Reconstruction de billon

Alexis Barthelemy

Reconstruction de billon

Rapport de stage M2 AVR

Alexis BARTHELEMY

Encadrants : Isabelle DEBLED-RENNESSON et Phuc NGO

Equipe : ADAGIo

Du 16 mars 2020 au 31 août 2020

Table des matières

1 Introduction

Ce travail a été effectué au LORIA. Le LORIA, Laboratoire lorrain de Recherche en Informatique et ses Applications est une Unité de Recherche, commune à plusieurs Etablissements : le CNRS, l'Université de Lorraine et Inria. Le LORIA développe une recherche fondamentale et appliquée dans le domaine des sciences et technologies de l'information et de la communication, et ses compétences portent sur des secteurs en pleine évolution et porteurs de développement économique potentiel. Les travaux de recherche sont menés au sein de 28 équipes regroupées en 5 départements sur les thèmes suivants : "Algorithmes, Calculs, Géométrie, Image", "Méthodes Formelles", "Réseaux, Systèmes et Services", "Traitement Automatique des Langues et des Connaissances", "Systèmes Complexes, Intelligence Artificielle et Robotique".

L'équipe qui m'accueille est l'équipe ADAGIo faisant partie du département 1 du LORIA. ADAGIo est une équipe du LORIA créée depuis janvier 2006 suite à la restructuration des anciennes équipes Adage et Modbio. La thématique directrice de cette équipe porte sur l'algorithmique discrète. Construire un modèle discret d'un problème ou d'un phénomène du monde réel fait appel, sur le plan mathématique, aux structures discrètes, telles que graphes, mots, arbres, ensembles de points dans un espace, etc. L'étude des diverses propriétés de ces structures est l'objectif principal de cette équipe. La bioinformatique et l'imagerie servent d'une part de source de problèmes et d'autre part de domaines privilégiés pour tester et appliquer les méthodes et algorithmes.

Dans le cadre d'une collaboration avec l'INRA sur le projet TreeTrace [1] lié au traitement et l'analyse d'images de bois, le but de ce projet de recherche est de permettre le suivi de troncs d'arbre depuis l'exploitation forestière jusqu'à la scierie en utilisant l'empreinte biométrique des arbres, et de mesurer la qualité du bois à partir d'images en considérant des caractéristiques comme des cernes annuels, la position de la moelle, le centre géométrique du tronc d'arbre...

Ce stage a pour objectif de retrouver la position des planches de bois dans la section du billon, plus précisément on possède deux images, l'image du billon avant le débit des planches et une image contenant des planches obtenues après le débit. On cherchera à avoir en sortie la position des planches par rapport aux billons et afficher dans l'image de résultat, pour chaque section, des planches trouvées et remises dans le repère billon. On peut diviser

1. https ://anr.fr/Projet-ANR-17-CE10-0016

le sujet en deux parties, la segmentation des planches dans les image où l'on a les resultat de débit des planches d'un billon et la reconstruction du billon, c'est-à-dire replacer les planches dans le billon. On peut résumer le sujet avec le schéma de la Figure 1.

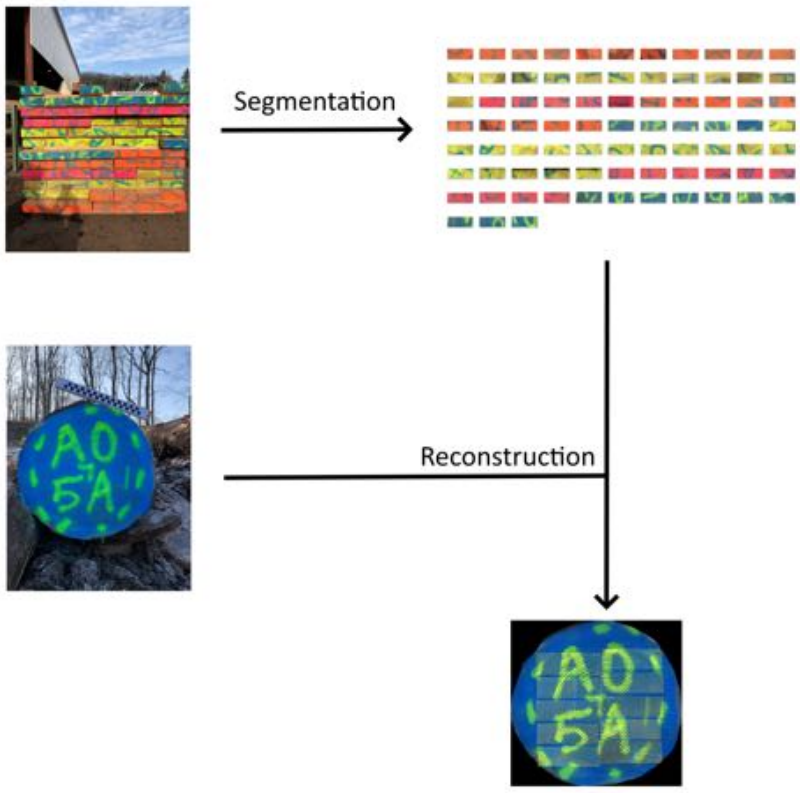


FIGURE 1 – Objectifs du projet

2 Segmentation

Dans cette partie l'objectif sera, à partir des images données d'un ensemble de planches, de récupérer chaque planche une à une. Ce travail a été réalisé sur deux jeux de donnés contenant des ensembles de planches, en commençant par le premier fourni au début du projet et un second fourni

après une première présentation de résultats.

Le premier jeu de données a été fourni au début du projet, puis le second jeu de données a été fourni en cours de projet. Le travail retenu est celui effectué sur le second jeu de données qui apporte plusieurs avantages et qui est plus récent.

Parmi les méthodes explorées dans le projet on peut distinguer deux types d'approches :

— Une approche basée sur les contours des planches
— Une approche basée sur les régions des planches

On présentera les deux jeux de données fournis puis on regardera la méthode basée sur les contours qui apportent des résultats plus satisfaisants enfin on parlera des pistes explorées pour résoudre le problème avec une approche basée sur les croissance de régions.

2.1 Jeux de données

2.1.1 Premier jeu de données

Ce jeu de données correspond à des images prises en 2015 dans une scierie, les images sont prises en intérieur. Dans ce premier jeu de données on dispose de deux piles de planches (Figure 2a) ainsi que les billons dont elles sont issues (Figure 2c). Les informations relatives aux planches sont les suivantes :

— Les piles de planches sont prise de différents points de vue de face.
— On ne voit toutes les planches sur chaque point de vue. (Figure 2b)
— Les planches apparaissent dans plusieurs de ces images.
— Les distances de capture des images sont aussi variables.
— Les planches sont posées les unes à côté des autres horizontalement.
— Plusieurs lignes sont superposées.
— L'écart entre deux planches sur une même ligne n'est pas fixe.
— L'écart entre deux lignes est fixe.
— On ne connaît pas le nombre de planches par ligne ni le nombre de lignes.
— Le nombre de planche par ligne n'est pas régulier.
— Les faces des planches ne sont pas sur un même plan.
— Les dimensions des planches ne sont pas connues.
— Aucune échelle de taille n'est disponible dans l'image

Les informations relatives aux billons sont les suivantes :

— Les billons sont vers le centre de l'image et sont totalement visibles.
— Certains autres billons sont visibles autours du billon principal.

— On peut différencier 4 types de marques colorées :
  — Un fond d'une couleur .
  — Au centre un disque d'une autre couleur.
  — Un chiffre écrit d'une différente couleur.
  — Des tirets proche des contours du billon d'une autre couleur que celle du fond.
— On a un marquage numérique en noir.
— Aucune échelle n'est disponible sur l'image.
— Une seule capture est effectuée par billon.

(a)

(b)

(c)

(d)

FIGURE 2 – (a) Une des piles de planches (b) Un des points de vue de la pile (c) Un des billons (d) Verité terrain de la reconstruction

2.1.2 Second jeu de données

Ce jeu de données correspond à des images prises en 2020 dans une scierie, certaines les images sont prises en intérieur mais la majorité est prise en extérieur. Dans ce second jeu de données on dispose de cinq piles de planches (Figure 3a) ainsi que les billons dont elles sont issues (Figure 3c). Les informations relatives aux planches sont les suivantes :

- On dispose de 2 points de vue des piles de planches :
  - La face colorée des planches (Figure 3a).
  - La face non colorée des planches où elles sont numérotées (Figure 3b).
- Les planches sont rangées les unes à coté des autres horizontalement.
- Plusieurs lignes sont superposées.
- L'écart entre deux planches est faible mais pas fixe.
- L'écart entre deux lignes est fixe.
- On ne connaît pas le nombre de planches par ligne ni le nombre de lignes.
- Le nombre de planches par ligne est régulier sauf pour la ligne au sommet de la pile.
- Sur la face colorée toutes les faces des planches ne sont pas sur un même plan
- Sur la face numérotée toutes les faces des planches sont sur un même plan
- Les planches sont regroupées par billon duquel elles ont été extraites.
- La dimension des planches est de 160 mm par 55 mm.
- Une échelle est disponible pour les deux points de vue.
- L'environnement est très présent sur chaque point de vue.

Les informations relatives aux billons sont les suivantes :

- Trois captures sont réalisées par billon :
  - Face colorée sans échelle
  - Face colorée avec échelle
  - Face non colorée sans échelle
- Les billons sont vers le centre de l'image et sont totalement visibles.
- Certains autres billons sont partiellement visibles autour du billon principal.
- On peut différencier 3 types de marques colorées :
  - Un fond d'une couleur.
  - La numérotation du billon d'une couleur différente du fond.
  - Des tirets proches du contour du billon.

(a)

(b)

(c)

(d)

FIGURE 3 – Une des piles de planche (a) face colorée (b) face numérotée | Un des billons de la pile (le plus bas numéroté 1 à 11) (c) sans l'echelle (d) avec l'échelle

Les échelles sont des mires placées dans l'image, ce sont des morceaux de bois où une feuille est agrafée avec un motif en échequier sur 2 lignes et 25

colonnes où chaque cellule est un carré de coté 10 mm.

Ce second jeu de données apporte de nouveaux avantages mais aussi quelques inconvénients.

Parmi les avantages on connaît maintenant les dimensions d'une planche et grace à nos échelles on peut retrouver pour la pile de planche la taille en pixel de nos planches et pour faire la reconstruction des billons on connaît la transformation d'échelle à faire pour positionner les planches dans le billon. Les marquages sur les billons sont plus simples, seulement deux couleurs sont présentes par billon. Le rangement des piles de planches aide à associer les planches entre elles, il sera plus simple d'associer les groupes de planches avec les billons par la suite. La disposition des planches plus régulière apporte aussi quelques avantages.

