université de BORDEAUX

Projet de fin d'études

Discrete Finger Pose Estimation for On-body Touch Interaction Through a Wearable Depth Sensor

Auteurs :
Carpentier Dylan
Maria Théo
Pacaud Maxime
Pedeau Adrien

Client :
Darbar Rajkumar

30/03/2020

Table des matières

Chapitre 1

Contexte

Dans le cadre de notre master informatique, nous avons été amenés à créer un système de reconnaissance de contact doigt-surface, ainsi que la reconnaissance de la rotation de celui-ci. Le projet a notamment pour objectif d'être ensuite utilisé pour contrôler une interface, à l'aide d'un pico-projecteur ainsi que d'une caméra de profondeur qui permettront d'utiliser un avant-bras comme surface de détection et de projection. L'ensemble pourra donc être utilisé de la même manière qu'un écran tactile de smart-phone. Ceci a pour avantage d'offrir un retour haptique lors de l'utilisation de l'interface, ce qui n'est pas forcément le cas dans le domaine des réalitées augmentées à l'heure actuelle. Dans ce projet, nous ne nous occupons pas de la partie projecteur qui ne sert qu'à offrir un retour visuel à l'utilisateur, mais uniquement à la partie de détection en utilisant comme matériel une Kinect V2. Cet appareil ayant un coût relativement peu cher et étant accessible facilement, il est très fréquemment utilisé dans des travaux de suivi visuel comme le notre. Nous avons dans un premier temps tenté d'implémenter la méthode utilisée dans l'article OmniTouch [3] pour ce qui est de la détection du doigt au dessus d'une surface ainsi que la détection du doigt en contact avec la surface. Pour ce qui est de la partie de détection de la rotation du doigt en contact, notre client a instauré une contrainte qui consiste à se baser sur un réseau neuronal.

Chapitre 2

Existant

Si nous cherchons à implémenter Omnitouch[3] c'est qu'il s'agit d'une solution simple et efficace, il s'agit d'un papier datant de 2011 et ne correspondant pas à l'état de l'art actuel. Dans le cadre de la segmentation et du suivi de mains et doigts en n'utilisant qu'une caméra de profondeur sans autres accessoires (marqueurs de couleurs sur les doigts, bracelet magnétique, etc), l'article de Duong Hai Nguyen et al.[4] publié en 2019 est un des plus récents en la matière.

Contrairement à Omnitouch qui utilise simplement du traitement d'image de l'image de profondeur, ce papier utilise un réseau d'apprentissage profond basé sur SegNet. Celui-ci est utilisé dans deux tâches : détecter les doigts ainsi que les bouts de ceux-ci. Comme son nom l'indique, ce réseau est spécialisé dans la segmentation d'images et la découpe en différentes sous-zones selon l'entraînement reçu. Cependant, ce papier traite uniquement de suivi et de segmentation de main, et pas de détection de survol et de toucher de poignet.

Sur la question du machine-learning, et particulièrement concernant les jeux de données, nous nous sommes basé sur une revue d'études visant à répertorier les solutions actuelles pour l'estimation de position des doigts. [1] Ainsi, nous avons pu trouver un jeu de donnée similaire à ce que nous recherchons, le First-Person Hand Action dataset (FHAD) [2]. Celui-ci contient plus de 100000 images (en version RGB, et en profondeur) de mains à la première personne, ainsi que d'actions effectuées avec divers objets. Il est accompagné d'un papier décrivant plusieurs approches intéressantes utilisant des technologies de Deep-Learning.

Chapitre 3

Besoins

Les besoins de ce projet peuvent être définit selon les actions suivantes :
— Détecter la position du doigt sur l'avant-bras
— Détecter si le doigt survole l'avant-bras ou s'il touche celui-ci
— Détecter l'angle du doigt (ongle dessus, ongle dessous, côté)

Afin de réaliser ces besoins principaux, nous avons dû réaliser des tâches détaillées dans la partie Implémentation. Cependant, nous avons en plus implémenté un module de sauvegarde et de chargement d'images de profondeur afin de pouvoir utiliser les suites d'images dont nous connaissions déjà les coordonnées et actions qui s'y passaient au préalable. Cela nous permet d'établir différents jeux de test, qui pourront être utiliser pour vérifier le bon fonctionnement de notre application après chaque ajout de fonctionnalité.

