FRANÇOIS Tanya 21952540
SAILLY Steven 21954116

Linguistique – Rapport de projet

Présentation et objectif

Ce projet porte sur la reconnaissance optique de caractères japonais. Le japonais présente la caractéristique d'utiliser à la fois des caractères d'origine chinoise (*kanji*), deux syllabaires dérivés des *kanji* (*hiragana* et *katakana*), et, dans une moindre mesure, l'alphabet latin (*rōmaji*). De plus, le japonais ne comprend pas d'espace, et peut s'écrire soit de haut en bas et de droite à gauche (format traditionnel, *tategaki*) ou de gauche à droite et de haut en bas (*yokogaki*). Enfin, certains kanji sont très similaires, voire impossibles à distinguer pour un œil non averti. Nous pouvons notamment citer la paire 土 - 士 (terre, *tsuchi* – guerrier, *shi*) ou le quadruplet か - カ - 力 - 刀 (*hiragana* « ka » - *katakana* « ka » - puissance, *chikara* – épée, *katana*). De ces particularités naissent des difficultés à extraire correctement les caractères d'un texte écrit en japonais.

Nous avons travaillé sur la reconnaissance de l'écriture manuscrite sous deux formes : l'une sur un support numérique, l'autre sur un support traditionnel. D'une part, nous avons réalisé une fenêtre permettant de tracer un unique caractère, et présentant les caractères les plus ressemblants. D'autre part, avant de procéder à la reconnaissance en tant que telle, nous extrayons chaque caractère avant de le soumettre au processus de reconnaissance. Le processus de reconnaissance consiste en la comparaison de l'image du caractère avec des images de référence (voir section « Bibliothèques et ressources utilisées »). Par nature, ce processus est limitant puisque le contexte n'est pas pris en compte.

Notons qu'afin de faciliter l'extraction des caractères sur support traditionnel, nous avons opté pour nous limiter à des textes écrits sur des *genkō yōshi*. Sur ces feuilles à carreaux, chaque carreau correspond à un caractère. Enfin, afin de ne pas alourdir inutilement les répertoires contenant les images de référence, nous nous sommes limités, dans la version finale, à environ 100 caractères sur lesquels la comparaison aura effectivement lieu.

Utilisation

Ce projet a été réalisé sous Java 11. Le répertoire *resources/* ainsi que le fichier *ocr_japonais.jar* sont nécessaires à l'utilisation.

Depuis l'invite de commande et depuis la racine du projet :

- pour la version « support numérique » : *java - jar ocr_japonais.jar*

- pour la version « support manuscrit » : *java - jar ocr_japonais.jar [fichier_à_traiter]*

Le fichier à traiter doit être placé dans le répertoire *resources/handwriting/*. Pour utiliser le fichier d'exemple *resources/handwriting/src.png*, il suffit de ne pas entrer de fichier à traiter.

Structure

Outre la classe *Main* permettant de lancer la version demandée par l'utilisateur, le projet se sépare en cinq classes, dont une partagée.

- La classe *App* permet de lancer l'interface graphique et de la mettre à jour en fonction de ce qu'écrit l'utilisateur. L'interface graphique se sépare en deux parties : à gauche se trouve le panneau où l'utilisateur peut écrire, tandis qu'à droite sont donnés les caractères les plus ressemblants au caractère en cours d'écriture, ainsi que leur taux de ressemblance avec ce caractère. L'attribut *ArrayList<Kanji> kanji* contient les caractères que représentent les images de référence.
- La classe *Kanji* contient notamment les attributs *String kanji* et *double similarity*, qui représente le taux de similarité d'un caractère avec le caractère en cours d'écriture par l'utilisateur. L'attribut *kanji* est initialisé grâce au nom du fichier de l'image de référence que l'objet va représenter : pour tout fichier *f* représentant un caractère *c*, le nom de *f* est le codage Unicode de *c*.
- La classe *SVGtoPNG* contient toutes les méthodes permettant de rendre utilisables les images de référence.
- La classe *PngManip* contient les méthodes qui permettent de passer de l'écriture sur l'interface graphique à un fichier PNG, et de calculer la similarité entre deux images. Le calcul de similarité est effectué en itérant sur les pixels de l'image de référence et consiste en la comparaison de la transparence des pixels, ce qui est possible car nous utilisons des fichiers PNG transparents, ne laissant opaque que les pixels appartenant au caractère écrit. Afin de réduire la marge d'erreur liée au décalage de l'écriture dans la case par rapport au caractère de l'image de référence, on prend aussi en compte les pixels situés autour du pixel ciblé.
- La classe *Handwriting* permet d'extraire, via la transformée de Hough, les caractères présents dans l'image. Les caractères sont ensuite écrits dans un fichier TXT situé dans le répertoire *resources/handwriting/*.

