

Algorithmes de détection de cercles et de lignes dans une image par la méthode de Hough

-

Environnement C++ et bibliothèque OpenCV

Interface Qt

Tuteur : Mohamed KERKECH

Nicolas VRIGNAUD, PeiP2

28 mai 2021

Projet d’informatique

Rapport

**Sommaire**

Introduction2

1. Outils et structure3
   1. Outils utilisés et cahier des charges3
   2. Structure du programme envisagée6
2. Méthode théorique et algorithme de Hough10
   1. Transformée de Hough et détection de formes 10
   2. Algorithme par la méthode de Hough11
3. Difficultés et possibles améliorations13
   1. Difficultés rencontrées et solutions13
   2. Points forts et faibles, améliorations (exemple de fonct interface)15

Conclusion18

Table des illustrations19

**Introduction**

A l’issue de ce projet d’informatique, j’ai travaillé sur le développement d’un programme et d’une interface permettant d’effectuer du traitement d’images. Le but principal était de réaliser un algorithme utilisant la méthode de Hough pour détecter les cercles ou les lignes présents sur une image. Cela avait donc pour objectif d’apprendre à utiliser une nouvelle bibliothèque sous C++, appelée OpenCV, ainsi que le logiciel graphique Qt pour réaliser l’interface utilisateur.

Plus précisément, cette interface est capable d’ouvrir une image choisie par l’utilisateur, puis de lui appliquer le filtre nécessaire afin de détecter les cercles présents. J’y ai aussi ajouté des éléments qui donne la possibilité à l’utilisateur de régler lui-même les paramètres de ces méthodes ou d’enregistrer les images obtenues après modification. En plus de cela, il m’a été possible de développer un algorithme similaire utilisant la méthode de Hough mais détectant les lignes sur une image.

Ainsi, j’ai dû me familiariser avec de nouveaux outils pour construire un programme structuré Je vais donc expliquer cette organisation du code selon les fonctionnalités prises en charge. Je vais ensuite présenter les différents concepts qui définissent la transformée de Hough et comment elle est utilisée dans le traitement d’images, dans le but de construire un algorithme reposant sur ce concept. Finalement, je ferais l’analyse des difficultés rencontrées ainsi que des possibles améliorations du programme dans la continuité de sa structure actuelle.

1. **Outils et structure**

Dans un premier temps, je vais introduire les principaux outils utilisés lors de ce projet et expliquer la structure qui a été envisagée pour le programme final. Cette explication se fera à l’aide de schémas synoptiques de la solution finale.

* 1. Outils utilisés et cahier des charges

Le projet a été écrit avec le langage de programmation C++. Cependant, il utilise deux outils différents pour prendre en charge les fonctionnalités voulues dans le programme final.

* Bibliothèque externe OpenCV

OpenCv est une bibliothèque qui ajoute de nombreuses fonctionnalités au sein de plusieurs langages de programmation dont le C++. Elle permet d’appliquer plusieurs procédés très utilisés en traitement d’images ou traitement vidéo.

Tout d’abord, une image est définie par un objet Mat qui est donc une matrice à trois dimensions représentant les abscisses, les ordonnées et la bande. La bande est définie par un vecteur à trois coordonnées pour une image en couleur bleu/vert/rouge et non rouge/vert/bleu comme on pourrait s’y attendre (dans la suite de ce rapport, les abréviations BGR et RGB seront utilisées). La bande est un tableau d’entiers pour une image en nuances de gris. On peut entre-autres déclarer une image matricielle en la créant nous-même ou en lui donnant un chemin à suivre. Le type de l’image (CV\_8U) définit ici une image à un seul canal, donc en nuances de gris et Scalar(B, R, G) lui donne une couleur, blanc dans ce cas.

Extrait  : Création d’une nouvelle matrice image

Extrait  : Lecture d’une image située avec un chemin path

Extrait  : Ecriture au chemin path de l’image

En mode console, il est très facile d’afficher cette image en utilisant les deux méthodes suivantes.

 Extrait  : Affichage de l’image dans une fenêtre OpenCV

Pour ce qui est des fonctions, on peut en citer plusieurs qui donnent accès à des transformations simples.

 Extrait  : Méthode OpenCV de changement de couleur

Cette première méthode prend en paramètres l’image source, l’image de destination et un paramètre prédéfini qui indique la transformation à effectuer (de nuances de gris à BGR, de BGR à nuances de gris, de BGR à RGB, etc.).

 Extrait  : Accès aux pixels des bandes d’une image

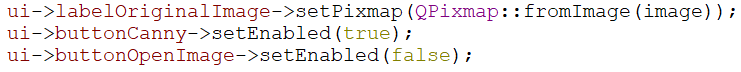
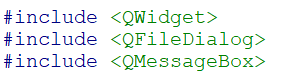
On peut aussi, comme le montre l’image ci-dessus, récupérer un pixel d’une image à une coordonnée précise et le décomposer selon les bandes BGR pour en changer la valeur. Cette valeur allant de 0 à 255 définit la couleur du pixel. [0] indique la bande bleue, [1] pour le vert et [2] pour le rouge. Enfin il existe d’autres méthodes spécifiques à la transformée de Hough que nous verrons plus en détail dans la partie 2.2).

* Logiciel et interface de programmation Qt

Qt, quant à lui, est un logiciel qui permet la réalisation d’interfaces graphiques pour plusieurs langages de programmation dont le C++. Il ajoute aussi des objets et des méthodes spécifiques pour la création de cette interface.

Il intègre le principe des formulaires à la manière du framework .NET de C# et nous simplifie la gestion du placement d’éléments sur les formulaires comme des boutons, des zones de texte, des sliders, tout en les ajustant grâce à des layouts horizontaux, verticaux ou en grille par exemple. Pour encore plus d’organisation, on peut de définir une fenêtre principale en tant que fenêtre parent MDI afin de contenir tous les autres formulaires ouverts à partir de celle-ci. La transmission de code entre les formulaires et les interfaces graphiques s’effectue grâce aux gestionnaires d’événements à l’intérieur desquels on écrit notre code.