Chapitre 4

Architecture

Pour la conception de notre logiciel, nous avons fait le choix de nous orienter sur une architecture de type modulaire. Notre programme se base sur un ensemble d'opérations, qui sont organisées en une chaîne de traitement. Ces opérations sont séparées en différentes classes, qui fonctionnent comme des blocs, que l'on peut ajouter les uns après les autres, et qui nous permettent d'effectuer les différentes opérations sur les images dans un ordre particulier.

Ainsi, nous avons différents modules implémentés dans l'application :

— un module qui permet de récupérer l'image de la caméra de profondeur et de la convertir en matrice
— ainsi qu'un module d'affichage qui permet d'afficher l'image à partir d'une matrice du même type que celle récupérée dans le précédent module
— un module permettant d'effectuer un seuillage (threshold) de l'image
— un module permettant de calculer la dérivée de l'image
— un module permettant d'effectuer la détection des doigts

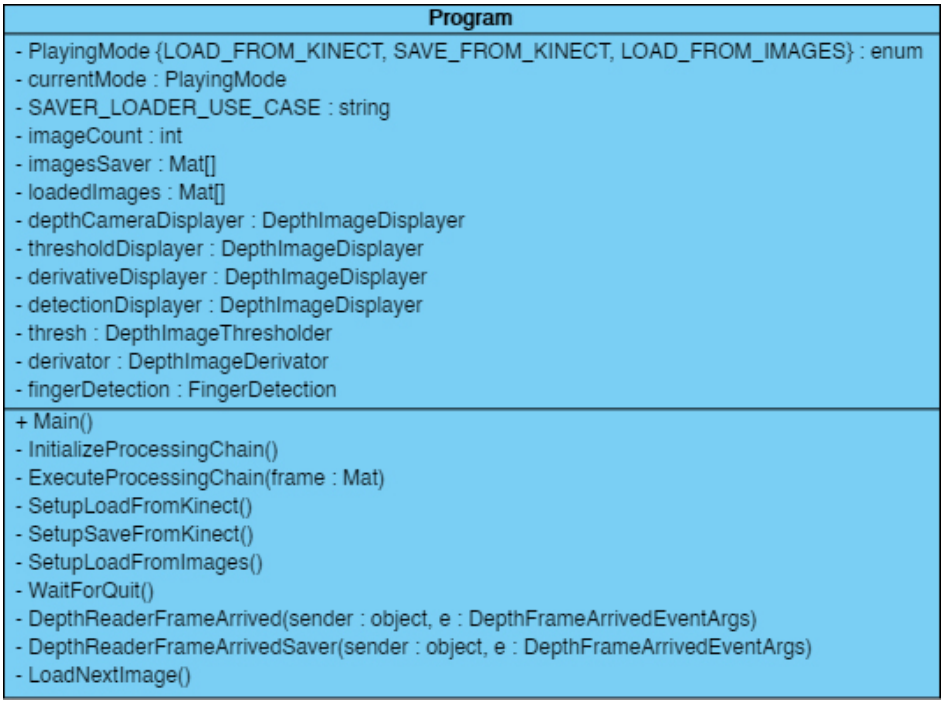

FIGURE 4.1 – Classe Program

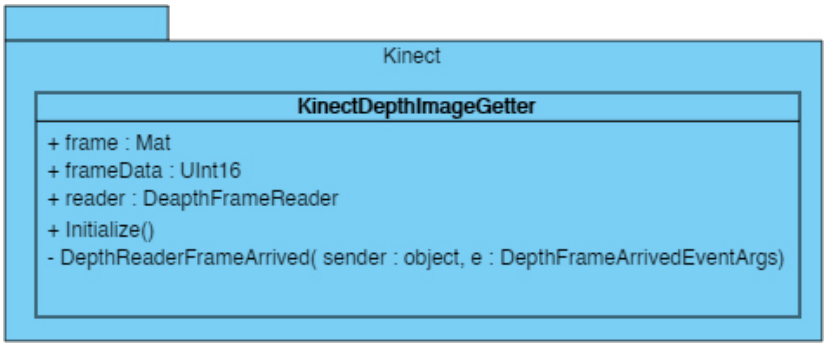

FIGURE 4.2 – Module Kinect

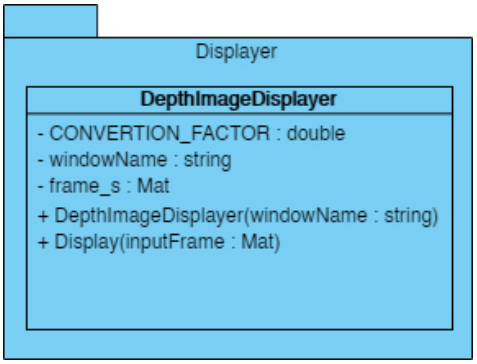

FIGURE 4.3 – Module Displayer

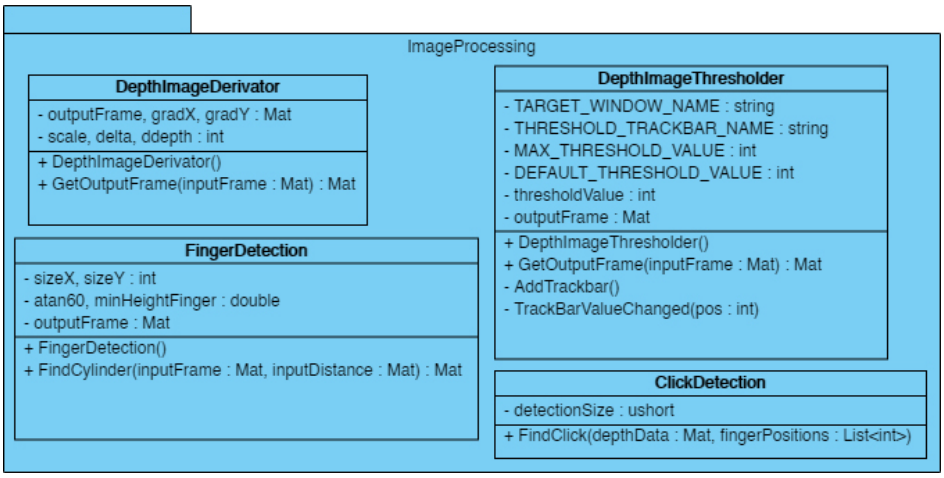

FIGURE 4.4 – Module ImageProcessing

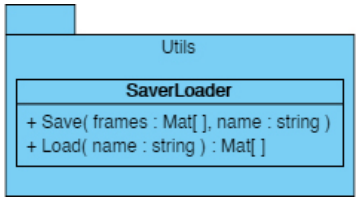

FIGURE 4.5 – Classe SaverLoader

Chapitre 5

Implémentation

Dans un premier temps, il a fallu déterminer quel langage et quel environnement utiliser. Pour ce faire nous avons dans un tout d'abord cherché a utiliser l'environnement de Unity3D, mais celui-ci était plus complexe que de simplement utiliser du C# avec la SDK de la kinect pour la récupération des images de profondeur, puis la bibliothèque OpenCV pour permettre l'affichage des images collectées. Nous avons choisi