
Extraction et analyse d'image

Projet d'étude et de Développement

Manson Tomy
Mestreau Nicolas
Ridel Fabien
Wen Jun

Client :
Jérémy Laviole

Table des matières

1	Introduction	3
1.1	Présentation	3
2	Objectifs	4
2.1	Cahier des charges	4
2.1.1	Pré-traitement	4
2.1.2	Détection des évolutions d'un dessin	5
2.1.3	Reconnaissance des formes dessinées	5
2.2	Planning	6
3	Travail accompli	8
3.1	Architecture logicielle	8
3.2	Pré-traitement	8
3.2.1	segmentation	8
3.3	Traitemet	12
3.3.1	Détection des évolutions d'un dessin	12
3.3.2	Reconnaissance de formes dessinées	13
4	Tests et résultats	18
4.1	Presentation des logiciels de test	18
4.1.1	Reconnaissance de formes simples dans un dessin . . .	18
4.1.2	Détection des évolutions d'un dessin	19
4.2	Résultats obtenus	19
5	Conclusion	20

1 Introduction

1.1 Présentation

L'objectif de notre projet d'étude et de développement est de produire une bibliothèque d'extraction et d'analyse d'images à partir d'un flux vidéo. Cette bibliothèque sera utilisée pour des applications de vision par ordinateur.

L'extraction d'images consiste à récupérer un espace de dessin dans la vidéo. cet espace de dessin peut être une feuille blanche, entourée par une planche de tags. Au début du projet, une bibliothèque d'extraction (`ARToolkit`) nous a été fournie. Nous devions donc nous concentrer sur les analyses des images obtenues.

Les analyses demandées par ce projet sont de deux types :

- Une analyse rapide, sur une image de faible qualité. Le but de cette analyse est de retrouver des formes simples (voir les templates ci-dessous) dans le dessin.
- Une analyse sur des images de haute qualité. Cette analyse a pour but de détecter les nouveautés dans le dessin.

Des applications utilisant la bibliothèque sont également à développer, notamment une application de détection de nouveautés dans un dessin, et une application de reconnaissance de formes simples.

2 Objectifs

2.1 Cahier des charges

Dans le but de développer la bibliothèque, nous avons établi un cahier des charges présentant et définissant les trois ensembles de fonctionnalités qui la constitue :

- Pré-traitement des images
- Détection des évolutions d'un dessin
- Reconnaissance des formes dessinées

2.1.1 Pré-traitement

Tout traitement sur une image nécessite d'effectuer un certain nombre de pré-traitements afin d'améliorer la qualité de cette image. Cette étape a pour but de favoriser l'obtention de bons résultats lors de la phase de traitement.

La bibliothèque devra permettre d'appliquer aisément un certain nombre de filtres, notamment :

Filtre morphologique Optimisation des méthodes de template matching ;

Binarisation Permettre de trouver les différences qui existent entre deux images ;

Correction de couleur Balance des blancs, permet d'annuler la coloration globale de l'image, dépendante de l'éclairage de la scène (lumière du jour, lumière artificielle, etc ...);

Filtrage par couleur Récupérer uniquement les formes dessinées avec une couleur précise.

La bibliothèque devra permettre de développer de nouveaux filtres. Tous les filtres devront pouvoir être utilisés de la même manière afin d'en simplifier l'usage, ils devront avoir une interface commune.

2.1.2 Détection des évolutions d'un dessin

La première fonctionnalité de la bibliothèque est de permettre la récupération de plus haute qualité possible d'un dessin à différents moments. Les moments de prise de vue du dessin seront considérés comme idéaux, pas de mains, ni d'objets dans le champs de prise de vue.

On cherche ici à permettre le développement d'applications permettant d'observer les moments clés de la création d'un dessin sans perturber le travail de l'artiste, c'est à dire, éviter de scanner la feuille.

Performance¹

La bibliothèque devra permettre l'analyse d'image à une cadence de 2 à 3 images par seconde. Le but ici n'est pas la performance mais bien la qualité de l'analyse. Les images à traiter seront acquises en haute définition, un minimum de 1080 lignes de 1920 pixels, ce qui correspond à la résolution d'un flux vidéo HDTV 1080p.

2.1.3 Reconnaissance des formes dessinées

La deuxième fonctionnalité de la bibliothèque est de reconnaître, dans un flux vidéo, des dessins ou pictogrammes pré-enregistrés.

Différentes méthodes pourront être développées selon que les formes à reconnaître dans l'image sont simples (e.g. : un carré, un cercle) ou complexe (e.g. : un visage).

On utilisera par exemple :

- Template Matching : Mise en correspondance d'une image avec une image de référence pour la détection de formes simples.
- Feature detection : Utilisation de caractéristiques invariantes dans l'image pour la reconnaissance de formes complexes (e.g. : descripteurs SURF).