En développant avec ce logiciel, on remarque cependant de nombreuses différences, notamment au niveau de la syntaxe du code. Il ressemble au C++ mais il est adapté quand il s’agit de récupérer et modifier les informations contenues dans les formulaires. Les éléments de formulaire et les objets Qt sont appelés et instanciés grâce à des mots clés tels que QButton, QFileDialog ou QMessageBox. La récupération d’un élément particulier dans le formulaire s’effectue grâce à une instruction spécifique.



Extrait  : Inclusion des objets Qt dans un formulaire Extrait  : Instruction d’accès aux éléments d’interface

* Cahier des charges

Ce projet a eu pour but d’acquérir de nouvelles compétences tout en utilisant celles dont nous disposions déjà. Ainsi, les objectifs sont en lien avec le traitement d’images et ses applications dans le domaine de l’informatique.

Les opérations à effectuer sur les images doivent utiliser les fonctions présentes dans la bibliothèque OpenCV. Il s’agissait de la première fois que j’ai dû installer une bibliothèque externe pour réaliser de nouvelles opérations avec des méthodes prédéfinies. C’est pourquoi il s’agit d’une compétence qui sera utile à chaque fois qu’il sera nécessaire d’ajouter de nouveaux outils à un langage de programmation de cette manière.

Dans le cas particulier du traitement d’images, il m’a fallu apprendre à se servir des méthodes de OpenCV pour réaliser des modifications d’images. La lecture, l’affichage et le traitement d’une image sont des connaissances nécessaires avant de se tourner vers un fonctionnement plus complexe.

L’objectif principal étant la détection de cercles et de lignes dans une image, il fallait bien s’intéresser aux méthodes en question. La méthode de Hough est déjà prédéfinie dans OpenCV, mais encore faut-il comprendre son fonctionnement pour pouvoir compéter l’algorithme de recherche et d’affichage de formes détectées encore une fois avec des méthodes précises créées dans ce but précis.

Enfin, pour que l’utilisateur se sente à l’aise lors du lancement du programme, il ne faut pas oublier de s’intéresser à l’interface graphique. C’est celle-ci qui permet à l’utilisateur de vraiment pouvoir interagir avec toutes les étapes d’une détection de formes.

* 1. Structure du programme envisagée

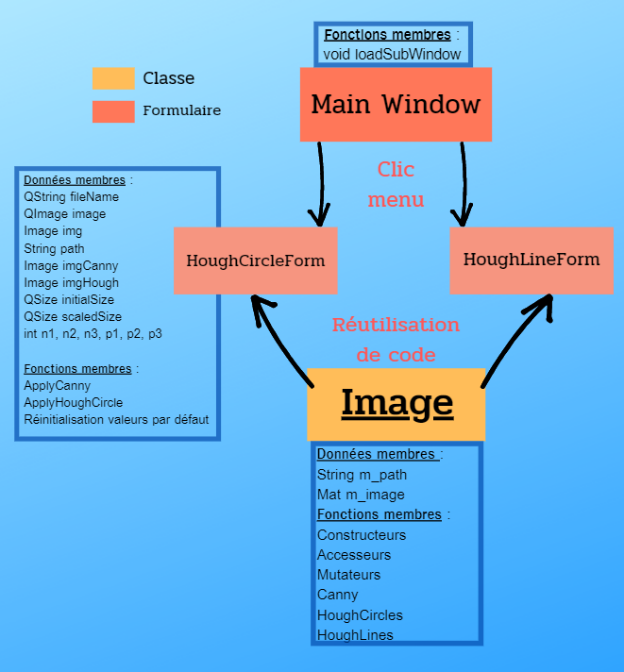
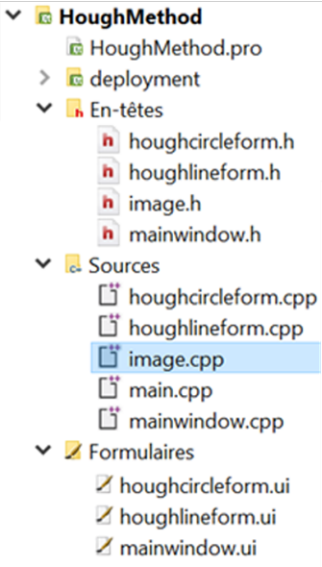
Le programme est structuré de manière organisée selon les fonctionnalités utilisées, selon les méthodes appliquées et leur provenance. Je vais d’abord revenir sur la structure envisagée sous forme de schémas, puis détailler les différentes classes et méthodes qui définissent une interface utilisateur fonctionnelle. Cela permettra de visualiser les liens existants entre les différentes fenêtres de l’interface.

Figure  : Structure statique du programme

Ainsi, l’organisation en classes est la suivante. Les fonctionnalités de traitement d’images avec OpenCV sont exploitées dans la classe Image (image.h, image.cpp). Elle contient deux données membres, une pour le chemin de l’image (m\_path) et une autre pour la matrice (m\_image) qui définit une image sous OpenCV. Elle comprend évidemment des constructeurs, ainsi que des accesseurs et des mutateurs. Enfin viennent les méthodes de traitement d’images, qu’il s’agisse des fonctions développées lors de la prise en main de la bibliothèque ou les méthodes utilisant la transformée de Hough. Cette classe regroupe donc tout ce qui concerne la bibliothèque au même endroit pour une meilleure visibilité.



Le fichier .pro définit quant à lui les paramètres d’utilisation de Qt. Il liste notamment les fichiers présents dans le projet et les paramètres d’utilisation de OpenCV, ce qui permet de tout lier ensemble. Nous ne verrons pas plus en détail de fichier puisqu’il est généré et modifié automatiquement par le logiciel.

Les fichiers d’interface graphique sont définis par les fichiers .h, .cpp et .ui. Les fichiers .h et .cpp ont les mêmes objectifs que pour une classe en C++, à savoir de lister puis de définir les données membres et le code des fonctions membres. Le fichier .ui s’occupe de la mise en page de la fenêtre. On peut déposer et organiser ici les différents éléments à afficher lors du chargement de chaque fenêtre. On fait ensuite le lien entre les 3 fichiers dans le .cpp qui contient les gestionnaires d’événement et permet d’effectuer toutes sortes d’actions au sein de la fenêtre en interagissant avec.

