# Rapport final de projet de fin d'étude

## P21 : Pilotage automatique de drone

Justine Senellart & Claire Vandamme

Février 2019

# Remerciements

Nous tenons tout d'abord à remercier nos encadrants, madame Claudine Lecocq et monsieur Aziz Nakrachi pour nous avoir proposé ce projet de fin d'étude, pour nous avoir apporté certaines connaissances indispensables au bon déroulement du projet et pour leur aide et le temps qu'ils nous ont accordé tout au long de celui-ci.

Ensuite, nous tenons à remercier tout particulièrement Vincent Coelen pour nous avoir donné de son temps lorsque nous avions besoin d'aide, pour ses explications, notamment sur l'environnement de travail ROS.

Merci également à l'équipe du laboratoire CRISTAL pour la mise à disposition du matériel. Et aussi à Abass Ayoub et Alexis Viscogliosi, binôme du projet P36, pour leur coopération et leur soutien tout au long de ce projet.

# Sommaire


**1 Introduction** **3**

**2 Présentation du projet** **4**
2.1 Cahier des charges . . . . . 4
2.1.1 Contexte . . . . . 4
2.1.2 Objectifs . . . . . 4
2.1.3 Besoins fonctionnels . . . . . 4
2.1.4 Résultats attendus . . . . . 5

**3 Travail Préliminaire** **6**
3.1 Etude du drone . . . . . 6
3.2 Étude de différentes technologies . . . . . 7
3.2.1 Le SDK de Parrot . . . . . 7
3.2.2 La programmation avec Python (PyParrot) . . . . . 7
3.2.3 L'environnement ROS . . . . . 7
3.2.4 Solution retenue . . . . . 9
3.3 Traitement d'image . . . . . 9
3.3.1 OpenCV . . . . . 9
3.3.2 Exploration d'OpenCV . . . . . 9

**4 Travail réalisé** **10**
4.1 Récupération de la vidéo . . . . . 10
4.1.1 Partie ROS . . . . . 10
4.1.2 Conversion format image . . . . . 11
4.1.3 Récupération d'un nombre d'images par seconde . . . . . 12
4.2 Traitement d'images . . . . . 13
4.2.1 Pré-traitement . . . . . 13
4.2.2 Traitement . . . . . 14
4.2.3 Conclusion . . . . . 18
4.3 Envoi de commandes au drone . . . . . 19
4.3.1 Quel topic utiliser ? . . . . . 19
4.3.2 Publier avec ROS via Python . . . . . 19
4.3.3 Fonctions de direction modulables selon la vitesse ou la distance . . . . . 20
4.3.4 Arrêt du programme . . . . . 21

**A Récupération d'images à partir de la vidéo** **23**

# Chapitre 1

# Introduction

Dans le cadre de notre dernière année en tant qu'étudiantes en Informatique, Microélectronique et Automatique à Polytech Lille, nous avons l'opportunité de mettre en œuvre les connaissances acquises durant nos années d'études en un projet.

Notre projet a pour but de contrôler les trajectoires d'un drone en suivant des consignes de déplacement visuelles. Pour ce projet, nous définissons que les consignes visuelles seront représentées par des flèches. Nous avons décidé de choisir ce projet afin de découvrir et/ou d'élargir nos connaissances en matière de traitement d'images et d'améliorer nos compétences en programmation.

Ce rapport apparaît dans le cadre de ce projet de fin d'études afin de rendre compte, au terme des 4 mois et demi alloués, du travail accompli. Ainsi, plusieurs points seront abordés dans ce rapport, comme le cahier des charges, le travail effectué, les difficultés rencontrées et une ouverture sur ce qu'il est possible d'améliorer.

# Chapitre 2

# Présentation du projet

## 2.1 Cahier des charges

### 2.1.1 Contexte

Le laboratoire CRISTAL de Lille a récemment acquis une flotte de drone Bebop2 de Parrot qu'il désire éventuellement utiliser pour la recherche en tant que flotte de drones pour réaliser un travail collaboratif ou pour l'enseignement de manière à réaliser une flotte avec les robotinos déjà présents à l'école. Le but de notre projet est d'explorer les capacités du drone Bebop 2 de manière à exploiter et documenter les différentes technologies qui peuvent être utilisées avec ce drone ainsi que de voir quel type de projet il est possible de réaliser avec ce drone. Ceci pourra alors être repris dans le cadre par exemple d'un TP, ou d'un futur projet.

### 2.1.2 Objectifs

L'objectif final de notre projet est donc de détecter les consignes visuelles proposées au drone. Ainsi, nous allons devoir détecter l'orientation d'une flèche dans les images provenant du flux vidéo du drone pour qu'ensuite il se déplace dans l'orientation de la flèche. Ceci nous permettra d'explorer les techniques de traitement d'images ainsi que les possibilités et facultés visuelles du drone. Pour cela, nous avons décidé de découper le projet en plusieurs sous-objectifs :

- Détecter des flèches de couleurs différentes dans l'image en associant une couleur avec une direction
- Détecter l'orientation d'une flèche.
- Traiter les informations perçues pour les transformer en instruction pour le drone
- Envoyer les commandes au drone pour le faire voler

### 2.1.3 Besoins fonctionnels

| Reconnaître une flèche dans la vidéo | |
|---|---|
| Repérer la flèche | Repérer une flèche grâce à sa couleur ou sa forme dans les images provenant du flux vidéo du drone |
| Détection du sens de la flèche | Traitement de l'image pour extraire la direction de la flèche grâce à sa couleur ou sa géométrie |
| Envoi des résultats | Envoi des résultats concernant la flèche pour traitement |

| Voler dans la direction d'une flèche | |
|---|---|
| Réception des données | Réception des données reçues concernant les flèches |
| Traitement des données | Traitement des données reçues pour les transformer en commandes à envoyer au drone |
| Faire voler le drone | Envoi des commandes au drone lui permettant de voler dans la direction de la flèche détectée |

**Contraintes techniques**

Tout au long du projet, nous allons devoir nous intéresser à l'aspect "temps réel" : en effet, le but est que le temps de réponse entre le moment où l'utilisateur lui montre la flèche et le moment où le drone effectue le déplacement soit le plus faible possible, et idéalement immédiat.

**Partie matérielle** Concernant ce projet, nous utilisons le drone Bebop 2 de Parrot qui est le drone utilisé par les chercheurs du laboratoire Cristal.