Dans les inconvénients on a maintenant une plus forte présence de l'environnement dans les images. Les écarts entre les planches, parfois très petits, rendent la distinction entre certaines planches difficiles, notamment sur la face colorée.

2.2 Approche basée contours

La détection de bord permet de repérer les points d'une image qui correspondent à un changement brutal de propriété. Ces changements de propriétés dans l'image indiquent que des éléments importants de la structure de l'objet représenté sont présents à cet endroit. Ces éléments peuvent être des discontinuités dans la couleur d'une surface, dans l'éclairage ou la profondeur de l'élément. La détection des bords réduit de manière significative la quantité de données en conservant des informations pertinentes. On pourra ensuite analyser ces informations pour en déduire les contours de nos planches.

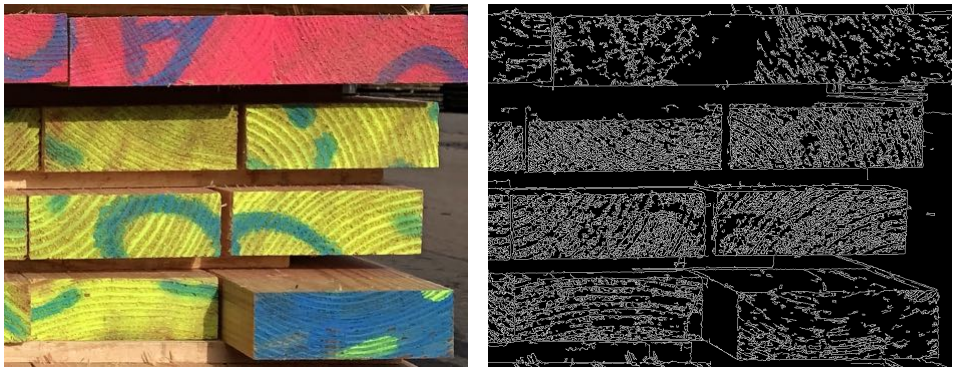
On utilisera dans cette section une méthode basée sur le filtre de Canny pour récupérer les bords dans l'image, puis une détection des contours des planches avec la transformée de Hough.

2.2.1 Filtre de Canny

Cette première approche vise à récupérer les contours des planches en utilisant le filtre de Canny [1]. Cet algorithme conçu par John Canny en 1986 est un algorithme composé de 5 étapes :

— On applique un filtre gaussien pour débruiter l'image.
— On trouve le gradient d'intensité de l'image.
— On supprime les points qui ne sont pas des maxima locaux.
— On effectue un double seuillage pour trouver les contours potentiels.

— On finalise la détection par hystérésis, c'est a dire qu'on garde point fort (supérieur au seuil haut) et faible (supérieur au seuil bas et inférieur au seuil haut) connecté à des points fort.



(a) (b)

FIGURE 4 – (a) Image source (b) Application du filtre de Canny sur l'image source

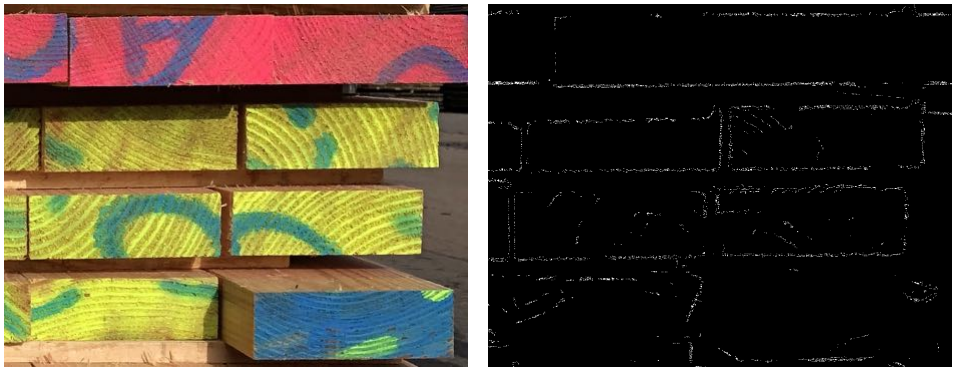
Après plusieurs réglages de paramètres, on remarque qu'il est difficile d'avoir les contours des planches sans avoir aussi les cernes de l'arbre comme on peut le voir dans la Figure 4.

Pour tenter de contourner ce problème, deux approches ont été testées ;

— Remplacer le filtre gaussien par un filtre bilateral [2]

— "Nettoyer" le résultat de la première application de l'algorithme.

La première approche représentée dans la Figure 5 montre que le nombre de bords détectés a grandement diminué au niveau des cernes de l'arbre. L'effet du filtre bilatéral semblerait avoir un impacte positif en vue du résultat recherché, cependant on perd aussi beaucoup de contours.



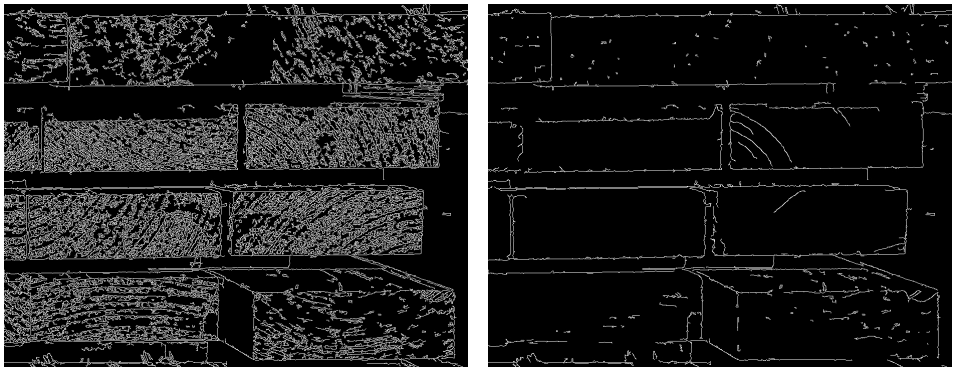
(a) (b)

FIGURE 5 – (a) Image source (b) Application du filtre de Canny avec un filtre bilateral sur l'image source

La seconde approche testée, représentée dans la Figure 6, montre que les régions où l'on avait beaucoup de bords parasites ont été largement vidées et que l'on récupère mieux les contours des planches. En tournant la surdétection des bords à l'intérieur des planches à notre avantage, si on suppose que les cernes dans les planches sont surdétectés alors on a une forte concentration de bord. En appliquant l'algorithme suivant on peut valoriser les régions au bord des régions où l'on remarquait une surdétection.

```
canny = filtreCanny(image_source)
// Récupération des régions surdétectés
region_planche = fermetureMorphologique(canny)
// Récupération des bords ce ces régions
element = rectangle(5x5)
dilatation = dilatationMorphologique(region_planche, element)
erosion = erosionMorphologique(region_planche, element)
mask_region = inverserCouleur(dilatation - erosion)
// Floutage des régions surdétectés
image_valorisee = flouGaussienFort(image_source, mask_region)
// Extraction des bords
bords = filtreCanny(image_valorisee)
```

En réalisant une fermeture morphologique avec un élément structurant assez grand on obtient les régions où l'on est censé trouver les planches. En récupérant les bords de ces régions et en floutant fortement ce qui n'est pas un bord, on peut après une nouvelle détection des bords avec le filtre de Canny récupérer seulement les contours de ces régions (Figure 6b).



(a) (b)

FIGURE 6 – (a) Filtre de Canny (b) Filtre de Canny "nettoyé"

Ces deux méthodes donnent des résultats positifs, la deuxième donne quand même des résultats plus nets ce qui favorise son utilisation dans la suite de la méthode de détection des planches.

2.2.2 Détection de lignes et affinement

Cette seconde partie de la détection des planches se base donc sur les bords détectés dans la section précédente. On va rechercher avec ces bords des informations plus globales dans l'image dans un premier temps puis on travaillera de plus en plus localement pour extraire les contours de nos planches. On peut décomposer la suite de la détection en 5 sous-parties :

— Détection des droites horizontales.
— Regroupement des droites horizontales.
— Découpage des régions contenant une ligne de planche.
— Détection des droites verticales dans les lignes de planches.
— Positionnement de boîtes englobantes.

La première partie de l'algorithme consiste à découper notre pile de planches en plusieurs lignes de planches. Il est possible d'extraire des droites horizontales d'une image à partir d'alignements de points avec la transformée de Hough [3].

Le principe est de trouver toutes les droites passant par chaque point considéré comme un bord et de trouver les droites qui sont communes à un maximum de points. On exprimera ces droites en coordonnées polaires ($r_\theta = P.x \cdot \cos\theta + P.y \cdot \sin\theta$), elles sont donc représentées par deux paramètres, que l'on représentera dans un espace d'accumulation. À chaque point dans l'image considérée comme un bord on incrémente les valeurs des accu-

mulateurs correspondant aux paramètres des droites passant par ces points. Les maxima locaux de cet espace d'accumulation correspondant à des droites probables dans notre image.

On utilisera ici la transformée de Hough pour trouver toutes les droites horizontales dans l'image à partir des bords détectées précédemment. C'est-à-dire les points dans notre accumulateur qui ont pour paramètre un angle proche de 0 ou 2π et ayant leur valeur r_θ supérieure à un certain seuil. On obtient les droites de la Figure 7a.

On obtient un grand nombre de droites détectées pour chaque ligne de planches, ces droites sont détectées entre deux lignes de planches, ce sont donc ces droites qui vont servir comme base de la récupération de la ligne de planche complète. Pour ça on va chercher à regrouper les droites pour obtenir une seule droite entre chaque ligne de planches, on va alors créer des groupes de droites. On utilisera l'algorithme suivant pour regrouver les droites :

```
Pour chaque droite faire
    G = groupe ou la distance entre le groupe et la droite est le plus faible
    Si distance(g, droite) < hauteur_planche alors
        On ajoute la droite au groupe G
        On met à jour la position moyenne du groupe
    sinon
        On ajoute un nouveau groupe auquel on ajoute la droite
    fin si
fin pour
```

On obtient finalement un certain nombre de groupes de droites avec leur position moyenne, si on dessine une droite pour chaque groupe de ligne on obtiens le résultat présenté dans la Figure 7b. On obtient une droite entre chaque ligne de planche.

(a) (b)

FIGURE 7 – Récupération des droites (a) toutes les droites (b) droites moyennes

À partir de ces droites, on peut alors extraire de l'image une région correspondant à une ligne de planches, on se sert de chaque droite moyenne comme base de la région d'une ligne et on récupère la partie au-dessus de la ligne sur une hauteur correspondant à la hauteur d'une planche.

Dans cette région on va ensuite détecter les droites verticales, elles devraient correspondre aux bords des planches dans la ligne. On remarque dans la Figure 8 que les bords des planches ne sont pas tous détectés, en effets certains sont difficiles à détecter même pour une personne, par exemple les deux planches roses à droite dans la troisième ligne en partant du haut.

