École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique

Rapport de projet de fin d'études

Approche multimodale pour le support temps réel d'algorithmes de traitement d'images sous systèmes mobiles

Étudiant :
Kévin TOUBLANC (kevin.toublanc@etu.univ-tours.fr)

Encadrant :
DELALANDRE Mathieu

Année Étude :
2011 – 2012

École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique

Rapport de projet de fin d'études

Approche multimodale pour le support temps réel d'algorithmes de traitement d'images sous systèmes mobiles

Étudiant :
Kévin TOUBLANC (kevin.toublanc@etu.univ-tours.fr)

Encadrant :
DELALANDRE Mathieu

Année Étude :
2011 – 2012

Sommaire

Introduction

Ce rapport s'inscrit dans le cadre du projet de fin d'études portant sur la conception d'une approche multimodale supportant le temps « réel » d'algorithmes de traitement d'images sous iPad.

Le projet fût encadré par Mathieu DELALANDRE et réalisé par Kévin TOUBLANC, étudiant en dernière année à Polytech'Tours.

Le but de cette étude et de concevoir un système permettant d'alléger la complexité des algorithmes de traitements d'images.

Le fil conducteur de ce projet est l'utilisation des nouvelles technologies présentent dans les smartphones et les tablettes tactiles : les capteurs de mouvements.

Dans un premier temps, ce document expliquera les aspects systèmes du projet : acquisitions des données matérielles et ordonnancement.

Il vous sera ensuite exposé le travail réalisé sur l'intégration de ces capteurs dans un algorithme de traitement d'images.

Partie 1 : Généralités
Présentation du projet

1. Présentation du projet

Mon projet de fin d'études (PFE) se place dans le domaine du traitement d'images, il vise à supporter les applications de traitement d'images en incluant une nouvelle méthode innovante : la gestion des périphériques de capture de mouvements.

- Gyroscope
- Accéléromètre

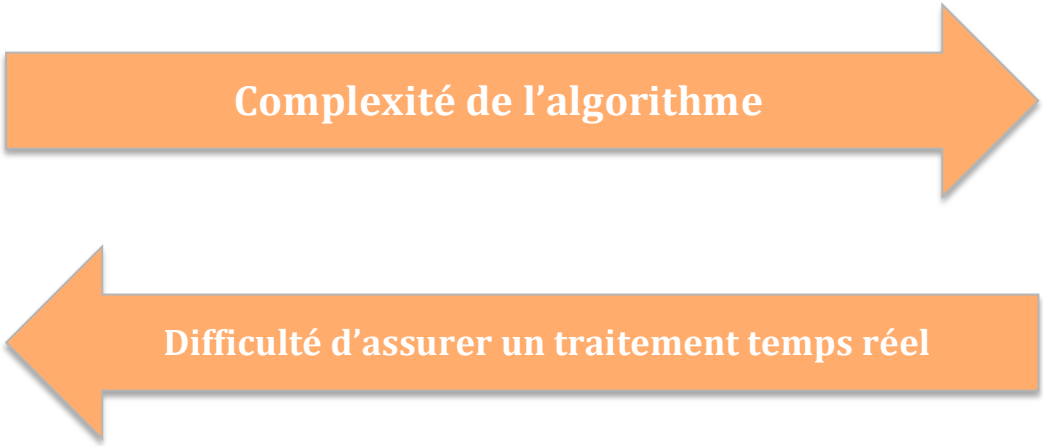
Ces deux capteurs permettront la localisation de points dans l'espace afin de pouvoir déterminer la position des différentes images à traiter.

1.1. Présentation générale du projet

Dans le domaine du traitement d'images sous système mobile, le champ d'action du développeur est limité par :

- La puissance de calcul de l'appareil,
- La mémoire RAM du mobile,
- La taille des capteurs vidéo (supérieurs à 5 mégapixels).

Bien que ces deux points s'améliorent à chaque nouveaux téléphone ou tablette, ils ne permettent pas un support d'applications de traitement d'images utilisant un algorithme lourd :



Pour supporter le temps réel, deux méthodes différentes existent :

- Filtrage du nombre d'images à traiter,
- Filtrage de l'image afin d'en dégager une plus petite portion traitable en temps réel.

Mon projet constitue une méthode pour filtrer la quantité d'informations en détectant les portions d'images déjà traitées. En effet, lors d'une capture vidéo (25 images par secondes), les images se recouvrent. Il est donc inutile de retraiter une image à peine 40 millisecondes après son traitement initial.

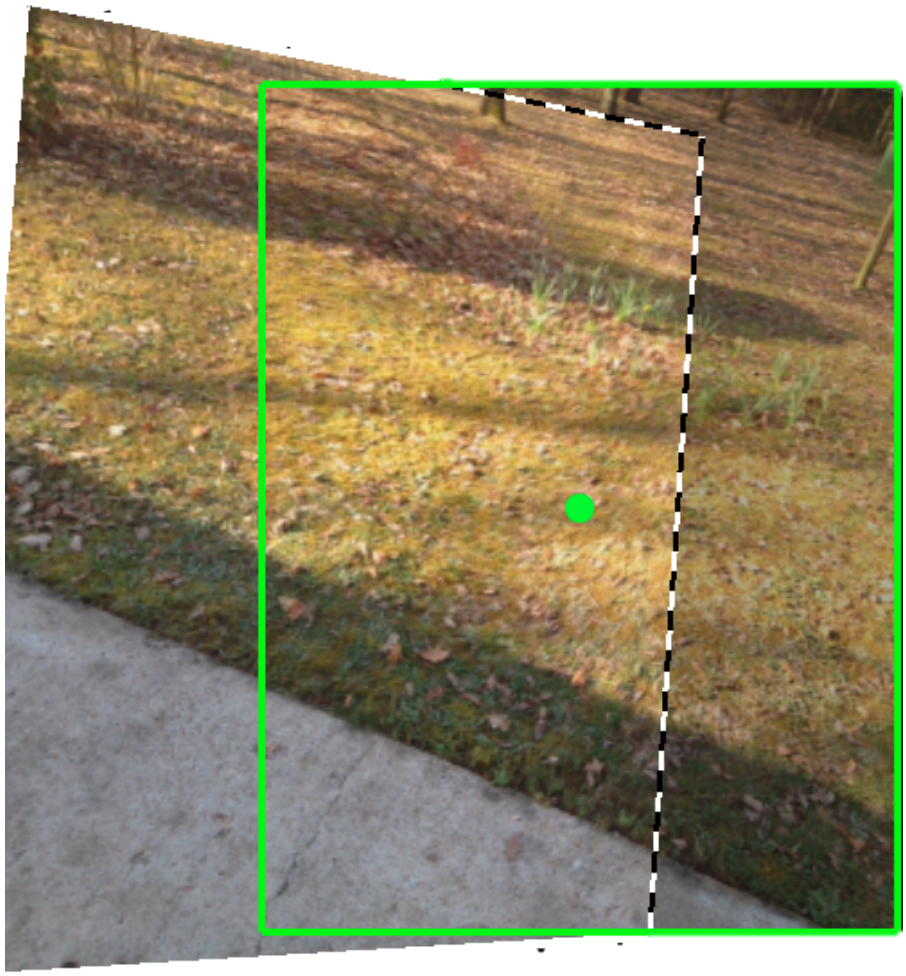


Figure 1 Illustrations de deux images d'un flux vidéo

Comme le suggère l'image en Figure 1, deux images successives d'un flux vidéo comportent une part non négligeable d'informations identiques. En Localisant les images dans l'espace, il est alors possible de déterminer les recouvrements entres les deux images et, d'agir en conséquence.

Pour être en mesure de déterminer la position d'image dans l'espace, il a été nécessaire d'étudier les différents capteurs mis à notre disposition par le constructeur Les différentes parties de ce rapport visent à expliciter les fonctionnements des capteurs et leurs utilisations.

1.2. Intérêt de l'approche

L'intérêt de l'approche proposée est de simplifier l'application d'algorithmes temps réel en sélectionnant des parties d'images à traiter. Le projet a pour but de réaliser une étude sur le facteur de diminution de la taille des images à traiter.

1.3. Mise en application

La mise en application sera faite par un algorithme de détection de points d'intérêts : Laplacien of Gaussian. Cet algorithme traite tous les pixels de l'image et s'exécute en quelques secondes. Il est donc impossible de traiter toutes les images dans leur totalité. Les points détectés seront projetés de frames en frames afin de permettre la continuité de l'algorithme et le traitement de toutes les frames.

1.4. Prérequis matériel

L'application a été créée pour fonctionner avec un iPad de deuxième génération. Il est nécessaire de lancer l'application sur un iPad 2 ou une version plus récente afin qu'il possède un gyroscope et une caméra (tous deux non-présents dans la première version).

Aussi, la reprise du code fournis nécessite le développement sur un ordinateur Apple, possédant la version 5.1 du Framework de développement. (Actuellement non disponible sur les ordinateurs de l'école).

1.5. Découpage fonctionnel du projet

L'étude réalisée au cours ce projet se décompose en trois parties :

- L'étude des composants systèmes,
- L'étude des projections entre frames,
- Le traitement d'images

Chacune des parties sont explicitées dans la suite du rapport.

