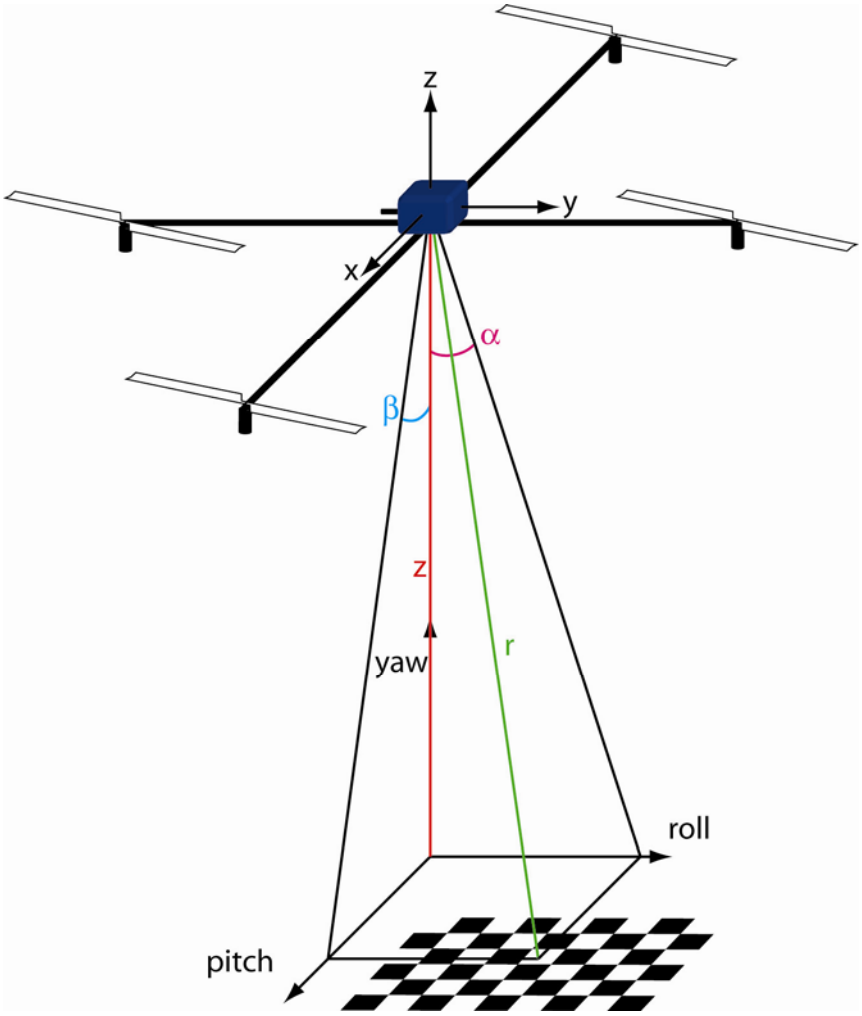
EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE LAUSANNE
POLITECNICO FEDERALE LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY

FACULTÉ DES SCIENCES ET TECHNIQUES DE L'INGÉNIEUR
SECTION DE MICROTECHNIQUE
LABORATOIRES DES SYSTÈMES AUTONOMES / ÉTÉ 04

Professeur: R. Siegwart
Assistant 1 : S. Bouabdallah
Assistant 2 : P. Lamon

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Projet de diplôme



Positionnement de robots volants par la vision

Thierry Frank – février 2005

EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE LAUSANNE
POLITECNICO FEDERALE DI LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY LAUSANNE

FACULTE SCIENCES ET TECHNIQUES DE L'INGENIEUR

LABORATOIRE DE SYSTEMES AUTONOMES
CH - 1015 LAUSANNE

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

PROJET DE DIPLOME – HIVER 2004/2005

Titre:	Flying Robot Positionning Using Vision		
Candidat:	Thierry Frank	**Section:**	MT
Professeur:	Roland Siegwart		
Assistant 1:	Samir Bouabdallah	**Assistant 2:**	Pierre Lamon

Donnée & travail demandé :

Dans le cadre d'un projet de thèse, on vise à concevoir et à contrôler un hélicoptère a quatre rotors. Afin de contrôler le robot, nous avons besoin de mesurer/extraire sa position relative ou absolue. Le but du présent projet est de permettre un positionnement à l'aide de la vision.
Le travail à effectuer est donc :

- Préparer l'électronique nécessaire au projet (montage électronique).
- Faire un choix final de l'approche à adopter (mono ou stéréo vision).
- Implémenter et tester les algorithmes.

Les différents choix doivent être pris en tenant compte du fait que la puissance de calcul et la masse utile sont limitées.

Remarques :

Un plan de travail sera établi et présenté aux assistants avant le 1er novembre 2004.

Une présentation intermédiaire (environ 10 minutes de présentation et 10 minutes de discussion) de votre travail aura lieu dans le courant du mois de décembre 2004. Elle a pour objectifs de donner un rapide résumé du travail déjà effectué, de proposer un plan précis pour la suite du projet et d'en discuter les options principales.

Un rapport, comprenant en son début l'énoncé du travail (présent document), suivi d'un résumé d'une page (selon canevas), sera remis le vendredi 18 février 2005 en 4 exemplaires au secrétariat de votre section. L'accent sera mis sur les expériences et les résultats obtenus. Le public cible est de type ingénieur EPF sans connaissance pointue du domaine. Une version préliminaire du rapport sera remise à l'assistant le 11 février 2005. Tous les documents en version informatique, y compris le rapport (en version source et en version pdf), le document de la présentation orale et un résumé au format pdf, ainsi que les sources des différents programmes doivent être gravé sur un CD-ROM et remis à l'assistant au plus tard lors de la défense finale.

Une défense de 40 minutes (environ 20 minutes de présentation et démonstration, plus 20 minutes de réponses aux questions) aura lieu dans la période du 7 au 14 mars 2005.

Le professeur responsable:	L'assistant responsable:
Signature : R. Siegwart	Signature :
Roland Siegwart	Samir Bouabdallah

Lausanne, le 18 octobre 2004

Flying robot positioning using vision

Thierry Frank, microengineering

Assistant 1: Samir Bouabdallah
Assistant 2 : Pierre Lamon
Professor: Roland Siegwart

This project is about developing an odometry system based on vision for flying robots.
The system is implemented on the existing OS4 helicopter, which have an on-board computer Geode SC1200 @266MHz

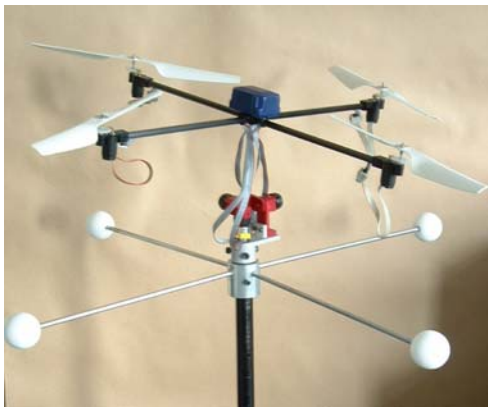


Fig. 1: OS4 helicopter

The system uses a small camera interface developed by the ASL to get the images. The interface to the camera is done via USB.
An inertial measurement unit (IMU) MT9-B from X-sens is used to get these angles (pitch and roll) but the yaw is very unreliable in indoor environments (magnetic fields, metallic structures, etc. so it is calculated with image processing.

The calculating power is limited on the helicopter and for good dynamic behaviour, image processing is done on an external computer.
Therefore, the architecture of the software is composed by a client/server application:

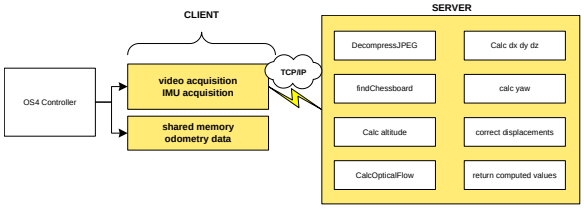


Fig.2: client/server architecture

The principle of the system is to estimate the displacements from the altitude and the optical flow.
For processing simplification, the floor is assumed flat and chessboards are placed on it in the environment.

First of all the client grabs an image and helicopter pitch and roll, and then sends it to the server. The server starts image processing by finding chessboard corners with OpenCV library functions.
As soon as square corners have been found, their size can be determinated and with a proper calibration of the system the altitude can be found.

Once the altitude is extracted, OpenCV grants points tracking between two frames and so x and y movements can be measured, and vectors tracking provides relative yaw.
Finally corrections on the observed point's motion due to angles change have to be computed (from pitch, yaw, roll and altitude).

The results were compared with the system mounted on a rolling robot, and both encoders and visual odometry systems where compared as showed below:

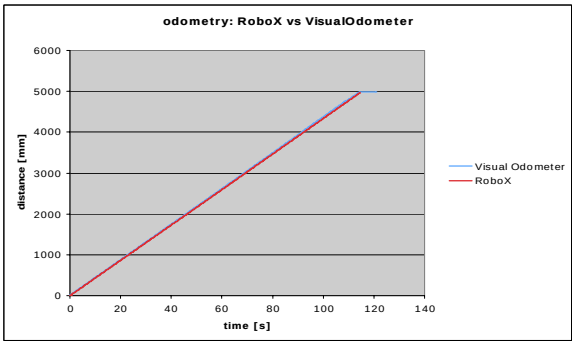


Fig. 3: odometry from RoboX and Visual Odometer

Table des matières

1. Introduction

Une thèse en cours vise à développer et contrôler de manière autonome un hélicoptère à quatre rotors. Le contrôle de ce robot volant a besoin de pouvoir mesurer ou extraire sa position relative ou absolue. Le but de ce projet consiste à développer un système de positionnement (odométrie) utilisant la vision. Une mini-caméra a déjà été conçue dans le Laboratoire des Systèmes Autonomes 3 est a été choisie pour utilisation sur l'hélicoptère.

2. L'hélicoptère OS4

L'hélicoptère OS4 est un appareil à quatre rotors, deux tournant dans un sens et deux dans l'autre. Un contrôleur a été développé pour garantir la stabilité de l'assiette. Celui-ci utilise une IMU (Inertial Measurement Unit) qui fourni les trois angles (pitch, roll, yaw) dans l'espace.

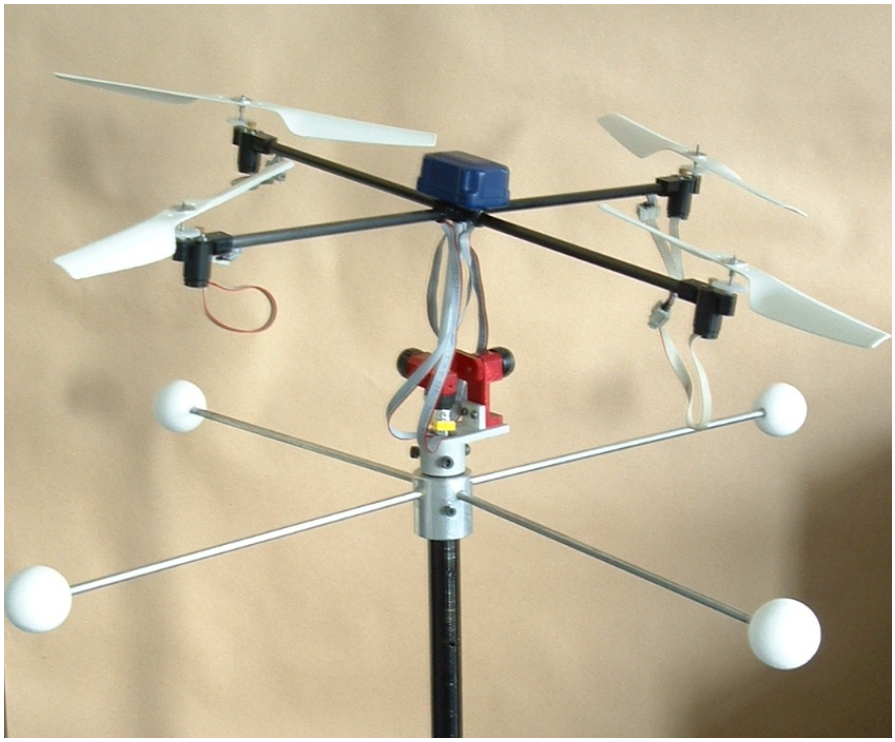


Figure 1: l'hélicoptère OS4 sur son socle

3. La X-Board

La X-Board est un ordinateur embarqué compact qui tient sur une carte de l'ordre de 10cm^2, qui est très légère (env. 20 grammes). Le processeur embarqué et un Geode SC1200 cadencé à 266MHz. Une distribution de linux minimale est installée sur une carte flash et est chargé en mémoire au démarrage. Le but est d'implémenter le contrôleur sur cette carte, puis de l'embarquer sur l'hélicoptère.

Figure 2: processeur Géode sur la X-Board

4. Cahier des charges

Dans un premier temps, l'objectif est de préparer l'électronique nécessaire pour le projet (composants servant au fonctionnement de la caméra). De même, le pilote pour linux doit être installé sur la x-board.
De plus le but est d'avoir le matériel le plus léger possible étant donné qu'il va devoir être porté par l'hélicoptère (10 grammes max.).
L'environnement ou évolue celui-ci est en intérieur, ce qui limite l'altitude entre zéro et 3 mètres maximum.
L'hélicoptère est conçu de manière à avoir la plus grande dynamique possible. Les déplacements peuvent monter jusqu'à 2 [m/s].

Un choix doit être fait quant au système de positionnement à utiliser. Pour cela un algorithme doit être choisi pour remplir les tâches demandées. Il doit être pris en considération que la puissance de calcul sur la x-board est limitée.
Au final l'algorithme doit être implémenté et testé sur l'hélicoptère.

5. Préparation de la caméra

Un module caméra compact a été développé à l'ASL 3 pour les yeux de Robota. Sa petite taille et sa légèreté font qu'il a été retenu pour l'application sur l'hélicoptère et doit être assemblé et testé.

5.1. Soudage des composants

La caméra est de type OV7648FB interfacée à un driver USB OV519 de Omnivision. Le PCB[1] double face comporte une cinquantaine de composants. Le montage final est représenté sur la figure ci-dessous :

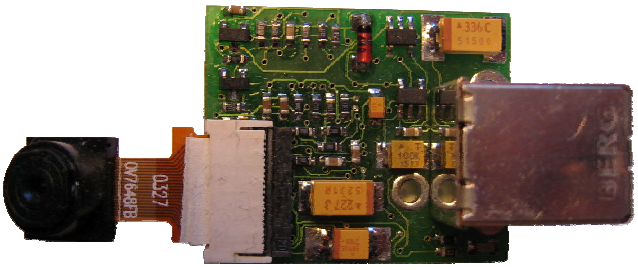


Figure 3: PCB de la caméra avec les composants soudés

5.2. Compilation du noyau linux avec les drivers requis

L'installation et l'utilisation de la caméra sur la x-board ainsi que tous les autres périphériques utilisés par le contrôleur (i^2c, port série), requièrent plusieurs modules du noyau linux.
Le fonctionnement de la caméra sous linux requiert les modules du noyau suivants :

- USB
- USB-UHCI
- video4linux
- ov51x

Les quatre premiers sont des modules standards présents dans les sources du noyau. Pour les activer il faut vérifier que les lignes suivantes soient présentes dans /usr/src/linux/.config:

- CONFIG_USB=m
- CONFIG_USB_OHCI_HCD=y
- CONFIG_USB_UHCI_HCD=y
- CONFIG_VIDEO_DEV=m

L'ov51x est un module développé pour les chips ov511, ov518 et le support ov519 a été ajouté mais n'est officiellement supporté. Ce module est disponible à l'URL http://alpha.dyndns.org/ov511/download.html#ov51x. [7]

De plus le contrôleur utilise le bus i^2c:

- CONFIG_I2C=m

[1] PCB : Printed Circuit Board

En utilisation normale, le système de fichier est chargé depuis une carte flash. Mais dans ce projet, les tests demandent une écriture fréquente de fichiers, et pour économiser le nombre d'écritures sur la carte flash (qui est limité), un noyau est chargé depuis le lecteur de disquette et la racine du système de fichiers est chargée par NFS[2]. Pour cela il faut encore activer les options suivantes dans le noyau:

- CONFIG_NFS_FS=y
- CONFIG_SUNRPC=y
- CONFIG_IP_PNP=y

Ensuite le noyau et les modules sont compilés: make bzImage ; make modules
Le noyau peut être copié directement sur une disquette, puis la disquette de démarrage est créée par XXX. Il reste à copier les modules dans le système de fichiers à exporter par NFS (dans */lib/modules/modules-<version_du_noyau>/*), puis de configurer les IPs de la xboard sur la disquette de démarrage (dans le fichier *linux.cfg*) (la procédure détaillée se trouve en annexe 19.5).
Le serveur NFS contient le système de fichier qui doit être chargé depuis la xboard. Pour cela un système minimal linux peut être créé dans un répertoire (par ex. /export/xboard/), puis une ligne est à ajouter dans /etc/exports:

- /export/xboard <ip-de-la-xboard>(rw,no_root_squash,sync)

Le système peut ensuite être réamorcé en démarrant sur la disquette de démarrage.

