Compte-rendu de projet : Contrôle d'un drone par les mouvements de la main

Responsable de Projet : Jérémy Fix
Paul Janin
Louis-Guillaume Dubois
Luc Absil

Lundi 8 Juin 2015

Table des matières

Compte-rendu 3

1 Cahier des charges 4

2 Présentation du matériel utilisé 6
2.1 Matériel physique utilisé . . . 6
2.1.1 Le drone . . . 6
2.1.2 La kinect . . . 6
2.2 Infrastructure logicielle utilisée . . . 6
2.2.1 *Robot Operating System* (R.O.S.) . . . 6
2.2.2 *Point Cloud Library* (PCL) . . . 7
2.2.3 Git . . . 7

3 Description des nœuds créés 8
3.1 Nœud `keyboard_cmd` . . . 8
3.2 Nœud `filter` . . . 8
3.3 Nœud `estimator` . . . 8
3.4 Nœud `commander` . . . 11

4 Réalisation pratique 13

Documentation 14

5 Installation 15
5.1 Installation des dépendances . . . 15
5.2 Installation du paquet . . . 15
5.2.1 Création d'un espace de travail catkin . . . 15
5.2.2 Déplacement du code . . . 15
5.3 Compilation . . . 15

6 Utilisation 15
6.1 Branchement de la Kinect et paramétrage . . . 15
6.1.1 Paramètres du filtre . . . 16
6.1.2 Autres paramètres . . . 16
6.2 Connexion au drone et pilotage . . . 17
6.2.1 Commande à la main . . . 17
6.2.2 Options et paramètres de la commande . . . 17
6.2.3 Notes sur `keyboard_cmd` . . . 18

7 Problème(s) rencontré(s) — Améliorations souhaitables 18

Première partie
Compte-rendu

Introduction

Ce projet mené durant trois mois s'est attaché à la commande d'un drone aéroporté par les mouvements de la main, détectés par un dispositif fixe. En effet, le contrôle d'un drone à l'aide d'un système de commande classique (joystick, clavier, smartphone) s'avère peu intuitif et inconfortable. Le développement d'un système de commande « à mains nues » permet de résoudre ces problèmes tout en ouvrant une dimension ludique et gratifiante pour l'opérateur du drone.

Afin de réaliser un tel système, nous avons mis en place une démarche de développement nous permettant de déterminer précisément les objectifs à atteindre et les structures à développer. Une fois le cahier des charges établi, nous avons construit une structure de commande par étapes, grâce à la flexibilité apportée par `ROS` et l'utilisation de `Git`, en éprouvant chaque étape de la commande par des tests dès que cela a été possible. Ces tests et l'impact qu'ils ont eu sur le développement sont décrits en dernière partie.

1 Cahier des charges

Afin de développer efficacement le système de commande, il est nécessaire de définir exactement les caractéristiques qu'il doit présenter.

On désire disposer d'une commande la plus intuitive possible, sans être équivoque du point de vue de la reconnaissance de mouvement. Afin de faciliter l'acquisition du mouvement et la mise en place du système, nous avons choisi de placer la Kinect au sol, pointée vers le plafond. L'opérateur peut alors commander le drone d'une main, debout ou assis.

On veut de plus que le mouvement de la main corresponde au mouvement du drone autour de son centre de gravité. Le détail des commandes désiré est donc donné ci-après :

- Une inclinaison de la main est associée à une translation du drone dans la direction impliquée par cette inclinaison ;
- Une rotation de la main autour de son axe vertical initie une rotation du drone autour du même axe, dans le même sens ;
- Un déplacement vertical de la main entraîne un déplacement vertical du drone dans le même sens, proportionnel à la vitesse de déplacement de la main. Cette commande testée durant le développement s'est révélée peu intuitive, c'est pourquoi elle à été modifiée pour que la vitesse de déplacement vertical du drone corresponde à l'écart entre la position vertical de la main et une position neutre arbitrairement définie ;
- Si la main n'est que partiellement ou pas du tout détectée par la Kinect, si celle-ci est immobile, ou si les mouvements détectés par la Kinect ne peuvent être interprétés de façon univoque, aucune commande ne doit être envoyée au drone, qui doit alors s'immobiliser. Il est notamment nécessaire d'ignorer les petits mouvements involontaires de la main lorsque l'opérateur désire garder le drone immobile.

