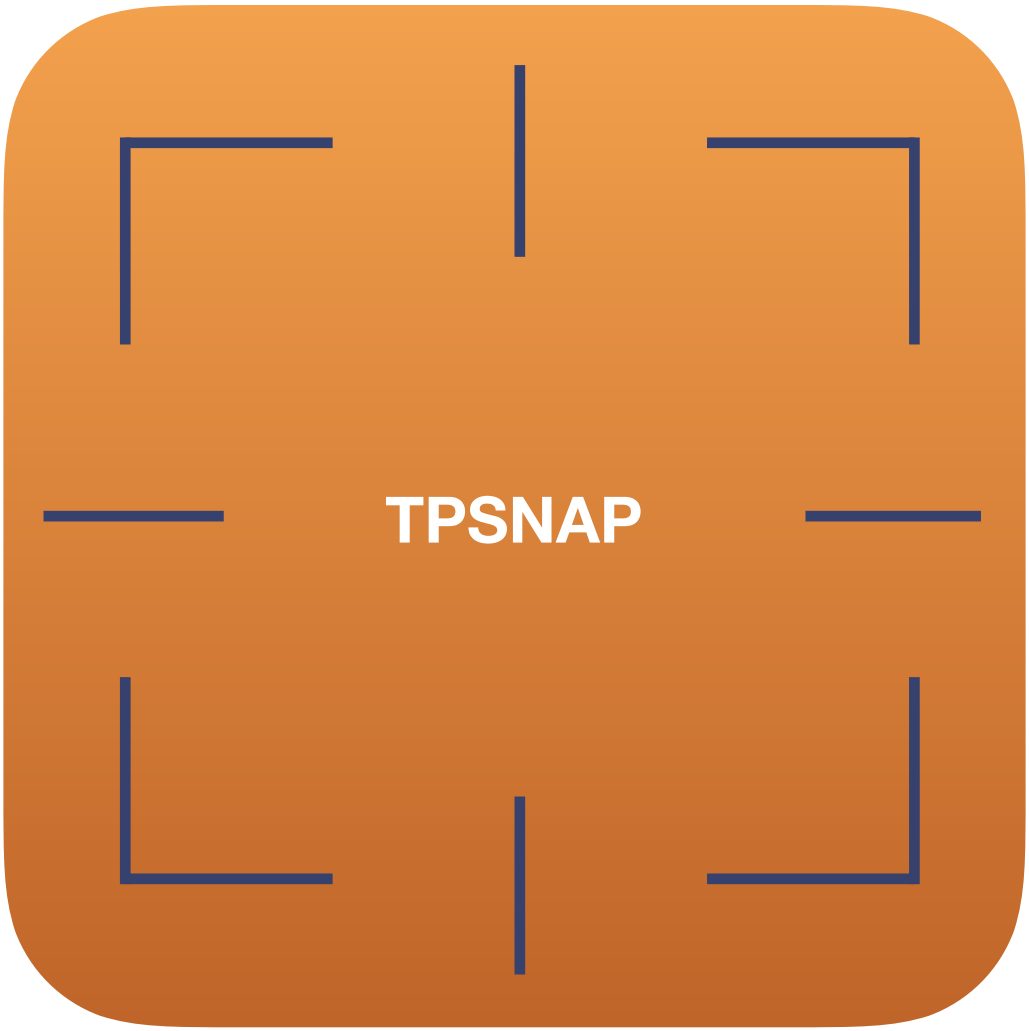


Documentation utilisateur TPSNAP

Préparé pour : Céline Meillier, Vincent Mazet
Préparé par : Groupe de projet : Fête de la science 2018
11 mai 2018