5.3. Tests de la caméra

C'est le module ov51x qui gère la caméra. Celui-ci permet d'e contrôler la caméra par /dev/video0. Si le descripteur n'existe pas, il faut le créer par la commande suivante:

- mknod /dev/video0 c 81 0

La caméra est reconnue si le système affiche les messages suivant lors de la connexion de la caméra au port USB:

```
ohci_hcd 0001:10:19.0: wakeup
usb 2-1: new full speed USB device using address 3
usb 2-1: Product: USB Camera
usb 2-1: Manufacturer: OmniVision Technologies, Inc.
ov51x.c: USB OV519 video device found
ov51x.c: Sensor is an OV7648
ov51x.c: Device at usb-0001:10:19.0-1 registered to minor 0
```

Le driver ov51x est fourni avec un application de test: getjpeg. Comme son nom l'indique l'image obtenue de l'OV519 est au format JPEG. Un programme standard ne pourra donc pas lire la vidéo provenant de la caméra si celui-ci ne supporte pas la décompression JPEG. Ce petit programme acquière une image et la sauvegarde dans un fichier. Quelques modifications de son code permettent de mesurer les vitesses d'acquisitions.

[2] NFS: network file system

sans décompression JPEG	
Résolution [pixels]	Vitesse d'acquisition [fps]
640x480	12.98
320x240	14.1
avec décompression JPEG	
Résolution [pixels]	Vitesse d'acquisition [fps]
640x480	6.3
320x240	**7.15**

Tableau 1: Vitesses d'acquisition de la caméra sur la xboard

Le hardware doit théoriquement fournir jusqu'à 30 fps, mais il n'a pas été possible de le réaliser. Le driver linux supporte le chip OV519 uniquement avec ses fonctions de base. Les modifications à la main faites dans les sources du driver n'ont pas permis de monter plus haut que 15 fps. De même, pour obtenir une image de 320x240 au lieu des 640x480, une modification directe de la source du driver est nécessaire:
Dans la fonction **ov519_mode_init_regs**, ligne 3681: ajouter width=320; height=240.

De plus l'OV519 fournit une image compressée en JPEG, ce qui ralentit sensiblement la vitesse d'acquisition comme on peut le voir dans le Tableau 1: , étant donné qu'il faut la re-décompresser avant le traitement d'image.

La vitesse maximale obtenue est donc de **7.15 fps**. L'angle d'ouverture de la caméra avec image réduite à 320x240 a été mesuré à **22°**. Cet angle se mesure en positionnant l'objectif à une distance connue d'un mètre et en lisant sur l'image acquise la graduation du mètre:

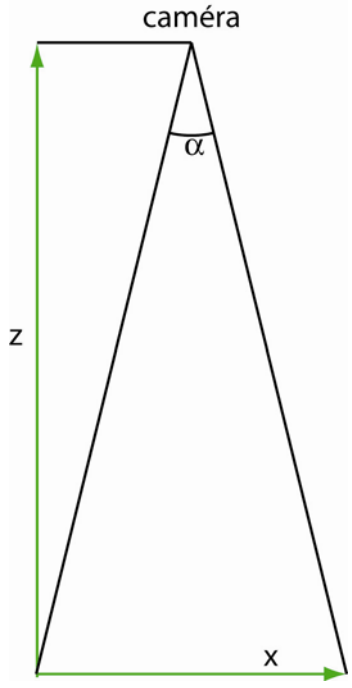


Figure 4: mesure de l'angle d'ouverture

Calcul de l'angle:

$$\tan(\frac{\alpha}{2}) = \frac{x}{2 \cdot z}$$

Distance de l'objectif: 500mm.
Graduation mesurée au bord de l'image:
194mm -> angle d'ouverture = 22°

5.3.1. Images obenues

L'objectif de la caméra étant petit, le temps d'intégration pour avoir une image est sensiblement élevé, ce qui pose problème lors des déplacements comme on peut le voir sur la figure suivante :

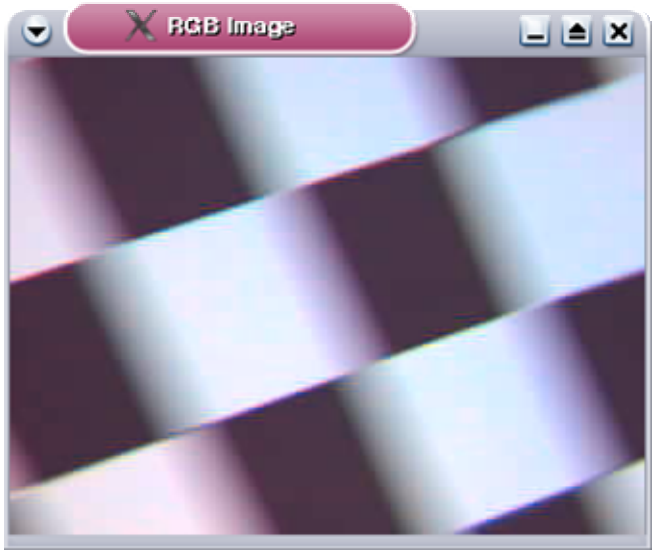


Figure 5: image prise en déplacement avec exposition automatique (vitesse environ 1[m/s])

Le driver ov51x étant incomplet, il ne permet pas de modifier la durée d'exposition.

Cependant une modification du code source à la main permet de résoudre le problème:
- ajout dans la fonction **ov51x_v4l1_ioctl_internal**: ligne 5840:

```
case 12345:{
        PDEBUG(3,"SETEXPOSURE");
        sensor_set_exposure(ov, 9);
        break;
      }
```

- Appel depuis le programme principal avec:

```
ioctl(video_device, 12345, &vquery);
```

Le résultat après changement du temps d'exposition permet d'obtenir une image satisfaisante, même lors de grand déplacements :

Figure 6: Exposition de 9 et déplacement d'environ 1[m/s] (à la main)

On voit cependant sur l'image suivante qu'une grande vitesse induit une déformation de l'image, en effet l'intégration sur le capteur optique se fait ligne par ligne et on observe un retard des lignes du bas sur les lignes du haut :

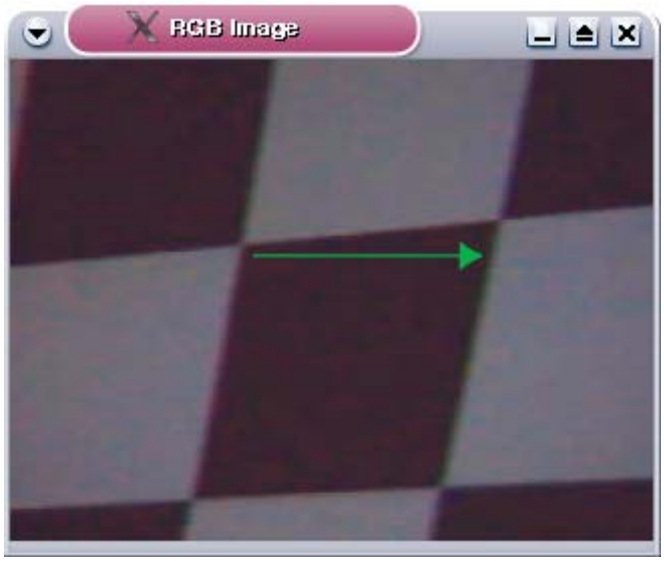


Figure 7: Exposition de 5 et déplacement de environ 2 [m/s], direction droite

Le cahier des charge spécifie un déplacement de 2 [m/s], cette caméra est trop lente pour acquérir des images suffisamment rapidement. Il faudrait augmenter la vitesse, ce qui est impossible avec le matériel existant. Il faut donc limiter la vitesse de déplacement du robot à basse altitude.

6. Catalogue des solutions

La recherche d'algorithme pouvant être utilisés pour concevoir un système d'odométrie visuelle a conduit à plusieurs méthodes qui peuvent se ranger dans deux catégories distinctes. Dans la première, aucune modification de l'environnement où évolue le robot n'est nécessaire. Au contraire, dans la seconde, il faut ajouter/modifier de manière spécifique des points clefs de l'environnement.

6.1. Méthodes sans modification de l'environnement

6.1.1. Stéréovision

Ce choix est le plus robuste, étant donné qu'il se base sur les informations visuelles à disposition dans la scène. Le système consiste à prendre des images simultanément depuis deux caméra (stéréovision) séparées par une distance connue au temps *t*, déterminer des coins facilement reconnaissables (points de Harris[3]) sur les deux images:

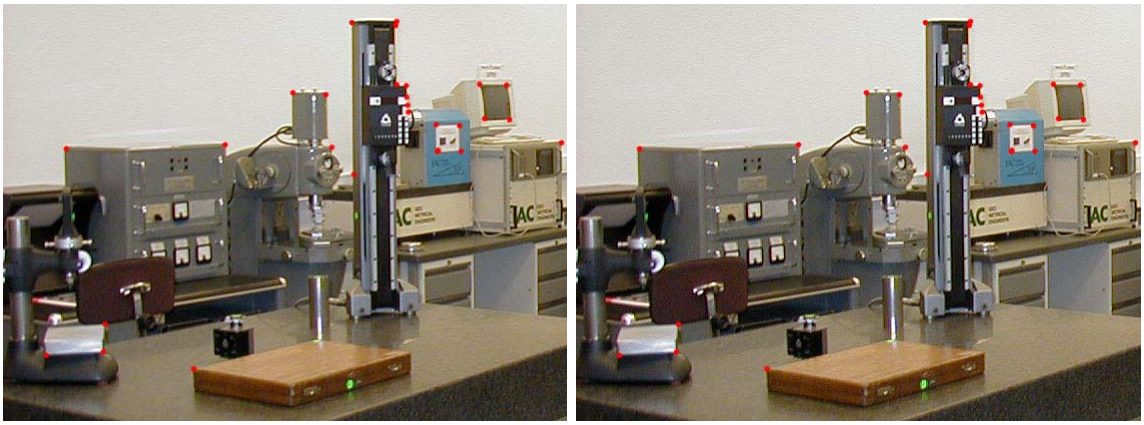


Figure 8: Correspondance des points de Harris au temps *t*

L'opération est répétée au temps *t+1*:

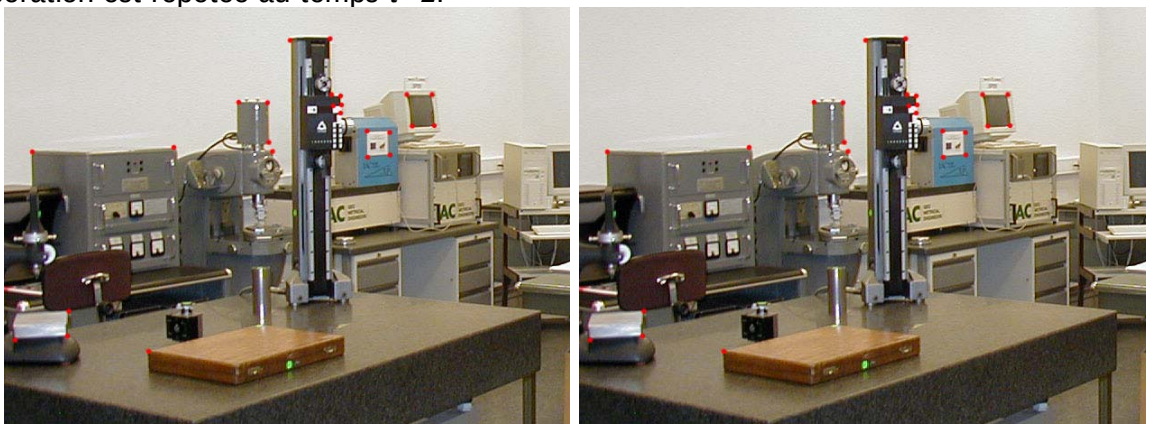


Figure 9: Correspondance des points de Harris au temps *t+1*

[3] Point de Harris : points spécifiques de par leur facilité à être retrouvés, qui sont détectés par l'algorithme de Harris

Il faut ensuite établir la correspondance de ces points sur les quatre images. La dernière étape est du calcul pur pour retrouver les déplacements dans les six degrés de liberté à partir des coordonnées des points trouvés.

6.1.2. Méthode basée sur l'intensité de l'image [4] [5]

Cet algorithme est basé sur l'estimation du déplacement à partir d'images de références. Trois images, f , f_{tx-} et f_{tx+} sont prises simultanément par trois caméras différentes: une caméra centrale, une caméra déplacée selon une distance m_{tx-}^{ref} en x et une troisième distante de m_{tx}^{ref} selon x respectivement. Pour estimer le déplacement, on prend une image centrale f , et on assume que cette image a été déformée de manière linéaire à partir des deux images aux positions de référence, ce qui est vrai dans le cas de petits déplacements.
L'approximation linéaire de f peut s'écrire $\hat{f} = f_0(\hat{m}_{tx} / m_{tx}^{ref})(f_{tx-} - f_{tx+})$ où $\hat{m}_{tx}$ est l'estimation du déplacement désiré.
Le problème se résume alors à minimiser l'erreur entre l'image f et son estimation: $E = \iint \psi (f - \hat{f})^2 dxdy$ où ψ est une fonction fenêtre. En dérivant E par rapport à $\hat{m}_{tx}$ et en égalisant à zéro, on obtient:

$\hat{m}_{tx} = \frac{2m_{ref}^{tx} \iint \psi (f - f_0)(f_{tx-} - f_{tx+}) dxdy}{\iint \psi (f_{tx-} - f_{tx+})^2}$. Le déplacement peut alors être calculé à partir des intensités des images capturées. Le principe est le même pour les déplacements en y et z.