FIGURE 8 – Récuperation des bords verticaux

Comme on ne détecte pas tous les bords de planches on ne va pas seulement couper nos lignes de planches avec les droites verticales détectées mais aussi avec des bords reproduits. Si la droite verticale la plus proche d'une droite est à une distance supérieure de la largeur théorique d'une planche plus une marge, alors on ajoute une droite verticale à cet endroit. On récupère ensuite tous les intervalles entre deux lignes verticales qui sont d'une taille proche de la largeur d'une planche dans l'image.

On obtient alors les boîtes présentées dans la figure 9a. On peut voir que la plupart des planches ne sont pas directement au-dessus des lignes moyennes trouvées précédemment et que les boîtes sont soit un peu au-dessus soit un peu au-dessous des lignes.

On peut corriger encore un peu la position de chaque boîte en faisant une nouvelle détection de lignes horizontales dans la région de la boîte, si on détecte plus d'une droite dans la partie inférieure ou supérieure de la boîte alors on décale la boîte.

(a) (b)

FIGURE 9 – Recuperation des boîtes (a) avec les verticales (b) ajustées verticalement

2.3 Approche basée région

La segmentation basée sur les régions est une méthodes qui manipule directement des régions. Deux approches sont possibles pour trouver ces régions :

— Soit elles partent d'une première partition de l'image, qui est ensuite modifiée en divisant ou regroupant des régions, et on parle alors de méthodes de décomposition/fusion.

— Soit elles partent de quelques régions, qui sont amenées à croître par incorporation de pixels jusqu'à ce que toute l'image soit couverte, et on parle alors de méthodes par croissance de régions.

On utilisera dans cette section une méthode par croissance de régions. L'algorithme de ligne de partage des eaux (watershed) [4] consiste à interpréter notre image comme un relief dont on vas simuler l'inondation. Pour fonctionner, l'algorithme a besoin d'une liste de marqueurs correspondant aux régions de l'image à segmenter.

On essayera deux approches différentes pour trouver ces marqueurs automatiquement, les deux méthodes sont dépendantes des informations connues sur les images traitées donc ne peuvent pas êtres utilisées dans l'autre cas.

— La première approche était basée sur les informations du 1er jeu de données, on avait de grands espaces entre les planches, on essaye alors de classer au mieux les régions contenant des planches et les régions qui n'en contiennent pas.

— La seconde approche consiste quant à elle à utiliser l'organisation régulière des planches dans la pile ainsi que les chiffres écrits sur la face non colorée des planches.

Suite à ces deux récupérations de marqueur on récupérera les régions dans l'image avec l'algorithme de la ligne de partage des eaux. De la même façon on récupérera les planches en fonction des informations de l'image.

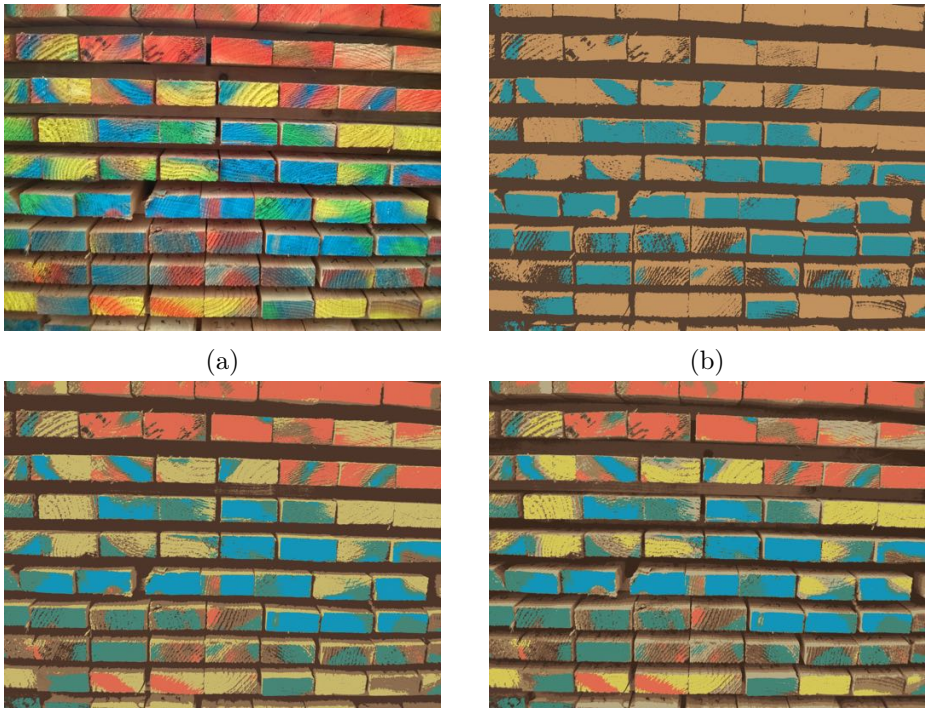
2.3.1 K-mean et flou de déplacement

Pour obtenir ces planches on va pour le premier jeu de données utiliser l'algorithme suivant :

```
// Simplification des informations de l'image
k_image = kmean(image_source, 3) // On crée 3 classes de couleur
// Récupération des marqueurs
distance = distanceTransform(k_image)
markers = seuillage(distance)
// Application du watershed
watershed = watershed(k_image,markers)
// Recupération des planches
boites = recuperationRectangles(watershed)
boites = boites + voisins(boites)
lignes = flouDeplacement(image_source)
planches = ajusterPosition(boites, lignes)
```

La première approche permettant de regrouper des informations est d'utiliser l'algorithme kmeans [5] dans notre espace de couleur. Ça consiste pour notre ensemble de pixels coloré, de les partitionner en k ensembles en minimisant la distance entre la couleur de chaque point de chaque ensemble et la couleur moyenne des points de l'ensemble. Cela aura pour effet de nous donner les k couleurs majoritaires dans l'image que l'on pourra utiliser pour remplacer dans notre image les couleurs des pixels par la couleur dans laquelle l'algorithme les a classés.

En observant les résultats obtenus dans la Figure 10 on peut voir par exemple qu'avec une valeur de K égale à 3 on peut facilement délimiter ce qui serait dans notre cas le fond de l'image, c'est-à -dire l'espace entre les planches, et ce qui serait important pour nous, les planches. On peut aussi noter qu'avec un K plus élevé on élimine plutôt bien la texture du bois et on obtient un rendu un peu simple de notre planche ce qui pourrait être utile pour la reconstruction à l'intérieur du billon. On utilisera cet algorithme avec $k = 3$ pour récupérer les planches (Figure 10b).



(a) (b)

(c) (d)

FIGURE 10 – (a) Image source (b) K = 3 (c) K = 6 (d) K = 9

On considère maintenant chaque région connexe de même couleur pour faire la suite de la détection. On calcule pour chaque pixel d'une région la distance au bord de sa région. On appliquera enfin un seuil sur notre distance pour ne récupérer que les pics de notre image.

On récupère des régions correspondant aux planches avec l'algorithme de ligne de partage des eaux et on observe que les régions sont séparées au sein d'une même ligne (Figure 11). À noter que l'on perd certaines planches dans les dernières lignes, qu'il y a quelques fusions de planches et que certaines planches sont sursegmentées ou ont des parties rognées par le fond de l'image.

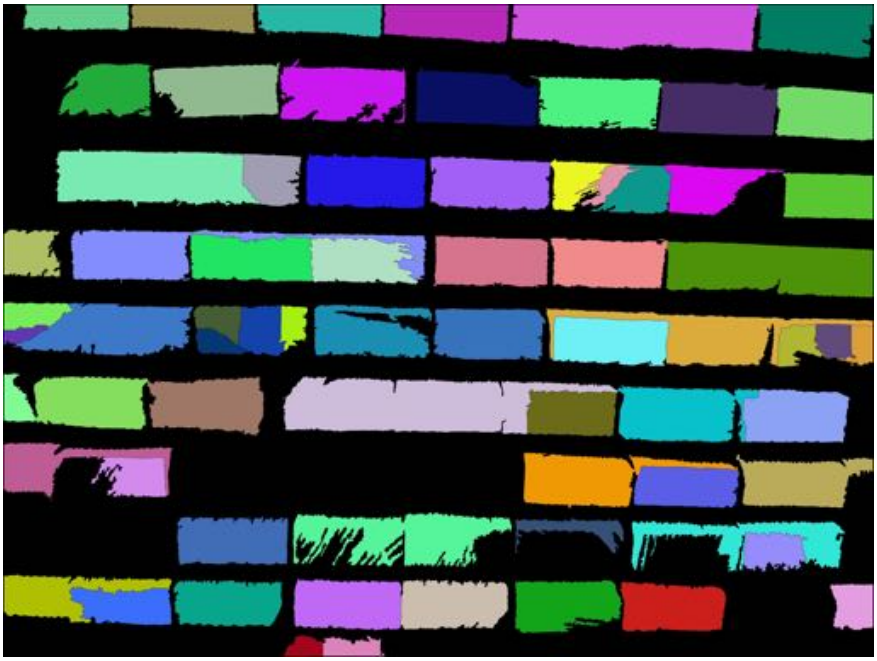
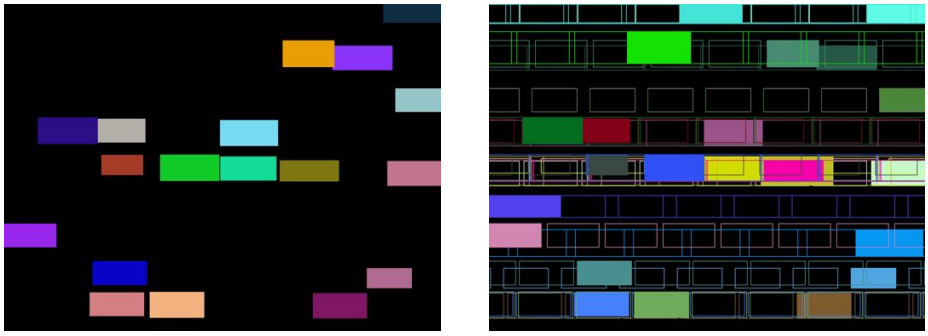


FIGURE 11 – Résultat apres kmean avec K = 3 et le watershed

On va ensuite extraire les bords de nos régions après la ligne de partage des eaux devant contenir les planches pour pouvoir faire une approximation de rectangle et ainsi trouver les planches. On obtient les boîtes présentées dans la Figure 12a.

À partir de ces boîtes on va trouver un gabarit moyen qui va servir à décaler des boîtes sur les lignes ou une boîte a été détectée. On obtient donc avec les boîtes détectées précédemment les boîtes répliquées dans la Figure 12b.

