Sciences à l'École

C.gENial
Fondation pour la culture scientifique et technique

COMMENT GUIDER UN ROBOT HORS D'UN LABYRINTHE
A L'AIDE D'UN CAPTEUR DÉPORTÉ ?

Dans le monde, de nombreux pays sont frappés par les catastrophes naturelles et nombreuses sont les victimes. Les secouristes doivent patrouiller sur terre et dans les cieux pour venir en aide aux survivants. La rapidité d'action est un des critères les plus importants pour sauver le plus de monde possible. L'automatisation de l'analyse de terrain et de l'action des équipes au sol permettrait d'augmenter cette rapidité et, ainsi, de sauver de nombreuses vies. Nous avons essayé, durant deux années, de simuler à l'échelle d'un labyrinthe cet automatisme.

A l'aide d'un drone, d'un robot, d'un ordinateur, d'un labyrinthe mais surtout d'une équipe motivée, nous nous sommes lancés dans cette aventure.

Thomas Agugliaro / Florian Begin / Gaëtan Bessot / Nathaël Chery / Samuel Gaiffe
Lycée Hélène-Boucher de THIONVILLE | Année 2015-2016

Table des matières

Introduction :

En 2011, un séisme de magnitude 9 a ravagé le Nord-Est Japon, provoquant plus de 16 000 morts, L'automatisation de l'analyse de terrain et de l'action des équipes au sol permettrait de détecter les survivants et de sauver de nombreuses vies. Nous avons tenté de créer cet automatisme à l'échelle d'un labyrinthe.

1. Les partenaires

- La formation de nos professeurs à l'INRIA de Nancy leur a permis de nous guider sur les notions nécessaires aux développement de notre projet. Ils ont aussi fait appel au soutien d'autres professeurs tels que Manu Dross et Stéphane Wolter.

- Voici une courte description des élèves ayant participé à ce projet :
  Thomas **AGUGLIARO**, élève de 1e S, passionné par l'informatique et l'algorithmique ;
  Florian **BEGIN**, élève de 1e S qui a apporté une précieuse aide concernant la création de l'algorithme d'asservissement de position du drone ;
  Gaëtan **BESSOT**, élève de 1e S lui aussi, passionné par la robotique et les sciences techniques;
  Nathaël **CHERY**, élève de 1e S, toujours prêt à expérimenter de nouveaux programmes Python ;
  Samuel **GAIFFE**, élève de 1e S, il adore les sciences et est toujours prêt à nous rendre service.

1. 1. Le labyrinthe

- Commençons par la présentation du labyrinthe réalisé et expliquons nos choix dans l'aboutissement de cette pièce. Il est important de préciser que notre labyrinthe est réalisé en deux dimensions , il ne possède donc aucune paroi.

- Pourquoi un tel choix ? Car pour analyser le labyrinthe, nous prenons une photo vue de dessus, des murs ou des parois ne seraient pas forcément détectables. Des cases de couleurs étaient ainsi beaucoup plus adaptées.

- Nous avons choisi les trois couleurs primaires pour constituer les cases de notre labyrinthe car celles-ci sont facilement détectables. Le vert fut défini comme étant la couleur des obstacles, le bleu, la case de départ et le rouge, celle d'arrivée.

- Il était important pour nous d'avoir un labyrinthe modulable selon nos envies mais aussi facilement transportable d'une salle à l'autre. Nous avons donc opté pour un labyrinthe constitué de quatre plaques de bois blanc, chacune quadrillée en 9 cases.

- Pour la taille de ces cases, nous nous sommes basés sur les dimensions de notre robot, à savoir 20 cm par 20 cm, nous voulions laisser une certaine marge à celui-ci durant ses rotations, les cases mesuraient donc 30cm par 30cm.

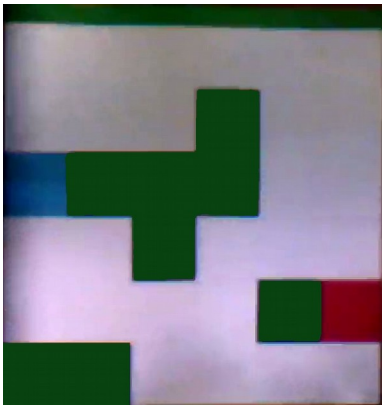

Figure 1 : Le labyrinthe vu de haut.

1.2. Le drone et le robot

- Les activités de l'atelier scientifique de Hélène-Boucher à Thionville furent souvent en lien avec l'informatique. Et c'est au début de l'année scolaire 2014-2015 que les professeurs encadrant nous ont proposé le projet.

- Comment guider un robot hors d'un labyrinthe à l'aide d'un capteur déporté ? Cette problématique annonçait déjà la future formation des élèves en programmation.

- Le choix du robot s'orienta vers un kit Lego Mindstorm car celui-ci, en plus d'être maîtrisé par nos professeurs, était utilisé dans le cadre de la spécialité ISN enseignée en Terminale S.

Figure 2 : Le robot Lego.

- Un drone fut choisi pour servir de capteur déporté et notre choix s'arrêta sur l'AR.Drone 2.0 de Parrot de par la présence d'une caméra verticale sur celui-ci et de la disponibilité d'une documentation complète et officielle.

Figure 3 : Le drone Parrot

2. Le programme

- Le programme se décompose en 3 parties reliées à une interface centrale. Un sous-programme sert de relais entre le drone et l'ordinateur, un autre entre le robot et l'ordinateur, et un dernier analyse les images envoyées par le drone.

- Ces parties sont liées entre elles par un serveur inter-programmes qui fait circuler les informations d'un sous programme à l'autre, remplaçant le système d'écriture-lecture d'un fichier qui était trop lent.

