Projet de Synthèse
Initiation à l'IA et robotique

Réalisé par Siqi LIU, Yuchen TIAN et Pierre-Louis LEE
Encadré par M.Hervé FREZZA-BUET

19 juin 2014

Table des matières

Table des figures

1 Introduction

Avec une agilité sans précédent et une mobilité importante, les drones deviennent de plus en plus présents dans notre vie quotidienne. Certaines applications commerciales sont d'ailleurs déjà envisagées par les entreprises. C'est pour cela que les applications se basant sur la technologie des drones nous intéressent particulièrement.

Traditionnellement, les drones sont commandés par des manettes de jeux, au clavier, ou sur des smartphones. Ces méthodes de commandes présentent des avantages comme la fiabilité et la rapidité mais également certains inconvénients. Par exemple, un utilisateur doit toujours se placer dans la référence du drone, et commander l'orientation par rapport au drone, ce qui fait que la commande reste loin d'être intuitive.

L'objectif du projet devient donc clair, on aimerait, avec les outils les plus accessibles possibles, réaliser une commande intuitive d'un drone.

Pour ce faire, nous avons décidé de nous exercer à la programmation des drones à l'aide d'un puissant framework robotique : ROS [1]. L'idée était au départ de prendre en main ROS puis de faire quelques essais pour voir si nous arrivions à contrôler le drone. Il s'est avéré que très vite nous avons pu mettre au point un noeud ROS qui interagit à partir du clavier avec le package *ardrone_autonomy*, développé sous forme d'un noeud ROS.

L'architecture de ROS se basant sur des noeuds communicants nous permet facilement d'ajouter des noeuds de traitement indépendants les uns des autres. Avec ce framework très flexible, nous avons donc ajouté des modules de traitement d'image et le module d'intelligence artificielle et d'automatisme tout en gardant la possibilité de commande au clavier que nous avions précédemment implémenté.

Nous allons maintenant décrire plus précisément les modules que nous avons utilisé et leur utilité, et pour finir nous verrons les différents problèmes rencontrés lors de la conception ainsi que diverses améliorations possibles.

L'ensemble du code du projet est disponible sur github.com en open-source via le lien suivant : `https://github.com/liusiqi43/robia`.

2 Architecture du programme

Le système de contrôle gestuel du drone se décompose en plusieurs sous modules qui traitent chacun un problème spécifique. Au cours du projet, nous nous sommes servis des diverses fonc-

1. Robotic Operating System

tionnalités de ROS afin de gérer la communication entre les différents modules.

Dans la suite de cette section, nous vous présenterons l'architecture globale du système et ensuite l'organisation des différents noeuds dans le cadre de ROS.

2.1 Architecture globale

Dans un premier temps, on peut essayer de décomposer le programme en trois modules principaux, notamment le module de traitement d'image, d'intelligence artificielle ainsi que le module de contrôle du drone. L'architecture générale est détaillée sous format d'un graphe dans la Figure 1.

L'utilisateur interagit avec le drone via la caméra frontale. À l'aide du système de gestion de drone [2], on récupère l'image capturée via un topic dédié et ensuite, on va la passer au module de traitement d'image, qui réalise un filtrage des couleurs, et ensuite une réduction de bruit par le biais des opérateurs de morphologies mathématiques. Les résultats ainsi obtenus sont donc mis sous la forme de trois nuages des points, qui représentent, dans le cas idéal, la main gauche, la tête et la main droite de l'utilisateur.

Après avoir récupéré les trois nuages de points principaux, on soumet ces points au composant vq2 [3] qui permet de déterminer les composantes connexes dans le graphe et réduire le bruit en tenant compte des propriétés topologiques. Ayant obtenu les composantes connexes, nous veillons à ne prendre en compte que les trois composantes connexes principales et ensuite, à calculer leurs barycentres respectifs pour pouvoir déterminer les positions des mains et de la tête de l'utilisateur.

Finalement, les positions des mains et de la tête sont transmises au module contrôleur qui prend des décisions en fonction des positions, en respectant certaines consignes pré-définies. Le contrôleur envoie enfin la commande vers le drone qui gère ensuite les mouvements.

2.2 Organisation des noeuds ROS

a. **ardrone_driver** : grâce au projet open-source ardrone_autonomy on peut interfacer le drone et les autres modules. Ce noeud publie sur les différents topics qui nous permettent de connaître l'état actuel du drone. Ce noeud souscrit en même temps aux divers topics qui nous permettent de gérer et commander le mouvement du drone.

b. **keyboard_cmd** : ce noeud nous permet de commander le drône depuis le clavier. Il est capable de commander le drone pour qu'il atterrisse, décolle, ou effectue tout autre

