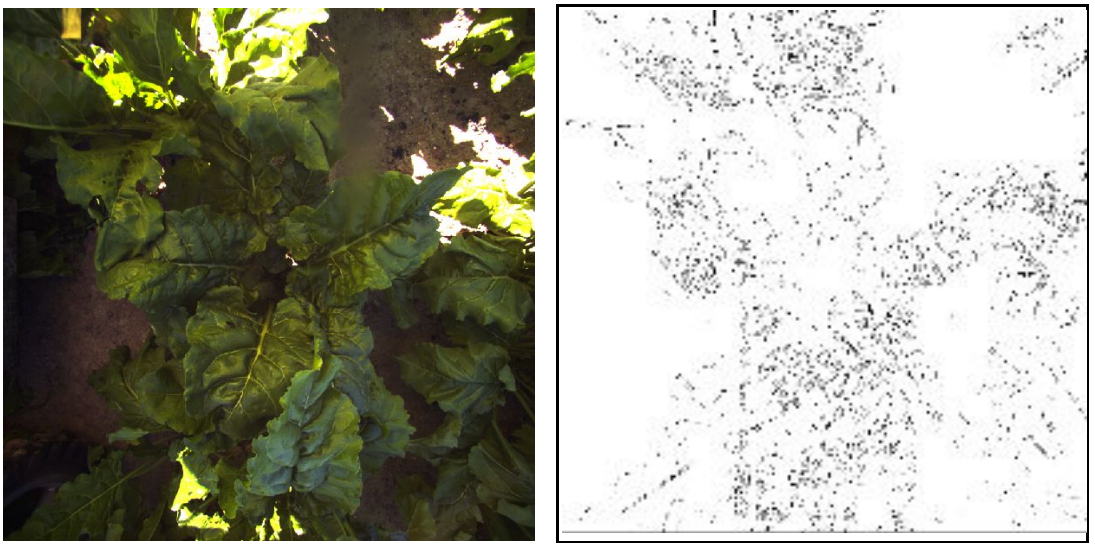


FIGURE 9 : *Utilisation du filtre de **Canny** sur une image de betterave sucrière*

- La fonction ***connectedComponents*** labellise des objets séparés sur une image binaire (les objets connectés entre eux ont le même label). Cette fonction prend en entrée une image binaire et renvoie l'image labellisée. Elle permet également d'obtenir des statistiques (surface, centre, rectangle autour de l'objet) sur les labels. (figure 10)

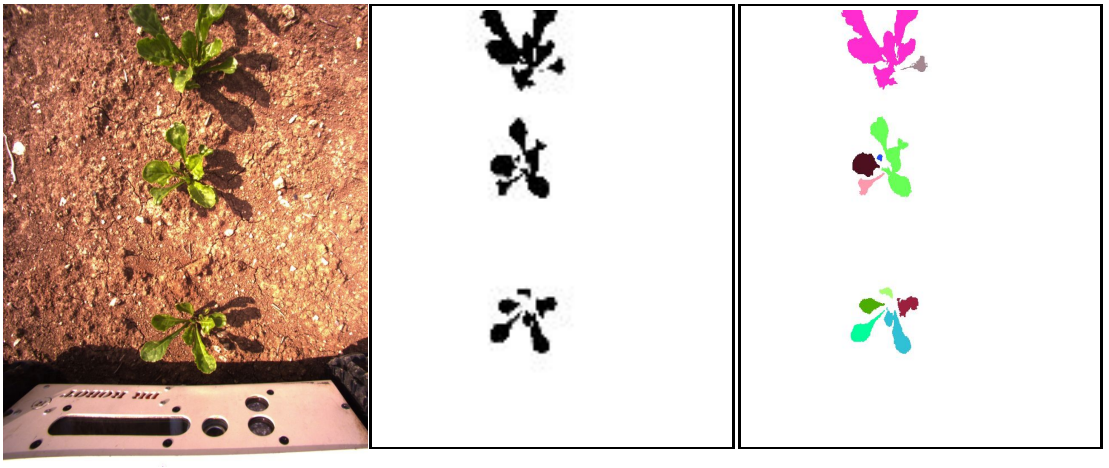


FIGURE 10 : *Utilisation de **connectedComponents** pour obtenir une image labellisée (droite) à partir de l'image binaire (milieu) d'une photo de betterave (gauche)*

- La fonction ***HoughLines*** detecte les lignes sur une image. Elle prend en compte quatre paramètres en entrée : l'image en niveau de gris, le seuillage sur le nombre de croisement dans l'espace de hough, la taille minimum des lignes et l'espace maximum entre deux points d'une même ligne. La fonction renvoie un tableau de lignes. (figure 11)

FIGURE 11 : *Utilisation de **HoughLines** pour trouver des lignes sur une plante*

- La fonction ***distanceTransform*** calcule la distance pour chaque pixel de l'image source entre ce pixel et le pixel de valeur nulle le plus proche. Cette fonction sert à récupérer les centres de forme sur une image et permet de faire de la squelettisation. (figure 12)

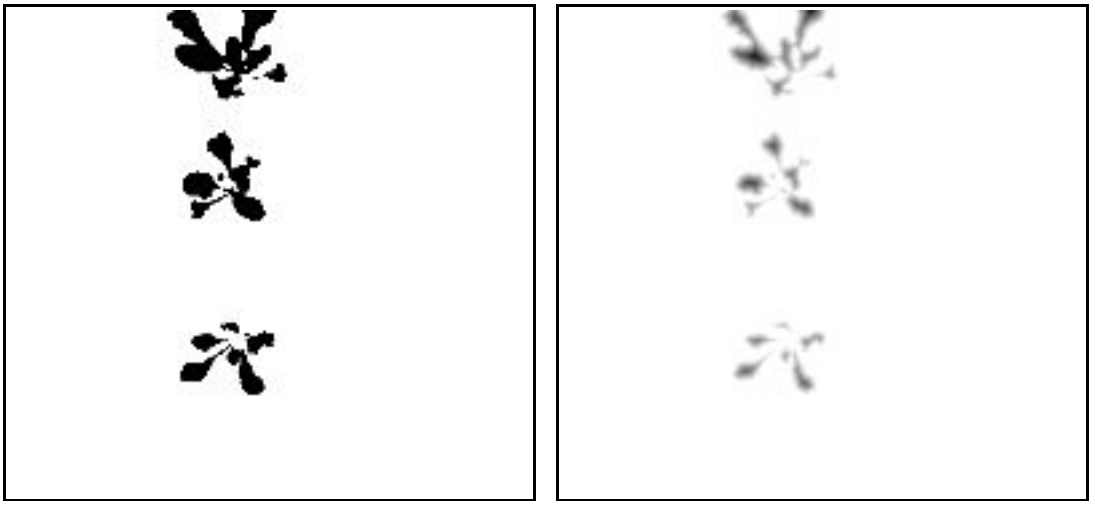


FIGURE 12: *Utilisation de **distanceTransform** pour obtenir la carte de distance (droite) à partir d'une image binaire (gauche)*

- La segmentation par ***Watershed*** est une transformation qui sépare des zones sur une image. Cette fonction prend en paramètre des marqueurs (un tableau de point), et des lignes de séparation (une image binaire). Les marqueurs correspondent aux centres des zones (il y a donc autant de zone que de marqueur). La fonction étend les zones jusqu'à ce qu'elles se rencontrent où rencontrent une ligne de séparation (figure 13).