Partie 2 : Système d'exploitation Architecture système et acquisitions matérielles

1. Ordonnancement sous iOS

1.1. Ordonnancement standard des systèmes BSD

Les informations de cette partie sont extraites de l'ouvrage « Operating Systems – Internal and Design Principles » (Stallings).

Mac OS X et iOS sont basés sur le système BSD, à ce titre, ils utilisent un ordonnanceur de type « Multi-level Feedback Queue ». Cet ordonnanceur permet l'utilisation de plusieurs files de traitements correspondants à une priorité (ou un intervalle de priorités). Les files sont exécutées suivant un algorithme de « Time-Slice » (Quantum de temps). Dans le cas de Mac OS X, le quantum de temps est basé sur 1 seconde préemptive.

Chaque file de traitements possède son propre algorithme d'ordonnancement. Selon les diverses recherches effectuées, il apparaît que l'algorithme utilisé est le Round Robin (RR). Le RR alloue un quantum de temps à chaque processus de la file sans se soucier de leurs priorités.

Les priorités de chaque processus sont recalculées chaque seconde suivant les formules suivantes :

$$CPU_j(i) = \frac{CPU_j(i-1)}{2}$$

$$P_j(i) = Base_j + \frac{CPU_j(i)}{2} + nice_j$$

- $CPU_j(i)$: Mesure du temps processeur utilisé par le processus j pendant l'intervalle de temps i,
- $P_j(i)$: Priorité du processus j pendant l'intervalle de temps i,
- $Base_j$: Priorité d'origine du processus j,
- $nice_j$: Facteur de contrôle ajustable par l'utilisateur.

Ainsi, un processus voit sa priorité augmenter s'il est exécuté et décrémenté s'il est en attente d'exécution. Ce système permet donc au processus de changer de file de traitements en fonction de son temps d'exécution.

1.2. Utilisation de l'ordonnanceur

La parallélisation des traitements sous iOS peut être effectuée avec trois méthodes distinctes :

- L'utilisation de threads utilisateurs,
- L'utilisation de Grand Central Dispatch (GCD),
- L'utilisation de queues de traitements.

L'utilisation de thread n'a pas été étudiée lors de ce PFE et ne sera pas explicitée. Les deux parties suivantes expliqueront le fonctionnement de GCD et des queues de traitements.

1.2.1. Grand Central Dispatch

GCD est un composant logiciel permettant une gestion totale des threads. Ce module est écrit en C et possède donc la gestion de plus bas niveau (du point de vue du programmeur) de la parallélisation des tâches.

Il fonctionne grâce à la déclaration de file de répartition (« DispatchQueue ») permettant d'y stocker des tâches. Ces files de répartitions possèdent des propriétés configurables telles que :

- La définition de la concurrence des tâches,
- Le niveau de priorité de la queue.

La particularité de l'utilisation de GCD est la gestion automatique des threads : lors de l'exécution d'une tâche, GCD décide de l'utilisation du thread :

- Si un thread est disponible (aucun travail en cours) alors GCD l'utilise.
- Si aucun thread n'est disponible, alors GCD créera un nouveau thread spécialement pour l'exécution de la tâche.

Grand Central Dispatch est donc un composant logiciel permettant de s'abstraire de toute gestion manuelle de thread.

1.2.2. Queues de traitements

Les queues de traitements sont une particularité de l'API Cocoa. Elles permettent, tout comme GCD de s'abstraire de la gestion de thread. La seule différence est que les classes permettant leurs utilisations sont écrites en Objective-C (surcouche objet du C) et sont donc de plus haut niveau.

Les deux classes remarquables sont :

- NSOperationQueue : Classe représentant la file de traitements,
- NSOperation : Classe représentant une tâche à placer dans la file de traitements.

2. Direct Memory Access

Le Direct Memory Access (DMA) est une solution matérielle permettant de faciliter les opérations de transfert de données entre la mémoire centrale du système et les périphériques.

2.1. Principe d'accès direct à la mémoire

Le principe permet de libérer le CPU de tout transfert de données vers la mémoire, permettant ainsi de ne pas le saturer et de lui confier des tâches plus complexes.

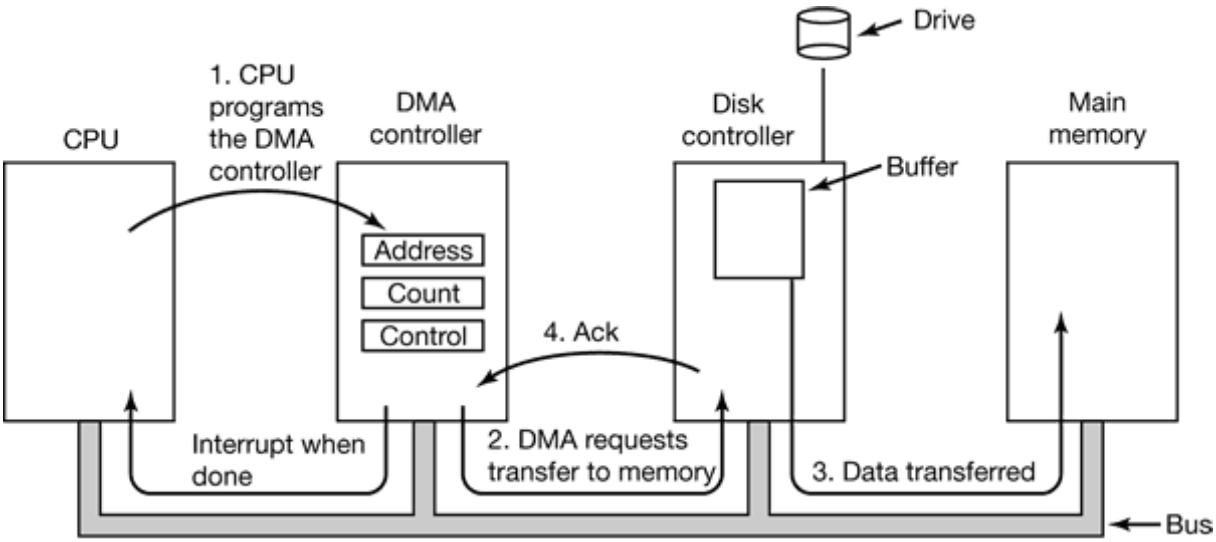


Figure 2 Illustration du principe DMA (Tanenbaum, 2001)

Le schéma ci-dessus illustre les actions effectuées lors de la configuration du DMA dans un système.

2.1.1. Programmation du contrôleur DMA

Cette étape permet la configuration du contrôleur afin qu'il puisse répondre à la question : « Qu'est-ce que je transfert, et où ? ». Le CPU apporte donc les éléments nécessaires à sa configuration comme :

- Quel périphérique utiliser,
- Vers quelle zone mémoire transférer les données,
- Le nombre de transferts de données désiré.

2.1.2. Requête de transfert vers la mémoire centrale

Après avoir été correctement configuré, le contrôleur DMA émet une requête de lecture vers le contrôleur de périphérique. Dans le cas de l'exemple, ce-dernier est représenté par le contrôleur de disque dur.

2.1.3. Transfert de données

Lors de la réception de la requête, le contrôleur de disque lit les données présentent sur le disque physique et les transferts vers la mémoire indiquée par le contrôleur DMA.

Lorsque la tâche est accomplie et vérifiée (éventuellement à l'aide d'un checksum sur l'ensemble des données), le contrôleur de périphérique émet un acquittement vers son prédécesseur.

2.1.4. Notification du CPU

Lors de la réception de l'acquittement (étape 2.1.3), le contrôleur DMA décrémente son compteur et, si celui-ci est égal à zéro, lance une interruption a destination du CPU. Sinon, les étapes 2.1.2 à 2.1.3 sont répétées.

2.2. Direct Memory Access sous iOS

2.2.1. Principe de fonctionnement

Cette sous-partie est consacrée à l'utilisation du DMA du point de vue de l'utilisateur. Sous iOS, nous retrouvons les étapes principales du fonctionnement vue dans la section précédente, seul l'étape de la notification du CPU diffère et peut être réalisée de différentes façons :

- Grâce à l'utilisation de GCD,
- Grâce à l'utilisation d'une queue de traitements,
- Sans aucune interruption vers le programme.

La figure ci-dessous illustre le fonctionnement avec l'utilisation d'une queue de traitements. Quel que soit le procédé utilisé, la configuration reste (au code près) la même. Le thread principal (ou la queue de traitement principale) de l'application configure le contrôleur DMA en lui fournissant les paramètres suivants :

- Le périphérique à utiliser,
- La zone mémoire dédiée,
- Les informations de configuration du contrôleur de périphérique.

Après les étapes classiques, le contrôleur DMA lève une interruption vers le CPU qui choisira alors la méthode d'interruption utilisateur à réaliser en fonction de sa configuration. Il en résultera donc trois options :

- Soit une interruption vers une queue de traitements de type NSOperationQueue,
- Soit une interruption vers Grand Central Dispatch,
- Soit aucune interruption pour l'utilisateur.