- Le système de commande interne au drone comporte de base une méthode de gestion des incidents (arrêt en cas de collision trop violente, ou de contact des hélices avec un obstacle). On ajoute cependant un plafond d'altitude du drone afin de limiter les risques de chute.

- Si nécessaire, l'opérateur utilisera un gant de couleur définie pour faciliter l'acquisition des mouvements.

Les grandeurs exactes associées à la vitesse de déplacement du drone par rapport à celle de la main sont variables, et pourront être modifiées dynamiquement pendant le fonctionnement du système.

2 Présentation du matériel utilisé

2.1 Matériel physique utilisé

Pour réaliser ce projet, nous avons utilisé un *Parrot AR.Drone* ainsi qu'une kinect.

2.1.1 Le drone

Il s'agit d'un quadricoptère, alimenté par batteries. Il utilise un réseau wifi pour communiquer avec l'utilisateur.

Ce robot est prévu pour fonctionner avec **ROS**, grâce au *package* `ardrone_autonomy`. Ainsi les déplacements du drone sont déjà implémentés par les fabriquants, ce qui nous a facilité l'écriture du nœud `keyboard_cmd` permettant de contrôler le drone au clavier (utile lors du décollage, et surtout pour reprendre la commande du drone lors des premiers essais, afin d'en éviter la chute).

2.1.2 La kinect

Pour commander le drone avec la main, le cahier des charges nous imposait d'utiliser une kinect. Il s'agit principalement d'une caméra capable de détecter la couleur et la profondeur des images qu'elle perçoit.

L'image mesurée par la kinect est composée de points qui ont chacun une information de profondeur en plus de leur position dans l'image. Il est possible, et c'est ce que nous avons fait, d'associer à chacun des ces points, repérés dans l'espace, une information de couleur, au format RGB, mesurée par la caméra.

Nous mettrons à profit ces deux caractéristiques, tout d'abord en filtrant sur les couleurs pour être bien sûr que le nuage de point sur lequel nous travaillerons correspond bien à la main de l'utilisateur (cf. figures de la page 10). En outre, grâce à la détection de la profondeur, il nous est possible de choisir l'altitude du drone en fonction de la hauteur de la main par rapport à la kinect, et, plus important, de saisir la différence d'altitude entre les points de la main pour régresser un plan qui contiendra les commandes de translation et de rotation du drone.

2.2 Infrastructure logicielle utilisée

2.2.1 *Robot Operating System* (R.O.S.)

Pour réaliser ce projet, nous avons utilisé ROS, une structure logicielle libre (*framework*, en anglais) permettant de développer des logiciels pour la robotique. Le logiciel est simple à installer sur *Ubuntu*.

Les principaux concepts de ROS ROS introduit différents concepts absolument nécessaires dans l'écriture d'un programme de robotique :

- Les nœuds (*nodes*) : un exécutable qui utilise ROS pour communiquer avec d'autres nœuds ;
- Les messages : un type de donnée utilisé lors de la *souscription* ou de la *publication* vers un « sujet » *topic* ;

- Les « *topic* » : les nœuds peuvent publier des messages vers un *topic*, ou souscrire à un *topic* afin de recevoir les messsages que celui-ci émet.

Différentes commandes de ROS permettent aussi d'obtenir des informations sur les programmes tandis qu'ils s'exécutent. En ligne de commande (avec bash), on peut ainsi utiliser `rostopic`, ou `rosmsg`. Pour lancer les nœuds on utilise `roslaunch` et `rosrun`.

