Rapport du Travail Encadré de Recherche

« Commande gestuelle d'un drone »

Liva Ralaivola

Quentin Cheynet, Cyriaque Serre, Yoann Moisset

Soutenu le 02 juin 2014

Première année de Master informatique – Aix Marseille Université – Campus de Luminy - 2013/2014

Table des matières

Remerciements

Nous tenons particulièrement à adresser nos remerciements à Monsieur Liva Ralaivola pour nous avoir encadré, orienté, aidé et conseillé tout au long de notre projet ainsi que pour sa gentillesse, sa confiance et sa pédagogie.

Nous tenons également à remercier l'ensemble des personnes de l'équipe de recherche QARMA (éQuipe AppRentissage et MultimédiA) pour nous avoir confié leur drone, aux multiples surnoms, afin de nous permettre de travailler.

Nous remercions le LIF (Laboratoire d'Informatique Fondamentale) pour nous avoir accueilli dans leurs bureaux.

Enfin, nous tenons à remercier tous nos professeurs pour la qualité de leurs enseignements. Ces derniers nous ont permis d'acquérir des connaissances qui nous ont aidées à mener à bien notre projet.

Introduction

Le TER (Travail Encadré de Recherche) est une UE (Unité d'Enseignement) clôturant notre première année de Master informatique d'Aix-Marseille Université. Le but est d'effectuer un projet d'une durée de deux mois et demi afin d'utiliser sur un cas pratique les notions présentées en cours. Le travail s'effectue sous la direction pédagogique d'un enseignant, et donne lieu à la rédaction d'un rapport et d'une soutenance orale.

Les intervenants de ce TER sont :

- Enseignant-chercheur et chef de l'équipe QARMA (éQuipe AppRentissage et MultimédiA) :
  - Liva RALAIVOLA
- Étudiant en première année de Master informatique :
  - Quentin CHEYNET
  - Yoann MOISSET
  - Cyriaque SERRE

Le sujet de notre projet s'intitule « Commande gestuelle d'un drone » et consiste à développer un programme de commande gestuelle pour le drone ARDrone 2.0 appartenant à l'équipe QARMA du LIF (Laboratoire d'Informatique Fondamentale) et conçu par la société française Parrot, connue notamment pour concevoir, développer et commercialiser des objets connectés à la fois de haute technologie et grand public.

L'idée générale est la suivante : le drone doit réaliser des actions (décoller, atterrir, etc) à partir d'ordres donnés sous la forme de gestes.

Pour réaliser cela, on effectue un geste devant la webcam d'un ordinateur ou devant la caméra du drone. Le flux vidéo est traité par un ordinateur qui est chargé de reconnaître le geste. Cette reconnaissance s'effectue à l'aide de la bibliothèque graphique libre OpenCV (Open Computer Vision) spécialisée dans le traitement d'images en temps réel dont la maintenance est assurée par la société de robotique Willow Garage. Une fois le geste reconnu, l'ordinateur envoie au drone, via le wi-fi, la commande associée à ce geste. Enfin, le drone exécute cette commande.

Notre rapport se compose de deux grandes parties et d'une conclusion. Dans la première partie, nous présenterons les tenants et les aboutissants du projet, nous exposerons nos choix en matière de gestion de projet et expliquerons les contraintes avec lesquelles nous avons dû composer.

Puis, dans la deuxième partie, nous expliquerons en détails le travail réalisé tout au long du projet. Nous présenterons les différentes étapes de la reconnaissance des gestes et du pilotage du drone, ainsi que le programme final.

Enfin, nous ferons un bilan du projet en exposant ce qui a été réalisé, ce qui n'a pas été fait et pourquoi, et ce que nous a apporté le projet.

I / Description détaillée du projet

1 / Objectif

Le projet consiste à créer un programme qui servira à piloter le drone grâce aux gestes de la main. Le pilotage doit pouvoir se faire avec la caméra d'un ordinateur, et la caméra frontale du drone. Il y a donc deux buts principaux : il faut d'une part développer un programme de reconnaissance gestuelle, et d'autre part faire bouger le drone en fonction du geste reconnu.

2 / Nos connaissances

Nous avions quelques connaissances qui nous ont permis d'appréhender le projet plus facilement, grâce à certaines UE. C'est le cas de l' IAA (Initiation à l'Apprentissage Automatique), qui est utile pour la reconnaissance gestuelle, et de l'UE Génie Logiciel : en effet, le TER ne consiste pas seulement à faire un programme. C'est aussi un travail plus proche du monde professionnel et de la recherche. Il faut travailler en groupe pendant deux mois et demi, rendre un rapport détaillé de ce travail et le présenter à l'oral. Cela demande une bonne organisation et une approche adaptée du travail à réaliser.

3 / Organisation du travail

Nous étions donc trois à travailler sur ce projet. Nous avions convenu au début de se baser sur les méthodes agiles pour travailler, et de faire régulièrement des rapports. Pendant deux mois et demi, nous nous sommes vu presque tous les jours, nous avons travaillé plus ou moins sur les mêmes parties, et nous discutions régulièrement de ce qui avait été fait, de ce qui restait à faire, et de comment nous pourrions procéder.

Ce comportement nous a permis à tous d'avoir une connaissance précise de l'avancée des différentes parties du projet, et donc d'être capable d'effectuer n'importe quelle tâche à tout moment. Cela a rendu la rédaction de rapport moins utile, c'est pourquoi nous nous en sommes vite passé.

Pendant toute la durée du projet, nous n'avions pu disposer du drone que le mardi car il était aussi utilisé par des doctorants. Il a donc fallu s'organiser avec cette contrainte : nous nous concentrions sur la reconnaissance gestuelle les autres jours, et nous faisions en sorte d'avoir des jeux de tests pour le drone prêts le mardi dans le but d'optimiser notre temps de travail.

4 / Méthode de développement

Lorsque nous devions nous partager le travail à faire, nous nous sommes généralement basés sur la méthode XP (eXtrem Programming) : nous formions un binôme et un monôme (les personnes formant ces groupes changeaient).

Notre flexibilité nous a aussi permis de travailler à trois sur la même tâche lorsque nous rencontrions un gros problème, ou de travailler sur trois parties complètement différentes si cela ne gênait pas l'appropriation du code par l'ensemble du groupe. Ceci nous a également permis de minimiser le risque de ralentissement du projet si une personne venait à être absente pour diverses raisons.

Lorsque nous attaquions une nouvelle étape, nous effectuions d'abord des recherches pour trouver des programmes que nous pourrions réutiliser, ou pour voir quelles méthodes d'autres personnes avaient utilisées, et quels problèmes avaient été rencontrés. Ainsi, nous avions adopté une

approche d'ingénieur, comme nous l'avait conseillé Liva Ralaivola.

Une fois que des éléments intéressants étaient trouvés, nous commencions rapidement à créer quelque chose. Nous sommes donc allés au plus simple, afin d'avoir un programme fonctionnel, que nous pourrions modifier et améliorer par la suite, et pour lequel nous connaissions déjà les problèmes rencontrés.

Nous avons donc adopté un cycle de développement en spiral afin de progresser par incrémentations successives de versions du prototype. Les itérations n'avaient pas de durées définies, mais nous reprenions régulièrement le programme total. Ainsi, le programme a reçu deux grosses restructurations. Lors de la deuxième restructuration, le design pattern State et une interface graphique ont été rajoutés, car le programme pouvait être exécuté de trois manières différentes (pour apprendre des gestes, pour piloter avec la webcam, et pour piloter avec la caméra du drone).

5 / Outils utilisés

Il nous a été conseillé de travailler avec la bibliothèque OpenCV de vision par ordinateur. Cette bibliothèque peut s'interfacer avec les langages C, C++, Java et Python. Elle fournit des fonctions importantes pour le traitement des images/vidéos, ce qui est nécessaire pour isoler une main sur un flux vidéo.