d'utiliser l'environnement Visual studio par commodité. Il est donc nécessaire que le logiciel puisse détecter la position d'un doigt dans l'espace, et plus précisément si celui-ci se situe en contact avec l'avant bras ou non. Il est ensuite nécessaire de déterminer la rotation du doigt lorsque celui-ci est en contact avec l'avant bras. L'ensemble du projet est organisé sous forme de modules qui seront détaillés ci-dessous.

5.1 Chaîne de traitement de l'application

Afin de pouvoir déterminer quel traitement sera appliqué sur chaque image, nous avons implémenté un système de chaîne de traitement. Pour chaque nouvelle image récupérée par la caméra, l'image est envoyé dans la chaîne de traitement, qui va faire passer l'image dans les différents modules, dans un ordre pré-établi. Il y a une possibilité d'afficher les images entre chaque étape en appelant un module d'affichage entre ces étapes.

5.2 Collecte des images

Dans un premier temps, il a fallu récupérer les images en provenance de la kinect dans un format sur lequel on puisse travailler de manière simple et efficace. Nous avons donc lié la bibliothèque OpenCVSharp à la SDK de la kinect afin de récupérer les images de profondeur sous forme de matrices. Cette première partie se fait aisément à l'aide des exemples d'utilisations fournit.

5.3 Traitement des images

Le traitement de l'image de profondeur se divise en trois grandes parties qui correspondent au étapes citées dans l'article OmniTouch.

5.3.1 Dérivation de l'image de profondeur

La dérivation est effectuée sur un patch de taille 5x5 sur toute l'image afin de faciliter la détection des doigts par la suite. Elle est faite simplement à l'aide des fonctions intégrées à la bibliothèque OpenCVSharp. Le résultat obtenue est un gradient de l'image de profondeur.