2.1 Le serveur

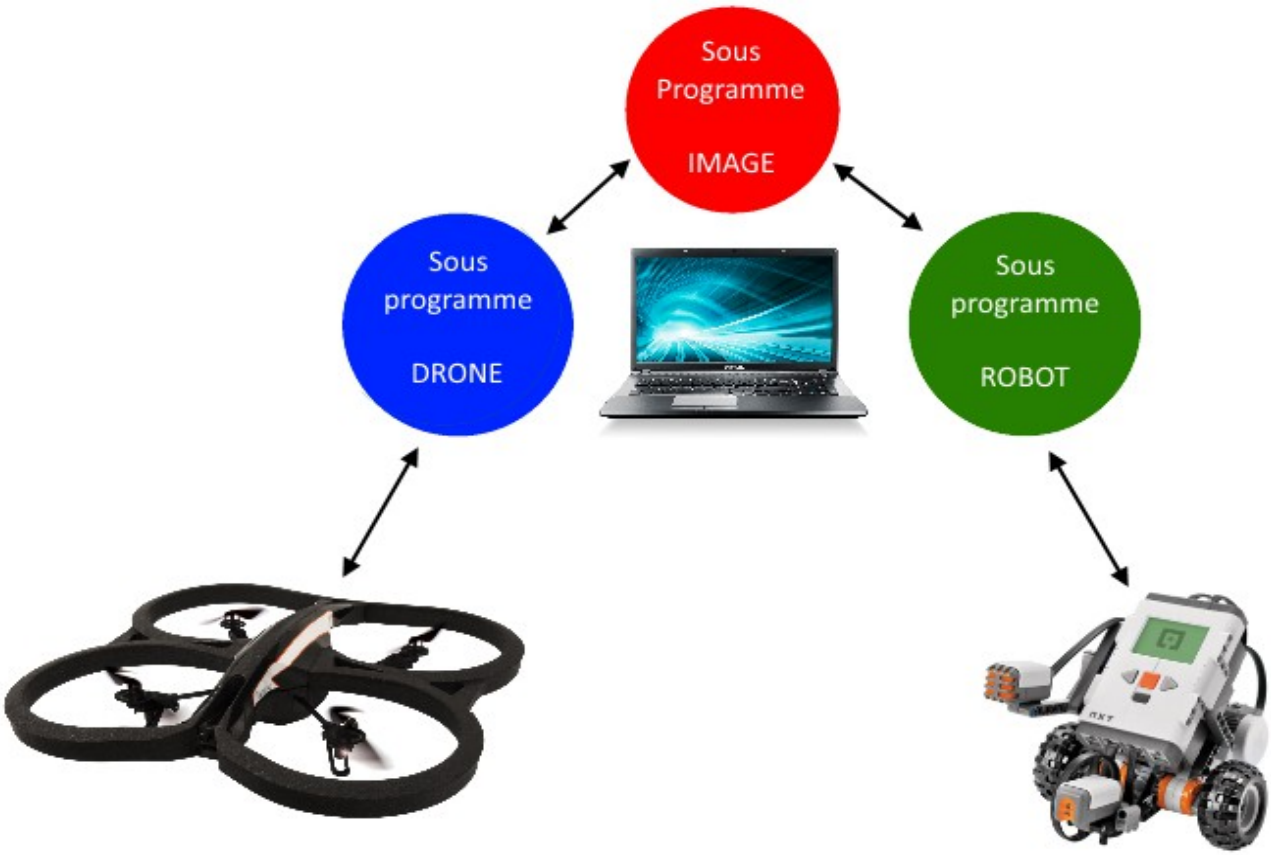

Figure 4 : Interactions des sous-programmes de notre projet

- Chaque programme utilise deux valeurs : la commande et l'unité. Une unité est une valeur définissant le mouvement du drone ou du robot, que celui ci soit un mouvement linéaire ou un mouvement de rotation. Ces informations sont enregistrées dans un octet avec la formule : $Octet = 4 \times Unité + Commande$. Cet octet est ensuite décomposé à l'arrivée grâce à une division euclidienne et à un modulo.

2.2 Les organigrammes

- Tout d'abord, voici l'organigramme de la hiérarchie du programme :