Figure 2 : Organisation des fichiers

On retrouve donc deux types de fenêtres. Les fichiers mainwindow correspondent à la fenêtre principale dans laquelle j’ai défini une MdiArea. Celle-ci contient toutes les autres fenêtres qui s’affichent lorsque l’on clique sur un bouton de menu. Le procédé d’ouverture étant la même quelle que soit le formulaire, la fonction loadSubWindow est donc appelé pour l’ouverture de chaque fenêtre. En particulier, il est possible d’ouvrir la fenêtre de détection de cercles ou de lignes et de l’utiliser pour réaliser ces opérations. On peut retrouver la construction du code d’une classe C++ dans le .h, à savoir les inclusions et la définition de la classe avec ses constructeurs, destructeurs, données membres publics, ses gestionnaires d’événements et ses fonctions membres privées. Dans le .cpp, on écrit le code qui sera réalisé lors de l’utilisation des gestionnaires d’événements et des méthodes. Les deux fenêtre HoughCircleForm et HoughLineForm étant presque identique dans le fonctionnement, elles contiennent des données et des fonctions membres équivalentes.

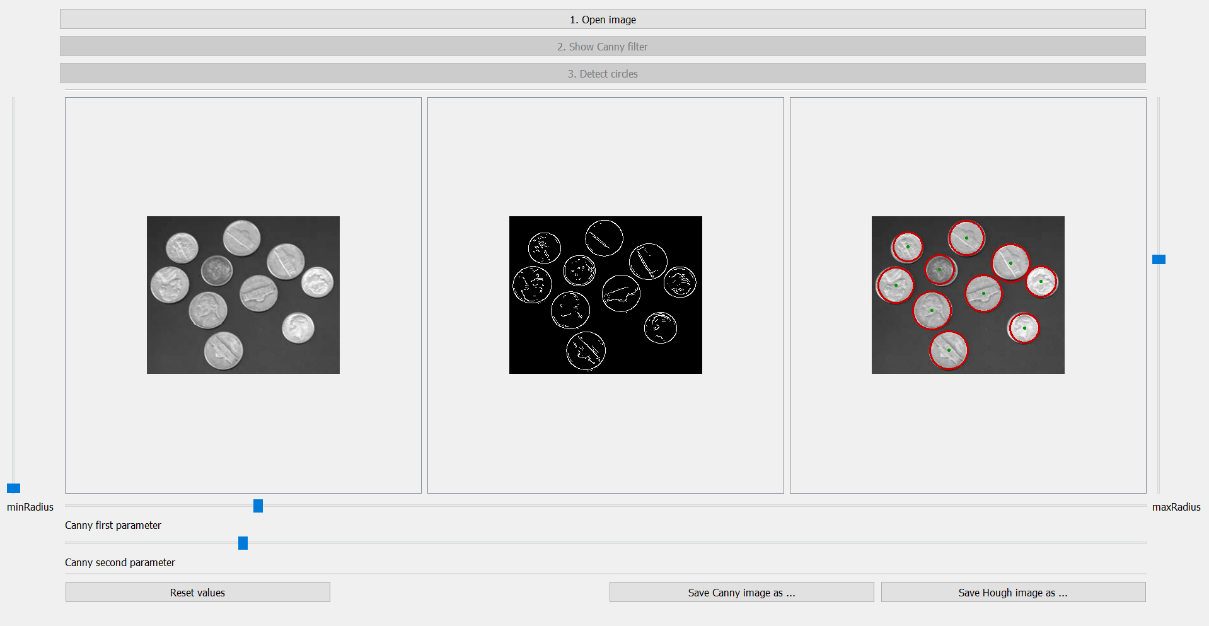


Figure 3 : Interface graphique de détection de cercles

En lançant le logiciel, on arrive sur l’interface principale mainwindow qui s’affiche par défaut en mode plein écran pour plus de facilité. Il est ensuite possible de choisir dans le menu quelle fenêtre on désire ouvrir. Comme les deux fenêtres de détection sont semblables, je me contenterai de décrire la fenêtre houghcircleform.

L’utilisateur commence par ouvrir l’image qu’il souhaite à l’aide d’un QFileDialog. S’il a ouvert une image, alors il peut visualiser les images modifiées de Canny et de Hough à l’aide d’autres boutons. L’option de modification de certains paramètres utilisés dans l’algorithme est ensuite accessible à l’aide de Sliders pour obtenir une bonne précision et voir les changements en temps réel à l’écran. Les paramètres de Canny modifient la prise en compte des détails comme faisant partie des bords. Les deux autres paramètres définissent les rayon minimum et maximum des cercles détectés et sont très utiles lorsque des faux cercles sont affichés. Si l’utilisateur le désire, il peut réinitialiser les paramètres et recommencer la recherche de précision. Il peut finalement enregistrer l’une ou l’autre des images modifiées avec là aussi une MessageBox qui avertit l’annulation ou le bon déroulement de la sauvegarde. Pour recommencer ce cycle, l’utilisateur peut soit fermer la fenêtre et choisir une autre fenêtre, soit ouvrir une image différente et effectuer la même opération de traitement d’images. Le principe est le même pour la détection de lignes par la méthode de Hough. La seule différence notable sont les paramètres modifiables avec les Sliders verticaux.