**Partie logicielle** Pour programmer le drone, nous travaillons en utilisant un environnement Linux avec lequel nous utilisons la plate-forme de développement logiciel pour robot ROS. De plus, la programmation du drone sera effectuée en Python car il s'agit d'un des langages les plus performants lors de l'utilisation de la bibliothèque OpenCV qu'on utilisera pour le traitement d'images. Cependant, étant donné qu'une partie de notre projet consiste à exploiter les capacités du drone et des différentes technologies utilisables avec ce dernier nous ne nous limiterons pas à ROS et Python lors de notre découverte du drone.

### 2.1.4 Résultats attendus

Suite à notre étude complète du drone et de notre travail sur le traitement d'images, à la fin de ce projet, le drone devrait être capable de suivre la direction d'une flèche qu'un utilisateur lui présente lorsqu'il est en vol. Cette réponse devant être évidemment la plus rapide possible pour respecter l'aspect temps réel.

# Chapitre 3

# Travail Préliminaire

## 3.1 Etude du drone

FIGURE 3.1 – Le drone Bebop2

Le drone Bebop 2 est un drone dernière génération conçu par Parrot. En plus de la caméra frontale qui permet de filmer pendant le vol le drone possède plusieurs capteurs :

- **Une caméra verticale de stabilisation** qui prend une image du sol toutes les 16 millisecondes et la compare à la précédente pour déterminer la vitesse du drone
- **Un capteur ultrason** qui analyse l'altitude de vol jusqu'à 4,87 m (16 feet)
- **Un capteur de pression** qui mesure la pression de l'air pour calculer l'altitude au-delà de 4,87 m.
- **Un gyroscope 3 axes** qui mesure l'angle d'inclinaison du drone
- **Un accéléromètre** qui mesure la position du drone sur trois axes et sa vitesse linéaire
- **Un système mondial de navigation par satellite (GNSS)** composé d'un GPS et d'un GLONASS pour la géolocalisation du drone et qui aide à mesurer la vitesse du drone pour sa stabilisation à haute altitude.

Nous aurons accès aux valeurs de ces différents capteurs pour la programmation du drone mais pas aux images prises par la caméra verticale.

## 3.2 Étude de différentes technologies

Pour la réalisation de ce projet, nous avons dans un premier temps étudié plusieurs technologies de manière à trouver celle qui répond le mieux à nos besoins pour le contrôle du drone.

### 3.2.1 Le SDK de Parrot

[3] Parrot dispose d'un outil de développement qui propose, dans un premier temps, une interface homme machine permettant de contrôler le drone ainsi que la possibilité de réaliser des programmes en C permettant de contrôler le drone et de recevoir les données de quelques capteurs comme par exemple l'indicateur de niveau de batterie.

### 3.2.2 La programmation avec Python (PyParrot)

Nous avons étudié plusieurs bibliothèques Python développées pour le drone Bebop2, principalement PyParrot, mais nous nous sommes rendues compte que pour la plupart de celles-ci, cela utilisait la même "structure" que le SDK.

### 3.2.3 L'environnement ROS

#### Qu'est ce que ROS ?

ROS (Robot Operating System) est un système open source qui permet à un utilisateur de contrôler un robot à partir d'un PC. Il n'a pas de langage de programmation défini et peut être programmé via plusieurs langages. Un système ROS comprend un certain nombres de nœuds indépendants qui communiquent avec les autres nœuds via une messagerie de type publication/abonnement. Par exemple, le driver d'un capteur peut être implémenté comme un nœud qui publie les valeurs du capteur dans un flux de messages. Ces messages pourraient être utilisés par n'importe quel nombre d'autres nœuds, comme des filtres, des collecteurs de données ou des systèmes haut niveau comme des systèmes de guidage ou de recherche de chemin.

#### Pourquoi utiliser ROS ?

ROS est une plate-forme de développement très populaire qui propose de nombreuses fonctions comme l'échange de messages, l'informatique distribuée, la réutilisation de code, ... ROS propose un support pour un grand nombre de capteurs et actionneurs ce qui le rend compatible avec un grand nombre de robots. De ce fait, la communauté ROS s'agrandit rapidement et cet outil s'est rapidement imposé comme un standard de la programmation robotique si bien que la plupart des entreprises de robotique haut de gamme utilisent ROS. De plus, ROS propose aux développeurs de nombreux outils pour le débogage, la visualisation des données et la simulation.

#### Comment fonctionne ROS ?

[2] Le calcul dans ROS est effectué à l'aide d'un réseau de processus appelé nœuds ROS. Ce réseau d'exécution peut être appelé graphe de calcul. Les principaux concepts du graphe sont les nodes ROS, le master, le parameter server, les messages, les topics, les services et les bags. Chaque concept du graphique contribue au réseau d'une manière qui lui est propre.

