POLYTECH LILLE

Université de Lille 1 SCIENCES ET TECHNOLOGIES

Polytech Lille

Rapport de projet de quatrième année

Projet réalisé pour le compte d'Ürbik

Réalisation d'une caméra intelligente

Semestre 8 - Janvier à Mai 2018

Thomas Hubert
Taky Djeraba
Promo 2019

Resp. École : M. Xavier Redon
Resp. Entreprise : M. Nathan Martin

Remerciements

Dans un premier temps, nous tenons à remercier l'équipe pédagogique de Polytech Lille pour nous avoir apporté toutes les compétences et les connaissances nécessaire au bon déroulement du projet.

Nous tenons aussi à remercier M. Xavier Redon, M. Thomas Vantroys et M. Alexandre Boe pour leur encadrement.

Enfin, nous remercions M. Nathan Martin et M. Thomas Aldeguer pour toute l'aide et les conseils qu'ils nous ont apportés.

Sommaire

Chapitre 1

Contexte

Ürbik Ce projet à été réalisé dans le cadre d'une collaboration entre Polytech Lille et Ürbik, il a pour objectif de mettre en place une caméra Wi-Fi procédant à de la reconnaissance d'image. L'entreprise Ürbik se spécialise dans la conception de mobilier urbain intelligent et propose une gamme de solutions en vue de mettre en valeur une ville et ses aménagements.

Travail attendu L'idée qui à été apporté par Ürbik consiste à développer une caméra accessible en Wi-Fi depuis une de leurs bornes. Cette dernière doit ensuite permettre aux utilisateurs l'accès à un stream en direct diffusant le panorama aux alentours. Des informations concernant des bâtiments y seront diffusées en réalité augmentée.

Objectif A travers ce projet, Ürbik souhaite explorer les possibilités qui s'offrent à elle dans le domaine de la reconnaissance d'image. L'objectif du projet est donc de réaliser un "Proof of Concept" pour le rajout d'une caméra Wi-Fi intelligente à leur matériel déjà existant.

Chapitre 2

Cahier des charges

2.1 Besoins

Il s'agit ici de développer un prototype de caméra intelligente et interactive permettant de mettre en valeur une ville, ses ressources et ses aménagements. Plusieurs utilisateurs devront pouvoir se connecter au même moment à un point d'accès Wi-Fi afin d'accéder à l'application qui intégrera les fonctionnalités prévues.

2.2 Contraintes

2.2.1 Contraintes techniques

Le flux vidéo doit être accessible aux clients connectés, indépendamment de leur nombre. Le mouvement de la caméra, se limite à une simple rotation afin de balayer le plus d'espace possible. On envisage de rendre notre application accessible via un site internet. Ce dernier devra donc être simple, ergonomique et sécurisé. La gestion de l'énergie est à prendre en compte, mais elle n'est pas réalisée par nos soins. Nous nous contenterons d'optimiser notre système pour réduire sa consommation. La caméra devra aussi être résistante et étanche, puisqu'elle se destine à priori à un usage extérieur.

2.2.2 Contraintes Matérielles

Pour réaliser notre caméra, nous utiliserons le matériel suivant :

— Une Banana Pi. Celle-ci intégrera toutes les applications prévus. (modification du flux (Réalité augmentée), gestion du serveur web)
— La caméra en elle même consistera en une Banana-Pi BPI-D1, open source. (Source du flux vidéo)

2.2.3 Fonctions de base

Orientation de la caméra

— Réalisation d'un mouvement cyclique
— Pas de modification de la trajectoire en fonctionnement (Mais possibilité en mode administrateur)

Traitement de l'image
- Ajout de réalité augmentée au flux vidéo

Création d'un réseau Wi-Fi lié à la caméra
- Permet d'accéder au flux vidéo
- Accès à la caméra par Internet (Pour administration)

Envoi de l'image par streaming
- Serveur de streaming proposé par la caméra
- Accès au stream via le point d'accès wi-fi

Réalisation d'un client web
- Adapté aux navigateurs web (mobiles ou non)
- Intégration du flux vidéo
- Interface utilisateur permettant de modifier les informations ajoutées par la VR

Priorités Voici l'ordre des tâches que nous envisageons pour ce projet :
- La transmission d'un flux vidéo depuis la caméra
- L'intégration de ce flux dans un site web
- La conception mécanique de la caméra
- L'implémentation de fonctionnalités de réalité augmentée

Facteurs de qualité Voici maintenant les critères de qualité, suivant leur ordre d'importance :
- Prototype fonctionnel
- Application web fonctionnelle et efficace
- Intégration de la réalité augmentée
- Application web esthétique et ergonomique
- Sécurité
- Consommation énergétique réduite

Chapitre 3

Déroulement du projet

3.1 Configuration du matériel

Le matériel utilisé dans le projet à été dans l'ensemble fournis par Ürbik et consiste en une Banana Pi Pro, une Banana Pi D1 et une webcam USB.

3.1.1 Banana Pi D1

La Banana Pi D1 est un concept assez particulier puisqu'il est destiné à faire tourner un système d'exploitation allégé couplé à une caméra. Cela permet de préparer l'image à un traitement postérieur, par exemple en calculant les points d'intérêts de l'image, et donc de soulager le micro-processeur dont les ressources seront surtout utilisé par le traitement de l'image.

Malheureusement, nous n'avons pas réussi à le faire fonctionner, pour l'usage que nous voulions en faire. Il s'est en fait avéré que le Banana Pi D1 n'est pas utilisable par une connexion Ethernet via USB. Il supporte seulement une connexion via Wifi.[1] Cela le rendait inutilisable pour notre projet, puisque la caméra devait être à l'extérieur d'une boîte métallique renfermant le banana-pro avec lequel elle devait communiquer.

Nous utiliserons donc une simple webcam, ce qui nous limite à notre simple BananaPro pour le traitement d'image.

3.1.2 Banana Pi Pro

Installation de l'OS Bananian La Banana Pi Pro devant héberger notre application, celle-ci doit être optimisée afin de consommer le moins de ressources possible, l'essentiel des capacités allant à la reconnaissance d'image et au serveur. Nous nous sommes donc tourné vers l'OS minimaliste Bananian. Celui-ci possède l'avantage d'être totalement compatible avec la Banana Pi Pro en possédant tout les drivers nécessaires.