Performance¹

La bibliothèque devra permettre l'analyse d'image à une cadence de 30 à 60 images par seconde. Le but ici est d'effectuer de l'analyse sur un flux vidéo temps réel, de basse qualité. Les images à traiter seront acquises en basse qualité, on prendra comme base des vidéos de résolution 480 lignes de 640 pixels, ce qui correspond au format VGA.

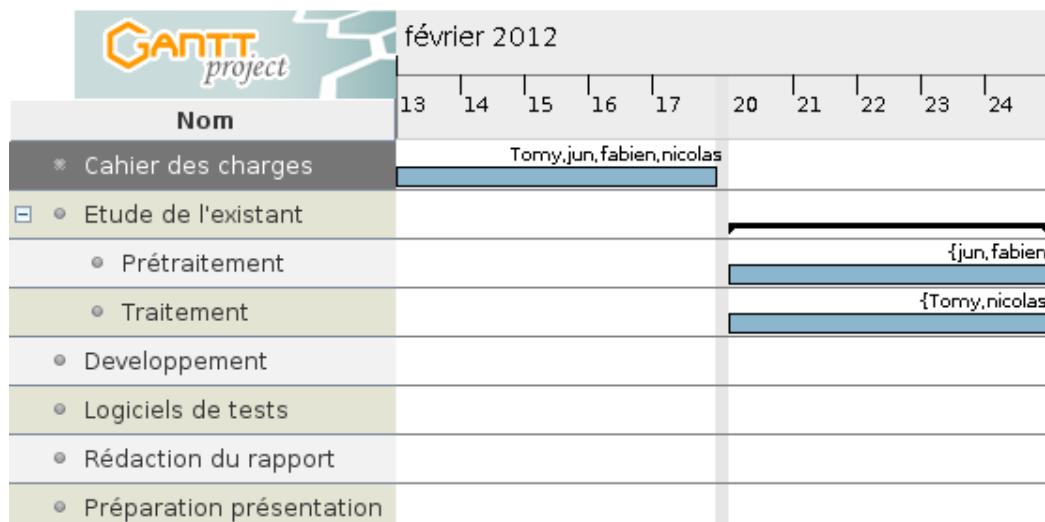
1. Les problématiques de performance sont secondaires, l'objectif principal est de développer une bibliothèque qui fonctionne.

2.2 Planning

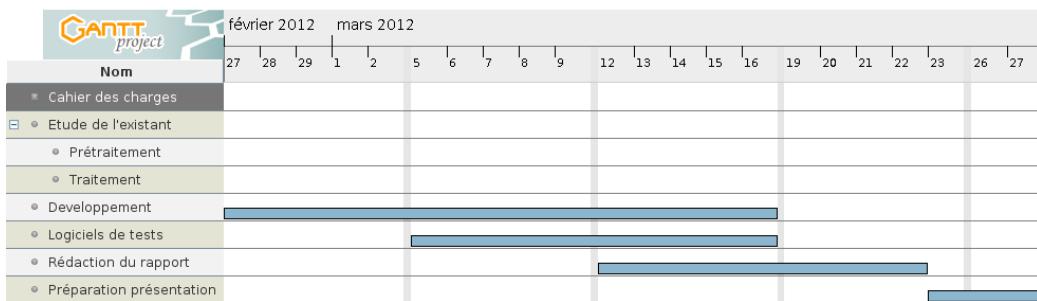
Le projet s'est déroulé suivant un planning se découplant en deux phases principales : une phase de préparation et d'organisation du projet et une phase de développement, de tests et de fin de projet.

La phase de préparation et d'organisation, présentée dans le diagramme de Gantt ci-après, est divisée en deux tâches :

- Établissement du cahier des charges par l'ensemble du groupe
- Étude de l'existant. Cette dernière étant subdivisée en deux sous-tâches :
 - Étude des méthodes de prétraitements d'image, réalisé par Fabien Ridel et Jun Wen ;
 - Étude des méthodes d'extraction de nouveautés et de reconnaissance de forme dans l'image, réalisé par Tomy Manson et Nicolas Mestreau.