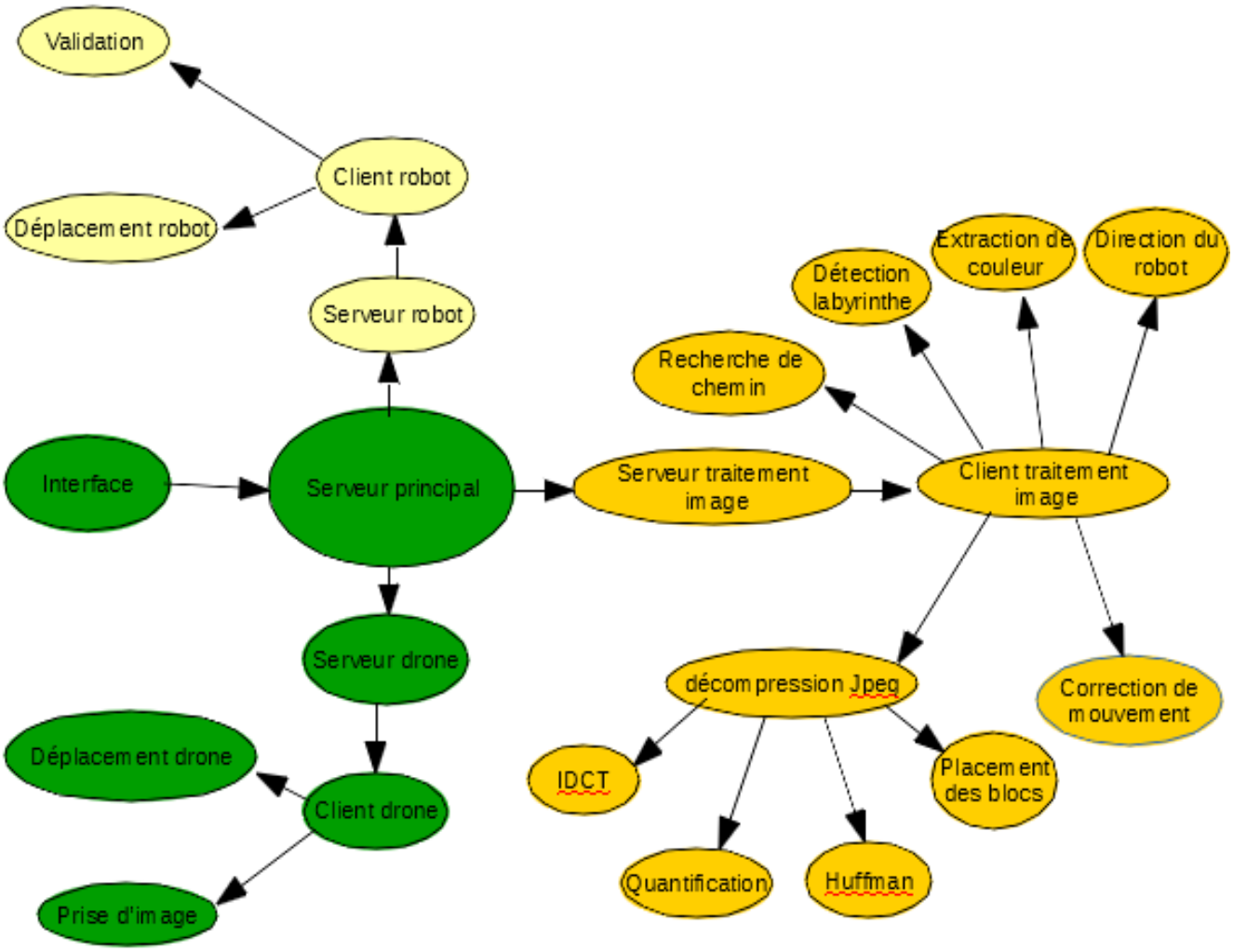

Figure 5 : Hiérarchie du programme dans notre projet.

- Dans ce schéma, plus une brique est loin du programme « Interface », plus sa hiérarchie sera basse. Une brique **verte** signifie que le programme est fait en C, une brique **orange** signifie que le programme est fait en Python 3 et une brique en **beige** signifie que le programme est fait en Python 2.

- Dans un second organigramme, voyons comment les informations circulent dans notre projet :