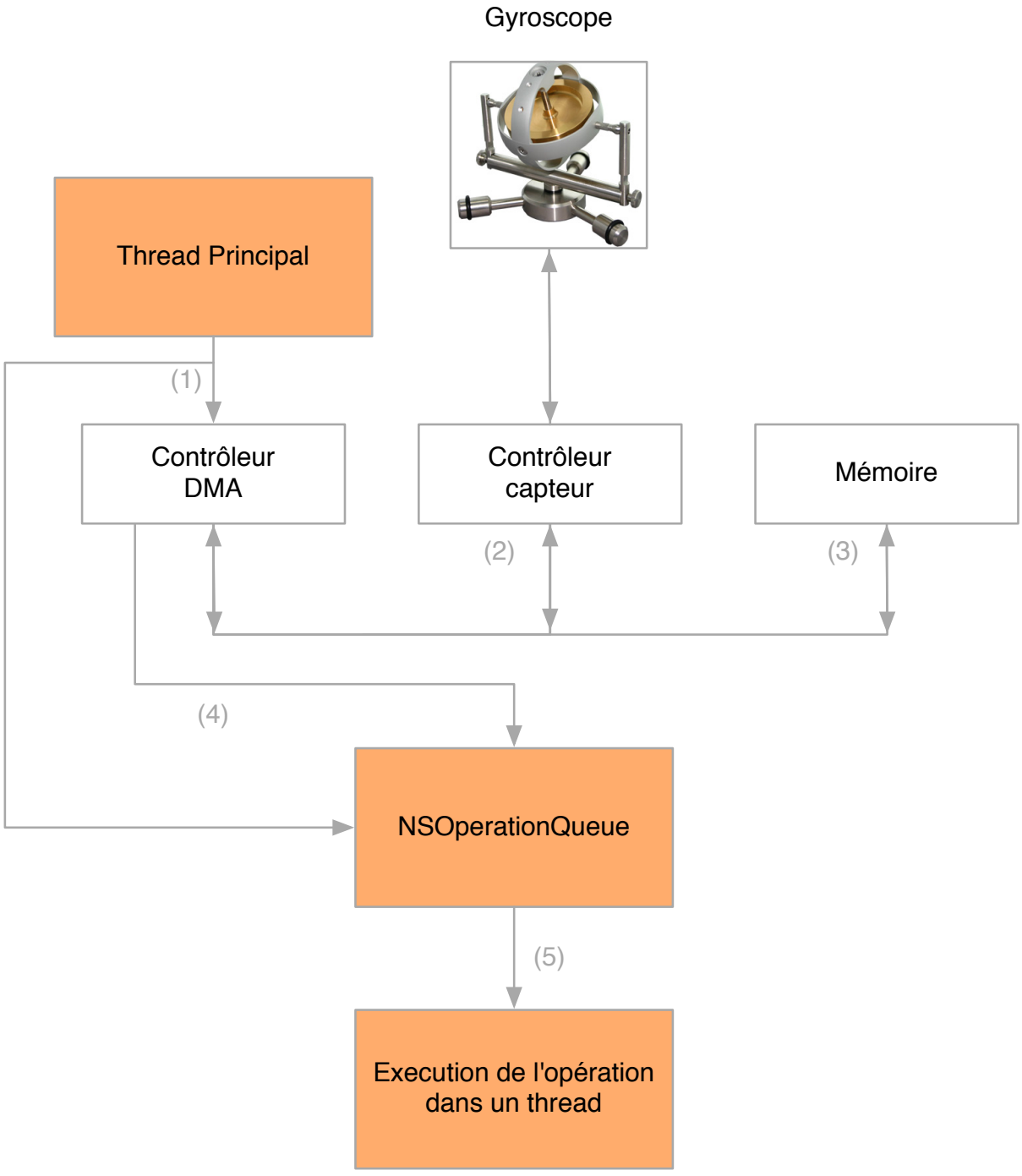


Figure 3 Fonctionnement DMA du gyroscope sous iOS

2.2.2. Fonctionnement sans interruption utilisateur

Dans ce mode de fonctionnement, il est nécessaire de venir « manuellement » récupérer les valeurs des périphériques. En effet, le contrôleur DMA ordonnera les transferts mémoires appropriés mais ne notifiera en aucun cas le programme.

3. Capteurs de mouvements

L'iPad 2 possède quatre capteurs permettant d'obtenir une information sur le mouvement ou la direction :

- La boussole,
- Le GPS,
- L'accéléromètre,
- Le gyroscope.

L'étude des capteurs de mouvements présents dans l'iPad a été réalisée dans le but de les utiliser pour calculer la position de l'appareil et permettre une géolocalisation précise. Par ailleurs, seul le gyroscope et l'accéléromètre ont été étudiés, les autres capteurs étant soumis à de fortes contraintes :

- Hyper-sensibilité au magnétisme (boussole)
- Signal trop faible en intérieur (GPS)

3.1. Acquisition des données

L'acquisition des données de ces deux capteurs est réalisée par le contrôleur DMA (Cf. 2.2). Le transfert des données est donc totalement géré par le DMA qui peut selon la configuration, exécuter un code (interruption de fin de transfert). Lors de la configuration du contrôleur, les paramètres suivants sont donnés :

- Fréquence d'acquisition,
- Code a exécuter (optionnel),
- Périphérique source.

Si aucune tâche n'est paramétrée à la réception des données, l'accès à celles-ci se fera grâce à la classe « *CMMotionManager* », qui permet la gestion des périphériques de capture de mouvements.

3.2. Gyroscope

3.2.1. Présentation

Le gyroscope est un composant permettant de mesurer la vitesse angulaire de l'appareil. Le périphérique présent dans l'iPad 2 permet d'obtenir une vitesse de rotation sur trois axes :

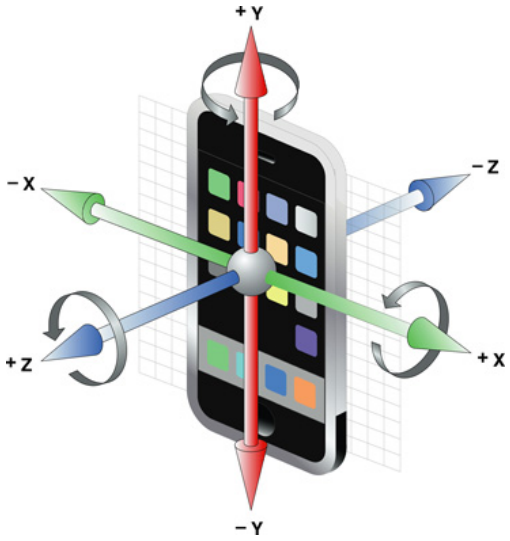


Figure 4 Illustration du gyroscope 3 axes

Les classes de gestions du gyroscope fournis avec le SDK de l'iPad permettent l'obtention des angles d'Euler en fonction des trois axes représentés par la figure ci-dessus.

3.2.2. Calibration

Lors de la configuration de la capture des données gyroscopiques, il est possible de paramétrer la référence du gyroscope. En effet, les axes du gyroscope peuvent être calibrés à l'aide de plusieurs méthodes :

- ***CMAttitudeReferenceFrameXArbitraryZVertical*** : ce mode définit l'axe Z comme l'axe vertical (par rapport à l'appareil) et définit arbitrairement les axes X et Y comme indiqués sur la Figure 4.

- ***CMAttitudeReferenceFrameXArbitraryCorrectedZVertical*** : ce mode est identique au précédent mais utilise le capteur magnétique de l'appareil afin de corriger les éventuelles erreurs.

- ***CMAttitudeReferenceFrameXMagneticNorthZVertical*** : L'axe Z est toujours définit comme l'axe vertical mais l'axe X n'est plus définit arbitrairement, il pointera vers le nord magnétique.

- ***CMAttitudeReferenceFrameXTrueNorthZVertical*** : Comme le mode précédant, l'axe Z est toujours définit comme l'axe vertical, l'axe X est orienté vers le nord géographique.

3.2.2.1. Mesures réalisées

Afin de déterminer la précision des acquisitions du gyroscope, des mesures ont été réalisées. Le graphique présenté en figure 4 est l'affichage des valeurs issues du capteur lors d'une rotation de 360° autours d'un axe (ici l'axe Z).