2. ardrone_autonomy
3. quantification vectorielle

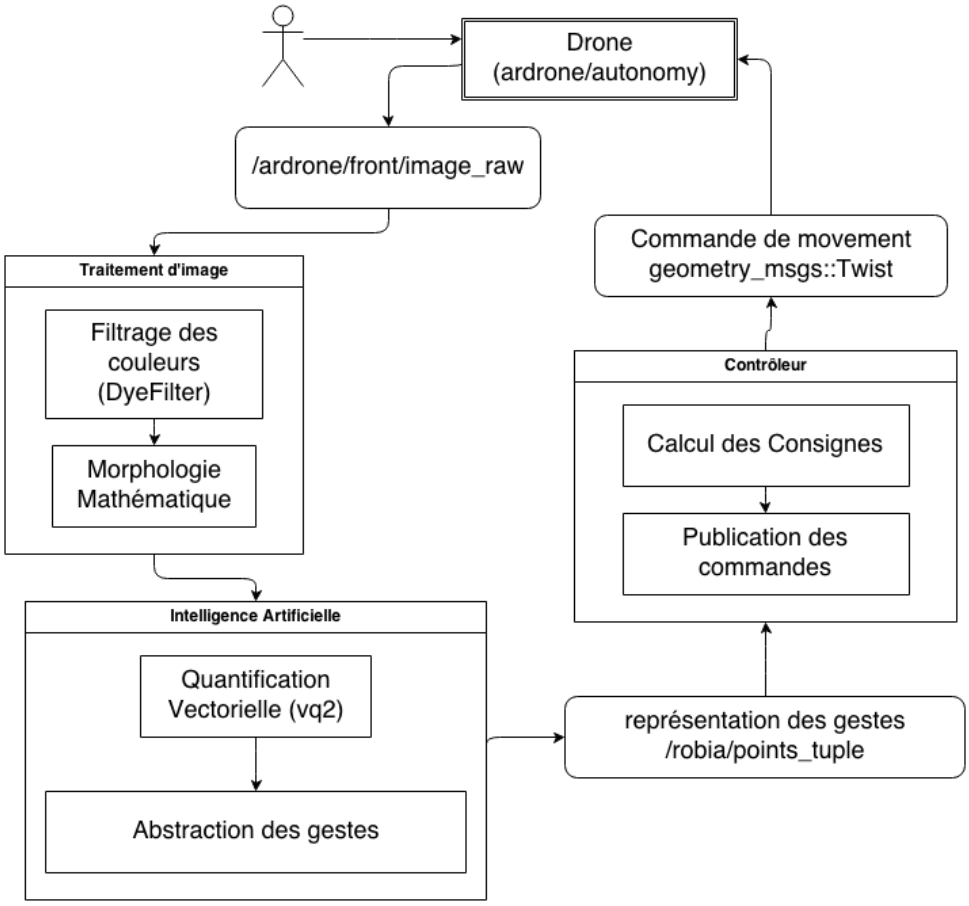

FIGURE 1 – Architecture générale du programme

commande disponible. Il est possible de demander un "reset" du drone également. Ce noeud est prévu pour commander le drone en cas d'urgence ou d'anomalies. Ce noeud publie donc sur les topics de contrôle du drone.

c. **AI** : ce noeud est le noeud central de la chaîne de traitement. À partir de l'ensemble des nuages des points, il est capable de structurer les points et générer ainsi les composantes connexes qui sont présentes dans l'image. Il est également chargé de déterminer les positions des mains et de la tête de l'utilisateur. Ce noeud publie donc sur le topic /robia/points_tuple qui contient l'ensemble de l'information obtenue à la fin de la chaîne de traitement.

d. **Control** : ce noeud sert à commander le drone en mouvement, connaissant la position des mains et de la tête de l'utilisateur. Il est chargé d'identifier les différents gestes et transmettre les ordres correspondants (geometry_msgs : :Twist) au drone.

e. **image_view** : ce noeud permet de visualiser en temps réel l'image envoyée par la caméra du drone.

3 Implémentation des modules

3.1 ardrone_autonomy

Ce package est un projet ROS à part entière développé grâce à la contribution de nombreuses personnes. Nous avons intégré ce package open-source à notre projet car il nous permet de faire l'interface entre les messages envoyés et le contrôle des moteurs du drone, il nous fournit également les informations sur les différents capteurs du drone comme la caméra que nous utiliserons plus tard, l'état de la batterie, le magnétomètre, l'altimètre, les capteurs de positions angulaires etc...

3.1.1 Les données disponibles

Comme énoncé précédemment, il existe de nombreux capteurs sur le drone et l'on peut recevoir si on le souhaite de nombreuses informations en souscrivant aux topics suivants :

— ardrone/navdata
— ardrone/mag
— ardrone/front/image_raw
— ardrone/image_raw
— ardrone/bottom/image_raw

Sur le topic ardrone/navdata sont en fait publiés des messages du type *ardrone_ autonomy : :navdata*, qui contiennent les nombreuses informations énoncées plus tôt comme les accélérations angulaires ou linéaires, ou encore l'état de la batterie. Le topic ardrone/mag permet quant à lui de récupérer les informations en provenance du magnétomètre. Comme plus tard il nous faudra récupérer l'image provenant de la caméra frontale, on souscrira au topic ardrone/front/image_raw.

3.1.2 Les commandes disponibles

Maintenant si l'on souhaite donner des ordres il va falloir publier différents messages sur les topics contrôlant les moteurs, ce qui veut dire que nous avons maintenant besoin d'objets de type publisher pour envoyer les messages aux différents topics suivants :

- ardrone/takeoff
- ardrone/land
- ardrone/reset

Et une fois le drone en vol on publie des messages de type *geometry_ msgs : :Twist* sur le topic *cmd_ vel*. Ces messages peuvent être de deux formes :

- linear (comporte les coordonnées x, y et z)
- angular (comporte juste une coordonnée z)

Pour utiliser ceci on va par exemple remplir le message avec linear.x, linear.y et linear.z en fonction des décisions [4]. Enfin il est nécessaire de signaler que lorsque l'on envoie une valeur positive l'appareil se déplace dans un sens et lorsque l'on envoie une valeur négative il se déplace dans l'autre sens.

3.2 Traitement d'Image

Le module de traitement d'image consiste à filtrer les couleurs et ensuite, enlever le maximum de bruit possible, ce qui facilite le traitement à réaliser dans les modules Intelligence Artificielle et Contrôle.

4. choix de l'utilisateur pour la commande au clavier et système décisionnel par comparaison pour la partie où seront implémentées les commandes automatiques

3.2.1 Filtrage des couleurs (DyeFilter)

Le filtrage des couleurs se fait avec le filtrage proposé par Dye, qui a été réalisé en se basant sur la bibliothèque de traitement d'images Mirage. Or, une implémentation plus récente et plus puissante proposée aujourd'hui est celle d'OpenCV. Nous procéderons donc à essentiellement porter l'implémentation de Dye vers OpenCV, en utilisant les représentations d'image adaptées au format OpenCV. L'algorithmique reste donc la même.