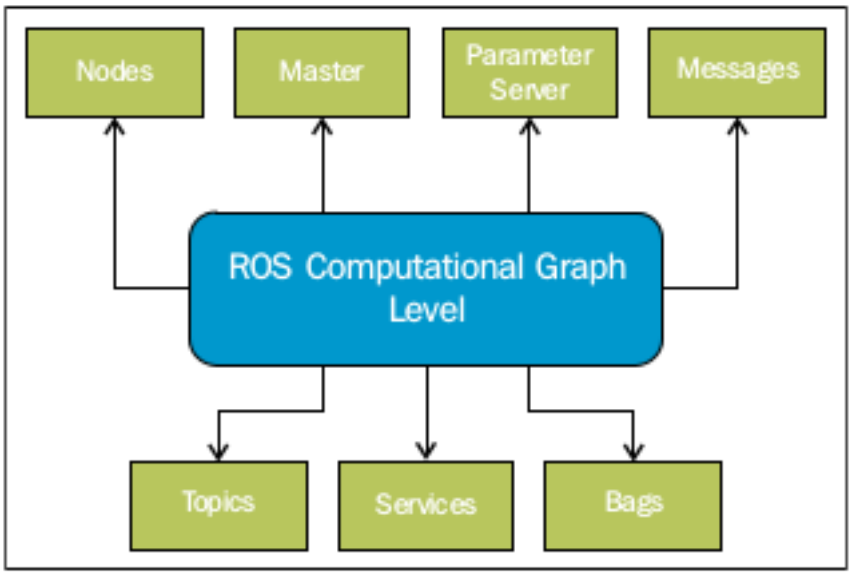


FIGURE 3.2 – Graphique de calcul de ROS

— Les **nodes** sont les processus qui effectuent les calculs. Chaque node ROS est écrit à l'aide d'une bibliothèque pour ROS telle que roscpp (pour le C++) et rospy (pour le python). Dans un robot, il peut y avoir plusieurs nodes pour exécuter différents types de tâches. En utilisant la méthode de communication de ROS, les nodes peuvent communiquer entre eux et échanger des données. Un des buts des nodes ROS est de découper un grand processus avec toutes les fonctionnalités en plusieurs petit processus avec une seule fonctionnalité.
— Le **Master** ROS fournit un registre des noms des nodes activés et surveille les changements au niveau des nodes. Les nodes ne sont pas capables de trouver les autres nodes, d'échanger des messages ou d'invoquer des services sans Master.
— Le **parameter server** permet à l'utilisateur de sauvegarder les données dans un endroit centralisé. Tous les nodes peuvent accéder à ses valeurs et les modifier.
— Les nodes communiquent entre eux en utilisant des **messages**. Les messages sont de simples structures de données contenant des champs typés qui peuvent contenir un ensemble de données qui peuvent être envoyé à un autre node. Il y a des types standards de données (integer, floating point, boolean ...) et ces types sont supportés par les messages ROS. Il est aussi possible de créer des types personnalisés de messages à partir de ces types.
— Chaque message dans ROS est transporté à l'aide de bus nommé appelé **topics**. Quand un node envoie un message au travers d'un topic, on dit qu'il publie (publish) un topic. Quand un node reçoit un message au travers d'un topic, on dit qu'il s'abonne (subscribe) à un topic. Le node qui envoie et celui qui s'abonne ne sont pas au courant de l'existence l'un de l'autre. Chaque topic à un nom unique et n'importe quel node y a accès et peut envoyer des données au travers de ce dernier tant qu'ils utilisent le bon type de message.
— Pour certains cas d'application, un modèle de communication publish/subscribe n'est pas suffisant il est alors nécessaire d'utiliser une interaction requête/réponse. Les **services** ROS permettent de mettre en place une interaction bi-directionnelle de type requête/réponse. On peut définir un service comme contenant deux parties : une pour la requête et l'autre pour la réponse. En utilisant les services ROS, il est possible de créer un node serveur et un node client. Le node serveur fournit le service sous un nom et le client envoie des message de requêtes à ce serveur

qui répondra en envoyant le résultat au client.

— Le **Bag** est un format de fichier permettant de sauvegarder et rejouer les données des messages ROS. Le fichier Bag est un mécanisme important pour la sauvegarde de données telles que les données provenant des capteurs qui peuvent être difficiles à acquérir mais qui sont nécessaires pour développer et tester les algorithmes.

### 3.2.4 Solution retenue

Suite à notre étude des différentes technologies, nous avons décidé d'utiliser ROS pour nous aider à réaliser la communication avec le drone ainsi que pour son contrôle car il s'agit de la technologie que nous maîtrisons le mieux et avec laquelle nous avons obtenu les résultats les plus prometteurs.

## 3.3 Traitement d'image

### 3.3.1 OpenCV

[5] OpenCV (Open Source Computer Vision) est une bibliothèque Open Source qui propose plus de 2500 algorithmes pour le traitement d'images et le machine learning. Ces algorithmes peuvent être utilisés pour détecter et reconnaître des visages, identifier des objets, classifier des actions humaines dans des vidéos, suivre les mouvements de la caméra, suivre des objets, extraire les modèles 3D des objets, produire des images de points en 3D à partir de caméras stéréos, assembler des images pour produire une scène haute résolution, trouver des images similaires dans une base de données, suivre les mouvements des yeux... Elle a été créée pour proposer une infrastructure commune aux projets de vision par ordinateur et a été réalisée avec une considération importante pour l'efficacité de calcul et les applications temps réel.

### 3.3.2 Exploration d'OpenCV

Nous allons donc utiliser la bibliothèque OpenCV en programmant en Python pour la mise en place des fonctions de base du traitement d'images. Pour nous aider à la prise en main de cet outil ainsi que pour découvrir le traitement d'images, nous avons mis en place une sorte d'application nous permettant d'appliquer diverses fonctions de traitement sur une image d'entrée. Cela nous a permis de visualiser l'effet des différentes fonctions ainsi que de nous donner des idées pour trouver des flèches dans une image et identifier leur couleur ainsi que leur direction. Ainsi, l'application nous propose d'effectuer les traitements ci-contre.

```
Application
├── Find a colored object
│   ├── Blue
│   ├── Green
│   └── Red
├── Skeletisation
│   ├── Rectangle kernel
│   ├── Ellipse kernel
│   └── Cross kernel
├── Contours
│   ├── Contours
│   ├── Clean contours
│   ├── Center of mass
│   ├── Area
│   ├── Bounding box
│   └── Enclosing circle
├── Canny edge detection
├── Corner detection
├── Find a circle
├── Find basic shapes
├── Gradient
│   ├── Laplacian
│   ├── Sobel X
│   └── Sobel Y
└── Threshold
    ├── Global thresholding
    ├── Adaptive mean thresholding
    └── Adaptive Gaussian thresholding
```

FIGURE 3.3 – Différentes possibilités de l'application

# Chapitre 4

# Travail réalisé

Pour réaliser ce projet, nous avons décomposé le travail en plusieurs étapes : tout d'abord, la liaison entre le drone et l'ordinateur de manière à récupérer la vidéo capturée par le drone et extraire une image toutes les $n$ secondes. Une fois l'image récupérée, on la passe en entrée de l'algorithme de traitement d'images qui nous donne en sortie s'il y a une flèche ou non dans l'image et si oui sa direction. Pour finir on utilise la direction détectée pour envoyer une commande au drone.

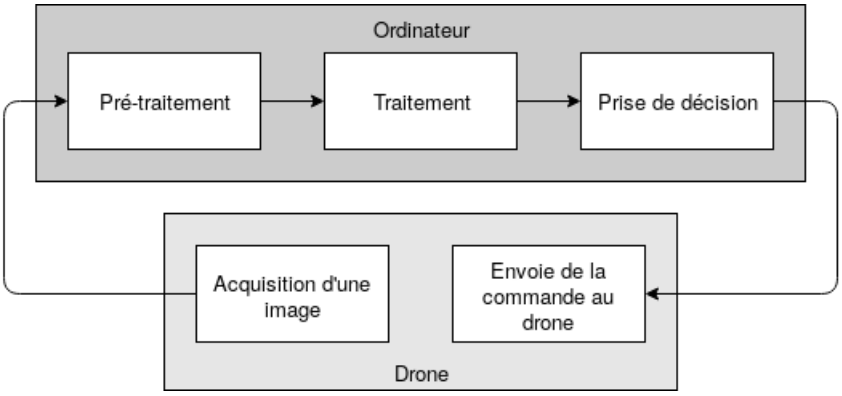


FIGURE 4.1 – Chaîne complète

## 4.1 Récupération de la vidéo

Pour réaliser la récupération de la vidéo nous allons devoir mêler deux des technologies présentées : ROS (en alliance avec le langage de programmation python) et OpenCV. Ensuite, nous allons expliquer comment récupérer uniquement une partie des images de ce flux vidéo.