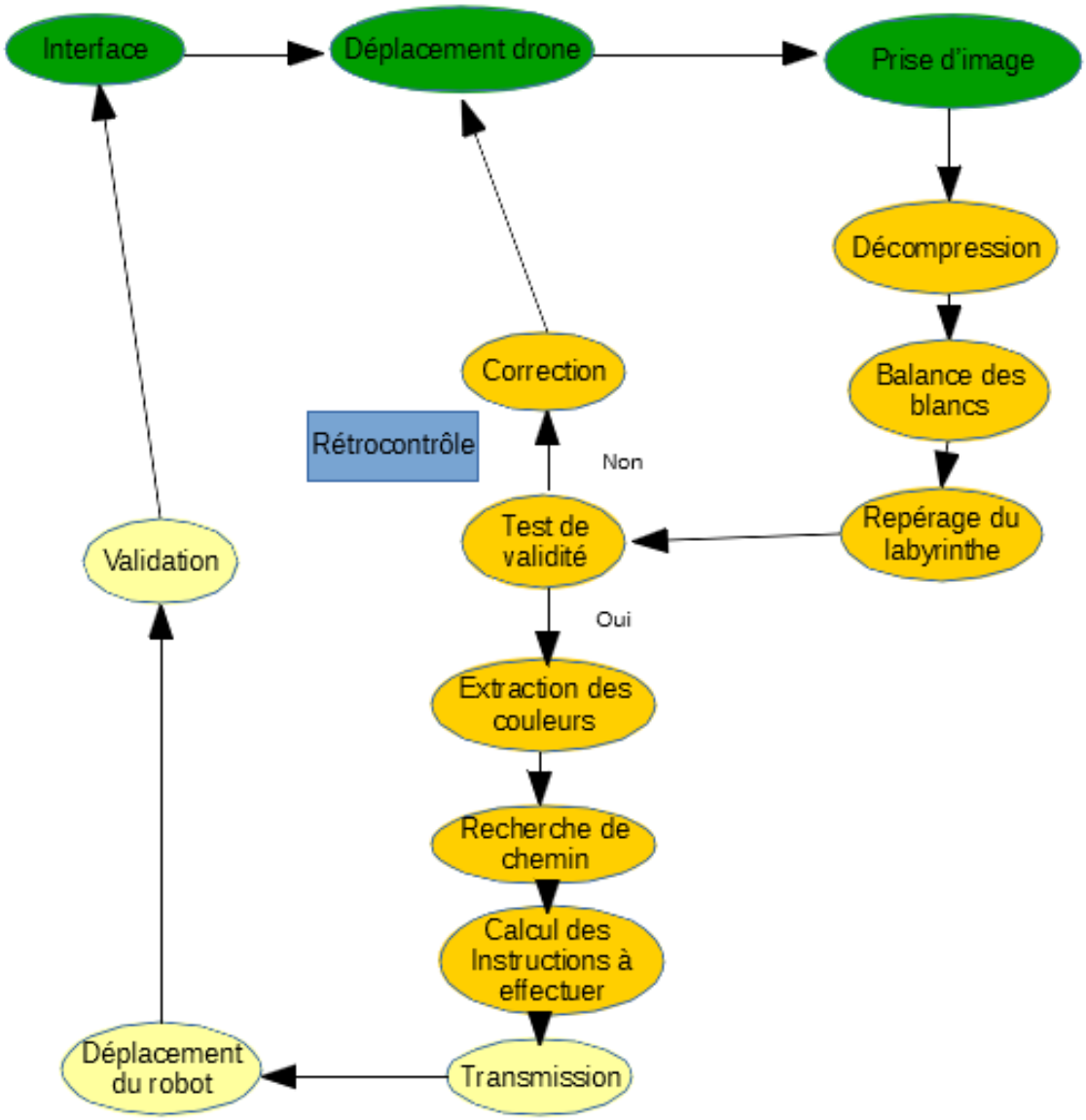

Figure 6 : Cheminement de l'information entre nos programmes.

- Nous pouvons voir que le départ de l'information se trouve encore dans le programme « Interface », car l'utilisateur doit garder le contrôle de la progression de l'enchaînement des parties.

3. Le drone

- Notre but est de prendre une photographie aérienne du labyrinthe telle que le programme de traitement d'image puisse l'analyser. La caméra doit donc pouvoir se déplacer dans toutes les directions pour obtenir un cadrage parfait. Le drone est ainsi piloté grâce à la librairie officielle en « C » de chez Parrot.

3.1. L'asservissement de position

- Parrot recommande la personnalisation de « sdk_demo » afin d'avoir un programme fonctionnel plus simplement. Celui-ci affiche en temps réel la position et l'inclinaison du drone et nous y avons intégré une fonction qui envoie au drone nos commandes. La fonction principale maîtrise les déplacements du drone :

  - ardrone_tool_set_progressive_cmd(*a*, *b*, *c*, *d*, *e*, *f*, *g*) ; *a* active ou non le « combined yaw » qui utilise l'inclinaison gauche-droite du drone et sa rotation sur lui même pour obtenir une trajectoire plus fluide, nous mettons cette valeur à 1 pour l'activer. *b* et *c* permettent de définir respectivement l'inclinaison avant-arrière et gauche-droite du drone. *d* est la hauteur que prend le drone et *e*, la rotation qu'il effectue sur lui même. *f* et *g* sont les paramètres la boussole du drone mais nous ne l'utilisons pas dans notre programme.

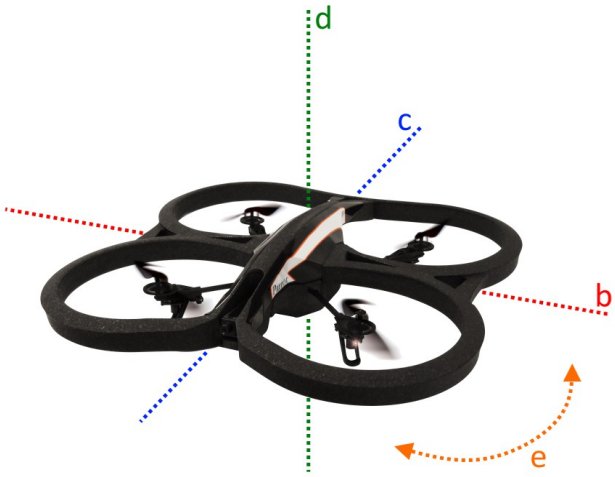

Figure 7 : Axes de déplacement et d'inclinaison du drone

  - Quand le drone décolle, il commence l'envoi de photographies à l'ordinateur. Celui-ci, renvoie au drone deux cordonnées, *x* et *y*, à suivre pour se rapprocher de plus en plus du labyrinthe afin d'obtenir un cadrage idéal.

  - La coordonnée x correspond à un déplacement gauche-droite, faisant varier le temps où le paramètre c incline le drone.

  - La coordonnée y correspond à un déplacement avant-arrière, utilisant le même procédé.

3.2. La prise d'image

- Une vidéo est composée de plusieurs images, appelée **frames**, constituée de nombreux bits définissant diverses valeurs. La communication entre le drone et l'ordinateur s'effectue en WIFI, les frames sont donc envoyées via des **sockets** (permettant des accès à des valeurs en dehors du programme). Chacune d'elles commence par un **header** qui donne plusieurs informations sur l'image. Le reste de la frame contient l'image encodée.