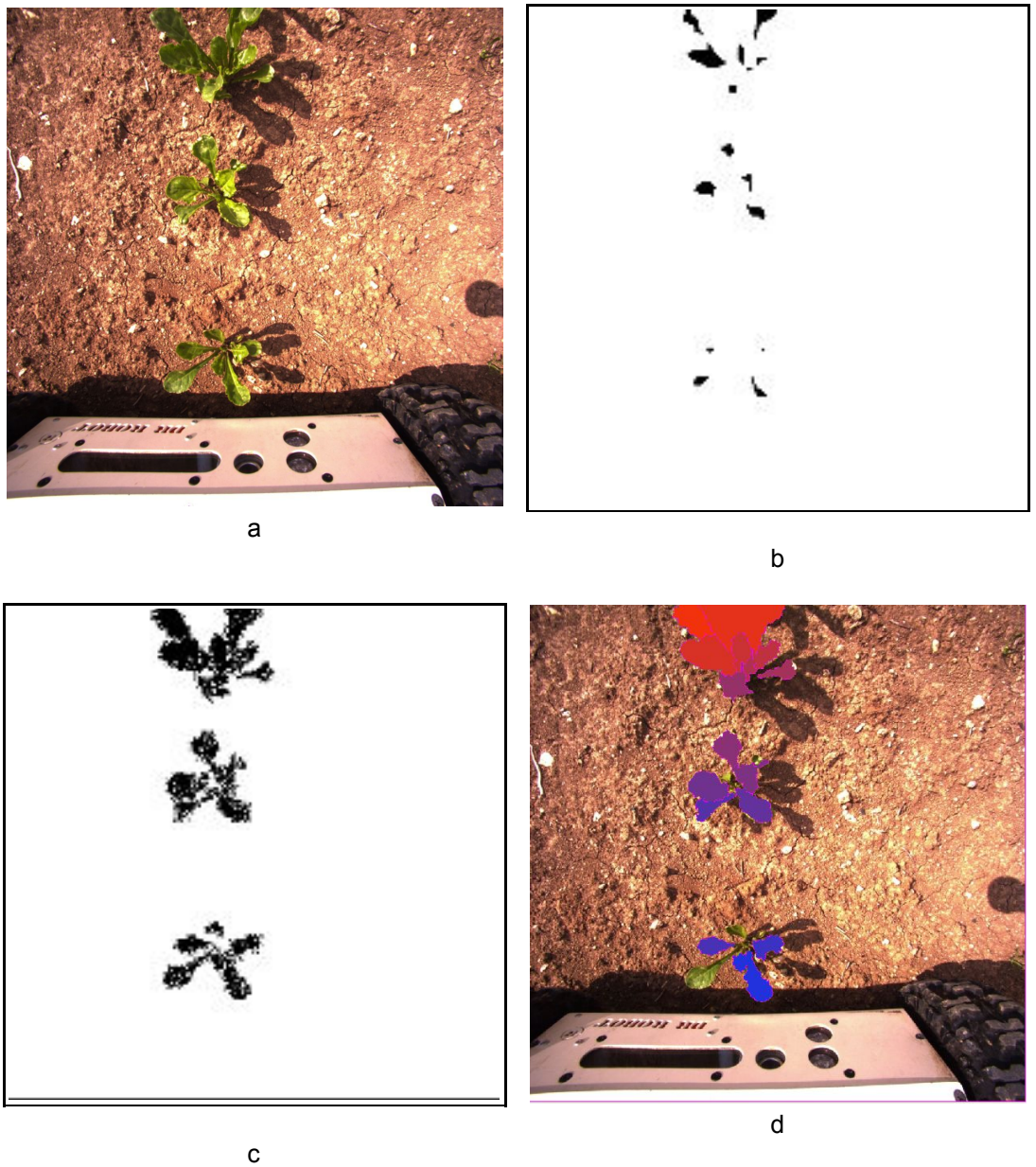


FIGURE 13 : *Application de l'algorithme de **Watershed** sur une image de betterave (a) : création de marqueurs (b), définition des limites (c) et résultat avec les zones segmentées (d)*

En plus de ces fonctions, Opencv permet de créer des ***trackbars*** et de faire varier les paramètres des fonctions tout en visualisant le résultat de ces variations.

Le framework ROS (Robot Operating System) regroupe un ensemble d'outils et de bibliothèques dont le but est de faciliter la programmation et l'intégration de systèmes robotisés. En plus d'être utilisé sur le robot (voir partie Environnement logiciel), il est également utilisé pour la compilation et le test des algorithmes développés. La compilation avec ROS assure la compatibilité à l'exécution entre les programmes de traitement d'image et le reste des processus présent sur bettybot lors de l'intégration au robot. Pour tester, corriger et améliorer les programmes développés l'outil "Rosbag" est utilisé. Cet outil enregistre les données des topics du robot en fonctionnement dans un fichier (données camera, données des articulations du robot). Ce fichier peut ensuite être relu sur le pc de travail ce qui simule le robot en fonctionnement.

La commande permettant l'enregistrement de données est *rosbag record*. Par défaut, tous les topics sont enregistrés mais on peut donner en paramètre le nom de topics spécifiques.
La commande permettant de parcourir les données enregistrées est *rosbag play*. Les paramètres optionnels que l'on peut ajouter sont la vitesse de lecture ou le point de départ de lecture.

TensorFlow [4] est une bibliothèque open source développée à l'origine par Google pour le machine learning et deep learning. Elle permet d'utiliser et d'intégrer des modèles dans une application ou un programme. Le deep learning est beaucoup utilisé en traitement d'image pour faire de la reconnaissance d'objets.

Cette bibliothèque permet également de faire du "transfer learning". Il s'agit d'une technique permettant d'éviter d'entraîner un modèle depuis le début, ce qui nécessite plusieurs millions d'images labellisées, et des centaines d'heures de calcul avec des ordinateurs puissants (possédant des cartes graphiques). La méthode consiste à récupérer un modèle déjà entraîné pour la tâche que l'on souhaite réaliser, comme par exemple de la reconnaissance d'objet dans une image, et de remplacer uniquement les dernières couches de ce modèle.

Développement

Méthodologie

La première partie du travail a consisté à réaliser une recherche bibliographique sur les algorithmes existants. Lors de cette recherche, quatre méthodes de segmentation de feuilles ont été trouvées [5]. Ces algorithmes concernent tous la segmentation individuelle de feuille sur des plants de tabac, ce qui est proche du sujet de stage.

La méthode de développement utilisé lors de ce projet suit le principe du modèle en spirale [6] (figure 14) : Après avoir défini les besoins, un premier prototype de programme est développé. On teste ensuite ce prototype et son efficacité est évaluée (voir partie Résultat). On réfléchit à “comment améliorer les algorithmes”, de nouvelles idées sont alors trouvées. On implémente l’algorithme modifié, qui est ensuite testé puis évalué. On peut ainsi voir l’impact des modifications sur le résultat et choisir, en fonction de cet impact, les modifications gardées pour la suite. Si l’on estime que des améliorations sont encore possibles, on retourne à l’étape “recherche de nouvelle idée”. Le but est d’obtenir un programme le plus fiable et robuste possible.

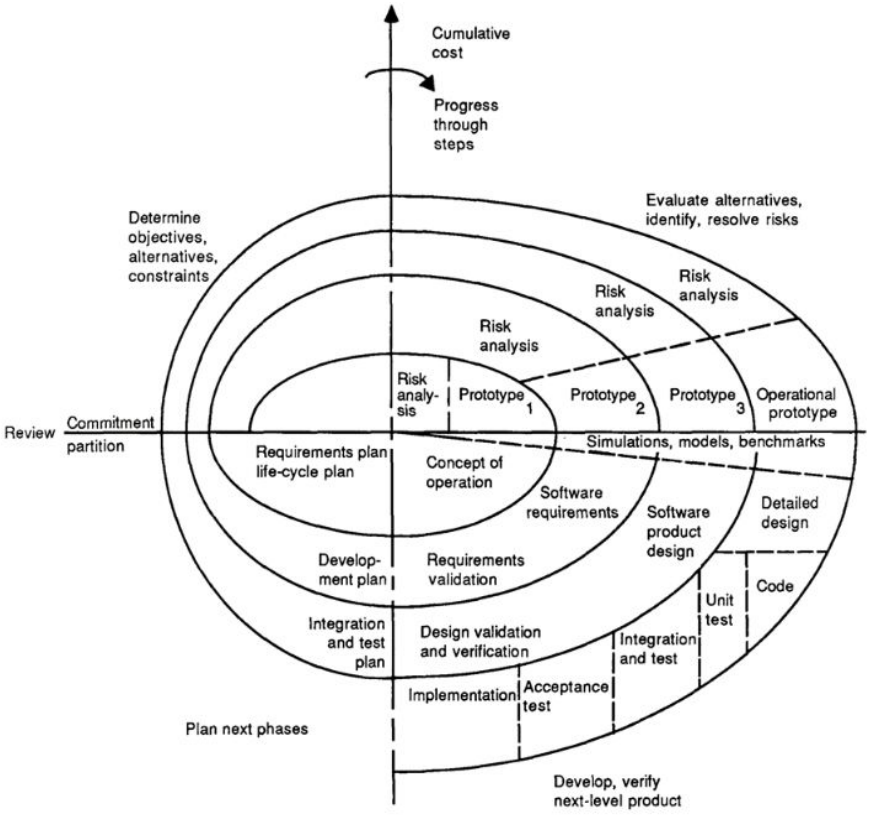


FIGURE 14 : *Modèle en spirale pour un processus logiciel*

On peut séparer le programme de traitement en deux étapes : la détection et localisation des plants de betterave, et la segmentation des feuilles. Ces deux étapes ont été développées parallèlement pendant mon stage car certaines fonctions utilisées sont communes aux deux types de traitement. L'objectif est de d'abord trouver le centre de la plante pour que la caméra se place au-dessus, puisse plus facilement détecter les feuilles, puis effectue plusieurs acquisitions autour des feuilles.

Présentation des Algorithmes

Première approche

Un premier essai est de tenter de reconstituer les betteraves et d'en obtenir un modèle 3D pour récupérer des informations dessus, notamment la position 3D des feuilles, ce qui permet ensuite de positionner la caméra au plus proche de cette feuille avec les angles qui nous intéressent. Lors de la recherche bibliographique, une publication [7] montrait des plants de betterave reconstitués. Pour cela, les auteurs ont utilisé un logiciel qui crée un modèle 3D à partir d'images 2D : micmac. Ce logiciel présente deux avantages : premièrement il est licence libre et open source. Et deuxièmement, il s'utilise en ligne de commande, ce qui permet de l'intégrer facilement dans un programme ou dans ROS.

Les résultats obtenus avec micmac sont satisfaisants (figure 10), cependant, le temps de calcul pour obtenir les modèles est beaucoup trop grand (plusieurs minutes par modèle) et ne correspond pas à l'objectif d'une analyse "temps réel".

On a ensuite estimé qu'une analyse sur des images 2D serait suffisante. En effet, une fois la feuille détectée sur l'image, il suffit d'avancer la caméra dans sa direction jusqu'à ce que la feuille occupe une grande partie de l'image, ce qui nous permet de nous passer de la dimension camera-feuille.

FIGURE 15 : *Passage d'une image (gauche) à un modèle 3D (droite) avec micmac*

Détection de centre de betterave

Avant de trouver les feuilles de betterave, une première étape consiste à trouver la présence de plantes et de localiser leur position. Pour cela, une première partie du programme consiste à repérer les centres de betterave sur une image.

Dans un premier temps, cette partie du programme consistait uniquement en un filtre sur l'excédent de vert[8] suivi d'une détection d'éléments avec le calcul du centre de gravité de ceux-ci. L'excédent de vert est calculé pour chaque pixel à partir de la formule suivante : ExG = 2g -r - b. Un seuil est ensuite défini (dans notre cas, la valeur 10 a été choisi de façon empirique) et seuls les pixels ayant un ExG supérieur à cette valeur sont gardés, les autres sont mis à zéro (figure 16).