### 4.1.1 Partie ROS

Dans un premier temps, ROS va nous permettre de récupérer le flux vidéo qui est donné par la caméra frontale du drone. Comme expliqué précédemment dans la présentation des technologies, avec ROS, nous pouvons souscrire à des topics. Dans notre cas, nous allons souscrire au topic image_raw qui est celui donnant les informations concernant la sortie de la caméra frontale.

```
justine@justine-Latitude:~/bebop_ws/src/projet/src$ rosmsg show Image
[sensor_msgs/Image]:
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step
uint8[] data
```

FIGURE 4.2 – Données du topic image_raw

Nous allons donc tout d'abord créer un programme pour essayer de récupérer ce flux vidéo.

Comme tout programme fonctionnant avec ROS, dans le main, on commence par initialiser un nœud que nous appellerons ici arbitrairement : image_listener. On définit ensuite dans une variable, le topic auquel on souhaite souscrire. Puis, on met en place la souscription en précisant le topic, le type de message et la fonction que l'on souhaite utilisée à ce moment là. Enfin, on ajoute la commande rospy.spin() afin de réaliser les instructions présentes dans le main en boucle, jusqu'à l'arrêt du programme (Ctrl+C).

```
def main():
        rospy.init_node('image_listener')
    # Define your image topic
        image_topic = "bebop/image_raw"
    # Set up your subscriber and define its callback
        rospy.Subscriber(image_topic, Image, image_callback)
    # Spin until ctrl + c
        rospy.spin()
if __name__ == '__main__':
        main()
```

Malheureusement en la récupérant via ROS, le format de sortie de la vidéo ne nous permet pas de la traiter avec OpenCV. Nous allons donc devoir effectuer quelques opérations afin de l'obtenir dans un format standard utilisable par OpenCV.

### 4.1.2 Conversion format image

Afin de transformer le format de sortie de la caméra que nous recevons sous ROS, nous allons devoir mettre en place un pont. En instanciant dans un premier temps ce pont, puis, en convertissant l'image ROS vers OpenCV, avec la fonction suivante :

$$\text{cv_image} = \text{bridge.imgmsg_to_cv2(image_message, desired_encoding)}$$

Avec :
- image_message, notre entrée (la sortie de la caméra)
- desired_encoding, le type de codage souhaité en sortie

Dans notre cas, le type de codage sera 'bgr8', afin d'obtenir des images couleur avec comme ordre de couleur bleu-vert-rouge.

Une fois ceci fait, il nous faut pouvoir sauvegarder les images de ce flux vidéo. Pour cela, nous allons pouvoir utiliser la fonction suivante :

$$\text{cv2.imwrite(filename, img, params)}$$

Avec :
- filename, le nom de sortie de l'image sauvegardée
- img, l'image d'entrée à sauvegarder
- params, qui est un argument facultatif, si l'on souhaite un format spécial de l'image

Au final, nous obtenons le code suivant :

```
# Instantiate CvBridge
bridge = CvBridge()

def image_callback(msg):
        try:
        # Convert your ROS Image message to OpenCV2
                cv2_img = bridge.imgmsg_to_cv2(msg, "bgr8")
        except CvBridgeError, e:
                print(e)
        else:
        # Save your OpenCV2 image as a jpeg
        cv2.imwrite('camera_image.jpeg', cv2_img)
```

### 4.1.3 Récupération d'un nombre d'images par seconde

Il nous faut maintenant penser à la suite du programme en l'adaptant à notre projet. En effet, pour le traitement d'images, il n'est pas nécessaire de récupérer toutes les images que peut nous fournir le drone. On va donc se limiter, et modifier le programme réalisé précédemment dans ce sens.

Premièrement, pour l'instant nous n'enregistrons qu'une image. Il faut alors modifier la fonction cv2.imwrite et trouver un nom d'image qui se modifie de lui même afin de pouvoir enregistrer plusieurs images à la suite, sans que la précédente soit écrasée (car elles auraient le même nom). La solution apportée à ce problème a été d'utiliser une fonction nous donnant le jour et l'heure actuelle avec une grande précision :

$$\text{date} = \text{datetime.datetime.now()}$$

En passant cette variable en string, elle peut être utilisée directement dans la fonction cv2.imwrite() comme nom d'image.

Ensuite, il nous faut récupérer les images qu'à une certaine fréquence, car toutes les images récupérées ne sont pas nécessaires. Nous avons donc eu l'idée d'utiliser la fonction que nous venions de trouver pour la date, en récupérant les microsecondes de l'heure actuelle, ce qui nous permettra de faire comme un timer en précisant que l'on sauvegarde l'image toutes les x microsecondes. Pour ce faire nous utilisons la fonction suivante :

$$\text{ms} = \text{date.microsecond}$$

Cependant, cette fonction nous renvoie une trop grande précision, car elle dispose de 6 chiffres pour décrire les microsecondes. Idéalement, seul les deux premiers chiffres nous sont utiles pour réaliser notre timer. Pour n'avoir que ces deux premiers chiffres nous allons utiliser une simple division ce qui nous permettra d'obtenir un float et de le passer en entier afin de garder uniquement les chiffres qui nous intéressent.

Il ne reste plus qu'à passer ce résultat en paramètre de notre condition d'exécution de la fonction cv2.imwrite() en réalisant un modulo. Ceci revient à la même logique qu'un timer, car la fonction sera exécutée que lorsque le modulo sera nul ce qui correspond à réaliser l'instruction toutes les x microsecondes (où x est le paramètre du modulo).

Le code complet pour la récupération d'image est disponible en annexe.

