GIF-4100 : Vision numérique
Projet de trimestre
La feuille électronique

Olivier Beaulieu,
Frédéric Bernard,
Kento Otomo-Lauzon

19 décembre 2018

0.1 Solution proposée

La solution proposée est constituée d'un projecteur et d'une caméra. Le projecteur est disposé de manière à illuminer une table, et la caméra sert à détecter la position de feuilles blanches et d'un doigt.

0.1.1 Fonctionnement général

Le système permet à un utilisateur, en utilisant son doigt, de dessiner sur une feuille en temps réel. En localisant la position du doigt et de la feuille blanche sur la table, le système superpose le tracé effectué par l'utilisateur sur la feuille. Dans le cas où plusieurs feuilles sont présentes sur la table, le dessin est dupliqué sur chaque feuille. Le dessin étant défini de manière relative à la feuille, celui-ci demeure à l'endroit, et ce, même si l'utilisateur déplace la feuille ou lui fait effectuer une rotation.

Finalement, le dessin est visible depuis une application web, laquelle est accessible depuis un ordinateur.

Dans un contexte de télétravail, on pourrait imaginer un scénario d'utilisation selon lequel deux collègues sont disposés autour de la table en ayant chacun une feuille. Les membres physiquement présents à la réunion pourraient participer à une séance de remue-méninges en dessinant sur leur propre feuille, et le résultat serait automatiquement reflété sur les feuilles des autres participants. De plus, des participants présents à distance pourraient effectuer le même exercice en se connectant à l'application web.

0.1.2 Description de la méthode employée

Cette section décrit les aspects techniques nécessaires à la réalisation du projet. L'application des algorithmes de vision artificielle est faite à l'aide de la bibliothèque *OpenCV*, l'affichage graphique pour le projecteur est faite au moyen du module *Python Tkinter*, l'application web est faite avec *VueJS* et la gestion du serveur et du flot d'exécution est faite par le *framework Jivago*.

Détection de la feuille blanche

La figure 1 montre une vue qu'aurait la caméra du la surface de travail. Les paramètres de saisie de la caméra ont été choisis de sorte à faire ressortir la surface blanche sous l'éclairage dominant du projecteur. Par la suite, un seuillage sur les valeurs d'illuminance est effectué, de sorte à ne conserver que les pixels dont la « valeur » dans le repère HSV est supérieure à 150, et ce, peu importe la valeur de *hue* ou de *saturation*.

Ensuite, on identifie le rectangle englobant d'aire minimale pour obtenir l'orientation de la feuille et la coordonnée des coins, dans le repère caméra. Un exemple de détection est présenté à la figure 2.

Détection du doigt

Similairement, la détection du doigt s'effectue par un seuillage appliqué sur la valeur HSV des pixels. Pour le moment, un seuil est prédéfini de manière à reconnaître la main de l'un des présentateurs. La position du doigt elle-même est choisie comme le point les plus éloigné du centre

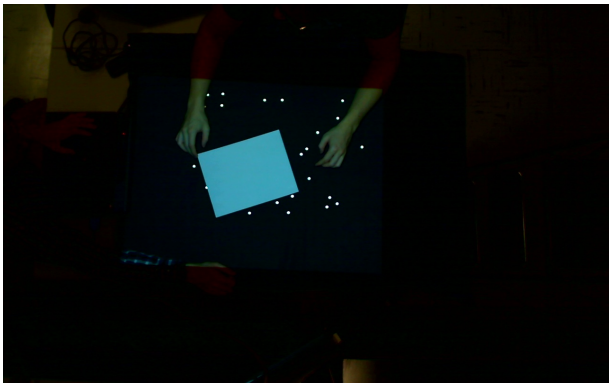

FIGURE 1 – Vue de la caméra

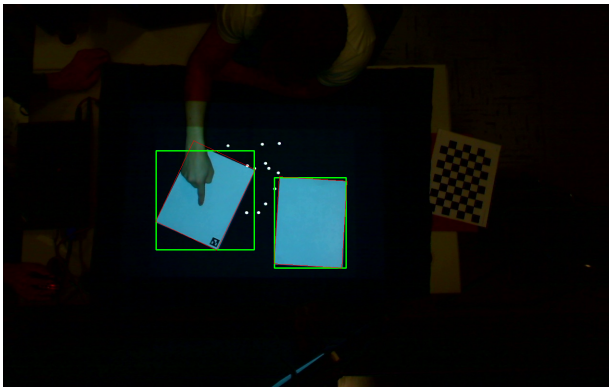

FIGURE 2 – Délimitation des feuilles blanches. Le rectangle rouge correspond au rectangle englobant d'aire minimale.

de masse de l'amas de pixels représentant le bras de l'expérimentateur. La figure 3 illustre le résultat de cette procédure.