Si on observe les boîtes répliquées on peut voir plusieurs problèmes, il y a une superposition de plusieurs nouvelles boîtes et les boîtes ne sont pas vraiment alignées sur une ligne contenant des planches. On va commencer par regarder comment on pourrait aligner les nouvelles boîtes répliquées avec l'image d'origine.



(a) (b)

FIGURE 12 – (a) boîtes englobante (b) boîtes englobantes et repliques

Une approche testée qui semble donner de bons résultats se base sur une méthode pour flouter une image de sorte à reproduire une impression de mouvement de la caméra lors de la capture de l'image. Pour donner cet effet on va simplement faire passer un filtre sur notre image en niveau de gris.

Le filtre utilisé est un filtre carré de côté D. Où la ligne centrale $\frac{D}{2}+1\times(D \mod 2)$ vaut 1 et tout le reste du filtre vaut 0.

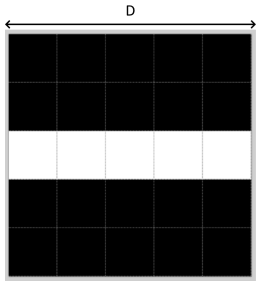
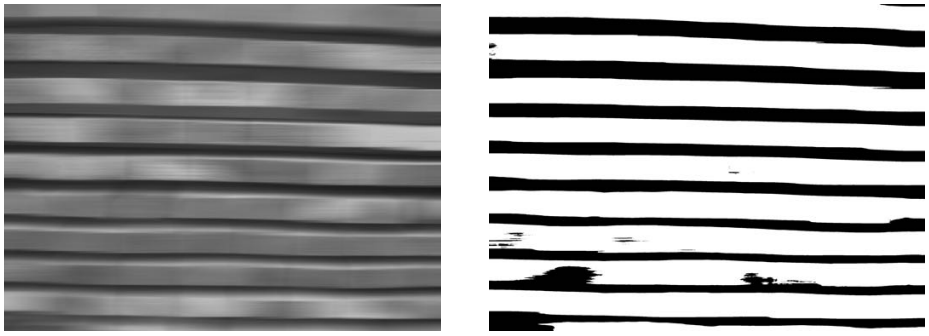


FIGURE 13 – Filtre carré de taille 5x5 (D = 5)

Plus D est grand plus l'impression de flou de déplacement sera forte.

Dans notre cas pour discriminer les lignes on voudra réduire au maximum les informations des planches et du fond pour pouvoir les différencier donc on choisira un nombre D grand. On obtient le résultat présenté dans la Figure 14a. Il suffit ensuite de faire une binarisation de l'image pour obtenir les lignes contenant les planches comme dans la Figure 14b.



(a) (b)

FIGURE 14 – (a) Flou de deplacement avec D = 500 (b) Lignes de planches

On va ensuite utiliser le résultat de la Figure 14b pour déplacer nos boîtes répliquées afin d'obtenir le meilleur alignement des boîtes. Au moment de la création de la réplique, on regardera si déplacer la boîte vers le haut ou vers le bas augmente le nombre de pixels que la boîte contient dans notre image des lignes. On gardera alors la position pour laquelle la boîte contient le plus de pixels blancs.

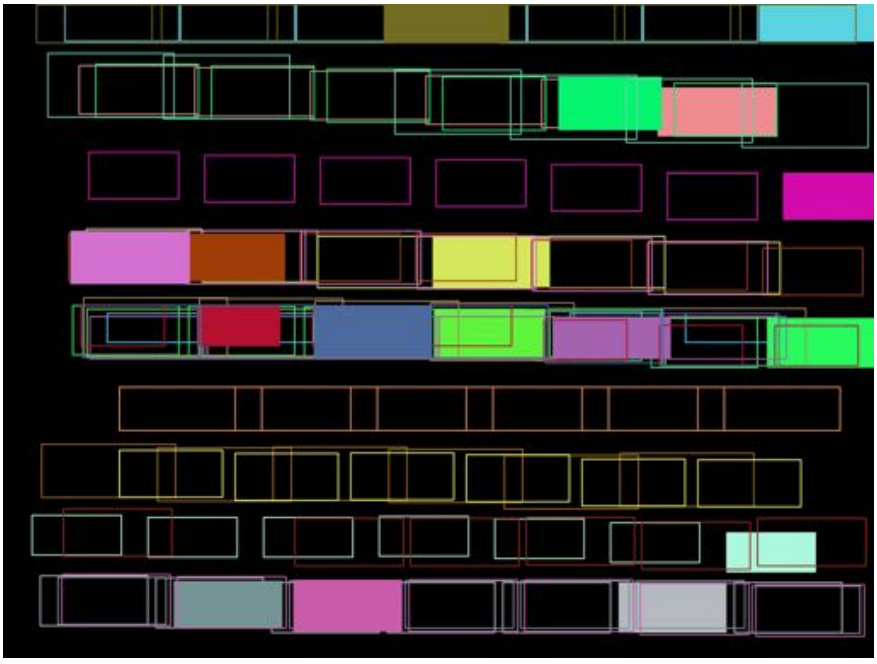


FIGURE 15 – boîtes englobantes et repliques alignés

Il faudrait toujours éliminer les superpositions de boîte et pour supprimer les superpositions, choisir laquelle de ces boîtes devrait être sélectionnée. On ne continuera pas d'explorer cette solution car le nouveau jeu de donné a été fournis à ce moment là.

2.3.2 Positionnement d'une grille

Dans cette section on va aborder une nouvelle stratégie, grace aux informations sur les planches on peut chercher à déposer un arrangement de boîte sur notre image. En effet, on connaît les dimensions des planches ainsi que leurs dispositions dans l'image. L'idée sera donc de créer une grille qu'il faudra placer correctement dans notre image pour qu'elle colle correctement aux régions des planches. On utilisera l'algorithme suivant pour récupérer les planches.

```
// Rotation de l'image source
rotation = detectionRotation(image_source_nonColorée)
// Extractions des zones ou sont écrit les nombres
bin = binarisation(rotation)
bin_propre = nettoyage(bin)
// Création de la grille servant de marqueur
grille = creationGrille()
grille.positionner(bin_propre)
// Application de l'algorithme de la ligne de partage des
//   eaux
watershed_result = watershed(image_source,grille)
// Positionnement d'une boite par marqueur en les alignant
//   au watershed
boites[] = poserBoite(watershed_result,grille)
// On décale les boites pour éviter les superpositions
Pour chaque boite dans boites faire
    boite.eliminerSuperposition(boites)
Fin pour
// On les retournes pour les placer dans l'image des
//   planches colorées
boites[] = boites.miroir()
// On trouve le coin de la pile de planche dans l'image
//   de la pile de planches colorées
coin = trouverCoinPile(iamge_source_colorée)
iamge_source_colorée.poserBoites(boites, coin)
```

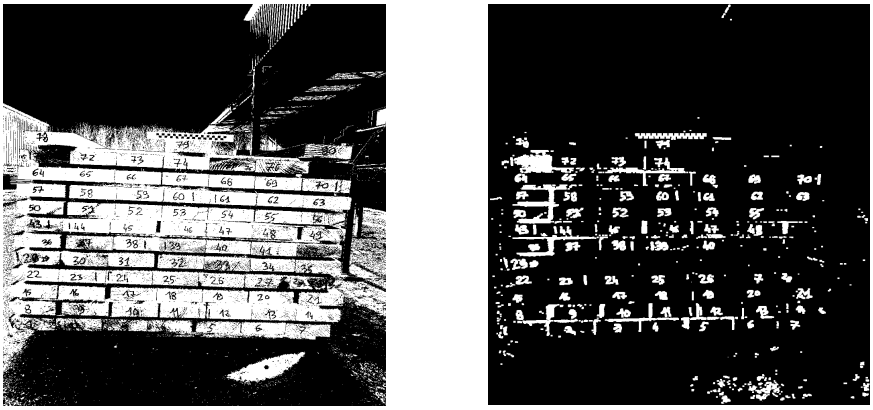
Il faut dans un premier temps déterminer de quel angle nos planches sont tournées dans l'image, pour trouver cet angle on va utiliser la transformée de Hough. Sachant que nos images sont composées de plusieurs lignes de planches empilées il n'est pas absurde de dire que les droites comportant le plus grand nombre de points sont les droites parallèles aux bords horizontaux des planches. On peut donc choisir l'angle dans lequel on doit effectuer la rotation de notre image par rapport à ces droites. Cependant on ne choisit pas la droite maximum selon Hough car on peut tomber sur des droites correspondant à des objets de l'environnement, on choisira plutôt l'angle median des 10 droites ayant les plus grandes valeurs dans leurs accumulateurs dans l'espace de Hough.

Une fois la rotation de notre image effectuée on peut essayer de positionner la grille.

Pour placer notre grille on utilisera à notre avantage les chiffres écrits sur les planches sans couleurs. Les planches étant plutôt de couleur claire et les chiffres écrits en noir au centre des planches, il suffit de binariser notre image pour obtenir une marque sur chaque planche (voir Figure 16a).

Il faut cependant traiter un minimum notre binarisation. Premièrement on peut inverser la binarisation car ce qui nous intéresse est la partie sombre des planches où sont écrits les numéros des planches. Ensuite on fait une dilatation de l'image avec un élément structurant carré de taille 5x5 pour augmenter la taille des régions des chiffres. On sait aussi que nos planches ne se trouvent pas directement sur les bords de l'image donc une région connexe au bord ne devrait pas correspondre à un chiffre, on peut donc éliminer ces régions. Enfin les chiffre étant écrit sur les planches leur aire de recouvrement ne peut donc pas être supérieure à l'aire d'une planche dans l'image, on peut aussi éliminer les régions de l'image ayant une aire supérieure à celle d'une planche.

On obtient alors un résultat similaire à la Figure 16b. On remarque la présence d'une grande partie des nombres inscrits sur les planches. Il y a quand même certaines régions restantes qui ne correspondent pas à des nombres, mais cela ne devrait pas gêner notre positionnement de la grille.