6.1.3. Mesure distance et flux optique

Cette méthode peut être très rapide car aucune reconnaissance ne doit être effectuée. Le flux optique est fonction de la vitesse et du déplacement, et une information de distance par rapport à la surface visualisée permet de connaître le déplacement réel.
Plusieurs solutions sont possibles pour mesurer la distance:

6.1.3.1. Capteur infrarouge Sharp:

Le capteur infrarouge sharp permet une mesure de distance entre 10cm et 80cm. La réponse du capteur peut être linéarisée afin d'obtenir une droite:

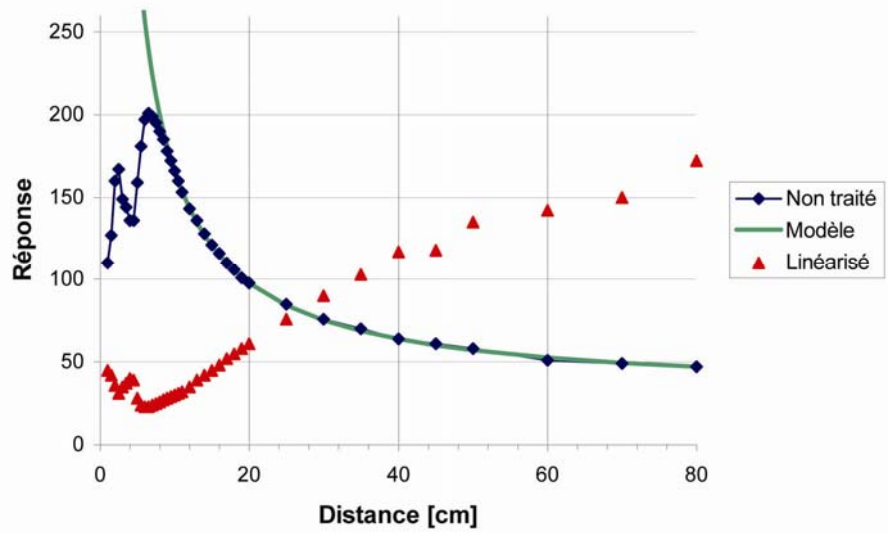


Figure 10: Réponse du sharp

6.1.3.2. Triangulation laser:

La triangulation par laser peut également servir à mesurer la distance selon le principe ci-dessous:

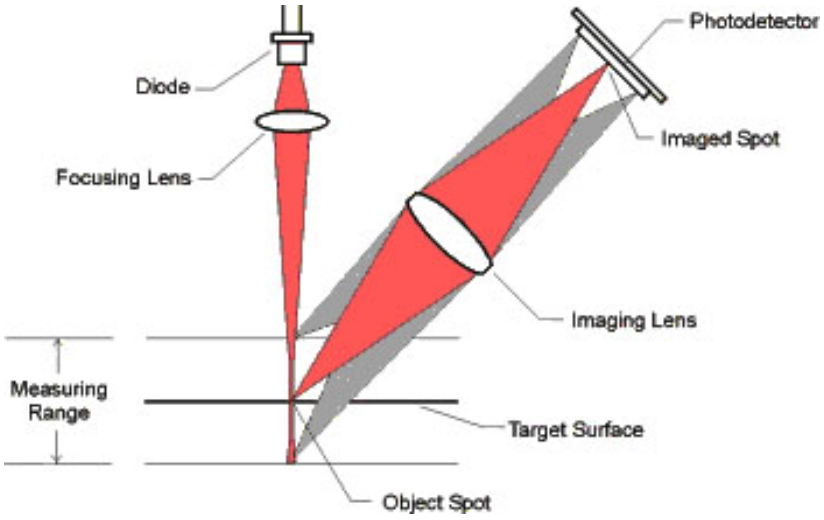


Figure 11: Triangulation par laser

Le calcul ici est simple. Suivant la position de la tache du laser sur l'image, il est possible d'avoir une information sur la distance de l'objet visé.

6.1.4. Caméra linéaire haute vitesse

Cette méthode requiert une fréquence d'acquisition très élevée, surtout en cas de grande vitesse de déplacement. D'après des tests à l'ASL2 il faut que le déplacement entre deux images successives soit de l'ordre de 3-4 pixels pour que l'algorithme fonctionne. La caméra d'OmniVision ne permet pas une vitesse d'acquisition assez élevée pour de tels calculs. Une solution peut être des caméras linéaires (64 pixels) qui sont beaucoup plus rapides. Les modules de caméras linéaires existants à l'ASL ne permettent pas une telle vitesse d'acquisition (de l'ordre de 10fps). Il faudrait concevoir l'électronique pour cette application.

6.2. Méthodes avec modification de l'environnement

6.2.1. Landmarks

Il s'agit de disposer des features facilement reconnaissables par le robot dans l'environnement où il évolue. Il y a deux systèmes possibles. Le premier est un système absolu, où le robot doit connaître la topographie de l'endroit où il se trouve. Des landmarks spécifiques (par exemple un chiffre) permettent de savoir ou il se trouve, et en fonction de sa géométrie, il peut trouver sa position exacte dans l'espace. Le second système est relatif, tous les landmarks sont les mêmes et le robot se repère en fonction du déplacement observé de ces derniers. La topographie de l'environnement n'a pas besoin d'être connue.

6.2.2. Damiers

Il s'agit de disposer au sol un damier de taille connue, puis la caméra située en dessous de l'hélicoptère mesure la taille des carrés et leur déplacement. La librairie d'Intel (OpenCV)

contient des algorithmes permettant une multitude de traitements d'images, comme la recherche de coins sur un damier.

6.3. Résumé des méthodes possibles

Méthode	Avantages	Inconvénients
stéréovision	- pas besoin de modifier l'environnement - précis - robuste	- lourd à calculer - requiert une bonne qualité d'image donc une bonne optique donc du poids sur l'hélicoptère
Intensité de l'image	- pas besoin de modifier l'environnement	- requiert une grande vitesse d'acquisition - lourd à calculer (itérations)
distance + flux optique	- rapide	- demande un capteur supplémentaire donc du poids en plus - risque de poser des problèmes lors d'une rotation pure
caméra linéaire	- très rapide	- requiert un nouveau développement hardware - requiert une grande vitesse d'acquisition
landmarks	- pas de dérive dans le cas des landmarks absolus	- demande de modifier l'environnement
damiers	- pas besoin de capteur supplémentaire - facile à implémenter	- demande de modifier l'environnement

La solution idéale serait de pouvoir implémenter un algorithme basé sur la stéréovision. C'est la méthode la plus robuste mais aussi la plus gourmande en calculs. Dans le cas de l'hélicoptère ce n'est pas possible de l'implémenter pour les raisons de qualité d'image et de temps de traitement.
Les méthodes basées sur l'intensité ne sont pas possibles non plus car les vitesses d'acquisition ne sont pas assez rapides (limite de notre caméra: 15fps). Les caméras linéaires de l'ASL ne sont pas assez rapides (vitesse d'acquisition: 5 fps) car les modules existants utilisent un protocole trop lent pour cette application.

Afin de simplifier le problème de calcul de l'altitude, on assumera que le sol est plat.

Finalement, une trop grande simplification du système afin de gagner en rapidité ne servirait à rien en raison des trop grandes erreurs de mesures. Il vaut mieux perdre en vitesse mais d'avoir un système qui fonctionne plutôt qu'un qui ne fonctionne pas.
Afin d'économiser du poids, la solution avec un capteur supplémentaire ne sera pas retenue non plus.
La solution choisie est donc celle des damiers, avec une seule caméra.
Le développement se fait d'abord sur un PC externe et ensuite des tests seront faits sur l'hélicoptère.

L'idée est premièrement d'obtenir l'altitude de l'hélicoptère par rapport au sol. Celle-ci peut être calculée à partir la taille réelle des carrés du damier et leur taille vue par la caméra.
Ensuite les déplacements peuvent être calculés à partir du mouvement des points sur des images successives, en tenant compte évidemment de l'altitude.
Il faut également pouvoir corriger les mouvements parasites vus par la caméra qui sont dus uniquement au changement d'assiette de l'hélicoptère (pitch, roll) ainsi que du yaw.

7. Architecture à développer

Le contrôleur de l'hélicoptère doit pouvoir obtenir les valeurs des déplacements selon x, y et z, ainsi que les vitesses selon les mêmes axes. De plus il a été demandé s'il était possible d'avoir une mesure du yaw, celle obtenue de l'IMU n'est pas fiable car elle est basée sur le champ magnétique terrestre et donc très sensible aux champs, comme celui d'un écran d'ordinateur par exemple.
Le software est sous la forme d'une librairie, qui peut être utilisée directement par le contrôleur. Celui-ci aura la capacité de démarrer l'odométrie et de permettre au contrôleur de récupérer à n'importe quel instant les valeurs des déplacements, ainsi qu'un timestamp et des drapeaux indiquant l'état de l'odométrie (erreur ou autres problèmes).

On peut résumer l'architecture comme ceci:

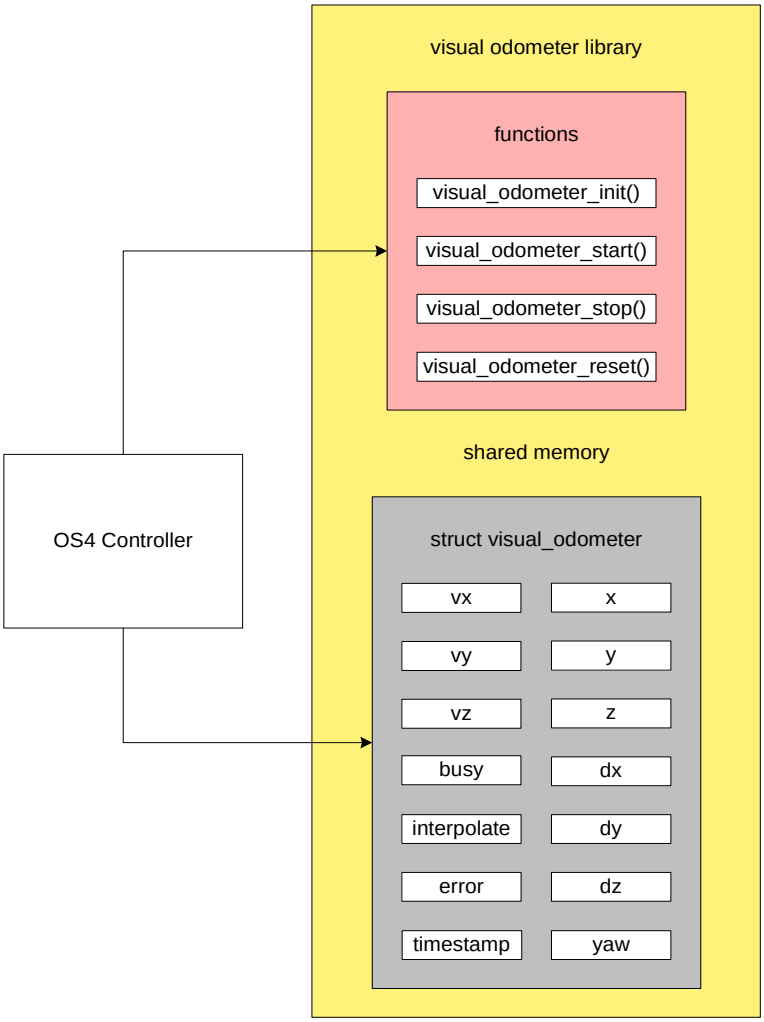


Figure 12: structure et utilisation de la librairie

8. Calibration de la caméra

Certaines caméras, surtout celles de petite focale, présentent des déformations de l'image sur les bords. Ce défaut peut être corrigé à l'aide d'une calibration préalable de la caméra. Une toolbox de calibration de caméra utilisable sur Matlab est disponible sur http://www.vision.caltech.edu/bouguetj/calib_doc/ [8]

La calibration se fait à l'aide d'un damier de taille connue, que l'on présente devant la caméra sous différents angles et différentes positions. Les images prises à chaque position sont traités par la toolbox et en fonction de la taille des damiers, on peut sortir les paramètres intrinsèques de la caméra, qui sont les suivants :

- focal length : longueur focale (en pixels) : vecteur 2x1 : **fc**
- principal point : coordonnées du point principal : vecteur 2x1 : **cc**
- skew coefficient : définit l'angle entre les axes des pixel x et y : **alpha_c**
- distortions : coefficients de distortion : vecteur 5x1 : **kc**

Aspect théorique:

Soit un point P définit par le vecteur $XX_c = [X_c; Y_c; Z_c]$ dans le repère de la caméra.

Soit x_n la projection normalisée selon le modèle pinhole : $x_n = \begin{bmatrix} X_c / Z_c \\ Y_c / Z_c \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$

Soit $r^2 = x^2 + y^2$.

En incluant la distortion de la lentille la nouvelle projection normalisée est définie par $x_d = \begin{bmatrix} x_d(1) \\ x_d(2) \end{bmatrix} = \left(1 + kc(1) \cdot r^2 + kc(2) \cdot r^4 + kc(5) \cdot r^6\right) \cdot x_n + dx$

Où dx est la distortion tangentielle : $dx = \begin{bmatrix} 2kc(3)xy + kc(4)\left(r^2 + 2x^2\right) \\ kc(3)\left(r^2 + 2y^2\right) + 2kc(4)xy \end{bmatrix}$

Les coordonnées du point P final est $\begin{cases} x_p = fc(1)\left(x_d(1) + alpha\_c \cdot x_d(2)\right) + cc(1) \\ y_p = fc(2)x_d(2) + cc(2) \end{cases}$

On peut définir alors la matrice de la caméra par $KK = \begin{bmatrix} fc(1) & alpha\_c \cdot fc(1) & cc(1) \\ 0 & fc(2) & cc(2) \\ 0 & 0 & 1 \end{bmatrix}$

L'image déformée peut être corrigée à l'aide de cette matrice selon la loi : $x = \begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix} = KK \begin{bmatrix} x_d(1) \\ x_d(2) \\ 1 \end{bmatrix}$

avec x les coordonnées du point et x_d les coordonnées du point sur l'image déformée.

8.1. Résultats de la calibration

La calibration a été faite avec un damier 15x15 de taille 50mm.

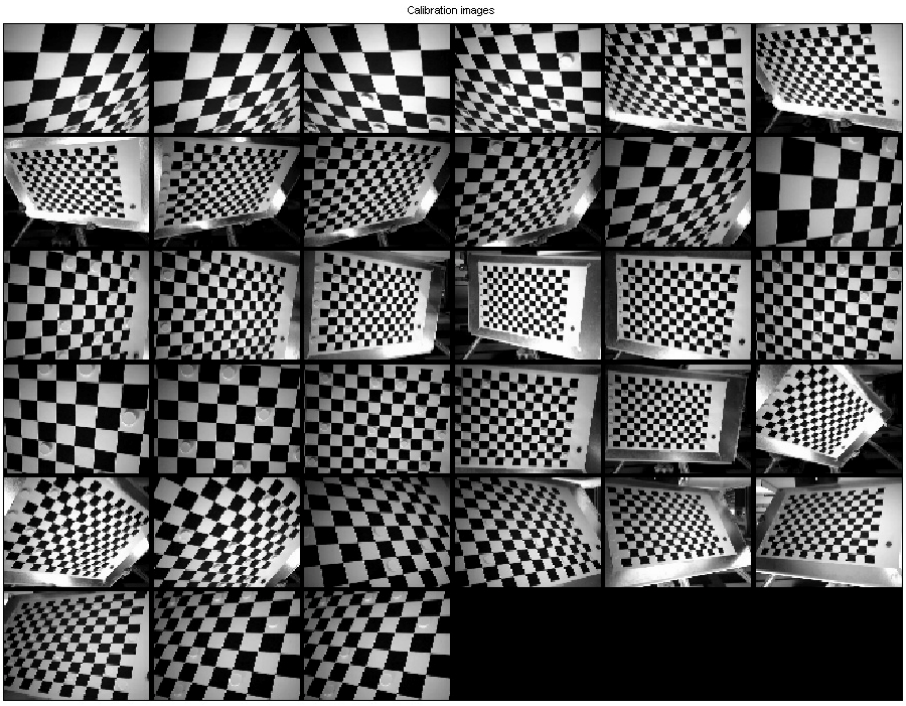


Figure 13: images prises pour la calibration

Une fois les images chargées, elles sont prises une à une et il est demandé de sélectionner les quatre coins du damier:

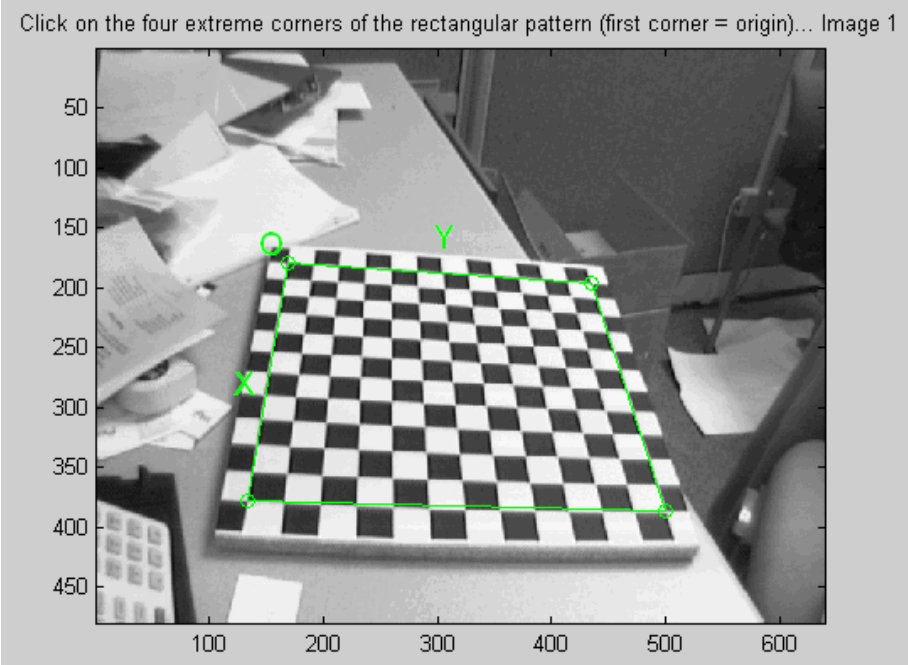


Figure 14: sélection des coins

L'étape suivante est la calibration. Les calculs prennent un certain temps, suivant la puissance de l'ordinateur.