Nous avons choisi d'utiliser C++ pour coder nos nœuds (et non python, autre langage supporté par ROS), car nous avons pris connaissance de la librairie *PointCloud* qui nous a été utile pour traiter les images reçues par la Kinect.

2.2.2 *Point Cloud Library* (PCL)

PCL est une librairie libre écrite en C++ spécialisée dans le traitement des nuages de point (comme son nom l'indique).

2.2.3 Git

Pour travailler le projet, nous avons aussi choisi d'utiliser `git`, afin de synchroniser facilement chacun de nos développements.

3 Description des nœuds créés

3.1 Nœud `keyboard_cmd`

Ce nœud est en marge du projet à proprement dit puisqu'il ne permet pas de satisfaire le cahier des charges, mais plutôt de combler un manque, en permettant de publier des commandes sur le *topic* `cmd_vel`, pilotant de façon ergonomique le drone et affichant quelques informations utiles reçues du drone.

Il s'utilise avec la commande : `rosrun hand_control keyboard_cmd`.

L'affichage utilise la bibliothèque `ncursesw5`.

3.2 Nœud `filter`

Ce nœud réalise le filtrage du nuage de point (cf. figures 3 page 10 et 1 page 9) publié par le nœud « pilote » de la Kinect, dans le but de ne conserver que les points correspondant à la main ou au gant (cf. figures 4 page 10 et 2 page 9).

Pour cela, pour tous les points dont l'altitude est inférieure à un seuil défini par l'utilisateur, le nœud calcule les coordonnées HSV de sa couleur (à partir des coordonnées RGB fournies par la Kinect), et filtre suivant les conditions choisis par l'utilisateur (teinte du gain ou de la main, tolérance en teinte, saturations minimale et maximale, valeurs minimale et maximale).

Le nuage de point ainsi filtré doit suffisamment correspondre à la main pour en extraire les deux premières *composantes principales* (les deux directions du plan de la main).

3.3 Nœud `estimator`

Ce nœud régresse un plan et calcule l'angle que fait la main (son axe plus long) avec l'axe « y » de la Kinect — soit dans notre cas l'axe parallèle au sol et orthogonal à la Kinect — à partir du nuage de point publié par le nœud `filter`.

Il calcule aussi l'altitude moyenne du nuage de point.

Le nœud `estimator` publie les coordonnées de la normale au plan régressé, son altitude, ainsi que l'angle de la main avec l'axe « y », dans un format de message ROS créé pour l'occasion, `Plan.msg`.

Le calcul est effectué avec les outils fournis par la librairie PCL, notamment la classe `PCA`, qui renvoie les vecteurs propres et valeurs propres de la matrice de variance-covariance (pour les coordonnées spatiales uniquement) d'un nuage de point. Cela s'appelle la *principal component analysis (PCA)*, ou *analyse en composantes principales*.

Les vecteurs propres ayant les valeurs propres associées les plus grandes correspondent aux « axes » les plus longs du nuage de point, c'est-à-dire aux droites qui maximisent la variance de la projection du nuage de point sur ces droites.

Dans notre cas, les deux premiers vecteurs propres définissent le plan, et sa normale (par produit vectoriel).

La projection du premier vecteur propre sur l'axe « y » fournit le sinus de l'angle recherché, l'angle que fait la main avec la direction de l'avant-bras.

Ces informations sur la main (normale au plan, altitude et angle) serviront au nœud `commander` pour élaborer les commandes à adresser au drone.

Fig. 1 : Image perçue par la kinect sans le nœud `filter`, dans le cas d'un gant **noir** porté par l'utilisateur (cf. figure 2 page 9).