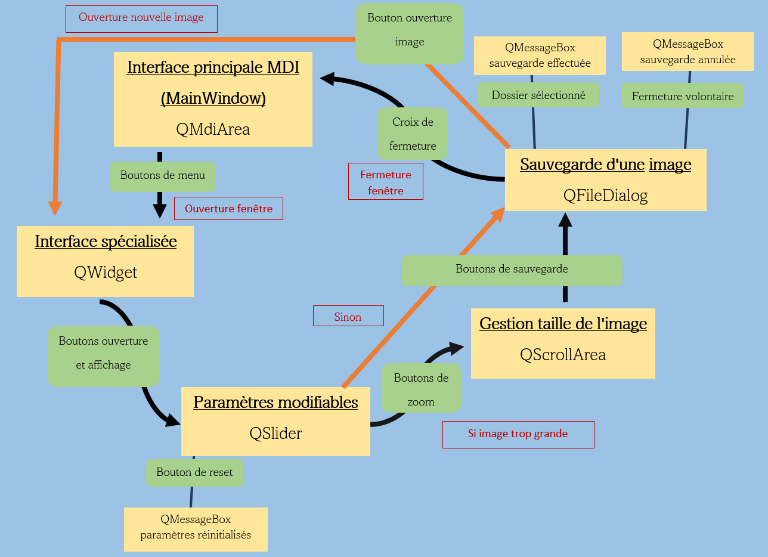


Figure 4 : Schéma de l’interface homme-machine

De plus, afin de gérer encore plus de cas, une partie de code sert à la prise en charge des images trop grandes pour être affichées en entier. Pour cette raison, les images sont placées dans des ScrollArea, et des boutons supplémentaires s’affichent si l’image est trop grande. Les images sont par défaut réduites pour entrer dans leur zone attribuée. Si l’utilisateur clique une première fois, il enclenche un zoom sur cette image et peut se déplacer avec des ScrollBar. Cela lui permet de se rendre compte en détail du résultat affiché. Au second clic de l’utilisateur, l’image se dézoome et retrouve sa taille précédente. J’ai réalisé cette partie de code après m’être aperçu de la difficulté à distinguer les formes détectées pour une image dont la taille a été adaptée.

1. **Méthode théorique et algorithme de Hough**

Dans cette deuxième partie, il va être question d’expliquer la transformée de Hough et son application dans le traitement d’images pour la détection de formes. Après cela, nous pourrons entrer en détail dans le fonctionnement de l’algorithme développé dans ce projet.

* 1. Transformée de Hough et détection de formes

La transformée de Hough est très utile dans le traitement d’images et rend possible la détection de formes dans une image, depuis les lignes et les cercles jusqu’aux ellipses. A partir de l’équation de la forme recherchée, il est assez facile de trouver les paramètres manquants de cette forme pour ensuite revenir à son équation et ainsi la tracer. Je vais expliquer ici comment la transformée de Hough permet de détecter un cercle ainsi qu’une ligne dans une image, à savoir expliciter le principe utilisé par les méthodes prédéfinies.

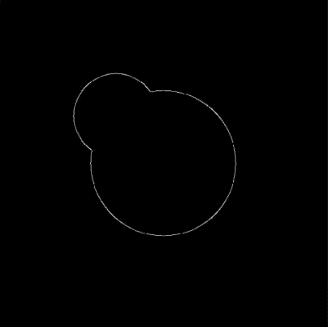
Dans les deux cas, il faut commencer par appliquer un filtre de Canny sur l’image à analyser. Ce filtre constitue le point de départ pour utiliser la transformée de Hough dans de bonnes conditions. L’image retournée est une image des bords des objets. Elle est composée d’un fond noir et les bords des objets de l’image sont colorés en blanc.

Figure 5 : Filtrage de Canny de 2 cercles

* Méthode pour les cercles

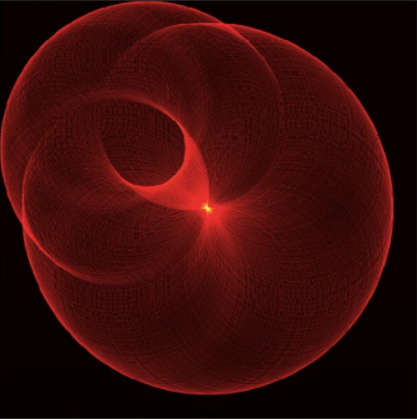
La méthode dans le cas d’un cercle de rayon R est plutôt facile à comprendre. On prendra le rayon du grand cercle dans cet exemple. Pour chaque pixel blanc de l’image filtrée, on va tracer un cercle de rayon R sur un accumulateur. Cet accumulateur est en quelque sorte une image dont chaque pixel augmente en intensité à chaque fois qu’un nouveau cercle tracé passe sur ce pixel. Ainsi, lorsque chaque cercle a été tracé, on récupère les maximums d’intensité de l’accumulateur qui définissent les centres des cercles de rayon R détectés sur l’image initiale. En répétant cela pour chaque rayon possible, on trouve tous les cercles présents sur l’image, avec une certaine précision qui est susceptible de varier.

Figure 6 : Accumulateur de cercles

* Méthode pour les lignes

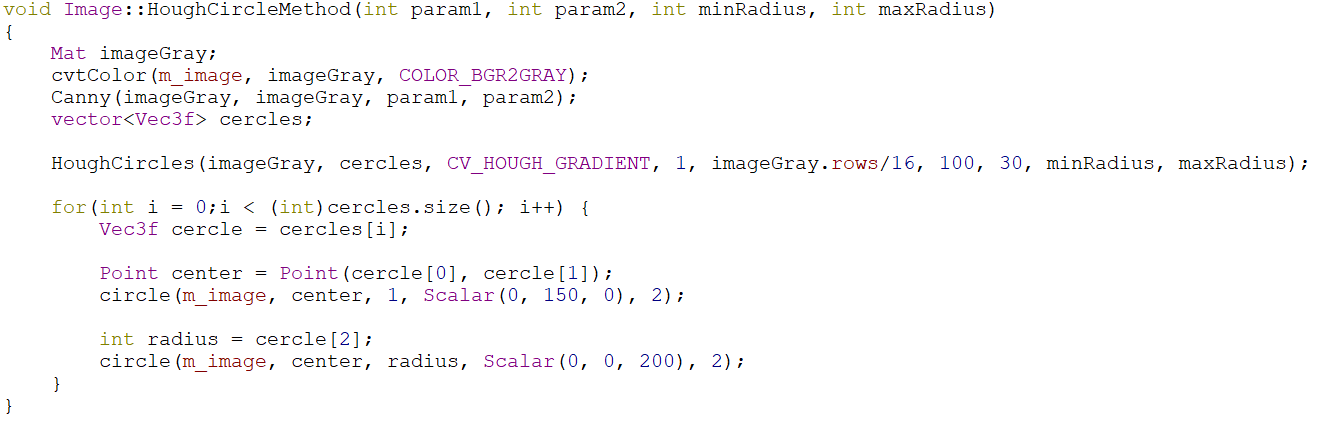
Dans le cas des lignes, il existe deux manières de procéder selon l’équation qu’on choisit pour les définir. Dans le cas du traitement d’images numériques, on utilise la définition de cette ligne en fonction de ρ, la hauteur de cette ligne passant par l’origine et θ, l’angle que fait cette droite avec l’axe des abscisses. On obtient ainsi l’équation d’une ligne ρ = x cos(θ) + y sin(θ). On prend donc l’image contenant des lignes et filtrée par filtrage de Canny. On fait varier θ et ρ pour chaque pixel blanc, ce qui trace une sinusoïde dans l’accumulateur (θ est sur l’axe des abscisses et ρ est sur l’axe des ordonnées). Lorsque chaque pixel a été analysé, on récupère les maximums d’intensité de l’accumulateur, indiquées en clair sur celui-ci. Les coordonnées de ces points nous donnent les valeurs de ρ et θ, qui nous servent à tracer les lignes sur l’image initiale en fonction de x et y.