## 4.2 Traitement d'images

### 4.2.1 Pré-traitement

La première étape du traitement d'images est le pré-traitement qui vise à améliorer les caractéristiques d'une image de manière à ce que le traitement effectué par la suite nous donne les résultats attendus et ce peut importe les conditions d'éclairages, de taille... de l'image passée en entrée. Dans notre cas, deux caractéristiques de l'image nous intéressent : la couleur et les contours. Nous avons donc essayé de réaliser un pré-traitement qui nous permettra de rendre plus net les contours des différents objets de l'image et de faire ressortir leurs couleurs. Pour cela, nous faisons dans un premier temps une correction de la luminosité de l'image en passant par l'image HLS (Teinte, saturation, lumière). Une fois la correction de la luminosité effectuée, nous filtrons l'image de manière à retirer une partie du bruit présent dans l'image tout en essayant de la rendre plus nette. Nous appliquons donc un filtre bilatéral sur notre image. Le filtre bilatéral est définit comme suit[1] :

$$I^{filtered}(x) = \frac{1}{W_p} \sum_{x_i \in \Omega} I(x_i) f_r(||I(x_i) - I(x)||) g_s(||x_i - x||)$$

où le terme de normalisation vaut :

$$W_p = \sum_{x_i \in \Omega} f_r(||I(x_i) - I(x)||) g_s(||x_i - x||)$$

avec :
$I^{filtered}$ l'image filtrée
$I$ l'image d'origine
$x$ les coordonnées du pixel en train d'être filtré
$\Omega$ la fenêtre centrée sur $x$
$f_r$ le noyau pour le lissage en intensité
$g_s$ le noyau pour le lissage en coordonnées

Le filtre bilatéral remplace l'intensité de chaque pixel avec la moyenne pondérée de la valeur des pixels de son entourage ce qui permet de préserver des contours bien définis. Nous nous sommes rendues compte qu'appliquer une nouvelle correction de luminosité après avoir filtré l'image nous donne de meilleurs résultats, nous appliquons donc la même correction que précédemment pour terminer l'étape de pré-traitement.

FIGURE 4.3 – Résultat du pré-traitement

#### Solution alternative

Pour la correction de la luminosité, nous avons pensé à faire une égalisation d'histogramme de manière à avoir une luminosité plus homogène dans l'image et sur les différentes images.

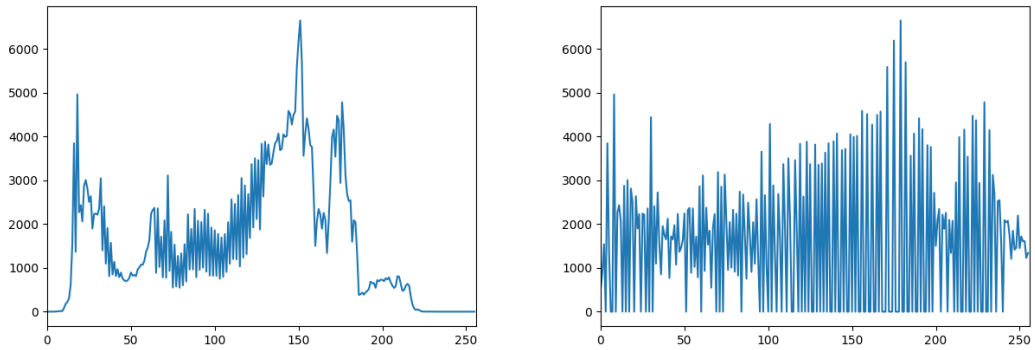


FIGURE 4.4 – Égalisation de la composante L d'une image HLS

Cependant, les résultats obtenus avec cette méthode ne sont pas aussi bon que ce qu'on pourrait espérer : on obtient des couleurs beaucoup moins vives et des contours moins nets ce qui empêche la bonne détection des flèches.

### 4.2.2 Traitement

#### Détection de la flèche

La première étape à réaliser est de détecter si il y a une flèche dans l'image et si oui de l'isoler de manière à pouvoir par la suite identifier sa couleur et sa direction. Pour cela, nous allons nous baser sur les contours des objets présents dans l'image. L'algorithme **Suzuki 85**[4] est appliqué à l'image binaire que l'on obtient en faisant un seuillage de l'image obtenue après le pré-traitement. Cet algorithme parcourt l'image à la recherche d'un pixel satisfaisant la définition d'un contour interne ou d'un contour externe qu'il marque comme point de départ d'un nouveau contour. Les pixels liés à ce pixel et répondant aussi à la définition d'un contour sont marqués avec le même label de manière à obtenir le contour fermé de l'objet. Il en va de même avec les autres objets de l'image chaque nouveau contour correspond à un nouveau label. Cet algorithme nous permet donc de récupérer les contours de tous les objets présents dans l'image une fois ces derniers récupérés, on effectue une approximation polygonale sur chacun d'eux de manière à pouvoir compter le nombre de côtés de chacun des objets. Ainsi, on définit que chaque objet possédant sept côtés est une flèche. On va donc dessiner l'aire de la flèche détectée en blanc sur une image noir de manière à faire un masque qu'on applique ensuite à l'image couleur de manière à obtenir une image ne comportant que la flèche.

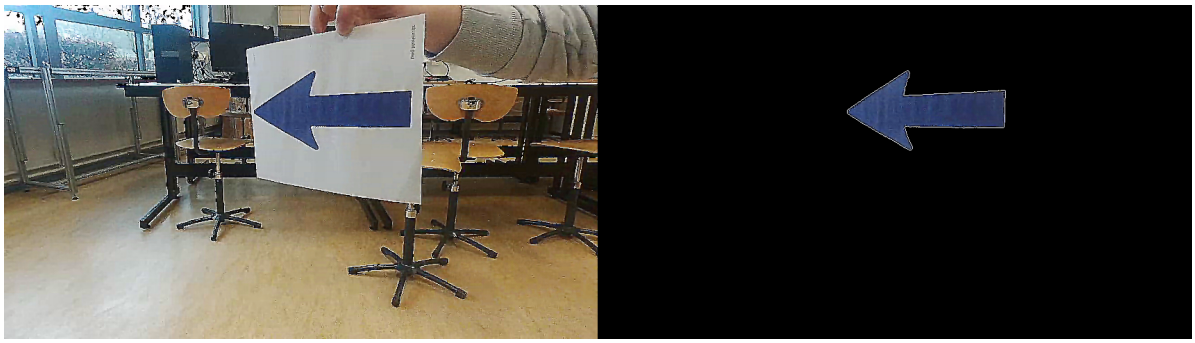

FIGURE 4.5 – Résultat de la détection d'une flèche

Les résultats obtenus sont corrects sur les 80 images possédant une flèche de notre banque de test, cet algorithme ne parvient pas à détecter la flèche dans 11 images. De plus, il est possible qu'une flèche soit détectée dans une image qui n'en contient pas étant donné qu'on base l'algorithme sur le postulat que toutes les formes à sept côtés sont des flèches.

**Détection des couleurs**

Pour pouvoir envoyer une consigne de déplacement au drone, il nous faut connaître la direction dans laquelle il doit se déplacer. Pour cela, nous avons eu une première idée qui consiste à associer une couleur à une direction il nous faudrait donc à partir de l'image de la flèche isolée précédemment effectuer un traitement qui nous permette de récupérer en sortie la couleur de la flèche. Pour nous faciliter la tâche nous avons défini les couleurs suivantes : rouge pur, bleu pur, vert pur et gris soit en format BGR : [0 0 255], [255 0 0], [0 255 0] et [127 127 127] respectivement.

