Pierre ETIENNE *13/01/2017*

INNOVALAB : Projet TEO

***Résumé** : Le projet TEO a pour but l'intégration d'un software embarqué sur Raspberry Pi permettant la détection et l'identification de panneau de circulation par l'intermédiaire de la PiCamera.*

Introduction

Les aides à la conduite (ADAS – « *Advanced Driving Assistance Systems* ») constituent un des enjeux primordiaux du futur de l'automobile. Le système TEO a pour ambition d'être adaptable à tout type de véhicule en ce sens où les algorithmes de traitement d'image sont embarqués dans une caméra intelligente que l'on pourra aisément accrocher ou décrocher du pare-brise. Les premières fonctions intégrées à cette caméra permettent la signalisation d'un danger comme notamment l'alerte d'un franchissement d'une ligne de marquage au sol ou encore la détection d'un panneau de circulation. C'est ce dernier point qui fait l'objet du présent document. L'algorithme choisi s'articule autour de deux grandes étapes à savoir :

- La détection de formes (cercle, triangle, quadrilatère) ;
- La reconnaissance des panneaux à partir des formes extraites.

La suite du document explique en détails l'algorithme et son implémentation en python. En effet, dans un souci d'optimisation temps réel, le langage choisi pour le développement du programme est le python qui permet l'accès à la librairie de *Computer Vision* OpenCV.

Matériel – Configuration

Le matériel mis à disposition pour le développement du projet est un Raspberry Pi 3 équipé d'une PiCamera. Le système d'exploitation du Raspberry est **Raspbian Jessie** (contraction entre Raspberry et Debian). La librairie OpenCV3 (+NumPy) a été installée et compilée pour un développement en python. La version de Python utilisée est la *3.4.2*.

On peut travailler de deux manières différentes sur Raspberry :

- **En local** à condition d'avoir le matériel adéquat soit : un écran relié au raspberry par un câble HDMI et un clavier(+souris) USB. Une fois le matériel installé, et le raspberry branché avec le câble d'alimentation, une étape d'identification peut être demandée. Par défaut, les identifiants sont les suivants :
  - **Nom d'utilisateur :** pi
  - **Mot de passe :** raspberry

  [Suite à l'identification, si l'interface n'apparaît pas (seulement une invite de commande) : il suffit de taper la commande **startx**.]
- **Via une connexion SSH sur votre ordinateur.** Sur Windows, cela nécessite l'installation des logiciels **Putty** et **FileZilla** (pour le partage de dossiers). Sur Linux, la simple ligne de commande « ssh –option <nom_utilisateur>@<adresse_ip> » permet d'arriver à ses fins. L'adresse IP du raspberry peut être obtenue avec la commande **ifconfig**. Afin de valider la connexion, les même identifiants mentionnés ci-dessus seront demandés.

  **/ ! ** en SSH, il est plus difficile d'observer les résultats obtenus directement sur la caméra, problème d'interface graphique.

Pour plus de détails sur la prise en main du raspberry : liens utiles à la fin du document.

Un environnement virtuel a été créé sur le Raspberry pour simplifier le déroulement du projet. Cela nécessite donc des manipulations avant l'exécution des fichiers.

1. Ouvrir un terminal ;
2. Entrer les lignes de commande suivantes :

```
pi@raspberrypi:~ $ source ~/.profile
pi@raspberrypi:~ $ workon cv
(cv) pi@raspberrypi:~ $
```

Le « (cv) » indique que l'on travaille actuellement sous l'environnement virtuel adapté pour Python où toutes les librairies sont correctement installées. Cette étape est très importante et permet d'éviter de nombreux bugs.

Détection de formes

Cette section a pour objectif l'explication de l'algorithme de détection de formes ainsi que le script Python associé.

Algorithme. Le bloc de traitement « détection de forme » repose dans un premier temps sur une sélection de régions d'intérêt. En effet, dans l'optique de la détection de panneau de circulation, il apparaît trivial que les couleurs concernées dans notre image initiale seront uniquement le bleu et le rouge (si on part du principe que l'on considère uniquement les panneaux d'obligation, d'interdiction, d'indication et de danger sans prendre en compte les panneaux de travaux). Le premier traitement à effectuer est donc une sélection des pixels rouges et bleus en effectuant en quelque sorte un seuillage de l'image avec les gammes de couleur utiles. La sortie obtenue est par conséquent une image binaire où les pixels d'intérêt sont à 1 et les autres à 0. Par la suite, il faut donc détecter les contours de l'image binaire et les analyser pour pouvoir les labéliser en fonction de leurs caractéristiques. Ainsi, on parvient à détecter des formes en ce sens où si la détection de contours nous renvoie 4 segments on peut affirmer que la forme est un quadrilatère (3 pour un triangle, etc…).

Implémentation. Dans le dossier PET-TEO, qui fait pour le moment office de dossier principal du projet, on peut retrouver un dossier *shapeDetection* (dans lequel on retrouve un fichier *shapedetector.py*) et un fichier *detect_shapes_stream_V2.py*. Le dossier constitue donc un module dans lequel on va retrouver la fonction qui permet d'analyser les contours obtenus et de renvoyer le nom de la forme détectée. Le fichier *detect_shapes_stream_V2.py* quant à lui constitue en quelque sorte le fichier principal que l'on va exécuter pour lancer le programme.