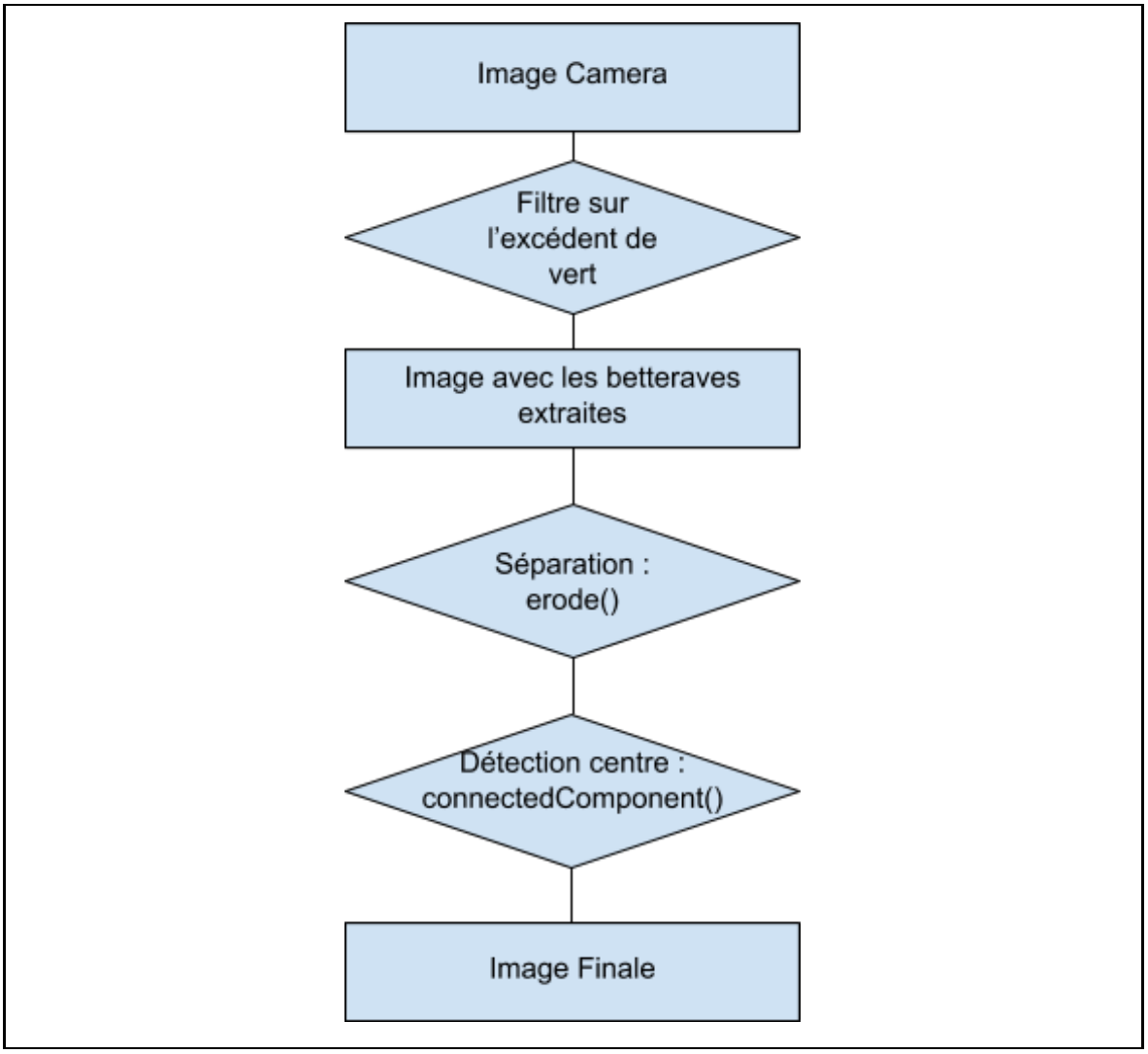


FIGURE 16 : *Présentation de la première version de l'algorithme de détection de centre de betterave*

Cependant, la méthode précédente ne fonctionne pas lors d'un fort recouvrement des betteraves. La détection de centre de betterave à l'aide des tiges et des nervures centrales des feuilles permet d'améliorer le programme dans ce cas de figure, qui est très courant lors d'un stade avancé de croissance des betteraves. En effet, on observe que les tiges et nervures centrales des feuilles d'une même betterave ont une forte tendance à être dirigées vers le centre de cette betterave. L'idée de l'algorithme est donc de trouver ces tiges et nervures et de calculer leur droite. Le centre des betteraves correspond alors au croisement de ces droites. Les tiges et nervures des betteraves sont d'une couleur plus claire que le reste des feuilles. On peut donc les discriminer sur ce point à l'aide d'un filtre sur la couleur (figure 17).

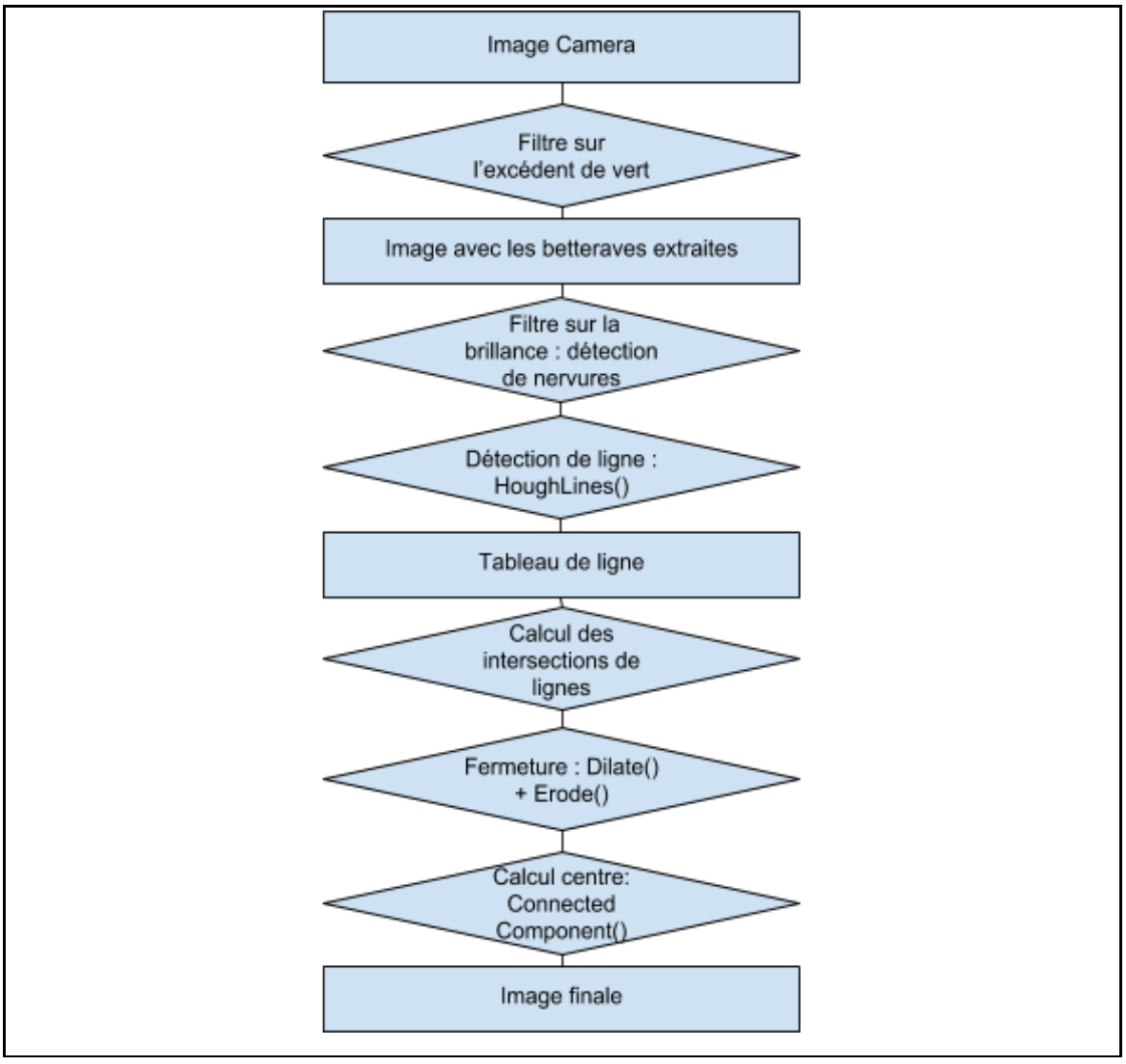


FIGURE 17 : *Présentation de la deuxième version de l'algorithme de détection de centre de betterave*

Ce programme a ensuite subi quelques modifications pour être efficace dans plus de cas:

- Remplacement du filtre par excédent de vert par un filtre HSV, qui est plus robuste aux variations de luminosité
- Remplacement de la détection des tiges et nervures avec filtre sur la couleur par un filtre de Canny qui se couple mieux à la détection de ligne avec la fonction HoughLines()

Avec l'algorithme ci-dessus, dans des cas de fort recouvrement, où des feuilles se croisent, il peut arriver que l'on trouve des faux positifs. C'est à dire que des "centres virtuels" sont présents sur l'image. On entend par "centre virtuel" un point sur l'image vers lequel converge plusieurs tiges et/ou nervures centrales de feuille, mais qui ne correspond pas à un centre réel. Pour remédier à ce problème, une première solution est de regarder les droites qui forment les centres et de supprimer les centres majoritairement formés par des droites utilisées par d'autre centres. Cette solution fonctionne dans les cas ou au moins trois centres sont détectés. Pour les cas où moins de trois centres sont trouvés, un couplage avec un réseau de neurones peut être utilisé. Le problème du *deep learning* est qu'il est assez lent comparé au reste du programme (de l'ordre de 1s par analyse), c'est pour cela qu'il ne peut pas être utilisé sur toute les zones de l'image. La sous-image testée est créée à partir d'un rectangle autour des points trouvés, on vérifie ensuite que tensor flow trouve bien un centre, sinon le point n'est pas considéré comme tel. Pour la création du modèle, j'ai utilisé le programme de réentraînement de *TensorFlow* avec pour origine le réseau de neurones Inception V3[9] entraîné sur la base de donnée d'image ImageNet. Trois classes constituent le nouveau modèle : "Centre de betterave", "Feuille de betterave" et "Fond". Les images utilisées lors de l'entraînement du modèle proviennent des expérimentations réalisées et de photos fournis par l'ITB. J'ai manuellement labellisé ces images : 33 images pour la classe "Fond", 84 images pour la classe "Centre de betterave" et finalement 99 images pour la classe "Feuille de betterave" (Annexe A). Pour tester ce modèle, j'ai développé un mini programme en python qui prend une sous partie d'une image donnée en entrée et donne le résultat du modèle en sortie (figure 18 et 19).