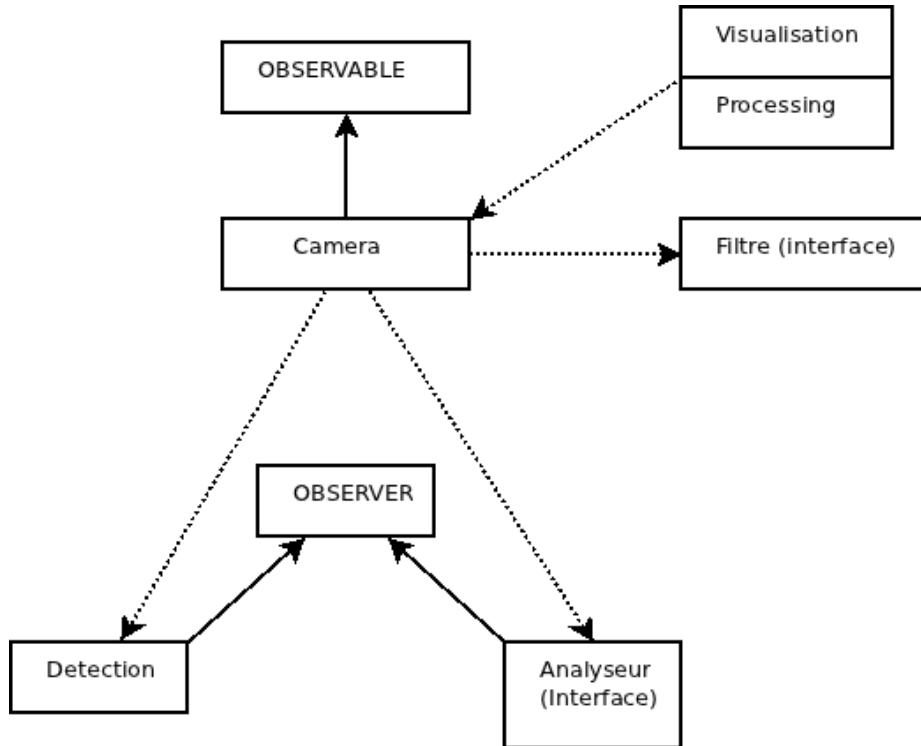
La phase suivante, présentée dans le diagramme de Gantt ci-après, est la phase de développement de la bibliothèque, de programme de démonstration et de tests. Nous incluons dans cette phase la rédaction du rapport et la préparation de la présentation, qui constituent la fin de projet.



Planning de développement et de fin de projet

3 Travail accompli

3.1 Architecture logicielle



3.2 Pré-traitement

3.2.1 segmentation

Les images à traiter sont bruitées, ne bénéficient pas de bonnes conditions d'éclairage et le support peut être de mauvaise qualité (pli de la feuille). Avant de commencer la détection de nouveautés ou de forme, il est donc nécessaire de segmenter notre image afin de détecter les objets sur l'image. Etant donné le type de support, un document avec des dessins noirs sur fond blanc, nous avons focaliser notre travail sur la binarisation ainsi que sur la détection de contours.

Binarisation par seuillage global

La binarisation a pour but de ségmenter l'image en 2 classes, le fond et l'image. Il existe différentes techniques pour binariser une image.

Nous avons dans un premier temps un algorithme naïf de binarisation par seuillage global, on choisit la classe du pixel à partir d'un seuil.

$$\forall i, j \in \mathbf{M} * \mathbf{N} \quad I(i, j) = \begin{cases} 1 & \text{si } f(i, j) > S, \\ 0 & \text{sinon} \end{cases} \quad (1)$$

Avec :

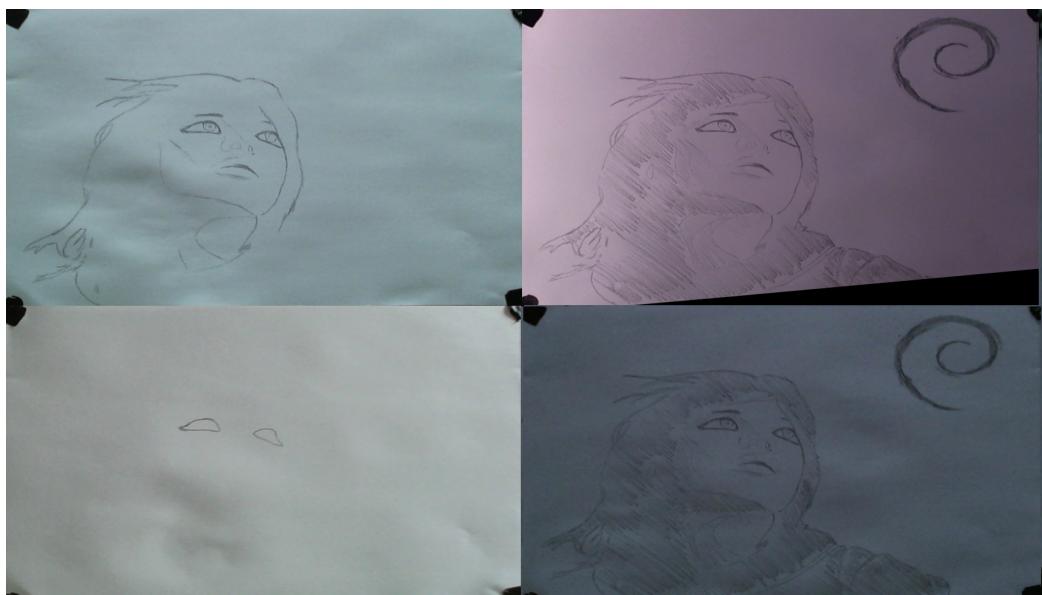
$\mathbf{M} * \mathbf{N}$ nombre de colonnes et de lignes de l'image ;

I image binarisée ;

S seuil de binarisation.

f valeur fonction de l'image d'origine ;

Le problème est que l'on utilise des documents dégradés (mauvaise illumination et papier marqué) donc bien souvent il n'existe pas de seuil global.