shapedetector.py : dans ce fichier, on définit la fonction « detect » qui prend en entrée le vecteur des points de contours préalablement détectés et qui renvoie la chaine de caractère correspondant à la forme identifiée. La détection repose sur la fonction de la librairie OpenCV **approxPolyDP** qui permet de faire une approximation des contours et ainsi d'en extraire les points d'intersection. La labélisation qui s'en suit est basée sur le nombre de points récupérés : 3 → triangle, 4 → quadrilatère, etc…

detect_shapes_stream_V2.py : le fonctionnement de ce script réside dans la boucle **for** qui capture le flux vidéo de la caméra « frame » par « frame » et également dans le module **PiRGBArray** qui permet de convertir automatiquement chaque frame reçu en matrice RGB pour simplifier le traitement par la suite : en effet, traiter des « array » est très simple en python grâce à la librairie numpy qui permet la manipulation de matrices. Revenons plus en détails sur le script : après avoir importé les packages adéquates (**L1-8**) et initialisé les différents paramètres de la caméra (**L10-14**), on définit les gammes de couleur *min* et *max* du bleu/rouge en HSV pour le futur traitement de détection

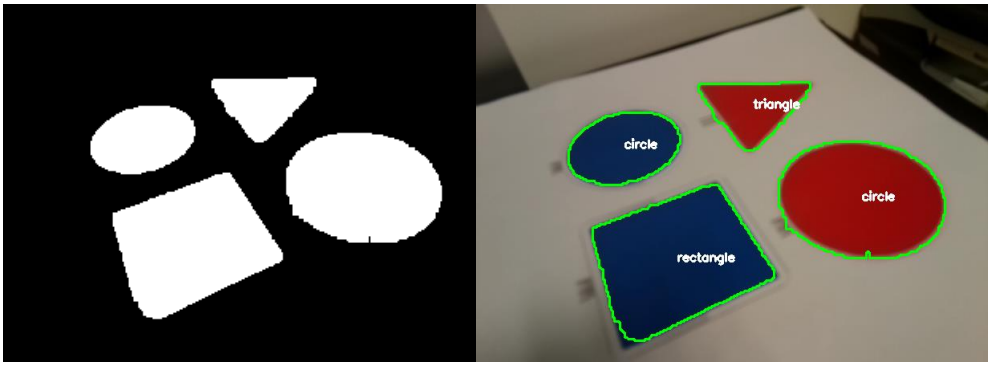
de couleur. **L29** : début de la boucle, on récupère chaque image du flux vidéo (« **capture_continuous** »), on redimensionne pour un traitement moins lourd et plus précis (**L36-37**) et la première étape majeure du traitement est le changement de système de couleur BGR<->HSV grâce à la fonction de cv2 : **cvtColor, mode = COLOR_BGR2HSV**. **L42-53** : création des masques pour récupérer uniquement les pixels de couleur rouge/bleu, la fonction **inRange** de OpenCV remplit parfaitement le rôle en prenant comme argument l'image dans le domaine HSV ainsi que les gammes précédemment définies → la fonction vérifie simplement si une valeur de pixel se situe dans l'intervalle qu'on lui donne en argument. L'utilisation de la fonction **findContours** (**L60**) renvoie les différents points de contours détectés sous forme de vecteur: on parcourt chaque contour et on fait appel à la fonction *detect* définie plus haut (**L72**). Le calcul des moments (**L68-71**) permet de déterminer les coordonnées du centre de la forme (pour y afficher son nom). Finalement, une fois les contours détectés et la forme identifiée, on les intègre sur l'image avec les fonctions **drawContours** et **putText** dont les noms sont explicites quant à leur utilisation (**L79-80**). L'affichage de l'image de sortie se fait à la ligne 83 avec la fonction **imshow**. Le traitement se fait de manière continue jusqu'à ce qu'on force le système à quitter en appuyant sur la touche « q » du clavier > **break** (**L93-94**). Important : la fonction *truncate* permet de libérer l'espace occupé par le frame pour le suivant, elle est indispensable pour le bon fonctionnement du code (si cette fonction n'est pas appelée à la fin de la boucle, Python renvoie une erreur) (**L87**).

Le lancement du programme se fait avec la ligne de commande suivante :

```
(cv) pi@raspberrypi:~ $ python detect_shapes_stream_v2.py
```

PS : il faut bien se trouver dans l'environnement (cv) !

On obtient les résultats suivants :



A ce stade de l'implémentation, on est déjà en mesure de catégoriser un panneau : effectivement, la détermination de couleur associée au nom de la forme permet d'indiquer si l'objet appartient à la classe des interdictions, obligations, … Ex : si la forme détectée est un triangle rouge, on peut affirmer qu'il s'agit d'un danger et ainsi restreindre la reconnaissance du panneau à cette unique catégorie. C'est l'objet de la prochaine partie qui vise à déterminer la nature propre du panneau en comparant notre forme extraite à une base de donnée (technique dite de *template matching*).

NB : le script implémenté ici constitue une version bêta très peu optimisée pour des conditions réelles : il a néanmoins le mérite de proposer une structure de code fonctionnelle. L'objectif sur le terme est de créer un script pour chaque étape de traitement pour diviser les blocs et les optimiser individuellement.

Reconnaissance des panneaux (à développer)

Les idées développées ici sont purement théoriques et n'ont pas été implémentées à ce jour, seulement simulées via le logiciel de calcul Matlab.

Ce second bloc de traitement vise à extraire la portion d'image correspondante à la forme identifiée ci-dessus et à la comparer à une base de données préalablement créée. Cette méthode, appelée *template matching*, calcul le taux de similitude (calcul de corrélation) entre l'image d'entrée et chacune des images de la base de données et renvoie l'image de la BDD qui correspond au maximum de ce calcul de corrélation. Robuste ?

Liens utiles

Prise en main du raspberry pi : http://raspbian-france.fr/tutoriels/

Algorithme d'extraction de panneau de signalisation :
http://recherche.ign.fr/labos/matis/pdf/articles_conf/2010/soheilian_rfia2010.pdf

Tutoriels de Computer Vision : http://www.pyimagesearch.com/