Bibliothèques et ressources utilisées

Les images de références sont issues du projet KanjiVG.

Afin de réaliser au mieux les différentes fonctionnalités, trois bibliothèques ont été utilisées.

- Apache Batik : pouvoir convertir aisément les images de référence, qui sont initialement au format SVG ; en revanche, le retrait des éléments inutiles des images de référence, comme l'ordre de tracé des traits, a été effectué par un simple parsing du fichier original, en retirant les lignes de code que contenaient les balises d'id *kvg:StrokeNumbers[…]*.
- JavaFX : réalisation de l'interface graphique ; nous l'avons préférée à l'utilisation de la bibliothèque Swing pour sa facilité d'utilisation.
- OpenCV : extraction des caractères depuis une image ; c'est à partir de sa méthode permettant d'appliquer la transformée de Hough que nous reconnaissons les lignes formant les carreaux du *genkō yōshi* : certains croisements de lignes correctement choisis permettent de retrouver les coordonnées à extraire.

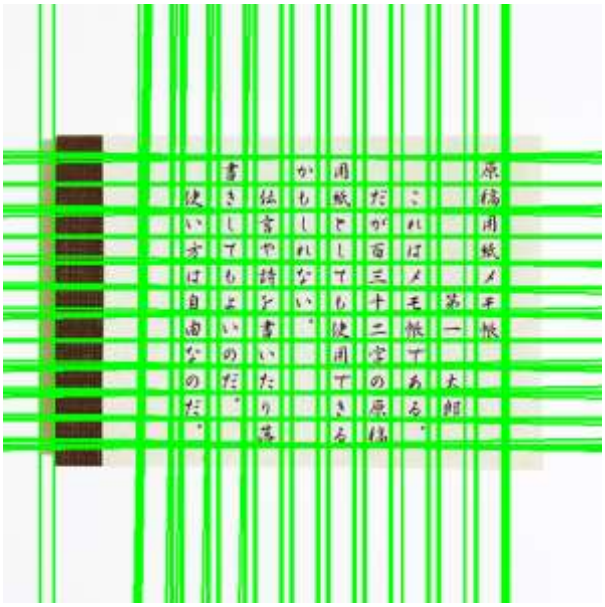

Figure 1 : Tracé des lignes issues de l'algorithme de Hough sur le fichier *resources/handwriting/src.png*

