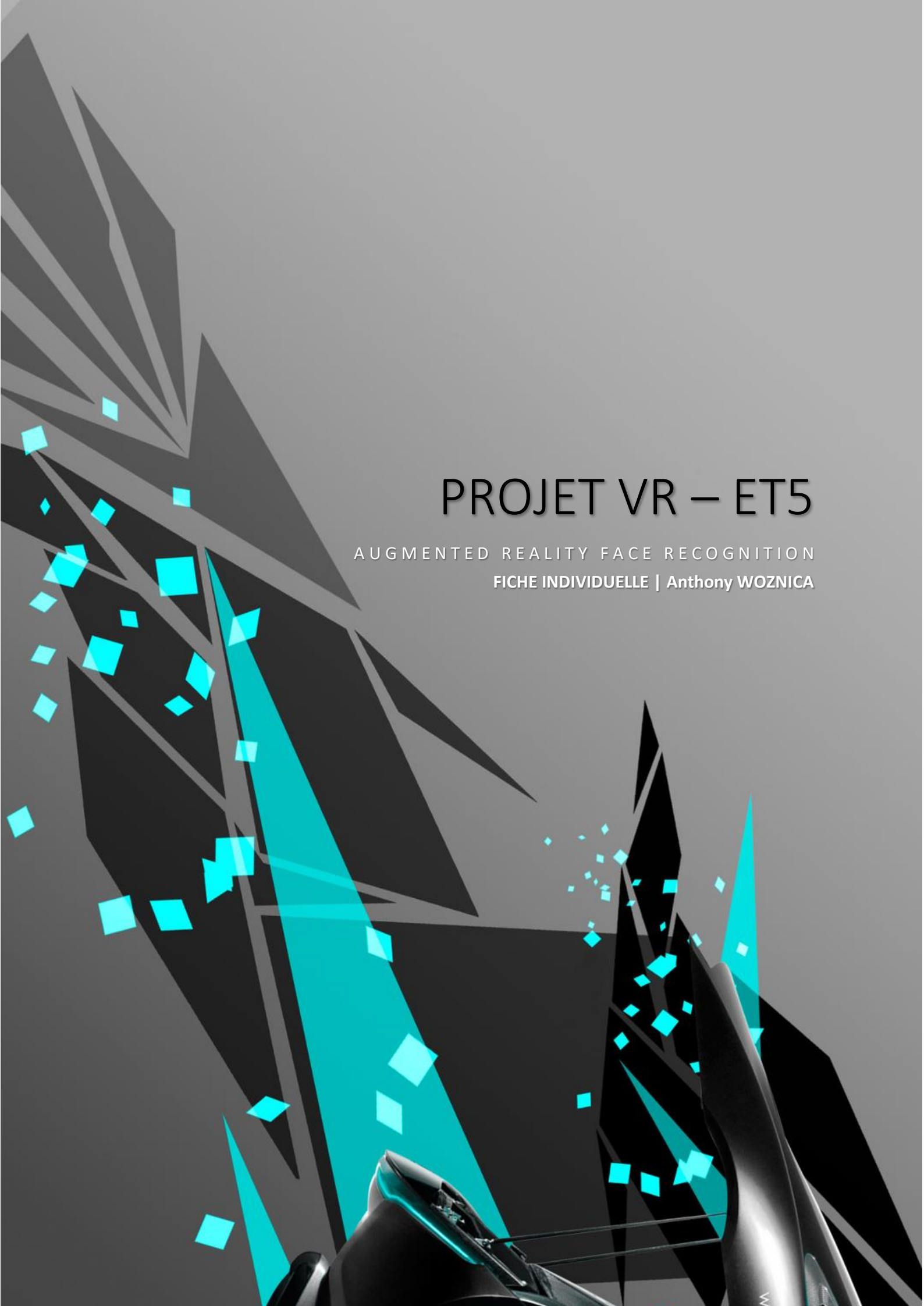

PROJET VR – ET5

AUGMENTED REALITY FACE RECOGNITION

FICHE INDIVIDUELLE | Anthony WOZNICA

INTRODUCTION

A l'heure actuelle, les applications faisant usage de réalité augmentée prennent une place croissante au sein du quotidien de chacun. En effet, quasiment chaque appareil actuel disposant d'une caméra a au moins une application en réalité augmentée, que ce soit les smartphones avec les filtres photos/vidéos intelligents de Snapchat, les jeux XBOX 360 exploitant la Kinect ou, plus récemment, les appareils HoloLens, se vouant exclusivement à des applications en réalité augmentée.

C'est avec la volonté de développer un programme pouvant s'insérer dans ce contexte que le projet présenté lors de la soutenance est une application de Réalité Augmentée, simplement nommée AUGMENTED REALITY FACE RECOGNITION. Le but de cette application est d'exploiter les caméras des appareils pour effectuer des tâches de reconnaissance, telles que la reconnaissance d'émotions ou de visage. Initialement prévue pour un déploiement uniquement sur HoloLens, le projet a connu d'importantes difficultés qui ont contraint l'équipe à repenser le mode de réalisation ainsi que le fonctionnement même de l'application, ce qui a mené à l'élaboration de trois applications indépendantes (Locale, Vuforia et HoloLens) usant d'un même Web Service.