FIGURE 18 : *Exemple de résultat utilisant le modèle créé et les fonctions tensorFlow sur différentes parties d'une image de betterave*

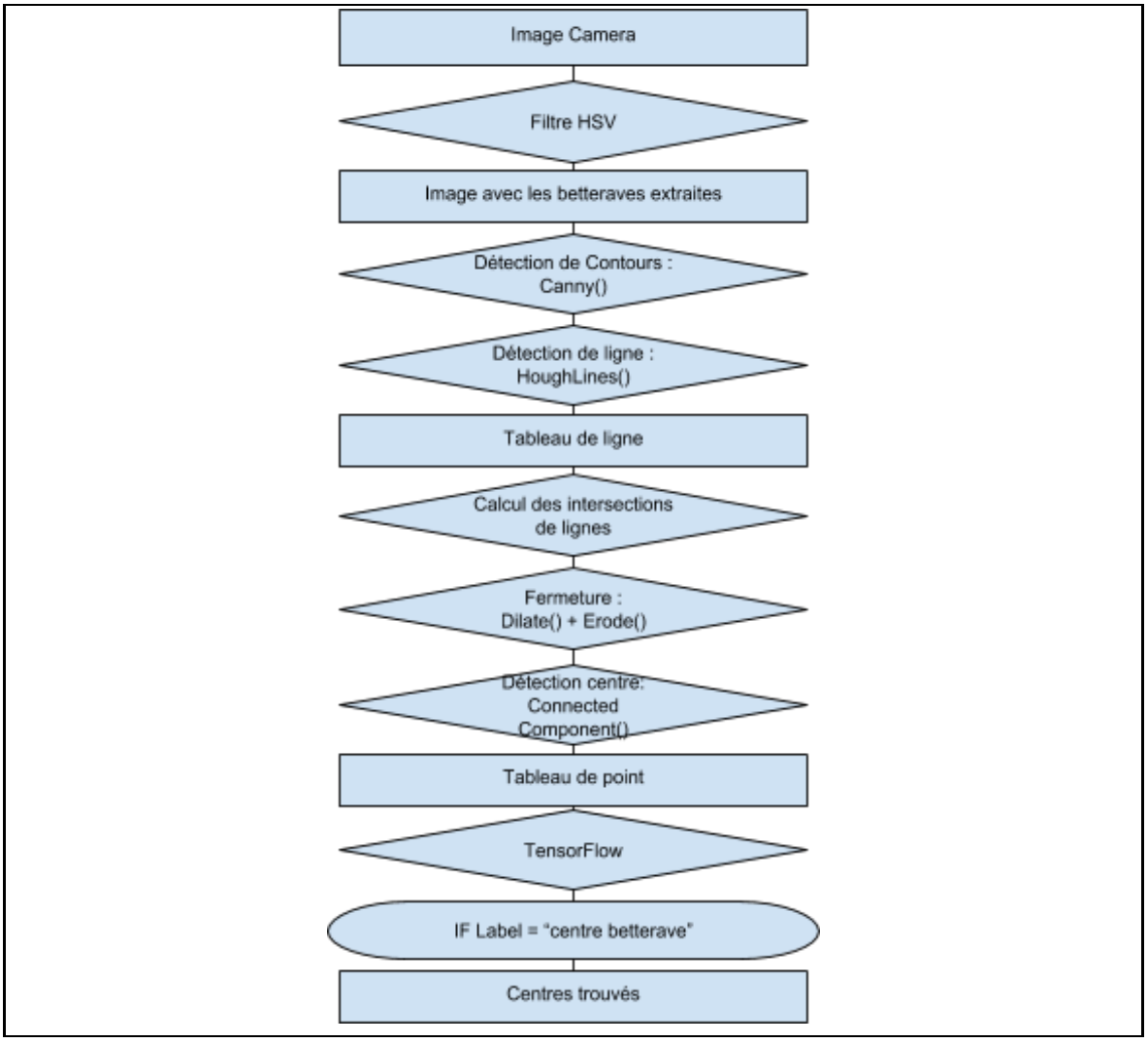


FIGURE 19 : *Présentation de la version finale de l'algorithme de détection de centre de betterave*

Segmentation de feuilles de betterave

La première méthode développée pour récupérer les feuilles sur l'image se basait sur le filtre par excédent de vert pour isoler la plante du sol, ainsi que sur des opérations morphologiques simples : la dilatation et l'érosion.
Pour réellement séparer les feuilles, un filtre de canny était ensuite utilisé. Cependant, les contours trouvés par le filtre Canny ne sont pas fermés : on ne peut donc pas utiliser la fonction "connectedComponents" sur le résultat. Pour fermer ces contours, une première solution trouvé fut de les approximer par une droite puis d'étendre les droites trouvées (figure 20).

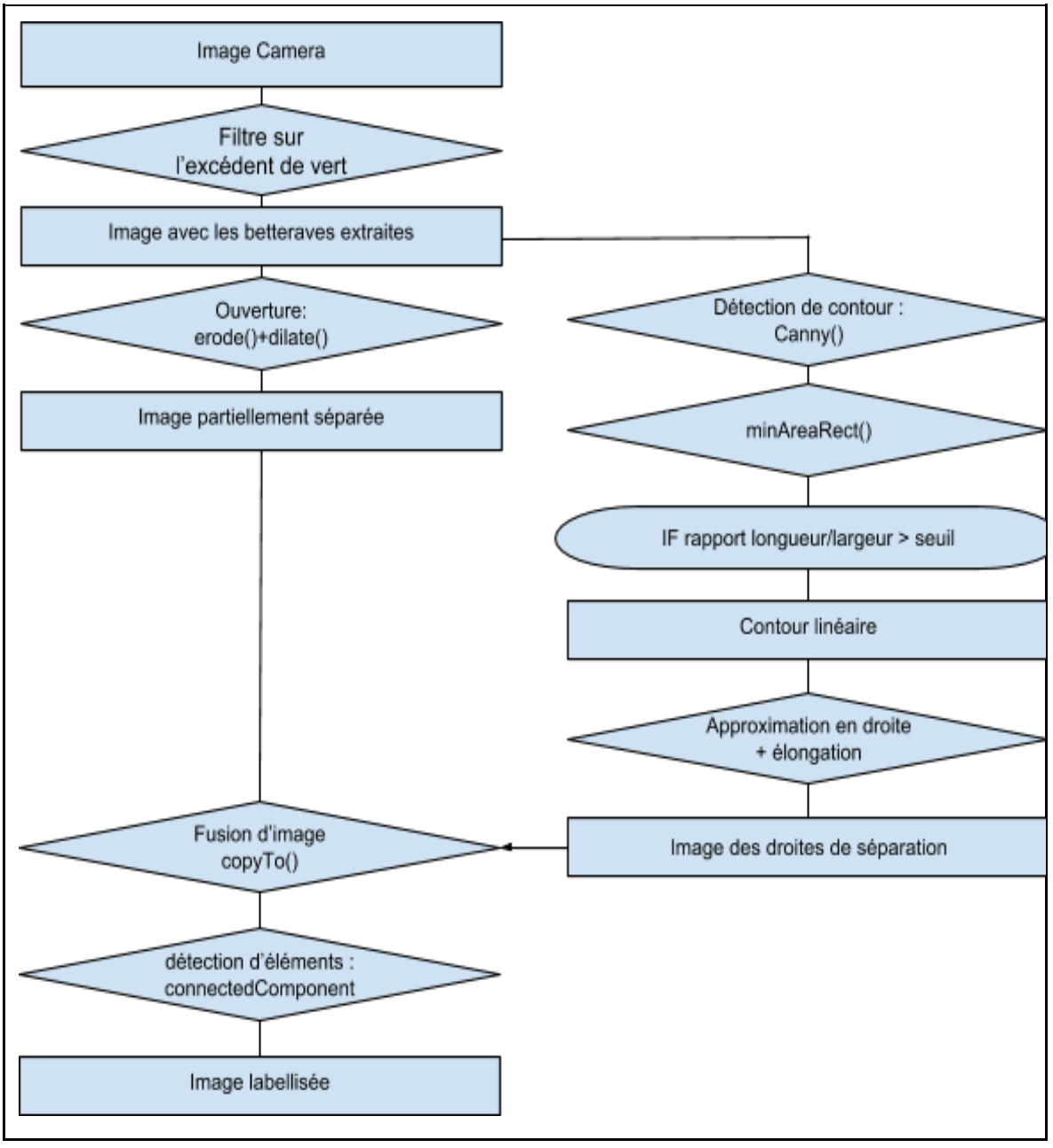


FIGURE 20 : *Présentation de la première version de l'algorithme de segmentation de feuilles*

La solution précédente ne donnant pas de bons résultats dans de nombreux cas à cause de l'approximation des contours par des droites (en effet les contours étendus ne correspondent pas forcément à des séparations de feuille), j'ai tenté d'implémenter l'une des méthode de segmentation trouvée dans la bibliographie : la segmentation utilisant l'algorithme SLIC superpixels [10]. L'avantage de cette méthode est qu'elle n'est pas supervisée, contrairement aux autres méthode trouvées, il n'y a donc pas besoin d'image labellisée pour qu'elle fonctionne (figure 21).

