PROJET SYNTHÈSE :

RECONNAISSANCE DE SIGNES DE LA MAIN

Par

Vincent Thomassin-Rochon

Présenté à

Jean-Christophe Demers

Cégep du Vieux Montréal

Synthèse

Cours 450-B65-IN, groupe 00001

02 Octobre 2019

Présentation générale

Le but du projet est de créer un programme qui va aider les gens à apprendre ou à perfectionner leurs connaissances du langage des signes via une reconnaissance des placements des doigts d'une main dans une photo prise par eux-mêmes. Pour ce faire, les usagers du programme pourront prendre une photo d'une de leurs mains imitant une lettre du langage des signes. Si le signe est bien fait, alors le programme affichera la lettre représentée. De plus, les usagers pourront consulter une liste d'image démontrant comment reproduire toutes les lettres du langage des signes. Pour finir, ils pourront aussi choisir de tenter d'imiter une lettre, dépendamment du placement des doigts, le système affichera un succès ou un échec. Le programme sera codé en C++ avec l'aide de la librairie Qt.

Présentation précise du projet

Pour entrer dans davantage de détails, le projet sera donc réalisé en langage c++ avec l'aide de la librairie Qt. Bien que la librairie Qt soit aussi disponible dans le langage python, le langage de programmation c++ est plus intéressant, car il va me permettre de manipuler bit par bit les images qui seront traitées lors du déroulement du programme. De plus, il est intéressant, car il va permettre de développer mes connaissances dans ce langage. La librairie Qt sera utilisée pour produire l'interface graphique.

L'interface graphique contiendra l'image prise avant que le traitement de celle-ci débute. Il y aura aussi un bouton pour créer ou récréer la carte d'éclairage et un message disant l'état de la carte (si elle est créée ou non). Comme il faut recréer la carte d'éclairage à chaque moment que l'éclairage change, le bouton sera activé par défaut, et ne seras pas désactivé une fois la carte créer au cas où l'usager voudrait recréer la carte. Il y aura un bouton pour prendre une capture d'image du vidéo représenter à l'écran au même moment. Étant donné que le traitement de l'image ne sera pas instantané, une barre de chargement se retrouvera sur l'interface graphique afin de montrer l'état du progrès du traitement de l'image. La lettre reconnue par le programme ainsi qu'un état (succès ou échec de la reconnaissance de la main) sera affichée afin que l'utilisateur puisse savoir s'il a réussi ou non à faire ce qu'il désirait. Avec l'aide de deux boutons mutuellement exclusif, l'usager

pourra décider s'il souhaite essayer de reproduire le signe d'une lettre ou s'il veut signer une lettre et laisser le programme deviner qu'elle lettre l'usager à signer. Dans le cas où l'usager voudrait signer une lettre, une image de référence de la lettre apparaitra pour guider l'usager. Étant donné certaines contraintes de difficulté, seulement une dizaine de lettres seront disponibles. Les lettres disponibles à l'usager seront : A, B, D, E, F, G, L, M et Y. Ces lettres ont été choisies, car lorsqu'elles sont signées dans le langage des signes français (LSF), les 5 doigts sont reconnaissables et distincts et sans obstruction. Ceci facilitera la reconnaissance des doigts lors du traitement des images. Dans le cas où l'usager voudrait spécifier une lettre qu'il voudrait imiter, un champ de texte pour spécifier la lettre qu'il veut sera affiché.

Pour la prise d'image, une caméra Logitech sera utilisée, car elle se trouve déjà en ma possession. Cependant, l'usager aura aussi la possibilité de faire traiter une image qu'il possède déjà sur son ordinateur. Donc à la place de prendre une image avec une caméra, il pourra simplement ajouter une image au programme et cette dernière se feras traiter. Pour traiter l'image, il faudra la filtrer une première fois afin d'enlever le bruit. Pour ce faire, un filtre médian sera appliqué individuellement pour chaque pixel de l'image. Dépendamment de la taille du filtre, une grille de taille correspondante sera créée autour du pixel souhaité et la valeur médiane de tous les pixels inclus dans cette grille sera placée sur notre pixel visé. Par la suite, une convolution avec une distribution de type gaussienne sera appliquée afin de réduire davantage les bruits de l'image et faciliter le traitement par la suite. Une convolution est un procédé qui multiplie chaque voisin avec son équivalent positionnel d'un filtre puis fait une sommation sur le tout afin de donner la nouvelle valeur RGB du pixel central[1].