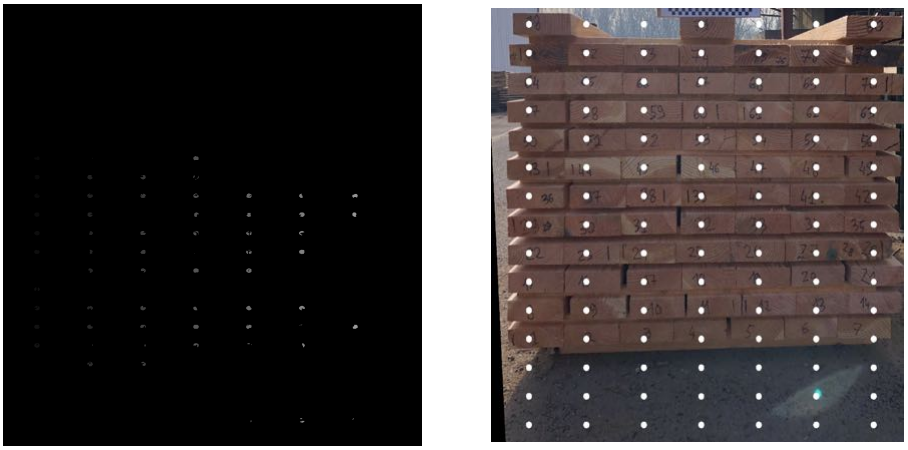
(a) (b)

FIGURE 16 – Image des planches avec les nombres (a) binarisée (b) tournée, binarisée et netoyée

À partir de cette détection des régions des nombres sur les planches on peut essayer d'aligner une grille naïve en essayant de créer une superposition entre nos points de la grille et les zones correspondant aux nombres.

On va créer la grille à partir du point haut gauche de l'image avec des points espacés de la dimension des planches, un point de la grille sera représenté par un cercle centré sur le point de rayon 10 pixels.

On compte ensuite le nombre de pixels en intersection (Figure 17a) entre la grille et les régions des nombres, plus le nombre de pixels en intersection est grand plus notre grille devrait être positionnée sur les planches. Pour trouver la meilleure position pour notre grille on parcourt toutes les positions possibles dans l'image en faisant une translation de la grille à partir du coin haut gauche jusqu'au point qui correspondrait au coin bas droit d'une planche placée avec son coin haut gauche à l'origine de l'image. On garde alors la position pour laquelle le nombre de pixels en intersection est le plus grand. On peut voir sur la Figure 17b que les points sont globalement placés sur une planche.



(a) (b)

FIGURE 17 – Grille (a) en intersection avec les régions des nombres (b) superposée sur l'image d'origine tournée

Il faudra ensuite pour chacun des points trouver la région de la planche à laquelle il correspond.

Pour retrouver ces régions nous allons utiliser l'algorithme de la ligne de partage des eaux. Cette fois-ci nous utiliserons donc les points de notre grille comme marqueurs.

Le résultat attendu serait d'avoir les régions contenant la planche seule pour chaque marqueur, or comme on ne définit pas de zone extérieure aux planches, il y a des débordements comme on peut le voir dans la Figure 18b. On remarque qu'on obtient quand même un certain nombre de régions de forme rectangulaire correspondant à une planche et qu'on a dans la majorité des cas au moins un bord auquel on peut se référer pour placer une boîte qui engloberait notre planche.

(a) (b)

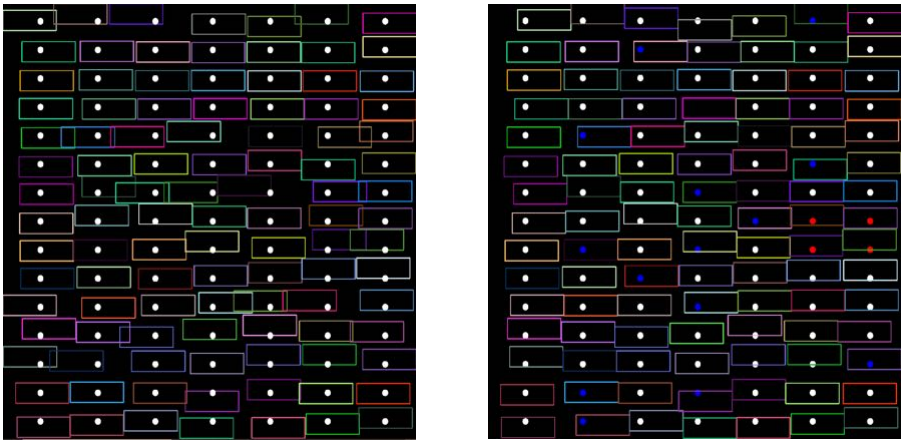
FIGURE 18 – Algorithme de la ligne de partage des eaux (a) Marqueurs (b) Résultat

L'étape suivante consiste donc à positionner pour chaque marqueur une boîte englobante dans la région associée au marqueur. On va donc se baser sur deux critères pour positionner nos boîtes : -

- Le nombre de pixels dans la boîte appartenant à la région du marqueur.
- Le nombre de pixels proches des bords de la boîte étant noté comme frontière entre deux régions.

Ces deux critères vont permettre de déposer les boîtes pour qu'elles aient un maximum de couverture de la zone associée à leur marqueur et qu'elles se collent sur les bords dans le cas où on aurait seulement deux bords corrects dans une région avec des débordements sur les autres cotés.

On obtient donc le résultat de la Figure 19a, on peut observer quelques superpositions de boîtes ainsi que des espaces vides or on ne peut trouver deux planches superposées dans l'image. Sachant que l'on a des espaces vides entre certaines boîtes et que les planches dans l'image sont plutôt alignées verticalement on peut dans le cas des boîtes en superposition choisir de les déplacer et de les coller bord à bord avec la boîte auquel elle est superposée, pour choisir quelle boîte devra être déplacée on peut regarder laquelle semble le plus alignée verticalement aux autres boîtes de sa colonne. En effectuant ce déplacement on obtient le résultat de la Figure 19b.



(a) (b)

FIGURE 19 – Positionnement des boîtes (a) suivant les bords (b) apres déplacement de superposition

À partir de ces boîtes correspondant aux planches de la partie non colorée on aimerait bien retrouver la correspondance dans dans l'image contenant les planches colorées pour pouvoir les replacer ensuite dans le billon.

On sait que les planches sont dans les mêmes positions les unes par rapport aux autres dans les deux côtés de la pile de planche donc retourner notre grille et nos boîtes et les repositionner dans l'autre image en conservant un coin de la pile de planche comme repaire devrait nous donner une bonne segmentation pour les planches colorées.

On va donc reproduire notre grille en récupérant son miroir et commençant à poser la première boîte dans le coin haut gauche de notre pile de planches colorées.

On remarque que la différence du point de vue entre les deux images pose problème, les points de la grille sont décalé par rapport aux planches dans la partie colorée, les boîtes le sont encore plus. On peut aussi voir que les faces des planches étant toutes sur le même plan dans l'image des planches numérotées et ne l'étant pas dans l'image des planches colorée ne favorisent pas la correspondance des boîtes avec les planches.

(a) (b)

FIGURE 20 – Positionnement des boîtes (a) coté numéroté (b) coté coloré

On abandonne donc la solution pour essayer d'obtenir une segmentation correcte des planches, ajouter la position dans l'image reviendrait presque à recommencer un traitement sur l'image.

3 Reconstruction

Dans cette seconde partie du projet, l'objectif sera à partir des planches segmentées et des billons, de reconstruire le billon, c'est-à-dire de placer chaque planche dans le billon à l'endroit d'où elle provient avant la découpe.

On esayera deux types d'approches :

— Gestion de la translation avec une comparaison pixel à pixel entre les planches et les billons.

— Associations de features.

La distance euclidienne et le template matching appartiennent à la première approche tandis que SURF se base sur les feature.

Les problèmes rencontrés en utilisant la distance euclidienne sont principalement résolus avec l'approche du template matching. Pour les problèmes rencontrés avec SURF aucune solution n'a été proposée. On retiendra donc la méthode de template matching qui donne de meilleurs résultats.

On comprend tout de suite qu'il y aura des problèmes si on ne gère que la translation pour les premières méthodes :

— La taille de l'image de la planche est inférieure à la taille de la planche dans l'image du billon. (Figure 21 à gauche)

— Les planches ne sont pas toutes dans le même sens, une planche peut être retournée par rapport à sa position sur la pile de planche. (Figure 21 au millieu)
— Les planches ne sont pas toutes coupées horizontalement dans le billon. (Figure 21 à droite)

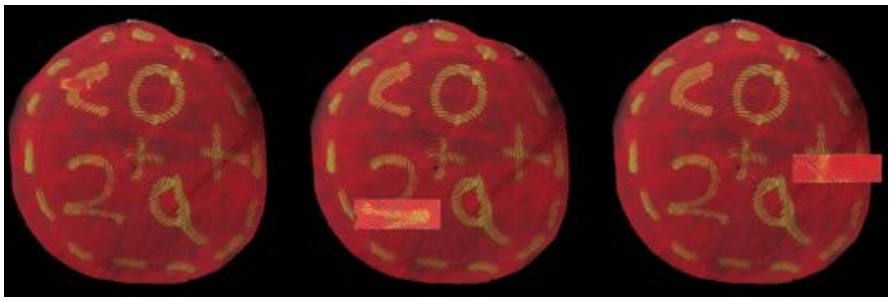


FIGURE 21 – Problèmes de la translation (les translations sont celles de la veritée terrain)

On peut résoudre le 1er problème grâce aux mires présentes dans les images des piles de planches et des billons, il suffit de mesurer le ratio pixel/cm dans chacune des images et d'ajuster la taille des images des planches pour qu'elles correspondent à la taille qu'elles devraient avoir dans les images de billons.

On devra utiliser une stratégie pour gérer à la fois la rotation et l'échelle dans le cas où on a toujours des problèmes après la mise à l'échelle en utilisant la mire. Deux petits algorithmes vont être utilisés :

```
// Ici les planches sont supposé être à la bonne echelle
Fonction trouverAngle(planches[], billon) : angle
    Pour chaque angles dans [0,360] avec un pas de 15 faire
        rotation(billon)
        score_angle = 0
        Pour chaque planche dans planches faire
            score_angle += match(planche, billon)
        Fin pour
        liste_score[angle].score = score_angle
    Fin pour
    Pour chaque angle dans [0,180] avec un pas de 15 faire
        liste_score[angle].score += liste_score[angle + 180].score
    Fin pour
```

```
    // Dans le cas de la distance on cherche le plus petit score
    retourner indiceMin(liste_score[0,180])
    // Dans le cas du template matching on cherche le plus grand score
    retourner indiceMax(liste_score[0,180])

// Ici on suppose qu'on à déjà le bon angle pour chaque planches
Fonction changerScaling(planches[], billon, score_planches[]) : scale_planche[]
    Pour rescale_factor de 0.85 a 1.15 avec un pas de 0.01 faire
        Pour indice_planche de 0 a planches.size() faire
            planche_test = planche[indice_planche].rescale(rescale_factor)
            score = match(planche_test, billon)
            Si score > score_planches[indice_planche] alors
                scale_planche[indice_planche] = rescale_factor
            Fin si
        Fin pour
    Fin pour
    retourner scale_planche[]
```

La première fonction sert à trouver l'angle général dans lequel les planches sont placées dans le billon. L'idée est de faire un matching pour chaque angle et pour chaque planche. On accumule le score de matching de chaque planche pour chaque angle puis on retourne l'angle qui a le meilleur score en fonction de la méthode de matching.