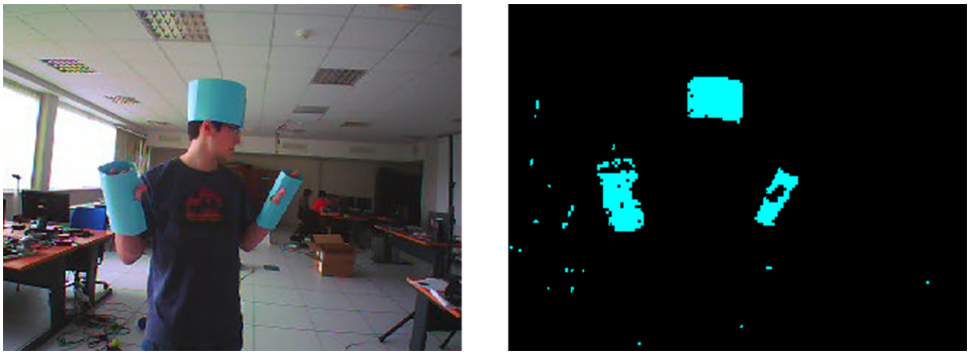

FIGURE 2 – Effet du filtrage des couleurs

Le filtrage par Dye, contrairement aux autres filtrages, consiste à normaliser les composantes (R, G, B) de chaque pixel de sorte que le minimum des trois composantes est soustrait de chacune des trois composantes. C'est à dire :

$$(R, G, B) = (R, G, B) - min(R, G, B); \tag{1}$$

En plus, nous adoptons une représentation indicielle des couleurs qui subdivise l'ensemble des couleurs possibles en 6 partitions différentes. Chaque partition est donc représenté par une valeur index comprise entre 0,0 et 6,0. La transformation d'une représentation RGB vers la représentation par index est donnée en détail dans le fichier dye.h. L'algorithme de conversion a été choisi lors de l'implémentation de dye et ne fait pas l'objet d'une discussion dans ce projet.

Un exemple de traitement est donné dans la Figure 2.

3.2.2 Morphologie mathématique

En appliquant simplement le filtrage des couleurs, on se rend compte qu'il existe toujours du bruit résiduel, pouvant perturber le calcul des composantes connexes au niveau de la quantification vectorielle.

En effet, le filtrage des couleurs n'est jamais parfait, c'est à dire que si dans le fond de l'image, des couleurs similaires à celle de la couleur cible sont présentes, on risque d'avoir des composantes connexes aberrantes, ce qui est indésirable pour le traitement dans le suite.

Une manière de réduire le niveau du bruit, c'est à dire d'enlever les points aberrants, est de se servir de morphologie mathématique.

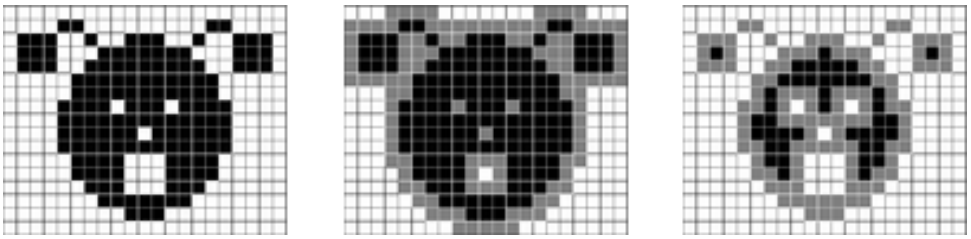

FIGURE 3 – L'effet de l'érosion (droite) et de la dilatation (centre) sur l'image originale (centre)

La transformation se déroule en deux étapes : dans un premier temps, on fait une érosion et ensuite, une dilatation. Ces deux opérations sont les opérateurs de base dans les morphologies mathématiques. L'érosion sur l'image originale enlève les points qui n'ont que très peu de voisins, ce qui nous permet de filtrer les points isolés. Or, en générale les points aberrants que l'on obtient après le premier filtrage correspondent bien à ce critère. Dans notre cas, on choisit comme élément structurant, qui définit ce voisinage, une ellipse dont la taille est de (5,5).

La formulation mathématique est donnée comme suit :

Soit X un sous-ensemble de E. L'élément structurant B est définie comme la somme de Minkowski [5] :

$$\epsilon_B(X) = X \ominus B = \{x \mid B_x \subset X\} \tag{2}$$

Les opérations d'érosion et de dilatation étant généralement non-inversibles, elles ont pourtant des effets qui d'une certaine manière vont se compenser. C'est à dire que l'on va dilater les points colorés de sorte à remplir leurs voisinages, qui encore une fois, est définit par l'élément structurant.

La formulation mathématique est donné comme suit :

Soit X un sous-ensemble de E. La dilatation morphologique avec l'élément structurant B est définie comme la somme de Minkowski

$$\delta_B(X) = X \oplus B = \{x + b \mid b \in B, x \in X\} = \cup_{x \in X} B_x \tag{3}$$

En appliquant successivement ces deux opérations, on obtient un résultat très robuste et satisfaisant (Figure 4).

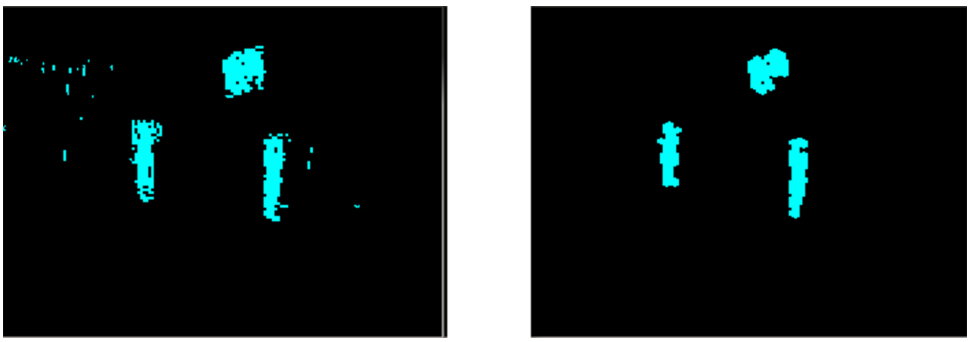

FIGURE 4 – Exemple après traitement avec l'érosion et la dilatation

3.3 Intelligence Artificielle

Partie centrale du programme, le module d'intelligence artificielle consiste à déterminer, à l'aide de la quantification vectorielle et du traitement des données, les positions des barycentres des nuages de points significatifs. Dans notre cas, ceci revient à dire qu'on doit être en mesure d'extraire les positions des mains et de la tête de utilisateur, à partir d'une image après filtrage des couleurs. Cette dernière étant une image binaire avec la couleur cible, et noire partout ailleurs.