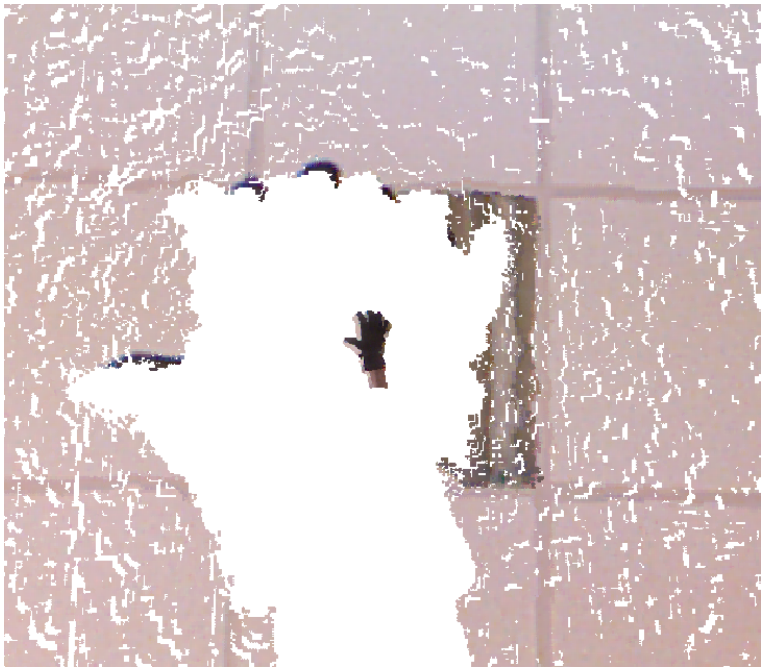


Fig. 2 : Image perçue par la kinect après filtrage par le nœud `filter`, dans le cas d'un gant **noir** porté par l'utilisateur (cf. figure 1 page 9).

Fig. 3 : Image perçue par la kinect sans le nœud `filter`, dans le cas d'un gant **vert** porté par l'utilisateur (cf. figure 4 page 10).

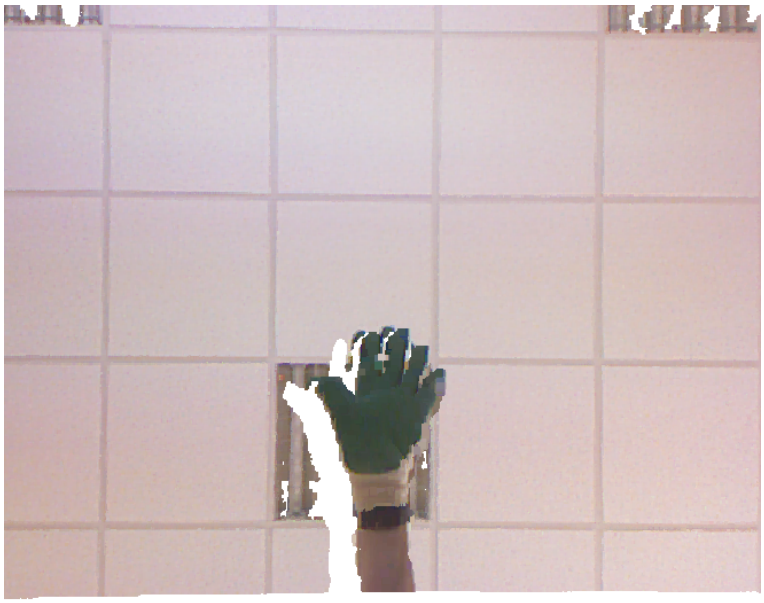


Fig. 4 : Image perçue par la kinect après filtrage par le nœud `filter`, dans le cas d'un gant **vert** porté par l'utilisateur (cf. figure 3 page 10).

Un exemple en deux dimensions d'une analyse en composantes principales est donnée sur la figure 5 page 12.

3.4 Nœud `commander`

Ce nœud souscrit au *topic* publié par `estimator`. L'orientation de la main et son altitude sont prises en compte pour établir la commande à envoyer au drone.