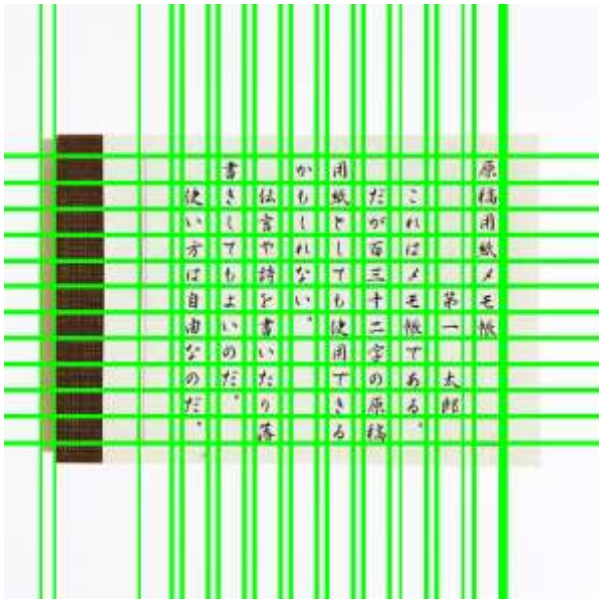

Figure 2 : Tracé affiné

Résultats et limites

Tout d'abord, la partie concernant la reconnaissance d'un caractère écrit sur support numérique s'avère passable : si certains caractères, ou certaines clés, semblent correctement identifiés, de nombreuses erreurs restent identifiables. Avec une gravité moindre, il est courant que le caractère correct ne soit pas identifié comme le caractère le plus similaire à celui écrit par l'utilisateur mais fasse tout de même partie des *kanji* proposés. Il est très plausible qu'en agrandissant la liste de caractères proposés, le caractère correct ait une plus grande probabilité d'apparaître dedans. Cependant, cela revient à réduire la précision de notre programme, tout en laissant l'utilisateur décider lui-même du caractère le plus ressemblant à celui qu'il aura écrit. Ce fonctionnement peut être admissible dans le cadre du développement d'une application de recherche de *kanji*.

Ensuite, la partie concernant la reconnaissance d'un texte écrit sur support traditionnel, si elle permet effectivement d'extraire chaque caractère, ne prend pas en compte le sens d'écriture. De plus, son manque d'optimisation ainsi que les nombreuses opérations nécessaires pour aboutir à l'écriture du texte reconnu dans un fichier – multiples conversions d'images, transformée de Hough, création des fichiers contenant les caractères, calcul de similarité, puis écriture du caractère le plus similaire – rend l'opération coûteuse. De surcroît, puisqu'il faut que les images à comparer lors du calcul de similarité aient la même taille afin de pouvoir comparer les pixels un à un, nous avons fait le choix de modifier uniquement les fichiers des caractères à reconnaître. Toutefois, dans le cas d'un fichier de dimensions inférieures à celles du fichier de référence, le redimensionnement effectué par une méthode de la bibliothèque *OpenCV* peut conduire à l'impossibilité de reconnaître le caractère. Ainsi, une meilleure solution serait de comparer les dimensions de l'image de référence et celles du caractère à reconnaître, puis de modifier le fichier aux dimensions les plus petites.

Enfin, dans les deux cas, le calcul de la similarité étant effectué de la même façon, il paraîtrait raisonnable de le revoir afin d'améliorer la précision de la reconnaissance.

Extensions envisageables

Concernant l'interface graphique, il pourrait être utile d'ajouter une fonction de retrait du dernier trait tracé. Pour mettre en œuvre cette extension, en raison de la nature des objets utilisés pour représenter un trait, il faudrait connaître le nombre d'objets *Path* utilisés pour chaque trait.

Lorsqu'un caractère – pour la version « support numérique » – ou bien un mot – pour la version « support traditionnel » – est détecté, il pourrait être utile d'afficher des informations sur le caractère ou le mot, en utilisant un dictionnaire local ou en ligne, tel que jisho.org. Dans le cadre d'un mot, cela permettrait de mettre en lumière la difficulté du parsing liée à l'absence de caractère d'espacement.

Dans le cadre de la reconnaissance d'un texte, il serait envisageable de fournir une traduction. Ce type de fonctionnalité pourrait par exemple s'avérer utile pour un apprenant autodidacte souhaitant vérifier sa propre compréhension. Dans un cadre plus universitaire, nous pourrions envisager de comparer plusieurs traductions, par exemple issues de DeepL ou du modèle mBART-50 many to many multilingual machine translation[1], afin d'évaluer les performances de ces systèmes.

[1] Tang, Yuqing, et al. (2020) Multilingual translation with extensible multilingual pretraining and finetuning. *arXiv preprint arXiv:2008.00401*