**Image HSV**

La première méthode que nous avons utilisé pour distinguer les différentes couleurs consiste à passer par l'image HSV (Teinte Saturation Valeur) qui est une représentation des couleurs sous forme de cylindre qui prend en compte l'intensité lumineuse des pixels ce qui rend la différentiation des couleurs plus simple. La première étape est donc de convertir nos valeurs RGB en valeur HSV et de faire une fourchette autour de cette valeur de manière à prendre en compte les imperfections de couleur lors de l'impression.

| couleur | HSV min | HSV max |
|---|---|---|
| [0 0 255] | [0 50 50] | [10 255 255] |
| [255 0 0] | [210 50 50] | [230 255 255] |
| [0 255 0] | [110 50 50] | [130 255 255] |

TABLE 4.1 – Tableau des fourchettes permettant de distinguer le rouge, le bleu et le vert en HSV

Pour trouver la couleur de la flèche on parcourt tous les pixels de l'image en faisant un test pour savoir si le pixel appartient à la fourchette voulue, si c'est le cas on le dessine en blanc sur une image noire de manière à obtenir un masque lorsque toute l'image a été parcouru. Nous obtenons en sortie

trois masques correspondants aux composantes rouge, bleue et verte de l'image.
Le gris étant plus compliqué à repérer on soustrait le rouge, le bleu et le vert trouvés à l'aide des fourchettes précédentes à l'image de la flèche seule pour ne conserver que le gris.
Cette solution détecte correctement les différentes couleurs lors de nos premiers tests, cependant, avec de nouvelles flèches imprimées avec une autre imprimante, l'algorithme ne nous donne plus des résultats corrects. Nous avons donc essayé d'agrandir les fourchettes pour détecter les couleurs qui a entraîné une perte de précision : on détecte du gris dans les autres couleurs.

**Image BGR**
Étant donné que notre deuxième test ne nous a pas donné les résultats attendus avec les fourchettes HSV, nous avons décidé d'appliquer le même principe mais cette fois-ci avec des fourchettes BGR. Les valeurs de minimum et maximum sont plus compliquées à trouver : nous avons pris trois images de chaque couleur et pour quelques pixels de chacune notées les valeurs des composantes BGR pour faire nos propres bornes.

| couleur | BGR min | BGR max |
|---|---|---|
| [0 0 255] | [31 50 142] | [59 79 170] |
| [255 0 0] | [81 49 36] | [140 79 65] |
| [0 255 0] | [56 105 51] | [89 130 80] |
| [127 127 127] | [93 93 101] | [121 117 116] |

Table 4.2 – Tableau des fourchettes permettant de distinguer le rouge, le bleu et le vert en BGR

Cette méthode fonctionne correctement, cependant, le calcul des bornes à la main est assez fastidieux et doit être refait à chaque fois que des nouvelles flèches sont imprimées ou qu'il y a un fort changement de luminosité. Il faudrait peut être réaliser un programme de calibration permettant de calculer automatiquement les bornes en prenant quelques images en entrée.

**Poids des composantes**
La dernière idée que nous avons eu pour éviter les problèmes de calibration est de récupérer le pixel correspondant au centre de gravité de la flèche et à partir des valeurs des composantes BGR de ce pixel déterminer la couleur de la flèche.
Cette méthode possède ces limites notamment si le pixel du centre de gravité est bruité. Pour améliorer l'algorithme on pourrait faire la moyenne des composantes BGR de tous les pixels de la flèche et décider à partir de ces valeurs la couleur de la flèche.

**Conclusion**
La détection correcte des couleurs dépend de plusieurs paramètres comme le bruit, la luminosité et la qualité de l'impression. Trouver une technique qui n'est pas influencée par ces paramètres n'est pas chose aisé. Cependant, en ajoutant un programme de calibration, il est possible de faire en sorte que l'algorithme fonctionne dans la plupart des cas. De plus, passé par la moyenne de tous les pixels composant la flèche, plutôt que par un seul pixel dans la méthode de comparaison du poids des différentes composantes pourrait nous permettre de négliger l'effet du bruit.
Pour ces deux méthodes nous obtenons tout de même des résultats exploitables : pour la première 4 flèches sur 69 n'ont pas de couleur détectée et pour la deuxième 9 flèches sur 69 ont une mauvaise couleur détectée.

### Détection de la direction

Étant donné que la détection des couleurs n'étaient pas assez fiable nous avons cherché à détecter la direction de la flèche.

**Horizontal ou vertical**
La première étape a donc été de trouver si la flèche est horizontale ou verticale. Pour cela, on cherche dans un premier temps le plus petit rectangle droit qui entoure la flèche.

FIGURE 4.6 – Rectangles entourant une flèche verticale et une flèche horizontale

La fonction *boundingRect* d'OpenCV nous retourne les points extrêmes situés en haut à gauche et en bas à droite de ce rectangle. A partir de ces points, on récupère les coordonnées des deux autres coins de manière à pouvoir calculer la distance entre les coins haut et les coins de gauche avec la formule suivante :

$$AB = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$$

Si la valeur distance entre les coins supérieurs est plus grande que la distance entre les coins gauches alors la flèche est horizontale et si la distance entre les coins gauches est plus grande que la distance entre les coins supérieurs alors la flèche est verticale. Nous ne nous occupons pas pour le moment des flèches diagonales.

### Trouver la direction de la flèche

**Centre de gravité**
La première méthode à laquelle nous avons pensé pour trouver la direction de la flèche est de comparer la position du centre de gravité de la flèche et la position de celui du rectangle. Le centre de gravité de la flèche devrait être au niveau du début de la pointe et celui du rectangle au milieu de la flèche. Cependant, en pratique les deux centres de gravité sont souvent confondus et donc pas exploitables.

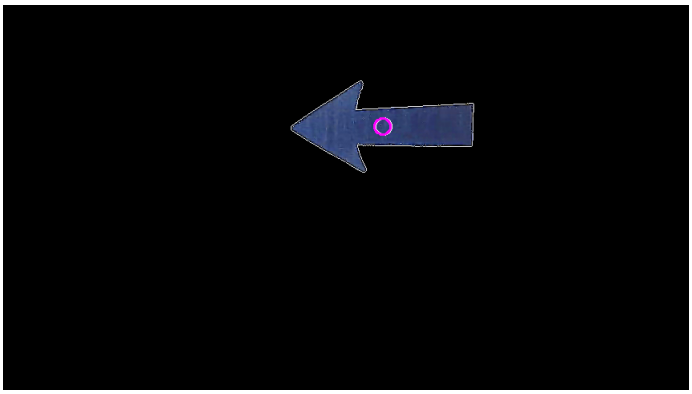

FIGURE 4.7 – Centres de gravité de la flèche et du rectangle

**Points extrêmes**
La deuxième méthode que nous avons mise au point consiste à exploiter les points extrêmes de la flèches que l'on trouve en récupérant les points avec les coordonnées les plus petites et les plus grandes en x et en y dans les points décrivant le contour de la flèche.