Translation dans un plan horizontal Pour des raisons pratiques apparues lors de nos tests, nous avons décidé qu'il était meilleur de discriminer l'orientation de la main selon x ou selon y : selon que la main est penchée vers l'avant (resp. l'arrière), ou sur la droite (resp. la gauche), le drone ira vers l'avant ou l'arrière, ou bien vers la droite ou la gauche. Nous refusons toutefois les mouvements simultanément de translation vers l'avant et la gauche. Il est bien sûr possible d'aller vers l'avant/l'arrière ou le côté, tout en tournant à gauche et à droite, en modifiant l'angle de la main avec l'avant-bras.

Translation selon (Oz) En revanche, le changement d'altitude (mouvement selon z) reste toujours possible. Dans une première version de notre *package*, nous avions choisi d'utiliser la vitesse de la main (vers le haut ou le bas) pour déterminer la vitesse selon z du drone.

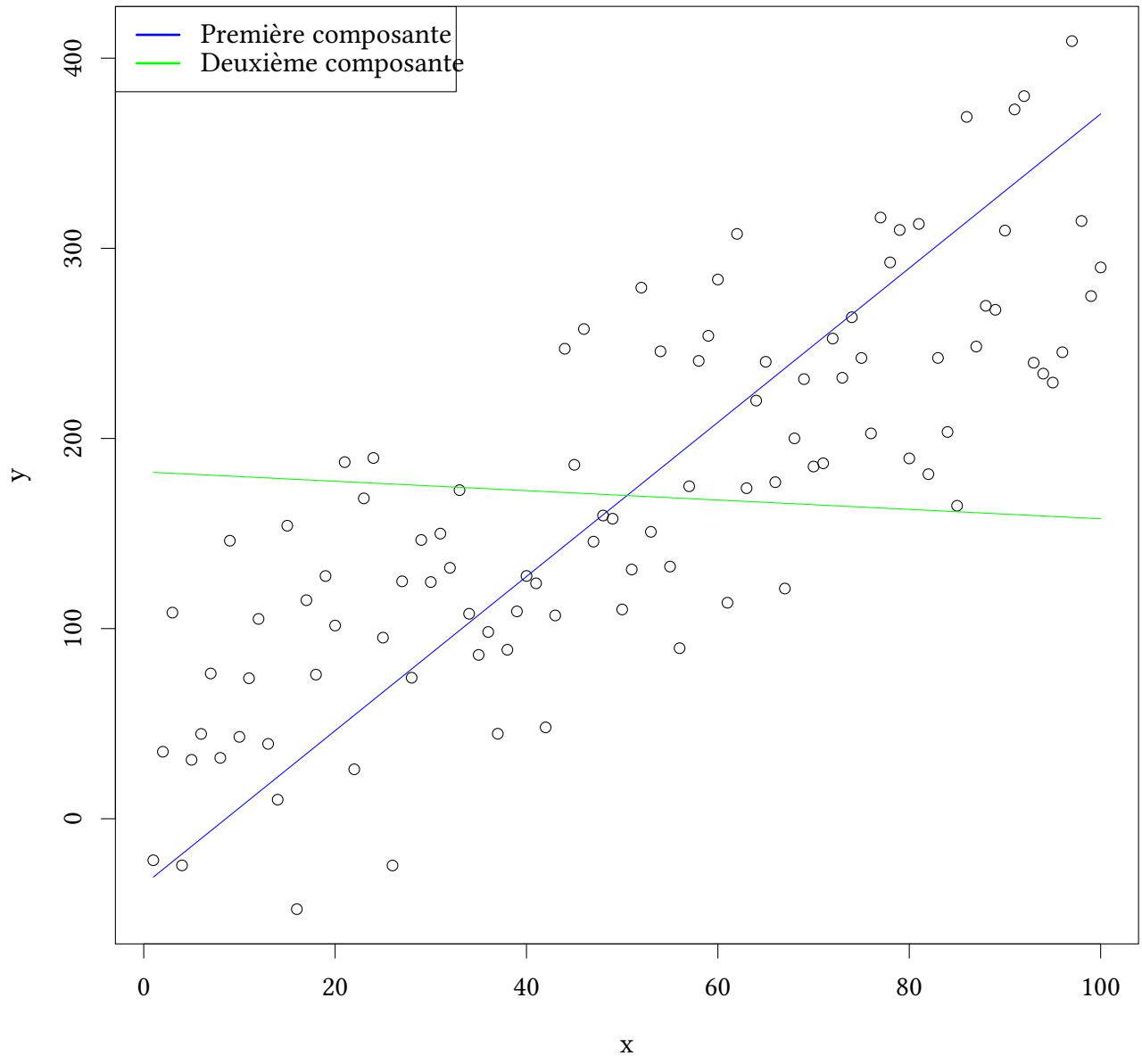
Toutefois, selon une recommandation de notre responsable de projet, Jérémy Fix, nous avons finalement choisi de considérer l'altitude absolue de notre main par rapport à la kinect. Au-dessus d'une certaine hauteur `neutral_z`, le drone s'élève avec une vitesse proportionnelle avec la hauteur de notre main par rapport à cette hauteur `neutral_z` [1] (et de même en-dessous pour que le drone se rapproche du sol).