NB : l'OS Bananian n'est plus mis à jour depuis Avril 2018

Installation de d'OpenCV Pour la reconnaissance d'image, nous nous sommes tourné vers la bibliothèque OpenCV. Cette dernière est très bien documenté et possède une gamme de méthodes prédéfinies permettant de faire du traitement d'image.

Configuration du point d'accès Wi-Fi Concernant le serveur, ce dernier est rendu accessible via le point d'accès Wi-Fi de la Banana Pi Pro. Il nous a donc fallut configurer ce point d'accès et mettre en place un serveur dhcp afin de d'affecter une adresse IP à chaque client se connectant.

Activation du point d'accès :

Fait en modifiant le fichier **/etc/network/interfaces** :

```
// STATION MODE
//auto wlan0
//iface wlan0 inet dhcp
//wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
// ACESS POINT MODE
allow-hotplug wlan0
iface wlan0 inet static
address 192.168.100.1
netmask 255.255.255.0
```

Installation d'hostapd :

On utilise l'utilitaire hostapd, et on modifie le fichier **/etc/hostapd/hostapd.conf** :

```
interface=wlan0
driver=nl80211
ssid=BananaPro
channel=6
hw_mode=g
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
```

Installation d'udhcpd :

Il nous a aussi fallu configurer un serveur dhcp, afin d'attribuer une adresse IP au appareils connectés. Pour cela, nous avons installé l'outil udhcpd. Ce dernier se configure via le fichier **/etc/udhcpd.conf** :

```
start 192.168.100.101 #default: 192.168.0.20
end 192.168.100.254 #default: 192.168.0.254
```

3.2 Reconnaissance d'image

Nous allons maintenant expliquer d'abord la stratégie adoptée, et ensuite nous verrons son implémentation.

3.2.1 Stratégie adoptée

Choix de la bibliothèque Parmi les différentes bibliothèques disponibles pour faire de la réalité augmentée, la bibliothèque OpenCV a été sélectionnée car d'une part elle est open-source, donc gratuite d'utilisation. Elle est ensuite communément reconnue comme étant assez performante, son support est assuré par Intel [2] et possède une importante communauté. C'est pour l'ensemble de ces raisons qu'il nous a été recommandé d'utiliser OpenCV pour ce projet.

Choix du langage OpenCV existe pour plusieurs langages, comme le C, le C++, le python et le Java. Nous avions d'abord choisi le C++. Mais très rapidement, suite à la première réunion avec Urbik, nous avons choisi d'utiliser le python pour plusieurs raisons :

— La présence d'un développeur Python chez Urbik, qui était capable de nous aider plus facilement sur le Python que le C++
— Une communauté très présente sur les forums, que ce soient ceux du projet OpenCV ou du langage Python
— La plus grande facilité de l'installation des bibliothèques, en l'occurrence la bibliothèque X2dFeatures dont nous avions besoin qui était plus simple à obtenir en Python
— La programmation plus haut niveau, qui nous a permis de moins nous attarder sur des détails de langages et plus sur l'algorithmique du projet

En contrepartie, cela nous imposait :
— Des performances légèrement plus réduites qu'avec un langage plus bas niveau
— Un temps d'apprentissage un peu plus long (Cela dit sans aucun doute compensé par la simplicité d'implémentation)

Choix de la méthode de reconnaissance Nous avions plusieurs choix concernant la méthode de reconnaissance d'image.
— Une reconnaissance de couleurs, mais qui n'est pas très fiable dans notre cas, à cause de potentiels changements de luminosité puisque la caméra est destinée à être en extérieur
— Une reconnaissance des formes, mais qui ne sera pas fiable non plus, à cause du risque de ressemblance entre les éléments du panorama
— Une reconnaissance de points-clefs dans l'image, qui semblait être la plus fiable des trois, et permise parce que notre caméra ne se déplacera pas. C'est la méthode qui a été retenue

Choix de l'algorithme Pour implémenter notre reconnaissance de points, il existe plusieurs algorithmes[3] :
— La methode Surf, pour Speeded-Up Robust Features [4], qui est une version optimisée de l'algorithme Sift [5]. Il est assez fiable, mais légèrement moins rapide comparé aux autre méthodes.
— La methode Orb, pour Oriented Fast and Rotated Brief [6]. Cette méthode est plus rapide, mais peut être moins précise que Surf.

Nous avons dans un premier temps implémenté les deux méthodes, et permis le passage de l'une à l'autre par la modification d'une variable.

Autres choix d'implémentation Nous avons également fait le choix d'utiliser deux threads, pour la reconnaissance d'image. Le premier doit obtenir une image de la caméra, et y appliquer le dernier masque de réalité augmentée qui a été calculé. Le second thread doit calculer l'image à incruster, en prenant en compte l'image reçue par la caméra et en la comparant aux images de référence.

Ici, séparer les deux permet d'augmenter le nombres d'images par secondes reçues par le client, puisqu'on peut en parallèle diminuer le nombre de calculs concernant le masque à appliquer. On a encore augmenter les performances en limitant ces calculs aux cas où un mouvement est détecté. (Voir 3.2.4 page 12 pour plus de détails)

3.2.2 Sous-programme de calcul du masque

Cette partie du programme est le centre de notre projet. Elle se charge de calculer la déformation et la position du masque de l'objet à intégrer à l'image reçue par la caméra.

Calcul des Points-Clefs Notre méthode utilise le calcul de points clefs. Ces points sont les endroits distinctifs de l'image qu'il est possible de retrouver entre une image de référence et l'image obtenue par la caméra. [7]

Sous OpenCV, on l'obtient facilement grâce aux lignes :

```
#on lit l'image
objet = cv2.imread('image.jpg')

#On  cree l'objet surf
surf = cv2.xfeatures2d.SURF_create()

#On obtient les points clefs et leurs descripteurs
PointsClefsObjet, DescripteursObjet = surf.detectAndCompute(objet, None)
```

Listing 3.1 – Calcul des points-clefs de l'image

Dans cet exemple, c'est la méthode Surf qui a été utilisée.