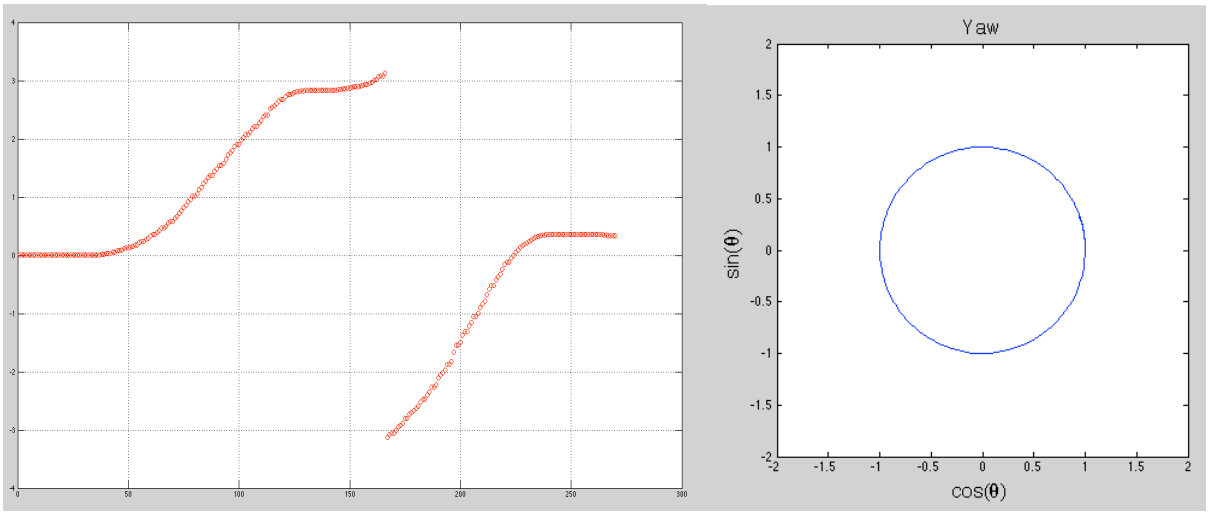


Figure 5 Rotation de 360° autours d'un axe

Le premier graphique représente l'évolution de l'angle d'Euler YAW au cours du temps. Il varie de 0 à 180° puis de 180° à 0 sans forte oscillation. Le second graphique illustre cette rotation en dessinant un cercle parfait.

3.2.2.2. Conclusions sur l'utilisation du gyroscope

La précision du gyroscope est suffisante pour pouvoir calculer une position. Les variations sont minimes et n'entraineront pas de fortes imprécisions lors du calcul.

3.3. Accéléromètre

3.3.1. Présentation

Un accéléromètre est un capteur permettant la mesure de l'accélération linéaire. Le capteur présent dans l'iPad 2 est un capteur mesurant une accélération sur 3 axes. En complément de la mesure de l'accélération linéaire, l'accéléromètre donne une mesure de la position de la gravité sur les trois axes. Ceci permet notamment de détecter la position de l'appareil (mode portrait, paysage).

3.3.2. Problèmes d'utilisations pour le projet

3.3.2.1. Compensation de la gravité

Afin de permettre la mesure de l'accélération linéaire fournie par l'utilisateur, la gravité doit être retirée des valeurs de l'accéléromètre. Ayant prévu cette fonction, le système réalise cette opération pour nous. L'inconvénient est que la méthode utilise un filtre passe-haut. En assumant que la gravité varie très doucement, cette technique serait la plus adéquate. Hors, la gravité est une force pouvant varier très rapidement (lors d'une chute par exemple), avec ce type de filtre, la gravité ne sera pas compensée et viendra altérer les valeurs

3.3.2.2. Mesures effectuées

A) Bruit de l'accéléromètre

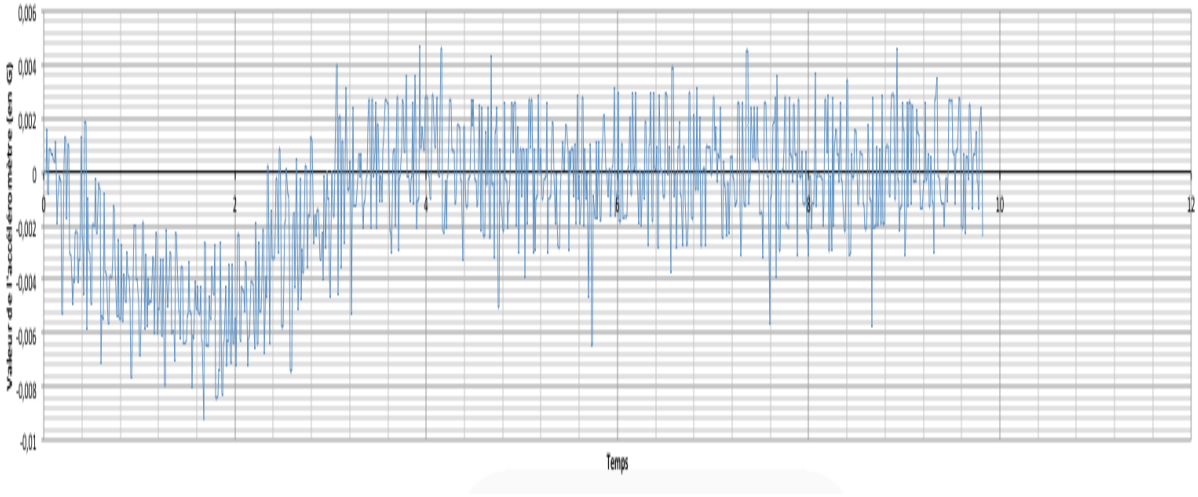


Figure 6 Captures des valeurs d'un axe de l'accéléromètre pendant 10 secondes

Ce graphique représente l'évolution des valeurs d'un axe de l'accéléromètre lors d'une capture de huit secondes. Lors de cette capture, l'iPad était positionné à plat sur une table sans aucune perturbation d'un utilisateur. Le graphique illustre donc l'erreur de mesure présente lors d'une simple capture :

- L'iPad obtient une valeur minimale d'accéléromètre de 0,01 G (environ 0,0981 $m.s^2$)
- L'iPad obtient une valeur maximale d'accéléromètre de 0,005 G (environ 0,04905 $m.s^2$)

Ces erreurs de mesure entrainent une erreur dans le calcul de vitesse et une grande erreur dans le calcul de la position.

Nous pouvons aussi constater la présence de différents pics dans les valeurs ceux-ci entrainent également une erreur dans les calculs.

B) Mesure d'un mouvement

Voici les constats qui sont ressortis lors de différents tests :

- Mouvement rectiligne à faible vitesse :
  Le capteur ne ressent aucune variation d'accélération (l'accélération est confondu au bruit), il est donc impossible de déterminer un mouvement.
- Mouvement rectiligne à grande vitesse :
  Le capteur de l'iPad varie fortement mais il est impossible de retranscrire ces variations en vitesse et en position car l'erreur de mesure est trop grande (conclusion faite par la capture des valeurs pour le « même » mouvement plusieurs fois).

Ces erreurs sont dues aux bruits émis en sortie du capteur et aux pics émis par le capteur : si un pic est émit, alors le calcul de la vitesse devient faux et par conséquent le calcul de la position l'est également. De plus, si ces pics sont fréquents les erreurs sont de plus en plus importantes.

3.3.2.3. Conclusions sur l'utilisation de l'accéléromètre

Les imprécisions du capteur ne permettent pas son utilisation pour un calcul de position précis. Néanmoins, il peut être utilisé pour détecter une forte variation entrainant une remise a zéro de l'algorithme de calcul de position.

4. Capteur vidéo

4.1. Acquisition

La capture des images vidéo suit le processus du DMA, les transferts des images vers la mémoire sont donc automatisés par le contrôleur DMA. L'accès à cette fonctionnalité est contrainte à l'utilisation de la librairie AVFoundation, librairie fournie par Apple et permettant la gestion basique des images et des contenues audio.

La librairie nous permet la configuration d'une session de capture. Dans notre cas, la session ne comportera qu'un périphérique d'acquisition unique : la caméra arrière de l'iPad. Plusieurs périphériques de sorties sont configurés :

- L'écran afin d'afficher les images capturées. L'affichage de l'image est effectué automatiquement, seule la zone de l'écran sur laquelle l'image doit apparaître doit être configurée.

- Une classe afin de permettre le traitement des images.

Le format de sortie de l'image obtenue est défini avec le paramètre « kCVPixelFormatType_32BGRA », il permet de définir un pixel codé avec les caractéristiques suivantes :

- 32 bits de données

- 1 octet pour la composante Bleu,

- 1 octet pour la composante Verte,

- 1 octet pour la composante Rouge,

- 1 octet pour l'opacité du pixel.

4.2. Fonctionnement du buffer d'image

Lors de la capture d'une image, celle-ci est transférée en mémoire par le contrôleur DMA. La session de capture peut être configurée selon deux modes de fonctionnements distincts (en matière de gestion des images) :

- Une seule image est stockée en mémoire, tant que l'utilisateur utilise l'image alors aucun nouveau transfert n'est engendré.

- Les images sont stockées dans un buffer dont la taille est gérée par le système. Dans ce mode de fonctionnement, si le traitement appliqué sur l'image dure plus de $\frac{1}{25}s$ alors les images s'enchaineront dans le buffer et occuperont beaucoup d'espace mémoire.

4.3. Accès à l'image

Quel que soit le mode de fonctionnement, une seule image est traitée à un moment donné. Ceci s'explique par la méthode d'accès à une image : lorsque l'utilisateur désire accéder à l'image présente dans le buffer, il est nécessaire d'utiliser un système de sémaphore afin de réserver la mémoire et empêcher toutes modifications des données. Même le système sera interdit de modifier les valeurs présentes en mémoire. Evidemment, le sémaphore doit être libéré à la fin d'un traitement si l'on désire accéder à la prochaine image !