Pour communiquer avec le drone nous disposions de l'API (Application Programming Interface) fournie par Parrot. Cette interface de programmation offre un ensemble de fonctions qui permet de se connecter au drone, de recevoir les flux vidéo et d'autres informations (le niveau de batterie, l'angle d'inclinaison, ...), ainsi que d'envoyer des ordres aux drones (décollage, atterrissage, mouvements, contrôle des LED, …).

Nous avons choisi le C++ car nous avions une bonne connaissance de ce langage, et un langage orienté objet nous semblait avantageux pour un projet de cette envergure. De plus, l'API du drone est écrite en C : faire un programme qui liait le C et le C++ ne nous semblait pas être problématique. Pour finir, nous avions trouvé plusieurs programmes écrit en C++ qui utilisaient OpenCV et qui étaient liés à nos objectifs.

Durant cette période, nous avons également utilisé les outils Qt Creator et GitHub. Qt Creator est un environnement de développement intégré multiplate-forme faisant partie du framework Qt. Il a été utilisé pour programmer avec plus d'aisance en C++. GitHub est un service web d'hébergement et de gestion de développement de logiciels, utilisant le programme Git qui est un logiciel de gestion de versions.

II / Description du travail réalisé

Le travail se découpe principalement en deux parties : la reconnaissance des gestes d'une part, et le contrôle du drone via l' API d'autre part. Une vidéo de ce travail peut être visionnée à cette adresse : https://github.com/desk07/ter/blob/master/rapport/video_demonstration.mp4.

1 / La reconnaissance des gestes

La reconnaissance des gestes pour piloter le drone se fait par l'apprentissage automatique. Le but est de transformer l'image d'une main en une matrice de nombres qui pourra être utilisée par un classifieur. Le classifieur doit être capable d'attribuer un label à cette matrice : on aura par exemple le label 0 pour une main avec le pouce vers le haut, et le label 1 lorsque le pouce est dirigé vers le bas.

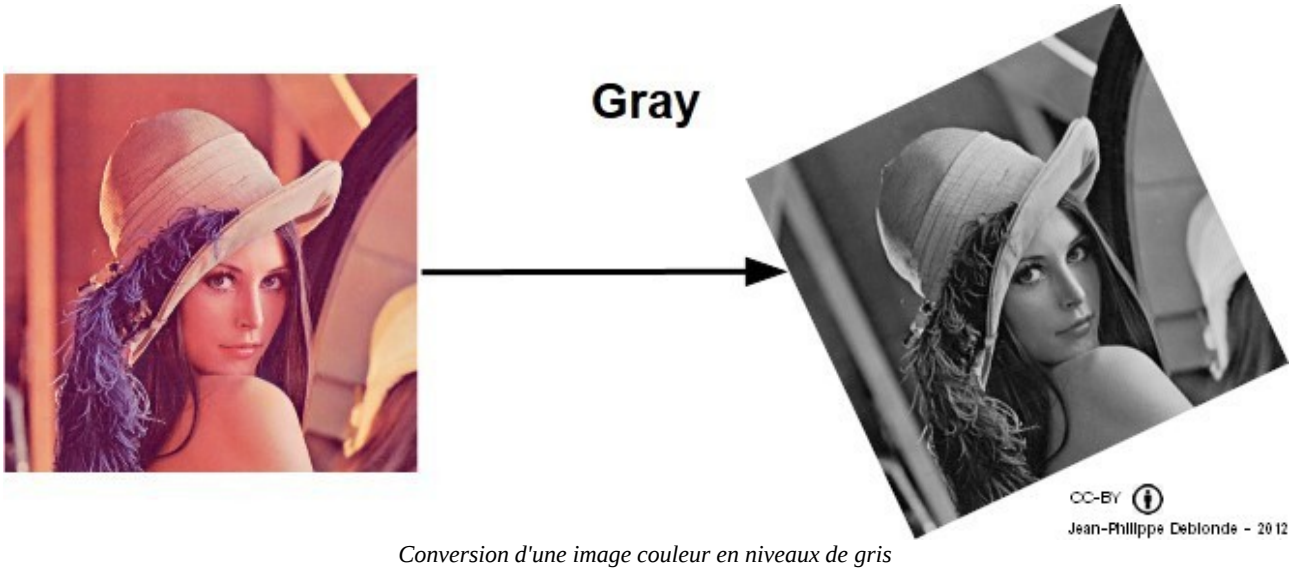
A partir de là, deux objectifs se sont posés : il faut trouver un moyen de transformer une image en matrice représentant correctement un geste, et trouver un classifieur adapté à cette reconnaissance.

A / Recherches sur le sujet

Dans un premier temps, nous avons mis en commun nos connaissances sur la reconnaissance d'image.

Dans la matière IAA , nous avons vu deux moyens assez simple pour reconnaître les images. Dans les deux cas, il faut d'abord convertir les images en niveaux de gris.

Le niveau de gris représente l'intensité lumineuse d'un pixel. La couleur du pixel peut prendre des valeurs allant du noir (0) au blanc (255) en passant par 254 niveaux intermédiaires.

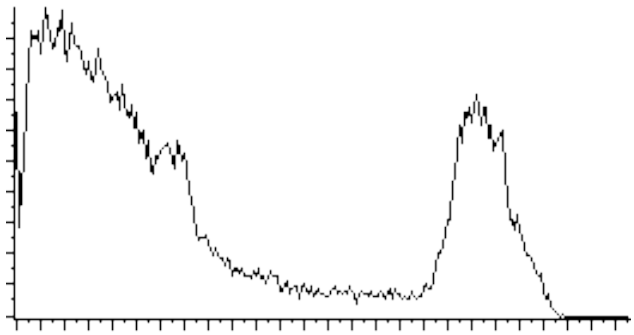


Conversion d'une image couleur en niveaux de gris

Le premier moyen représente les images à l'aide d'un vecteur : toutes les images ont la même taille, et chaque valeur du vecteur est égale au niveau de gris d'un pixel de l'image.

Image avec une matrice de 5x5 pixels contenant les valeurs de niveau de gris

La deuxième technique utilise un histogramme des niveaux de gris de l'image. Un histogramme est une courbe statistique indiquant la répartition des pixels selon leur valeur.



Histogramme des niveaux de gris correspondant à l'image ci-contre

Pour notre travail, nous avons plutôt pensé à découper l'image en régions, et à utiliser un histogramme par région, car un pouce dirigé vers le bas et un pouce dirigé vers le haut donne à peu près le même histogramme.

Dans un second temps, nous avons effectué plusieurs recherches sur la reconnaissance gestuelle. Nous sommes alors tombés sur le travail d'un étudiant, Ming Cai, dont le programme permet de reconnaître précisément les gestes des mains. Nous avions donc à disposition des vidéos de son travail, et un rapport de cinq pages qui a été fondamental pour l'avancée de notre programme.

Lien vers le rapport de Ming Cai

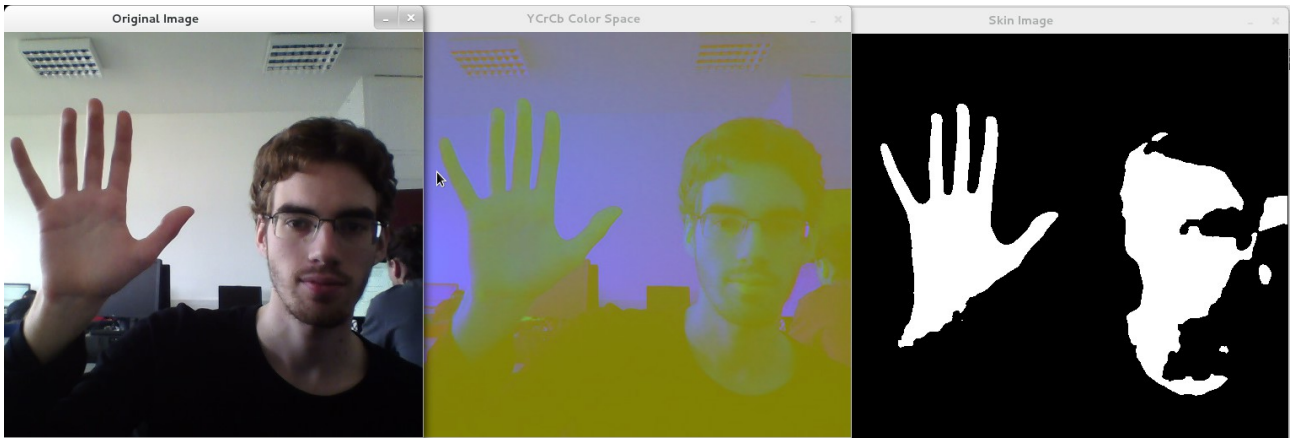
B / De l'image à la représentation du geste

1 / Trouve ta main !