La seconde fonction sert à corriger l'erreur de scaling au qui peut venir du moment de la mesure manuelle de la taille de la mire dans une des images ou des positions des planches dans la planche de planche. On reprend la même idée, pour chaque planche et pour chaque échelles de -15% à +15% de l'échelle mesurée, on fait un matching et on garde le meilleur score pour chaque planches.

On aura dans chaque méthode seulement la méthode de calcul de score et de comparaison de score qui différera.

3.1 Distance euclidienne

La première approche testée est une approche plutôt naïve, elle consiste à comparer pixel à pixel les couleurs de l'image de la planche et la couleur de l'image du billon. On compare l'image de la planche à chaque endroit de l'image du billon qui pourrait accueillir l'image de la planche. Pour comparer deux pixels entre eux on compare leurs composantes RGB, on realise un simple calcul de distance euclidienne entre leurs vecteurs de couleur RGB. Plus la distance entre deux pixels est élevée moins les pixels se ressemblent.

Pour comparer l'image de la planche à la région dans l'image du billon on prend alors la somme des distances de chaque comparaison de pixels. On obtient alors pour chaque pixel où l'on peut mettre l'image de la planche dans l'image du billon une valeur R correspondant à la distance entre les vecteurs couleurs de limage des planches T et l'image du billon I, soit :

$R(x, y) = \sum_{x',y'}(T(x', y') - I(x + x', y + y'))^2$

avec x et y les coordonnées du point de l'image du billon et x et y les coordonnées dans l'image de la planche.

Avec la rotation calculée pour la planche on semble avoir la planche placée à une position incorrecte, en observant les scores de matching dans la carte de score (image en noir et blanc) en chaque point (image en niveau de gris, plus le point est noir plus le matching est bon) on remarque dans la Figure 22 qu'il y a une zone remarquable, le cercle gris plus foncé semble correspondre à la bonne position de la planche mais il n'a pas de score plus bas que la partie basse ou on a placé la planche initialement.

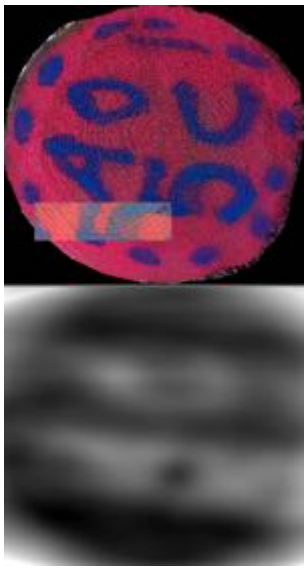


FIGURE 22 – Matching avec la distance en retournant la planche

3.2 Template matching

Dans la section sur la distance euclidienne on a déjà vu que le matching n'était pas très bon. Une simple différence de luminosité entre les planches et le billon entraîne des décalages pour les planches ou des scores très bons

pour des positions qui ne sont pas les bonnes. Les planches n'ayant pas la même couleur due à l'ensoleillement sur une des images et pas l'autre ou une ombre sur une partie de la planche ou du billon brouillent la comparaison.

On cherche donc un moyen de réduire l'impact de la luminosité des images sur la mesure du score d'une position de la planche. Deux solutions sont possibles :

— Changer d'espace de couleur pour moins prendre en compte la luminosité.

— Prendre en compte la moyenne de couleurs sur chaque partie à faire correspondre.

OpenCV propose une fonction de matching [2] entre deux images, une image (template) étant une partie de l'image dans laquelle on cherche à la localiser. Dans notre cas le template serait notre planche et l'image celle du billon. Une des méthodes de matching proposée CCOEFF définie par :

$$R(x,y) = \frac{\sum_{x',y'}(T'(x',y')\cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x',y+y')^2}}$$

Avec

$T'(x',y') = T(x',y') - 1/(w\cdot h)\cdot \sum_{x'',y''} T(x'',y'')$

$I'(x+x',y+y') = I(x+x',y+y') - 1/(w\cdot h)\cdot \sum_{x'',y''} I(x+x'',y+y'')$

réalise la tâche que l'on souhaite, utiliser la moyenne de la couleur dans la région pour réduire les différence entre les deux images.

Dans la formule précédente on a T' qui correspond à la couleur dans l'image de la planche auquel on soustrait la moyenne de la couleur au sein de l'image complète de la planche. De même pour I' qui correspond à la couleur dans l'image du billon auquel on soustrait la couleur moyenne de l'image du billon dans une zone de la taille de l'image de la planche. On fait ensuite le produit des deux vecteur couleurs obtenus et on somme sur tous les pixels sur la région de la taille de la planche. On normalise enfin par les couleurs obtenues sur chaque pixel.

On obtient donc pour chaque pixel une comparaison entre un pixel de la planche et du billon auquel on a soustrait une moyenne de la région, si on augmente la luminosité d'une planche extraite du billon on a alors une moyenne plus grande sur la planche et au moment de la comparaison on aura les couleurs qui devraient être les mêmes après soustraction de la moyenne donc deux couleurs proches devraient nous donner un score positif et donc augmenter le score de la somme des comparaisons de pixels. Si par contre deux couleurs sont différentes alors on aura soit des scores négatifs si une couleur est positive et l'autre négative soit des scores bas. Notre meilleure

2. https ://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/template_matching/template_matching.html

position de matching sera donc la valeur la plus élevée.

On obtient le résultat suivant pour le matching de la planche dans le billon, on remarque que cette fois la position choisie est la bonne mais aussi que la confiance sur la position est bien meilleure.

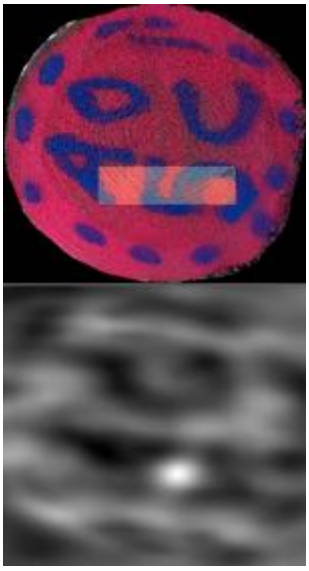


FIGURE 23 – Matching avec la methode CCOEFF

Pour améliorer encore les performances de matching on peut changer l'espace de couleur utilisé. Plutôt que d'utiliser l'espace RGB, on va utiliser l'espace Lab composé de 3 composantes dont L correspondant à la clarté de la couleur et a et b correspondant à l'écart de la couleur par rapport à celle d'une surface grise. Si on utilise seulement les composantes a et b on ne prend alors plus en compte la clarté de l'image. On remarque bien dans la Figure 24 que la planche et le billon ont l'air d'être plus proche en terme de couleur dans l'espace Lab privé de L que dans l'espace RGB. Ceci améliore le score global des bons matchings de planche dans le billon.

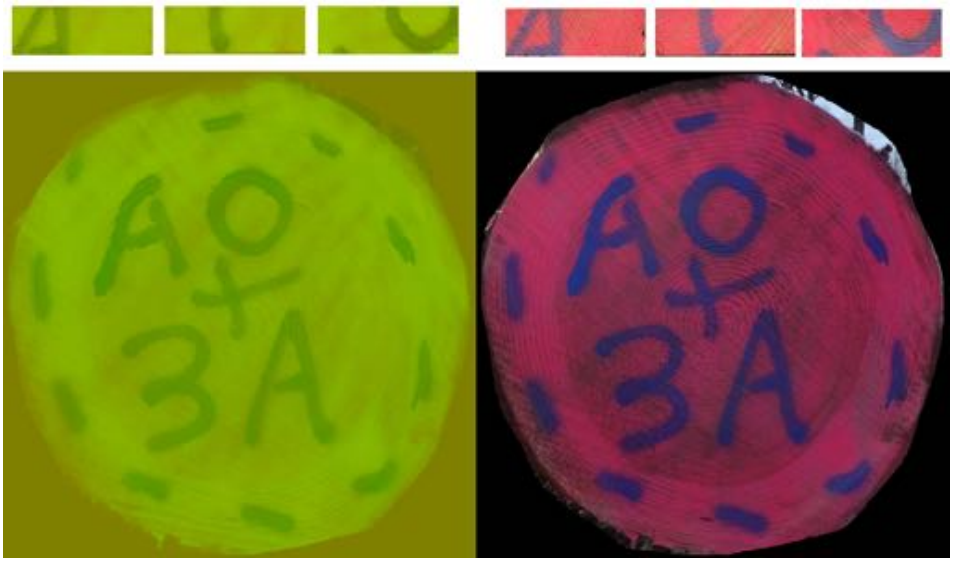
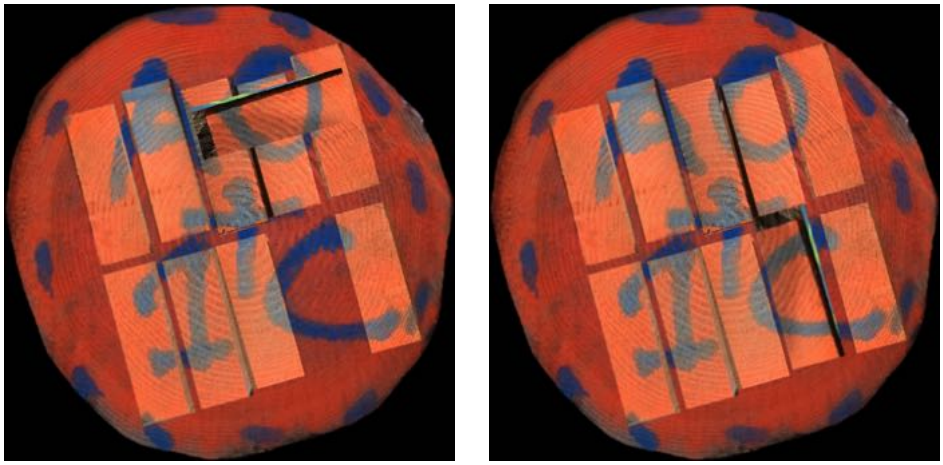


FIGURE 24 – Comparaison Lab (gauche) et RGB (droite) avec les planches

Il reste cependant un problème lors des matching, en effet on essaye de récupérer le meilleur angle pour avoir à choisir entre celui-ci et son opposé mais dans certains billons comme celui de la Figure 25 on a des planches qui ne se trouvent pas dans un de ces deux sens. On doit aussi permettre à notre algorithme de poser des planches à l'angle orthogonal au meilleur angle et non qu'à l'angle opposé. On obtient alors les bons matching dans les cas où l'on avait ces planches à un angle orthogonal au meilleur angle.