5.3.2 Détection de la position des doigts

En suivant la méthode utilisée dans l'article, il n'est possible de détecter que des doigts dans une position horizontale. Cette méthode consiste à rechercher verticalement des "tranches" de doigts d'une taille comprise en 5mm et 25mm, ce qui couvre la majorité des tailles de doigts communs. Pour se

faire on effectue un simple parcourt d'abord horizontalement, puis verticalement à la recherche d'une valeur très élevée (fort pas dans le gradient) qui correspond au début d'un doigt, puis on continue jusqu'à tomber sur une valeur très négative ou si la taille dépasse celle d'un doigt. L'inconvénient majeur de cette méthode est le fait que n'importe quel objet ressemblant un tant soit peu à un doigt sera détecté comme tel (ex : Un stylo). Une fois une tranche trouvée, on stocke dans une liste dans l'ordre suivant x (de départ), y début (de départ), y fin (de départ), x (de fin), y début (de fin) et y fin (de fin). On supprime ensuite les faux positifs en vérifiant qu'au moins 5 tranches soient consécutives pour les considérer comme un doigt, puis on retourne le résultat final.

5.3.3 Détection d'un clic

La partie détection du clic à été simplifié et considère un clic peu importe le doigt, pour retourner simplement si un clic à été effectué. La seconde simplification réside dans le fait que l'on ne vérifie que si les distances extrémités gauches et droites du doigt sont quasi identiques aux distances récupérées autour d'eux.

5.4 Retour visuel

La bibliothèques OpenCVSharp permet aussi un affichage facile des matrices utilisées. Cela permet d'avoir un retour visuel des images et des résultats obtenus tels que le résultat de la détection des doigts.

5.5 Possibilités d'implémentation pour la solution machine-learning

Bien que nous n'ayons pas implémenté la solution utilisant le deep-learning, nous avons pu analyser les différentes solutions proposées par les personnes ayant publié le dataset FHAD. [2]

5.5.1 Choix du modèle du réseau

Dans ce papier, la solution de base proposée est d'utiliser un réseau de neurone récursif avec un module long-short term memory (LSTM). Il s'agit en effet d'un type de réseau très utilisé dans la détection de mouvements sur des suites d'images.

La autres solutions proposées correspondent bien plus à l'état de l'art. En effet, ils ont par exemple essayé une solution qui permet de combiner une approche spatiale et temporelle, à l'aide d'un réseau à deux flux. Cependant, la méthode qu'ils considèrent comme offrant les meilleurs résultats est celle se basant sur le calcul de matrices de Gram pour détecter la position des doigts. [5]

5.5.2 Description du jeu de donnée

Quelques soient les actions effectuées et peu importe l'objet utilisé sur les images, chaque image est annotée avec la position des différentes articulations de doigts, ce qui rend cette solution intéressante pour notre projet. Pour chaque séquence, on dispose d'un fichier texte contenant les coordonnées 3D de ces positions.

Chapitre 6

Tests

Le traitement des images doit être suffisamment rapide pour permettre une exécution en temps réel. Le logiciel devra aussi pouvoir s'adapter aux couleurs de peau afin de pouvoir être utilisé par n'importe quel utilisateur. De même que la luminosité ne doit pas gêner le bon fonctionnement du logiciel (Un minimum et un maximum de luminosité seront tout de même fixé).

Nous n'avons pas eu le temps de mettre en place les tests suivants par manque de temps et dû aux circonstances exceptionnelles, cependant nous voulions faire les 3 types de tests suivants.

6.1 Tests de performance

6.1.1 Test de rapidité

Notre application ayant pour but de tourner en temps réel, il est nécessaire qu'il tourne au moins à 24FPS pour pouvoir traiter le flot d'images envoyées par la caméra de profondeur. Afin de faire cela, il est nécessaire de fixer une certaine architecture de PC et de tester sur celle-ci tout le temps afin d'avoir des résultats constants et comparables.

6.2 Tests de validation

Grâce au module de sauvegarde et de chargement d'images, il nous est possible de charger des vidéos dont nous connaissons parfaitement le comportement à l'avance. Ainsi, il nous est possible de tester les actions suivantes et de vérifier que nos algorithmes ne régressent pas dans le temps.

6.2.1 Validation du matériel

Il est tout d'abord nécessaire de vérifier que le matériel est bien fonctionnel afin de pouvoir tester notre code et être sûr que les éventuels problèmes viennent de celui-ci. Afin de faire cela, nous pensions principalement tester si la caméra de distance fonctionnait précisément en posant des objets à des distances connues de la caméra et en vérifiant si les distances estimées renvoyées par celle-ci étaient corrects.