Le problème qui s'est posé est l'isolation de la main : en effet, l'image capturée par la caméra est entièrement analysée, et la main a au final assez peu d'influence sur la représentation vectorielle de l'image.

a / Détection de la peau

Tous les programmes de reconnaissance gestuelle que nous avons trouvés utilisaient la binarisation de l'image pour isoler la main. Cette technique consiste à changer l'espace colorimétrique de l'image (on passe de RGB à HSV, ou YCbCr), à choisir une plage de couleur correspondant à la couleur de la peau, puis à colorier tous les pixels qui sont dans cette plage en blanc, et ceux qui ne le sont pas en noir.



De gauche à droite : image source, image en YCbCr, image binarisée

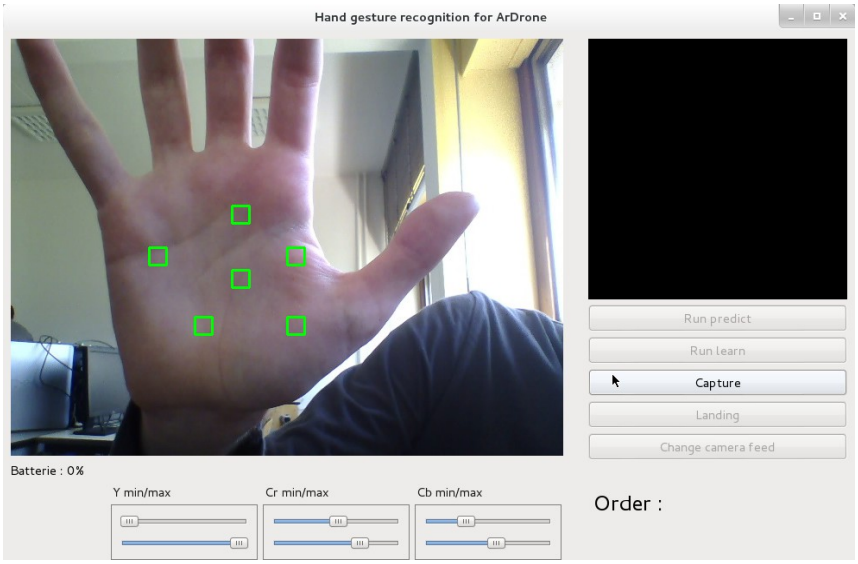
RGB est un format représentant les pixels avec trois valeurs : rouge, vert et bleu. HSV correspond à la teinte, la saturation et la valeur. Quand à YCbCr, il représente une image grâce à la luminance, la chrominance bleu et la chrominance rouge.

Nous avons essayé la conversion en HSV et YCbCr pour isoler la main, et YCbCr était bien plus efficace. Nous avons alors fixé des valeurs correspondantes aux couleurs de peau.

Cette technique permet d'isoler la couleur de la peau : le visage ne disparaît donc pas, et les éléments dont la couleur est proche de la peau apparaissent sur l'image. De plus l'image peut présenter un niveau de bruit plus ou moins élevé selon les conditions.

Le bruit d'image est la présence d'informations parasites qui s'ajoutent de façon aléatoire aux détails de la scène photographiée numériquement.

Plusieurs transformations sont alors effectuées sur l'image pour éliminer le bruit (dilatation, érosion, floutage). Un calibrage peut également être réalisé afin d'affecter les valeurs YCbCr de la main qui se trouve devant la caméra. La main doit couvrir les six zones représentées par des carrés verts afin de faire une moyenne. Cela permet également, par exemple, l'usage d'un gant de couleur afin de faciliter le calibrage et l'utilisation du programme dans des conditions d'utilisation extrêmes.



Zones de calibrage

Enfin, un réglage de la plage de couleur a été ajouté plus tard, afin d'éliminer avec précision le bruit tout en gardant la main sur l'image.

Curseurs permettant d'affiner la détection de la peau

Cette isolation garde cependant un défaut qui ne peut pas être corrigé : étant donné que la caméra renvoie une image 2D, il est impossible de différencier deux formes qui sont l'une derrière l'autre. Ces formes se touchent sur l'image 2D, et créent donc une seul forme composée par exemple de la main et du bruit, ou de la main et du visage.

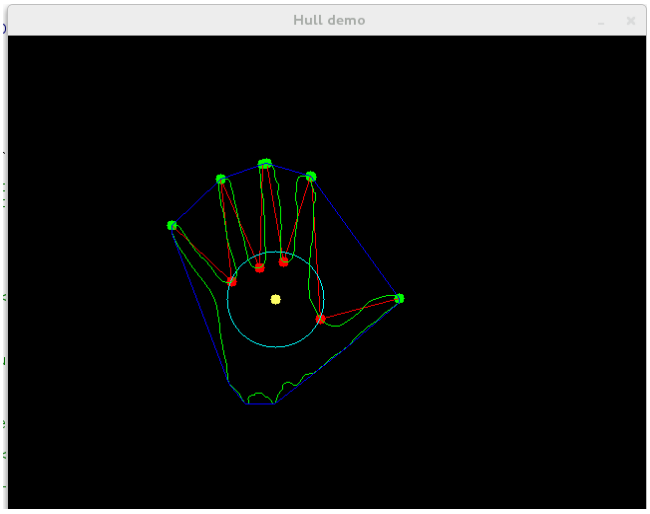
b / Extraction de la main

Le but est d'obtenir un carré entourant la main. L'image contenue dans ce carré servira ensuite au formatage de la main, format qui sera exploitable par le classifieur.

Pour obtenir ce carré, Ming Cai a localisé le centre de la paume, et dessiné un carré de taille fixe autour de ce point. Pour trouver la paume, il a utilisé l'enveloppe convexe de la main (le plus petit polygone convexe qui contient touts les points de la forme représentant la main), qui permet ensuite d'obtenir les points de convexité et les *defect points*.

Le centre du plus petit cercle contenant tous les *defect points* est considéré comme le centre de la paume.

Nous avons créé notre propre programme qui effectuait toutes ces étapes, et obtenu l'isolation de la main.



Localisation du centre de la paume (point jaune) grâce aux defect points (point rouge). Les points de convexité sont les points verts, l'enveloppe convexe est en bleu.