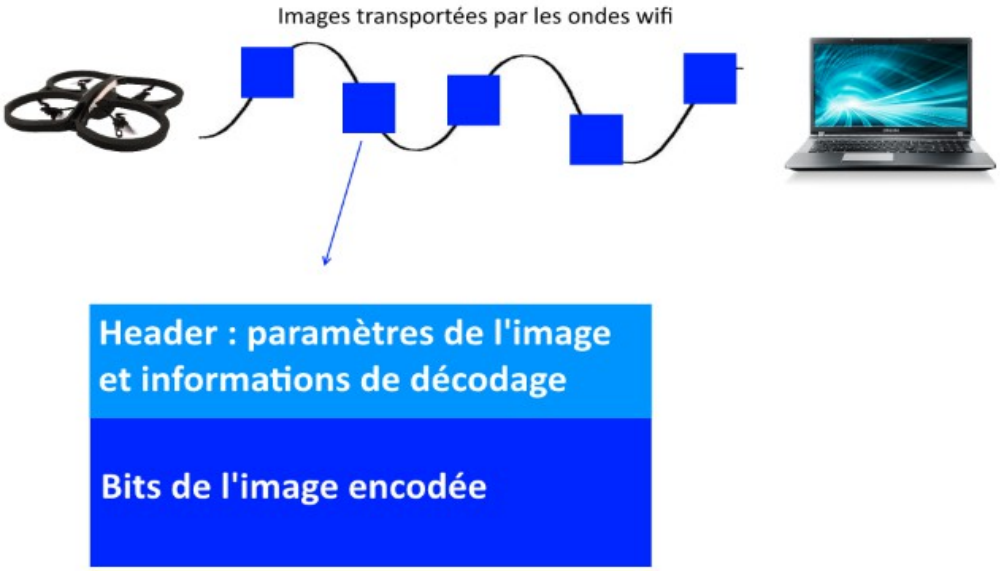

Figure 8: Transmission d'une image encodée

- Le sous-programme du drone reçoit l'information et la renvoie au programme traitement image. Deux moyens sont envisageables: nous pouvons envoyer l'information encodée ou enregistrer l'image grâce à un système de décodage automatique.

- Dans le premier cas, un problème se pose, nous ne pouvons pas traiter l'information d'un seul bloc. Il est donc nécessaire d'enregistrer les informations petit à petit à l'aide d'un **buffer**, qui est comparable à notre mémoire à court terme.

- Dans le second cas, la réception de la frame est très simple si nous utilisons la bibliothèque OpenCV. En effet, la fonction « VideoCapture » lit le flux vidéo du drone et il ne reste plus qu'à utiliser la fonction « read » pour isoler une frame. Nous pouvons ensuite l'enregistrer grâce à « imwrite ». Dans les deux cas, l'information est enregistrée.

- Le programme traitement image peut ainsi utiliser l'image enregistrée pour guider le drone ou pour le calcul de la trajectoire qui sera envoyé au robot.

4. Le traitement de l'image

4.1. La conversion de l'image

- Avant de pouvoir extraire des informations de l'image, notre programme doit comprendre celle-ci. Or, les données sont compressés, étudions ce mécanisme appelé "jpeg » pour le remonter.

- D'abord, les composantes utilisées(RGB) sont changées en YCbCr, ce qui permet de séparer l'information de la luminance(Y est l'image en nuance de gris) des informations de chrominance(Cb et Cr sont les couleurs de l'image) et nous pouvons nous permettre de perdre plus de qualité sur les informations de chrominance sans que l'œil humain ne perçoive de différence. Voici les formules utilisées :

$$Y = 0.299 * R + 0.587 * G + 0.114 * B$$
$$Cb = -0.1687 * R - 0.3313 * G + 0.5 * B + 128$$
$$Cr = 0.5 * R - 0.4187 * G - 0.0813 * B + 128$$

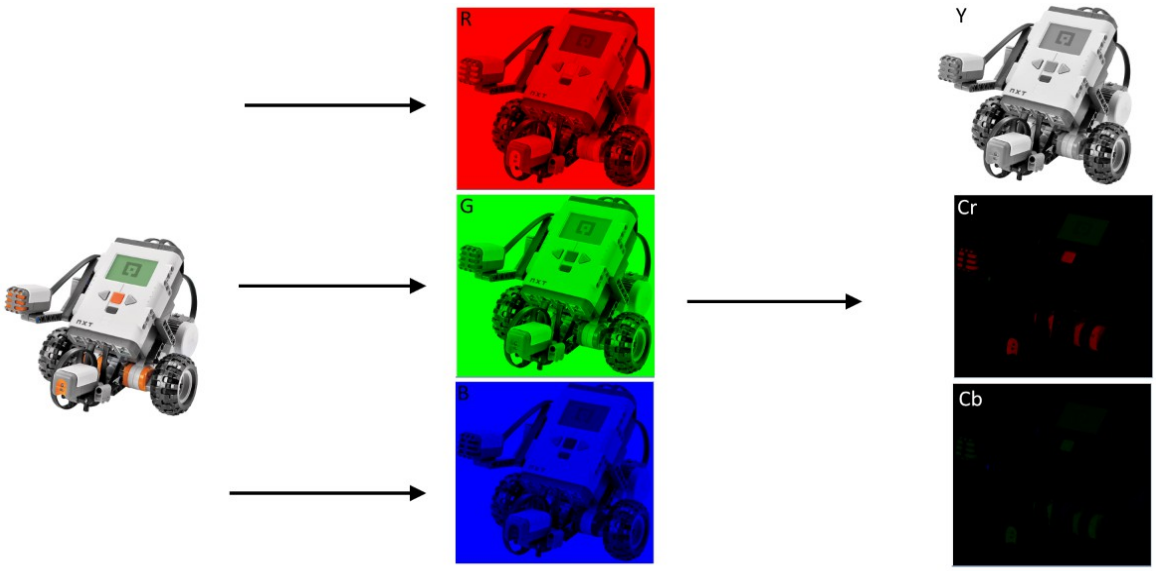

Figure 9 : Représentation en RGB et YCbCr d'une image.

- Pour économiser de la place, seule la moyenne de quelques pixels consécutifs va être sauvegardée et non tous ces pixels, et ce, d'une manière horizontale et verticale. La quantité d'information à sauvegarder est ainsi divisée par le produit du nombre de cases de la compression verticale et horizontale.

- L'image est alors découpée en blocs de 8 pixels sur 8 pixels.