Résultats :

- `fc = [ 799.784627201098370 ; 799.674384621097720 ];`
- `cc = [ 340.856302662624610 ; 247.269018672155710 ];`
- `alpha_c = 0.000000000000000;`
- `kc = [ 0.077139917335662 ; -0.091885690882282 ; -0.009107603984836 ; 0.001960525815210 ; 0.000000000000000 ];`
- `fc_error = [ 0.592226997298519 ; 0.555977865895232 ];`
- `cc_error = [ 1.194952800144069 ; 0.949142547285693 ];`
- `alpha_c_error = 0.000000000000000;`
- `kc_error = [ 0.005526298862917 ; 0.030245089013729 ; 0.000498853595427 ; 0.000655831662502 ; 0.000000000000000 ];`

La matrice de la caméra ainsi obtenue est:

$$KK = \begin{bmatrix} 799.784627201098370 & 0 & 340.856302662624610 \\ 0 & 799.674384621097720 & 247.269018672155710 \\ 0 & 0 & 1 \end{bmatrix}$$

La toolbox permet également de visualiser les distortions :

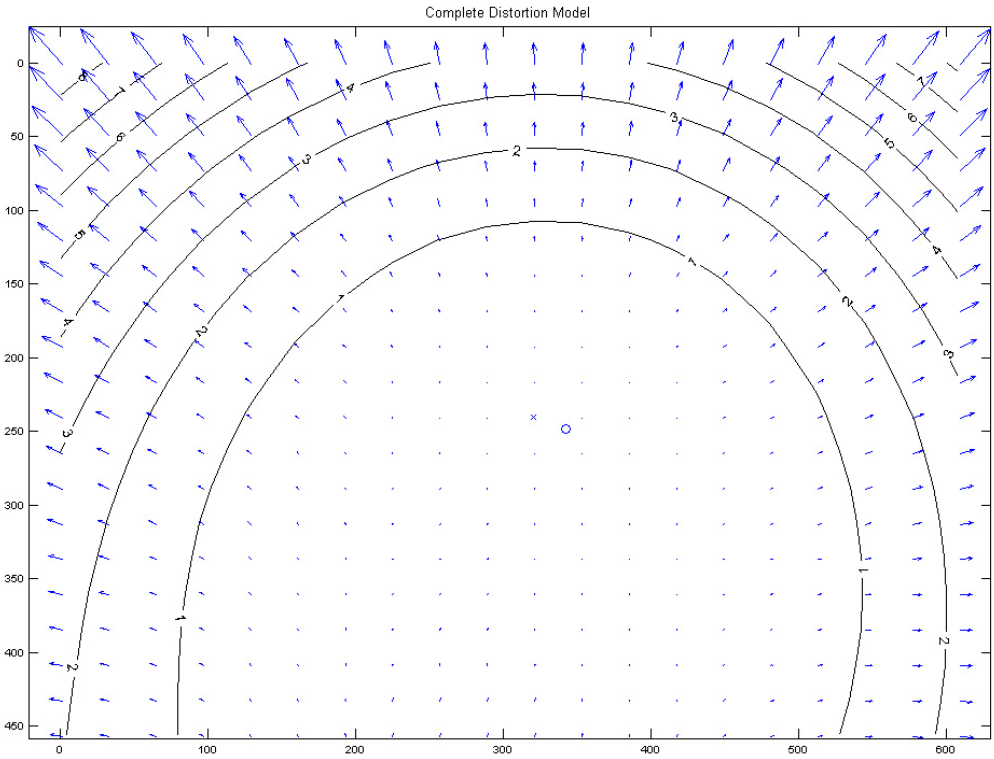


Figure 15: distortions

La calibration effectuée, l'image capturée par la caméra peut être corrigée pixel par pixel comme décrit plus haut. Cette opération est supportée dans la librairie OpenCV.

9. Librairie OpenCV [9]

Cette librairie opensource est utilisée pour accomplir diverses tâches utiles dans le programme:

Conversion de l'image couleur en gris:

- `cvConvertImage(RGBImage, GreyImage, 0);`

Correction de la déformation de l'image du à la lentille, avec les paramètres trouvés lors de la calibration:

- `cvUnDistortOnce(GreyImage, TempImage, cameraMatrix, dist_coeffs, 0);`

Recherche des coins des carrés sur le damier:

- `cvFindChessBoardCornerGuesses(GreyImage, TempImage, MemStorage, cvSize(win_size,win_size), corners, &numcorners);`

Recherche affinée de ces mêmes coins (avec une précision < 1 pixel)_

- `cvFindCornerSubPix(GreyImage, corners, numcorners, cvSize(1,1), cvSize(-1,-1), cvTermCriteria(CV_TERMCRIT_ITER|CV_TERMCRIT_EPS,20,0.03));`

Tracking des coins des carrés entre deux images successives:

- `cvCalcOpticalFlowPyrLK(PreviousImage, GreyImage, prev_pyramid, pyramid, old_corners, corners, numcorners, cvSize(win_size,win_size), 3, status, 0, cvTermCriteria(CV_TERMCRIT_ITER|CV_TERMCRIT_EPS,20,0.03), flags );`

10. Librairie VisualOdometer

Cette libraire est la pièce principale de tout le software. Elle est utilisée directement par le contrôleur de l'hélicoptère.
La première étape pour son utilisation est l'appel de la fonction **visual_odometer_init()**, qui initialise la caméra et réserve un espace en mémoire pour placer les données de l'odométrie.

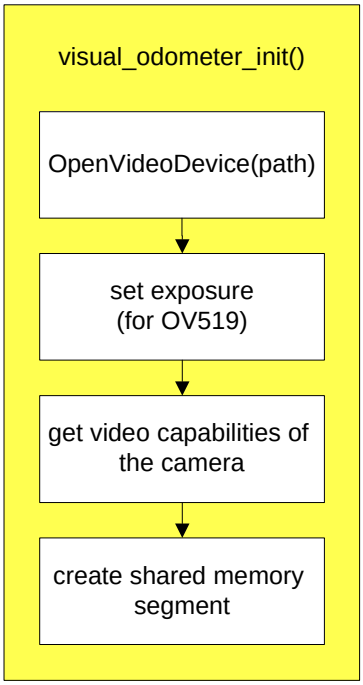


Figure 16: Flux de la fonction visual_odometer_init()

La mémoire partagée sert à pouvoir accéder aux données de l'odométrie depuis plusieurs processus différents (la librairie et le contrôleur). Un drapeau est utilisé pour éviter les accès simultanés.
La structure des données est la suivante:

```
struct odometry_data{
  //! absolute altitude
  double altitude;
  //! absolute yaw
  double yaw;

  //! relative x displacement, can be initialized
  double x;
  //! relative y displacement, can be initialized
  double y;
  //! relative z displacement, can be initialized
  double z;

  //! x  displacement between last and current frame
  double dx;
  //! y displacement between last and current frame
  double dy;
  //! z displacement between last and current frame
  double dz;
  //! yaw difference between last and current frame
  double dyaw;

  //! x speed
  double vx;
  //! y speed
  double vy;
  //! z (vertical) speed
  double vz;

  //! timestamp of last displacement
  double timestamp;

  //! flag: 0: normal measure, 1: measure error: the displacement and positions have been interpolated
  short interpolate;

  //! flag: 1: library is updating data
  int busy;

  //! flag: library is running and updating data
  int running;

  //! flag: more than MAX_ERRORS have been reached, interpolated data could be incorrect
  short error;
};
```

A ce stade le système est prêt et peut être démarré par la fonction **visual_odometer_start()**. Cette fonction tourne en boucle tant que l'odométrie est calculée. Afin de ne pas bloquer le contrôleur, la fonction fait appel à **fork(),** ce qui crée une copie en mémoire du processus en cours, ce qui permet d'une part de commencer la boucle de calcul, et d'autre part de retourner au contrôleur le pointeur vers l'espace mémoire partagé pour lire les données.

```
pid = fork(); // create a copy of the process for giving back
control to controller

if(pid){ // parent process
    return helicopter_variables;
} else { // child process
```

La boucle peut en tout temps être arrêtée à l'aide de la fonction **visual_odometer_stop().**

Le traitement d'image se fait en plusieurs étapes. Premièrement les données des angles du pitch et roll sont obtenues de l'IMU, puis une image est acquise par la caméra. Si l'image est au format JPEG, celle-ci est décompressée. Ensuite le traitement se poursuit par le redressement de l'image, la détection des coins des carrés, la recherche de la taille de ces derniers, puis l'altitude peut être calculée. Le flux optique est également calculé et sert à déterminer les déplacements en x et y. Le calcul du yaw vient ensuite et finalement une correction des mouvements observés est effectuées à partir des données des angles:

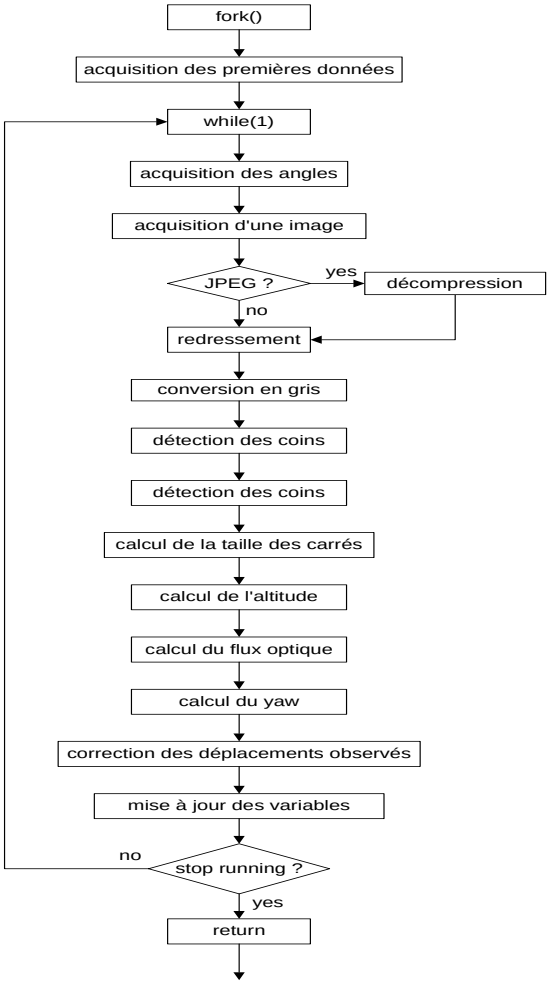


Figure 17: schéma bloc de visual_odometer_start()

10.1. Acquisition des angles

Les angles sont rapidement lus par le port série par l'appel à la fonction *MT9_GetAngles(float &angles)*.
Avant de passer à l'acquisition de l'image, le timestamp est sauvegardé dans la variable *futur_timestamp*. En effet, le timestamp qui est pris maintenant correspond à l'itération

suivante, car comme on le verra lors de l'acquisition de l'image, celle-ci se fait en deux étapes, premièrement la capture, puis le traitement se fait à l'itération suivante.

10.2. Acquisition de l'image

L'acquisition se fait au travers des appels système standards de video4linux, donc indépendamment de la caméra utilisée, pour autant qu'elle soit supportée par video4linux. Une autre caméra ou webcam peut ainsi fonctionner de manière transparente.
L'OV519 comporte deux buffers[4] pour l'image, c'est-à-dire que l'on peut commander l'acquisition d'une image pendant qu'on rapatrie une autre. Ceci permet de gagner du temps, étant donné qu'on peut effectuer le traitement pendant que l'image suivante est en cours d'acquisition.
La première étape est de donner l'ordre à la caméra de faire une acquisition (VIDIOCMCAPTURE), puis la seconde est de rapatrier l'image depuis la mémoire de la caméra dans le userspace[5] (VIDIOCSYNC).

La procédure utilise deux appels standards de video4linux:

```
      if (ioctl (video_device, VIDIOCMCAPTURE, &vmmap) == -1){
        perror ("VIDIOCMCAPTURE()");
      }

      vmmap.frame = (vmmap.frame + 1) % vmbuf.frames; // get other
buffer

       if (ioctl (video_device, VIDIOCSYNC, &vmmap.frame) == -1) {
        perror ("VIDIOCSYNC()");
       }
```

Le pointeur *framepointer* accède directement aux données de l'image reçue.

10.3. Décompression JPEG

Le driver OV519 compresse l'image au format JPEG avant de la transférer. Il ne permet pas pour l'instant de désactiver cette option. Afin de garder la transparence de la librairie, un test est fait sur les premiers bytes des données de l'image pour savoir si celle-ci est au format JPEG:

```
    memcpy(buffer, framepointer+6, 4);
    buffer[4] = '\0';
    if(!strcmp(buffer, "JFIF")){
      DecompressJPEG(sz, framepointer, rgbdata, WIDTH, HEIGHT);
```

Après cette opération, l'image est au format RGB et le traitement peut commencer.
L'image au format RGB est convertie en niveaux de gris par la fonction *cvConvertImage(RGBImage, GreyImage, 0)*. Cette étape est nécessaire pour les fonctions suivantes qui travaillent uniquement avec des images grises. De plus la taille d'une image en gris est trois fois plus petite qu'une image RGB.

[4] buffer: mémoire tampon
[5] userspace: espace mémoire utilisé par le programme

10.4. Correction de la géométrie de l'image

L'image donnée par la caméra est déformée et doit être corrigée à l'aide des paramètres intrinsèques de la caméra trouvés à lors de la calibration (chap. 8.1).

- `cvUnDistortOnce(GreyImage, TempImage, cameraMatrix, dist_coeffs, 0);`

dist_coeffs et cameraMatrix sont les matrices des coefficients obtenus:

- float cameraMatrix[] = {799.78462, 0, 340.8563, 0, 799.67438, 0, 0, 0, 1};
- float dist_coeffs[] = {0.07714, -0.091886, -0.0091076, 0,0019605};

L'image est maintenant prête à être traitée.

10.5. Extraction des coins

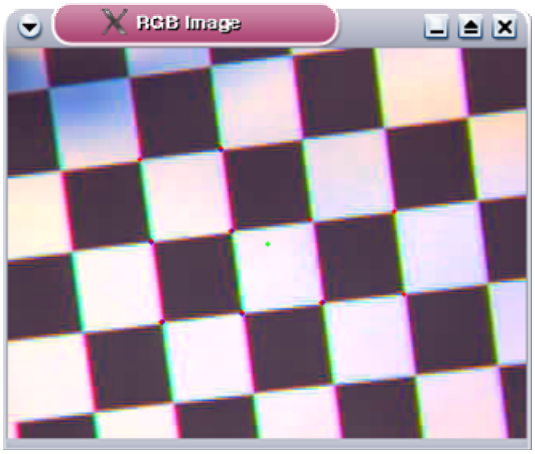
L'extraction des coins se fait à l'aide de deux fonctions de la librairie OpenCV.
Ces deux fonctions retournent un tableau de points contenant les coordonnées des coins sur l'image. La variable *numcorners* et égale au nombre de points trouvés sur l'image.