Figure 7 : Accumulateur de lignes

* 1. Algorithme par la méthode de Hough

Après avoir expliqué de manière théorique la transformée de Hough et les méthodes prédéfinies, je vais détailler le code qui m’a servi à détecter les formes présentes sur une image. Il se compose de plusieurs parties, qui sont la conversion de l’image initiale en ne gardant que les bords, l’application de la méthode de Hough pour les cercles ou les lignes et enfin le dessin des formes détectés sur l’image initiale.

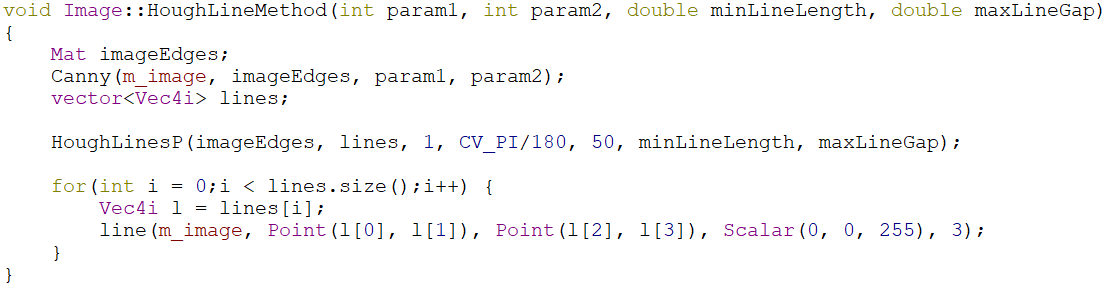
* Algorithme de détection de cercles



Extrait 9 : Algorithme de Hough dans le cas des cercles

Cet algorithme détecte les cercles présents sur une image avant de les dessiner. On définit en premier lieu une matrice imageGray qui va subir les différentes transformations. On la convertit en nuances de gris et on lui applique le filtrage de Canny. Cette fonction a comme paramètres l’image source, l’image de destination et 2 paramètres qui pourront varier par la suite pour ajuster la précision voulue. On définit ensuite une liste de vecteurs à 3 coordonnées (abscisse, ordonnée, rayon), soit les cercles détectés lors de l’appel à la fonction HoughCircles. Celle-ci prend les paramètres suivants : l’image source, la liste des cercles détectés, la méthode utilisée (ici la méthode du gradient de Hough), la résolution de l’accumulateur, la distance minimale entre les cercles détectés, un paramètre de seuil pour l’image et un autre pour l’accumulateur, le rayon minimum et le rayon maximum des cercles détectés. Après ce calcul, on récupère chaque cercle de la liste, on dessine un centre et le cercle associé. La fonction utilisée est circle qui est définie selon l’image source, un point correspondant au centre du cercle, son rayon, la couleur voulue et l’épaisseur du trait.

* Algorithme de détection de lignes



Extrait 10 : Algorithme de Hough dans le cas des lignes

Ce deuxième algorithme détecte les lignes présentes sur une image pour les tracer ensuite à l’aide de méthodes spécifiques. Il est très semblable à l’algorithme précédent dans sa logique d’application. Tout d’abord, on applique le filtrage de Canny sur l’image que l’on copie dans la matrice imageEdges. On crée une liste de vecteur à 4 coordonnées représentant les coordonnées des deux points utilisés pour tracer cette ligne. On applique ensuite la fonction HoughLinesP qui prend en paramètres l’image source, la liste de lignes, la résolution de l’accumulateur sur la distance ρ, sa résolution en angle θ, son seuil, la longueur minimale des segments et la distance maximale autorisée pour deux points d’une même ligne. Enfin, pour chaque ligne, on récupère les coordonnées des deux points et on utilise la fonction line qui se définit avec l’image source, deux points, une couleur et une épaisseur.

1. **Difficultés et possibles améliorations**

Pour finir, je vais détailler un peu plus ici les difficultés qui se sont présentées lors de la réalisation du projet. Celles-ci ont été surpassées le mieux possibles en réalisant des tests et en appliquant les solutions appropriées. De plus, il est toujours intéressant d’ajouter au programme final des améliorations compte tenu des points forts et des points faibles qu’il suscite.

1. Difficultés rencontrées et solutions

Ces deux nouveaux outils que sont OpenCV et Qt ont été assez difficile à prendre en main, même s’ils ne diffèrent pas tellement du C++ ou du C#, langages que je connaissais déjà avant de réaliser ce projet. D’autres difficultés sont aussi apparues lors de la construction de l’algorithme de Hough pour arriver à configurer les paramètres des méthodes. Après cela, le passage du code en mode console vers une interface utilisateur réutilisant le code initial a été plutôt délicat.