- Ensuite, une formule mathématique est appliquée aux matrices 8 par 8 précédemment créées:

$$\mathrm{DCT}(i,j) = \frac{2}{N} C(i) C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \mathrm{pixel}(x,y) \cos\left[\frac{(2x+1)i\pi}{2N}\right] \cos\left[\frac{(2y+1)j\pi}{2N}\right]$$

- Cette formule, appelée la DCT(« Discrete Cosine Transform ») va transformer les matrices 8 par 8 contenant les amplitudes de l'image en matrices 8 par 8 contenant les fréquences.

- Chaque composante de la matrice obtenue va être divisée(puis arrondie à l'unité) par un nombre réel, souvent faible pour les petites coordonnées (Basses fréquences) et plus grand pour les grandes coordonnées (Hautes fréquences) car l'œil humain est moins sensible à celles-ci. Cette étape est la quantification.

- Ensuite, comme tous les nombres de la matrice ont été divisés et arrondis, il y aura beaucoup de répétitions de nombres faibles. Une compression RLE (Run-Lenght Encoding) est alors effectuée, et celle-ci dans un ordre bien précis car les 0 s'enchaîneront de plus en plus dans le coin en bas à droite, grâce à quoi, le codage RLE sera beaucoup plus efficace.

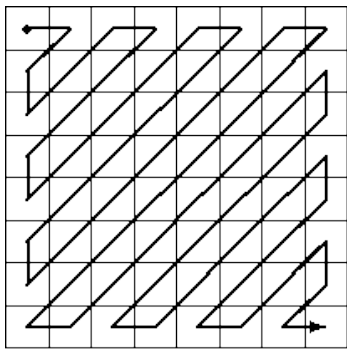

Figure 10 : Parcours d'un bloc 8 par 8 en « zig-zag »

- Ce codage consiste à remplacer une chaîne consécutive de caractères par le nombre de caractères puis le caractère, et un caractère spécial est ajouté, quand il n'y a plus de nombres autres que 0.

- Un codage fréquentiel est ensuite réalisé, c'est le codage de Huffman, qui va faire correspondre des plus petits codes binaires aux informations les plus présentes dans l'image. Par exemple si le nombre 2 est très présent, il aura une représentation ayant moins de coût que, par exemple le nombre 5 si celui-ci n'est pas beaucoup présent.

- La décompression est alors très simple lorsque nous avons compris la compression, en effet, il suffit de suivre les étapes dans l'ordre inverse: ouverture du fichier ".jpeg" décompression de Huffman, décompression RLE, quantification inverse, application de la formule DCT inverse et recomposition de l'image depuis les blocs 8 par 8.

- Une fois l'image obtenue, nous prendrons soin de lui appliquer une balance des blancs, pour permettre à la suite du programme de reconnaître les couleurs des cases du labyrinthe quelque soit l'éclairage. Pour effectuer cette balance des blancs, nous cherchons dans l'image les valeurs maximum de Rouge, de Vert et de Bleu, puis nous multiplions chaque composante par $\frac{C_{max}}{255}$ où C_{max} est la valeur maximale d'une couleur.

Figure 11 : Exemple d'une image avant et après une balance.

4.2. L'extraction de l'information

- Une fois l'image obtenue et la balance des blancs appliquée, nous allons essayer de détecter le labyrinthe contenu dans cette image. Pour ce faire, le programme va parcourir l'image selon un schéma Haut/Bas puis Bas/Haut en sauvegardant les coordonnées du premier pixel appartenant au labyrinthe pour chaque abscisse de l'image.

- A l'aide de la structure de données obtenue, on sélectionne les points les plus proches de chaque coin de l'image et on obtient les sommets du rectangle représentant le labyrinthe.

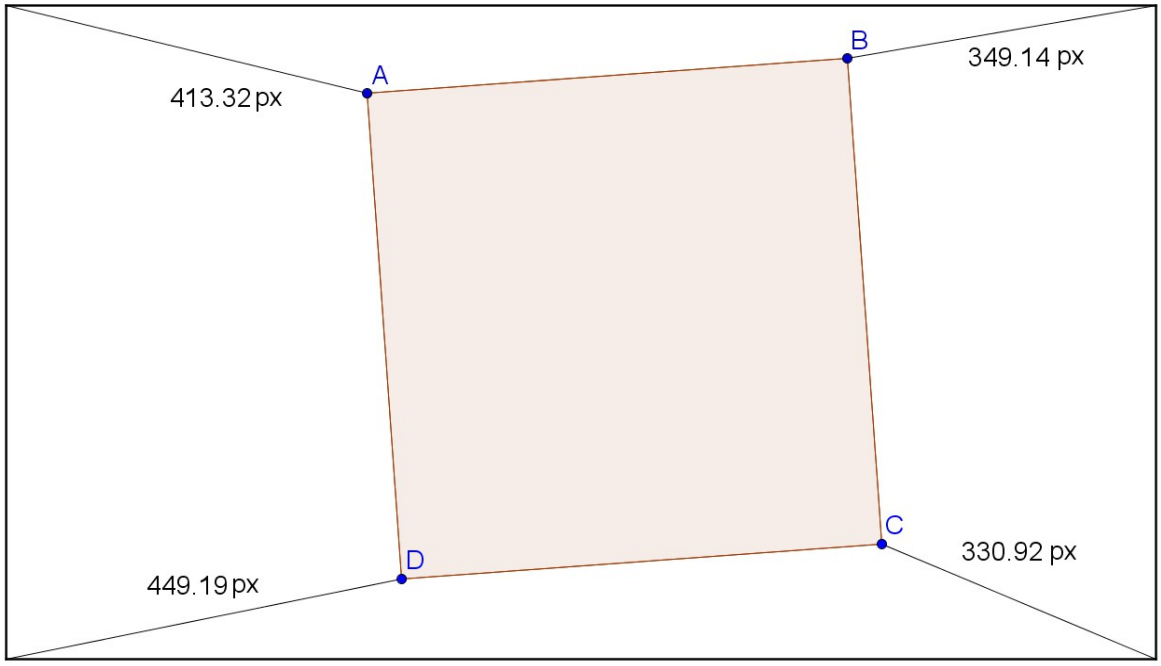

Figure 12 : Exemple du positionnement d'un labyrinthe dans l'image.