- `cvFindChessBoardCornerGuesses(GreyImage, TempImage, MemStorage, cvSize(win_size, win_size), corners, &numcorners);`

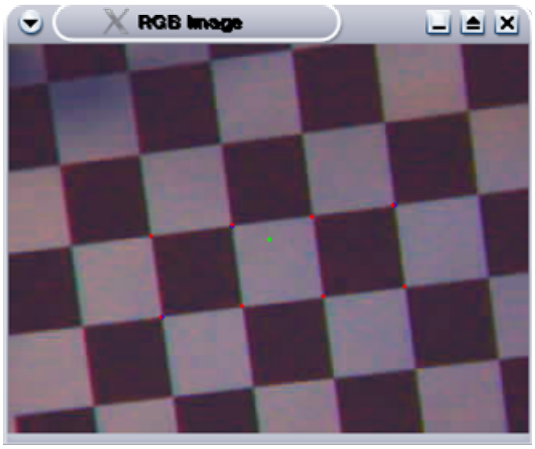
```
if(numcorners > 0)
```

- `cvFindCornerSubPix(GreyImage, corners, numcorners, cvSize(1,1), cvSize(-1,-1), cvTermCriteria(CV_TERMCRIT_ITER|CV_TERMCRIT_EPS, 20,0.03));`

Lors des tests il a été remarqué que les points sont plus faciles à trouver avec une bonne luminosité:



10 points trouvés



8 points trouvés

Figure 18: coins trouvés par l'algorithme

10.6. Recherche de la taille des carrés

Si l'algorithme a trouvé au minimum deux points, la suite des calculs peut être exécutée.
La taille du côté des carrés se trouve en calculant la plus petite distance entre deux points. Il se peut cependant que des points parasites apparaissent de temps en temps (par ex. un faux coin détecté). C'est pourquoi il faut filtrer ces éventuelles erreurs avec un algorithme adéquat:

1. calcul de toutes les distances entre chaque points et stockage dans un tableau
2. tri du tableau par ordre croissant des distances
3. commence la recherche à partir du début du tableau. Une moyenne est initialisée. A chaque itération, si la distance est proche dans un intervalle de **MIN_CARRE_TOLERANCE** par rapport à la précédente, elle est ajoutée à la moyenne. Si ce n'est pas le cas, une nouvelle moyenne est initialisée à partir de cette valeur.
4. Si une moyenne est définie au minimum par **MIN_DISTANCES** valeurs, on considère que la valeur est celle du côté des carrés et on sort de la boucle.

10.7. Calcul de z (altitude)

Le calcul de l'altitude se fait de manière assez simple du moment donné que l'on connaît la taille réelle du damier sa taille sur l'image 2D.
La distance du côté des carrés sur l'image est inversement proportionnelle à la distance du damier par rapport à la caméra.

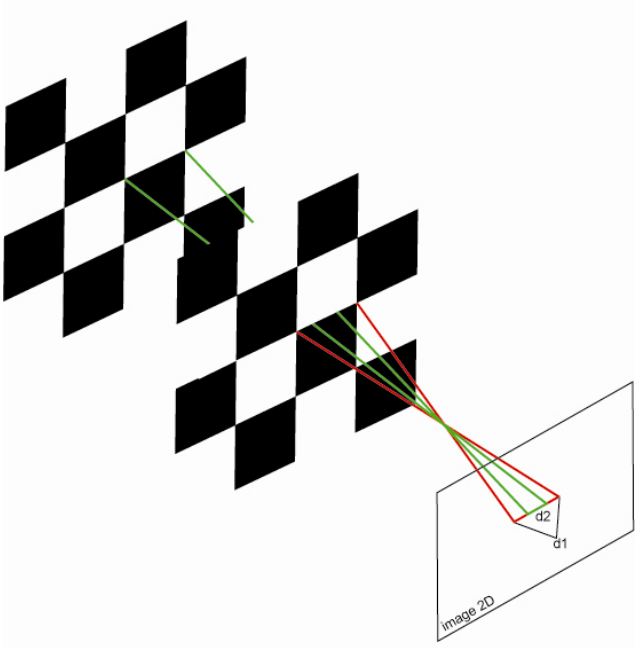


Figure 19: calcul de z

Il est alors possible de sortir une relation du type:

$z = \frac{CALIB\_SQUARE\_SIZE}{SquareSize} \cdot CALIB\_SQUARE\_DISTANCE$ avec SquareSize la distance du côté trouvée d'après l'image.

10.7.1. Calibration

L'outil de calibration **calibrate_altitude** affiche la valeur en pixel de la taille d'un carré, pour autant que l'algorithme ait trouvé au moins deux coins.

La calibration, de manière à être le plus précis possible, se fait à une distance minimale du damier, la précision sur la distance de la caméra ne change pas, par contre la précision en pixels sur l'image et inversement proportionnelle à la distance.
La calibration à été faite à **535mm** (`CALIB_SQUARE_DISTANCE`) du damier et la taille des carrés de 40mm de côté est de **179.67 pixels** (`CALIB_SQUARE_SIZE`)

10.7.2. Tests

L'évolution de l'erreur de l'altitude en fonction de la distance et de la taille des damiers:
Trois tailles de damiers ont été testées:

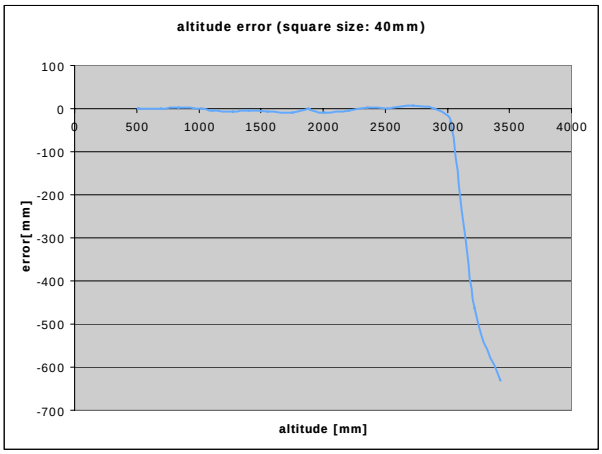


Figure 20: damier de 40mm

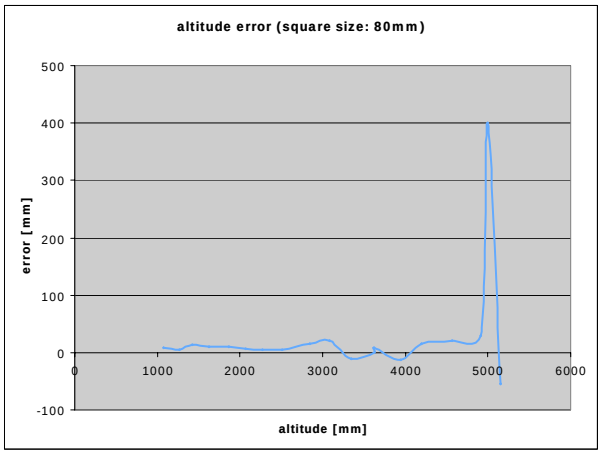


Figure 21: damier de 80mm

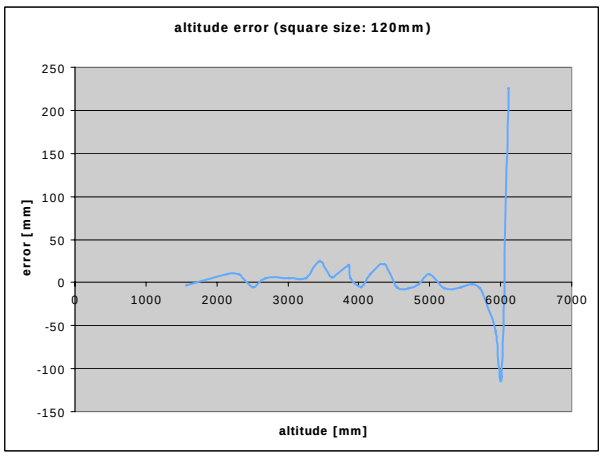


Figure 22: damier de 120mm

On peut observer que la taille des carrés n'influence pas l'erreur. Il change simplement le range.

Valeurs aberrantes:
Il arrive que l'algorithme sorte une valeur aberrante, celle-ci est filtrée en comparant la valeur avec la moyenne des trois dernières mesures. Si celle-ci est au-delà de **MAX_Z_JUMP**, elle n'est pas prise en compte.
Il arrive aussi que l'altitude ne puisse pas être calculée, par exemple si aucun coin de carré n'a été trouvé.
Dans les deux cas, si un précédent calcul a été fait, le même est repris (drapeau interpolation = 1) jusqu'à trois fois, puis une erreur est signalée (drapeau error = 1).

Les altitudes minimum et maximum dans lesquelles l'algorithme fonctionne correctement sont résumées dans le tableau ci-dessous:

Taille des carrés du damier [mm]	Range [mm]
40	500 – 3000
80	1000-4000
120	1500 – 6000

Tableau 2: relation taille des carrés - range

Le problème rencontré alors est l'approche de l'hélicoptère du sol. Le damier observé par la caméra est de plus en plus gros et une fois trop près l'algorithme ne voit plus assez de carrés pour trouver des coins.
Le problème qui se pose est que d'un côté la petite taille du damier permettrait de s'approcher plus du sol, mais de l'autre l'hélicoptère peut aller moins haut car il n'arrive plus à distinguer les carrés.

Etant donné que l'hélicoptère va évoluer en intérieur, la hauteur à laquelle il peut aller est limitée et donc le damier le plus petit (taille 40mm) est choisi.
Pour la mesure de l'altitude en dessous de 50cm, il faut trouver un système qui prend le relais (chap. 0). Pour la mesure du déplacement, on assume qu'à ce moment là l'hélicoptère est en mode d'atterrissage et donc son assiette est plate et il ne se déplace que verticalement.

10.8. Calcul du flux optique

Pour obtenir une information de quantité de déplacement de la caméra, il faut pouvoir tracker des points d'une image à l'autre. La librairie OpenCV comporte une fonction intégrée de tracking:

```
cvCalcOpticalFlowPyrLK(PreviousImage, GreyImage, prev_pyramid, pyramid,
old_corners, corners, numcorners, cvSize(win_size,win_size), 3, status, 0,
cvTermCriteria(CV_TERMCRIT_ITER|CV_TERMCRIT_EPS,20,0.03), flags );
```

On passe en paramètre à la fonction les coordonnées des points à suivre, puis elle trouve celles des points correspondants dans la nouvelle image.
Un tableau (**status**) de taille égale à la taille du tableau contenant les coordonnées des points permet d'avoir un drapeau pour savoir si chaque point a été suivi correctement. S'il ce n'est pas le cas, le point est peut-être sorti du champ de vision et il ne doit pas être pris en compte.

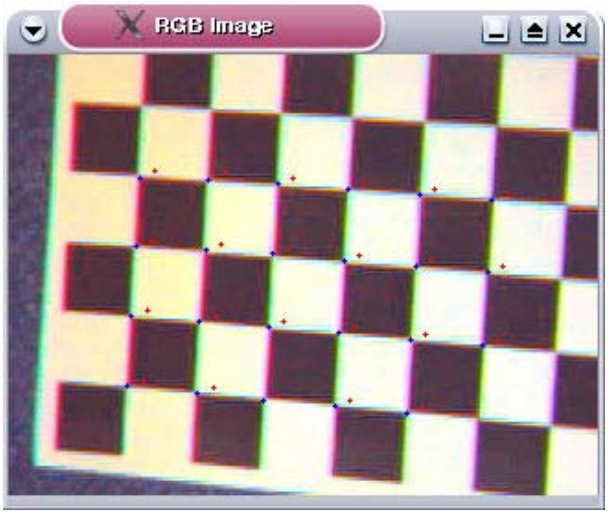


Figure 23: points trackés (bleu) et points correspondants sur l'image précédente (rouge)

Dans le cas d'une rotation selon z (yaw), tous les points ne se déplacent pas dans la même direction. C'est pourquoi avant tout chose il faut éliminer cette composante de rotation.

10.9. Calcul du yaw

Le principe utilisé est basé sur les angles des vecteurs trackés. En effet, si des points sont trackés, il en est de même pour des vecteurs. Ici chaque angle absolu de chaque vecteur est calculé par rapport au centre de rotation (qui peut être calibré avec **CENTER_POS_X** et **Y**), ensuite la différence est faite pour connaître le delta_yaw entre les deux images. La valeur est moyennée sur tous les vecteurs, ce qui permet une meilleure précision.

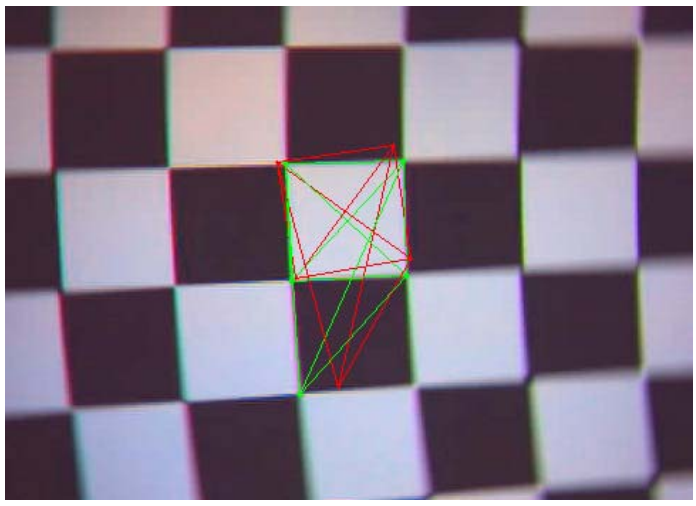


Figure 24: tracking des vecteurs (en rouge les vecteurs au temps *t-1*)

10.9.1. Tests

La précision est de l'ordre du centième de degré, à des vitesses mesurées jusqu'à 90°/sec environ. De plus la dérive n'est pas extrêmement élevée comme on peut le voir:

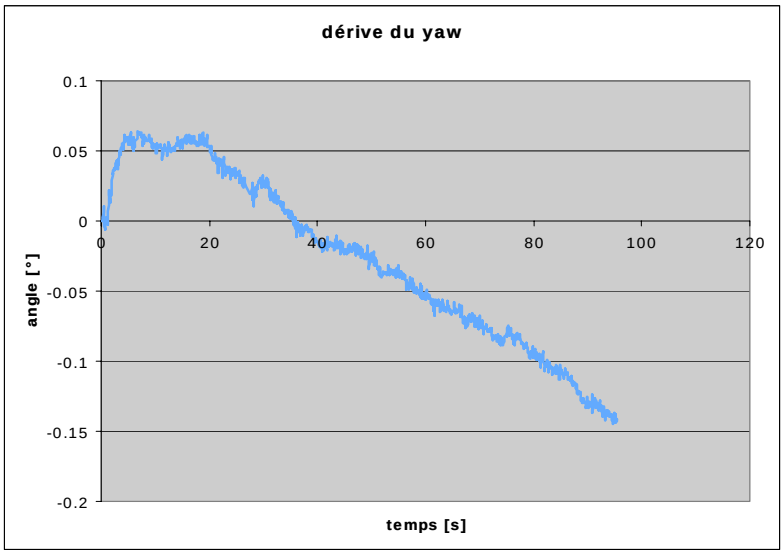


Figure 25: dérive du yaw en fonction du temps

Des tests de calcul du yaw ont été fait avec l'hélicoptère penché, mais aucune correction n'est à faire en dessous de 20° de pitch ou roll, le calcul reste fiable.

10.10. Calcul de dx et dy

Le fait que l'hélicoptère puisse tourner sur lui-même (yaw) engendre un déplacement observé par la caméra qui n'est pas voulu.
Le calcul du flux optique a noté que le point p1 s'était déplacé en p2. Ces informations sont erronées car en fait l'hélicoptère a tourné d'un angle Ψ et la translation réelle est celle du point p1' en p2. Pour corriger cela il faut calculer les coordonnées du point p1 après rotation de Ψ, puis soustraire la différence p1-p1' au déplacement observé.