Différence d'éclairage



Résultat seuillage global

Binarisatoin par détection de contour

Il existe de nombreuses méthodes de détection de contour, nous avons appliqué le filtre de sobel. Il s'agit d'appliquer deux matrices de convolution G_x G_y à l'image qui calcule la dérivé verticale et horizontal.

$$G_x = \begin{vmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{vmatrix}.$$

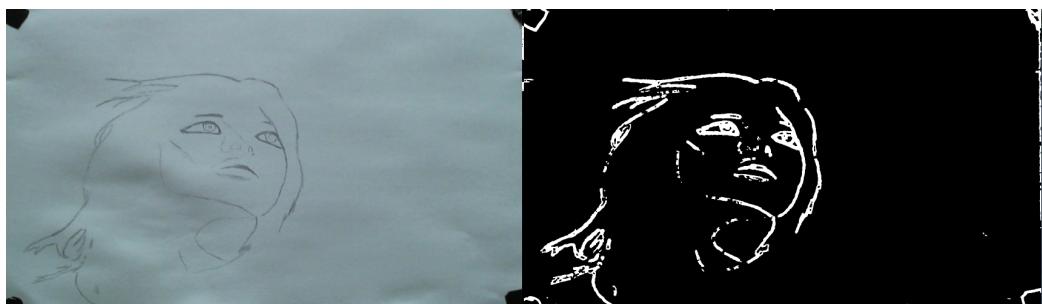
$$G_y = \begin{vmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{vmatrix}.$$

Puis en chaque points de calculer une approximation de la norme du gradient.

$$G = \sqrt{G_x^2 - G_y^2}$$

Les résultats étaient bien sur efficace sur les traits fin mais moins bon dans les zones homogènes (coloriées). Toutefois il est possible d'obtenir un masque des zones contenant un objet, en y combinant des filtres morphologiques et un seuillage globale. Voici les étapes de notre binarisation.

- Flou Gaussian. (lissage pour atténuer le bruit)
- Sobel
- Fermeture morphologique (pour éliminer le bruit produit par sobel)
- Ouverture pour combler les trous de l'objet
- Binarisation seuillage global



Résultat masque par binarisation par détection de contour

Binarisatoin par détection de contour

L'étude de , compare et classe les différents algorithmes de binarisation

emet un classement des algorithmes pour les images dégradés. La méthode de kittler se basant sur le clustering ... nous avons abandonné cette méthode car les résultats obtenus étaient insuffisants.

et les méthodes par seuillage local, son principe est d'utiliser une étude localisée autour du pixel pour déterminer quel seuil utiliser. Pour réaliser cette étude locale, les techniques utilisent une fenêtre d'étude centrée sur le pixel à étudier. Cette fenêtre peut avoir différentes tailles, de nombreuses méthodes ont été proposées bernsen, Niblack , Sauvola.

Sauvola est indiqué comme obtenant les meilleurs résultats

$$S(i, j) = \mu(i, j) + \kappa.((\sigma(i, j)/R) - 1)) \quad (2)$$

- Avec : - $S(i, j)$: seuil à appliquer pour le point i, j ;
 - $\sigma(i, j)$: valeur de l'écart type dans une fenêtre centrée en i, j de taille $N * M$;
 - $\mu(i, j)$: valeur moyenne des niveaux de gris dans la même fenêtre ;
 - κ : constante fixée le plus généralement à 0, 2 ;
 - R : constante permettant d'ajuster la dynamique de l'écart type, généralement 128. - N et M appartenant à N .

La méthode de Sauvola calcule le seuil de binarisation en fonction de la moyenne et de l'écart type des pixels ajuster par la constante R , le rapport entre les deux est défini par la constante κ . Sauvola conseille un $\kappa \in [0.2; 0.5[$, mais ce type de paramètre est fixé pour binariser des textes, or nos documents sont des dessins effectués au crayon sur papier et peuvent avoir des traits très fins. Dans son article , fixe κ à 0.05, ce qui augmente l'influence de l'écart type et permet d'obtenir de meilleurs résultats, notamment lorsqu'il y a un faible contraste comme dans notre cas.