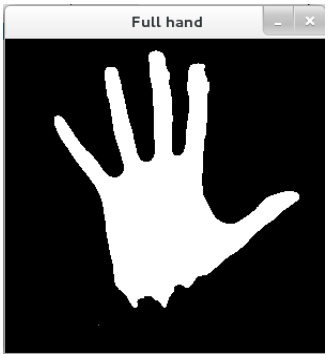
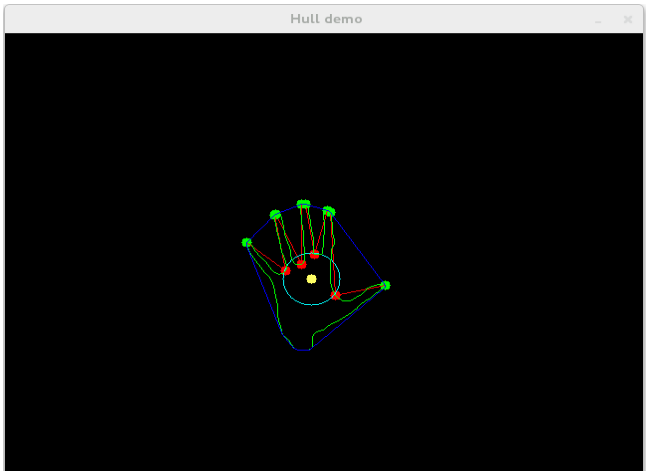
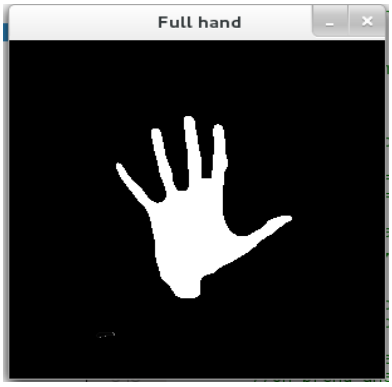


Image obtenu à partir d'un carré de 300x300 centré sur le point jaune.





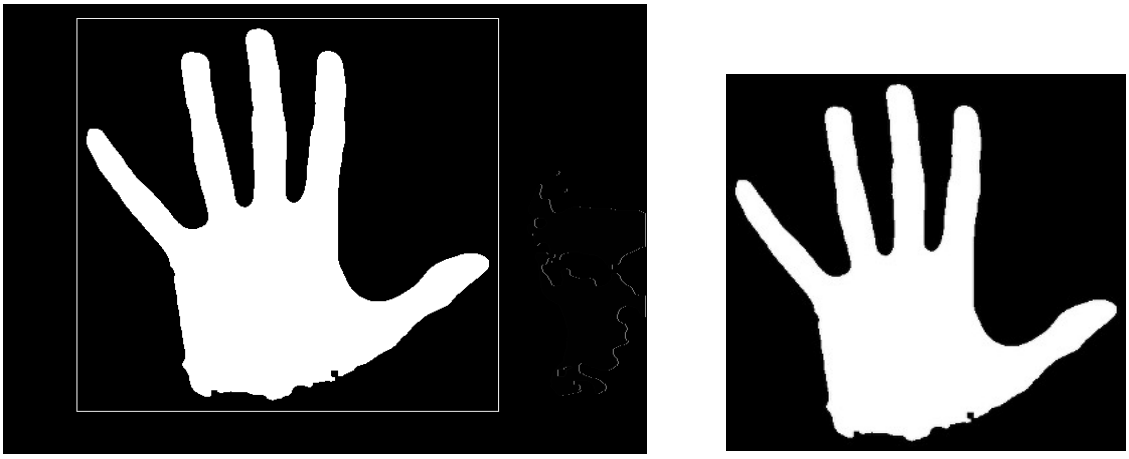
Pareil que précédemment, mais en étant plus loin de la caméra

Cependant, nous nous sommes rendus compte que tous ces calculs n'étaient pas utiles, et que localiser la main de cette manière posait des problèmes pour notre objectif.

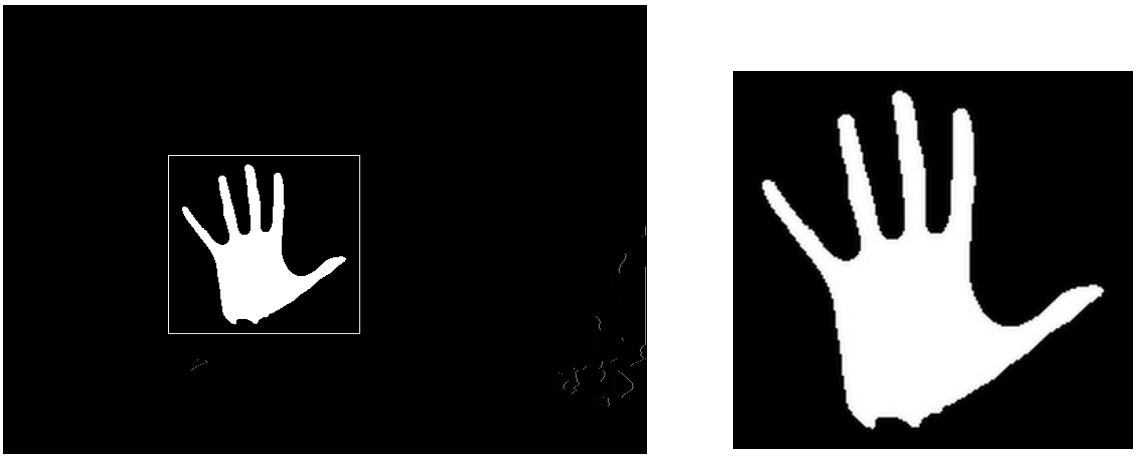
En effet, nous avions uniquement besoin d'isoler la main, pas de trouver le bout des doigts. Ceux-ci sont utilisés, dans le programme de Ming Cai, pour réaliser des actions comme dessiner et redimensionner une fenêtre. De plus, Ming Cai reste toujours à une certaine distance de son ordinateur, tandis que nous devons utiliser la caméra du drone, et donc être à des distances très variables de la caméra.

Nous sommes donc repartis de l'image binarisée et nous avons récupéré le contour des zones qui apparaissent en blanc sur l'image. Seul la zone possédant la plus grande surface est retenue, ceci permet de supprimer les éventuels bruits restants. Une seconde image est créée à partir du plus petit rectangle contenant touts les points du contour, que l'on agrandit de 10 pixels de chaque côté. Cette image est redimensionnée afin d'avoir un carré de taille fixe (300x300), sans déformer les proportions de la main. Toutes ces étapes sont réalisées avec des fonctions d' OpenCV.

Ainsi, quelle que soit la distance à laquelle on se trouve de la caméra, on obtient une image de 300x300 pixels, avec une main ayant toujours la même taille, située au centre de l'image. Cela permet d'améliorer les performances de reconnaissance. C'est cette image qui sera utilisée pour obtenir un vecteur représentant les gestes, et ainsi apprendre ou reconnaître des gestes.



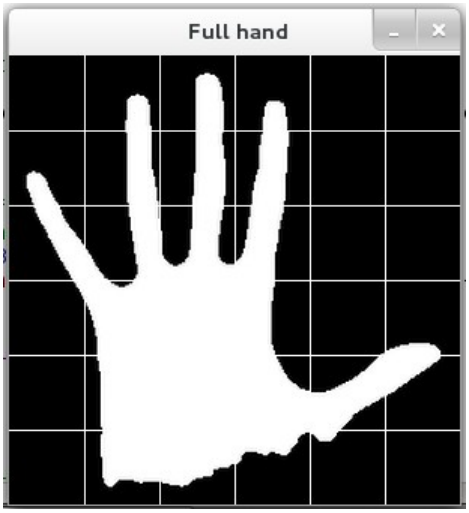
Localisation de la main grâce au contour de la forme, qui sert à former un carré de 300x300, qui sera utilisé pour l'image analysée



Pareil que précédemment, mais en étant plus loin de la caméra

2 / Formatage de la main

L'image, de 300x300 pixels, obtenue précédemment est ensuite quadrillée et on regarde dans chaque case s'il y a au moins un pixel blanc. Si c'est le cas, on rajoute un 1 au vecteur représentant l'image, sinon on rajoute un 0.



Quadrillage de la main pour obtenir le vecteur représentant la main

L'exemple ci-dessus donne le vecteur 011100 111100 111100 111111 011111 011110 exploitable par un classifieur.