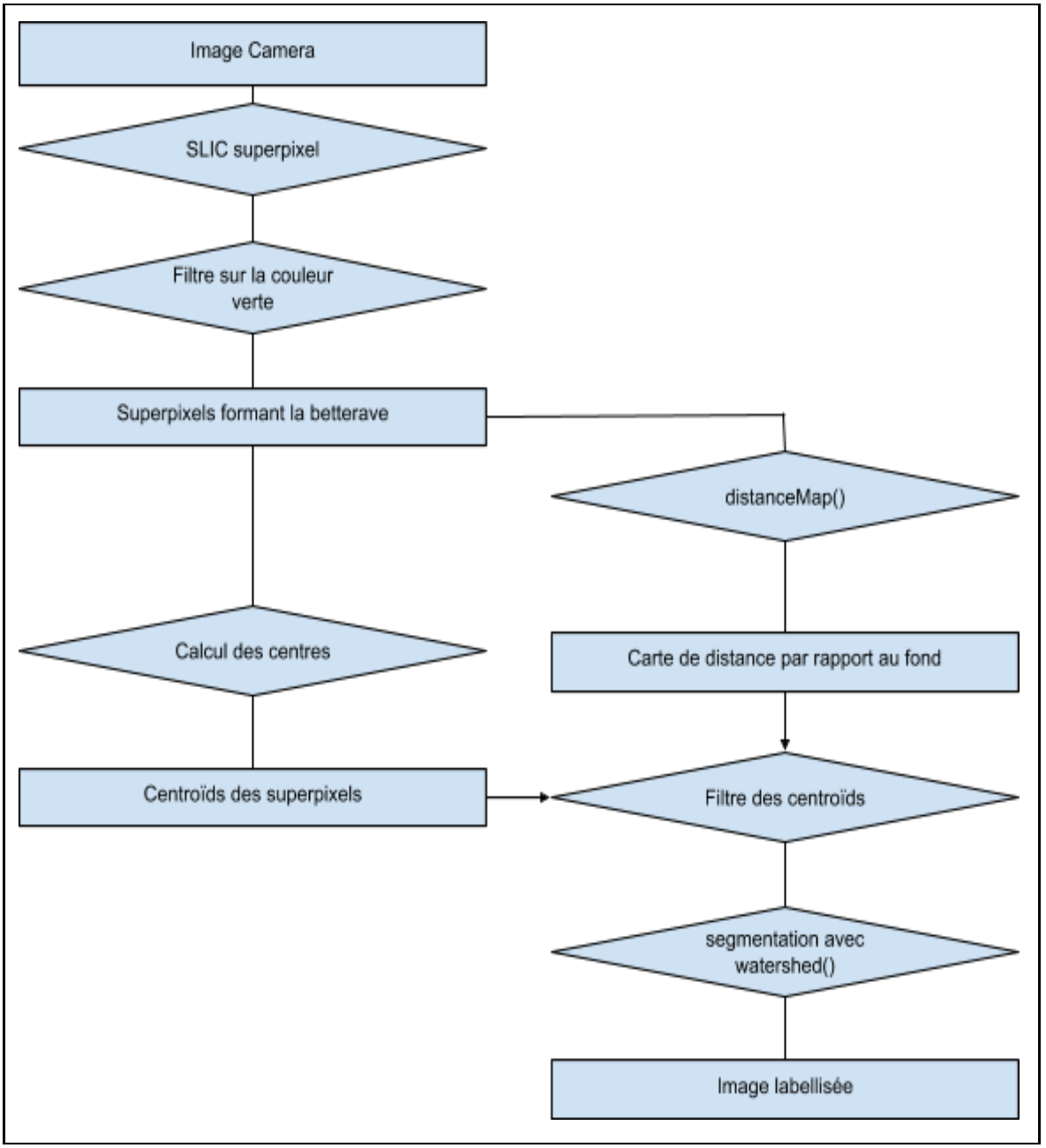


FIGURE 21 : *Présentation de la deuxième version de l'algorithme de segmentation de feuilles*

L'utilisation des superpixels n'apporte pas réellement d'avantages car les centroïdes trouvés sont très proches de ceux trouvés en utilisant uniquement le maximum de la carte de distance. L'utilisation d'une carte de distance et de la fonction *watershed* donnent de bons résultats, même lors de fort recouvrement de plante. Cependant, appliquer le même seuillage de carte de distance uniformément sur toute l'image pose un problème : avec un seuillage trop important, les feuilles les plus petites ne sont pas détectées et avec un seuillage trop faible les feuilles trop proches ne sont pas séparées. Pour pallier à ce problème, j'ai codé une fonction utilisant la carte de distance de façon récursive sur les éléments trouvés dans l'image (Annexe B) (figure 22).

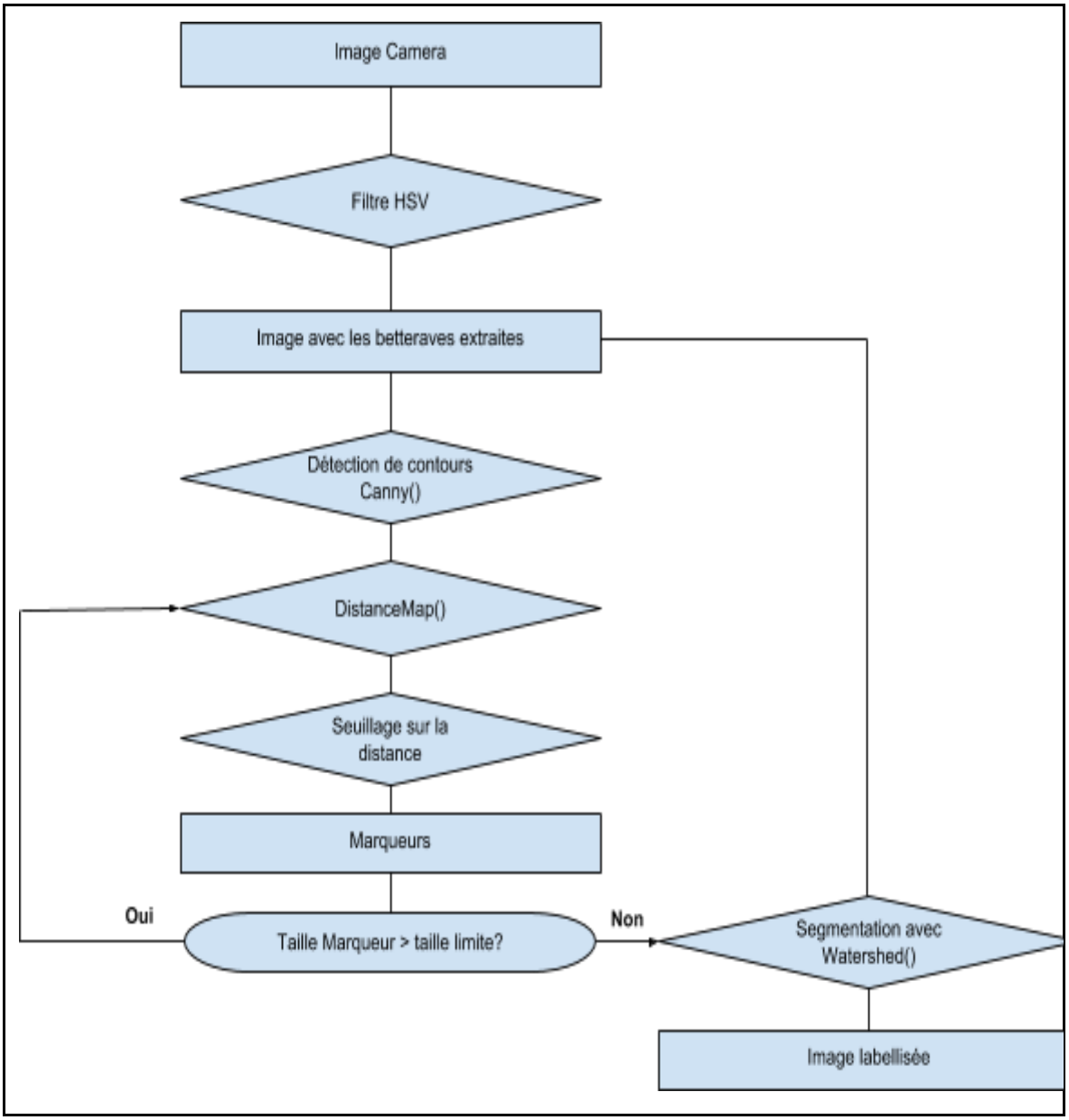


FIGURE 22 : Présentation de la version finale de l'algorithme de segmentation de feuilles

Expérimentation

Pour réaliser des acquisitions de données et tester les algorithmes, plusieurs expérimentations ont été effectuées :

La première sur le site de recherche et d'expérimentation de Montoldre. Les acquisitions ont été faites avec le robot RobuFast (figure 23) sur un champ de betteraves sucrières d'une surface de 50 x10 mètres. Les données ont été récupérées à différents moments de la journée pour permettre d'avoir des images avec des intensités lumineuses variées. RobuFast est un robot mobile d'une masse de 400 kg et pouvant aller jusqu'à 10 m/s, ce qui est rapide pour un robot expérimental de ce type. Il est équipé de la même caméra que le robot Bettybot.

Pour cette expérimentation, nous avons dans un premier temps appris la trajectoire à réaliser au robot (un premier passage lent), ce qui lui a permis d'enregistrer les coordonnées GPS des points par lesquels passer. Le robot parcourt ensuite en autonomie les lignes dans le champ de betterave à une vitesse de 1 m/s.

FIGURE 23 : *Robot RobuFAST lors d'acquisition d'image dans le champ de betterave à Montoldre*

Environ 4000 images ont été récupérées avec la caméra Baumer, qui ont ensuite été utilisées par les algorithmes de traitement d'images. La figure 24 présente des exemples d'images obtenues par la caméra.