itii ALSACE

École d'ingénieurs
Télécom Physique
Université de Strasbourg

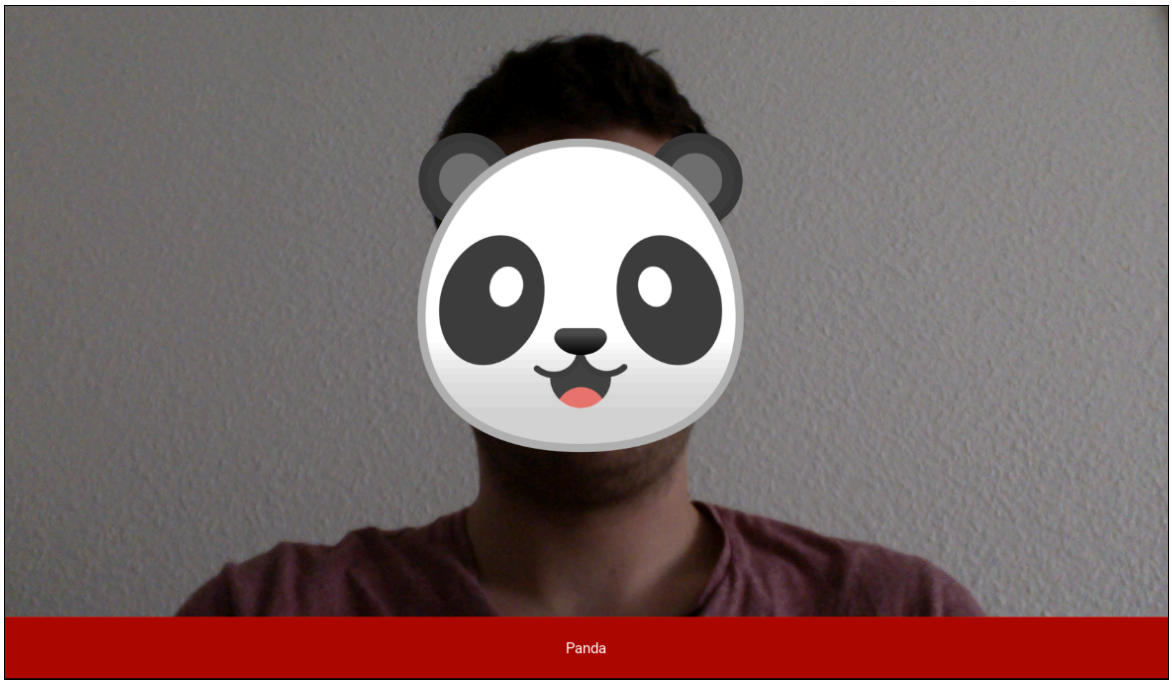
TABLE DES MATIÈRES

1. Présentation du programme

TPSNAP est un programme développé pour la fête de la science 2018, dans le cadre des projets de 2ème année FIP EII à Télécom Physique Strasbourg. Ce programme réalise une détection de visage et permet d'ajouter des filtres de différentes sortes au dessus de l'image en temps réel. L'acquisition de l'image est réalisée via une caméra/webcam. Le programme à été codé en Python et utilise une interface graphique « Kivy ». Cette interface à été simplifiée pour maximiser son ergonomie et permettre une utilisation facile.

Nom de l'application : TPSNAP : - TPS = Télécom Physique Strasbourg
- SNAP = En référence à la célèbre application « Snapchat » qui réalise des fonctions similaires

Aperçu :



L'interface dispose d'un seul bouton d'interaction. Celui-ci prend le nom du filtre sélectionné (par défaut = normal, sans filtre). Un clic sur le bouton déroule une liste avec les filtres disponibles, il suffit de cliquer sur le filtre voulu pour l'appliquer immédiatement.

2. Exigences de fonctionnement

Pour faire fonctionner le programme, voici les éléments requis :

- Un ordinateur sous Linux ou équivalent
- Une caméra/webcam connectée au PC
- La totalité du programme TPSNAP fourni :

Ceci comporte le main.py, le fichier pour la détection de visage haarcascade_frontalface_default.xml. A noter que ce dernier fichier doit rester à la racine du programme main.py. Le dossier « canevas » contenant deux dossiers « Face » et « Hat » dans lesquels se situent les filtres.

La suite des éléments requis est inclue dans un fichier « requirement.txt » à partir duquel on peut tout installer en une seule commande à l'aide de « *pip install* ».