Dans notre programme, nous avons choisi un quadrillage avec des cases de 30x30 pixels, ce qui donne 100 cases. Ces paramètres auraient pu être modifiés pour améliorer la reconnaissance mais nous avons obtenu de bon résultats avec, c'est pourquoi nous les avons gardés. Il aurait été intéressant de présenter les résultats en fonction des valeurs des paramètres (y compris ceux du classifieur) pour voir leurs influences sur les performances de la reconnaissance des gestes.

C / Apprentissage et reconnaissance

Ces vecteurs sont utilisés de deux manières différentes : pour apprendre de nouveaux gestes, et pour les reconnaître. Ces deux fonctionnalités utilisent un classifieur. Un classifieur permet d'attribuer une classe (ici le label 1, 2, 3 …) à un groupe d'objets (ici les vecteurs représentant les différentes positions de la main).

Choisir le bon classifieur et trouver les meilleurs paramètres est une étape importante du projet, qui doit être réglé assez tôt.

Nous avons donc suivi les décisions de Ming Cai : ce dernier a d'abord utilisé un réseau bayésien, mais cette méthode ne s'est pas révélée assez efficace. Il a alors utilisé un SVM (Support Vector Machine). Ce SVM peut être utilisé pour apprendre et reconnaître les gestes, et OpenCV possède déjà ce classifieur, ce qui le rend facile à implémenter.

1 / SVM (Support Vector Machine)

a / Principe général

Les SVM (Support Vector Machine) peuvent être utilisés pour résoudre des problèmes de discrimination, c'est-à-dire décider à quelle classe appartient un échantillon.

Formalisation :

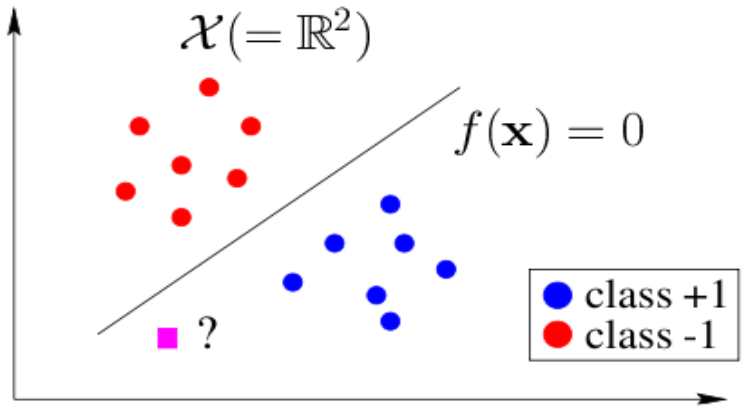
- Soit X (par exemple $\mathbb{R}^d$, d > 0) l'espace de données, appelé *espace d'entrée*.
- Soit Y (par exemple {-1, +1}) l'*espace d'arrivée*.

S = $\{(x_i, y_i)\}$ i allant de 1 à n et $x_i \in X$, $y_i \in Y$, représentant l'ensemble des données.

Afin de résoudre ce problème de discrimination, il faut construire une fonction (le classifieur)

$f : X \rightarrow Y$
$x \rightarrow w.x + b$ avec $w \in \mathbb{R}^d$ et $b \in \mathbb{R}$

tel que, ici dans notre exemple de discrimination binaire, h(x) = signe(f(x)) fasse peu d'erreurs. Autrement dit, que la probabilité que h(x) ≠ y, pour x et y indépendamment et identiquement distribués, soit petite.

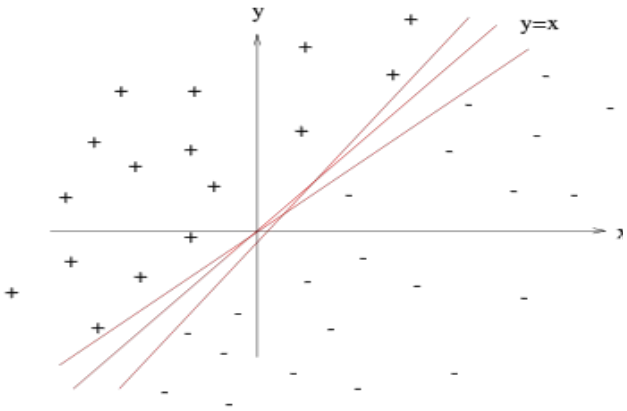


Exemple de discrimination binaire

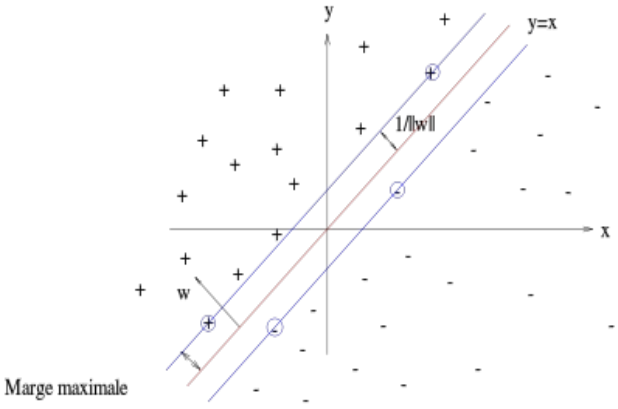
b / SVM à marge maximale (hard-margin SVM)

On se place dans le cas où le problème est linéairement séparable, c'est-à-dire que la fonction f définit précédemment est une droite. Même dans ce cas simple, il existe une infinité d'hyperplans séparateurs, dont les performances en apprentissage sont identiques, mais dont les performances en généralisation peuvent être très différentes. Pour résoudre ce problème, il a été montré[1], qu'il existe un unique hyperplan optimal, défini comme l'hyperplan qui maximise la marge entre les échantillons et l'hyperplan séparateur.

La marge est la distance entre l'hyperplan et les échantillons les plus proches. Ces derniers sont appelés vecteurs supports.



Il existe une infinité d'hyperplans séparateurs pour un ensemble de points linéairement séparables.



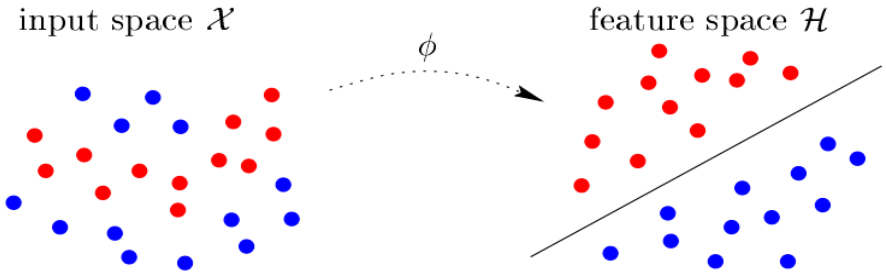
L'hyperplan optimal (en rouge) avec la marge maximale. Les échantillons entourés sont des vecteurs supports.

La formulation des SVM à marge maximale s'exprime sous la forme suivante :

$$\text{Min} \quad \frac{1}{2} \|w\|^2 \qquad \text{sous les contraintes,}$$
$$y_i (w.x_i + b) \geq 1 \quad \forall i$$

c / Cas non séparable : Kernel trick

Certains ensembles de données ne sont pas linéairement séparables. Sans entrer dans les détails, il faut construire une fonction de mapping Φ de telle sorte qu'en appliquant cette fonction aux données de l'ensemble d'entrée, les données de l'ensemble de sortie soient linéairement séparables.