FIGURE 24 : *Exemple d'images acquises dans le champ de betterave à Montoldre*

Les autres expérimentations ont été faites à l'Irstea, dans le hall technologique du site des Cézeaux à Aubière. Pour ces expérimentations, des betteraves sucrières ont été amenées d'une part du champ expérimental de Montoldre lors des expérimentations présentées au paragraphe précédent, et d'autre part par l'ITB. Les betteraves ont été mises en pot pour pouvoir les déplacer facilement. Les manipulations ont été effectuées dans un premier temps à l'intérieur du hall technologique puis à l'extérieur (figure 25) pour obtenir des images dans des conditions de luminosité les plus variées possibles.

FIGURE 25 : *Expérimentation avec le robot Bettybot*

Les expérimentations dans le hall technologique ont permis de récupérer environ 2700 images (figure 26).

FIGURE 26 : *Exemple d'images acquises avec le robot Bettybot*

Présentation et analyse des résultats

Détection de centre de betterave

L'algorithme de détection de centres permet de trouver des éléments sur des images de betterave avec fort recouvrement. Les meilleurs résultats sont obtenus avec les paramètres présentés dans le tableau en figure 27. Les valeurs des paramètres ont été modifiés grâce aux trackbars d'OpenCV jusqu'à ce qu'elles donnent de bons résultats sur un échantillon significatif d'images.

HoughLines	Seuillage	38
	Taille min. ligne	33
	Espace max.	6
Filtre HSV	H (couleur)	30 - 110
	S (saturation)	40 - 255
	V (brillance)	0 - 255
Nombre de dilatation		23
Nombre d'érosion		19
Elongation des lignes		870 pixels

FIGURE 27 : *Tableau des paramètres utilisés pour la détection de centre*

L'utilisation d'un filtre pour séparer les plantes du sol est nécessaire car le sol est très irrégulier. Les détections de contours sur les images ont tendance à trouver beaucoup d'éléments sur le sol, ce qui masque et dégrade les informations que l'on cherche à extraire (figure 28).

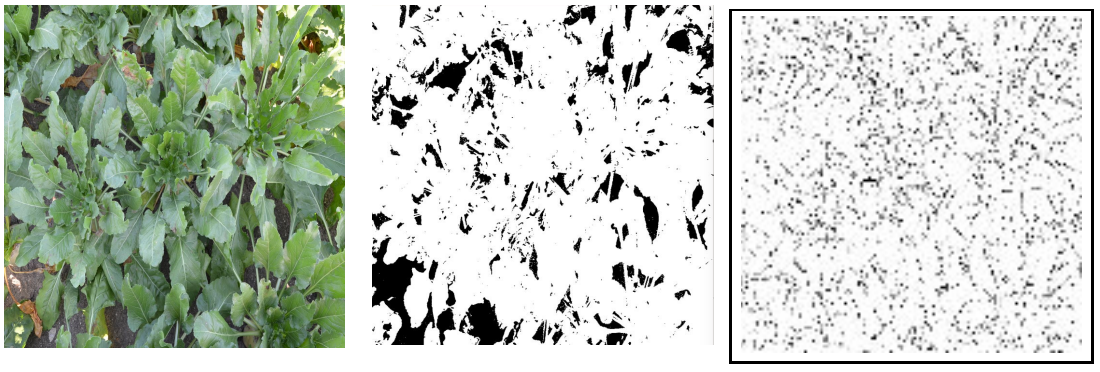


FIGURE 28 : *Application d'un filtre HSV (milieu) et d'un filtre de* ***Canny****(droite) sur une image de betteraves sucrière (gauche)*

Comme on peut le voir sur la figure 29, la fonction ***HoughLines*** permet de trouver de nombreuses lignes. Les lignes trouvées ne correspondent pas toujours aux nervures ou aux tiges, et une partie des tiges et nervures ne sont pas détectées (sur cette image, environ 40%).

FIGURE 29 : *Lignes trouvées par la fonction **HoughLines** (gauche) et lignes étendues (droite)*

Ces lignes sont ensuite étendues et leurs intersections sont calculées. La détection de centre avec les fonctions ***dilate*** et ***erode*** permet de regrouper les points d'intersections (figure 30 et 31) et les plus gros groupements correspondent aux centres.

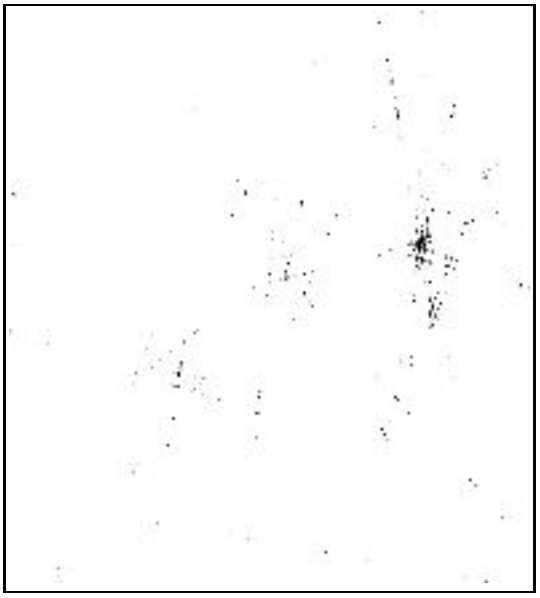


FIGURE 30 : *Points d'intersection trouvés sur l'image*

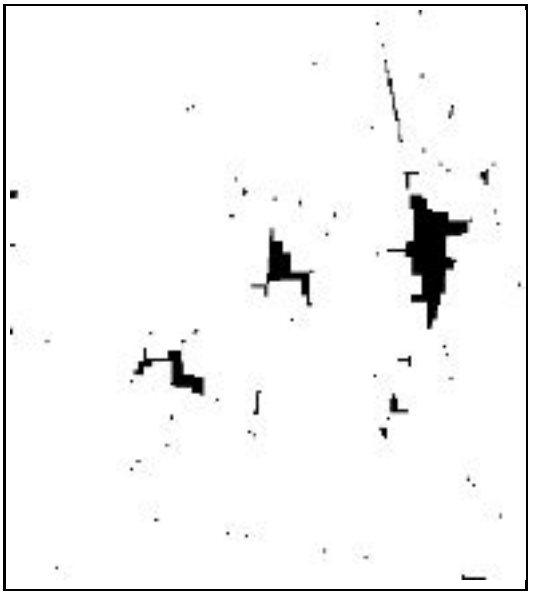
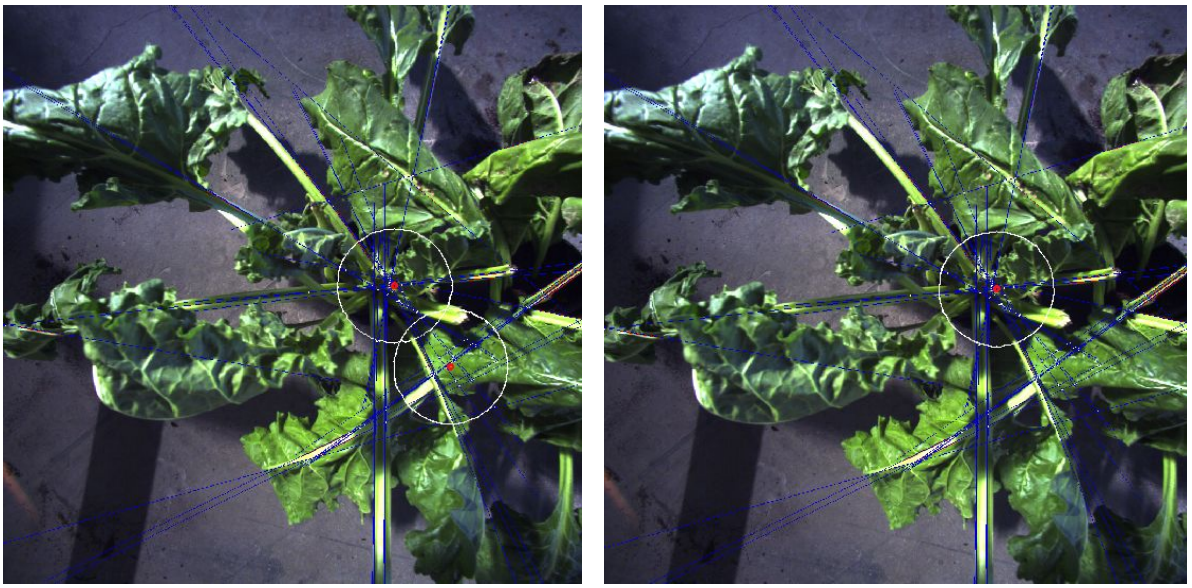


FIGURE 31 : *Application des fonctions **dilate**/**erode** à ces points pour trouver les groupement de points*

Sur la figure 32, on peut voir un résultat final de l'algorithme de détection de centres sur des betteraves. Le léger décalage des centres trouvés par rapport aux centres réels provient de deux facteurs sur cette image : les tiges et nervures utilisées ne convergent pas toutes vers le centre (surtout le point 1), et il y a une déformation due à l'utilisation des fonctions ***dilate*** et ***erode***.

FIGURE 32 : *Trois centres trouvés sur des betteraves sucrières*

Dans le cas où plusieurs centres sont retrouvés sur une image, TensorFlow est utilisé pour discriminer les centres et trouver le bon centre pour chaque plante.
A titre d'exemple, voici ci dessous un résultat de TensorFlow. La figure 33 présente une première image (a) contenant deux centres de betterave. L'application du deep learning a permis de détecter le bon centre (b).



a b

FIGURE 33 : (*a) Plante avec deux centres (b) Centre trouvé avec TensorFlow*

Le principal problème rencontré dans cette partie est la variation de luminosité (surtout en milieu extérieur). En effet, les fonctions de détection de contours et de détection de lignes y sont très sensibles et même une petite variation de lumière (indétectable à l'œil nu) peut entraîner une variation des résultats trouvés. L'un des autre paramètres impactant négativement la détection de contour est la distorsion de la caméra, qui est importante aux bords de l'image. Un autre problème rencontré est celui des "centres virtuels", mais ce problème a été très majoritairement résolu par l'utilisation du réseau de neurones et de TensorFlow. Le temps d'exécution de la fonction de détection de centre sans TensorFlow est rapide : environ 100ms. Pour chaque centre trouvé, l'analyse réalisée par les fonctions de TensorFlow, pour déterminer le bon centre, ajoute une seconde au temps d'exécution.