FIGURE 3.1 – En bleu, les points-clefs

L'avantage de ces point-clef, c'est qu'il est très probable qu'ils soient identiques entre deux images représentant les mêmes objets. Cela permet donc de retrouver la transformation opérée entre les deux objets. Une condition pour que la méthode fonctionne parfaitement est de limiter les changements de position de la caméra entre les prises de vue. Cela dit, la méthode est robuste à toutes les modifications induites par la rotation de la caméra et aux déformations dues à l'objectif utilisé.

Identification des Points-clef Une fois les points clefs obtenus sur l'image de référence et sur l'image obtenue par la camera, il faut les identifier. On va comparer les points de chaque image et regarder leur ressemblance, en comparant leurs paramètres : position, orientation, gradient...

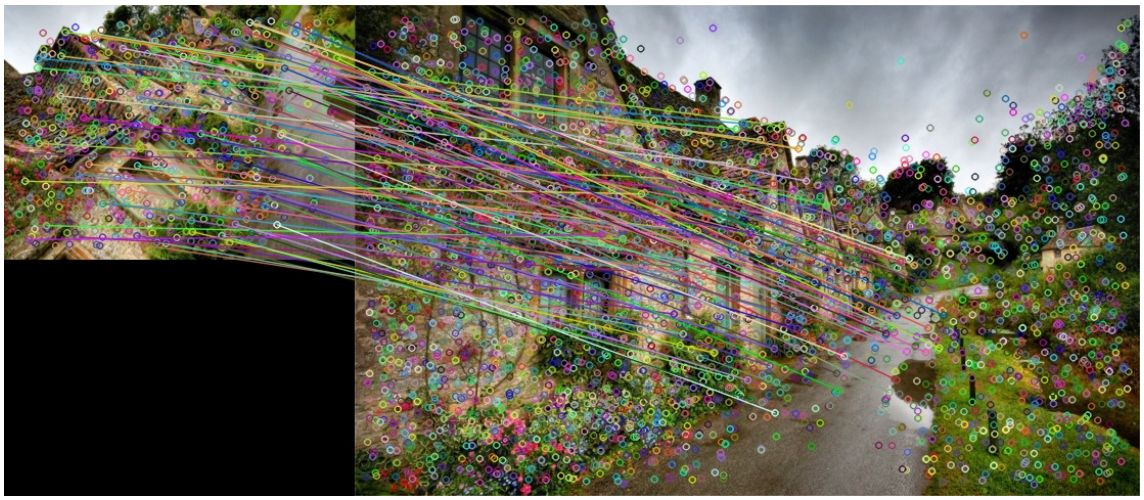

FIGURE 3.2 – Identification des points-clefs

On va obtenir des couples de points, et on va pouvoir éliminer les points les moins ressemblants. Dans les faits, OpenCV utilisera pour matcher les points-clefs leurs descripteurs, qui permettent de comparer les points-clef plus efficacement. [8] Et finalement, grâce aux points sélectionnés, on va pouvoir évaluer la transformation à effectuer pour obtenir l'image de base.

Pour cela, nous utilisons les lignes :

```
#On cree un objet BFMatcher
bf = cv2.BFMatcher()

#On obtient les matches a partir des descripteurs
matches = bf.knnMatch(desmask,despano, k=2)

#On elimine les matches incoherents
good = []
for m,n in matches:
    if m.distance < 0.45*n.distance:
        good.append([m])
```

Listing 3.2 – Match des points-clefs et élimination des résultats aberrants

Calcul de la matrice de déformation et application sur le masque Maintenant qu'on a la transformation des points-clefs, on va pouvoir calculer la déformation et la translation à appliquer à l'image de base pour retrouver son placement obtenu à la caméra.

En appliquant cette même déformation au masque, qui est créé en fonction de l'image de référence, on va pouvoir replacer l'objet contenu dans le masque à l'endroit voulu dans l'image de la caméra.

Les lignes suivantes permettent de trouver et d'appliquer une transformation à un masque à appliquer :

```
#On commence par reorganiser le stockage des points
src_pts = np.float32([ pointsClefDuFond[m.trainIdx].pt for m in good ]).reshape
    (-1,1,2)
```

FIGURE 3.3 – Application du calque (en bas à gauche) à l'image par application de la matrice de transformation

```
dst_pts = np.float32([ pointsClefObjet[m.queryIdx].pt for m in good ]).reshape
    (-1,1,2)

#On peut ensuite trouver une matrice de deformation
M, _ = cv2.findHomography(dst_pts, src_pts, cv2.RANSAC,5.0)

#On l'applique ensuite a l'objet a incruster
objetTransforme =cv2.warpPerspective(objet,M,(640,480))
```

Listing 3.3 – Calcul de la matrice de déformation

Gestion d'objets multiples Un autre aspect de notre programme est la gestion de multiples objets du panorama. Pour ce faire, nous avons créé une classe, qui est chargée de stocker ces objets.

C'est donc le thread chargé de calculer la déformation du masque qui va gérer le défilement entre les différents objets. Il va donc sélectionner la bonne image de référence, le masque associé et les coordonnées à appliquer aux servomoteurs et stockées dans un objet de la classe ObjetCible.

Pour optimiser l'exécution du programme, nous stockons également les points-clefs dans l'instance de la classe ObjetCible. Ces points-clefs ne doivent pas varier, puisqu'ils sont basés sur une image fixe. Cela nous évite d'avoir à les recalculer à chaque nouvelle estimation de la déformation, sachant que le calcul des points-clef est l'une des opérations les plus longues à l'exécution, d'après nos mesures.