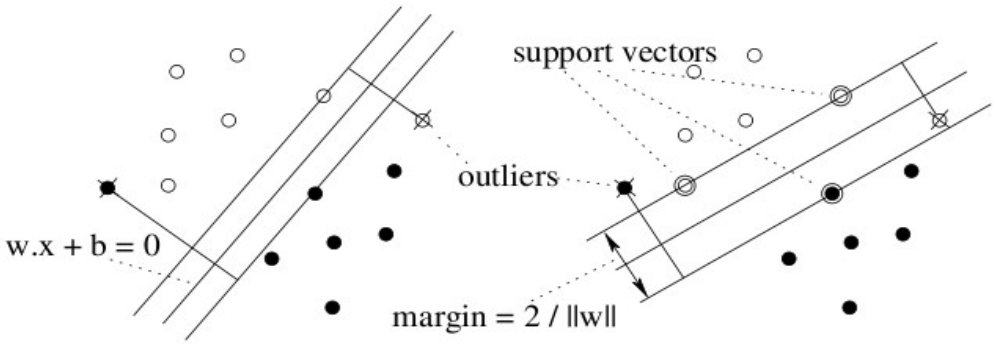


Passage d'un ensemble de données non linéairement séparables à un ensemble de données linéairement séparables

1 V. Vapnik, et S. Kotz, *Estimation of Dependences Based on Empirical Data*, Springer Series in Statistics, 1982, ISBN 978-0387907338.

d / SVM à marge souple (soft-margin SVM)

La technique de marge souple tolère les mauvais classements. Elle cherche un hyperplan séparateur qui minimise le nombre d'erreurs grâce à l'introduction de variables ressort ξ_k qui permettent de relâcher les contraintes sur les vecteurs d'apprentissage.



Exemple d'erreurs de classification

La formulation des SVM à marge souple s'exprime sous la forme suivante :

$$\text{Min} \quad \frac{1}{2} \|w\|^2 + C \sum \xi_i \qquad \text{sous les contraintes,}$$
$$y_i (w.xi + b) \geq 1 - \xi_i \qquad \forall\, i$$
$$\xi_i \geq 0$$

où C est une constante qui permet de contrôler le compromis entre nombre d'erreurs de classement, et la largeur de la marge.

2 / Application à notre programme

Pour apprendre de nouveaux gestes, il suffit de montrer une main dans la bonne position à la webcam, en la bougeant un peu pour que toutes les variantes de ce geste soit prises en compte. Nous avons choisi de garder une frame sur dix, de la transformer en vecteur, et de représenter un geste avec trois cent vecteurs (nous sommes passés à cent vecteurs plus tard) qui seront écrits dans un fichier texte (un fichier par geste).

Lors de l'exécution du programme, le classifieur doit être entraîné : il prend tous les fichiers textes créés par l'apprentissage, les labels attribués aux gestes (présent dans le nom du fichier), et utilise la méthode *train* d'OpenCV pour créer le fichier *model.txt.* Ce fichier n'a besoin d'être créé qu'une seul fois. Il est ensuite utilisé dans la fonction *predict* du SVM, qui prend aussi en paramètre le vecteur de l'image courante de la main. Cette fonction renvoie le label correspondant au geste effectué, qui sera ensuite utilisé pour envoyer un ordre au drone grâce à l'API.

2 / Communication avec le drone

A / Choix de l'API

En début de projet, afin d'interagir avec le drone et sur les conseils de M.RALAIVOLA, nous nous sommes tournés vers l'API officielle du drone « l'ARDroneLIB ».

À la suite de la consultation de la documentation, cela semblait un bon choix car nous pensions qu'elle serait facile d'utilisation c'est à dire qu'il serait aisé d'implémenter les fonctionnalités dont nous avions besoin et suffisamment complète pour la récupération du flux vidéo provenant du drone et l'envoi de commande au drone.

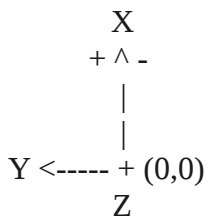
Dans les faits, il a été plutôt simple d'implémenter la récupération du flux vidéo et l'envoi de commande. En revanche, l'intégration à notre programme fut rendu difficile à cause d'une compilation peu intuitive. Le problème de cette API est qu'il nous était impossible de faire avancer, reculer, ou de bouger latéralement le drone bien que nous ayons suivi les recommandations de la documentation. En faisant des recherches, nous avons vu que de nombreuses personnes avaient le même problème.

À la suite de plusieurs essais infructueux, nous avons décidé d'utiliser une API non officielle CV Drone (= openCV + ar.Drone) consultable à cette adresse : https://github.com/puku0x/cvdrone. Cette API utilise directement le protocole de communication « AT Commands » défini dans le chapitre 6 de l'ARDrone Developer Guide SDK 2.0 consultable ici : http://www.msh-tools.com/ardrone/ARDrone_Developer_Guide.pdf pour interagir avec le drone. Grâce à celle-ci, nous avons pu exécuter tous les déplacements du drone tout en conservant les autres fonctionnalités que nous avions avec l'API officielle.

B / Comment cela fonctionne ?

1 / Le déplacement

Le drone se déplace selon les trois axes présentés sur le dessin :



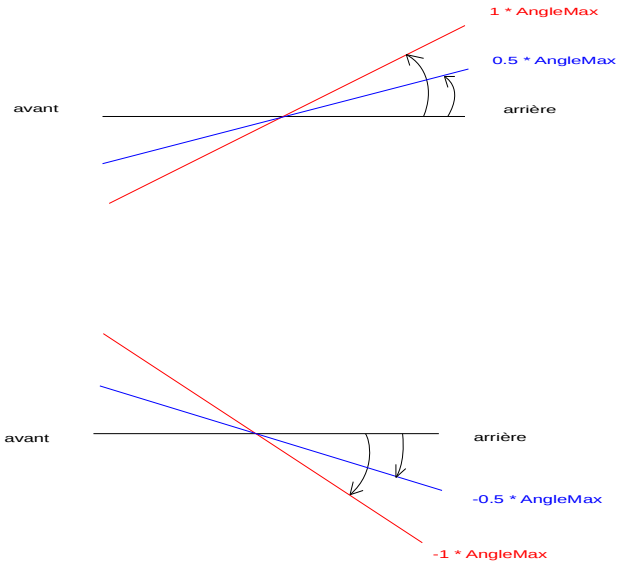
Légende

- L'axe X représente le déplacement avant(+)/arrière(-).
- L'axe Y représente le déplacement gauche(+)/droite(-).
- L'axe Z représente le déplacement haut(+)/bas(-).

On peut donc représenter une commande envoyée au drone par un quadruplet (x,y,z,r) où x,y,z,r appartiennent à $\mathbb{R}$ et [-1,1], où r représente la rotation gauche(+)/droite(-).

On remarque que la valeur absolue de chaque paramètre est comprise entre 0 et 1 mais elle a une signification différente selon le paramètre pour x et y, elle représente le pourcentage de l'angle d'inclinaison maximale par rapport à l'axe concerné. Pour z, elle définit le pourcentage de la vitesse verticale maximale. Et pour r, c'est le pourcentage de la vitesse angulaire maximale. En d'autres termes, plus la valeur est proche de 1, plus le drone se déplacera vite dans la direction choisie ou tournera vite lors de la rotation.

Exemple sur l'axe X :



2 / L'envoi de commande

Dans notre programme, chaque ordre déclenche une commande avec le paramètre adéquat affecté à -1, 0 ou 1, par exemple l'ordre « se déplacer à droite » déclenche (0,-1,0,0).

La documentation préconise de laisser un délai de 30 millisecondes entre chaque envoi de commande pour avoir un mouvement fluide, et d'envoyer une commande au moins toute les 2 secondes pour maintenir la connexion ouverte entre le drone et l'application. Afin de satisfaire cette dernière recommandation, nous possédons un geste « ne rien faire » qui envoie (0,0,0,0). En outre, il sert à arrêter l'exécution de la commande précédente car si une commande est envoyée au drone, celle-ci s'exécute tant qu'il n'y a pas de nouvelle commande envoyée.

Au début, notre programme envoyait la commande gestuelle au drone dès que le geste était reconnu. Cette approche a engendré le problème suivant : un geste de transition entre deux gestes pouvait être reconnu et provoquer l'envoi d'une commande non désirée. Par exemple, le geste de transition entre le geste « décoller » et le geste « avancer » était reconnu comme étant le geste « atterrir ».