Concernant l'accès aux pixels, la librairie fournie des méthodes permettant l'obtention d'un tableau de pixels, de la largeur, de la hauteur ainsi que des informations sur le codage des pixels. Le tableau de pixels ne possède qu'une seule dimension et stocke les pixels comme indiqué par la Figure ci-dessous.

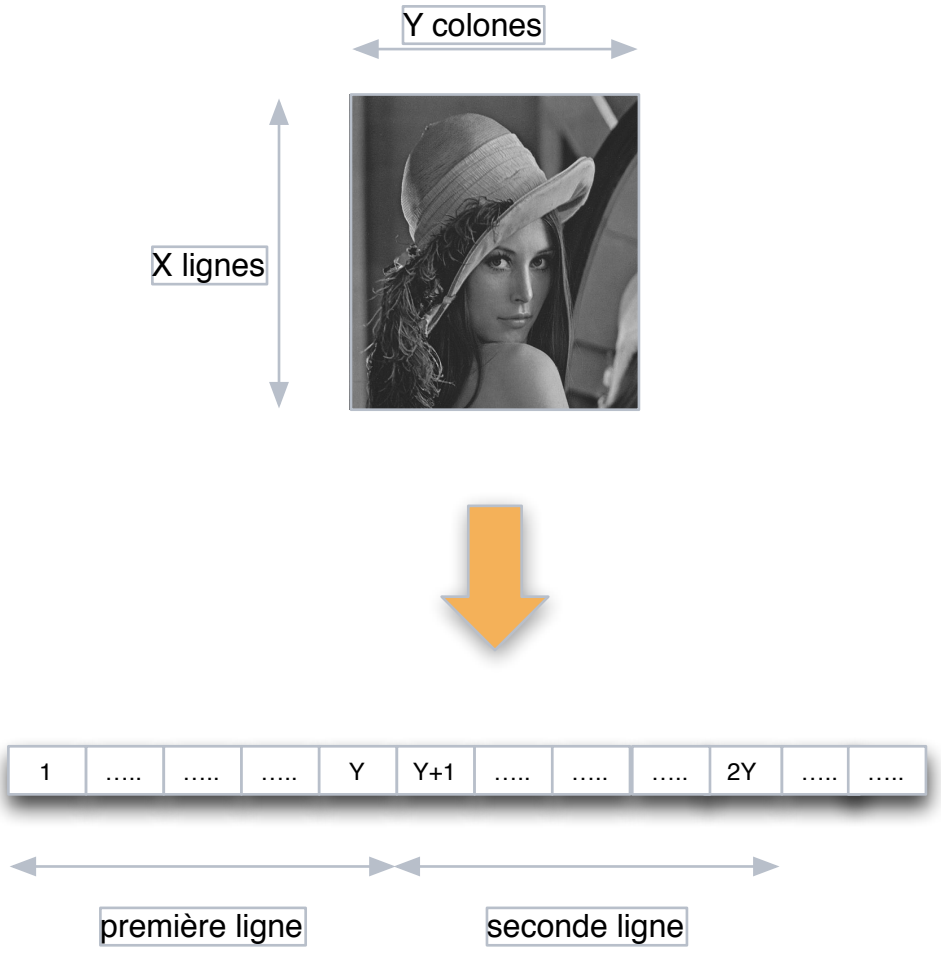


Figure 7 Codage d'une image

4.4. Problèmes liés à la méthode d'accès à l'image

Le problème majeur de la méthode d'accès aux images est qu'il est impossible d'obtenir deux images simultanément. Les images étant stockées dans un buffer et

protégées par un sémaphore, la seule solution pour pouvoir traiter deux images simultanément est de dupliquer l'image en mémoire. Hors, si l'on désire garder un temps de traitement rapide (s'approchant du temps réel), cette opération est prohibée.

Le second problème important est l'effet résultant d'un traitement trop long sur une image. En effet, si le traitement d'une image dure plus de $\frac{1}{25}s$ alors une succession d'images seront stockées dans le buffer d'image (si le mode d'empilage de toutes les images est actif). Si ce problème de temps de traitement est présent sur toutes les images du flux vidéo, alors la taille du buffer prendra une dimension conséquente et limitera les performances de l'application et de l'appareil.

Si le mode d'empilage des images dans le buffer est désactivé, alors le problème de mémoire sera résolu mais au détriment d'images et donc, d'informations potentiellement intéressantes.

Partie 3 : Géométrie
Projections de points entre deux frames successives

1. Représentation des frames

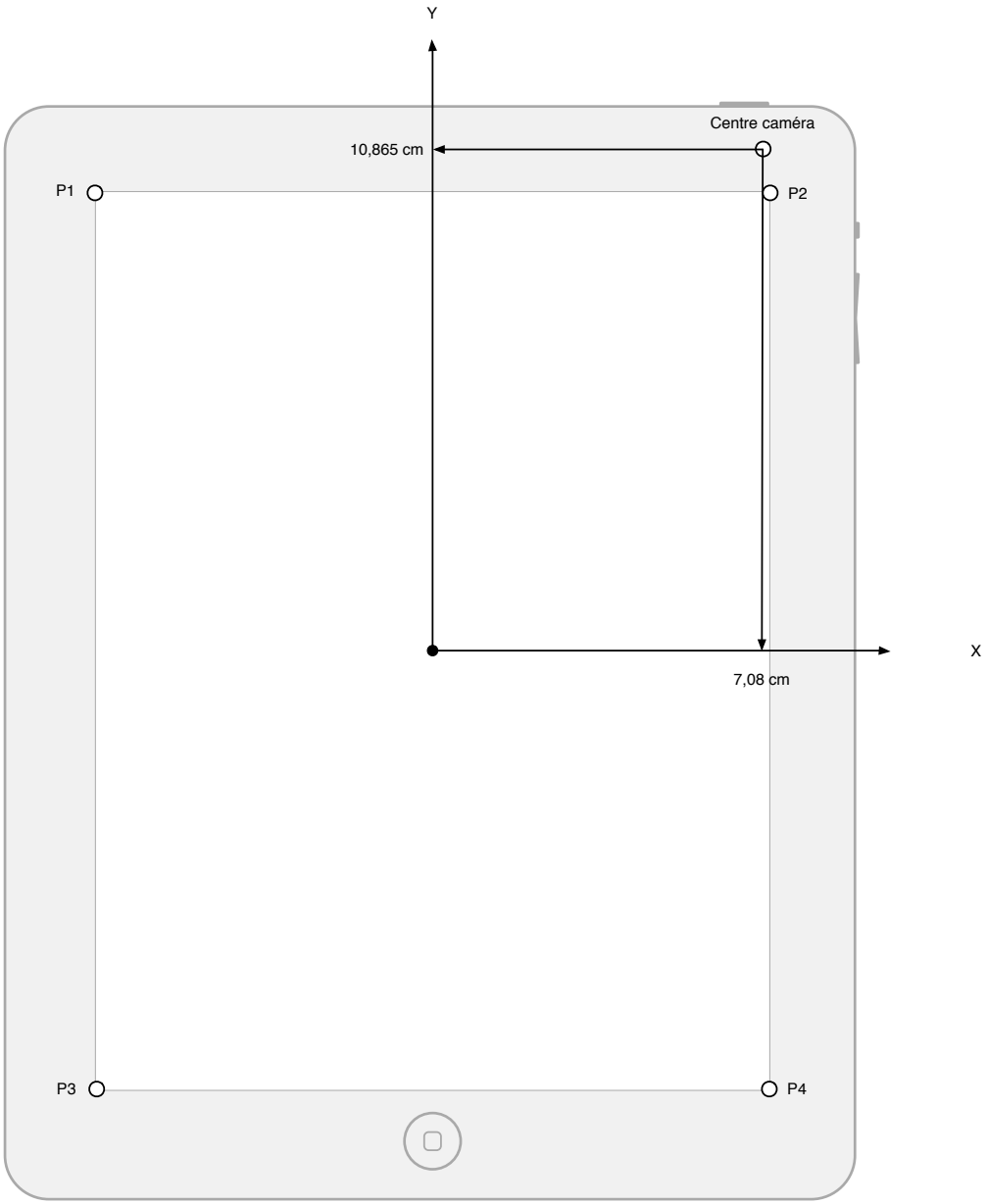


Figure 8 Illustration du repère de l'iPad

1.1. Vision de la caméra à 1 mètre

Pour permettre la mesure de l'angle de vision du capteur vidéo, l'expérience suivante a été réalisée : L'iPad est placé à un mètre d'un mur afin de capturer les images situées à cette distance. Les résultats suivants ont été obtenus :

- Vision en largeur : 60 cm (Axe X de la figure ci-dessus)
- Vision en longueur : 80 cm (Axe Y de la figure ci-dessus)

- Distance : 100 cm (Axe Z de la figure)

La caméra de l'iPad n'est pas positionnée au centre de celui-ci, ainsi les valeurs suivantes sont observées :