3.2.3 Sous-programme de génération de l'image

Cette partie du programme est chargée de générer les frames qui seront affichées sur le serveur. Le fait d'avoir séparé cette partie du programme du calcul de matrice de déformation nous permet d'augmenter le nombres de frames par secondes émises, en diminuant le nombre de calcul de déformation par secondes. Nous détaillerons par la suite nos choix d'optimisation. (Voir 3.2.4 page 12)

Récupérer les frames depuis la caméra OpenCV permet très simplement de récupérer les frames depuis une caméra. Nous avons utilisé les lignes suivantes :

```
#cree l'objet camera, ID_CAMERA = N
#avec /dev/videoN le chemin de la camera a choisir
camera = cv2.VideoCapture(ID_CAMERA)

#Lit une frame de la camera
_, img = camera.read()

#Destructeur de l'objet camera
camera.release()
```

Listing 3.4 – Récupération d'une frame depuis la camera

Incruster l'image récupérée à la frame La deuxième fonction de ce thread est d'incruster le masque à la frame à fournir au serveur. Comme OpenCV ne gère pas le canal alpha directement (le canal alpha est un canal qui contient l'inforamtion de transparence d'une image), cela a compliqué très légèrement l'incrustation de notre masque sur l'image de base. Il a donc fallu récupérer d'un côté l'image en elle-même, de l'autre l'information de transparence, et fusionner le tout. Voici comment nous avons procédé.

```
  #On recupere d'abord l'image
    objet =cv2.imread("objet.png")

    #On recupere ensuite le canal de transparance
    #La fonction cv2.split permet de recuperer un canal,
    #et l'argument -1 de cv2.imread force le programme a ouvrir le calque
    masqueObjet = cv2.split(cv2.imread("objet.png",-1))[3]

    #On calcule l'inverse du masque
    mask_inv=cv2.bitwise_not(masqueObjet)

    #On met a zero les pixels de fond de l'objet a incruster
  objet = cv2.bitwise_and(objet, objet, None, mask_inv)

    #On fait de meme pour les pixels sur lesquels l'objet va s'afficher
  frame = cv2.bitwise_and(frame, frame, None, masque)

    #On additionne les deux pour obtenir le resultat
    resultat=cv2.add(pano,contour)
```

Listing 3.5 – Incrustation du masque sur la frame

3.2.4 Optimisation

Nous allons maintenant présenter les différentes conclusions auxquelles nous sommes parvenus au cours de notre projet, concernant son optimisation.

Premiers résultats Lors de la première exécution du programme sur BananaPro, nous avons obtenu une cadence de génération d'image proche d'une image toutes les deux secondes.

Ce résultat étant largement insuffisant, nous avons cherché les causes de pertes de temps lors de l'exécution du programme. Un premier profiling nous a révélé que les deux opérations les plus longues étaient :

— La génération des points-clefs
— L'acquisition des images par la caméra
Nous avons donc orienté notre optimisation sur ces deux pistes.

Choix de la méthode de génération des points-clefs OpenCV propose plusieurs algorithmes de génération des points-clef. Le choix de la méthode est crucial, puisque c'est la première cause de perte de temps lors de l'exécution du programme. Si on a d'abord utilisé l'algorithme Surf, plus lent mais plus robuste, il s'est avéré que l'algorithme Orb suffisait largement pour notre application, et permettait un calcul à peu près trois fois plus rapide. (Pour plus de détails sur les méthodes, voir 3.2.1 page 8).

Ce premier choix a déjà permis un gain de temps assez conséquent.

Limitation des calculs du masque Une autre idée est de limiter le nombre de calculs de la déformation du masque par secondes. Pour ce faire, nous avons donc commencé par séparer le calcul du masque de la génération de la frame en leur attribuant chacun un thread.

Nous avons ensuite estimé les cas où un calcul du masque était nécessaire, c'est à dire en cas de trop grand changement entre l'image qui est acquise et l'image pour lequel il a été calculé.

Nous estimons la différence entre ces frames d'une manière assez simple. On fait la différence entre les matrices formant les frames, on obtient alors une matrice contenant la différence pixel par pixel. On somme alors l'ensemble des points de la matrice, pour obtenir une valeur qui quantifie le mouvement dans l'image. On établit enfin un seuil au-dessus duquel on s'autorise de calculer la déformation.

Cela peut se faire sous OpenCV avec les lignes :

```
#On obtient la matrice differentielle entre les deux frames
matDiff=cv2.absdiff(frame,oldFrame)

#On fait la somme de la matrice
diff=np.sum(matDiff)

#Si la somme est superieure a une limite, on actualise la frame et on calcule la
    deformation
if diff > limite :
    oldFrame = frame
```

Listing 3.6 – Calcul de la quantité de mouvement

Un autre moyen simple pour améliorer les performances a été de limiter artificiellement le nombre d'actualisation du masque par seconde, et en libérant le processeur. Nous l'avons fait en rajoutant la ligne :

```
time.sleep(0.1)
```

Impact de la caméra Lors de notre premier profiling, il s'est avéré que la caméra avait un temps d'acquisition non négligeable. Il a de plus été observé que les conditions d'éclairage influaient sur le temps d'acquisition, sur PC le programme de reconnaissance d'image passait de 30FPS à 15FPS suivant que éclairage soit suffisant ou non. Cela vient du fait que la caméra prenne plus de temps pour capturer la lumière dans les basses luminosités.

Mais un tel écart n'a pas été observé sur BananaPro, il se peut que ce soit par le fait qu'OpenCV utilise un thread séparé pour gérer la caméra, et suspende l'exécution du programme

si il n'a plus de frames dans une queue qui les stocke. Comme le PC est puissant, il épuise rapidement la pile, et doit attendre la génération des nouvelles frames. Mais comme la bananaPro est plus lente, elle n'épuise pas la pile et n'est pas impactée par ce problème.

Nous n'avons pas pu vérifier cette théorie dans la documentation d'OpenCV, mais si ce n'est pas le cas, peut-être que capturer la caméra dans un Thread spécifique pourrait permettre d'améliorer les performances, en permettant l'utilisation du processeur par un autre Thread lors de l'acquisiton.

Résultat finaux Les résultats obtenus finalement sont une douzaine de frames par secondes, avec à peu près six actualisations du masque par secondes, dans un cas moyen, obtenu sur le BananaPro. L'image de fond est donc assez fluide, en comparaison avec les premiers résultats que nous avions. En contrepartie, l'actualisation du masque est un peu plus lente après un mouvement de la caméra.

3.3 Mise en place du serveur

La mise en place du serveur est une composante essentiel de notre projet étant donnée que la caméra est destiné à être utilisé par un nombre d'utilisateurs que l'on voudrait être le plus grand possible (Cette application s'applique supposément à des lieux touristiques). Le challenge consiste donc à mettre en place un serveur qui fasse tourner notre l'application de reconnaissance d'image de manière stable et qui soit accessible à plusieurs clients.