Par conséquent, une commande associée à un geste est envoyée au drone si et seulement si ce geste est reconnu au moins trois fois dans les cinq derniers gestes analysés.

Conclusion

Pour conclure, nous pouvons dire que l’objectif principal est à peu près atteint : le drone répond au geste, et se déplace là où on le souhaite.

Cependant, ce pilotage ne marche vraiment bien qu'avec la caméra d'un ordinateur, et le sujet du TER précisait que cela devait être possible avec la caméra frontale du drone. Avec notre programme, nous pouvons choisir un mode pour piloter le drone de cette façon, mais les problèmes de bruits, de perspectives et de champs de la caméra rendent ce pilotage peu efficace.

D'un point de vue personnel, ce projet nous a beaucoup apporté : nous avons approfondi nos connaissances en apprentissage automatique, nous avons mis en pratique des notions de gestion de projet, et nous avons énormément appris sur le traitement des images. Faire des recherches sur un sujet relativement récent, avec assez peu de travaux, a également été un aspect intéressant du projet.

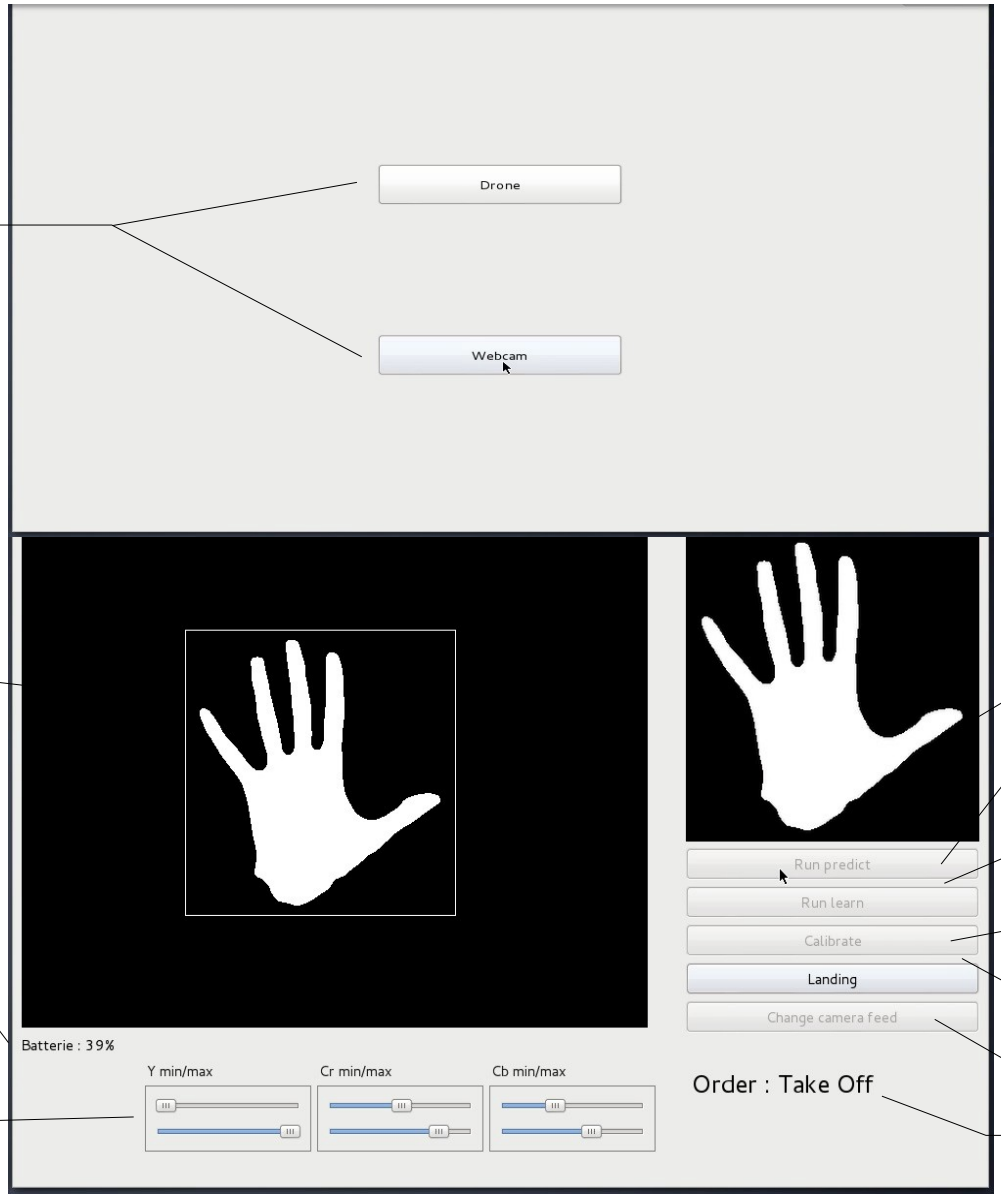
Nous avons conçu notre programme de tel sorte qu'il soit possible d'utiliser la reconnaissance de gestes (ou plus généralement, de formes) indépendamment du pilotage du drone. Ainsi, notre programme pourrait être utiliser pour d'autres objectifs. De même, il est possible de facilement rajouter une autre source d'images, comme une Kinect. Toutes ces possibilités pourraient faire l'objet de nouveaux TER. Il en va de même avec l'amélioration du pilotage via la caméra du drone.

Références

- http://images.math.cnrs.fr/Le-traitement-numerique-des-images.html
- http://www.commentcamarche.net/contents/1216-traitement-d-images
- https://docs.google.com/file/d/0B75V_Mz3x2GnN29kTWtNWW1Rc1U/edit?pli=1
- https://otik.uk.zcu.cz/bitstream/handle/11025/10641/Rios-Soria.pdf?sequence=1
- http://fr.wikipedia.org/wiki/Machine_%C3%A0_vecteurs_de_support
- http://docs.opencv.org/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html
- **Liva Ralaivola** : Introduction to kernel methods