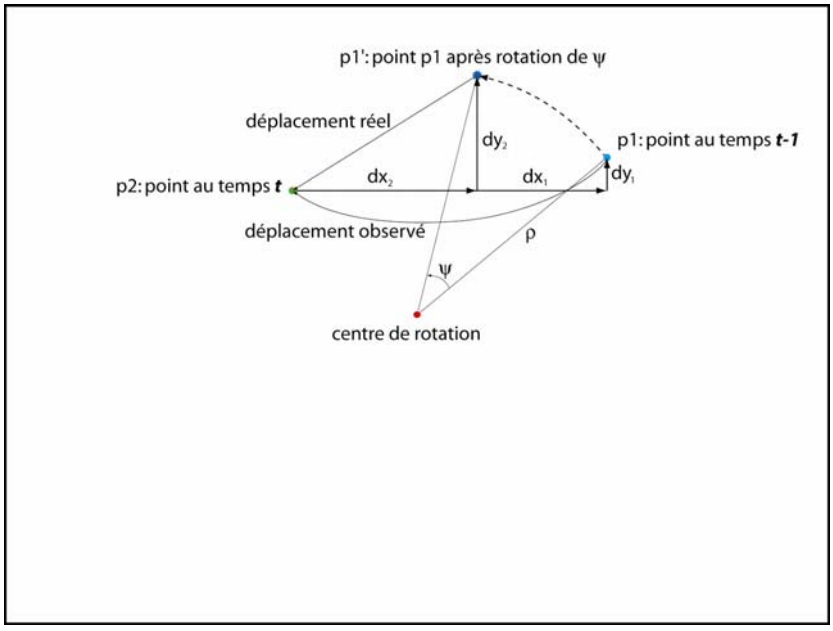


Figure 26: correction du déplacement dû à la rotation pure de l'hélicoptère

A partir de cet instant une moyenne peut être faite sur tous les points pour avoir le déplacement (en pixels).
La conversion depuis le déplacement en pixels au déplacement en millimètres se fait selon le rapport suivant:

déplacement_mm = déplacement_pixel * altitude / focale.

10.10.1. Tests

Résultats obtenus en déplaçant à la main sur une distance de 40cm:

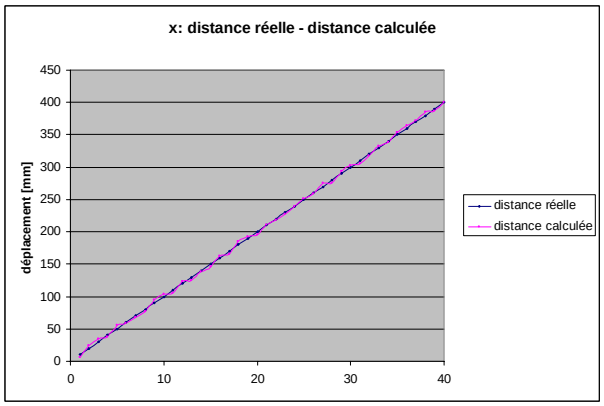


Figure 27: déplacement réel et mesuré en x

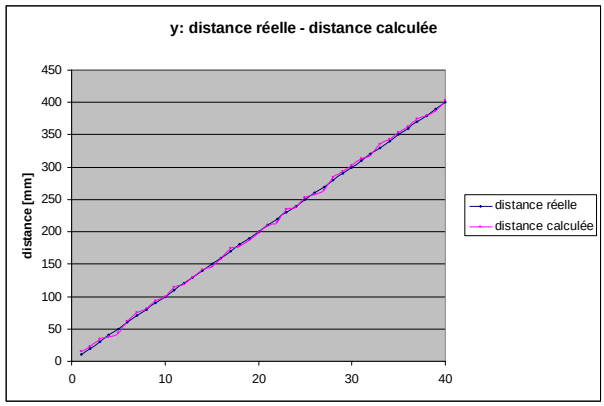


Figure 28: déplacement réel et mesuré en y

Les valeurs des déplacements sont correctes pour autant qu'on ne dépasse pas une certaine vitesse. Ceci vient du fait que d'une part la vitesse d'acquisition est limitée à 15fps et d'autre part le damier représente un motif périodique (voir chap. 15).

Tous les calculs faits jusqu'à présent supposent que l'hélicoptère est horizontal, ce qui n'est pas le cas en réalité. Le fait que sont assiette ne soit pas stable implique un mouvement perçu par la caméra qui doit être corrigé.

11. Fusion des données avec l'IMU

11.1. Correction de l'altitude

La première mesure à corriger est l'altitude. En effet si l'hélicoptère est penché, les carrés les plus proches seront un peux plus grand que ceux qui sont au loin. En considérant que ceux-ci sont présents sur toute la surface de l'image et que l'on fait la moyenne, aucune correction n'est à faire pour cela. Par contre le z calculé doit être projeté sur la verticale donc $real\_z = \cos(\alpha)\cos(\beta) \cdot z\_mesuré$ où α est le pitch et β le roll.

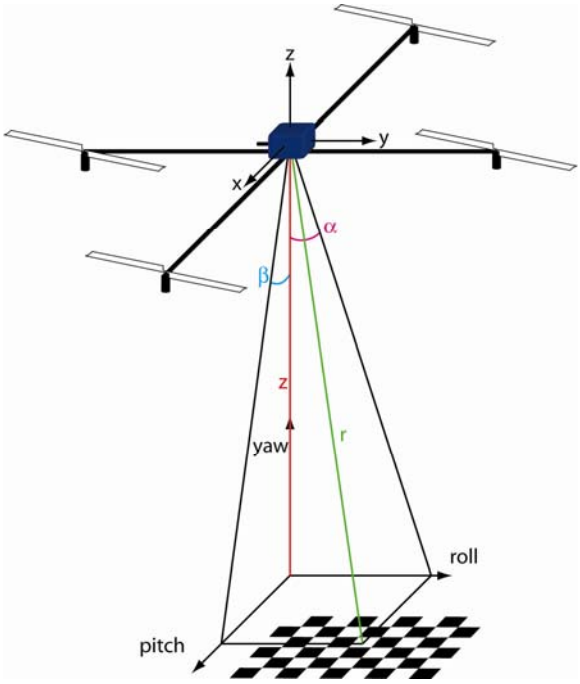


Figure 29: angles de l'hélicoptère

Les graphiques suivants montrent l'évolution de l'erreur de l'altitude en fonction de l'angle:

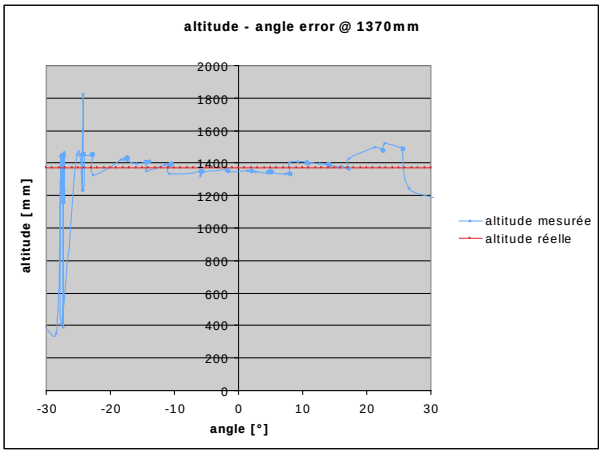


Figure 30: erreur de mesure de l'altitude

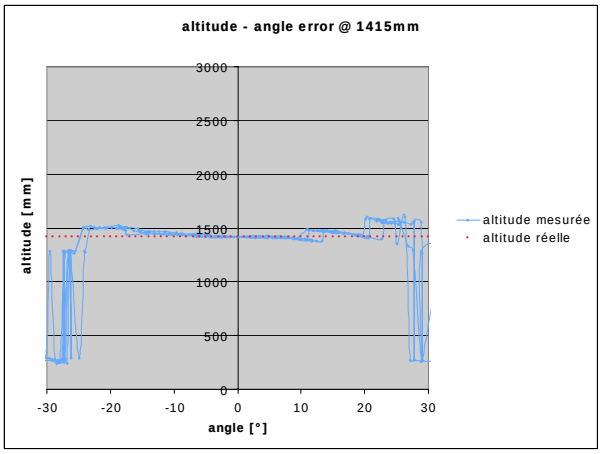


Figure 31: erreur de mesure de l'altitude

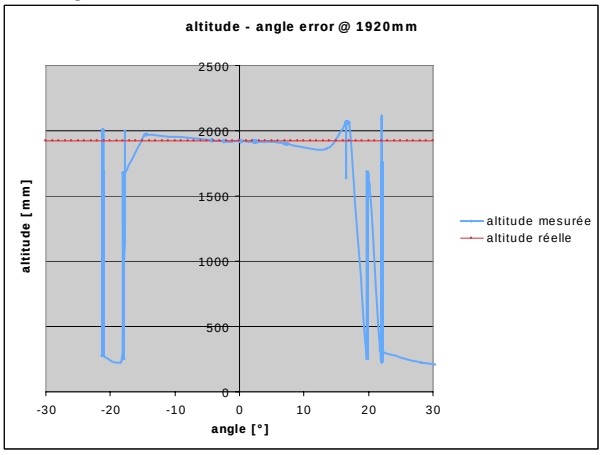


Figure 32: erreur de mesure de l'altitude

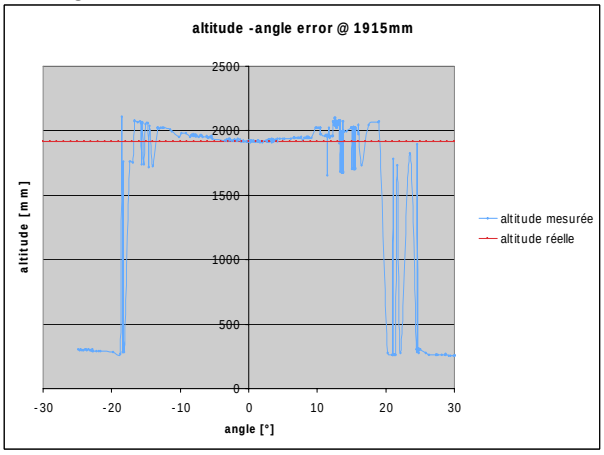
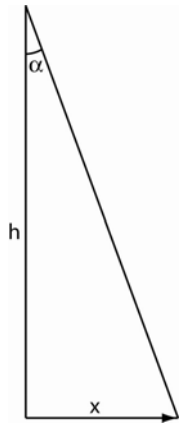


Figure 33: erreur de mesure de l'altitude

Les valeurs calculées pour l'altitude restent assez précises dans un intervalle de plus ou moins 20°, ce qui convient tout à fait étant donné que l'hélicoptère ne devraient pas atteindre ces valeurs extrêmes. On remarque qu'à haute altitude, les erreurs sont plus fréquentes à partir de 15°. De plus une légère dérive de forme quadratique semble s'ajouter à la mesure. Il est tout à fait possible de l'implémenter dans le code pour la corriger.

11.2. Correction de x et y

Une inclinaison de l'hélicoptère selon l'axe du pitch implique un déplacement de l'image selon y. Le principe est le même pour le roll et l'axe x.
Ce déplacement est fonction de l'angle et de l'altitude:



$$x = \arctan(\alpha) \cdot h$$

Il suffit de soustraire ce déplacement au déplacement observé avec la caméra.

12. Calcul des vitesses

Les vitesses sont calculées à partir des timestamps et des déplacements. Cependant avec les valeurs qui fluctuent beaucoup d'une image à l'autre, cela donne des vitesses instantanées trop élevées, il faut donc les moyenner sur un nombre déterminé par **SPEED_MOY_NUM**. Pour cela un tableau "roulant" de la taille de la constante garde en permanence les dernières mesures pour en calculer la moyenne. Le nombre de valeurs typique pour avoir une mesure stable est de 5.

13. Système d'approche

La mesure de distance avec les damiers ne permet pas d'approcher du sol en dessous d'une certaine limite (ici 500mm). C'est pourquoi un système parallèle doit prendre le relais lorsque le damier devient trop gros sur l'image et que l'algorithme ne peut plus calculer la distance. Il y a la possibilité d'utiliser un capteur, mais le but était de ne pas surcharger l'hélicoptère, le plus simple est d'utiliser le matériel et le software existant, en ajoutant juste un pointeur laser:

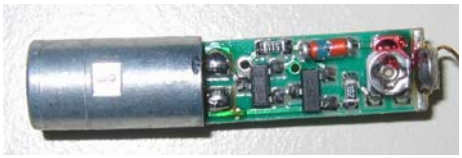


Figure 34: module laser

Ce petit module laser provient d'un stylo pointeur et pèse ~1g pour des dimensions de 5x30mm.

13.1. Principe de triangulation

Une triangulation permet à partir du point du laser sur l'image de retrouver l'altitude.
La distance *hmax* est fixée à 500mm puisqu'à partir de là les damiers sont utilisables.
Cela fixe donc avec l'angle d'ouverture α la distance *d* maximale à laquelle peut être placé le laser de la caméra: $d\max = \tan(\frac{\alpha}{2}) h\max = \tan(11) \cdot 500 = 88mm$.

Une fois le système monté, une calibration est à faire pour connaître la distance *b* (en pixels) du point à partir du bord de l'image lorsqu'on est à *hmax*.

Lors de l'utilisation, l'altitude se trouve par la formule $h = \frac{h\max \cdot a}{b}$, avec *a* la distance du point du bord trouvée sur l'image courante.

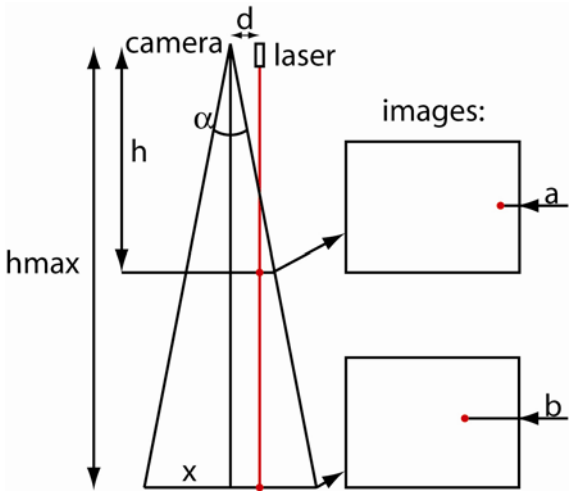


Figure 35: triangulation

13.2. Détection de la tache du laser

Pour détecter la tache du laser sur l'image, on peut recourir à l'espace de couleur YUV.
Dans le cas où la caméra fournit une image RGB, la conversion se fait à l'aide des formules suivantes:

- **Y = 0.299*R + 0.587*G + 0.114*B**
- **U = -0.169*R - 0.331*G + 0.500*B + 128.0**
- **V = 0.500*R - 0.419*G - 0.081*B + 128.0**

La conversion doit se faire pixel par pixel. La composante Y représente la luminosité, et les composantes U et V les couleurs. Il est possible alors de représenter dans l'espace U-V la couleur du laser:

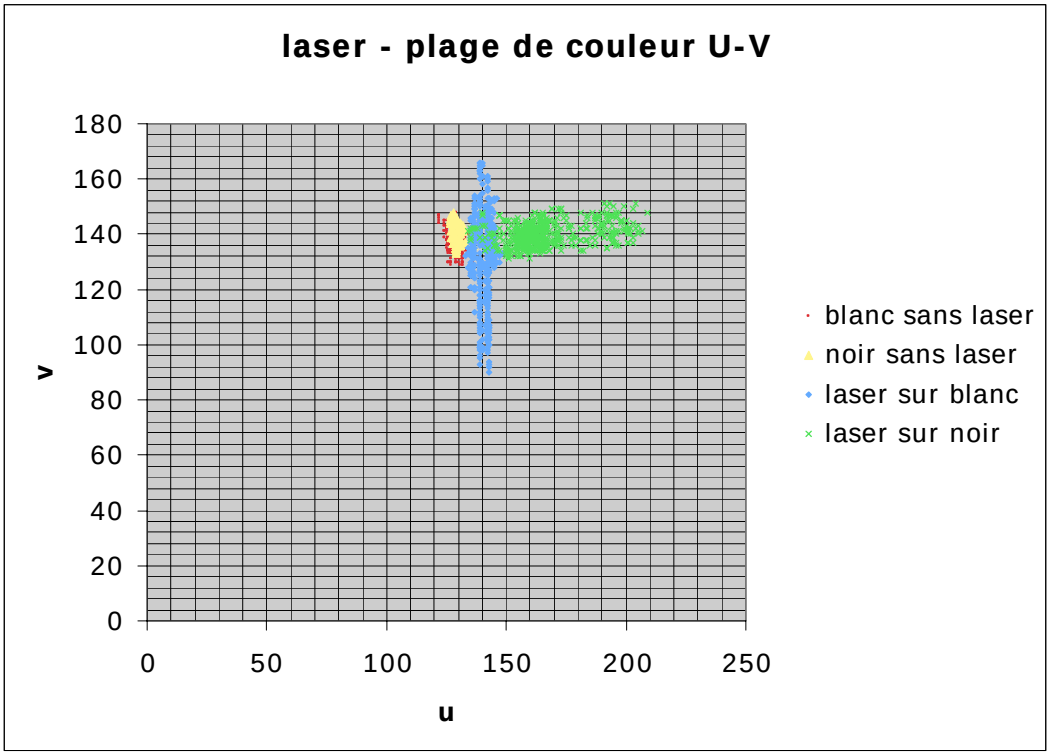


Figure 36: couleurs du laser dans l'espace U-V

La détection du point sur fond blanc fonctionne bien mais sur fond noir, il persiste des lignes parasites provenant de l'intersection des carrés blancs et noirs.
Malheureusement par manque de temps il a été impossible de résoudre ce problème et le système ne peut pas être terminé et testé.
Explication du probleme exacte