- Python 3.6.4
- Cython 0.26
- Kivy 1.10.1.dev0
- Kivy-Garden 0.1.4 **EXPLIQUER CHAQUE MODULE**
- KiviSolver 1.0.1
- Numpy 1.14.2
- Opencv-python 3.4.0.12
- Pillow 5.0.0

Le lancement de l'application s'effectue par la commande : *python main.py*
L'arrêt de l'application s'effectue par une pression de la touche « Échap » ou en fermant la fenêtre principale.

3. Le code

Le seul fichier main.py permet le fonctionnement de l'application. Il a été commenté pour permettre une meilleure compréhension lors de sa lecture. Néanmoins, nous allons résumer son fonctionnement dans cette partie.

Le code comporte : - 3 classes **EXPLIQUER CHAQUE CLASSE BRIÈVEMENT AVANT**
- 13 fonctions

Pour un total d'environ 300 lignes.

3.1. Classe « canevas »

La classe « canevas » contient toutes les fonctions permettant de récupérer les filtres, ainsi que leur informations (nom du filtre, offset si il y en a un, type de filtre). **REFORMULER**

Les filtres sont divisés en catégories :

- Face, pour un filtre placé sur le visage
- Hat, pour un filtre sur la tête de type chapeau
- None, pour les autres filtres*

*d'autres catégories ont été préprogrammés si l'on souhaite en ajouter, comme « scene » qui pourrait changer tout l'arrière plan en gardant le visage.

Fonctionnement des différentes fonctions de la classe :

GetName

Arguments en entrée : *self*, et *fullpath* (chemin du fichier)

Éléments de sortie : *string* contenant le nom du filtre

Cette fonction réalise un simple découpage du chemin du fichier pour récupérer son nom. Ainsi le nom du fichier est important, c'est lui qui se retrouvera sur le bouton.

GetType

Arguments en entrée : *self*, et *fullpath* (chemin du fichier)

Éléments de sortie : *string* contenant le type du filtre

La fonction cherche dans quel dossier se trouve le filtre. Le nom du dossier détermine le type du filtre.

GetOffset

Arguments en entrée : *self*, et *fullpath* (chemin du fichier)

Éléments de sortie : *float* contenant l'offset du filtre

Pour les filtres de type HAT. En fonction de l'image choisie pour le filtre, celui-ci peut se retrouver décalé par rapport à la tête (trop haut ou trop bas). Chaque filtre est unique et doit pouvoir être pré-réglé pour corriger cet offset. Ainsi, pour ce type de filtre, nous avons donné la possibilité de modifier cet offset directement dans le nom du filtre de la manière suivante : nom_du_filtre$offset.png. La fonction GetOffset récupère ce chiffre qui doit être dans l'intervalle]0;1[.

EXPLIQUER 0 ET 1

ToString

Arguments en entrée : *self*

Éléments de sortie : *string* contenant un description de l'objet **SUPPRIMER?**

3.2. Classe « KivyCamera »

La classe « KivyCamera » contient tout le processus de détection de visage, ainsi que l'application du filtre.

SetImage
Arguments en entrée : *self*, et *img* (l'image du filtre) **DONNER TYPE (matrice?)**
Éléments de sortie : *void*
Cette fonction sert de lien entre les classes. Ainsi on récupère le bon filtre lors de l'appel.

SepiaFilter
Arguments en entrée : *self*, et *frame* (l'image actuelle récupérée de la caméra) **DONNER TYPE**
Éléments de sortie : *void*
Cette fonction permet d'appliquer le filtre sépia à l'image en cours.

Make_Linear_Ramp
Arguments en entrée : *white* (vecteur R,G,B)
Éléments de sortie : *ramp* **DONNER TYPE**
Cette fonction crée une rampe linéaire avec en entrée une matrice de trois entiers R G et B. En choisissant précisément ces paramètres, on réalise la conversion en sépia. D'autres valeurs peuvent changer la couleur. Dans notre cas, un sépia optimal comporte les paramètres suivants : R = 255, G = 240, B = 192. **EXPLIQUER**

UnfilteredTexture
Arguments en entrée : *self*, *frame*
Éléments de sortie : *void*
Cette fonction affiche l'image sans traitement.