FIGURE 25 – Billon avec des planches orthogonale au meilleur angle

Dans certains cas comme celui présenté dans la Figure 26a, on a une planche qui avait un bon matching avec les deux seuls angles possibles qui perdent leurs positions. Cependant ces cas sont plutôt faciles à détecter, on a une superposition de la planche or on ne peut pas avoir de superposition dans la réalité comme on découpe les planches dans le billon. On prend alors, pour ces planches qui ont un fort taux de superposition par rapport aux autres planches et un moins bon score, le prochain matching dans un des angles autorisés qui à un meilleur score et ne créer pas de nouvelles superpositions. On obtient donc de nouveau un bon matching pour cette planche.



(a) (b)

FIGURE 26 – Billon avec un planche mal tournée (a) Planche mal tournée (b) Planche replacée

3.3 SURF

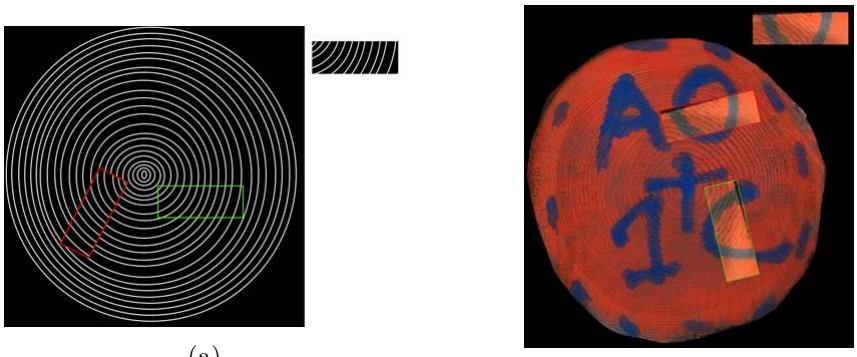
Dans cette section nous allons utiliser l'algorithme de détection de feature SURF (Speeded-Up Robust Features) présenté dans l'article [6]. Il se base sur une mesure utilisant une approximation de matrice Hessienne pour la détection de feature, le calcul de l'approximation est très efficace grâce à l'utilisation d'images intégrales pour faire une convolution sur l'image avec des filtres. L'algorithme est aussi résistant à la rotation et à l'échelle des images. Dans notre cas on cherchera des features dans notre billon et dans les planches pour essayer de les associer.

On regarde dans un premier temps quels sont les features trouvées dans le billon et dans la planche. On observe dans la Figure 27b que les features pour le billon (représenté par des cercles de rayon proportionnel à la valeur de l'approximation de la matrice Hessienne pour ce point) sont très présentes au niveau des cernes du billon. Le matching se ferait donc principalement sur les cernes du billon ou utiliser uniquement les informations sur les cernes du billon n'est pas suffisant, en effet on peut créer un exemple comme la Figure 28a où l'on a schématisé les cernes d'un billon puis extrait une boîte (verte) pour représenter une planche. En effectuant une rotation de notre boîte autour de la moelle du billon on obtient une autre position (en rouge). Dans ce cas même pour une personne humaine il est difficile de dire que la position en rouge n'est pas correcte si on ne connaît pas la position de la

“planche” en vert. Même si ce cas théorique semble tordu on peut facilement le reproduire avec des images de billon et une planche comme dans la Figure 28b.

(a) (b)

FIGURE 27 – Features détectées dans le billon (a) Le billon (b) Représentation des features



(a) (b)

FIGURE 28 – Matching basé sur les cernes (a) exemple théorique (b) exemple réel

4 Présentation des résultats

Finalement on propose une solution pour chaque partie du sujet. On a pour la segmentation des planches la méthode de détection des lignes puis le travail plus local pour trouver les planches et affiner le placement. Pour la reconstruction du billon à partir des planches et des billons on a la méthode de template matching suivie de test de superposition et d'échelle.

On ne dispose pas de vérité terrain pour la segmentation permettant de faire une évaluation quantitative, on peut tout de même faire une évaluation qualitative des planches obtenues. Parmi les remarques sur la détection on a :

- Extraction de "planches" qui n'en sont pas, dans les lignes supérieures des pile de planches on n'a pas toujours des planches dans toute la longueur de la ligne.
- Extraction de "planches" qui n'en sont pas, avec la droite en haut de le derniere ligne de planche on détecte une dernière ligne au-dessus, on lance une détection des lignes verticales et un remplissage pour extraire des planches alors qu'il ne devrait pas en avoir.
- Certaines frontières entre deux planches sont facilement identifiables mais la détection de bord ne permet pas de les trouver avec la détection des lignes verticales, on se retrouve donc avec des répliques moins correctes que si l'on avait trouvé ces bords.
- Des planches sont coupées soit dans la largeur soit dans la longueur, on a aussi des morceaux d'autres planche dans la "planche".

On récupère quand même dans toutes les piles de planches les plus grandes majorités des planches, seules quelques planches ne sont pas détectées, on retrouve 343 planches sur les 346 planches, les 3 non trouvées sont coupée en deux (Figure 31 ligne 1 colonne 8 et 9, Figure 33 ligne 1 colonne 10 et 11).

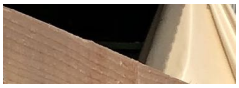
Au final on récupère :

- 343/346 planches segmentées (99.1% des planches)
- 56 non planches (14% des détections)
- 198 planches sans débordements (58% des planches) (50% des détections)
- 145 planches avec débordements (42% des planches) (36% des détections)

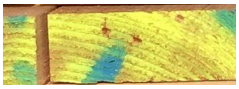
Ce qui est plutôt un résultat très encouragent.

Le temps moyen de traitement et d'extraction des planches est de 6 secondes.

(a)



(b)



(c)

(d)

FIGURE 29 – Exemple de planches segmentées (a) Fausse "planche" (b) Entre planche de la ligne du haut (c) Mauvais bord (d) Planche coupée

On dispose cependant d'une vérité-terrain permettant de quantifier la qualité des reconstructions des billons. On possède un total de 32 billons, parmi ces billons il y en a un que l'on n'utilise pas car ses planches dans la pile de planches sont de trop mauvaises qualités.

On réalise donc nos tests sur les 31 billons restants. En comptabilisant à la main les bons placements de planches dans les billons on obtient un taux de réussite des placements de 97.6% (330/338 planches bien placées).

Parmi les 8 planches mal placées on note que 2 planches ont une ombre bien marquée due à une des planches au-dessus d'elle dans la pile de planches, 5 planches n'ont aucune marque de couleur et la dernière planche possède une marque colorée assez remarquable de courbe mais se positionne mal, elle devrait être sur un O mais se place sur un C puis sur un 2 pour l'angle et l'angle opposé trouvé puis sur un D et enfin sur le C à nouveau pour les deux angles orthogonaux.

Pour les mesures par rapport à la vérité-terrain on utilise la précision, le rappel et la F-mesure. La précision correspond a : $\frac{vp}{vp+fp}$. Le rappel correspond a : $\frac{vp}{vp+fn}$. Avec $vp = vrai\,positif$, $fp = faux\,positif$ et $fn = faux\,ngatif$. La F-mesure correspond à la moyenne harmonique de la précision et du rappel : $2 \times \frac{precision \times rappel}{precision + rappel}$.

On a utilisé pour les tests les mesures manuelles de taille de mire dans les images de planches et du billon, et on utilise un pas pour les angles de rotation testé de 15°. Les planches utilisées pour les matching sont les planches segmentées à la main pour mesurer les capacités de matching sans prendre en compte les erreurs de segmentation.

On obtient finalement en moyenne une précision de 0.86, un rappel de 0.87 pour une F-mesure de 0.87 et la différence sur les angles trouvés est en moyenne de 4.7°.

La précision ne dépasse pas les 0.94, même si on place les planches du mieux possible on a toujours une petite erreur sur l'angle pour chaque

planche, elles n'ont pas toutes exactement le même angle mais à quelques degrés près l'angle est identique et on considère seulement 1 angle dans l'algorithme.

On a quand même un bon score de F-mesure, la précision et le rappel sont proches même si le rappel est un peu plus haut que la précision. On peut expliquer cette différence par les problèmes d'échelle, même dans certains billons ayant des bons matching on a quand même quelques superpositions sur les bords de planches si elles sont trop grandes après passage à l'échelle dans le billon.

Le temps d'exécution moyen est de 3.9 secs par planche par billon, on constate quand même un écart entre les billons ou on fait un test de changement d'échelle. Pour les billons sans changements d'échelle le temps moyen est de 2.9 secs par planche par billon et quand il y a changement d'échelle on a une moyenne de 6.1 secs ce qui correspond à plus du double de temps de calcul.

Les résultats obtenus sont disponibles dans les annexes (Segmentation : Figure 31,32,33,34,35 , Matching : Figure 30).

5 Conclusion

L'objectif du projet était de retrouver la position des planches de bois à partir de l'image du billon avant le débit des planches et une image contenant des planches obtenues après le débit. On a divisé le sujet en deux parties, la segmentation des planches dans les images où l'on a les résultats de débit des planches d'un billon et la reconstruction du billon.

Deux méthodes pour résoudre ces parties ont été proposées ainsi que d'autres pistes ayant été explorées durant le stage.

Les résultats obtenus pour la segmentation sont plutôt bons, la méthode utilisée semble être une bonne approche mais sa dépendance à une détection de bord lui fait défaut. La méthode de récupération des bords utilisée est très basique, utiliser une autre méthode améliorerait grandement le résultat.

Pour la reconstitution de billon les résultats sont consistants, on réussit dans la plupart des billons à obtenir toutes les planches à leurs positions. On peut quand même proposer des pistes d'améliorations, notamment au niveau du temps de calcul, actuellement on traite chaque angle et chaque planche les unes après les autres mais une parallélisation améliorerait grandement le temps de calcul. Dans le cas des planches sans marques de couleur utiliser les informations des cernes en plus des positions des autres planches aideraient à les localiser plus facilement.

Parmis les autres améliorations possibles non abordées on peut proposer :

— D'utiliser une méthode automatique pour mesurer les mires et pour segmenter les billons réduirait les interactions nécessaires avec l'homme dans le pipeline.

— D'automatisatiser la sélection des planches à placer dans le billon.

6 Références

Références

[1] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6) :679–698, 1986.

[2] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, pages 839–846, 1998.

[3] Richard O. Duda and Peter E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1) :11–15, January 1972.

[4] Serge Beucher and Fernand Meyer. *The Morphological Approach to Segmentation : The Watershed Transformation*, volume Vol. 34, page 433–481. 01 1993.

[5] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1 : Statistics*, pages 281–297, Berkeley, Calif., 1967. University of California Press.

[6] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf : Speeded up robust features. volume 3951, pages 404–417, 07 2006.