- 10 cm suivant l'axe Y
- 7 cm selon l'axe Y

1.2. Positionnement dans l'espace de la première frame

La première frame est initialisée avec les valeurs suivantes :

$$p1 = \begin{pmatrix} xCamPos - \left(\frac{l}{2}\right) \\ yCamPos + \left(\frac{L}{2}\right) \\ 100 \end{pmatrix} \quad p2 = \begin{pmatrix} xCamPos + \left(\frac{l}{2}\right) \\ yCamPos + \left(\frac{L}{2}\right) \\ 100 \end{pmatrix}$$

$$p3 = \begin{pmatrix} xCamPos - \left(\frac{l}{2}\right) \\ yCamPos - \left(\frac{L}{2}\right) \\ 100 \end{pmatrix} \quad p1 = \begin{pmatrix} xCamPos + \left(\frac{l}{2}\right) \\ yCamPos - \left(\frac{L}{2}\right) \\ 100 \end{pmatrix}$$

$$avec \begin{cases} l = 60cm \\ L = 80cm \\ xCamPos = 7cm \\ yCamPos = 10cm \end{cases}$$

1.3. Calcul de la position des frames suivantes

Pour chaque nouvelle frame, sa position est calculée par rapport à la précédente. Dans un premier temps, la position angulaire de la frame précédente est récupérée et utilisée pour calculer le déplacement relatif entre les deux frames. Une fois le déplacement angulaire obtenu, une matrice de rotation est calculée. La nouvelle frame est simplement obtenue en appliquant la matrice de rotation sur la frame précédente.

2. Projection d'un point entre deux frames successives

Le schéma représente la projection d'un point présent sur la première frame vers la seconde.

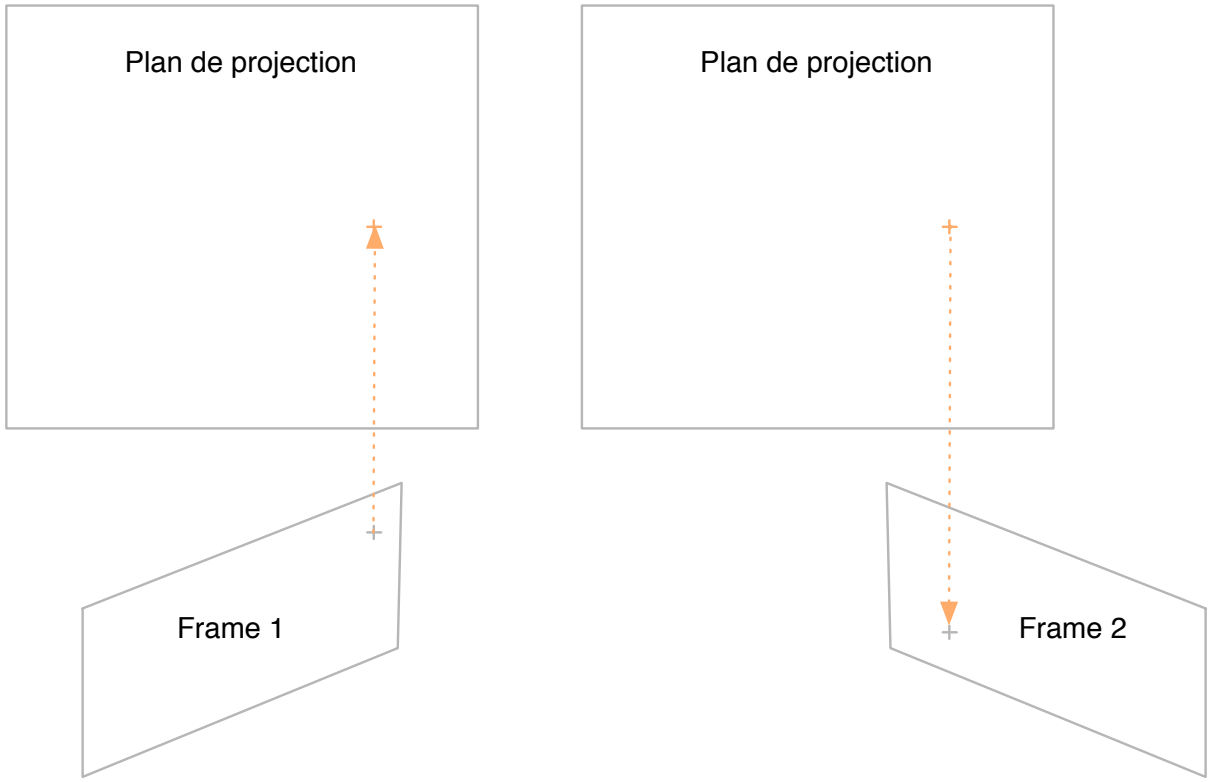


Figure 9 Principe de fonctionnement de la projection d'un point entre deux frames

Cette projection est le résultat de plusieurs opérations successives :

- La création d'un plan de projection commun
- La projection du point sur le plan commun
- La projection du point (du plan commun) sur la seconde frame

2.1. Représentation d'un plan

Chaque frame est représentée par un plan orienté dans l'espace 3D par les coordonnées calculées précédemment. Les sommets A, B, C et D représentent les sommets des frames. Ils sont calculés grâce aux données du gyroscope.

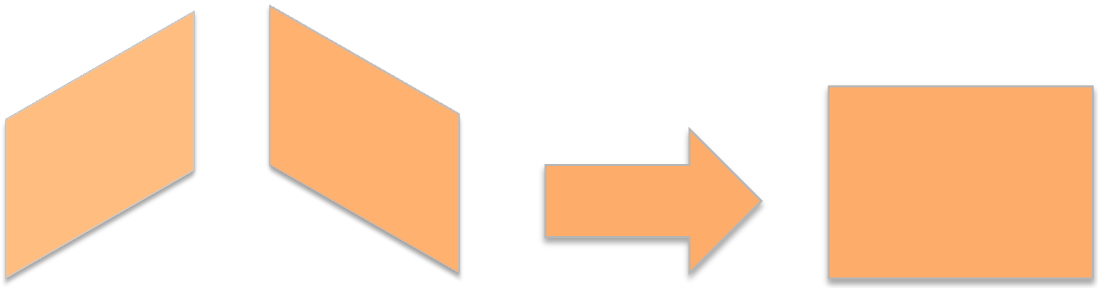
Le plan est représenté par trois caractéristiques :

- Son point d'origine : $A = \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix}$
- Deux vecteurs directeurs normalisés :

$$\overrightarrow{AB} = \begin{pmatrix} b_x - a_x \\ b_y - a_y \\ b_z - a_z \end{pmatrix} * \frac{1}{\|\overrightarrow{AB}\|} \; et \; \overrightarrow{AC} = \begin{pmatrix} c_x - a_x \\ c_y - a_y \\ c_z - a_z \end{pmatrix} * \frac{1}{\|\overrightarrow{AC}\|}$$

2.2. Création du plan commun

Afin de pouvoir effectuer des calculs de recouvrements, il est nécessaire de projeter les frames sur un plan commun. Ce plan devra être le plus proche possible des deux frames afin qu'il permette la meilleure représentation possible. Pour sa création, les valeurs des deux plans des frames sont moyennées comme montre la figure suivante :



Le plan de projection sera donc représenté par :

- Son point d'origine :

$$A = \frac{A' + A''}{2} = \begin{pmatrix} \frac{a'_x - a''_x}{2} \\ \frac{a'_y - a''_y}{2} \\ \frac{a'_z - a''_z}{2} \end{pmatrix}$$

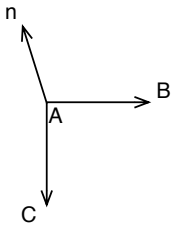
- Deux vecteurs directeurs normalisés :

$$\overrightarrow{AB} = \frac{\overrightarrow{AB}' + \overrightarrow{AB}''}{2} \; et \; \overrightarrow{AC} = \frac{\overrightarrow{AC}' + \overrightarrow{AC}''}{2}$$

2.3. Calcul de vecteur normal

Un vecteur normal au plan ($\overrightarrow{An}$ *sur le schéma noté par la suite* $\vec{n}$) est obtenu par la formule suivante :

$$\vec{n} = \overrightarrow{AB} \; \Lambda \overrightarrow{AC}$$



2.4. Projection d'un point P sur un plan

Soit un point M, M appartient au plan (ABC) si et seulement si M est perpendiculaire au vecteur normal $\vec{n}$.

$$M \in (ABC) \begin{cases} \Leftrightarrow \overrightarrow{AM} \perp \vec{n} \\ \Leftrightarrow \overrightarrow{AM}.\vec{n} = 0 \end{cases}$$

Afin de projeter le point P sur le plan (ABC), on cherchera le point M tel que :

$$\begin{cases} M \in (ABC) \; (1) \\ \overrightarrow{PM} = k\vec{n} \quad (2) \end{cases}$$

Déductions faites à partir de l'équation (1).

$$(1) \Leftrightarrow \overrightarrow{PM} = k\vec{n}$$

$$(1) \Leftrightarrow \begin{pmatrix} M_x - P_x \\ M_y - P_y \\ M_z - P_z \end{pmatrix} = k \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix}$$

$$(1) \Leftrightarrow M = \begin{pmatrix} kn_x + P_x \\ kn_y + P_y \\ kn_z + P_z \end{pmatrix}$$