Ce module doit également gérer le cas où il est impossible de déterminer les positions des barycentres en partant de l'image d'entrée et commander le drone en conséquence.

3.3.1 VQ2

Partant d'une image binaire simple, on peut, à l'aide des techniques de traitement d'image classiques (Figure 5), identifier les composantes principales assez facilement [6]. En revanche, ces méthode de traitement d'image présentent certains inconvénients.

6. certains algorithmes de traitement d'image que sont la détection des contours et le calcul de gradient permettent de trouver l'endroit où les différences entre les pixels sont les plus marquées, ce qui définit le contour

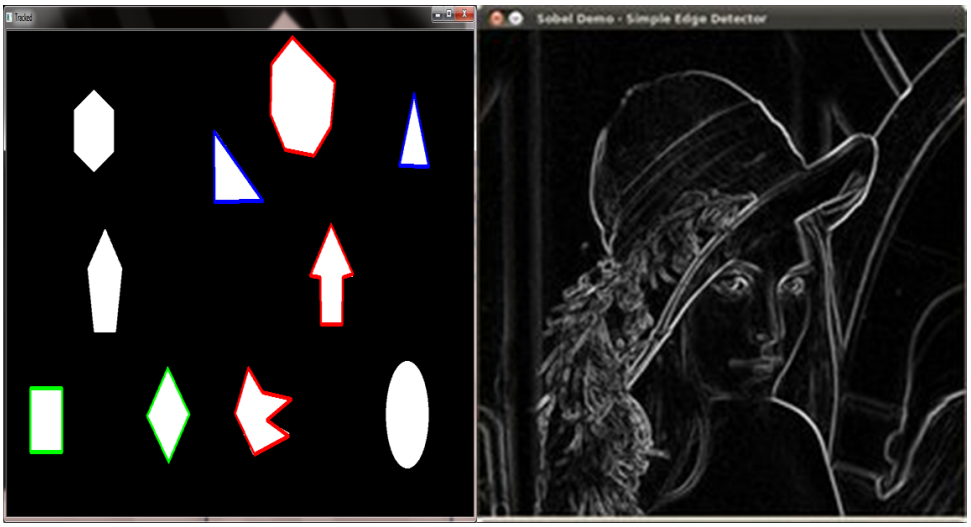

FIGURE 5 – Traitement des nuages de points par la recherche des contours (gauche) et par le calcul des gradients (droite)

a. **calcul coûteux** : simple en théorie, mais les méthodes de traitement d'image comme la recherche des contours et le calcul de gradient s'avèrent assez coûteuses en temps de calcul. En effet, la manipulation se fait sur toute l'image. Dit autrement, chaque opération de base nécessite au minimum de parcourir tous les pixels, ce qui alourdit le calcul.

b. **robustesse faible** : dans le domaine du traitement d'image ou plus généralement en intelligence artificielle, on ne peut jamais être assuré de la qualité de l'image en entrée. C'est à dire que la robustesse reste une priorité pour nous. Or, l'image en entrée, même avec l'application des opérateurs de morphologies mathématiques, présente souvent des points aberrants voire du bruit, néfaste pour le module d'intelligence artificielle. Les algorithmes classiques de traitement d'image sont par définition sans mémoire (le traitement de l'image au temps t est indépendant de celui au temps t-1), ce qui limite gravement leur robustesse face aux changements brutaux d'une image à l'autre.

c. **abstraction faible** : le fait qu'on manipule directement les pixels de l'image nous limite dans l'abstraction des nuages des points. En effet, si on fait tourner le nuage de points sur l'image, les algorithmes de traitement d'image ne peuvent pas identifier le nuage de points initial et celui légèrement tourné comme étant le même objet. Ces informations de continuité sont donc perdues. Dans la suite de ce projet, on pourrait éventuellement envisager de définir plus de geste avec ces informations comme notamment la rotation des mains. En plus, une abstraction plus haut niveau nous permettrait de séparer les composantes connexes significatives même si les nuages de points se superpose. Ceci est impossible à envisager dans le cas des algorithmes de traitement d'images classiques. Une comparaison à titre d'exemple est donnée Figure 6 et Figure 7.

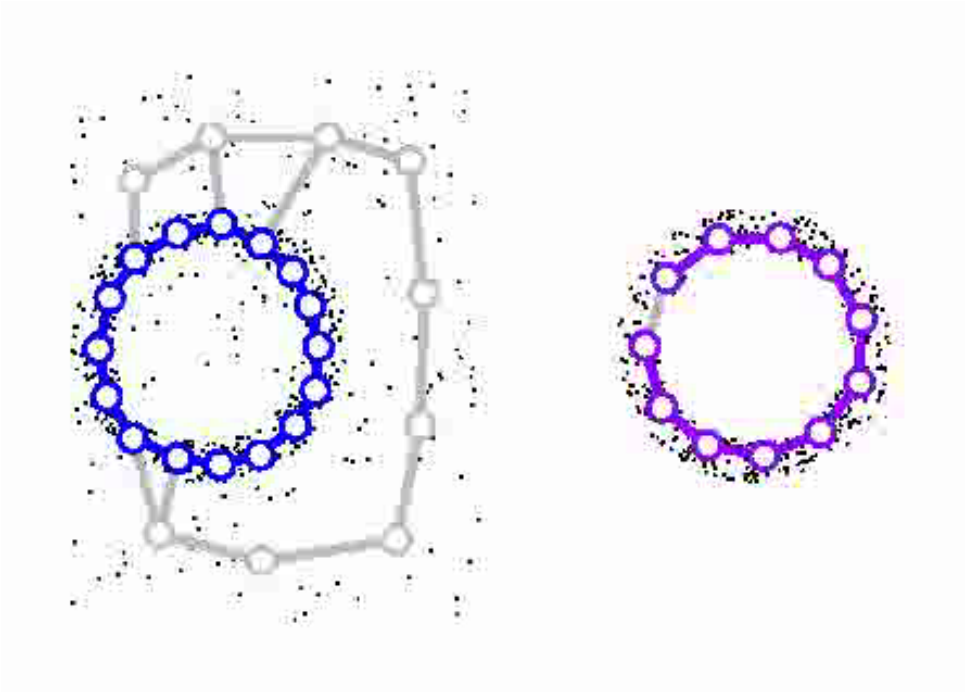

FIGURE 6 – VQ2 sur un nuage de points, composantes connexes séparées avec les nuages de points superposés