Le groupe chargé de mettre en place ce projet se constitue de AZEMARD Thomas, BRINDAMOUR Benjamin, MBOURRA Max, SHI Yao et WOZNICA Anthony.

TRAVAIL REALISE

Peu après les débuts du projet, nous avons constaté qu'il n'était pas possible de réaliser ce que nous voulions uniquement en C# avec les librairies standard. Par conséquent, nous nous étions séparé les tâches, de manière à pouvoir avancer indépendamment les uns des autres.

Nous avions en premier temps conçu l'idée d'une HoloLens connectée par protocole TCP avec un serveur Python, de manière à faire passer le flux vidéo de l'HoloLens vers le serveur Python à la manière d'une Webcam. Cependant, les premiers tests ont mis en évidence un retard important dans la capture du flux vidéo, rendant difficilement exploitable une application en temps réel.

En revanche, j'ai pu élaborer en étroite collaboration avec les membres responsables des autres parties du projet un Web Service fiable et réutilisable. Plus en détail, la problématique de mon travail était avant tout de reconnaître un visage et de réussir à en déterminer la position, de rogner un visage d'une image, puis d'en reconnaître l'émotion associée et, si possible, afficher des informations complémentaires.

Ayant déjà eu l'occasion de travailler sur des thématiques de traitement d'image, j'ai décomposé mon travail en back comme suit (chacune des tâches sera expliquée par la suite) :

Tâche	Solution choisie
Elaboration d'un environnement de test	Usage de la librairie OpenCV pour capturer les frames de la Webcam du PC, ces dernières pouvant correspondre au flux d'images reçu par un autre appareil.
Détection de visages	Utilisation d'un classifieur en cascade en parallèle à des fichiers xml donnés.
Elaboration d'un dataset d'entraînement pour la prédiction d'émotions	Utilisation du dataset d'affective computing (gratuit) suivant : http://www.consortium.ri.cmu.edu/ckagree/
Nettoyage du dataset	Découpage de chaque image du dataset (détection de visage + rognage de l'image suivant les coordonnées retournées).
Formatage des données d'apprentissage	Transformation de l'image en une série de 48 landmarks à l'aide de la librairie dlib afin d'effectuer la reconnaissance d'émotions non pas sur des photos mais sur un ensemble de coordonnées.
Choix d'une méthode de reconnaissance	Utilisation de Scikit-Learn avec un choix porté sur une SVM linéaire.
Prédiction d'émotions en temps réel	Optimisation de code pour atteindre un seuil de fluidité dans la prédiction, qui doit s'effectuer en temps réel.
Elaboration d'un script de Web Service en collaboration avec Thomas	Librairie Flask décidée en fin de compte

N'ayant pas encore de lien avec l'HoloLens et connaissant les problèmes que nous rencontrions avec la connectique, j'avais besoin de m'appuyer sur un support local me permettant d'effectuer des jeux de tests en parallèle (support local qui deviendra plus tard notre « application locale » pour rejoindre le reste de l'équipe au plus vite.

Ainsi, je me suis tourné vers **OpenCV** et sa fonction **cv2.VideoCapture(0)** permettant d'effectuer sans trop de paramétrage la capture vidéo à partie d'une WebCam et la décomposition en frames.

J'ai en second temps déterminé les éléments qu'un support aurait besoin de connaître pour détecter un visage. Ces derniers étant sa position et sa taille, respectivement marquées par un ensemble (x, y, w, h), correspondant aux coordonnées en x et y sur l'écran, ainsi qu'à sa largeur et sa hauteur.

J'ai ensuite téléchargé plusieurs classifieurs en cascade pré-entraînés sur https://github.com/opencv/opencv/tree/master/data afin de permettre la détection de

visages sur des images. J'ai pu immédiatement réaliser cela sur mon environnement de test en observant la pertinence des classifieurs en retour vidéo. Cela a permis de poser les bases du « *face tracking* » de l'application. Dans la pratique, il ne s'agit pas réellement de face tracking, puisqu'il ne s'agit là que de donner l'illusion que le visage est suivi en retour visuel. Dans la pratique, un face tracking aurait pu être implémenté, mais pour cela il aurait été nécessaire d'implémenter un second niveau de reconnaissance à l'aide, par exemple, d'un apprentissage par renforcement hors de portée pour des étudiants non spécialisés dans l'IA. L'idée était initialement d'exploiter l'API Azure Visage (basée sur l'apprentissage par renforcement) pour suivre un visage **nommé**, mais cela aurait pour conséquence de limiter le framerate de l'application, et de perdre en fluidité (les requêtes sur l'API Azure Visage étant limitées), et donc de **perdre l'illusion de Face Tracking apportée par le repositionnement du carré autour du visage à chaque frame**. Ainsi, on privilégie un « faux » face tracking en temps réel plutôt qu'un vrai en différé et saccadé.