Il faut ensuite passer à l'uniformisation de l'éclairage. Pour ce faire, une carte d'éclairage sera créée afin de faciliter la segmentation et la classification éventuelle. La carte sera créée avec l'aide d'une cinquantaine d'images de fond identique ne contenant aucun signe de la main. Ces images passeront ensuite parmi plusieurs filtres ainsi que des fusions afin que de finir qu'avec une seule image d'arrière-plan qui correspondra à la carte d'éclairage. Donc les images passeront dans un filtre de convolution 9x9 avec une distribution normale,

[1] http://www.f-legrand.fr/scidoc/docimg/image/filtrage/convolution/convolution.html

par la suite une moyenne des images sera effectuée afin d'obtenir une image synthèse. Puis l'image passera dans un filtre maximum d'une taille de 9x9. Un filtre maximum prend en considération la valeur du pixel et des voisins formant un carré de la taille du filtre alentour du pixel et garde la valeur la plus élevée de ceux-ci. Par la suite, un autre filtre de convolution avec une distribution uniforme, mais cette fois-ci avec une taille de 151x151. Puis pour finir, une uniformisation et une normalisation. L'uniformisation est le procédé de diviser l'image source par la carte d'éclairage (pixel par pixel) tout en évitant la division par 0. En ce qui concerne la normalisation, le but est d'éviter les cas divergents et s'assurer que la valeur des pixels est dans une plage de valeur acceptable telle que [0,1] ou [0,255]. Pour normaliser, on prend en compte la valeur minimale et maximale de tous les pixels de l'image ainsi que la valeur de normalisation souhaitée. Il s'agit de soustraire la valeur du pixel par la valeur minimale, puis diviser le résultat par la différence entre la valeur maximale et minimale. Il ne reste qu'à multiplier cette dernière valeur par la valeur de normalisation afin d'avoir la valeur normalisée souhaitée. Dans mon cas, je choisis de normalisée pour une plage de valeur de [0,255], car c'est la même plage de valeur que la plage des couleurs RGB traditionnelle.

Afin de poursuivre le traitement de l'image, une segmentation de l'image traitée sera effectuée. Cette segmentation a pour but identifier les pixels de l'image qui correspondent à notre recherche. Dans notre cas, les pixels que nous désirons être capables d'identifier sur l'image sont les pixels verts. Il s'agit des pixels verts étant donné que pour faciliter la reconnaissance des signes, les usagers seront équipés de sphères vertes en mousse sur le bout de chaque doigt et du pouce. La forme sphérique des objets au bout des doigts est très importante, car cette forme implique qu'elle ne sera jamais déformée par la perspective contrairement à la déformation qu'un losange ou un carré pourrait subir. La couleur verte des sphères porte moins d'importance. Elle a été sélectionnée, car elle est différente de tous les tons de couleur de peau possible chez l'humain, et donc les pixels verts ont moins de chance d'appartenir à la main d'un humain. Étant donné que la segmentation va identifier tous les pixels verts, la segmentation va donc éliminer tout ce qui n'est pas un pixel vert en changeant la valeur des pixels non verts pour 0 et la valeur des pixels verts pour 1. Le résultat sera donc une image binaire ou tous les pixels valant 1 seront les éléments recherchés, et les pixels valant 0 sont ceux qui ne sont pas importants au traitement.

Cependant, pour que la segmentation soit effectuée de manière adéquate, il faudra séparer l'image traitée en trois images. Ces trois images correspondront aux bandes rouge, bleues et vertes qui constituent la première image. Lorsqu'un pixel sera dans la plage acceptable pour le rouge, le bleu et le vert. Le pixel sera gardé, sinon le pixel se verra être attribuer la valeur de 0.