Déductions faites à partir de l'équation (2).

$$(2) \Leftrightarrow AM \bullet \vec{n} = 0$$

$$(2) \Leftrightarrow \begin{pmatrix} M_x - A_x \\ M_y - A_y \\ M_z - A_z \end{pmatrix} \bullet \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} = 0$$

$$(2) \Leftrightarrow n_x\left(M_x - A_x\right) + n_y\left(M_y - A_y\right) + n_z\left(M_z - A_z\right) = 0$$

$$(2) \Leftrightarrow n_x\left(kn_x + P_x - A_x\right) + n_y\left(kn_y + P_y - A_y\right) + n_z\left(kn_z + P_z - A_z\right) = 0$$

$$(2) \Leftrightarrow kn_x^2 + n_xP_x - n_xA_x + kn_y^2 + n_yP_y - n_yA_y + kn_z^2 + n_zP_z - n_zA_z = 0$$

$$(2) \Leftrightarrow k(n_x^2 + n_y^2 + n_z^2) + \alpha + \beta + \chi = 0$$

$$\begin{cases} \alpha = n_xP_x - n_xA_x = n_x\left(P_x - A_x\right) \\ \beta = n_yP_y - n_yA_y = n_y\left(P_y - A_y\right) \\ \chi = n_zP_z - n_zA_z = n_z\left(P_z - A_z\right) \end{cases}$$

$$(2) \Leftrightarrow k = -\frac{\alpha + \beta + \chi}{n_x^2 + n_y^2 + n_z^2}$$

Le calcul de k est à effectuer pour chaque projection de points dans le plan.

2.5. Changement de repère

Afin de faciliter les calculs de recouvrements, nous utiliserons le repère 2D du plan précédemment créé :

$$M = \begin{pmatrix} kn_x + P_x \\ kn_y + P_y \\ kn_z + P_z \end{pmatrix} \equiv m = \begin{pmatrix} m_{\vec{i}} \\ m_{\vec{j}} \end{pmatrix}$$

$$\vec{i} = \overrightarrow{AB}$$

$$\vec{j} = \overrightarrow{AC}$$

Partie 4 : Traitement d'images
Application d'une corrélation avec l'opérateur LoG

1. Application d'un masque de corrélation

La corrélation est un opérateur standard pour le traitement linéaire d'images et permet l'application d'un masque sur une image. Le masque est définit comme une matrice de taille prédéfinie.

$$W = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{pmatrix}$$

Une matrice de corrélation possède deux paramètres variables :

- Les coefficients w_{ij} qui sont calculés en fonction du masque à appliquer,
- Sa taille.

L'application d'une matrice de corrélation se fait selon la formule suivante :

$$I_r(i,j) = \sum_{u=-n}^{+n} \sum_{v=-n}^{+n} w_{u+n+1,v+n+1} \times I(i+v, j+u)$$

Prenons pour exemple, un masque d'une ligne et de trois colonnes que nous allons appliquer sur une image de trois lignes sur quatre colonnes :

$$W = \begin{pmatrix} -1 & 0 & 1 \end{pmatrix}$$

$$I = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 2 \\ 0 & 0 & 2 & 2 \end{pmatrix}$$

$$I_R = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & -2 \\ 0 & 2 & 2 & -2 \end{pmatrix}$$

$$i_{R_{23}} = -1 \times 1 + 0 \times 2 + 1 \times 2 = 1$$

2. Laplacian of Gaussian (LoG)

Dans notre étude, l'opérateur LoG est utilisé pour effectuer une détection de points d'intérêts. Il se décompose en deux applications de filtre :

- Un filtre Gaussien
- Un file Laplacien

2.1. Filtre Gaussien

2.1.1. Principe

Un filtre Gaussien est un filtre dit « passe-bas » car il atténue les variations de l'image en appliquant un flou, on parle alors de flou Gaussien. Pour comprendre l'effet de ce filtre, il est nécessaire de regarder sa fonction :

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{x^2+y^2}{2\sigma^2}}$$

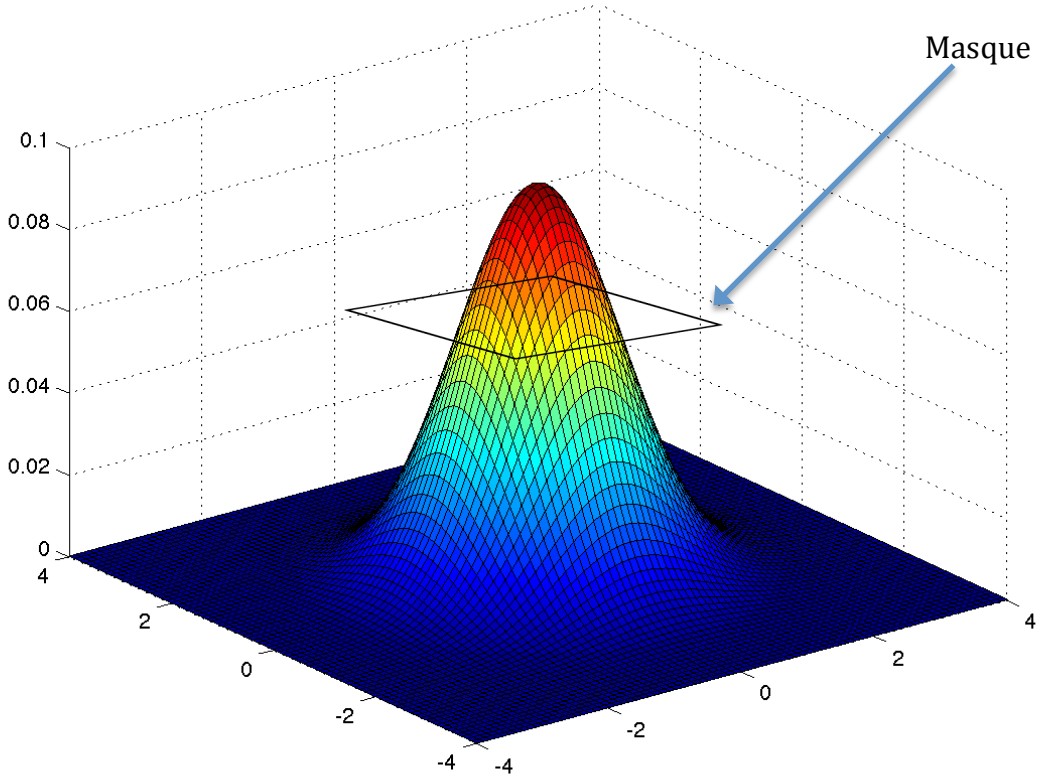


Figure 10 Courbe de la fonction du masque Gaussien

Comme indiqué sur la figure précédente, le masque est créé en centrant les points de la courbe sur la matrice du masque. Les coefficients du masque obtenu sont donc de la forme :

$$G = \begin{pmatrix} 16 & 26 & 16 \\ 26 & 41 & 26 \\ 16 & 26 & 16 \end{pmatrix}$$

L'application d'un tel filtre permet, pour chaque pixel, de calculer une nouvelle valeur fonction de lui-même et de ses voisins. Les coefficients sont de tel sorte que

- Plus le voisin est éloigné, moins sa valeur affecte le résultat,
- Le pixel central est multiplié par un coefficient plus important que ses voisins.

Ainsi la forme de l'image n'est pas modifiée mais les éventuels « bruits » sont atténués par le flou.

2.1.2. Caractéristiques du masque

Le masque se comporte différemment si l'un de ses facteurs déterminant change.

2.1.2.1. Valeur de σ

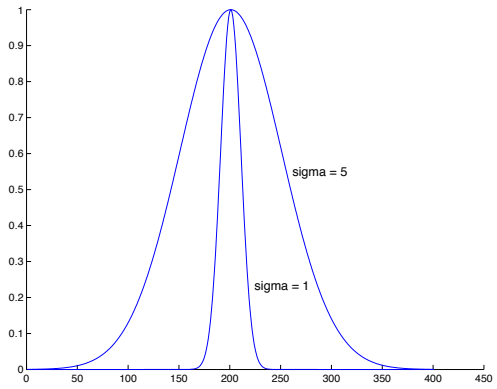


Figure 11 Effets de la variation de sigma sur la courbe gaussienne

Plus la valeur de σ augmente, plus la courbe s'élargie. Cela à pour effet d'augmenter l'importance des pixels voisins et d'accentuer le flou appliqué à l'image.

2.1.2.2. Taille du masque

Plus la taille du masque est grande, plus le nombre de pixels voisins pris en compte dans le calcul augmente. Le filtre est donc plus pertinent mais la corrélation consommera plus de ressource processeur.