La taille de la fenêtre est fixée en fonction de la taille de l'image, nous ne prenons pas la même taille de fenêtre pour l'image en haute résolution que pour le flux vidéo en basse résolution.

Optimisation.

Les images intégrales sont une représentation sous la forme d'une image, de même taille que l'image d'origine, qui en chacun de ses points contient la somme des pixels situés au-dessus et à gauche de ce point. Plus formellement, l'image intégrale ii est définie à partir de l'image i par :

$$ii(x, y) = \sum_{\substack{x' \leq x \\ y' \leq y}} i(x', y') \quad (3)$$

résumé des différentes techniques

Seuillage global

detection de contour + filtre morphologique

Seuillage local Opencv moyenne et gaussian
seuillage locale Sauvola

3.3 Traitement

3.3.1 Détection des évolutions d'un dessin

Un premier objectif du projet était d'extraire les moments clés de la réalisation d'un dessin.

Pré-traitement

La capture des images étant faite en haute qualité, l'étape de pré-traitement n'a pas une grande influence sur les résultats obtenus. Seul une correction de couleur (balance des blancs) est réalisé afin de corriger le changement de coloration générale de l'image acquise. On pourrait penser que cette correction pourrait être effectué sur la camera directement, ce qui y est généralement fait, mais le processus de réalisation d'un dessin peut être long. De ce fait l'éclairage la coloration due à l'éclairage de la scène peut varier¹. Dans la plupart de cas, cette correction doit être effectué sur chaque image acquise.

Extraction des nouveautés

Une fois cette petite étape de pré-traitement effectuée nous pouvons commencer à extraire les nouveautés.

Dans un premier temps, nous avons effectué de simple différences entre les images mais cela n'a pas été concluant. En effet, l'acquisition de deux images à deux instant différents induisent des différences de positions du dessin ainsi que de légère variation d'éclairage. Ainsi, comme on le voit sur les images ci-dessous, le résultat obtenu n'est pas celui attendu.



1. Cette dérive d'éclairage ne s'observe pas dans un environnement contrôlé, tels qu'un studio



Image du bas : résultat de la simple différence en les images du haut

Nous avons donc, dans un second temps, extrait un masque de nouveauté, c'est-à-dire, une image binaire dans laquelle les zones ayant changées sont blanches, toutes les autres étant noire. Ce masque nous permet de récupérer, dans l'image original, uniquement les nouveautés du dessin.



Masque de différence obtenu à partir des deux images précédentes

Pour obtenir ce masque, nous binarisons les deux images à partir desquelles nous voulons extraire les nouveautés. Puis nous faisons la différence des deux images obtenues, les images étant binarisé on obtient directement un masque de nouveauté.

3.3.2 Reconnaissance de formes dessinées

Un des objectifs de notre projet était de détecter des formes simples dans une vidéo. Ci-dessous sont présentés deux exemples de formes simples.



Exemples de templates

Afin de reconnaître ces pictogrammes dans la vidéo, nous avons mis en place un système de détecteur.

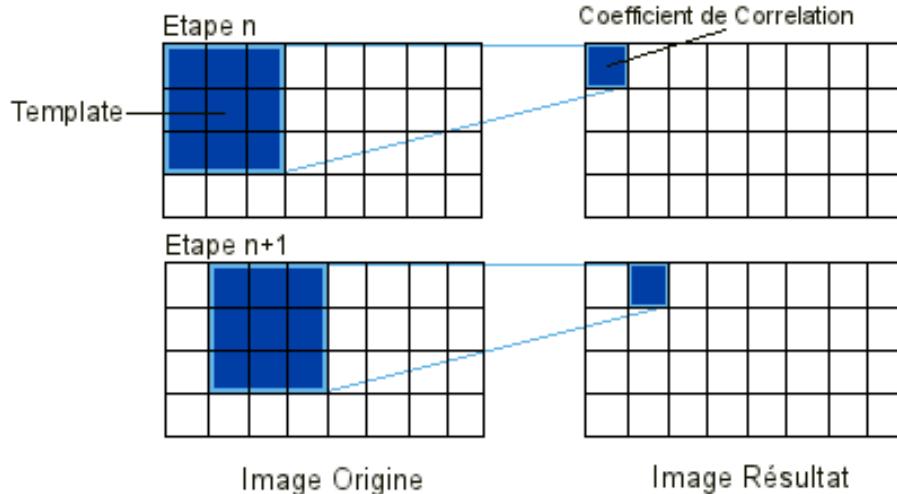
Nous avons implémenté deux détecteurs différents. Le premier s'appuie sur du template matching pour reconnaître une forme. Le second utilise les descripteurs SURF.