Rotation autour de l'axe (Oz) Nous avons en outre amélioré notre projet par rapport au cahier des charges initialement imposé en corrélant la rotation de notre main avec celle du drone, suivant une proposition de Jérémy Fix.

Un coefficient de proportionnalité, paramétrable, transforme l'angle de la main en une commande de rotation du drone autours de l'axe vertical.

[1] Il est possible de modifier cette hauteur (ainsi que d'autres paramètres) sans recompiler notre code, grâce à l'utilisation de `dynamic reconfigure`, recommandée par Jérémy Fix.

Fig. 5 : Composantes principales d'un nuage de points (deux dimensions)



4 Réalisation pratique

Cette partie décrit les problèmes rencontrés et les évolutions amenées lors des tests de la commande du drone.

Les premiers tests ont été menés avec une version primitive des noeuds `commander`, `filter` et `estimator`. À ce stade, le mouvement angulaire n'était pas encore implémenté, le système de régression du plan était sommaire, et le filtrage des couleurs était réalisé sur les valeurs RGB transmises par PCL. De nombreux problèmes sont ressortis de ces tests ; il était notamment impossible de commander le drone à la main, du fait de fluctuations trop importantes dans la régression planaire et le filtrage des couleurs. D'autre part, les commandes du drone se sont révélées peu intuitives et mal ajustées pour un contrôle dynamique.

Suite à ces tests, la plupart des nœuds ont été modifiés pour répondre à ces problèmes. Le nœud `filter` a été revu afin de filtrer les couleurs selon une échelle HSV (Hue, Saturation, Value) et non plus RGB afin de permettre une meilleure gestion des fluctuations d'éclairage. Le nœud `estimator` a été réécrit en utilisant des fonctions de la classe `PCA` permettant une meilleure régression planaire et le calcul du déplacement angulaire du plan régressé. Par suite, le nœud `commander` a également été modifié par l'implémentation du mouvement angulaire et la mise en place de commandes plus intuitives pour les translations longitudinales et verticales.

Ces modifications ce sont révélées probantes, le drone ayant été nettement plus aisé à contrôler lors d'essais ultérieurs.

Conclusion

Au terme de ce projet, nous pouvons dire que les démarches utilisées pour satisfaire le cahier des charges établi ont fourni des résultats satisfaisants, la commande du drone étant simple et précise. Ainsi par exemple, nous avons pu constater que des élèves n'ayant pas de connaissances a priori sur notre code ont facilement pu le commander à l'aide de la main. Plusieurs tests dans des conditions diverses au cours du projet nous ont permis de bien appréhender le matériel utilisé ainsi que les méthodes à implémenter pour atteindre notre objectif. Quelques améliorations restent possible cependant, la commande ne fonctionnant de façon optimale qu'avec un gant de couleur vive, sous un éclairage élevé ; son utilisation dans un milieu quelconque, quoique possible, reste imparfaite. Cependant, le mode de commande proposé reste très satisfaisant tant par sa simplicité que par son intuitivité, et pourrait être appliqué à d'autres systèmes utilisant des commandes traditionnelles.