Segmentation de feuilles de betterave

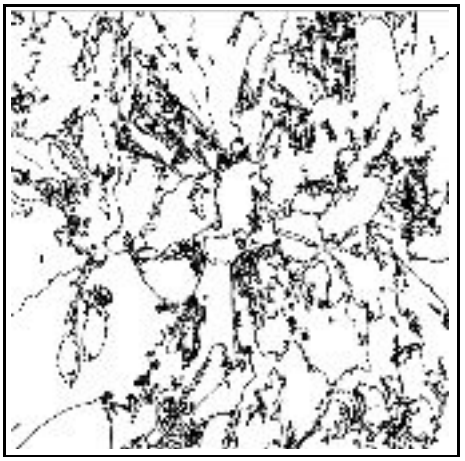
Les valeurs utilisées pour les paramètres de la segmentation de feuilles sont présentés dans le tableau en figure 34. Ces valeurs ont été obtenues par variations grâce aux trackbars sur un échantillon significatif d'image.

Filtre HSV	H (couleur)	30 - 110
	S (saturation)	40 - 255
	V (brillance)	0 - 255
Canny	Seuillage bas	41
	Seuillage haut	120
Carte de distance	Seuillage	40
	Taille minimale	5000 pixels
Floutage	Taille du noyau	7*7 pixels

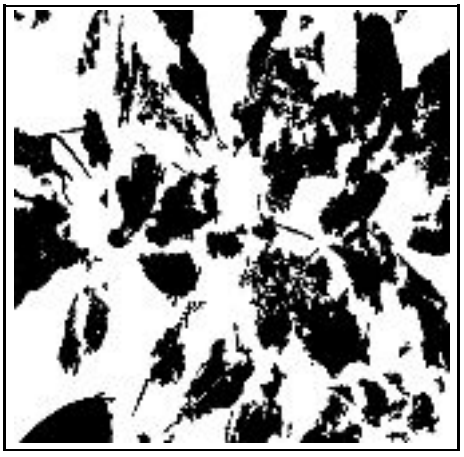
FIGURE 34 : *Paramètres utilisés pour les fonctions de séparation de feuilles*

Dans cette partie, le filtre de ***Canny*** couplé au filtre HSV, appliqué sur l'image en figure 35 a, créé une première séparation entre les feuilles (figure 35 b et c). Cependant, on peut constater que cette séparation n'est pas suffisante, c'est pour cela que l'on utilise la fonction ***distanceTransform*** (figure 35 d), qui donne une meilleure séparation. Le résultat de cette fonction est ensuite donné en paramètre à la fonction ***Watershed***, qui permet de récupérer la forme originale des feuilles (figure 35 e).

a



b



c

d

e

FIGURE 35 : *Résultat de différentes étapes de la séparation de feuille : image originale (a), détection de contours avec filtre de* ***Canny*** *(b), filtre HSV (c), et seuillage sur la carte de distance(d) et résultat final (e)*

Sur la figure 36, sur l'image à gauche, on peut voir que le programme détecte 13 feuilles, tandis que l'on peut en compter 16 en réalité. En ce qui concerne le résultat de l'image de droite, il est trop compliqué de séparer toute les feuilles. Cependant, on peut constater que les feuilles sont regroupées en paquets (de 2 à 4 feuilles) dans les zones où il y a le plus de recouvrement et sont bien séparées en cas de recouvrement faible. Quand les feuilles sont bien séparées, l'information concernant leur forme est en général bien récupérée également.

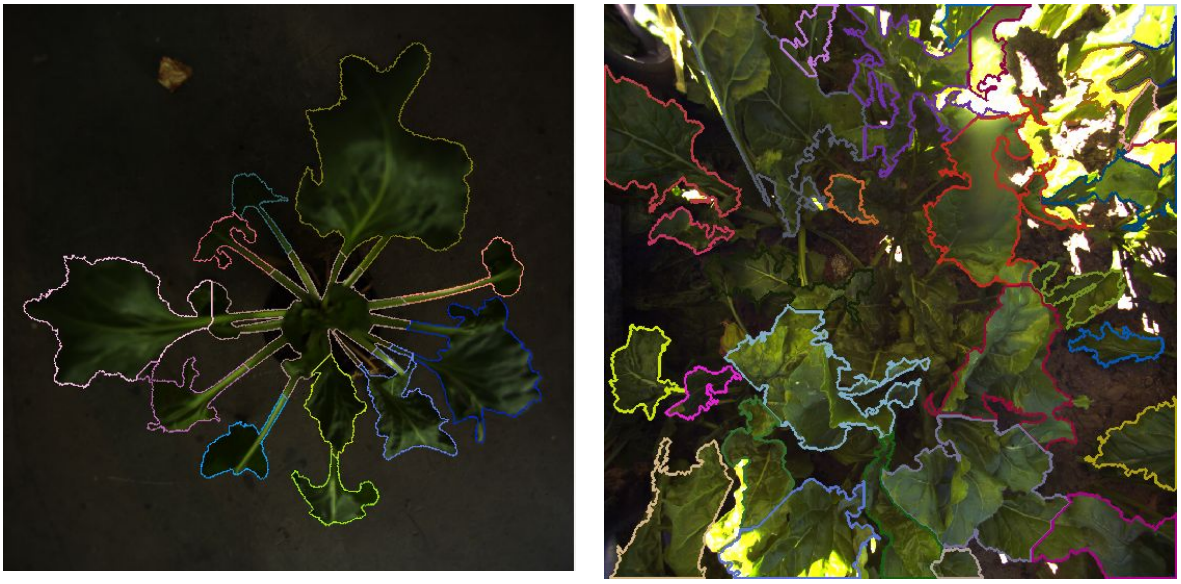


FIGURE 36 : *Résultat de séparation de feuilles sur un cas "simple" avec une plante (gauche) et un cas plus complexe dans un champ de betterave (droite)*

L'une des principales difficultées rencontrées dans la séparation des feuilles est, comme pour la détection de plante, la variation de luminosité, qui gêne les algorithmes de détection de contours (Canny). Un autre problème important est le recouvrement des feuilles : si des feuilles se chevauchent, il est plus compliqué de les séparer. Les fonctions de segmentation (comme ***Watershed***), mais aussi la fonction récursive prennent plus de temps à l'exécution (la fonction de séparation de feuilles s'exécute en environ 900 ms) que les transformations morphologiques ou seuillage.

Bilan

Ci-dessous les résultats des algorithmes les plus avancés testés sur 118 plants de betterave à partir des fichiers d'images enregistrés à Montoldre dans des conditions complexes : acquisition en champs en extérieur, avec fort recouvrement, présence de mauvaises herbes et variations de luminosité importantes. Les lignes de betteraves de Montoldre n'avaient pas les même caractéristiques (espacement entre les plantes plus faible, plus de mauvaises herbes) que les cultures de betteraves sur lesquelles seront réalisées les expérimentations avec le robot Bettybot. Ces conditions étaient particulièrement défavorables à la qualité des résultats obtenus avec les algorithmes développés.

Pour évaluer la validité d'un résultat de l'algorithme de détection de centre, une estimation est donnée à vue d'œil sur la position du centre réel. On compare ensuite approximativement la distance entre les points donnés par le programme avec le point réel estimé. On considère que si cette distance est inférieure à 50 pixels, alors le point obtenu par l'algorithme de traitement d'image est considéré comme valide (figure 37).

Résultat de l'algorithme final sans utilisation de TensorFlow.	Centre trouvé valide	30
	Centre trouvé invalide	18
	Plusieurs centres trouvés dont un valide	37
	Plusieurs centres trouvés non valides	18
	Aucun centre trouvé	15
Résultat de l'algorithme final avec utilisation de TensorFlow	Centre trouvé valide	51
	Centre trouvé invalide	10
	Plusieurs centres trouvés dont un valide	10
	Plusieurs centres trouvés non valides	5
	Aucun centre trouvé	42

FIGURE 37 : *Tableau de résultats évalués pour l'algorithme de détection de centres*

On peut constater que l'utilisation des fonctions de tensorflow augmente les centres trouvés valides, mais augmente également les chances qu'aucun centre ne soit trouvé.

Pour évaluer la séparation des feuilles, on commence par regarder autour d'un centre de betterave (estimé à vue d'œil). Si les feuilles sont séparées de façon "égale" autour de ce centre et que leur forme détecté par l'algorithme est proche de leur forme réelle, alors on considère que la séparation est valide. Si des feuilles collées entre elles sont assemblées en petit paquet (2/3 feuilles), on considère tout de même que la séparation peut être valide (figure 38).

Nombre de plante avec des feuilles bien séparées	Nombre de plante avec des feuilles mal séparées
70	48

FIGURE 38 : *Tableau de résultats évalués pour l'algorithme de séparation de feuilles*

Conclusion

Les algorithmes de traitement d'image développés pendant ce stage ont permis de détecter des centres de plant de betterave et de séparer leurs feuilles. Le travail a porté sur un échantillon d'environ 6700 images de plants de betterave. Ces deux algorithmes fonctionnent très bien à faible recouvrement et avec peu de variation de luminosité (conditions de laboratoire), mais ont plus de difficulté dans des champs de betteraves. Les données obtenues par traitement d'image sont les centres de gravité des plants de betterave et des données géométrique sur leurs feuilles (axe d'inertie et points aux extrémités). Ces données sont envoyées via ROS aux programmes de contrôle commande du robot Bettybot (6 degrés de liberté du bras UR5 et axe linéaire), pour positionner automatiquement la caméra située à l'extrémité du robot aux positions et orientations désirés par rapport aux différentes feuilles de betteraves détectées.