Il est possible que certaines « anomalies » se créent lors du traitement ce qui aurait comme résultat une image n'ayant pas des formes sphériques bien définie. Par exemple, une réflexion lumineuse vers la sphère pourrait causer les pixels plus illuminés d'être éliminé lors de la segmentation. Pour résoudre ce problème, un algorithme de remplissage de forme sera appliqué à l'image afin que chaque forme individuelle soit complétée autant que possible si elles ont besoin d'être complétées. La première étape de cet algorithme est d'identifier un pixel qui appartient à l'arrière-plan. Pour ce faire, il faudra trouver le premier pixel du contour ayant plus de voisins pareil le précédent que le nombre de pixels fessant la taille du plus gros trou de la sphère imparfaite. Par la suite, avec un simple algorithme de « flood filling » récursif à 4 voisins tout l'arrière-plan sera attribué la valeur de 1 afin qu'il ne reste que les trous avec une valeur de 0. Une fois l'arrière-plan rempli, toute l'image sera inversée afin que les trous représentent le 1 et que tout le reste de l'image ait une valeur de 0. Ceci nous permettra de fusionner notre image inversée avec notre image traitée de base (l'image avec les trous et les valeurs désirées valant 1) afin d'obtenir une image qui ne contient que des formes pleines et qui a fini d'être traitée. Or, le programme ne s'arrête pas là.

La prochaine étape consiste à sortir toutes les informations appartenant aux sphères de l'image qui a fini d'être traitée. On veut notamment savoir la taille de chaque sphère, son centre approximatif, et la position d'un pixel faisant partie de la forme. Pour ce faire, un algorithme d'extraction de blob sera appliqué à l'image. Cet algorithme d'extraction va appliquer une « flood filling » afin de détecter tous les pixels contigus d'une forme. Par la même occasion l'air occupé par chaque forme sera calculé afin de sauvegarder cette mesure, ainsi qu'une position en X et en Y et une position approximative du centre de la forme. Avec toutes ces informations, une position sera attribuée à chaque forme qui va correspondre à quel doigt elle est supposée représenté.

Peu importe si l'usager a choisi de laisser le système deviner la lettre qu'il signe ou s'il a décidé d'en imiter une, une comparaison entre les données recueillies par l'image traitée et les données d'une base de données doivent être effectuées afin de savoir si l'usager fait le bon signe de la main. La comparaison doit donc vérifier si le placement des sphères sur l'image correspond au placement des sphères dans la base de données. Cependant, les « sphères » de la base de données ne contiendront que les informations sur les angles et les distances minimales que chaque sphère doit avoir avec les autres. Cette façon de sauvegarder des données va éviter les problèmes des différentes tailles de main possible des usagers, car aucune position absolue ne sera définie. L'engin de base de données utilisée sera MySQL et il sera intégré dans le programme. De cette façon, les utilisateurs n'auront pas à installer de base de données par eux-mêmes. De plus, les données des signes (distances et angles) seront déjà incluses lors du téléchargement du programme. MySQL sera utilisé, car je suis déjà familier avec le SQL et MySQL s'implémente relativement bien dans un projet de c++.

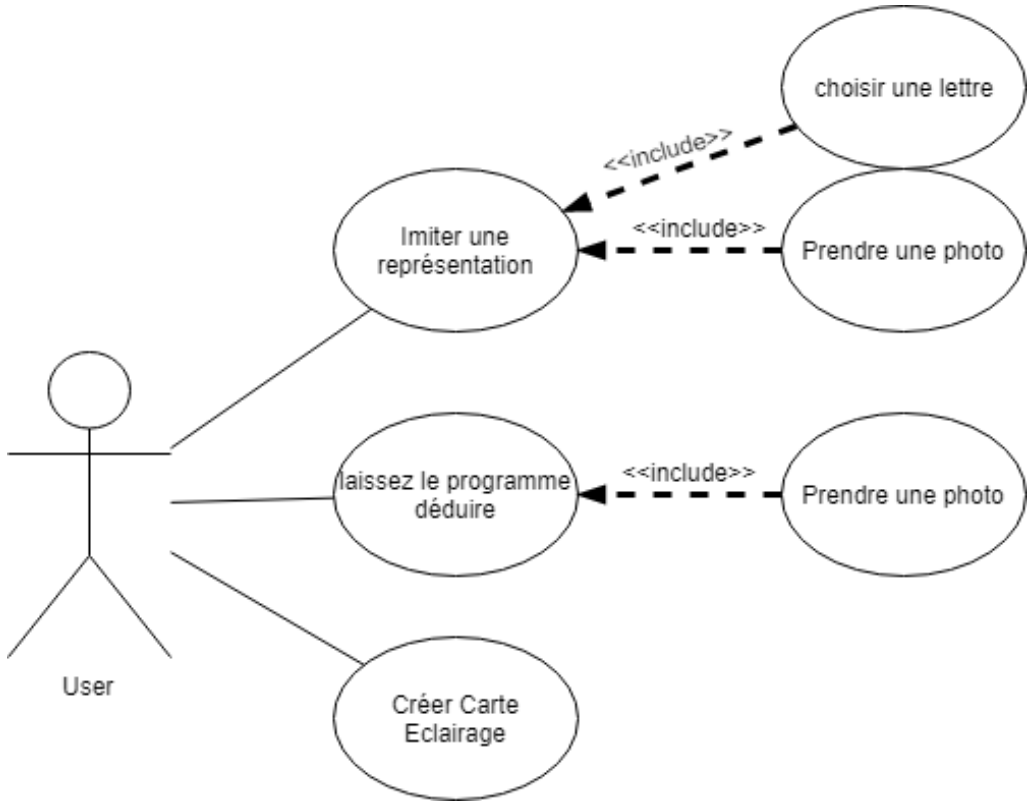
Présentation des patrons de conceptions envisagés