Pour faire ainsi, nous avons mis en place une classe abstraite `Detector` qui présente les fonctions de base d'un détecteur, à savoir ajouter, récupérer ou supprimer un template, et définit la fonction permettant de lancer la détection. Chacun de nos détecteurs (`MatchingDetector` et `SurfDetector`) doit donc implémenter cette fonction de détection.

Template matching [?] [?]

Notre premier détecteur est le `MatchingDetector`, qui utilise l'algorithme de template matching.

L'algorithme de template matching consiste à faire glisser un template sur la surface de l'image, et de calculer un score de "corrélation" entre le template et la zone qu'il recouvre.



Principe du Template Matching

Pour effectuer cette procédure, nous parcourons la liste des templates du détecteur. Pour chacun de ces templates, nous avons utilisé la fonction

`cvMatchTemplate` d'OpenCV pour calculer une image de corrélation entre le template et l'image tirée de la vidéo, qui montre les endroits où le template a le plus correspondu à l'image. Après normalisation, nous parcourons cette image pour récupérer le minimum (ou maximum suivant la méthode passée en paramètre à `cvMatchTemplate`), ce minimum/maximum est l'endroit de meilleur matching. Nous stockons alors le quadruplet (x, y, w, h) avec :

- x,y** Les coordonnées du point de meilleur matching correspondant au coin supérieur gauche de la zone rectangulaire représentant l'objet détecté.
- w,h** Les dimensions du template, correspondant aux dimensions de la zone rectangulaire représentant l'objet détecté.

Une fois la position du meilleur matching récupérée, nous retirons de l'image source la zone rectangulaire décrite par le quadruplet (x, y, w, h) . Cette étape permet d'éviter que deux templates trop proches détectent le même objet alors que deux objets sont valides (exemple : Sur un dessin, deux bonhommes sont dessinés. Avec deux templates de bonhomme, il est possible que les deux matchings trouvent comme meilleur résultat le premier bonhomme et ne détectent pas le second. Notre étape de suppression permet de faire en sorte qu'une fois que le premier matching a détecté le premier bonhomme, celui-ci n'est plus pris en compte pour les détections des autres templates. On augmente ainsi les chances de détecter tous les objets de l'image.)

Une fois tous les templates parcourus, nous retournons l'ensemble des quadruplets (x, y, w, h) correspondant aux détections.

Un des défauts de notre `MatchingDetector` est que le retour de `cvMatchTemplate` ne nous permet pas de calculer un seuil de confiance pour accepter ou rejeter le résultat du matching. On peut récupérer l'endroit où le template a le plus correspondu à l'image, mais on ne sait pas si il a matché à 99% où à 10%.

Ce défaut, couplé à la normalisation du résultat de `cvMatchTemplate` (première tentative, infructueuse, de déterminer un seuil de confiance) entraîne un faux positif lorsque le template ne matche pas un des objets de l'image.

Le problème est double : pour le moment, nous avons un template = un résultat. On a donc forcément des faux positifs lorsque le nombre de templates d'un type d'objet dépasse le nombre d'objets de ce type du dessin, ou quand le template ne matche pas à un des objets du dessin. Nous avons

également des faux négatifs, donc des objets qui ne sont pas détectés, lorsque le nombre de templates d'un type d'objet est inférieur au nombre d'objets de ce type dans le dessin.

Une solution au problème serait d'arriver à calculer un indice de confiance du matching. Si cet indice de confiance ne dépasse pas un seuil à définir, on ne prend pas en compte le résultat du matching pour ce template.

Cette solution permet de boucler sur un template jusqu'à ce que l'indice de confiance calculé ne passe plus le seuil. Cela permet d'avoir un template par type d'objet, et non un template = un résultat positif comme à présent. Avec peu de templates de bonhomme, on pourrait détecter un grand nombre de bonshommes dans le dessin. On réduirait ainsi le problème de faux négatifs lorsque le nombre de templates d'un type d'objet est inférieur au nombre d'objets de ce type. Et comme on ne prend pas en compte les matchings qui ne passent pas le seuil, on réduit également le nombre de faux positifs.

SURF : Speeded Up Robust Features [?]

Le second type de détecteur que nous avons implémenté se base sur les descripteurs SURF.

L'algorithme SURF permet de déterminer des points d'intérêt dans une image, ainsi que de détecter ces points dans une autre image. L'intérêt de SURF est qu'il n'est ni sensible à la rotation ni à la mise à l'échelle.