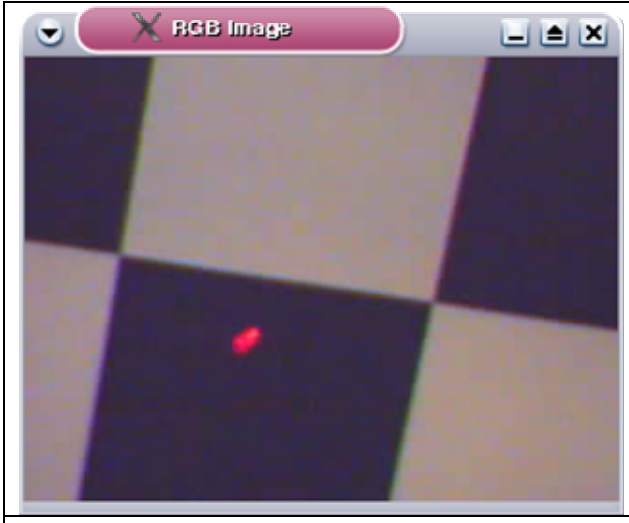


Figure 37: Laser sur fond noir

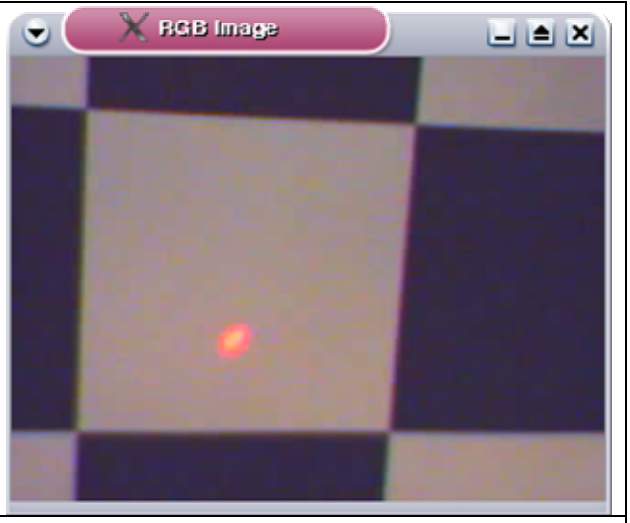


Figure 38: Laser sur fond blanc

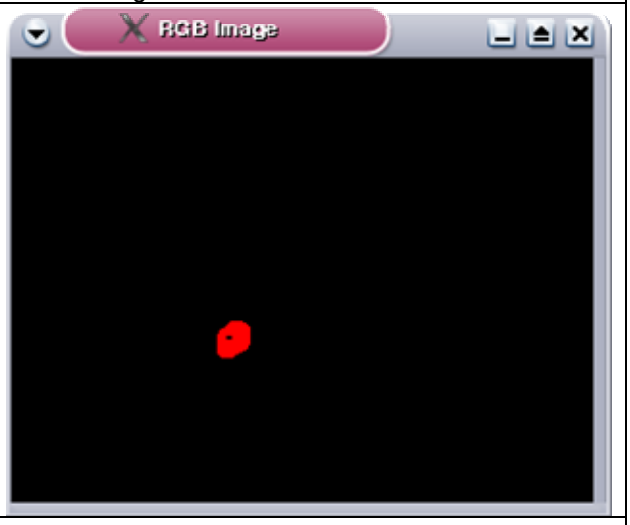


Figure 39: Laser sur fond blanc après détection

14. Implémentation du système sur l'hélicoptère

Le système complet a été testé sur la xboard. La vitesse du processeur se fait sentir sur la lenteur des calculs:

opération	Durée [s]
demande l'acquisition d'une image	0.001588
récupère l'image	0.06237
décompression JPEG	0.127694
correction des déformations	0.032388
recherche des coins	0.15289
calcul des distances	0.006251
calcul de l'altitude	0.001375
tracking des points	0.055319
calcul du yaw	0.002276
calcul de dx dy dz	0.001422

Figure 40: durée des différents calculs

Le résultat est un rafraîchissement à 2.26 Hz, ce qui est médiocre. On voit en rouge les opérations les plus coûteuses qui sont la décompression JPEG, le tracking des points et la recherche des points.
La décompression JPEG peut être évitée en utilisant un autre driver que l'OV519 ou une autre caméra. Cela amènerait la vitesse à 3.2Hz.
Une telle utilisation du processeur risque en plus de pénaliser le contrôleur de l'hélicoptère et mener à une catastrophe.
La solution sans devoir retoucher le hardware est de faire les calculs sur une machine distante, puis de récupérer les résultats. L'architecture du software est revue dans le chapitre suivant.

15. Architecture client/serveur

Le concept ici est d'avoir une application la plus légère possible sur la x-board (client) et tout le gros des calculs se fait du côté du serveur. Les informations sont échangées par une connexion directe TCP/IP.

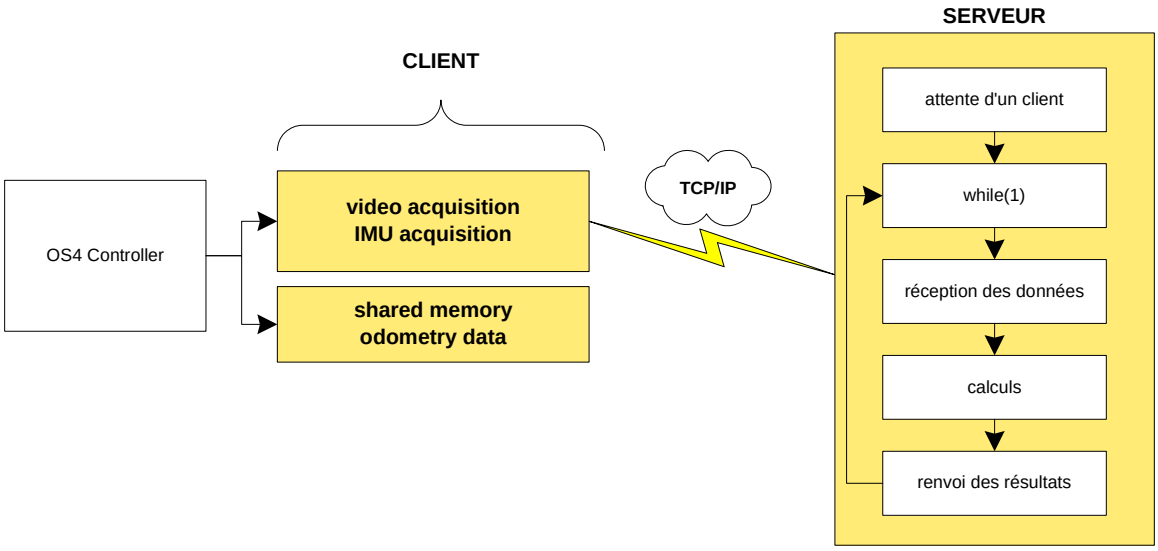


Figure 41: architecture avec calcul sur serveur distant

15.1. Client: librairie VisualOdometer

La structure du client est similaire à la première, à l'exception près qu'au lieu du traitement d'image, le programme envoie l'image et les données de l'IMU et reçoit en retour les résultats (dx, dy, dz, altitude, dyaw, erreurs). Il met à jour les données (x, y, z, vx, vy, vz, yaw, altitude) et passe à la boucle suivante.

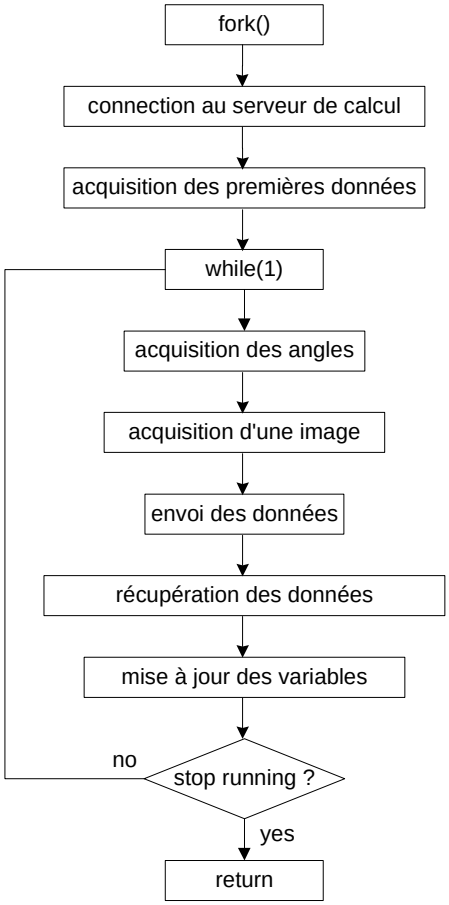


Figure 42: flux du client

15.2. Serveur: application ComputingServer

Ici aussi la structure est similaire à l'application de base, dans sa partie traitement d'image. Le serveur attend une connexion d'un client puis reçoit les données, les traite et renvoie les résultats au client.

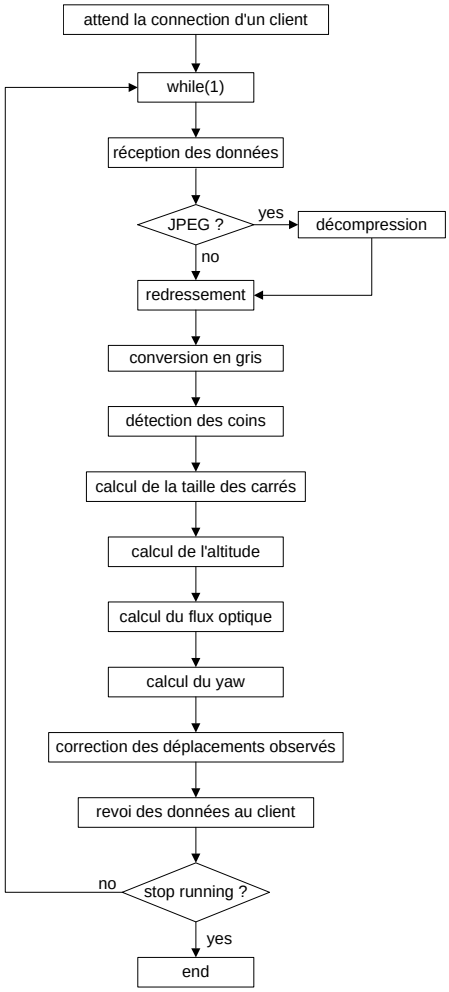


Figure 43: flux du programme serveur

15.3. Protocole de communication

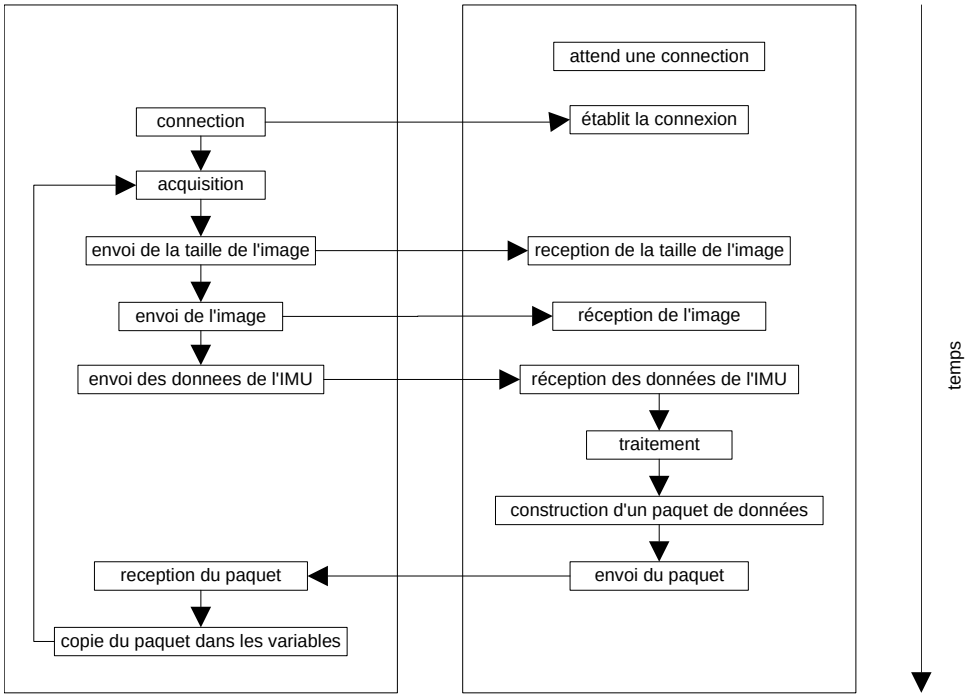


Figure 44: Protocole de communication

16. Tests finaux

16.1. Vitesses sur la xboard

16.1.1. Caméra Omnivision

Le même test qu'au chapitre 14 a été fait mais avec la nouvelle version du software. Les calculs sont faits sur un PC pentium 3 à 3GHz:

```
1108561193.256243 – decompressing JPEG
1108561193.260205 - image processing 2: Find corners
1108561193.264446 - image processing 3: Calculating distances.
1108561193.265578 - image processing 4: Finding square size and Z.
1108561193.265583 - image processing 5: Caluclates optical flow.
1108561193.266746 - image processing 6: Calculate yaw.
1108561193.266849 - image processing 7: Calculates dx-dy-dz.
1108561193.266884 - End of calculation.
Processing time: 0.068892 (14.515 FPS)
1108561193.266894 475.499605 -60.338071 3611.030433
1108561193.325135 – decompressing JPEG
1108561193.329092 - image processing 2: Find corners
1108561193.333311 - image processing 3: Calculating distances.
1108561193.334669 - image processing 4: Finding square size and Z.
1108561193.334674 - image processing 5: Caluclates optical flow.
1108561193.335881 - image processing 6: Calculate yaw.
1108561193.335997 - image processing 7: Calculates dx-dy-dz.
1108561193.336035 - End of calculation.
Processing time: 0.069151 (14.461 FPS)
```

On voit ici que si la puissance de calcul n'est pas critique, la vitesse approche celle d'acquisition de la caméra (ici 15 FPS).

16.1.2. Caméra Philips

Des tests ont été faits avec une autre caméra: une Philips avec le driver PWC. Cette caméra fourni l'image au format YUV, donc il n'y a pas besoin de décompresser le JPEG, et pas besoin de convertir l'image RGB en gris, car la composante Y peut directement être utilisée comme composante de gris.
Résultats affichés sur le serveur de calcul:

```
1108562476,122374 - network: receiving data.
1108562476,140587 - image processing 2: Find corners
1108562476,141996 - image processing 3: Calculating distances.
1108562476,149547 - image processing 4: Finding square size and Z.
1108562476,149650 - image processing 5: Caluclates optical flow.
1108562476,150697 - image processing 6: Calculate yaw.
1108562476,151096 - image processing 7: Calculates dx-dy-dz.
1108562476,151227 - End of calculation.
Processing time: 0.028853 (34.658 FPS)
1108562476,162321 - network: receiving data.
1108562476,170583 - image processing 2: Find corners
1108562476,172253 - image processing 3: Calculating distances.
1108562476,179387 - image processing 4: Finding square size and Z.
1108562476,179397 - image processing 5: Caluclates optical flow.
1108562476,180356 - image processing 6: Calculate yaw.
1108562476,180628 - image processing 7: Calculates dx-dy-dz.
1108562476,180681 - End of calculation.
```

```
Processing time: 0.029454 (33.951 FPS)
```

On voit ici que les résultats sont bien meilleurs qu'avec le système de base.