Un point important à prendre en compte est le format du stream vidéo. On a vu précédemment que notre application permet générer des frames. Afin de diffuser le stream vidéo produit par notre application, nous avons donc opté pour du "motion-jpeg". Le principe consiste à récupérer chaque frame séparément et à l'encoder au format jpeg.

3.3.1 Implémentation du serveur via Flask

La première solution qui à été retenu a consisté à utiliser le micro-framework Flask. Ce micro-framework possède l'avantage d'être open-source et d'être simple à utiliser. Flask se base sur l'utilisation de générateur pour générer un flux vidéo continu en motion "jpeg"

Nous nous sommes basé sur le un tutoriel de Miguel Grinberg[9] pour le code qui suit :

```
app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

def gen(camera):
    while True:
        frame = camera.get_frame()
        yield (b'--frame\r\n'
        b'Content-Type: image/jpeg\r\n\r\n' + str(frame) + b'\r\n')

@app.route('/video_feed')
def video_feed():
    return Response(gen(Camera()),
                    mimetype='multipart/x-mixed-replace; boundary=frame')
```

Listing 3.7 – Serveur de stream Flask

Le code ci-dessus fonctionne de manière très simple : En réponse à un appel à "/", la racine de notre serveur, l'application flask renvoie la page html importé (ici : index.html) :

```
<html>
  <head>
    <title>Video Streaming Demonstration</title>
  </head>
  <body>
    <h1>Video Streaming Demonstration</h1>
    <img src="{{ url_for('video_feed') }}">
  </body>
</html>
```

Listing 3.8 – template html importé

Cette page comporte une image dont la source est obtenu par "video_feed". En réponse à un appel à "/video_feed", notre serveur lance la méthode gen(Camera), il s'agit du générateur question chargé de retourner une frame sous la forme d'une "multipart response". La frame est ensuite généré par un appel à la méthode get_frame implémenté dans notre application de reconnaissance d'image.

Par cette méthode, nous avons été capable de streamer avec une faible latence mais seulement sur un seul client à la fois, étant donné que le serveur généré par Flask n'est pas destiné à une utilisation très poussée.

Nous avons donc cherché à porter cette application Flask sur un serveur web plus efficace. Notre choix s'est donc porté vers Nginx. Pour ce faire, nous avons utilisé un serveur uWSGI afin de permettre aux clients de faire du reverse proxy pour accéder à notre application.

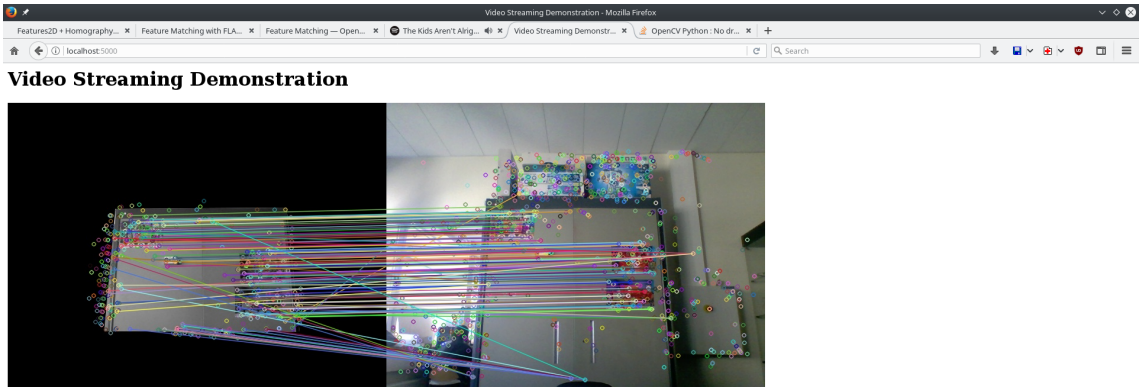

FIGURE 3.4 – Test du serveur uWSGI avec affichage des points clefs

Cette solution ne s'est finalement pas avéré efficace en raison de son incapacité à gérer plusieurs clients. Il s'est finalement avéré que la raison de cet échec était lié au fait que plusieurs clients ne pouvait accéder aux frame généré par l'application au même moment.

Statégie Nous avons finalement du repenser la manière avec laquelle fonctionne notre serveur en prenant en compte les contraintes suivantes :

— Étant donné les ressources limités de la Banana Pi Pro, le traitement doit se faire une seule fois sur le micro-contrôleur.
— Chaque client doit se voir donner l'accès aux frames générés par l'application sans perturber le fonctionnement du traitement d'image ou du serveur.

3.3.2 Serveur HTTP

La solution qui a ensuite été testée et qui a finalement été retenue consiste en la création d'un serveur TCP. L'idée est donc la suivante : lorsqu'un client se connecte au serveur, ce dernier se voit donner l'accès à une socket sur laquelle on écrit le contenu de la frame préalablement encodé au format jpeg.

Coté Application Afin que plusieurs utilisateurs puissent avoir avoir accès à la frame généré par l'application, nous avons mis en place une application multi-threadé. Le traitement d'image est ainsi réalisé dans deux thread en tache de fond. Durant son exécution, les thread écrivent la frame générée dans une file de message. Pour accéder à cette frame, la méthode get_frame lit sur cette même file de message. La frame va ainsi pouvoir être lue par différents threads.

Coté Serveur Pour mettre en place notre serveur, nous avons utilisé la bibliothèque *http.server*, sous-classe de *socketserver.TCPServer*. Cette bibliothèque permet de créer un serveur web basique de manière très simple.

Notre programme est composé des classes suivantes : ThreadedHTTPServer et CamHandler, qui est de type BaseHTTPRequestHandle.