Annexes

Annexe 1 : Interface graphique



Choix de la source du flux vidéo

Image provenant de la source du flux vidéo sélectionnée

Indication du niveau de batterie du drone

Curseurs permettant d'ajuster la plage des valeurs YCrCb

Image servant à la reconnaissance du geste

Mode pour piloter le drone avec la commande gestuelle

Lance l'apprentissage d'un nouveau geste

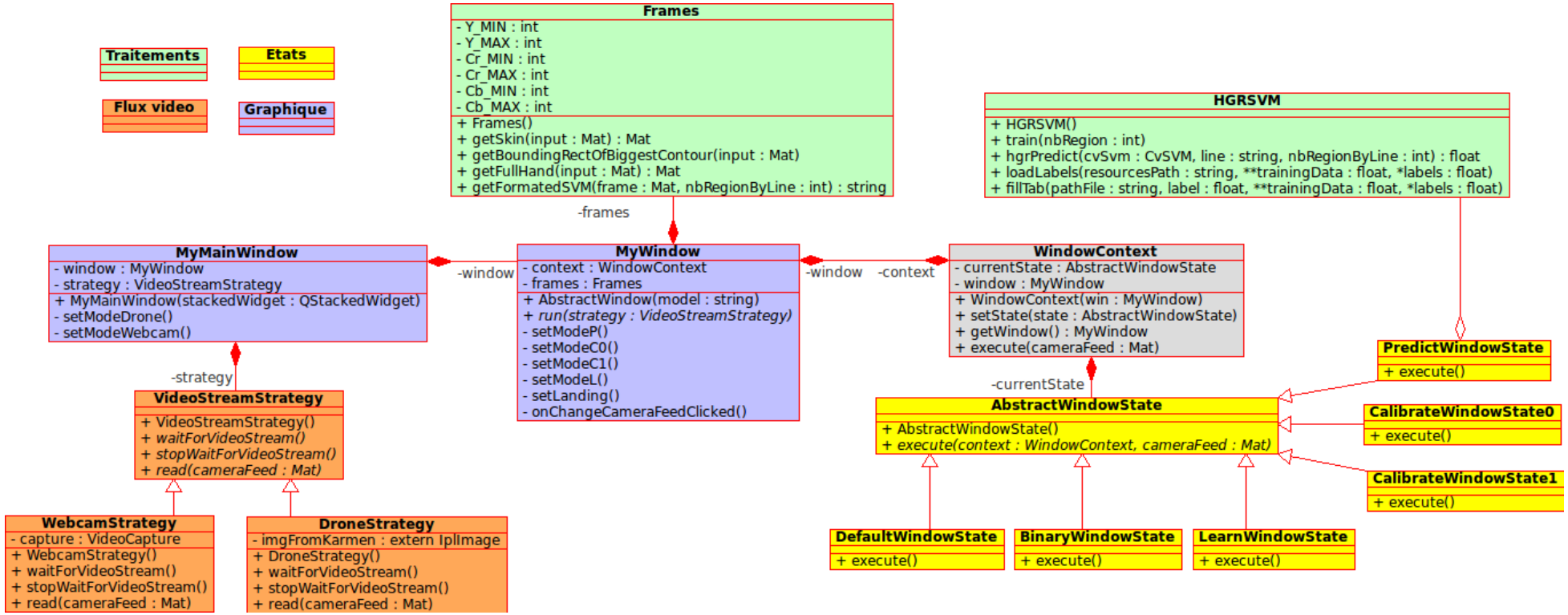
Mode calibrage

Envoi l'ordre d'atterrissage puis retour au mode par défaut

Retour à la fenêtre précédente

Indication de l'ordre prédit par le SVM

Annexe 2 : Diagramme de classe



Ce diagramme vise à montrer l'architecture générale de l'application et est donc volontairement incomplet au niveau de certains attributs et méthodes, dans un souci de lisibilité.

Afin d'identifier facilement l'utilité des différentes classes de l'application, nous avons choisi de les grouper par « famille » en utilisant différentes couleurs.

- Les classes bleues modélisent la partie graphique
- Les classes jaunes modélisent les différents états possibles de l'application
- Les classes oranges modélisent les sources de flux vidéo disponibles
- Les classes vertes modélisent la partie traitement

Les classes de la partie graphique

La classe ***MyMainWindow*** représente la fenêtre principale (démarrage) de l'application et présente, à travers des boutons, les différentes sources de flux vidéo disponibles.

La classe ***MyWindow*** correspond à la fenêtre présentant les informations suivantes :

- les images provenant du flux vidéo sélectionné
- les images recadrées servant à la reconnaissance
- des sliders permettant d'affiner les plages de valeurs YCbCr
- des boutons permettant de choisir le mode d'exécution (voir §**Les classes représentant les états de l'application**)

Les classes représentant les états de l'application

Afin de pouvoir faire évoluer l'application facilement en ajoutant de nouveaux états, nous avons mis en place le patron de conception State.

La classe ***AbstractWindowState*** définit l'abstraction des comportements du patron et la classe ***WindowContext*** s'occupe de l'interfaçage du patron avec le reste de l'application.

Les classes représentant les états sont :

- ***DefaultWindowState :*** mode par défaut, ne réalisant aucun traitement de l'image
- ***BinaryWindowState :*** mode de binarisation de l'image
- ***LearnWindowState :*** mode permettant l'apprentissage d'un nouveau geste depuis la source du flux vidéo sélectionnée
- ***CalibrateWindowState0 :*** mode permettant d'afficher les zones de calibrage sur l'image
- ***CalibrateWindowState1 :*** mode calculant la moyenne des valeurs YCbCr des zones de calibrage
- ***PredictWindowState :*** mode réalisant la reconnaissance des gestes de la main

Ces classes étendent la classe ***AbstractWindowState***.

Les classes représentant les sources de flux vidéo

Au cours de ce projet, nous avons principalement travaillé avec la webcam. Cependant, l'application permet également de réaliser les mêmes tâches avec la caméra du drone. Plus généralement, et afin de pouvoir ajouter facilement de nouvelles sources de flux vidéo, comme par exemple la Kinect, nous avons mis en place le patron de conception stratégie.

La classe ***VideoStreamStrategy*** est une classe abstraite définissant les trois méthodes utilisées par la classe ***MyWindow*** afin de récupérer, sous un format OpenCV unique (*Mat*), les images à traiter. Celles-ci sont ensuite données à la classe représentant l'état actuel de l'application via le contexte.

Les classes ***WebcamStrategy*** et ***DroneStrategy*** étendent ***VideoStreamStrategy,*** et permettent de récupérer les images depuis respectivement la webcam, via la classe ***VideoCapture*** d'OpenCV, et la caméra du drone, via la variable externe de type *IplImage* (pour plus d'informations concernant la communication entre le drone et l'application, voir **II/2/Communication avec le drone**).

Les classes de la partie traitement

La classe ***Frames*** s'occupe du traitement des images. Les méthodes importantes de cette classe sont :

- ***getSkin (input Mat) : Mat***

  Transforme l'image RGB en YCbCr, puis binarise cette dernière suivant la plage de valeurs YCbCr correspondant à la couleur de la peau. Des traitements additionnels (dilatation, érosion, floutage) sont effectués afin de diminuer le bruit.

- ***getFullHand (input Mat) : Mat***

  Détecte la zone ayant le plus grand contour. Cette zone est ensuite extraite de l'image originale de façon à avoir une image de 300x300 pixels.

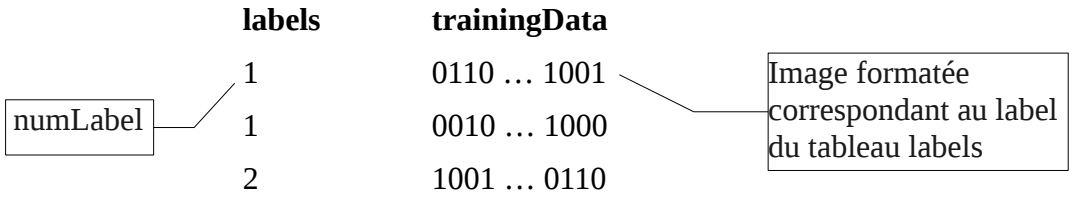
- ***getFormatedSVM (frame Mat, nbRegionByLine int) : string***

Retourne une chaîne de caractères suivant le format détaillé dans la partie **II/1/B/2/Formatage de la main**

La classe ***HGRSVM*** s'occupe de l'apprentissage et de la reconnaissance des gestes de la main. Les méthodes importantes de cette classe sont :

- ***loadLabels (ressourcesPath string, **trainingData float, *labels float) : void***

  Charge les fichiers labels se trouvant dans le répertoire dont le chemin est précisé par ***ressourcesPath***. Le nom des fichiers est de la forme *label_numLabel*. Les données permettant de nourrir le SVM sont « stockées » de la manière suivante :



Exemple avec deux labels et les images correspondantes formatées comme décrit dans la partie **II/1/B/2/Formatage de la main**

- ***train (nbRegion int) : void***

  Charge les labels, configure le SVM puis l'entraîne afin d'obtenir un modèle qui est sauvegardé dans un fichier.

- ***hgrPredict (cvSvm CvSVM, line string, nbRegionByLine int) : float***

Utilise le ***cvSvm*** de openCV afin de prédire la valeur du label correspondant à l'image décrite par la chaîne de caractères ***line***.

Annexe 3 : Ordres et gestes correspondants

Décoller

Atterrir

Ne rien faire

Monter

Descendre

Aller vers la gauche

Aller vers la droite

Avancer

Reculer