**Détection de feuille**

**Principe de base :**

A partir de la photo qui est prise, nous allons rééchantillonner l’image afin de diminuer le nombre de pixels présents sur l’image (pour que la détection se fasse plus rapidement). A partir de cette image rééchantillonnée, nous allons appliquer un seuillage afin de différencier la feuille du support. Ce seuillage va permettre d’obtenir une première base de pixels qui sont constitué les rebords de la feuille. Afin d’affiner le résultat, nous allons nous servir de l’agorithme de RANSAC afin d’obtenir un résultat plus robuste. Nous nous trouvons alors avec 4 droites, donc les intersections vont former les 4 points caractéristiques nécessaires à la localisation de la feuille.

**Seuillage :**

La détection de contour consiste à repérer les grandes variations de couleur entre des pixels proches. Le but étant de faire apparaitre les contours des objets présents sur une image. En effet, Ce qui différencie un objet d'un fond est leurs quantifications. Exemple, une feuille blanche sur fond rouge. Afin de détecter les contours de l'objet nous allons observer la différence entre les pixels voisins de l'image. Si la différence est grande alors nous avons détecté un contour.

**Pseudo-code du seuillage :**

Entrées :

image rééchantillonnée

Sorties :

Image seuillée

Algorithme :

**Pour** chaque couple (a,b) de pixel de l’image

differenceRouge := valeurAbsolue(a.rouge - b.rouge)

differenceVert := valeurAbsolue(a.vert - b.vert)  
 differenceBleu := valeurAbsolue(a.bleu - b.bleu)

differenceCouleur(a,b) := differenceRouge + differenceVert + differenceBleu

**si** differenceCouleur(a,b) < 100

a et b appartiennent à deux regions de pixel differents.

**Fonctionnement de l’algorithme de RANSAC :**

Deux points sont choisis et une droite est tracée. A partir de cette droite, on regarde les points à côté de cette droite, si la distance est suffisamment petite, on l’ajoute à l’ensemble de points. Quand tous les points ont été examiné, si le nombre de points dans l’ensemble de points est suffisamment élevé, on réajuste la droite et on mesure l’erreur, c’est-à-dire la somme des distance entre les points et la droite.

On choisit le jeu de donnée avec l’erreur la plus faible.

**Limites et avantages de l’algorithme :**

Un avantage de RANSAC est sa capacité à calculer de manière robuste les paramètres du modèle, c'est-à-dire qu'il peut estimer les paramètres avec un degré élevé de précision, même si une quantité importante de valeurs aberrantes (outliers) est présente dans les données. Un inconvénient de RANSAC est qu'il n'y a pas de limite supérieure sur le temps de calcul de ces paramètres. Lorsqu'une limite est utilisée (un nombre maximal d'itérations), la solution obtenue peut ne pas être la solution optimale. Un autre inconvénient de RANSAC est qu'elle suppose de fixer des seuils spécifiques au problème traité.

**Pseudo-code RANSAC**

Entrées :

donnees : image seuillée

modeleDroite : une droite qui peut être ajusté à des données

echantillonMinimum : le nombre minimum de données nécessaires pour ajuster la droite

iterationsMax : le nombre maximal d'itérations de l'algorithme

seuilAppartenance : une valeur seuil pour déterminer si une donnée correspond à un   
 modèle

nombrePointsMin : le nombre de données proches des valeurs nécessaires pour faire   
 valoir que le modèle correspond bien aux données

Sorties :

meilleurModele : les paramètres du modèle qui correspondent le mieux aux   
 données (ou null si aucun bon modèle a été trouvé)

meilleurEnsemblePoints : données à partir desquelles ce modèle a été estimé

meilleureErreur : l'erreur de ce modèle par rapport aux données

Algorithme :

iterateur := 0

meilleur\_modèle := aucun

meilleur\_ensemble\_points := aucun

meilleure\_erreur := infini

**tant que iterateur < iterationsMax**

pointsAleatoires := echantillonMinimum valeurs choisies au hasard à partir des données

modelePossible := paramètres du modèle correspondant aux pointsAléatoires

ensemblePoints := pointsAléatoires

**Pour** chaque point des données pas dans pointsAleatoires

**si** le point s'ajuste au modelePossible avec une erreur inférieure à seuilAppartenance

Ajouter un point à ensemblePoints

**si** le nombre d'éléments dans ensemblePoints est > nombrePointsMin

*(ce qui implique que nous avons peut-être trouvé un bon modèle,*

*on teste maintenant dans quelle mesure il est correct)*

modelePossible := paramètres du modèle réajusté à tous les points de ensemblePoints

erreur := une mesure de la manière dont ces points correspondent au modelePossible

**si** erreur < meilleureErreur

*(nous avons trouvé un modèle qui est mieux que tous les précédents,*

*le garder jusqu'à ce qu'un meilleur soit trouvé)*

meilleurModele := modelePossible

meilleurEnsemblePoints := ensemblePoints

meilleureErreur := erreur

iterateur := iterateur +1

**retourne** meilleurModele, meilleurEnsemblePoints, meilleureErreur.