Deuxième partie
Documentation

5 Installation

5.1 Installation des dépendances

```
#!sh
sudo apt-get install ros-indigo-desktop-full ros-indigo-freenect-stack
ros-indigo-ardrone-autonomy libncursesw5-dev
```

5.2 Installation du paquet

5.2.1 Création d'un espace de travail catkin

Par exemple :

```
#!sh
source /opt/ros/indigo/setup.bash
mkdir -p ~/hand_control_ws/src
cd ~/hand_control_ws/src
catkin_init_workspace
```

5.2.2 Déplacement du code

Renommer si besoin est le dossier qui contient les sources en `hand_control`, et le déplacer dans `~/hand_control_ws/src/`, ou dans le sous-dossier `src` de votre espace de travail catkin.

5.3 Compilation

Il est ensuite possible de compiler :

```
#!sh
cd ~/hand_control_ws # ou votre espace de travail catkin
catkin_make
```

Puis pour pouvoir utiliser les commoandes ROS, en remplaçant si besoin "hand_control_ws" par votre espace de travail catkin :

```
#!sh
source /opt/ros/indigo/setup.bash
source ~/hand_control_ws/devel/setup.bash
echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc
echo "source ~/hand_control_ws/devel/setup.bash" >> ~/.bashrc
```

6 Utilisation

6.1 Branchement de la Kinect et paramétrage

1. Brancher la Kinect (sous tension) à l'ordinateur par USB ;

2. Poser la Kinect sur le sol, pointant le plafond, votre bras devra être perpendiculaire à la Kinect pour pouvoir bien piloter le drone ;
3. Lancer le "launchfile" kinect_commander.launch :

   ```
   roslaunch hand_control kinect_commander.launch
   ```

4. Vérifier les paramètres du filtre :
   - lancer rviz : `rosrun rqt_rviz rqt_rviz`
   - visualiser la sortie du filtrage et repérer la main ; (topic : `/filter/output` ; frame : `/camera_depth_optical_frame`)
   - lancer rqt_reconfigure : `rosrun rqt_reconfigure rqt_reconfigure` pour :
   - modifier les paramètres du filtre jusqu'à ne voir que les points de la main/gant/-pancarte sur rviz (voir ci-dessous).
   - modifier le paramètre `neutral_alt` du nœud `commander` à la hauteur souhaitée (en mètres). C'est la hauteur de la main qui correspondra à l'immobilité de l'altitude.

6.1.1 Paramètres du filtre

Les paramètres du filtre (modifiables avec `dynamic_reconfigure` et en particulier `rqt_reconfigure`) sont :

- `z_max` : en mètres, altitude maximale de la main, doit être inférieure à la hauteur du plafond.
- pour un gant ou un carton *coloré* (vert, bleu etc.), on a généralement :
  - `hue` : par exemple 220 (bleu ciel) ou 150 (vert) ou 0 (rose/rouge) ;
  - `delta_hue` : entre 10 et 20 ;
  - `sat/val_min` : 0.0 ;
  - `sat/val_max` : 1.0 ;
- pour un gant *noir* :
  - `hue` : 0 ;
  - `delta_hue` : 180 ;
  - `sat_min` : 0.0 ;
  - `sat_max` : 1.0 ;
  - `val_min` : 0.0 ;
  - `val_max` : 0.3 (à modifier à votre convenance) ;

6.1.2 Autres paramètres

Toujours avec `rqt_reconfigure`, cette fois pour le nœud `estimator` :

- `reverse` : échange x et y (axes de la Kinect) (valeur par défaut pour une utilisation normale : faux [décoché])
- `reverse_angle` : modifie l'axe choisi pour calculer l'angle de la main (valeur par défaut pour un utilisation normale : faux [décoché])