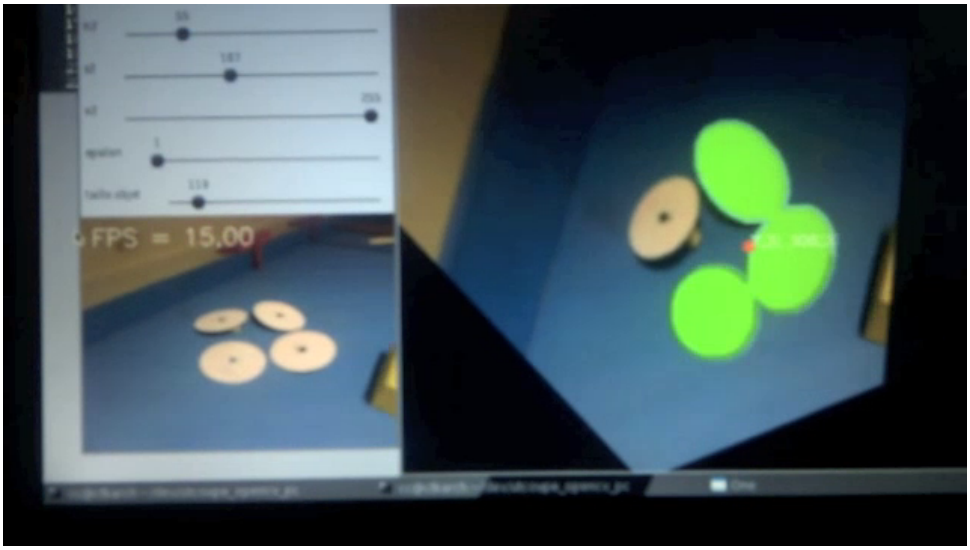

FIGURE 7 – recherche de contours avec les nuages de points superposés

La quantification vectorielle par "structuration de nuages de points dynamiques via des réseaux de neurones auto-organisants incrémentaux" [1] répond donc à nos besoins. On fait appel ici à la bibliothèque vq2, qui est une implémentation de cette théorie développé au sein du laboratoire MALIS à Supélec. Tout d'abord, vq2 récupère les coordonnées des points qui correspondent aux points avec la couleur cible. Une fois les points récupérés, vq2 va tenter de construire une composante connexe associée à chaque nuage de points, et faire évoluer ces composantes par les techniques d'apprentissage machine afin de les stabiliser.

On va commencer par une présentation des différents paramètres et terminologies centrales de vq2. Les détailles scientifiques et techniques ne font pas l'objet de ce projet et sortent de notre cadre de connaissances actuelles.

— **Epoch** : en effet, pour chaque frame capturée du flux vidéo, l'algorithme GNGT va essayer d'améliorer les composantes connexes calculées à chaque itération. La notion d'Epoch représente donc un certains nombres d'itération dont l'ensemble nous donne un résultat plutôt stable. Il est à noter que si toutes les itérations se font sur la même image, les points soumis à GNGT ne sont pas les mêmes. Ceci étant du au fait que les points sont soumis de manière aléatoire. Soit le nombre de points désiré N_d, le nombre de points dans le vecteur des points correspondant à la couleur cible N_c. Le nombre des points N soumis est donc donné par :

$$N = min(N_d, N_c) \tag{4}$$

Par des considérations empiriques, nous avons décidé de prendre $N_d = 2000$.

— **Nombres d'échantillons** : la notion de nombre d'échantillons N_e, centrale au calcul de GNGT, donne l'information sur le nombre de points qui seront soumis à GNGT. N_e est donc le nombre de points proportionnels au nombre de points soumis par rapport au nombre total de points ayant la couleur cible. Soit l'image initiale de taille (l, h) où l et h représentent le nombre de pixels en largeur et en hauteur, on donne ainsi la formulation suivante :

$$N_e = (l \times h) \times \frac{N}{N_c} \tag{5}$$

— **Target** : une fois le nombre d'échantillons correctement calculé, la notion de Target nous donne une indication sur la densité des unités de composantes connexes. Ainsi, on définit le paramètre Target de sorte que les composantes connexes puissent être adaptées autant pour les grands nuages de points que pour les plus petits.

— **learningRate** : couramment utilisé en apprentissage machine, le taux d'apprentissage représente le pas d'ajustement des paramètres appris par l'algorithme. Un pas d'apprentissage plus élevé se traduit par une vitesse de convergence plus grande, alors qu'un taux faible conduit à une convergence plus lente. En revanche, plus ce taux d'apprentissage est élevé, plus on risque de diverger lors de l'apprentissage. Ici, on choisit pour des raisons empiriques, un taux d'apprentissage de 0.01 pour avoir un compromis raisonnable entre

FIGURE 8 – VQ2 appliqué dans le cas du contrôle gestuel de drones

la vitesse de convergence et le risque de divergence.

— **lambda** : également souvent utilisé en apprentissage machine, le paramètre lambda représente le coefficient de la régularisation des paramètres. La régularisation sert, dans l'apprentissage machine à éviter les effets dits "over-fitting". En fait, les phénomènes d' "over-fitting" représentent le cas où le modèle obtenu se rapproche beaucoup trop des données d'entraînement, ce qui fait que le modèle ne s'adapte pas forcément pour les données qui ne sont pas dans l'ensemble des données d'entraînement. L'introduction de la régularisation nous permet de limiter l'adaptation du modèle, ce qui nous donne une meilleure performance sur les données en dehors des échantillons d'entraînement.

Sans exposer la théorie scientifique derrière la bibliothèque VQ2, on s'en sert pour construire les composantes connexes calculées à partir de chaque image capturée du flux vidéo. Un exemple d'application de VQ2 est donnée Figure 8. Les sommets et les arrêtes des composantes sont ainsi superposés sur l'image. Nous procédons ensuite au traitement des composantes pour obtenir les positions des mains et de la tête.