Update
Arguments en entrée : *self*
Éléments de sortie : *void*
Update est la fonction principale, elle prend la dernière image de la caméra, réalise la détection de visage et applique le filtre. En résumé elle met à jour l'image en fonction du filtre sélectionné.

3.3. Classe « CamApp »

La classe CamApp construit l'IHM. Elle génère les boutons en se servant de « canevas » et fait appel à « KivyCamera » pour créer la fenêtre d'affichage.

Btn_Click

Arguments en entrée : *self*, *component* (un bouton)

Éléments de sortie : *void*

Cette fonction permet de réaliser une action sur un objet (dans notre cas de type bouton), lors d'un clic.

Build

Arguments en entrée : *self*

Éléments de sortie : *root* (l'IHM)

La fonction build construit toute l'IHM, elle crée le gros bouton principal. Ensuite, en fonction du nombre de filtres disponibles dans « canevas », elle va créer une nombre X de sous-boutons dans la liste, avec les noms des filtres associés.

OnStop

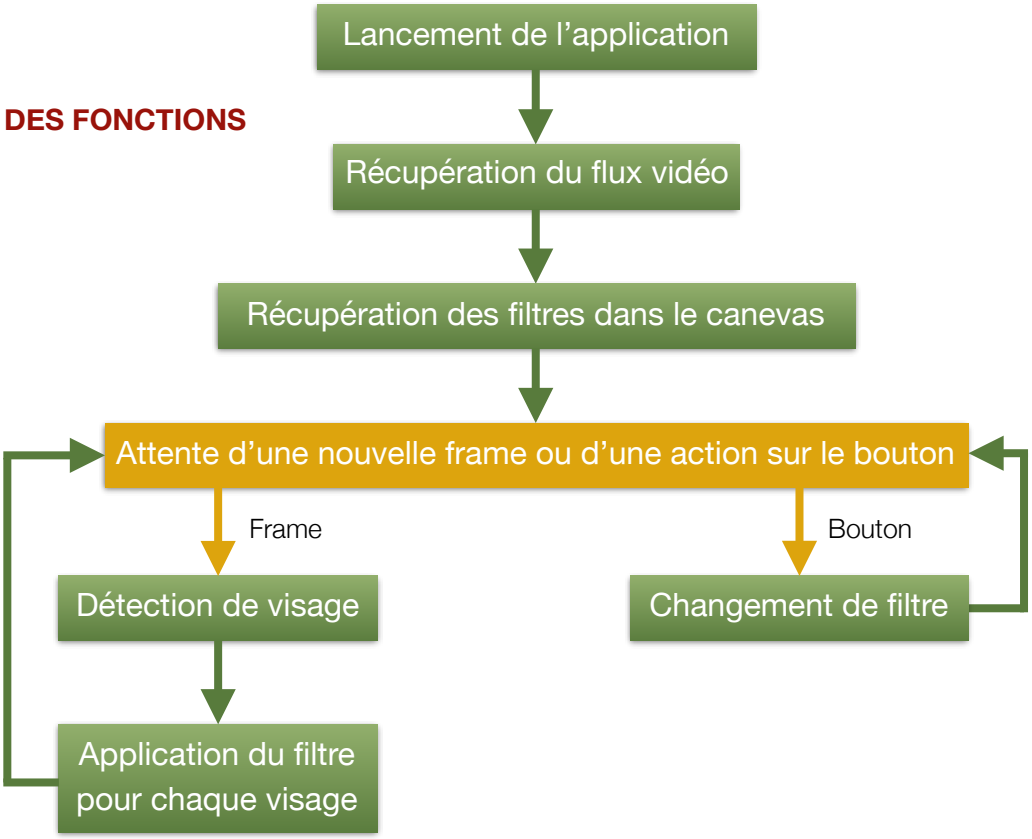
Arguments en entrée : *self*

Éléments de sortie : *void*

OnStop permet simplement d'arrêter la capture d'image à la fermeture de l'application.