Enfin, je me suis penché sur la reconnaissance d'émotions. Cette problématique me paraissait très intéressante car elle permettait de juxtaposer au visage de l'information utile. Pour permettre cela, il était en premier temps nécessaire de trouver un dataset d'affective computing gratuit, ce qui fut chose faite en cherchant sur des forums de discussion (http://www.consortium.ri.cmu.edu/ckagree/). Une fois le dataset en main, j'ai entrepris de rogner chaque visage présent sur les photos, de les redimensionner, les catégoriser suivant l'émotion reflétée, puis de le mapper. L'intérêt de la procédure de mappage de visage a été de réduire la taille des éléments en ne tenant compte que des points d'intérêt du visage (passage d'une image de taille 350x350 à une matrice de taille 48x2. Dans un cas où on cherchera à faire du temps réel, la taille des données est un facteur clé dans l'optimisation). Ces points ont ensuite été normés suivant le barycentre du visage, afin de réduire l'influence des critères tels que la rotation du visage ou sa position dans le cadre.

L'ensemble des matrices a été séparé en des sets d'entraînement et de test pour une application d'apprentissage statistique. Le choix s'est porté sur une architecture de machine learning plutôt que sur des Réseaux de Neurones Convolutionnels, qui auraient potentiellement été plus adaptés au problème dans un contexte où la vitesse d'exécution ne compterait pas.

Enfin, plusieurs algorithmes ont été testés afin de déterminer lequel sera le plus à même de produire les meilleures prédictions dans un délai raisonnable : le choix s'est donc porté sur une SVM. Néanmoins, le dataset étant très petit, on m'a conseillé de ne conserver que reconnaître que les émotions pour lesquelles le dataset était suffisamment important, ce qui nous laisse avec 5 émotions reconnaissables : *joie, dégoût, colère, surprise, neutre*.

Néanmoins, j'ai pu remarquer que la qualité de la prédiction fluctuait de plus 26% (oscillation entre 79 et 95% de succès) entre plusieurs SVM consécutives. J'ai donc entrepris de remélanger les données de test et d'apprentissage, les séparer à nouveau et relancer l'entraînement, afin de faire tendre la qualité de la prédiction vers une moyenne stable (en l'occurrence, aux alentours des 85%).

De retour dans l'environnement de test, j'ai pu chercher à implémenter une reconnaissance d'émotions en temps réel. J'ai vite vu qu'il était impossible de conserver une fluidité dans le face tracking avec la prédiction d'émotions à chaque frame. Par ailleurs, la boucle principale du programme commençant à devenir imposante, j'ai cherché le compromis entre la prédiction en temps réel et la fluidité de l'animation. Ainsi, en expérimentant, j'ai déduit qu'avec la nouvelle configuration, il était possible, sur un ordinateur moyen d'avoir une relative fluidité à 30 Frames par secondes pour le face tracking. A cela s'ajoute la prédiction de l'émotion, étant malgré tout relativement chronophage (il faut prendre la frame de la caméra, rogner le visage, convertir en nuances de gris, puis prédire l'émotion), qui elle n'est exécutée en local que toutes les 5 frames.

Par la suite, en cherchant une API pouvant fournir des indications supplémentaires, j'ai trouvé une FaceAPI combinant Kairos et FaceApp (permettant de reconnaître d'autres éléments que les émotions). N'ayant pas de clé pour Kairos, il est nécessaire de passer par le biais de cette API pour accéder à la reconnaissance de ces autres éléments (de toute évidence, l'auteur de cette API a une clé pour les deux API susmentionnées).

Ainsi, il est nécessaire d'effectuer une requête sur la page Web de l'API pour en récupérer des informations, cela étant très long et surtout limité en nombre de demandes par minutes. Par conséquent, pour conserver une relative fluidité, les mises à jour des informations supplémentaires ne sont faites que toutes les 300 frames dans le programme en local.

Par la suite, j'ai pu rejoindre le reste de l'équipe qui avait la connectique avec un Web Service factice (renvoyant un *.json* du type que l'on souhaitait obtenir !), et pour laquelle j'ai élaboré une fonction de prédiction en fonction de l'image entrée en paramètres. Le temps d'exécution d'une requête sur serveur en local est de moins de 10ms, permettant ainsi une certaine fluidité dans les applications distantes.

J'ai par la suite pu assister aux tests respectivement sur Vuforia et sur l'HoloLens.

Un mot de la fin : Avec plus de temps devant nous, cependant, j'aurais souhaité implémenter la reconnaissance de plusieurs visages en temps réel comme ce qui était initialement prévu. Malheureusement, les contraintes liées à la connectique m'ont davantage poussé à étudier le problème avec mes camarades qu'à avancer sur les thématiques de reconnaissance de plusieurs visages, pourtant partiellement implémentées (le server_back_update.py renvoie une liste de fichiers json, plusieurs visages sont déjà détectés sur une photo, etc. L'implémentation est quasi-complète). Pour finir, je regrette beaucoup que l'essentiel de nos problèmes majeurs aient été liés à la connexion plutôt qu'à des réels problèmes comme la synchronisation et le temps réel qui se sont finalement révélées plus ou moins aisées à surmonter.