3.3.2 Traitement des données

Après que nous ayons successivement reçu les données contenant toutes les composantes, même avec du bruit, nous devons les traiter afin d'obtenir les trois plus grandes composantes. Ensuite,

nous avons créé une paire de <Déviation, Point> et défini la façon de comparer les points, qui consiste en fait à les comparer avec leurs déviations.

Soit D la déviation cumulée et x_{bc}^{i}, y_{bc}^{i} la position du barycentre tenant compte des i premiers points traités.

$$D = \sqrt{(x_{bc}^{i-1} - x_i)^2 + (y_{bc}^{i-1} - y_i)^2} \tag{6}$$

où x_{bc}^{i-1} et y_{bc}^{i-1} sont calculés comme suit :

$$x_{bc}^{i-1} = \frac{\sum_{k=1}^{i-1} x_k}{i-1} \tag{7}$$

$$y_{bc}^{i-1} = \frac{\sum_{k=1}^{i-1} y_k}{i-1} \tag{8}$$

Utilisation des déviations : nous choisissons les trois composantes avec les déviations les plus grandes possibles.

Ayant récupérer les trois plus grandes composantes, nous allons pouvoir créer des commandes en utilisant leurs trois barycentres. Nous trions les trois barycentres par abscisse et les labellisons par R (Droite), H (Tête) et L (Gauche). Nous avons également créé un message ROS points_tuple qui contient les six valeurs des coordonnées de RHL.

Si le drone ne voit pas les trois composantes, par exemple (Figure 9) si il peut seulement voit R et L quand il décolle, on va envoyer les messages -1 (cas spécial). Sinon, on va publier un message contenant les coordonnées des trois barycentres des composantes.

Nous allons ensuite diviser les coordonnées des trois points par la longueur et la largeur de l'image. Car cela rend le module Contrôle indépendant de la résolution de la caméra utilisée (dans le cas où l'on changerait de caméra il suffirait de diviser les coordonnées des points par les nouvelles valeurs, sans toucher au code du noeud Contrôle). Ceci permet également à la partie contrôle de traiter plus facilement les positions, par exemple de mettre H (Tête) dans le milieu de l'image (ici c'est 0.5).

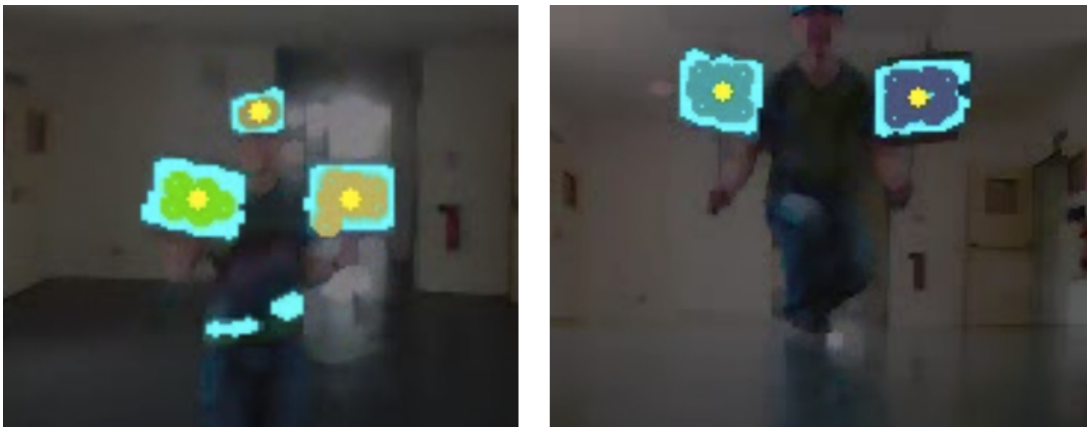

FIGURE 9 – Image avec plus de trois composantes (gauche) et moins de trois composantes (droite)

3.4 Contrôle

Ce noeud est à la fois de type subscriber et publisher, c'est à dire qu'il reçoit des messages contenant les coordonnées des 3 points (en souscrivant au topic : */robia/output_positions*). Il extrait ensuite les coordonnées pour calculer les distances entre ces points ainsi que leur barycentre (attention ces coordonnées sont normalisées entre 0 et 1), et pour finir publie des commandes (sur le topic : *cmd_vel*) via le système dédié.

3.4.1 Consignes

Le calcul se déroule ensuite à l'aide de trois fonctions, résultant sur trois consignes :

a. **centerWithRespectToBarycenter**
   Soit x_{bc} l'abscisse du barycentre des trois points R-H-L, y_{bc} son ordonnée et ϵ les seuils à partir desquels le drone essaye de se recaler. Ces incertitudes sont au nombre de 5, et définies comme suit :

   a) ϵ_r pour les rotations
   b) ϵ_h pour la hauteur
   c) ϵ_t lorsqu'il s'agit de se translater
   d) ϵ_d pour contrôler la profondeur
   e) $\epsilon_{activDep}$ pour que le drone n'essaye pas de se rapprocher si l'utilisateur se tourne (en effet une rotation de l'utilisateur a pour conséquence de modifier les distances d1 et d2)

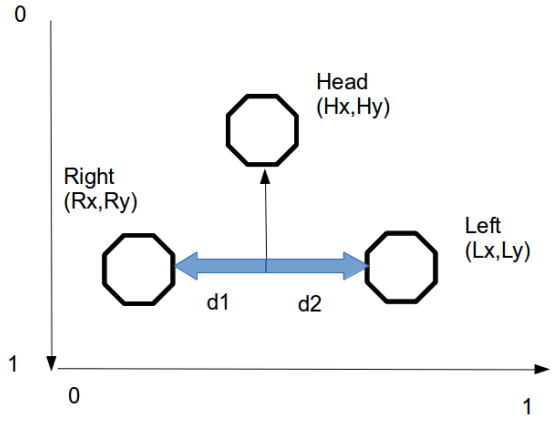

FIGURE 10 – Figure utilisée pour établir l'asservissement visuel

$$x_{bc} - 0.5 > \epsilon_r \tag{9}$$

$$x_{bc} - 0.5 < -\epsilon_r \tag{10}$$

$$y_{bc} - 0.5 > \epsilon_h \tag{11}$$

$$y_{bc} - 0.5 < -\epsilon_h \tag{12}$$

Cette première fonction ordonne au drone de positionner le barycentre des trois points au milieu de l'image (Si Équation 9 vraie : rotation à gauche, Si Équation 10 vraie : rotation à droite, Si Équation 11 vraie : on descend, Si Équation 12 vraie : on monte).