Les améliorations envisagées pour la suite sont de mieux prendre en compte les problèmes liés à la luminosité de l'environnement en agissant sur les paramètres d'acquisition d'image de la caméra : durée d'exposition automatique ou utilisation d'un flash synchronisé avec la caméra pour être insensible aux variations de luminosité. Il est également possible d'améliorer la partie de traitement d'image correspondant à l'extraction des plants de betterave par rapport au sol, en remplaçant le filtre HSV par une discrimination réalisée en utilisant la technique de classification SVM (machines à vecteurs de support, en anglais *support vector machine*). De plus, un réglage automatique des paramètres permettant de s'adapter au niveau de recouvrement des plantes, pourrait donner de meilleurs résultats sur certains champs de betteraves. D'autre part, il serait intéressant de créer un réseau de neurones plus robuste avec TensorFlow en utilisant plus d'images labellisées pour l'entraînement, pour la détection de centre de betterave, mais aussi pour la séparation de feuille.

D'un point de vue personnel, ce stage de recherche dans le domaine de l'agriculture de précision a été très enrichissant. Il m'a permis de développer en C++ et python, de travailler avec ROS et de découvrir le deep learning avec l'utilisation de la bibliothèque TensorFlow. J'ai également progressé dans le domaine du traitement d'image avec l'utilisation de nombreuses fonctions de la bibliothèque OpenCV. Enfin, ce stage m'a donné l'opportunité de travailler avec une plateforme mobile et un bras manipulateur UR5.

Bibliographie

[1] Open Source Robotics Foundation. (Page consulté le 1 mai 2018). *ROS*, http://www.ros.org

[2] CHACON, scott, et Ben STRAUB. *Pro Git*, New York, APress, 2009, 288 p.

[3] Opencv Team. (Page consulté le 1 mai 2018). *OpenCV library*, https://opencv.org

[4] Google Brain Team. (Page consulté le 1 mai 2018). *TensorFlow*, https://www.tensorflow.org/

[5] H. Scharr, M. Minervini, A. P. French, C. Klukas, D. M.Kramer, X. Liu, I. Luengo, J-M.Pape, G. Polder, D. Vukadinovic, X. Yin, and S. A. Tsaftaris. Leaf segmentation in plant phenotyping: a collation study. *Machine Vision and Applications*, 2016, pp. 585–606.

[6] Boehm B, "A Spiral Model of Software Development and Enhancement", *IEEE Computer*, 1988, pp. 61–72.

[7] Sylvain Jay, Gille Rabatel, Xavier Hadoux, Daniel Moura and Nathalie Gorretta. In-field crop row phenotyping from 3D modeling performed using Structure from Motion. *Comput. Electron. Agric.*, 2015, pp. 70–77.

[8] M. Guijarro, G. Pajares, I. Riomoros, P.J. Herrera, X.P. Burgos-Artizzu, A. Ribeiro. “Automatic segmentation of relevant textures in agricultural images”, *Comput. Electron. Agric.*, 2011, pp. 75-83

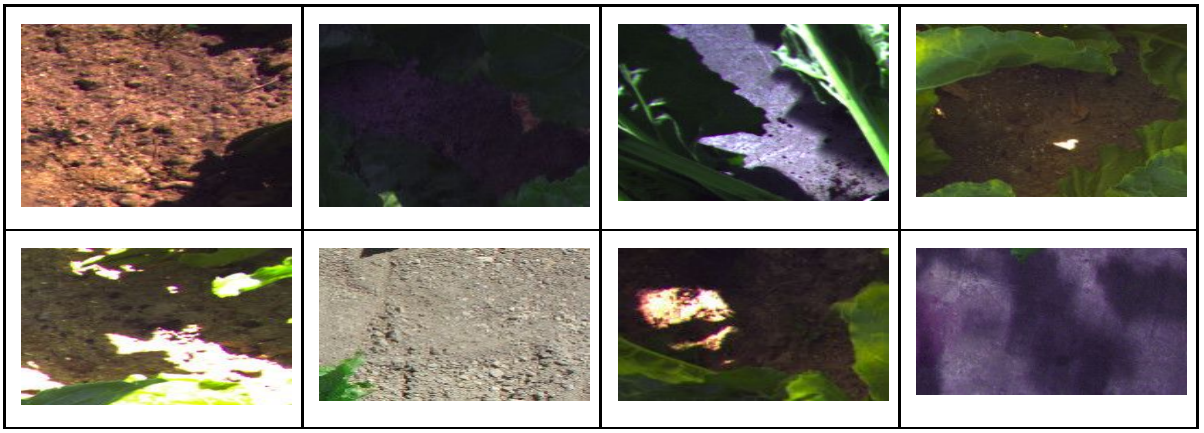
[9] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna: "Rethinking the Inception Architecture for Computer Vision", 2015.

[10] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Susstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2012, pp. 2274–2282.

Annexes

Annexe A -Exemples d'images utilisées avec TensorFlow

Images de fond



Images de centre de betterave

Images de feuilles de betterave

Annexe B - Exemple de code C++ : Utilisation du réseau de neurones (graph) avec TensorFlow

```
Tensor ReadTensorFromMat(Mat image, int depth, int32 height, int32 width, float mean, float std)
{
  tensorflow::Tensor input_tensor(tensorflow::DT_FLOAT,
      tensorflow::TensorShape({1, height, width, depth}));
  float *p = input_tensor.flat<float>().data();
  cv::Mat no_image(height, width, CV_32FC3, p);
  // l'image donnée est défini sur 255 valeur (1 octet) en BGR mais on doit la convertir
  // en float et RGB pour etre compatible avec le type tensor. Les valeurs de pixels en float vont de 0.0 à 1.0
  cvtColor(image, image, CV_BGR2RGB);
  image.convertTo(no_image, CV_32FC3, 1.0/255.0);

  return input_tensor;
}

string im_annalyse(Mat mat_image)
{
  string graph = "./output_graph.pb";
  string labels ="./output_labels.txt";
  int32 input_width = 299;
  int32 input_height = 299;
  float input_mean = 0;
  float input_std = 255;
  string input_layer = "Placeholder";
  string output_layer = "final_result";
  bool self_test = false;
  string root_dir = "";

    std::unique_ptr<tensorflow::Session> session;
  string graph_path = tensorflow::io::JoinPath(root_dir, graph);
  Status load_graph_status = LoadGraph(graph_path, &session);
  if (!load_graph_status.ok()) {
    LOG(ERROR) << load_graph_status;
  }

    std::vector<Tensor> resized_tensors;
  const Tensor& resized_tensor = ReadTensorFromMat(mat_image,3, input_height, input_width, input_mean,
                                   input_std);

    std::vector<Tensor> outputs;
  Status run_status = session->Run({{input_layer, resized_tensor}},
                                   {output_layer}, {}, &outputs);
  if (!run_status.ok()) {
    LOG(ERROR) << "Running model failed: " << run_status;
  }
  vector<string> result = PrintTopLabels(outputs, labels);

  return result[0];
}
```

Annexe C - Exemple de code C++ : Fonction récursive pour séparer les feuilles

```cpp
//Fonction pour separer les formes de facon recursive en fonction de leur taille
void recursivDistanceMap(Mat &originalDark, Mat LabelImage_,int nLabels, Mat stats, double dist_thresh, int lim_size)
{
  // Condition minimal pour ne pas avoir une trop grande recursivité
  if (lim_size<300)
  {
    lim_size = 300;
  }
  if(dist_thresh<0.2)
  {
    dist_thresh= 0.2;
  }

  int ind = 1;
  //Pour chaque forme sur l'image
  while(ind<nLabels)
  {
    int area = stats.at<int>(ind, CC_STAT_AREA);//calcul de sa taille

    //isolement de la forme
    Mat dark = LabelImage_.clone();
    dark.convertTo(dark, CV_32F);
    threshold(dark,dark,ind-1,0,THRESH_TOZERO);
    threshold(dark,dark,ind,0,THRESH_TOZERO_INV);
    threshold(dark,dark,ind-1,255,THRESH_BINARY);

    if (area>lim_size) // Si elle est trop grande, on la separe avec la methode distancetransform+threshold
    {
      Mat Dark2;
      dark.convertTo(dark, CV_8U);
      distanceTransform(dark,Dark2,CV_DIST_L2,3);
      normalize(Dark2, Dark2, 0, 1., NORM_MINMAX);
      threshold(Dark2,Dark2,dist_thresh,1.0,THRESH_BINARY);
      Dark2.convertTo(Dark2, CV_8U);
      //On relance la fonction sur les formes obtenues pour reséparer ou ajouter à l'image final suivant la taille
      Mat LabelImage_, stats, centroids;
      int nLabels = connectedComponentsWithStats(Dark2, LabelImage_, stats, centroids, 4, CV_32S);
      recursivDistanceMap(originalDark, LabelImage_, nLabels, stats, dist_thresh, lim_size);
    }
    else
    {
      originalDark += dark;//Ajout de la forme non modifiée à l'image "originaldark"
    }
    ind++;
  }
}
```

Annexe D - Glossaire

Terme	Définition
Phénotypage	Le phénotypage végétal est l'identification des caractéristiques observables d'un lot de plantes et son suivi lors de sa croissance.
Segmentation	Opération de traitement d'images qui a pour but de rassembler des pixels entre eux suivant des critères pré-définis.
Seuillage	Méthode de segmentation d'image. Séparation de deux classes à partir d'une valeur de seuil.
Framework	Ensemble d'outils, de bibliothèques et de conventions dont le but est de faciliter la création de logiciel.
Labellisation	De l'anglais label (étiquette). Fait d'associer des éléments à un nom pour les identifier.