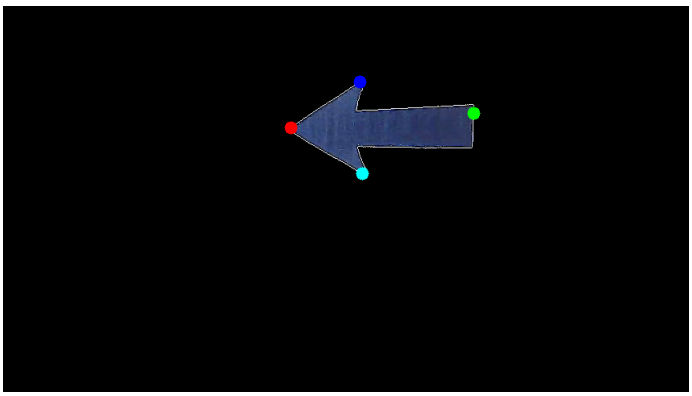

FIGURE 4.8 – Points extrêmes d'une flèche

Une flèche pointe vers la gauche si la flèche est horizontale et que la distance entre le point rouge et le point bleu est plus petite que la distance entre le point bleu et le point vert et inversement pour la droite.
Une flèche pointe vers le haut si la flèche est verticale et que la distance entre le point bleu et le point rouge est plus petite que la distance entre le point rouge et le point cyan et inversement pour le bas.

### 4.2.3 Conclusion

En conclusion, la partie traitement d'images prend une image en entrée, effectue un pré-traitement suivit d'un traitement nous permettant de connaître la direction de la flèche. Pour le moment, la détection de la couleur ne nous donne pas de résultats suffisamment bons pour pouvoir être exploitée. Il nous reste aussi à nous occuper des flèches en diagonales qui pour le moment nous donnent des "faux positifs" il devrait être possible de les considérer comme nul en étudiant l'angle de la *fitted line* qui traverse la flèche de part et d'autre en passant par la pointe.

## 4.3 Envoi de commandes au drone

### 4.3.1 Quel topic utiliser ?

Afin de pouvoir envoyer des commandes au drone, il faut déjà savoir, dans un premier temps, sur quoi nous pouvons agir et publier afin de contrôler le drone. Nous avons déjà connaissance du topic odometry qui nous permet de récupérer les positions linéaires et angulaires du drone. Cela nous est utile pour savoir quel type de données nous devons envoyer pour déplacer le drone. Cependant, le drone ne fait que publier sur ce topic donc ce n'est pas celui ci qui nous permettra de le déplacer en lui envoyant des données.