b. **checkForEqualDistanceBetweenLeftAndRight**
Soit $d1$ et $d2$ les distances définies sur la Figure 10

$$d1/(d1 + d2) - 0.5 > \epsilon_t \tag{13}$$

$$d1/(d1 + d2) - 0.5 < -\epsilon_t \tag{14}$$

Ici on vérifie l'alignement horizontal du drone avec l'utilisateur (Si Équation 13 vraie : translation à gauche, Si Équation 14 vraie : translation à droite).

c. **controlOfDepth**

On ajoute ici une autre constante : depthPosition, qui va permettre de définir une distance standard à laquelle l'utilisateur doit se situer par rapport au drone. x_H représente l'abscisse de la tête, x_L l'abscisse de la main gauche et x_R celle de la main droite.

$$checkfor|x_H - (x_L + x_R)/2| < \epsilon_{activDep} \tag{15}$$

$$(d1 + d2) > depthPosition + \epsilon_d \tag{16}$$

$$(d1 + d2) < depthPosition - \epsilon_d \tag{17}$$

Et enfin cette dernière fonction est utilisée pour contrôler l'éloignement du drone par rapport à l'utilisateur (Si Équation 16 vraie : on recule, Si Équation 17 vraie : on avance)

Ces fonctions vont donc calculer les actions que le drone doit effectuer, puis les inscrire dans une liste et cette liste sera publiée d'un seul coup à l'aide d'une fonction commit.

Les commandes sont à nouveau réalisées par la publication de messages sur le topic cmd_vel.

Pour finir, les distances sont à chaque fois calculées à epsilon près ce qui permet de modifier la précision de positionnement du drone, en effet si epsilon est trop petit il va osciller indéfiniment puisqu'il n'arrivera pas à se caler sur la bonne position, et si epsilon est trop grand il risque d'y avoir une erreur de positionnement importante.

3.4.2 Mécanisme de publication de message

La publication de message, au départ, se faisait immédiatement après chaque décision prise par le module décisionnel. En effet, si à une itération donnée le module Contrôle demande au drone d'avancer et de reculer en même temps, le drone va recevoir successivement deux messages lui demandant d'exécuter les deux ordres séparemment.

Même si mécaniquement le drone réagit assez rapidement aux ordres reçus, ce mécanisme par défaut présente des inconvénients. Premièrement, le drone ne peut pas réagir simultanément aux deux ordres. Ceci se traduit physiquement par le fait que seule la dernière commande est exécuté par le drone. Deuxièmement, le nombre de commandes publiées sur le topic est beaucoup trop élevé, ce qui peut conduire à des réactions inattendues du drone.

Afin de résoudre ce problème, nous avons conçu un autre mécanisme de publication, qui consiste à agréger toutes les commandes obtenues après une itération, dans un seul message pour seulement après l'envoyer. Lors d'une même itération, nous remplissons chaque champ au fur et à mesure,

et publions à la fin un seul message rempli. Ceci permet au drone de réagir à plusieurs ordres simultanément si toutefois ceci est mécaniquement possible. Et il faut de plus limiter le nombre de message publiés à chaque itération.

3.5 Contrôle Clavier

Le module de contrôle au clavier a été développé au début, afin de vérifier que toutes les actions pouvaient être effectuées. Tout d'abord, nous avons ajouté des opérations comme décoller, tourner à gauche, etc. Puis nous les avons chacune testé pour pouvoir détecter d'éventuelles erreurs.

Au début et à la fin de nos tests, nous devons utiliser les contrôles décoller et atterrir, cela ne se fait pas par commande gestuelle, et cela nous permet également d'arrêter le drone en cas d'urgence. Nous nous sommes également servi du contrôle clavier pour déterminer si le drone pouvait exécuter deux actions en même temps en appuyant simultanément sur deux touches, cependant le résultat n'a pas été très probant.

4 Problèmes rencontrés et améliorations possibles

4.1 utilisation des batteries

La première chose à signaler est l'épuisement rapide des batteries, chose qui devient problématique lorsque l'on souhaite utiliser longtemps le drone et faire plusieurs essais. Nous n'avons malheureusement pas de solution à apporter à ce problème en attendant que les constructeurs proposent de nouveaux modèles de batteries qui soient plus performantes.

4.2 Limites mécaniques du drone (réception de commandes contradictoires)

Ensuite nous avons rencontré un problème sur l'envoie des messages, en effet lorsque l'on envoie trop vite plusieurs messages du type avancer-rester en position ou encore avancer-tourner à droite, le drone n'exécute aucune des actions hormis rester en position.

4.3 Automatisme (Oscillation au niveau des consignes-décalage entre la réalité et l'image)

Nous avons également observé des oscillations sur la position du drone, même lorsque les cibles sont faces à lui et qu'il les détecte effectivement toutes il n'arrive pas à se stabiliser sur le barycentre des trois et il se déplace sur plusieurs positions proches de celle du barycentre. Cela provient sans doute du décalage entre la réalité et l'image reçue. Pour corriger cela il faudrait mettre au point une commande prévisionnelle, de façon à prévoir la prochaine position et savoir où il va arriver pour pouvoir corriger les paramètres de vitesse et de position.

4.4 Algorithme de recherche

Une autre suggestion intéressante serait l'utilisation d'un algorithme de recherche permettant de trouver une cible lorsqu'il ne voit rien. Actuellement, lorsque le drone ne voit pas l'une des trois cibles il se met en attente jusqu'à ce qu'il voit à nouveau les trois. On pourrait donc mettre au point une commande qui dit au drone de tourner sur lui même jusqu'à trouver une cible, voir le faire se déplacer mais dans une limite raisonnable pour ne pas risquer de heurter quelque chose.