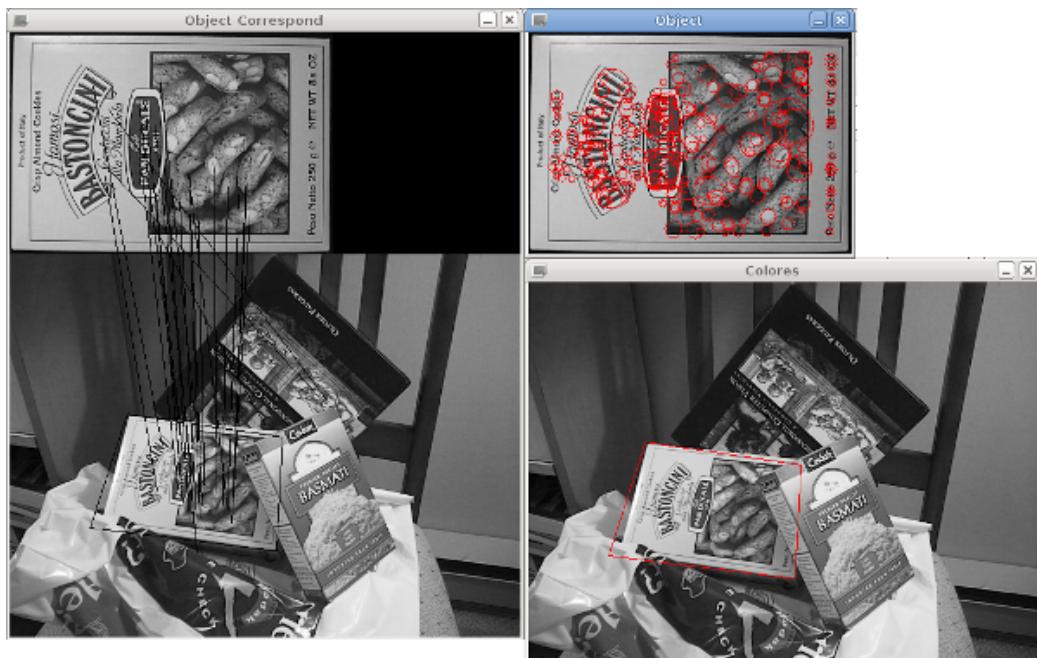
Dans OpenCV, SURF est implémenté par la classe `ObjectFinder`, notre `SurfDetector` utilise donc cette classe pour détecter les objets.

Dans un premier temps, le détecteur est initialisé avec la liste des templates que nous voulons détecter, c'est à cette étape que les descripteurs des templates sont calculés. Ce sont ces descripteurs qui sont utilisés, et comparés à ceux trouvés dans l'image pour déterminer si un objet est présent dans l'image.

La fonction qui fait cette détection est `find` de la classe `ObjectFinder`. Cette fonction calcule la liste des descripteurs contenus dans l'image, et compare ensuite ces descripteurs à ceux trouvés dans les templates. Si un objet est détecté, elle retourne ensuite un ensemble de points qui correspondent

aux coins du polygone englobant l'objet détecté. À partir de ces points, nous calculons le quadruplet (x, y, w, h) présenté dans la partie précédente pour garder une cohérence entre les retours des différents détecteurs.

Le principal problème que nous avons rencontré avec ce détecteur utilisant SURF, est que la vidéo est de mauvaise qualité. Les templates que nous utilisons étant tirés de la vidéo, l'algorithme de calcul de descripteurs ne retourne que très peu de résultats, pas suffisamment pour pouvoir détecter un objet. Après test sur des images en haute qualité, nous avons pu vérifier que le détecteur repère bien des objets qui ont subi une rotation, ou qui ont changé d'échelle.



Exemple de SURF

Sur la figure ci-dessus, nous pouvons voir différentes étapes de l'algorithme SURF. La partie gauche présente la correspondance entre les descripteurs du modèle de boîte, et la scène contenant la boîte parmi divers objets. La partie droite de la figure présente elle deux choses : elle présente en haut les marqueurs trouvés sur la boîte, et en bas montre le résultat de la détection.

4 Tests et résultats

4.1 Présentation des logiciels de test

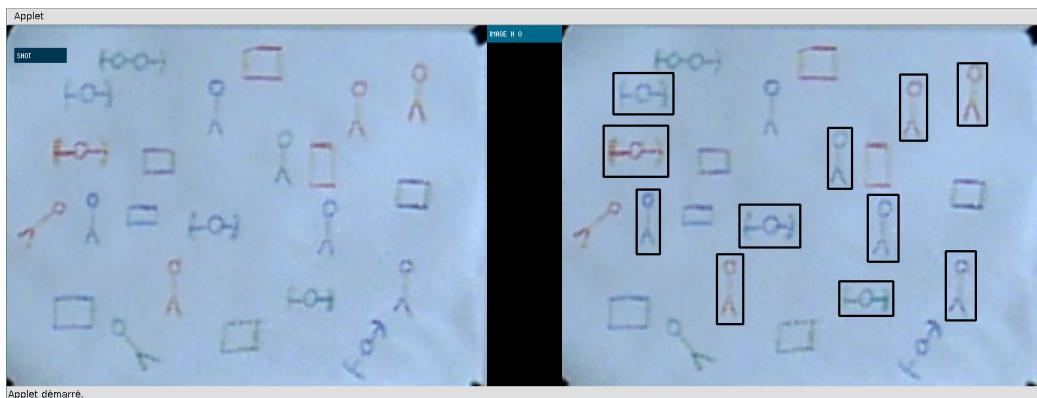
4.1.1 Reconnaissance de formes simples dans un dessin