- Maintenant, dans le repère $(\vec{AB}, \vec{AD})$, nous pouvons représenter la position de chacune des cases du labyrinthe par la formule $\vec{AP} = \frac{x}{6} * \vec{AB} + \frac{y}{6} * \vec{AD}$ où P est le point au centre de la case, x est son abscisse dans le labyrinthe et y est son ordonnée dans le labyrinthe et la division par 6 est appliquée car le labyrinthe a une taille de 6 par 6. Ce qui permet l'asservissement de position.

- Nous pouvons maintenant extraire les couleurs de chaque case du labyrinthe, que nous allons stocker dans un tableau de taille 6 par 6.

- Maintenant, il nous suffit de trouver le chemin le plus court entre le départ et l'arrivée. Pour ce faire, nous utilisons l'algorithme A*. Son implémentation est gérée à l'aide d'une file d'attente des cases à parcourir triée à l'aide d'une fonction heuristique, dans notre cas, c'est la distance euclidienne car elle peut donner une bonne représentation du chemin le plus court dans la majorité des cas.

- Cet algorithme nous retourne la liste des cases consécutives à parcourir, mais le robot attend une autre information: les actions à effectuer (avancer, tourner à gauche ou à droite).

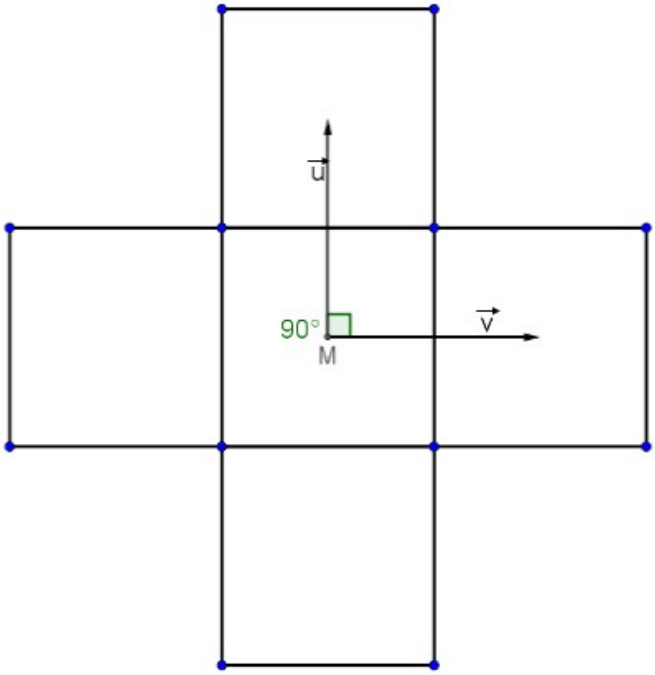

Figure 13 : Représentation vectorielle

- Soit $\vec{u}$ le vecteur représentant le sens et la direction du robot avant le déplacement, et $\vec{v}$ le vecteur représentant le sens et la direction du robot après le déplacement souhaité. Nous pouvons calculer les coordonnées de ces vecteurs: $\vec{u}\begin{pmatrix}0\\-1\end{pmatrix}$ (dans le tableau, l'axe des y est dirigé "vers le bas" donc comme le vecteur se dirige "vers le haut", il aura une ordonnée négative) et $\vec{v}\begin{pmatrix}1\\0\end{pmatrix}$. Nous avons donc cherché à calculer l'angle entre ces deux vecteurs et nous avons simplifié la formule en sachant qu'il n'y avait que 3 angles possibles.

- Il faut maintenant envoyer les informations au programme gérant le robot. Pour cela, nous utilisons encore le serveur inter-programmes que nous avons créé. Nous avons donc choisi un protocole pour envoyer ces informations sous la forme d'un seul octet .

5. Le robot

5.1. Les mouvements du robot

- Le premier langage de programmation utilisé était CROBOT, mais il fut rapidement remplacé par Python car celui-ci était plus complet. Divers éléments sont connectés au cerveau du robot.

- Nous avons bien entendu deux moteurs pour les déplacements du robot, au début du projet, nous les faisions tourner avec la fonction « run(vitesse) » qui enclenche la rotation des roues à une certaine vitesse ainsi que la fonction « brake() » qui arrête la rotation des roues. Nous avons ensuite gagné en précision avec la fonction « turn(vitesse, angle) » qui fait tourner le moteur à une certaine vitesse d'un angle de rotation que nous pouvons choisir. Pour cela, la fonction fait appel au codeur incrémental.