* Installation et prise en main des nouveaux outils :

Pour commencer ce projet, il m’a fallu installer la bibliothèque OpenCV. Cela a été assez compliqué de comprendre les étapes nécessaires pour enfin arriver à afficher une image de test sans obtenir d’erreur. Une fois l’installation terminée, il a été nécessaire de prendre en main les différentes méthodes et l’organisation du logiciel Qt. En codant quelques fonctions de base de conversion d’images, j’ai pu me rendre compte du fonctionnement des nouvelles méthodes. Le développement de code utilisant la bibliothèque OpenCV est ensuite devenu beaucoup plus facile à comprendre et à réaliser.

* Paramètres des méthodes de Hough :

Cependant, en appliquant certaines méthodes plus complexes et utiles à mon projet, j’ai été confronté à de nouvelles difficultés. En effet, les méthodes pour la détection de formes dans une image, ainsi que la fonction de filtrage de Canny comprennent des paramètres que j’ai eu du mal à comprendre. Ces paramètres ont tous une signification précise indiquée dans la partie 2.2). Pour remédier à cela, j’ai réalisé plusieurs tests en faisant varier tour à tour les paramètres et ainsi observer le résultat. J’ai pu ensuite définir les paramètres qu’il serait bon de fixer et ceux qui pourraient être modifiés par l’utilisateur pour améliorer la précision de la détection de formes.

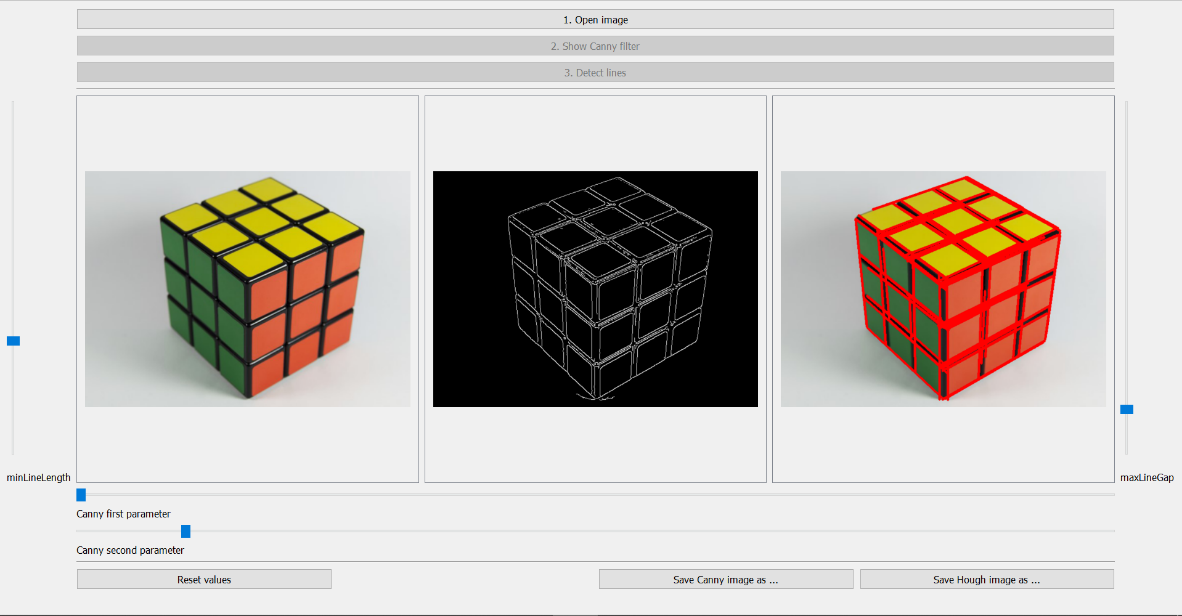


Figure 8 : Interface graphique de détection des lignes

* Passage du code console vers une interface graphique :

Après avoir vérifié que le code marchait correctement, je me suis tourné vers Qt pour réaliser une interface graphique qui rend ce code plus convivial et facile de compréhension pour un utilisateur. Malgré un fonctionnement similaire au framework .NET de C#, les méthodes et la syntaxe sont différentes et cela m’a pris du temps avant de maitriser toutes ces différences. En ayant recours à la documentation, j’ai finalement assimilé et appliqué ce nouveau vocabulaire afin d’ajouter du code supplémentaire pour inclure du contenu visible à l’utilisateur. Cet ajout de code a été l’occasion de tester mon code petit à petit tout en incluant les nouvelles actions possibles à l’utilisateur. Chaque bloc de code a été soumis à une mise en route du programme pour la vérification de son bon fonctionnement.

* Détails et corrections éventuelles, gestion de nouveaux cas possibles :

Si l’on s’attarde sur le programme final, les plus grandes difficultés se trouvent dans les détails d’amélioration de cette interface. L’exemple principal est la gestion d’une image de grande taille dans l’interface.J’ai dû trouver un moyen de les afficher en entier dans la fenêtre. Il a été très simple d’ajuster la taille de l’image, mais dans ce cas, les formes affichées ne sont plus distinguables. Afin d’effectuer un zoom sur les images et voir correctement les cercles ou les lignes s’afficher dans l’image, j’ai longtemps réfléchi à une solution. J’ai finalement opté pour inclure les images dans des ScrollArea et demander à l’utilisateur lorsqu’il veut zoomer ou non par l’intermédiaire de boutons. Ceux-ci s’affichent uniquement lorsque l’image originale a dû être redimensionnée. Ainsi, pour un clic impair (comme le premier), l’image s’agrandit et des ScrollBar s’affichent, alors que pour un clic pair, l’image se réduit. Toute cette partie de code a fait l’objet d’une réflexion en amont afin de trouver la solution la plus adaptée ainsi que de plusieurs tests pour vérifier notamment que la couleur de l’image n’est jamais inversée (car une image OpenCV est en BGR alors qu’en Qt elle est en RGB, ce qui nécessite une inversion).