4.5 Commande en vitesse plus poussée

Pour parfaire l'asservissement visuel, on pourrait essayer d'introduire une fonction de commande en vitesse faisant en sorte que le drone aille plus vite lorsqu'il est loin de l'utilisateur, et moins vite lorsqu'il est près.

4.6 Détection fiable de l'image

Nous arrivons maintenant au problème le plus gênant, la détection des couleurs. En effet selon les conditions de luminosité, l'environnement (intérieur d'une salle, extérieur, couloir...), de nombreux phénomènes peuvent survenir. En effet il arrive parfois que le drone perde l'une des cibles car l'alignement n'est plus bon ou que la luminosité est insuffisante ou encore parce qu'il est trop loin des cibles. Il peut également arriver lorsque les cibles sont trop proches que leurs graphes fusionnent, ce qui une fois de plus fait que le drone perd ses repères et s'immobilise. L'utilisation de caméra plus précises et notamment full HD (à comparer à celle du drone en 320×240) permettrait de grandement améliorer la détection des couleurs. On pourrait également utiliser plusieurs caméras, voir pourquoi pas une caméra thermique qui permettrait de détecter une plus grande partie du spectre.

4.7 Stabilisation du drone (idle count)

Au cours de nos expérimentations, nous avons découvert que le drone exécutait toujours la dernière action retenue. Autrement dit, si le drone est en déplacement et s'il ne reçoit pas d'autres instructions, il va continuer à se déplacer sans s'arrêter. Ceci nous oblige donc à envoyer l'ordre de stabilisation de temps en temps pour que le drone puisse se stabiliser lorsqu'il ne reçoit pas d'instruction.

Dans notre cas, nous avons prévu une variable de comptage de la durée inactive. Un ordre de stabilisation est déclenché lors qu'un certain seuil d'inactivité est atteint. Cette variable est réinitialisé lors que le drone reçois une commande qui le met en état stabilisé.

En revanche, cette solution présente certains inconvénients. Dans un premier temps, cette solution fait que le drone peut éventuellement recevoir des ordres de stabilisations en même temps que les autres ordres actifs (au moment où le traitement d'image présente un décalage, lorsqu'il existe du bruit sur l'image à traiter). Dans ce cas de figure, le drone ne réagit donc pas comme prévu, ceci se traduit finalement par une petite oscillation au niveau du drone.

4.8 Compléter les gestes disponibles

Pour ce premier prototype de système de commandes gestuelles de drone, nous avons exploité certaines propriétés des composantes connexes obtenues à partir de l'image de la caméra. En revanche, les gestes que l'on a définis restent des gestes de base et ne se servent que de caractéristiques simples comme les positions des barycentres.

Au cours du développement futur du projet, on peut imaginer utiliser un ensemble de gestes plus riches qui exploitent plus de propriétés des composantes connexes notamment la rotation, la taille relative, la vitesse de mouvement... Avec une collection plus complète de gestes, on pourrait éventuellement commander le drone pour décoller, atterrir... ainsi de suite.

5 Conclusion

Au cours de ce projet nous avons pu découvrir de nombreux aspects de la robotique comme l'utilisation de caméras et le traitement des données associé. Par exemple lorsque nous avons voulu utiliser les images reçues par la caméra du drone, il nous a fallu d'abord filtrer la couleur qui nous intéressait puis nous pencher sur la réduction du bruit de fond. C'est à cette occasion que nous avons pu découvrir ce qu'étaient les mathématiques morphologiques.

Dès lors que la détection d'image était opérationnelle nous avons mis au point une partie au-

tomatisme pour gérer les déplacements du drone consécutivement à l'image détectée. Ce projet nous a donc de plus permis d'avoir une introduction à l'automatisme. La réalisation de ce projet nous a donc apporté une meilleure compréhension des problématiques associées au vol des drones, à la reconnaissance d'image et à l'automatisme.

Enfin au cours des trois mois que nous avons consacré à ce projet, nous avons également découvert l'utilisation des outils de gestions des versions, les outils de compilation comme *pkg-config* et *CMake* etc. L'utilisation de ROS nous a permis d'avoir un projet logiciel bien structuré et extrêmement bien conçu. Finalement nous avons pu apprendre à gérer le code source de sorte que chacun puisse travailler en local et collaborer ensuite sur la même version de code.

6 Remerciement

Nous tenons à remercier M.Hervé FREZZA-BUET pour son soutien durant les trois mois que nous avons passé sur ce projet. Projet qui était relativement conséquent pour un projet de synthèse de première année, mais qui a pu avancer grâce à l'aide que nous avons reçu de la part de M.FREZZA-BUET.

Nous remercions également le laboratoire MALIS, de Supélec, sans lequel le projet n'aurait pas été possible. Le laboratoire nous a en effet prêté tout le matériel et donné le support informatique pour mener à bien ce projet. Nous remercions également toute l'équipe du projet ROBIA qui nous propose de faire face aux problèmes dans la vraie vie et d'acquérir une première expérience dans le domaine de l'intelligence artificielle et de l'apprentissage machine. Enfin, nous avons utilisé plusieurs projets open-source dont nous tenons à remercier les auteurs pour leur excellent travail.

Nous gardons de très bons souvenirs de cette extraordinaire expérience et remercions tous ceux qui ont suivi de proche ou de loin ce projet de synthèse. Nous espérons que notre travail pourra s'avèrer utile pour de futurs étudiants voulant en apprendre d'avantage sur la commande gestuelle de drone et les interfaces homme-machine.

Références

[1] Adrian Drumea Georges et Hervé Frezza-Buet. "Tracking fast changing non-stationary distributions with a topologically adaptive neural network : Application to video tracking". Anglais. In : *Proceedings of 15th European Symposium on Artificial Neural Networks (ESANN2007)*. Bruges, Belgique, avr. 2007, p. 43–48. URL : `http://hal-supelec.archives-ouvertes.fr/hal-00250981`.