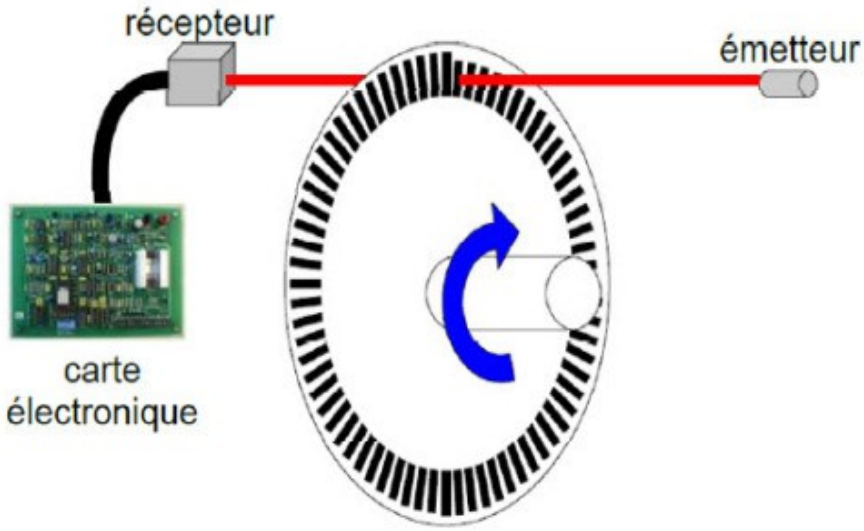

Figure 14: Le codeur incrémental.

- Pour calculer l'angle à utiliser en fonction de la distance, nous utilisons la formule suivante : $angle = 360 \times \frac{distance}{Périmètre\ De\ La\ Roue}$ notre labyrinthe étant constitué de case de 30 cm par 30 cm, nous devons faire avancer le robot de 30 cm pour que celui-ci passe du centre d'une case à l'autre. Le périmètre de la roue est de 17,6 cm. L'angle est donc de $360 \times \frac{30}{17,6} = 614$ (l'angle est arrondi à l'entier supérieur).

- Nous avons aussi utilisé la boussole magnétique pour faire tourner notre robot. Nous utilisons la fonction « run(vitesse)», précédemment présentée, pour faire tourner le robot sur lui même (une roue allant dans une direction et l'autre dans la direction opposée) quand le robot a tourné comme nous le voulons, la rotation s'arrête.

5.2. La configuration et les instructions

- Un simple programme de « question – réponse » programmé par nos soins permet la configuration du robot. Les propriétés du robot (roues, boussole .. etc) sont enregistrées dans un fichier texte, séparant les différentes variables par un retour à la ligne.

- Le sous-programme gérant le robot ouvre le fichier texte ainsi créé et remplace les valeurs utilisées dans le code en fonction des propriétés du robot.

- Une fois le robot configuré, celui-ci doit recevoir les commandes du programme traitement image. Ces informations transitent à travers le serveur inter-programmes. Le fonctionnement est le même que celui du client drone. Un octet est envoyé et est séparé en deux à l'arrivée : le numéro de la fonction et l'argument de celle-ci. Nous faisons ensuite appel à la fonction « executer » qui envoie au robot la fonction à effectuer et l'argument de celle-ci.

- Ainsi, le robot effectue la commande envoyée par le programme traitement image. L'opération est répétée et le robot finit par atteindre la case rouge, objectif accompli.

6. Conclusion :

- Grâce à l'union de ces trois parties, notre projet est enfin fonctionnel, le drone vole au dessus du labyrinthe, prend celui-ci en photo et envoie cette image à l'ordinateur qui en extrait, après décompression et analyse, une trajectoire que le robot effectue.

- Plusieurs points peuvent être améliorés dans le projet. L'utilisation d'un robot Lego apporte, en contre-partie de sa simplicité, une imprécision relative qui pourrait être corrigée via l'utilisation d'un robot plus complexe.

- Le terrain observé est en deux dimensions et ne comporte aucun relief, chose peu représentative de la réalité. De plus, il pourrait être intéressant de se renseigner sur la nature du sol : eau, pierre, végétation .. etc dans le but d'équiper le robot qui partira sur le terrain.

- Le drone comporte aussi quelques imperfections, comme son autonomie trop faible pour une véritable action sur le terrain, mais aussi la variabilité de ses vols car aucun moyen de calcul précis de ses déplacements n'était à notre disposition.

- Ce projet est une grande aventure riche en collaboration, qui a vu exploser une véritable dynamique autour du laboratoire de physique-chimie. Ces salles sont devenues, durant un temps, le spectacle de nombreuses réussites mais aussi de nombreux échecs.

- Merci à Jérôme Metzler, professeur de physique-chimie et d'ISN, pour avoir créer l'atelier scientifique, pour l'encadrement et le suivi de notre travail et pour les nombreux conseils qu'il nous a apportés durant ces années.

- Merci à Pascal Pierre, professeur de physique-chimie et d'ISN, pour l'aide qu'il nous a apportée durant notre initiation à la programmation mais aussi pour les nombreuses pistes qu'il nous a fait découvrir.

7. Ressources :

Bibliothèque de Bastian Venthur
https://github.com/venthur/python-ardrone

Bibliothèque officielle de Parrot
http://developer.parrot.com/ar-drone.html

Balance des blancs
https://fr.wikipedia.org/wiki/Balance_des_blancs

Le format JPEG
https://fr.wikipedia.org/wiki/JPEG

JPEG, idée et pratique
https://en.wikibooks.org/wiki/JPEG_-_Idea_and_Practice

Codage de Huffman
https://fr.wikipedia.org/wiki/Codage_de_Huffman

Transformée en cosinus discrète
https://fr.wikipedia.org/wiki/Transform%C3%A9e_en_cosinus_discr%C3%A8te

YcbCr
https://fr.wikipedia.org/wiki/YCbCr

Algorithme de recherche de chemins
https://fr.wikipedia.org/wiki/Algorithme_A*

Bibliothèque NXT-PYTHON
https://github.com/Eelviny/nxt-python