16.2. Test de l'odométrie

16.2.1. En mouvement

Les tests ont été réalisés à l'aide d'un robot d'où l'on pouvait sortir l'odométrie et la comparer à l'odométrie visuelle. L'acquisition est faite avec la caméra Omnivision donc la vitesse est limitée à 15 fps. Le traitement est fait sur un PC portable.

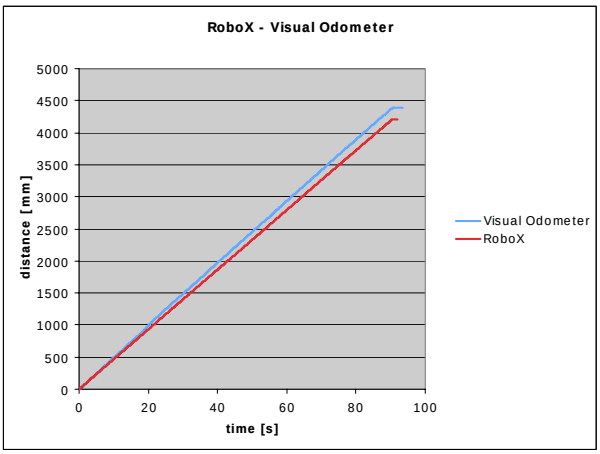


Figure 45: odométrie à 46mm/s

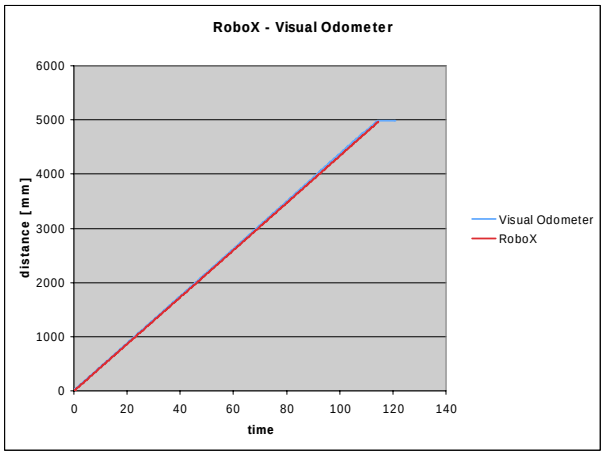


Figure 46: odométrie à 43mm/s

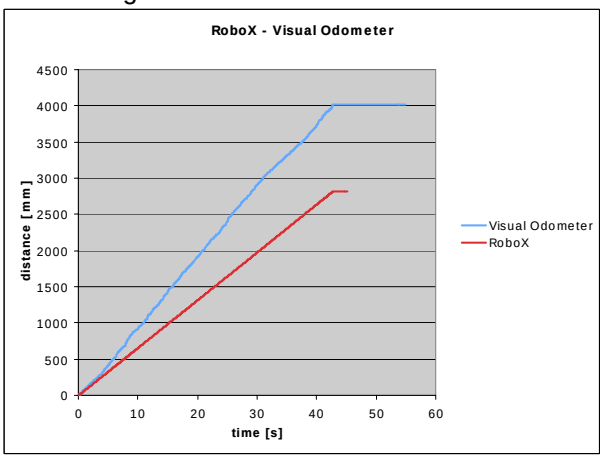


Figure 47: odométrie à 60mm/s

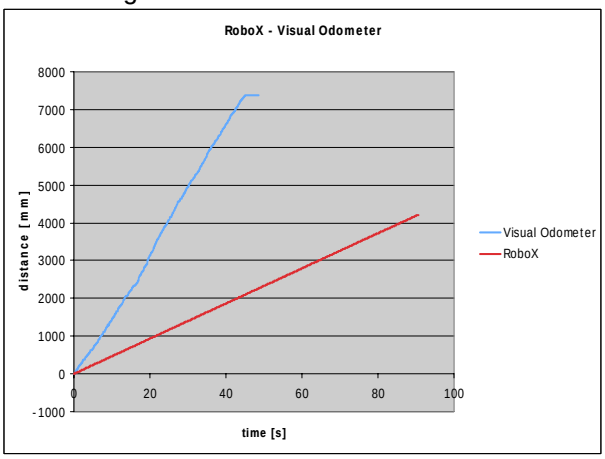


Figure 48: odométrie à 46mm/s

16.3. Vitesse limite

D'après les graphes, on dirait que la limite en vitesse du système est atteinte. A environ 40mm/s l'odométrie est excellente, et lorsqu'on monte l'erreur devient plus grande. Il est intéressant de noter que l'erreur est linéaire, c'est peut-être du à un phénomène de stroboscopie que subit la caméra qui voit le motif répétitif du damier. Des tests approfondis pourraient confirmer cette théorie.

Si on suppose que la vitesse maximum de déplacement est de 40mm/sec pour une vitesse d'acquisition de 15 images par seconde, on peut déterminer à quelle fréquence il faudrait acquérir les images pour se déplacer à une certaine vitesse.
Les erreurs viennent de la taille des carrés, qui sont confondus si le déplacement devient trop grand entre deux images, le tracking se "trompe".

Soit la vitesse d'acquisition et la vitesse de déplacement. On a le déplacement maximal entre deux images de 40mm/15fps = 2.6mm. En faisant le calcul de l'altitude à l'envers, pour trouver le déplacement en pixel on a $d\max|_{pixels} = d\max|_{mm} \frac{focale}{altitude}$.

La taille d'un carré calculée à partir de la distance des coins vaut:

$$SquareSize|_{pixels} = \frac{CALIB\_SQUARE\_SIZE \cdot SQUARE\_REAL\_SIZE}{CALIB\_SQUARE\_REAL\_SIZE} \cdot \frac{CALIB\_DISTANCE}{altitude}$$

$$= \frac{179.67 \cdot 40}{120} \cdot \frac{535}{684.45} = 46.81|_{pixels}$$

Le déplacement maximal entre deux images est

$$d\max|_{pixels} = d\max|_{mm} \frac{focale}{altitude} = 2.6 \frac{800}{684.45} = 3|_{pixels} = \frac{1}{15.4} \cdot SquareSize$$

On peut alors établir la relation entre le déplacement maximum et la taille des damiers:

$$d\max|_{pixels} = K \cdot SquareSize$$

$$d\max|_{mm} \frac{focale}{altitude} = K \frac{CALIB\_SQUARE\_SIZE \cdot SQUARE\_REAL\_SIZE}{CALIB\_SQUARE\_REAL\_SIZE} \cdot \frac{CALIB\_DISTANCE}{altitude}$$

avec $K = \frac{1}{15.4}$

On voir que le déplacement maximum ne dépend pas de l'altitude !

$$d\max|_{mm} = \frac{CALIB\_SQUARE\_SIZE \cdot SQUARE\_REAL\_SIZE}{CALIB\_SQUARE\_REAL\_SIZE} \cdot \frac{CALIB\_DISTANCE}{focale}$$

$$\Rightarrow FPS = \frac{d\max|_{mm}}{vitesse\_voulue}$$

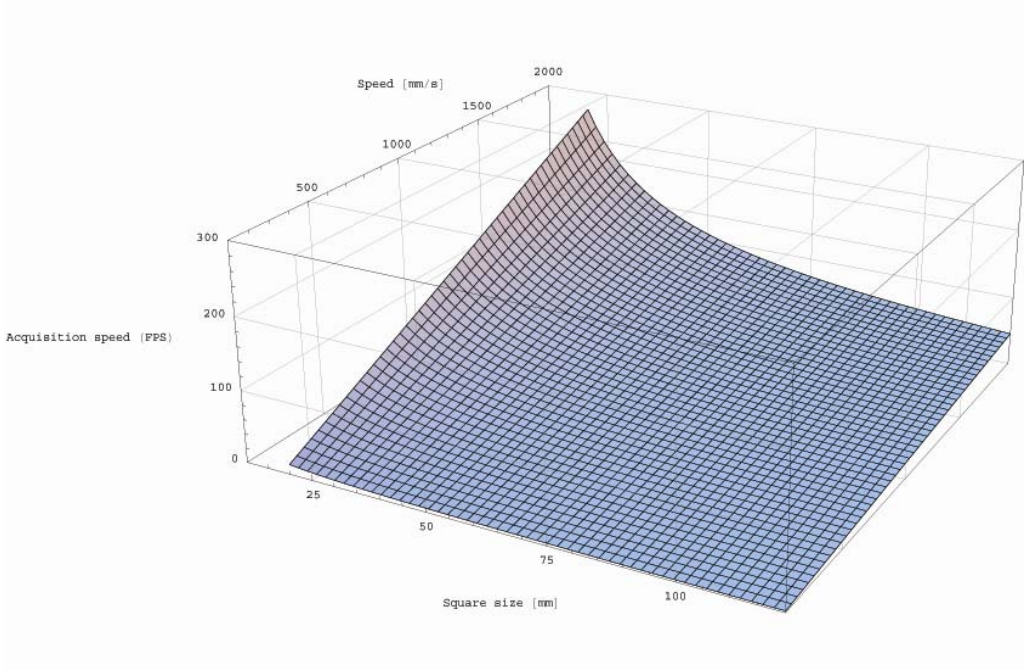


Figure 49: vitesse d'acquisition en fonction de la taille des damiers et de la vitesse du robot

Typiquement pour atteindre une vitesse de 2 [m/s] avec des damiers de 40mm, il faudrait une vitesse d'acquisition de 150 fps!

16.3.1. dérive en mode stationnaire

Le système à été posé et les mesures prise en mode stationnaire. On voit que la dérive est à peut près nulle même si on observe un bruit autour de zéro:

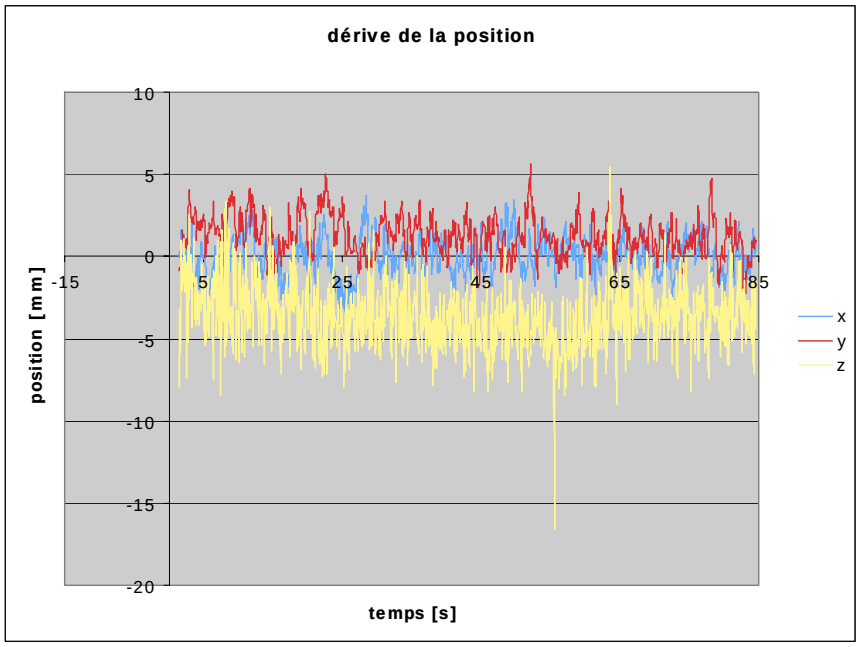


Figure 50: dérive de la position en fonction du temps

17. Discussions

17.1. Résultats

Les résultats obtenus aux cours des expériences ont montré que le système fonctionne, pour autant qu'on reste dans les limites d'utilisation qui est la vitesse de déplacement. Le moyen d'améliorer ce point est d'augmenter la vitesse de la caméra.

17.2. Développements futur

17.2.1. Nouveau driver OV519 sans compression JPEG

Comme expliqué dans le chapitre 5.3, le driver de la caméra OmniVision ne permet pas de monter à plus de 15FPS sous linux, et de plus il compresse les images en JPEG. Le capteur d'image OV7648FB, quant à lui, fournit les images en YUV et peut monter jusqu'à 60fps, donc une nouvelle électronique pourrait considérablement augmenter les performances.

17.2.2. Nouveau PCB pour la caméra plus petit et plus léger

Le PCB existant est assez gros et épais, et surtout le connecteur USB est énorme. Il pourrait être remplacé par un plus petit, voir par un connecteur non-USB et un câble non-blindé, si la distance n'est pas trop grande cela peut fonctionner et le gain en poids serait non-négligeable.

18. Conclusion

Le projet s'est bien déroulé dans l'ensemble. Les milestones ont été respectés dans l'ensemble, aucun retard n'a été pris sauf juste à la fin en ce qui concerne le laser. Il a été montré que le calcul embarqué sur l'hélicoptère n'est pas encore possible pour l'instant, la quantité d'informations à traiter est trop élevée à ce jour.
D'après les expériences, un système d'odométrie visuelle peut être extrêmement précis, même avec une caméra moyenne qui ne fournit pas une bonne résolution. Par contre cela demande une grande vitesse d'acquisition pour pouvoir suivre les mouvements, surtout si ceux-ci sont rapides.
Ce projet a été très intéressant car il a permis de se sensibiliser avec les contraintes que posent le domaine des embarqués, de plus sur un robot volant.
Je tiens à remercier S. Bouabdallah pour son dévouement envers les étudiants du laboratoire, qui était toujours là quand on avait besoin de lui. P. Lamon, pour ses nombreux conseils en ce qui concerne la vision, D. Burnier, aussi toujours là si on a besoin de quoi que ce soit de matériel, F. Pont, qui a peut mesurer sont odométrie à l'odométrie visuelle. Enfin, X. Raemy et Y. Stauffer pour le temps qu'ils ont consacré à m'aider à résoudre mes problèmes.
Pour finir je remercie le professeur R. Siegwart pour m'avoir permis de faire ce projet de diplôme dans son laboratoire.

19. Annexes :

19.1. Références

[1] D. Nistér, O. Naroditsky, J. Bergen : "Visual Odometry", *Sarnoff Corporation*, 2004

[2] D. Nistér: "Preemptive RANSAC for Live Structure and Motion Estimation", Sarnoff Corporation, 2003

[3] J.-C. Zufferey: "Toward 30-gram Autonomous Indoor Aircraft: Vision-based Obstacle Avoidance and Altitude Control", Autonomous Systems Lab, Federal Institute of Technology in Lausanne (EPFL), 2004

[4] M. G. Nagle, M. V. Srinivasan, D. L. Wilson: "Image interpolation technique for measurement of egomotion in 6 degrees of freedom", *Australian National University*, 1997

[5] M. V. Srinivasan: "An image-interpolation technique for the computation of optic flow and egomotion", *Australian National University, 1994*

[6] J.-C. Zufferey, D. Floreano: "Fly-inspired Visual Steering of an Ultra-light Indoor Aircraft", *EPFL-ASL, 2004*

[7] M. McClelland : "Linux OVCam Drivers", http://alpha.dyndns.org/ov511/

[8] J.-Y. Bouguet: "Camera Calibration Toolbox for Matlab", http://www.vision.caltech.edu/bouguetj/calib_doc/

[9] SourceForge: "Open Computer Vision Library", http://sourceforge.net/projects/opencvlibrary/

19.2. Datasheet de l'OV519

19.3. Datasheet du FB...

19.4. Schéma éléctronique du PCB

19.5. HowTo – compilation et installation du noyau et modules linux

19.6. HowTo – compilation et installation de la libraire opencv

- Décompresser la librarie :

tar xvfz libopencv-0.9.6.tar.gz

- Changer de répertoire et configurer le répertoire d'installation
- Cd libopencv-0.9.6
- Configure –prefix=<répertoire d'installation>
- Compiler et installer la librairie
- Make ; make install

19.7. HowTo – compilation de la librairie visualodometry

19.8. Calibration de la caméra

19.9. Calibration de Z

19.10. Calibration du laser