Le topic qui nous sera utile pour cela, sera cmd_vel. En effet, grâce aux commandes rostopic info et rosmsg show nous avons pu remarquer que le drone souscrivait à ce topic et que celui-ci avait aussi comme données les positions linéaires et angulaires.

```
justine@justine-Latitude:~/bebop_ws/src/projet/src$ rostopic info bebop/cmd_vel
Type: geometry_msgs/Twist

Publishers: None

Subscribers:
 * /bebop/bebop_driver (http://justine-Latitude:33659/)


justine@justine-Latitude:~/bebop_ws/src/projet/src$ rosmsg show Twist
[geometry_msgs/Twist]:
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

FIGURE 4.9 – Données du topic cmd_vel

### 4.3.2 Publier avec ROS via Python

Suite à nos recherches précédentes, nous savons contrôler le drone, mais uniquement en passant par des lignes de commande, notre objectif maintenant est de réaliser la même chose mais en lançant un programme avec comme langage de programmation python.

Pour cela, nous devons savoir comment publier sur un topic dans un programme, car jusqu'à présent nous n'avions que souscrit à des topics. Nous allons alors débuter en essayant d'envoyer la commande afin de faire décoller le drone, le topic correspondant à cette action est takeoff. Suite à diverses recherches nous avons une idée des fonctions à utiliser pour publier sur un topic.

Il s'agit des fonctions suivantes :

— pub = rospy.Publisher('nom_topic', type_msg, queue_size=1)

— pub.publish(type_msg())

Cependant, nous ne pouvons pas utiliser aussi simplement ces fonctions lorsque nous souhaitons publier. En effet, après plusieurs tests, si nous laissons tel quel le code, le programme n'arrivera à réaliser qu'une publication et non pas plusieurs (qui sera notre cas, rien que pour décoller et atterrir). La solution pour répondre à ce problème est d'ajouter une temporisation après la publication. Celle ci peut être instanciée de deux façons différentes :

— En imposant une fréquence de 10Hz avec la fonction rospy.rate(), et en utilisant celle ci avec la fonction ropsy.sleep() dans une boucle
— En ajoutant un paramètre à la fonction rospy.Publisher(), latch=True, celui ci permet de rendre prioritaire la publication.

### 4.3.3 Fonctions de direction modulables selon la vitesse ou la distance

Pour "répondre" au traitement d'images, nous devons être capables de controler le drone selon 4 directions : aller à gauche, à droite, en haut et en bas. Nous allons donc devoir créer une fonction pour chacune de ces directions.

#### Fonctions de direction

De part nos précédents tests, nous savons uniquement que la variable linéaire selon x du topic cmd_vel permet au drone d'aller tout droit.

Nous allons commencer par les directions haut et bas, car en toute logique, en suivant le fait que la variable linéaire selon x permet d'aller tout droit, la variable linéaire selon z permet de se déplacer de haut en bas et de bas en haut, selon la valeur. Toujours en suivant la même logique, on en déduit qu'une valeur positive de la variable linéaire selon z nous permet de monter alors qu'une valeur négative de variable linéaire selon z permet au drone de descendre. Cependant, lorsque l'on souhaite imposer une valeur sur une variable en particulier, il ne faut pas oublier d'imposer une valeur pour toutes les variables linéaires et angulaires, même si celle-ci est nulle.

Ensuite pour les directions droite et gauche, nous allons devoir passer aux variables angulaires et non plus linéaires. Après divers tests, nous remarquons que c'est la variable angulaire selon z dont nous allons avoir besoin pour faire tourner le drone. Comme auparavant, la valeur de cette variable va déterminer si nous pivotons le drone pour aller sur la gauche ou sur la droite. Toujours suite aux tests, nous en concluons qu'une valeur positive de la variable angulaire z permet une rotation sur la gauche alors qu'à l'inverse, une valeur négative permet une rotation sur la droite.

Par contre, si on envoie juste une fois l'instruction au drone, celui-ci se déplacera à peine. Pour tourner de 90 degrés, par exemple, nous avons dû imposer, pour une fréquence de 10Hz, de réaliser 90 fois l'instruction en boucle pour atteindre ce but. De même pour les directions droite et gauche, nous devons faire attention à ne pas changer les variables angulaire selon z et linéaire selon x en même temps, et de bien commencer par celle angulaire pour que dans un premier temps, il tourne complètement, puis seulement après qu'il avance grâce à la variable linéaire.

#### Modularité en vitesse et en distance

Pour ajouter une fonctionnalité supplémentaire, et avoir un programme modulable, nous avons décidé de laisser le choix à l'utilisateur de contrôler le drone en vitesse ou en distance. Cette décision interviendra avant chaque déplacement du drone.

Le choix utilisateur est réalisé grâce à la fonction input() de python, et permet de récupérer une valeur saisie dans le terminal. L'utilisateur aura alors toujours deux questions qui lui seront posées :

— Souhaitez vous controler le drone en vitesse ou en distance ? (vitesse : 1, distance : 2)

  Si la réponse est la vitesse : A quelle vitesse souhaitez vous voler ? (en m/s, entier entre 0 et 9)

  Si la réponse est la distance : Quelle distance souhaitez vous parcourir ? (en m, entier entre 0 et 9)

Nous donnons un intervalle de valeur à respecter, pour des mesures de sécurité du drone, n'ayant jamais pu tester le drone sur de plus grandes étendues de terrain.

Cependant, la valeur de vitesse ou de distance que nous recevons, ne peut pas être utilisée telle quelle, nous devons trouver la relation qui existe entre la valeur de la vitesse ou de la distance demandée, et la valeur à imposer, par conséquent, à la variable linéaire x ou z selon la direction.

Pour la vitesse, nous avons repris les résultats d'un autre binôme également en projet sur le drone. Pour la distance, nous avons fait différentes mesures, en imposant une certaine fréquence. Suite à ça, nous avons pu en déduire un ratio entre la distance demandée et la valeur à imposer.
En ayant ces deux relations, nous avons pu ajouter dans chacune des fonctions direction les conditions pour déterminer si nous étions selon la vitesse et la distance. Puis remplacer par la valeur adéquate selon le ratio, chaque variable linéaire x pour les directions droite et gauche, et z pour haut et bas.

### 4.3.4 Arrêt du programme

Pour plus de sécurité, nous avons fait en sorte que l'utilisateur puisse choisir d'arrêter le drone assez régulièrement. De ce fait, après chaque déplacement nous allons alors demander à l'utilisateur s'il souhaite continuer le contrôle du drone ou non.
Pour ce faire, on lui demande plus précisément :

"Si vous voulez arrêter, appuyez sur e. Pour continuer, appuyez sur n'importe quelle autre touche".

Suite à cette demande, nous récupérons la réponse de l'utilisateur à l'aide la fonction getch(), qui permet de lire un caractère écrit dans le terminal. Cette fonction a d'ailleurs dû être codé par nous même, car inexistante pour le langage python.
Si la touche e est reçue, alors on entame la phase d'atterrissage du drone, sinon on repart dans la boucle et donc on redemande à l'utilisateur quel type de contrôle il souhaite réaliser.

# Conclusion

Pour conclure, nous sommes satisfaites du rendu final de notre projet, car certes il n'est pas parfait, mais reste néanmoins totalement opérationnel. Avec quelques améliorations supplémentaires comme une interface graphique et une meilleure précision dans le contrôle du drone, notre projet peut être réutilisé sans problème dans le cadre d'un autre projet, pour y ajouter une autre fonctionnalité par exemple.

D'un point de vue technique, ce projet assez complet, nous a beaucoup apporté, notamment grâce à l'apprentissage et la maîtrise de la technologie ROS, qui nous était inconnue au départ. Mais aussi en terme de traitement d'images, où nous n'avions que de faibles connaissances dans ce domaine. Et enfin, en programmation, de l'approfondissement et de l'apprentissage concernant le langage de programmation Python.

D'un point de vue gestion de projet, celui-ci nous a permis de travailler en binôme, en apprenant à devoir s'adapter à la façon de travailler de chacun, mais aussi de devoir respecter un certain délai, nous obligeant à travailler en parallèle sur différents domaines du projet. Cela reste une expérience humaine très enrichissante qui nous aidera certainement lors de notre stage ou encore en milieu professionnel une fois diplômé.

# Annexe A

# Récupération d'images à partir de la vidéo

```
#!/usr/bin/env python

# rospy for the subscriber
import rospy
# ROS Image message
from sensor_msgs.msg import Image
# ROS Image message -> OpenCV2 image converter
from cv_bridge import CvBridge, CvBridgeError
# OpenCV2 for saving an image
import cv2
import datetime
from time import sleep

# Instantiate CvBridge
bridge = CvBridge()
def image_callback(msg):
        try:
        # Convert your ROS Image message to OpenCV2
                cv2_img = bridge.imgmsg_to_cv2(msg, "bgr8")
        except CvBridgeError, e:
                print(e)
        else:
        # Save your OpenCV2 image as a jpeg
                date = datetime.datetime.now()
                ms = date.microsecond
                msec = int(ms/10000)
                if msec%4==0 :
                        print("Received an image!")
                        cv2.imwrite(''+str(date)+'.jpeg', cv2_img)

def main():
        rospy.init_node('image_listener')
    # Define your image topic
        image_topic = "bebop/image_raw"
    # Set up your subscriber and define its callback
        rospy.Subscriber(image_topic, Image, image_callback)
    # Spin until ctrl + c
        rospy.spin()
if __name__ == '__main__':
        main()
```

# Bibliographie


[1] BenFrantzDale. Bilateral filter. https://en.wikipedia.org/wiki/Bilateral_filter, 2008.

[2] Lentin Joseph & Jonathan Cacace. *Mastering ROS for Robotics Proramming*. Packt Publishing Limited, 2018.

[3] Parrot. Ardronesdk3 api reference. https://developer.parrot.com/docs/bebop/#general-information, 2O16.

[4] KeiichiA be Satoshi Suzuki. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 1985.

[5] OpenCV team. Opencv library. https://opencv.org/.