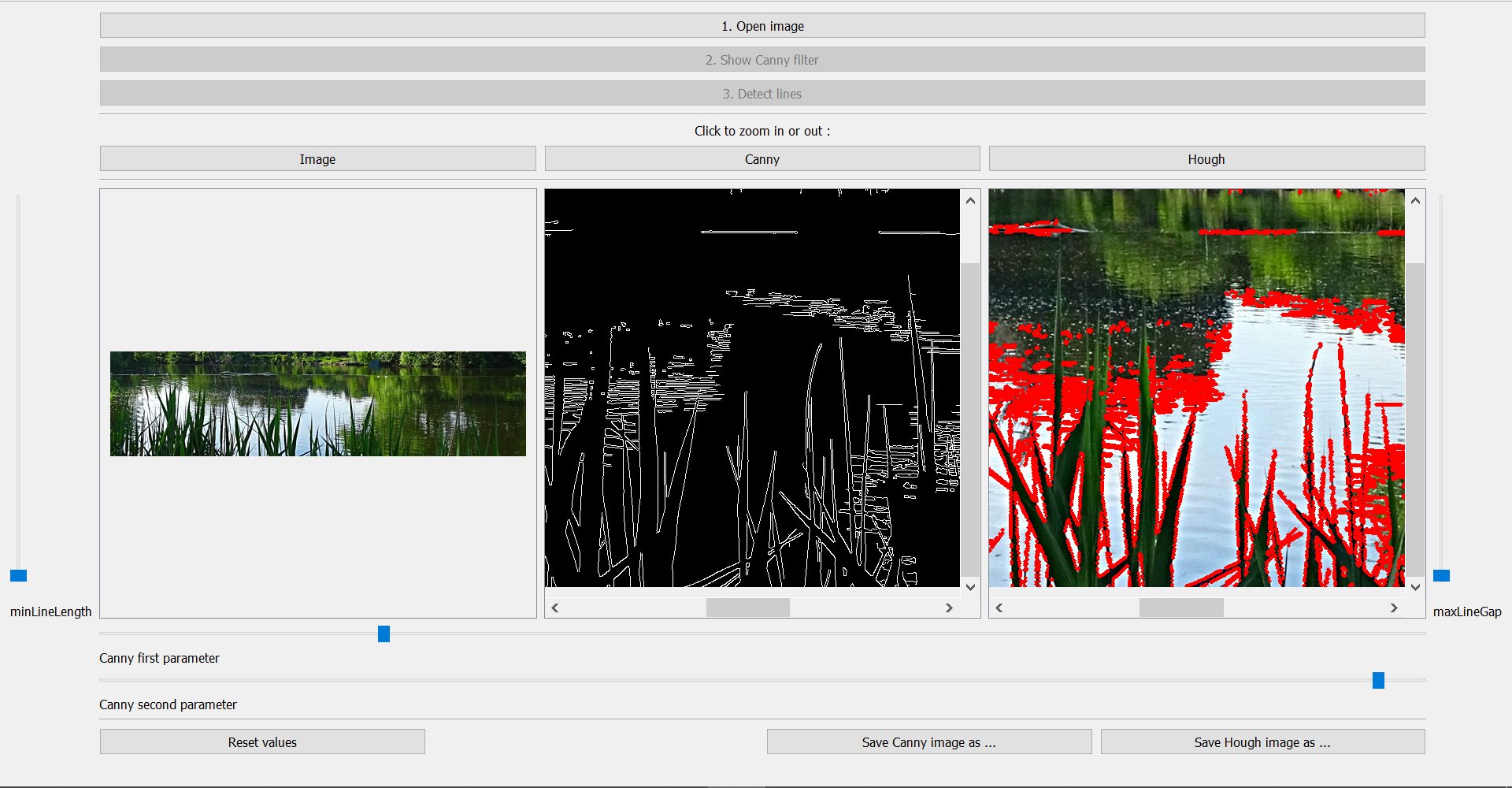


Figure 9 : Interface incluant un zoom pour les grandes images

1. Points forts et faibles, améliorations

Finalement, le programme et l’interface permettent à l’utilisateur d’effectuer convenablement la détection de formes. Il manque forcément quelques éléments qui pourraient rendre ce logiciel encore plus convivial, contrant ainsi les points faibles qu’il comporte et renforçant ses points forts.

* Points forts :

Le code est d’abord bien structuré, en séparant distinctement les différentes fonctionnalités offertes par OpenCV et par Qt. Tout ce qui concerne le traitement d’une image, sa lecture ou sa modification sont regroupés dans la classe Image. Lorsqu’une autre partie du code a besoin de cette fonctionnalité, il va faire appel à cette classe. En ce qui concerne l’interface graphique, elle est composée de plusieurs formulaires qui échangent entre eux comme nous l’avons vu dans la partie 1.2). De plus, les objets Qt QMessageBox et QFileDialog sont appelés de manière à donner le droit à l’utilisateur d’ouvrir une image, de la sauvegarder ou l’avertir lorsqu’une action se passe comme prévu ou non.

Le programme final permet de gérer pas mal de problèmes qui pourraient être rencontrés lors de son utilisation, notamment la taille de l’image, mais aussi l’ordre d’appui sur les boutons qui causerait de nombreux problèmes s’il n’était pas mis en place. Pour finir, cela donne une interface qui opte pour une certaine adaptabilité aux envies de l’utilisateur, en particulier en ce qui concerne le réglage de la précision voulue. Il peut ainsi, en plus d’effectuer son traitement d’images, en apprendre davantage sur l’utilité de ces paramètres et leur rôle dans la modification des formes détectées.

* Points faibles :

Cependant, il est fort probable que de légers problèmes soit encore présents. On ne peut jamais prévoir l’intégralité des cas possibles et cela demande de nombreux tests, voire de recevoir l’avis des utilisateurs qui peuvent déceler ces problèmes par hasard. En termes d’expérience utilisateur, l’interface n’est pas la plus esthétique. Même si ce n’est pas un objectif réel de ce projet, dans le cas du développement d’un logiciel, il est important de se pencher sur cette question avant qu’il soit publié et utilisé par d’autres personnes.