6.2.2 Détection du doigt en survol

Via la vidéo de survol que nous avons, il nous aurait fallu traiter celle-ci via notre algorithme et comparer le résultat avec un fichier annoté à la main précisant que le doigt ne touche pas l'avant-bras ainsi que la position estimée du survol. Ce fichier aurait précisé pour chaque image de la vidéo quel était l'état de la détection (pas de doigt, survol, pression) ainsi que la position de celui-ci. Si nous avons pu récupérer des vidéos, les annoter à la main n'a pas pu être fait dans les temps.

6.2.3 Détection du doigt en pression

Nous aurions utiliser la même méthodologie que le test précédent, en changeant le fichier vidéo ainsi que celui annoté à la main.

6.2.4 Détection de la rotation du doigt

Dans le cadre de la détection du doigt, nous aurions simplement envoyé des images de doigts sous différents angles (dessus, dessous et côté) et comparé le résultat de l'algorithme avec le notre, cela nécessitant encore une fois un travail d'annotation, mais bien moindre cette fois.

6.2.5 Vérification de paramètres extérieurs

Même si cela n'est pas censé influer une caméra de profondeur, tester si la luminosité, la couleur de la peau influent sur les résultats aurait été nécessaire afin de prouver que notre application fonctionne sur une diversité de situation et non pas un seul cas spécifique.

6.3 Tests de profilage

Aussi, faire du profilage et optimiser les parties appelées de nombreuses fois aurait été judicieux afin d'optimiser le besoin de rapidité que nous avons.

Chapitre 7

Démonstration du logiciel

Comme expliqué précédemment, notre application fonctionne suivant plusieurs modules, symbolisant plusieurs étapes du traitement. Nous allons donc présenter ici une partie de ces modules, ainsi que leur résultat.

7.1 Affichage de la caméra de profondeur

Parmi les différents modules que nous avons implémenté, l'un d'eux permet d'afficher une image de profondeur. Cela nous permet d'avoir un retour visuel en temps réel à chaque étape du traitement. Celui-ci prend en entrée une image de profondeur, et l'affiche dans une nouvelle fenêtre.

7.2 Démonstration de la fonctionnalité de threshold

La première étape de traitement que nous appliquons est une opération de seuil. En effet, celle-ci permet de "nettoyer" l'image de son arrière plan, qui peut être indésirable pour la détection. Comme indiqué sur les figures 7.1, 7.2, 7.3, à l'aide d'un curseur, l'utilisateur peut choisir en temps réel la limite à partir de laquelle l'image de profondeur sera coupée.

FIGURE 7.1 – Seuillage faible

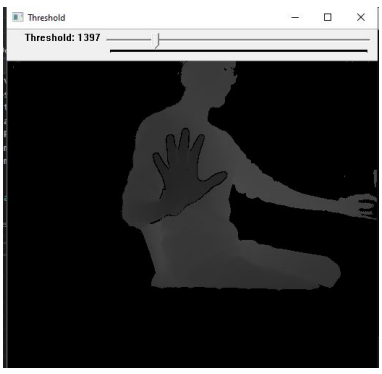

FIGURE 7.2 – Seuillage modéré

FIGURE 7.3 – Seuillage fort

7.3 Dérivée de l'image

Pour permettre la détection, il nous faut calculer la dérivée de l'image. Pour ce faire, on applique un flou gaussien, ainsi qu'un filtre de Sobel sur notre image. Ainsi, on obtient en sortie une image contenant les contours des doigts.