Homographie entre l'image projetée et l'image capturée par la caméra

La calibration initiale de la caméra est effectuée au moyen d'un damier. De cette manière, nous sommes en mesure d'annuler les effets de la distorsion sur les images captées.

Afin d'effectuer la conversion des coordonnées entre les repères de la caméra et du projecteur, l'appariement initial des points pour le calcul de l'homographie est effectué au moyen d'un marqueur *ArUco*. Avant de débuter la démonstration, le script de calibration est lancé, et projette un marqueur. L'appariement des quatre coins est effectué en utilisant les algorithmes de détection d'*ArUco* disponibles dans *OpenCV*.

Logique de gestion de dessin

Le dessin à afficher est sauvegardé de manière vectorielle. Chaque ligne est enregistrée individuellement, et les distances sont encodées en millimètres, et sont calculées en utilisant la définition

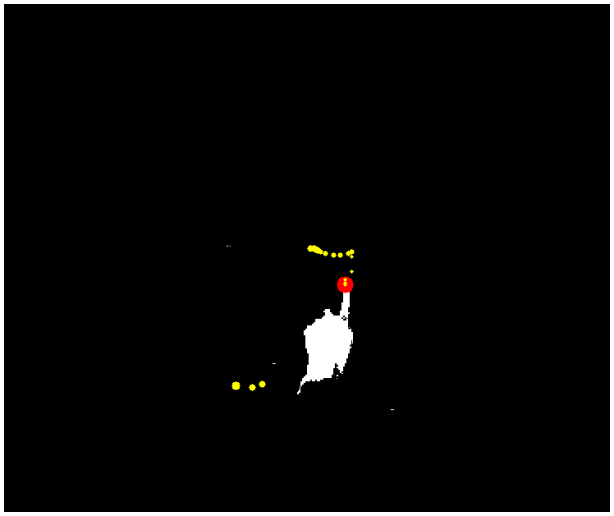

FIGURE 3 – Filtre appliqué pour détecter le doigt. Le point rouge correspond à la position identifiée pour le doigt, et les points jaunes correspondent aux positions passées.

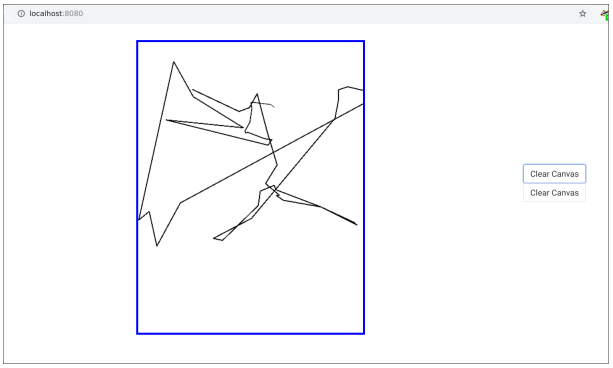

FIGURE 4 – Affichage du canevas en direct depuis un navigateur web.

du format de papier *US Letter* (8.5po x 11po). Cette gestion permet de maintenir la taille du dessin, dans le cas où la feuille est placée plus ou moins loin du projecteur.

Un *thread* en arrière plan se charge de maintenir à jour le canevas qui est projeté, de sorte que l'affiche s'effectue en temps réel.

Application web de visualisation et d'administration

Enfin, plusieurs routes d'API REST sont exposées au moyen d'une application *Jivago*, ce qui permet à une application web de visualiser en direct le dessin. L'application web elle-même, récupère l'information du canevas à chaque seconde pour l'afficher (figure 4).

0.2 Performances

La plus grande force de notre système est que le traitement s'effectue en temps réel. En effet, la détection des feuilles blanches et du doigt sont relativement peu complexes, et nous permettre de mettre à jour en temps réel l'affichage pour se conformer aux mouvements de l'utilisateur.