Le fonctionnement du serveur se fait donc de la manière suivante :

1. Un serveur TCP est d'abord créé par la construction d'un objet de classe "ThreadedHTTPServer". Cette classe va nous permettre d'instancier notre serveur[10]. Le premier paramètre à passer est l'adresse et le port sur lequel créer le serveur (l'adresse doit être celle de machine sur le réseau). Le second paramètre est le handler. Il s'agit de la méthode devant être lancé à chaque fois qu'une connexion entre la socket et un client est accepté.
2. Cette méthode, de type BaseHTTPRequestHandler, lancé dans un thread pour chaque client, se charge ensuite d'écouter les requêtes des clients et d'agir en conséquence.

```
1 class ThreadedHTTPServer(ThreadingMixIn, HTTPServer):
2     """Gere les requetes des clients dans une thread"""
3     def __init__(self, server_address, RequestHandlerClass, bind_and_activate=True
      ):
4         HTTPServer.__init__(self, server_address, RequestHandlerClass,
      bind_and_activate)
5         ThreadingMixIn.__init__(self)
6
7     def serve_forever(self, poll_interval=0.5):
8         super().serve_forever(poll_interval)
```

Listing 3.9 – ThreadedHTTPServer

Le démarrage du serveur se fait via la méthode serve_forever(), qui appartient au module *SocketServer()*, surclasse de *TCPServer*

```
def main():

    #Demarage du thread de traitement d'image
    process = Thread( target=camera_opencv.process , args=("Thread-process",queue)
    )
    process.daemon = True
    process.start()

    #Demarage Serveur HTTP
    server = ThreadedHTTPServer((ADRESSE, PORT), CamHandler)
    print("server started")
    server.serve_forever()

if __name__ == '__main__':
    main()
```

Listing 3.10 – Programme principal

En paramètre du thread de traitement d'image, on retrouve la queue sur laquelle la frame est écrite.

Traitement des requêtes Notre classe CamHandler permet seulement d'écouter les requêtes reçues sur le serveur. Afin de les traiter, nous utilisons un ensemble de méthodes pré-existante propre à chaque type de requêtes :

- do_GET : pour les requêtes de type de type GET.
- do_POST : pour les requêtes de type de type POST.
- do_AUTHHEAD : pour les requêtes d'authentification.

Par exemple, les connexion à l'adresse IP :PORT/public.html sont traitées ainsi :

```
if self.path.endswith('public.html'):
            self.send_response(http.client.OK)
            self.send_header('Content-type', 'text/html')
            self.end_headers()
            self.wfile.write(self.html_public_page.encode())
```

Listing 3.11 – Python example

On se sert des méthodes send_response et send_header afin de générer une page web en html. La page en elle même est importée (ici : *self.html_public_page*) et écrite sur la socket.

Pour importer un fichier javascript ou css, on procède de la même manière en écrivant sur la socket le contenu du fichier :

```
        elif self.path.endswith('.css'):
            self.send_response(http.client.OK)
            self.send_header('Content-type', 'text/css')
            self.end_headers()
            file = open(self.path[1:],"r")
            css = file.read()
            self.wfile.write(css.encode())
```

Listing 3.12 – Fichier CSS

```
        elif self.path.endswith('.js'):
            self.send_response(http.HTTPStatus.OK)
            self.send_header('Content-type', 'text/js')
            self.end_headers()
            file = open(self.path[1:],"r")
```

```
            js = file.read()
            self.wfile.write(js.encode())
```

Listing 3.13 – Fichier Javascript

Diffusion du Stream La diffusion du stream se fait via des requêtes GET. Nous nous sommes pour cela inspiré de notre programme flask. Lorsqu'un client se connecte au serveur à l'adresse IP :PORT/public.html, une requête GET est envoyée au serveur qui écrit sur la socket le code html de la page en question. Ici, il s'agit de la page de diffusion du stream video.

Cette page contient un lien vers une source, ici IP :PORT/cam.mjpg. Ce lien va à son tour provoquer l'envoi d'une requête GET vers notre serveur, en réponse de quoi le serveur va écrire la frame récupérée sur la socket :

```
def do_GET(self):

        if self.path.endswith('.mjpg'):
            self.send_response(http.client.OK)
            self.send_header('Content-type', 'multipart/x-mixed-replace; boundary
    =--jpgboundary')
            self.end_headers()
            img=[]
            while True:
                try:
                    #On recupere la frame dans la file de message via un appel a
    get-frame()
                    img = camera_opencv.get_frame(queue)

                    # Si la frame recupere est invalide (car erreur de l'
    application), on prend la derniere frame lue
                    OldImg=img
                    if img is None:
                        img=OldImg

                    # Frame encode au format jpg
                    jpg = cv2.imencode('.jpg', img)[1]
                    jpg_bytes = jpg.tobytes()

                    # Generation de la page html
                    self.wfile.write("--jpgboundary\r\n".encode())
                    self.send_header('Content-type', 'image/jpeg')
                    self.send_header('Content-length', len(jpg_bytes))
                    self.end_headers()

                    # Ecriture de la frame sur la socket
                    self.wfile.write(jpg_bytes)
                except (IOError, ConnectionError):
                    break
```

Listing 3.14 – Ecriture de la frame sur la socket

Page admin Afin de pouvoir prendre les photos que nous utiliserons pour la reconnaissance d'image, nous avons prévu un mode administrateur permettant de contrôler la caméra et de prendre une photo dont le titre et la description peuvent être renseignés. Les données sont envoyées par le client sous la forme de requêtes de type POST qui sont traitées par le Handler *CamHandler*. Le mode administrateur à été implémenté dans un autre fichier : Admin.py. Ceci permet :

— D'éviter les conflits liés au contrôle des servo-moteurs.

— De renforcer la sécurité, puisque le serveur "public" et le serveur "admin" ne peuvent être lancé en même temps.

```
def do_POST(self):

        content_length = int(self.headers['Content-Length'])
        body = self.rfile.read(content_length)
        requete = body.decode()
```

Listing 3.15 – Lecture des requêtes de type POST envoyé le client

Authentification Nous avons aussi mis en place un système d'authentification afin de protéger l'accès à notre page administrateur. Pour ce faire, à chaque connexion à la page admin, on vérifie si le client est autorisé à accéder à l'application en vérifiant dans les headers de la requête envoyé :

```
if self.headers['Authorization'] == 'Basic '+ cle_encode:
        self.send_response(http.client.OK)
        self.send_header('Content-type', 'text/html')
        self.end_headers()
        self.wfile.write(self.html_admin_page.encode())

else:
        self.do_AUTHHEAD()
        self.wfile.write('veuillez vous identifier'.encode())
```

Listing 3.16 – Accès à la page d'authentification

```
def do_AUTHHEAD(self):
        self.send_response(401)
        self.send_header('WWW-Authenticate', 'Basic realm=\"Test\"')
        self.send_header('Content-type', 'text/html')
        self.end_headers()
```

Listing 3.17 – Envoi d'une requête d'authentification

Si le client inscrit les bons identifiants, ce dernier accède à la page admin.