7 Annexes

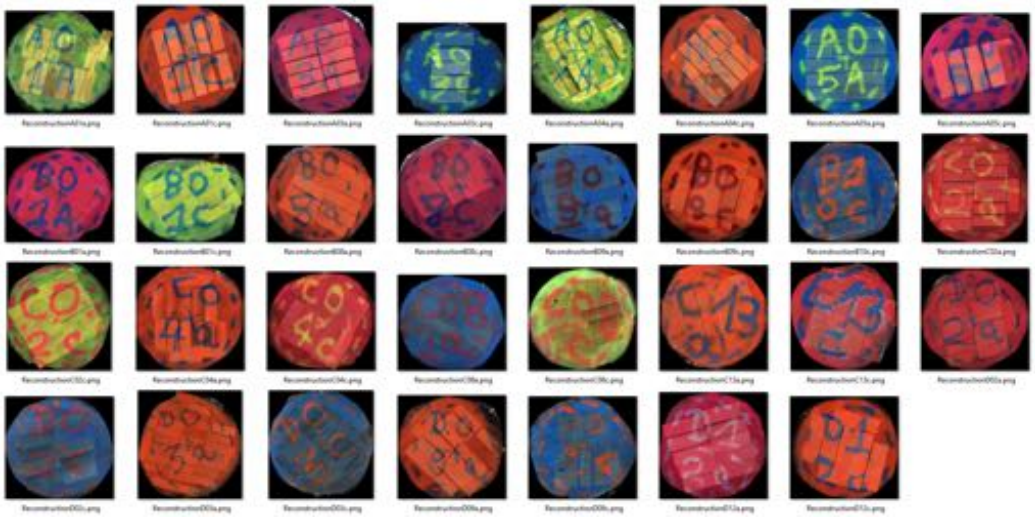


FIGURE 30 – Résultats des matching

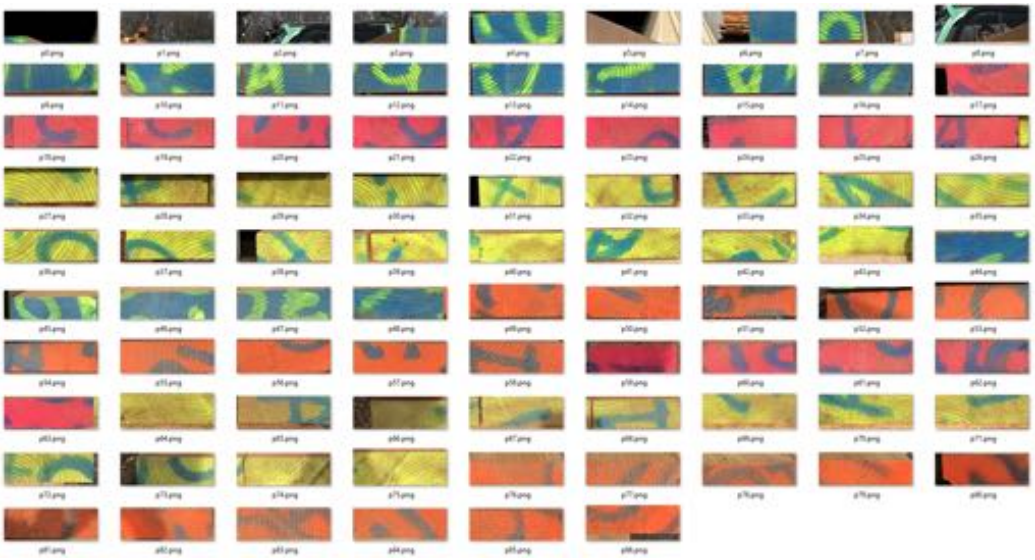


FIGURE 31 – Résultats segmentation de la pile 1

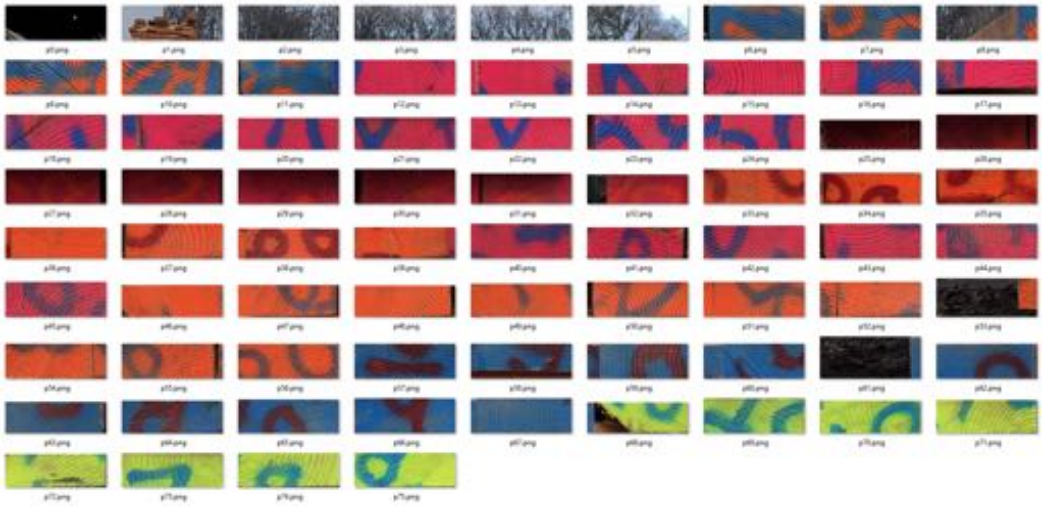


FIGURE 32 – Résultats segmentation de la pile 2

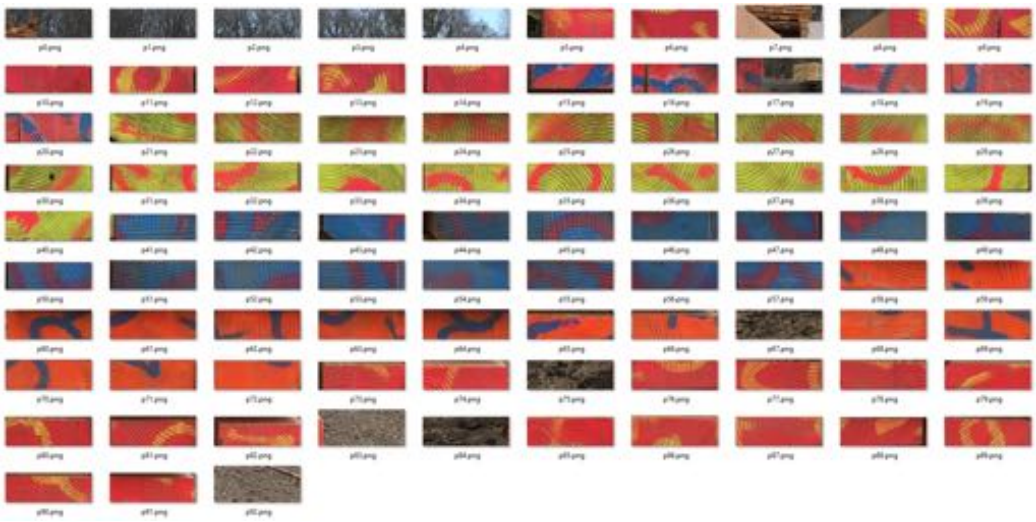


FIGURE 33 – Résultats segmentation de la pile 3

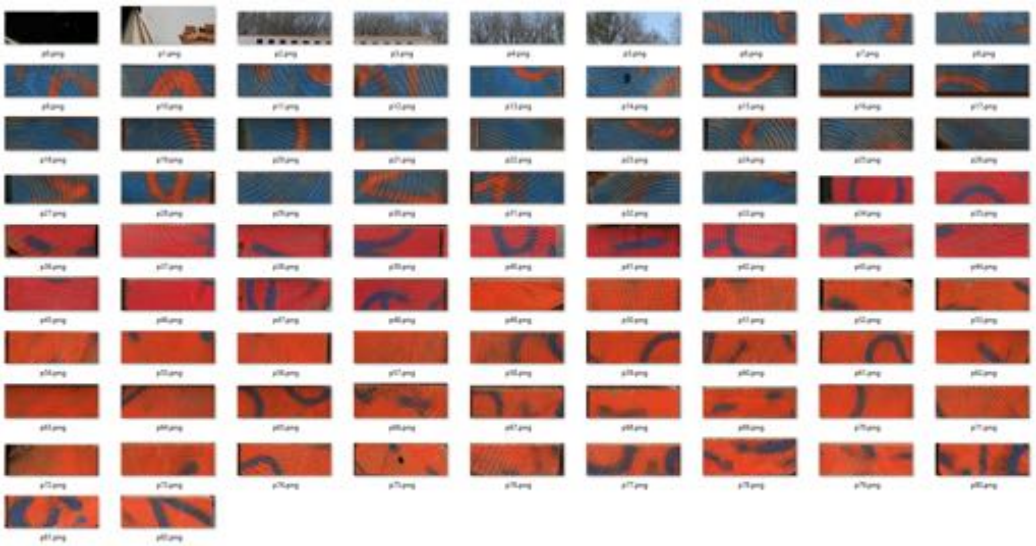


FIGURE 34 – Résultats segmentation de la pile 4

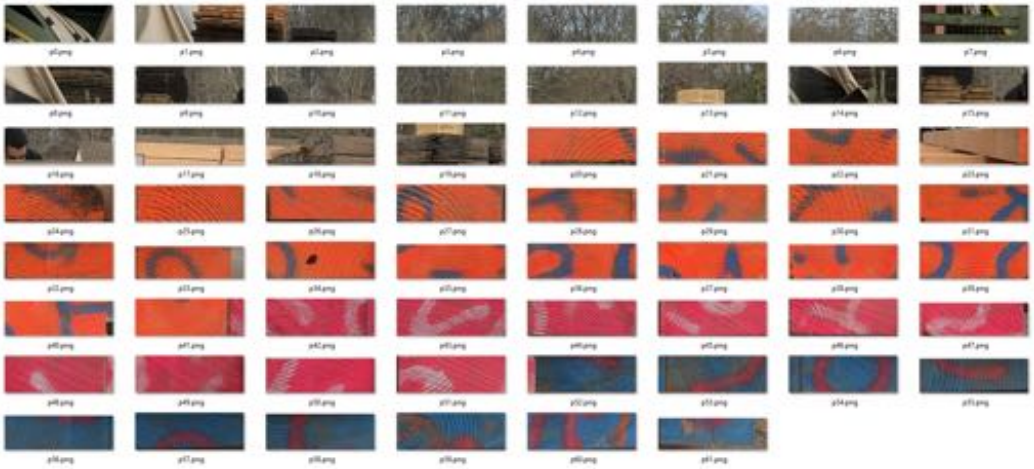


FIGURE 35 – Résultats segmentation de la pile 5