De plus, la détection de la feuille et du doigt sont plutôt précises, en tenant compte des limites matérielles. La localisation des feuilles, lorsque celles-ci sont immobiles, ne varie que d'un ou deux pixels, de manière générale. De même, la détection du doigt, en considérant la taille relativement inférieure de celui-ci par rapport aux autres éléments de la scène, est assez précise pour être utilisable dans notre application. On reconnaît, somme toute, le tracé effectué par le démonstrateur.

Par ailleurs, l'homographie qui lie les points de l'image captée à ceux des projetés s'effectue de manière rapide et précise. En effet, la correspondance entre les feuilles est cohérente et stable, et ce, peu importe la position de celles-ci sur la table. À titre de validation supplémentaire, le script de calibration affiche des points spécifiques en utilisant la matrice d'homographie lorsque le calcul initial est terminé, ce qui permet de juger en un coup d'œil de la validité de la matrice crée. Puisque la calibration est effectuée avant chaque démonstration et que le montage est fixe, il n'y a pas de réelle perte de précision durant cet intervalle de temps.

0.3 Améliorations possibles

Toutefois, le système comporte aussi son lot de failles et d'améliorations potentielles. Par exemple, malgré que le système soit utilisable en « temps réel », il existe une latence entre le moment où l'action est effectuée et l'apparition de la rétroaction à l'écran. Une partie de cela est dû à la relative lenteur de la prise d'image (limitée entre 10 et 15 fps dans notre cas), et le va-et-vient entre la logique de haut-niveau effectuée en *Python* et les routines de calcul matriciel d'*OpenCV* et de *numpy*. Cette seconde cause est visible puisque l'on observe un délai entre le tracé de la première ligne du dessin et la dernière, alors que les calculs s'effectuent sur la même image.

À ce sujet, une première amélioration pourrait être de réduire la résolution de prise d'image, ce qui aurait pour effet premier d'augmenter la vitesse de capture, par exemple à 30 fps dans le cas la caméra utilisée. Toutefois, il faudrait chercher le compromis entre la vitesse de traitement et de précision dans la localisation. Pour les besoins de la démonstration, il a été jugé valable de laisser le tout à la résolution de 1280x800.

On pourrait aussi chercher à minimiser les copies de matrices et les allers-retours coûteux entre les matrices *numpy* et les objets *Python*. À ce stade-ci, les gains en performance sont de basse priorité par rapport aux autres améliorations possibles.

Pour améliorer la précision du système, notamment en ce qui concerne la localisation et la détection du doigt, une première amélioration consisterait en la sélection d'un seuil de couleur en fonction de l'expérimentateur. Pour l'instant, les seuils étant prédéfinis dans le code, il est difficile de tenir compte des différences entre plusieurs utilisateurs. Dans certains cas, le teint différent d'un utilisateur entraîne une forte instabilité dans la détection, allant jusqu'à rendre le système inutilisable. Ce phénomène est présent même pour des petites variations entre les membres de l'équipe. À cet effet, une approche pourrait être de choisir un seuil de couleur au démarrage de l'application, où le système pourrait demander à l'utilisateur de montrer sa main, pour qu'il puisse choisir un seuil en fonction de l'histogramme des valeurs d'illuminance des pixels.

Une autre approche serait d'explorer les méthodes de séparation d'arrière-plan, plusieurs desquelles sont incluses dans *OpenCV*. Par exemple, la librairie propose des approches par MoG (*Mixture of Gaussian*) pour effectuer la segmentation de l'arrière-plan [1]. Dans ce cas, une difficulté dont il font tenir compte est que les feuilles peuvent se déplacer dans la scène, ce qui pourrait les identifier, de manière erronée, comme faisant partie de l'avant-plan.

Finalement, la détection du doigt, dans son implantation actuelle, ne tient pas compte de la distance qui sépare le bout du doigt de la surface de la table. Ce problème mène régulièrement à l'ajout de traits non intentionnels sur la surface de dessin. Pour remédier à la situation, l'utilisation d'une seconde caméra, placée plus près de la table, pourrait permettre l'amélioration de la détection. La localisation serait toujours effectuée par la caméra monde, mais la détection du clic pourrait être effectué selon ce deuxième point de vue.

1. `https://docs.opencv.org/3.1.0/d1/dc5/tutorial_background_subtraction.html`