6.2 Connexion au drone et pilotage

- Connecter l'ordinateur au réseau wifi du drone ;
- Lancer le "launchfile" ardrone.launch : `roslaunch hand_control ardrone.launch` ;
- Pour décoller :
  - soit `rostopic pub /ardrone/takeoff std_msgs/Empty` ;
  - soit lancer le nœud keyboard_cmd : `rosrun hand_control keyboard_cmd` et utiliser la touche *t* du clavier.
- Pour atterrir :
  - soit `rostopic pub /ardrone/land std_msgs/Empty` ;
  - soit, avec keyboard_cmd, utiliser la touche *b* du clavier.
- Arrêt d'urgence :
  - soit `rostopic pub /ardrone/reset std_msgs/Empty` ;
  - soit, avec keyboard_cmd, utiliser la touche *g* du clavier.

6.2.1 Commande à la main

- Avancer/reculer & translations latérales : inclinaison de la main ;
- Tourner (rotation autours de l'axe z) : angle de l'axe de la main avec l'axe parallèle au sol et perpendiculaire à la Kinect ;
- Monter/descendre : altitude de la main.

6.2.2 Options et paramètres de la commande

Pour éditer les options de la commande, lancer si ce n'est déjà fait `rosrun rqt_reconfigure rqt_reconfigure` :

- `max_curvature` : non utilisé pour l'instantt ;
- `x/y/z/theta_minimal_deviation` : seuils à partir desquels le mouvement de la main est pris en compte. Tout mettre à 0.0 rend le comportement complétement linéaire.
  - x, y : entre 0. et 1. (il s'agit des composantes x et y de la normale au plan) ;
  - z : en mètre ;
  - theta : en degrés.
- `neutral_alt` : hauteur de la main qui correspond à l'immobilité de l'altitude du drone ;
- `min_points_number` : nombre minimal de points (du nuage de points qui a servi à régresser le plan reçu) nécessaire pour qu'une commande soit envoyé au drone.
- `angle/x/y/z_vel` : coefficients de proportionnalité à appliquer aux données en entrée pour établir la commande envoyée au drone. Les augmenter augmentera la vitesse du drone.
- `up_fact` : coefficient de proportionnalité à appliquer à la commande augmentant l'altitude du drone, par rapport à la commande équivalente la diminuant (pour corriger l'effet de la gravité). Une valeur de 1.5 a été bonne dans nos essais.

6.2.3 Notes sur `keyboard_cmd`

Il permet de publier des commandes sur le topic `cmd_vel` et ainsi de piloter le drone. Il est prévu pour les claviers azerty. Pour le lancer :

```
#!sh
rosrun hand_control keyboard_cmd
```

Pour augmenter/diminuer les vitesses (expliqué sur l'affichage du programme) : touches a,z,e,r et w,x,c,v.

L'affichage des informations reçues du drone n'est mise à jour qu'à l'occasion de l'appui sur une touche.

Pour quitter : Ctr+C, et appui sur "Entrée" pour retrouver l'affichage de la console.

7 Problème(s) rencontré(s) — Améliorations souhaitables

- Si des commandes sont publiées sur `cmd_vel` (depuis la Kinect par exemple) après le lancement du fichier `ardrone.launch` et avant le décollage du drone, alors, après le décollage, le drone semble obéir aux commandes publiées avant le décollage.
- Comme écrit plus haut, l'affichage des informations reçues du drone sur `keyboard_cmd` n'est mise à jour qu'à l'occasion de l'appui sur une touche, et peut donc rester fixe quand on n'utilise pas la commande au clavier.
- Le décollage/atterrissage n'est pas possible à commander avec la main. Il faut utiliser le clavier (`keyboard_cmd` ou `rostopic pub`) à la place. Il est possible de de remédier à cela en créant deux nouveaux seuils, minimaux et maximaux, pour la hauteur de la main : une main très basse ferait atterrir le drone, une main très haute le ferait décoller.