En ce qui concerne les patrons de conceptions, un singleton sera utilisé pour la connexion à la base de données embarquée afin de limiter le nombre de fois qu'une connexion sera fait à la base de données, ou le nombre de fois qu'elle sera instancier.

De plus, le patron de conception « stratégie » sera utilisé afin de permettre la permutation des algorithmes de convolution, de médiane, de segmentation, de remplissage de zone interne et de calcul des descripteurs de forme de l'image lors du processus du traitement de l'image.

Présentation des aspects techniques

Cas d'utilisation



Interfaces usagers

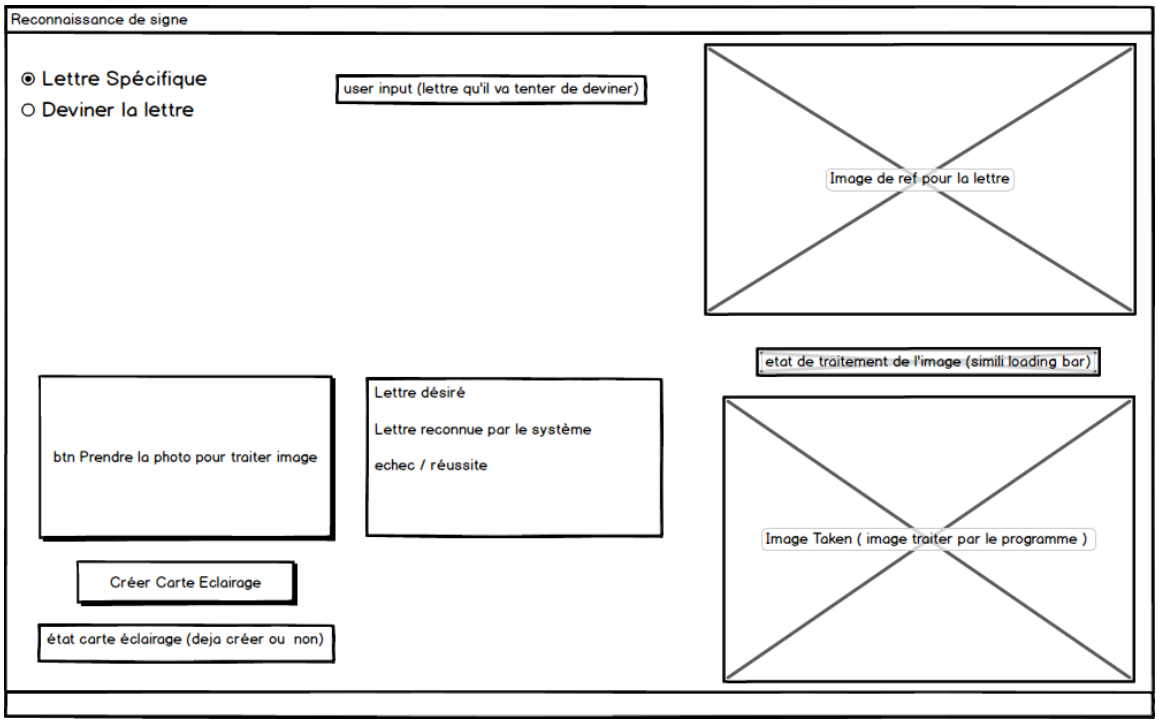


Diagramme de classe UML