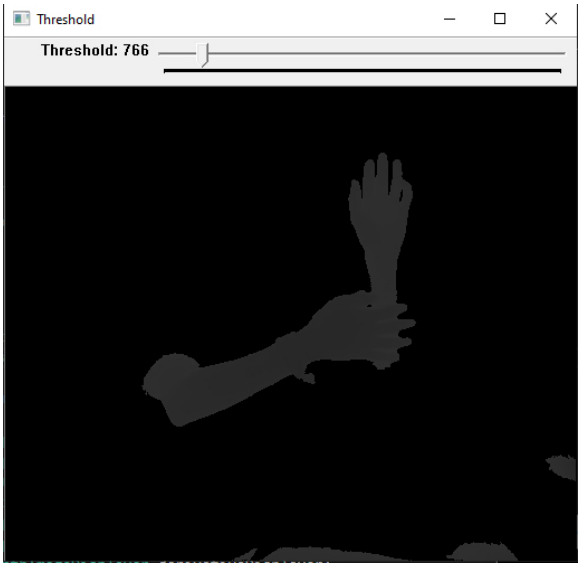

FIGURE 7.4 – Image avant la dérivée

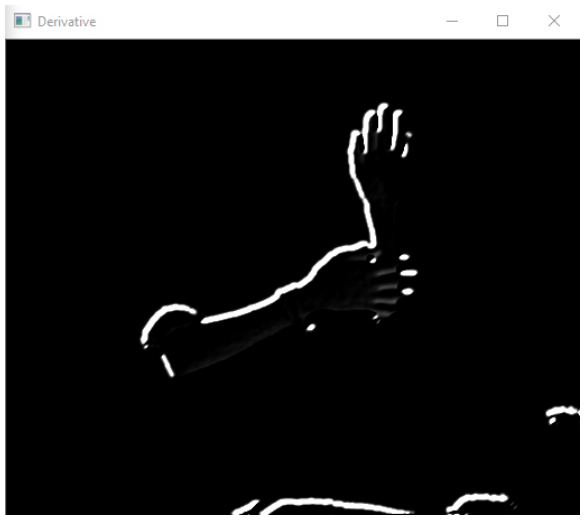

FIGURE 7.5 – Image après la dérivée

7.4 Détection des doigts

Comme indiqué plus tôt, la détection des doigts n'est à l'heure actuelle pas fonctionnelle. La détection du contact entre un doigt et une surface, bien qu'elle soit implémentée, n'est donc pas testable actuellement.

7.5 Chargement et export d'images

Actuellement, notre application permet à l'utilisateur de fonctionner en 3 modes différents :

— LOAD_FROM_KINECT : Les images sont directement récupérées et affichées depuis la Kinect, et passent par toute la chaîne de traitement.

— SAVE_FROM_KINECT : Les images sont récupérées depuis la Kinect, et sont directement enregistrées dans un dossier correspondant au cas d'utilisation choisi. Cela nous permet notamment de créer des ensembles de test.

— LOAD_FROM_IMAGES : Les images sont chargées depuis le cas d'utilisation choisi, et sont envoyées dans la chaîne de traitement, puis affichées. Ce mode permet de charger les ensembles de test pré-établis, afin de vérifier efficacement l'implémentation des diverses fonctionnalités.

Chapitre 8

Conclusion

Nous n'avons pas atteints tous les objectifs fixés : la rotation n'a pas été mise en place, le machine learning nous ayant pris trop de temps notamment à cause de la nécessité de ré-annoter les images pour prendre en compte la rotation du doigt. Et nous n'avons trouvé de papier utilisant une méthode autre que du machine learning pour ça, avec les contraintes matériels (caméra de profondeur au niveau de la tête, pas de marqueurs sur la main) que nous avions.

Aussi nous avons un problème liée à la détection de la main : si celle-ci est trop proche d'une surface, la détection du contour se fait pour toute la surface plus la main, non pas les deux différemment. Ainsi la détection de contour au toucher ne marche pas contrairement au survol.

Nous ne sommes pas allés aussi loin que nous le souhaitions pour mener à bien ce projet, mais sommes contents d'avoir pu appliquer une méthodologie aussi propre que possible lors de ce projet, que ce soit via la gestion de projet ou l'architecture de celui-ci. De plus, Vva ce projet nous avons pu pour la première fois faire du traitement d'image sur autre chose que du RGB ou niveaux de gris mais des données de profondeur et donc acquérir du savoir et des compétences liées à cela.

Bibliographie

[1] Bardia Doosti. Hand pose estimation : A survey. *CoRR*, abs/1903.01013, 2019.

[2] Guillermo Garcia-Hernando, Shanxin Yuan, Seungryul Baek, and Tae-Kyun Kim. First-person hand action benchmark with rgb-d videos and 3d hand pose annotations.

[3] Chris Harrison, Hrvoje Benko, and Andrew Wilson. Omnitouch : Wearable multitouch interaction everywhere. pages 441–450, 10 2011.

[4] Hai Duong Nguyen and Soo-Hyung Kim. Hand segmentation and fingertip tracking from depth camera images using deep convolutional neural network and multi-task segnet. *CoRR*, abs/1901.03465, 2019.

[5] X. Zhang, Y. Wang, M. Gou, M. Sznaier, and O. Camps. Efficient temporal sequence comparison and classification using gram matrix embeddings on a riemannian manifold. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4498–4507, 2016.