Sur les fenêtres de détection de formes, on ne retrouve pas la totalité des fonctionnalités qui pourraient être disponible. On pourrait imaginer avoir plus de Sliders pour modifier l’intégralité des paramètres des méthodes OpenCV, ou bien avoir des boutons de réinitialisation pour chacun de ces paramètres. Pour plus de personnalisation, on aurait aussi pu penser à laisser le choix des couleurs des formes détectées à l’utilisateur, par exemple. Cela nécessiterait de changer l’apparence de cette fenêtre et lui ajouter de nouveaux contrôles, qui seraient ordonnés dans un menu.

* Améliorations possibles :

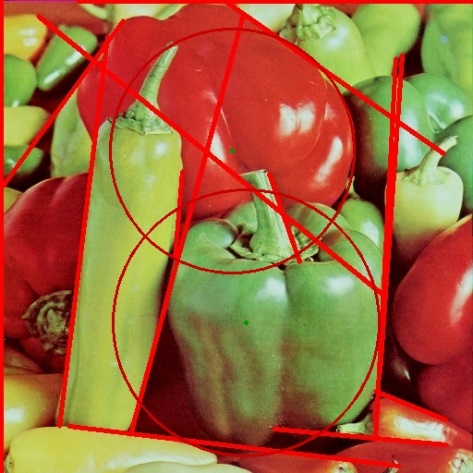
Il y a donc plein d’améliorations qui seraient utiles d’ajouter à ce programme. Nous avons vu quelques exemples de nouvelles fonctionnalités pour les fenêtres existantes précédemment, mais cela ne se restreint pas à cette limite. On peut tout à fait imaginer développer de nouvelles fenêtres à ouvrir grâce au menu principal, qui convertiraient une image d’une nouvelle manière. Entre autres, le passage d’une image en nuances de gris, l’utilisation d’un masque, l’affichage d’un histogramme, la modification de couleur, etc. On pourrait aussi penser, pour rester dans le thème, à une nouvelle fenêtre qui détecterait les deux types de formes en même temps. On peut tout à fait effectuer cette opération dès à présent mais cela nécessite d’effectuer deux fois la détection de formes, une fois pour les lignes et une fois pour les cercles. C’est moins précis que d’avoir les deux algorithmes qui fonctionnent sur la même image, dont on verrait le changement en temps réel.

Figure 10 : Résultat possible des deux détections en une seule image

En ce qui concerne le travail des détails techniques, certains nécessitent des connaissances très poussées dans les langages et les logiciels utilisés. Si on charge une image lourde et de grande taille dans le programme et qu’on souhaite lui appliquer l’algorithme de Hough, cela va demander beaucoup de temps avant d’avoir un résultat complet et l’interface graphique risque de planter jusqu’à la fin des calculs. On pourrait ici penser à transférer ce code gros en calculs dans un second fil d’exécution afin de ne pas bloquer l’interface. En plus de cela, il faudra avertir l’utilisateur du temps restant avant l’affichage de l’image finale et donc programmer une barre de progression qui s’affichera lors du calcul par exemple. Enfin, comme toutes les fenêtres du logiciel sont sur le modèle d’une modification de l’image originale, il faudra penser à réutiliser le code déjà présent pour l’ouverture d’une image, sa sauvegarde ou encore la gestion de sa taille. Cela sera d’autant plus utile qu’il y aura de fonctionnalités différentes, c’est pourquoi une classe parente qui contiendrait tout ce code commun serait très utile. Il n’y a aucune limite de création de nouvelles fonctionnalités et on pourrait augmenter cette liste indéfiniment pour aboutir à un logiciel tel que les utilisateurs le préfèrent.

**Conclusion**

Finalement, ce projet m’a donné l’occasion de développer un programme répondant à un objectif précis. A l’aide du langage de programmation C++ et de la bibliothèque OpenCV, le traitement d’une image est rendu beaucoup plus facile. En particulier, les méthodes de filtrage et de Hough détectent les cercles et les lignes présents dans une image. On peut ensuite les indiquer par un dessin sur cette même image grâce à plusieurs fonctions de traçage de forme.

De plus, il m’a été possible de créer une interface graphique détaillée avec le logiciel Qt pour qu’un utilisateur puisse effectuer ces opérations de lui-même. Il peut ainsi choisir l’image à analyser, lui appliquer les différentes méthodes de détection de formes et régler leurs paramètres avant de sauvegarder le résultat final.

Malgré les difficultés rencontrées, j’ai pu aboutir à un programme fonctionnel répondant aux attentes du projet. Il est bien entendu intéressant de modifier ce logiciel pour lui ajouter de nouvelles fonctionnalités ou bien améliorer le rendu de l’interface de détection de cercles et de lignes pour la rendre encore plus simple d’utilisation pour un utilisateur.

**Table des illustrations**

Extrait 1 : Création d’une nouvelle matrice image 3

Extrait 2 : Lecture d’une image située avec un chemin path 3

Extrait 3 : Ecriture au chemin path de l’image 3

Extrait 4 : Affichage de l’image dans une fenêtre OpenCV 3

Extrait 5 : Méthode OpenCV de changement de couleur 4

Extrait 6 : Accès aux pixels des bandes d’une image 4

Extrait 7 : Inclusion des objets Qt dans un formulaire 5

Extrait 8 : Instruction d’accès aux éléments d’interface 5

Extrait 9 : Algorithme de Hough dans le cas des cercles 11

Extrait 10 : Algorithme de Hough dans le cas des lignes 12

Firgure 1 : Structure statique du programme 6

Firgure 2 : Organisation des fichiers 7

Firgure 3 : Interface graphique de détection des cercles 8

Firgure 4 : Schéma de l’interface homme-machine 9

Firgure 5 : Filtrage de Canny de 2 cercles 10

Firgure 6 : Accumulateur de cercles 10

Firgure 7 : Accumulateur de lignes 11

Sources consultées : <https://www.youtube.com/watch?v=Ltqt24SQQoI> <https://www.youtube.com/watch?v=4zHbI-fFIlI>

Firgure 8 : Interface graphique de détection des lignes 14

Firgure 9 : Interface incluant un zoom pour les grandes images 15

Firgure 10 : Résultat possible des deux détections en une seule image 17