4. Organigramme de fonctionnement

+ORGA DES FONCTIONS



5. Justification des choix

Dans cette partie, nous allons expliquer pourquoi nous avons décidé d'utiliser tel ou tel outil dans notre programme.

5.1. Kivy

Kivy est une librairie **python OpenSource** pour le développement **rapide** d'applications avec des interfaces **innovantes**. Ces idées nous ont séduit et lors de nos recherches, nous avons apprécié les réalisations que d'autres utilisateurs ont produit. Kivy est également **Cross-platform**. Cela permet le développement sur Windows, Linux, Mac et d'obtenir toujours le même rendu. En résumé, c'est sa **simplicité** d'utilisation, sa compréhension **facile**, son **design** et sa **multi-compatibilité** qui nous on fait choisir Kivy.

5.2. Pillow

La nécessité de Pillow n'est apparue que plus tard dans le développement de TPSNAP. En effet, c'est lors de l'implémentation du filtre « sépia » que nous nous sommes rendus compte que le programme ralentissait énormément en raison du traitement d'image. Avant, on traitait chaque pixel de l'image pour lui appliquer une correction. Cela demandait beaucoup de ressources et causait le ralentissement. En convertissant au préalable la frame au format PILimage, on accède à des fonctions de PIL telles que « *putpallete* » qui permet de réaliser la fonction voulue, mais de manière **optimisée** et **rapide**. Maintenant, le filtre est totalement fluide et le rendu est satisfaisant.

5.3. OpenCV

OpenCV est une librairie **OpenSource** dans le traitement d'image. C'est même une librairie de **référence** dans le domaine, ses fonctions sont reconnues pour être **faciles** à la mise en oeuvre et le résultat fonctionne parfaitement. Nous l'avons choisi car cette librairie ne nous était pas inconnue, mais également car elle se prête très bien à l'**implémentation** en Python. Enfin, on peut envisager beaucoup d'**améliorations** du programme en utilisant d'autres fonctionnalités d'OpenCV.

5.4. Numpy

Notre programme traite les images comme des matrices. Ainsi les différentes fonctions réalisent des calculs avec ces matrices. Le meilleur outil pour le calcul matriciel en Python est Numpy. Il est reconnu dans de nombreux domaines pour être la référence en calcul scientifique.

6. Limites et améliorations possibles

De nombreuses améliorations peuvent être apportés à TPSNAP pour rendre la détection de visage plus efficace, et en ajoutant d'autres fonctionnalités à partir de notre code qui n'attends que d'être complété dans ce but.

Nous avons remarqué que la détection de visage était compromise lorsque le visage était penché. En effet, le programme ne cherche que des visages « droits » dans l'image. Une idée que nous avons eu pour corriger ce problème est de faire pivoter l'image dans un sens et dans l'autre et de réaliser la détection à nouveau sur l'image pivotée. En réalisant des recherches, nous avons remarqué que c'est exactement la manière employée par des applications plus complexes pour palier à ce problème. L'implémentation est tout à fait réalisable avec notre programme. Cela permettrait également de faire pivoter les filtres avec la tête.

Pour améliorer l'expérience, on peut combiner les détections. Après avoir détecté un visage, on peut réaliser une détection des yeux dans la zone du visage, et ainsi changer la couleur de yeux, ou mettre des lunettes avec précision. On peut détecter le nez dans cette même zone ou même les oreilles. Le changement du type de détection se fait en faisant appel à un autre fichier que haarcascade_**frontalface**_default.xml. Le « FrontalFace » ici nous indique qu'il s'agit d'un fichier de détection des visages de face. Mais d'autres fichiers existent pour détecter par exemple la tête d'un chat, un corps, un visage de profil, etc. Une liste non exhaustive de ces fichiers peut être récupérée à cette adresse : https://github.com/opencv/opencv/tree/master/data/haarcascades.

Enfin, en réalisant une détection de la bouche et des yeux, on pourrait faire « bouger » le filtre pour avoir l'impression de parler avec le filtre.