En résumé Nous avons trois fichiers python :

— **camera_opencv.py** : contient la thread de traitement d'image et la méthode get_frame() permettant de retourner la frame traité quand appelé
— **Serveur.py** : contient le serveur. Il s'agit du programme principal qui se charge de lancer la thread de traitement d'image.
— **Admin.py** : contient le serveur de la page admin

Chapitre 4

Tests et mode d'emploi

4.1 Mode d'emploi

Ce mode d'emploi a été écrit le 15 mai 2018, il est susceptible d'évoluer avec d'éventuelles avancées sur ce projet. (Voir le lien vers le répertoire du projet[11]).

Lancement de la bananaPro Pour lancer la bananaPro, vérifier que la dernière version de l'image soit bien présente sur la carte mémoire. (Voir le lien sur le wiki du projet [12]) Une fois la carte lancée, il faut se logger avec l'identifiant **root** et le mot de passe **pi**

On doit alors sélectionner l'environnement virtuel par les commandes, puis on change de dossier :

```
source ~/.profile
workon cv
cd dev
```

Listing 4.1 – Selection de l'environnement virtuel et du dossier

On peut alors lancer le serveur public ou la page de configuration de l'administrateur

```
#Page d'administration
python Admin.py
#Serveur public
python Serveur.py
```

Listing 4.2 – Lancement du script

Connection au site Pour se connecter à la page publique, il faut utiliser l'url : **IP/public.html** Pour se connecter à la page d'administration, il faut utiliser l'url : **IP/admin.html**

Avant toute utilisation, il faut paramétrer la caméra, c'est à dire préparer les points d'intérêts qu'elle va cibler successivement.

Création des points d'intérêts La page admin permet de créer facilement la liste des points d'intérêts. A l'aide des flèches sur le site, il faut diriger la caméra vers l'objet à pointer. Une fois qu'il est correctement pointé, il faut rentrer un nom, une description, et cliquer sur envoyer. Il faut répéter l'étape pour tous les points d'intérêts saisis.

On va alors pouvoir créer les masques à appliquer par-dessus les images. Pour ce faire, il faut récupérer dans le dossier _dev les fichiers qui portent le nom : <nom_du_point_d_/interet>_ref.jpg, et les éditer un par un, de préférence sur un ordinateur personnel. Il faut alors créer un nouveau calque sur fond transparent, et placer l'élément de réalité augmenté sur ce calque transparent, placé à l 'endroit où l'on voudrait qu'il apparaisse par rapport à l'image de référence. Il faut alors exporter l'image obtenue sous le nom : <nom_du_point_d_/interet>_fond.png. L'éditeur d'image Gimp permet de réaliser facilement ces étapes.

Quelques remarques sur la création de calque :
- Il est important que les images finissant en fond.png contiennent le masque uniquement, et sur fond de transparence.
- Il est également important de garder un masque de même dimension que l'image de référence.
- Il est recommandé de ne pas modifier l'emplacement de la caméra après avoir fait son initialisation, cela risque d'altérer les capacités de reconnaissance d'image si le déplacement est trop important.

Quant au masque, on est tout à fait libre de ce qu'il contient, il est capable d'incruster des images colorées quelles qu'elles soient : un bâtiment de style futuriste pour remplacer l'existant, un carré vert qui entoure une statue... Les seules limites sont l'imagination, le talent artistique et la taille de l'image.

Il ne reste alors qu'à relancer le serveur public pour voir les modifications prises en compte.

4.2 Tests du serveur

Les tests réalisé sur le serveurs se sont avérés très concluants, et ce sur le BananaPro. Nous avons on effet été capable de mettre en place un serveur efficace pouvant servir plusieurs clients avec une faible latence.

Les captures d'écran en annexe donnent un aperçu des site publique et admin. On voit aussi que notre application fonctionne efficacement sur plusieurs ordinateurs ainsi que sur smartphone.