Le premier logiciel que nous avons développé est un logiciel de reconnaissance de pictogrammes simples dans un dessin. L'affichage de ce logiciel est composé de deux parties : à gauche, le logiciel lit et affiche la vidéo. La partie droite de l'affichage sera utilisé pour afficher les captures de la vidéo sur lesquelles la détection a été faite.

À tout moment pendant la vidéo, un utilisateur peut prendre une capture d'image, à l'aide du bouton associé dans le logiciel. Lorsqu'une capture est prise, notre logiciel effectue des pré-traitements sur l'image obtenue, pour en améliorer la qualité, puis pour binariser l'image, et lance enfin l'analyse de template matching. Une fois l'analyse terminée, l'image obtenue est ajoutée à la liste des captures prises, que l'on peut voir entre les deux zones d'affichage. En cliquant sur le nom de cette capture, elle s'affiche dans la zone d'affichage de droite. Le résultat de l'analyse correspond à la capture prise dans la vidéo enrichie de boîtes englobantes autour des objets détectés.

Lorsque l'utilisateur a pris plusieurs captures, il est possible de naviguer entre ces différentes captures à l'aide des boutons situés entre les deux zones d'affichage.

Pour faciliter la prise de captures, nous permettons à l'utilisateur de mettre en pause la vidéo à l'aide de la touche espace.

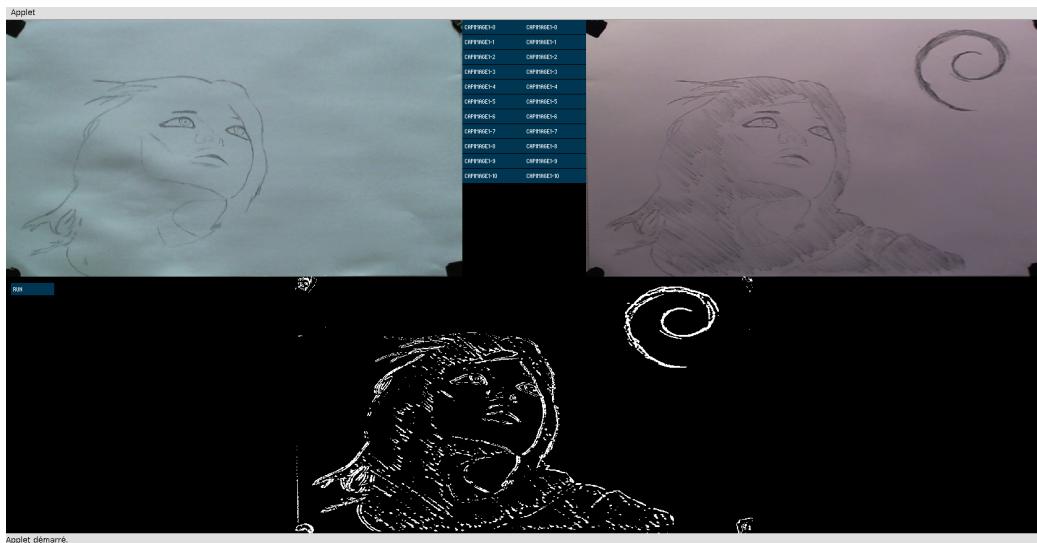


Présentation du logiciel de reconnaissance

4.1.2 Détection des évolutions d'un dessin

Le second logiciel que nous avons développé est un logiciel de détection des évolutions dans le temps d'un dessin. L'affichage de ce logiciel est composé de trois zones d'affichage : les deux zones supérieures présentent les deux images qui sont comparées, ces zones sont séparées par la liste des images affichables. La zone inférieure présente le résultat de la différence entre les deux images.

Ce logiciel prend en entrée un dossier contenant différentes captures d'un dessin prises au cours du temps. À l'aide des boutons centraux, l'utilisateur peut choisir les deux images qu'il veut comparer. Une fois les deux images choisies, on peut lancer la comparaison à l'aide du bouton run. Une fois la comparaison achevée, le résultat s'affiche dans la zone inférieure. Ce résultat correspond à la différence entre les deux images d'origine binarisées. Le résultat est également stocké, afin d'éviter de recalculer l'image si l'utilisateur demande un calcul déjà fait.



Présentation du logiciel de détection d'évolutions

4.2 Résultats obtenus

5 Conclusion

Bilan

Évolution possible applications