Annexe

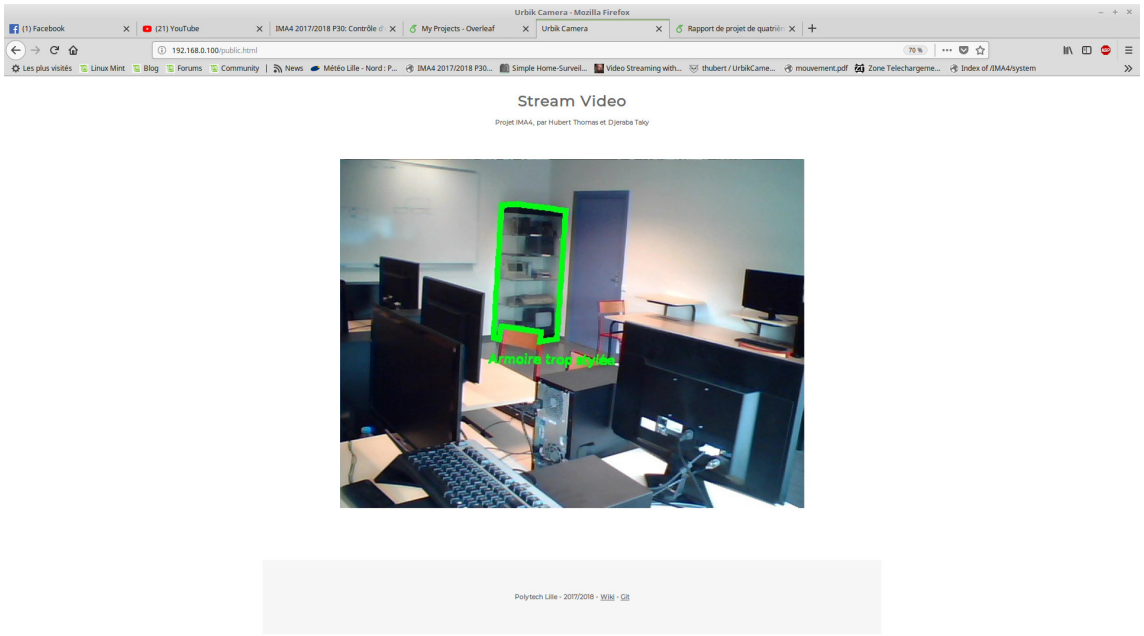

FIGURE 4.1 – Page publique avec incrustation

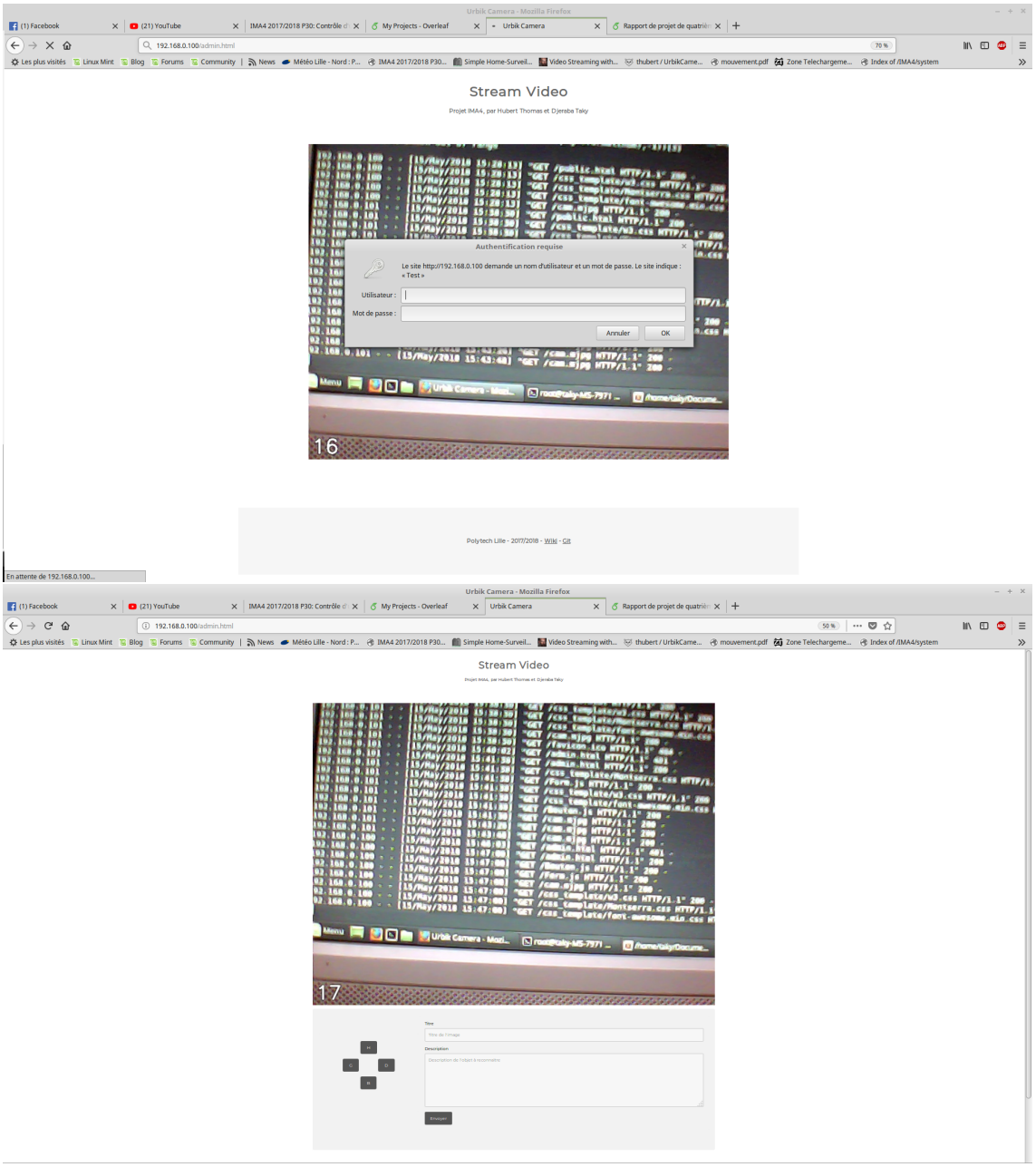

FIGURE 4.2 – Authentification et page admin

FIGURE 4.3 – Test de robustesse du serveur

Bibliographie

[1] bananaPi.org. Banana pi bpi-d1 faq. http ://forum.banana-pi.org/t/banana-pi-bpi-d1-faq/250.

[2] wikipedia.org. Opencv. https ://en.wikipedia.org/wiki/OpenCV.

[3] Ebrahim Karami Siva Prasad and Mohamed Shehata. Image matching using sift, surf, brief and orb : Performance comparison for distorted image. https ://arxiv.org/pdf/1710.02726.

[4] OpenCV.org. Introduction to surf. https ://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html.

[5] OpenCV.org. Introduction to sift (scale-invariant feature transform). https ://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html

[6] OpenCV.org. Orb (oriented fast and rotated brief). https ://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_orb/py_orb.html.

[7] rayryeng. What are keypoints in image processing ? https ://stackoverflow.com/questions/29133085/what-are-keypoints-in-image-processing.

[8] ARG. Why do we use keypoint descriptors ? https ://dsp.stackexchange.com/questions/10423/why-do-we-use-keypoint-descriptors.

[9] Miguel Grinberg. Projet flask. https ://blog.miguelgrinberg.com/post/video-streaming-with-flask.

[10] Python Software Foundation. Python socketserver. https ://-docs.python.org/2/library/socketserver.html.

[11] Thomas HUBERT et Taky DJERABA. Git du projet ima4. https ://archives.plil.fr/thubert/UrbikCamera.

[12] Thomas HUBERT et Taky DJERABA. Wiki de projet ima4. https ://projets-ima.plil.fr/mediawiki/index.php/IMA4_2017/2018_P30 :_Contr