2.2. Filtre Laplacien

La filtre Laplacien est un filtre dérivatif utilisant la dérivée seconde afin d'effectuer une détection de point d'intérêt dans une image. Il s'appui sur les variations des valeurs de pixels d'une image en niveaux de gris :

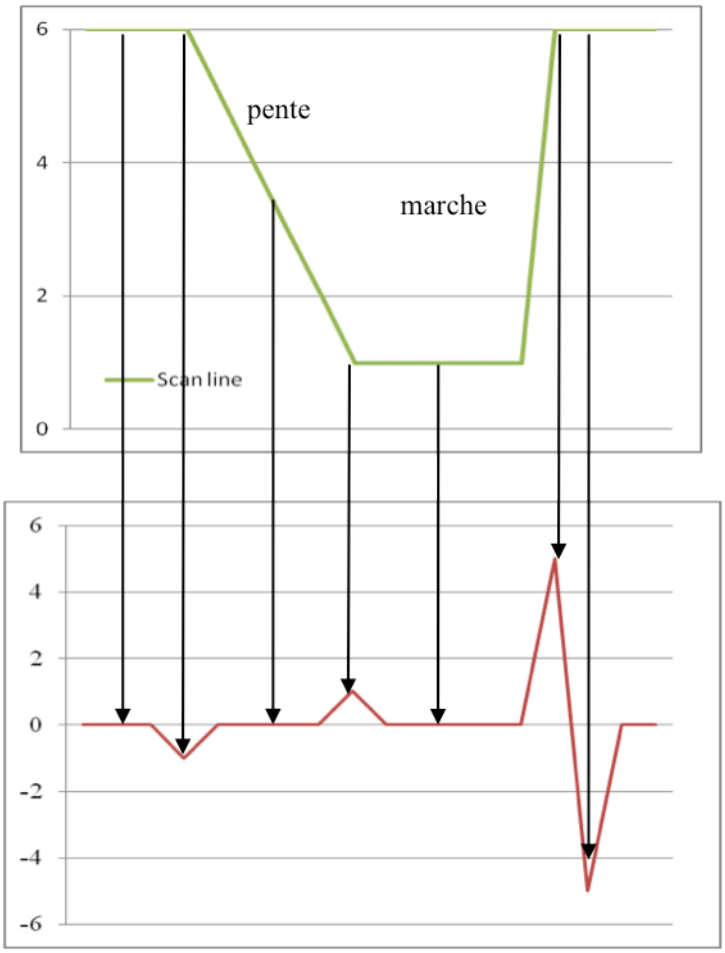


Figure 12 Illustration du fonctionnement d'un filtre dérivatif

Lors de la présence d'une pente dans les valeurs de pixels d'une image, la dérivée seconde oscille au commencement de la pente et à sa fin. Lors de la présence d'un saut dans les valeurs de pixels de l'image, la dérivée seconde oscille en prenant des valeurs positives et négatives pour deux pixels successifs. C'est cette dernière oscillation qui est détectée dans notre application. Le postulat effectué est que ce point est significatif dans l'image.

Les oscillations provoquées par une pente sont apparentées à un bruit sur l'image.

Le masque Laplacien est obtenu par la fonction :

$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial^2 x} + \frac{\partial^2 f}{\partial^2 y}$$

2.3. Filtre LoG

Le filtre LoG (Lapacian of Gaussian) se base sur les deux filtres précédents et permet dans notre cas de réaliser la détection de points d'intérêts dans notre image. Il est le résultat de la combinaison des deux opérateurs précédents :

Après la transformation de l'image en niveaux de gris, un masque gaussien est appliqué afin de lisser l'image et d'éliminer toute trace de bruits.

Une fois le lissage effectué, le filtre dérivatif est appliqué. Il résulte en la détection de points d'intérêts : de sauts dans les valeurs des pixels. L'effet du filtre gaussien précédemment appliqué est la suppression des petites variations de la dérivée seconde. Le bruit de l'image est filtré. La détection de points est ensuite effectuée par la détection du croisement entre l'axe des abscisses et la dérivée seconde.

Pour la rapidité des traitements, le masque gaussien et le masque de Laplace sont fusionnés. Les valeurs des coefficients du masque sont obtenues par la formule suivante :

$$LoG = \nabla^2 G(x,y) = \frac{1}{\sigma^2}\left[\frac{x^2 + x^y}{\sigma^2} - 1\right] e^{-\frac{x^2+y^2}{2\sigma^2}}$$

La formule correspond à la courbe suivante :

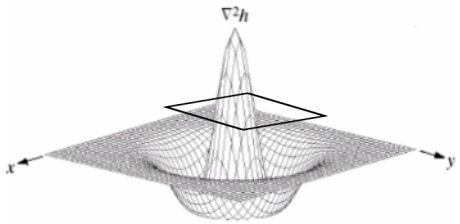


Figure 13 Courbe 2D du LoG : Mexican Hat

$$LoG = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

3. Matrice LUT (Look Up Table)

Pour permettre la détection des points d'intérêts, l'image traitée doit être codée en niveaux de gris. Transcrire tous les points d'une image en niveaux de gris prend un énorme temps processeur. Pour optimiser ce traitement, la transcription en niveaux de gris est faite par l'utilisation d'une matrice LUT.

Une matrice LUT est une matrice en trois dimensions qui associe un niveau de gris à chaque valeur possible des trois composantes d'un pixel. Ainsi, la conversion d'un pixel ne nécessite qu'un seul accès mémoire dans un tableau améliorant ainsi fortement la rapidité.

Elle sera initialisée avant le début des traitements sur une image et ne sera jamais recalculée. Voici comment elle est obtenue :

$$\begin{cases} val = \alpha * red + \beta * green + \chi * blue \\ \alpha + \beta + \chi = 1 \end{cases}$$

Ainsi pour chaque valeur des composantes de couleurs $[0;255]$ la valeur équivalente en niveau de gris est calculée. La valeur des coefficients a été fixée à $\frac{1}{3}$ afin de ne pas privilégier une couleur par rapport à une autre.

Partie 5 : Synthèse
Fusion des différents composants

1. Synchronisation des acquisitions

1.1. Modélisation d'une frame

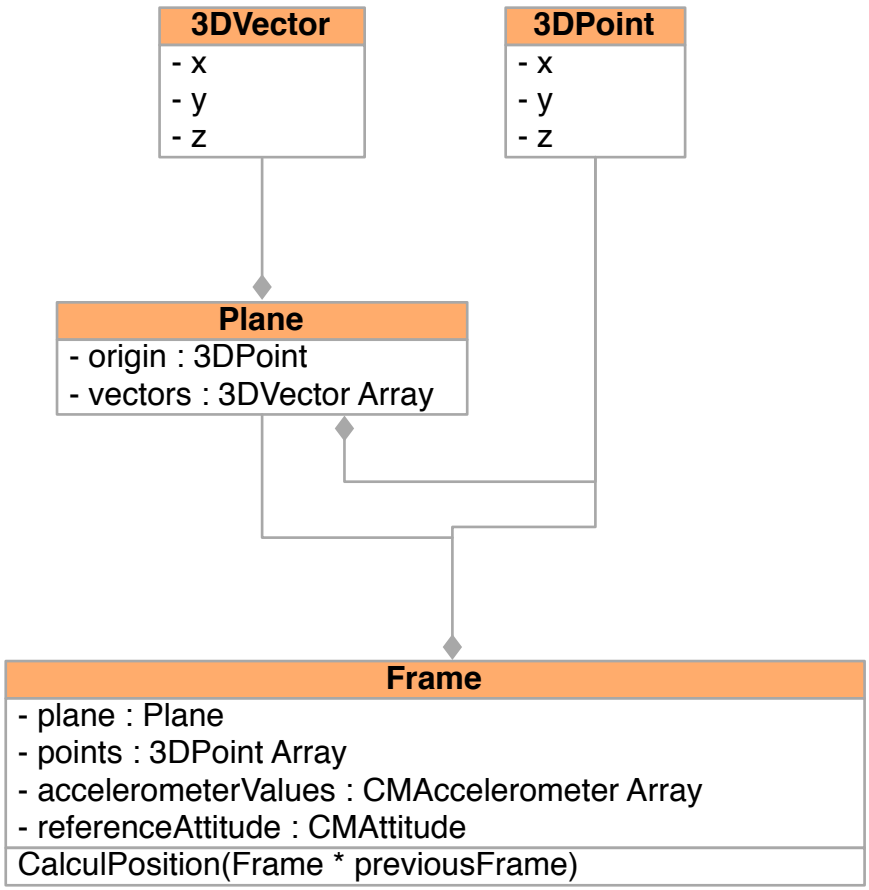


Figure 14 Diagramme de classe d'une frame

Voici la structure implémentée pour la modélisation d'une frame, elle possède un tableau de points correspondant aux quatres extrémités de l'image et un attribut correspondant au plan généré par les points. Les valeurs d'accéléromètre sont stockées dans un tableau et permettront le calcul d'une moyenne afin de compenser les imprécisions du capteur. Les valeurs du gyroscope sont stockées dans l'attribut « refereceAttitude » et ne nécessitent pas de corrections.

La méthode « CalculPosition » permet le calcul de la position de la frame par rapport à la frame précédente.

1.2. Queue de traitements des frames

Afin de stocker les frames et de pouvoir accéder aux N frames précédentes, celles-ci seront stockées dans une liste circulaire (C.f. Figure 15).

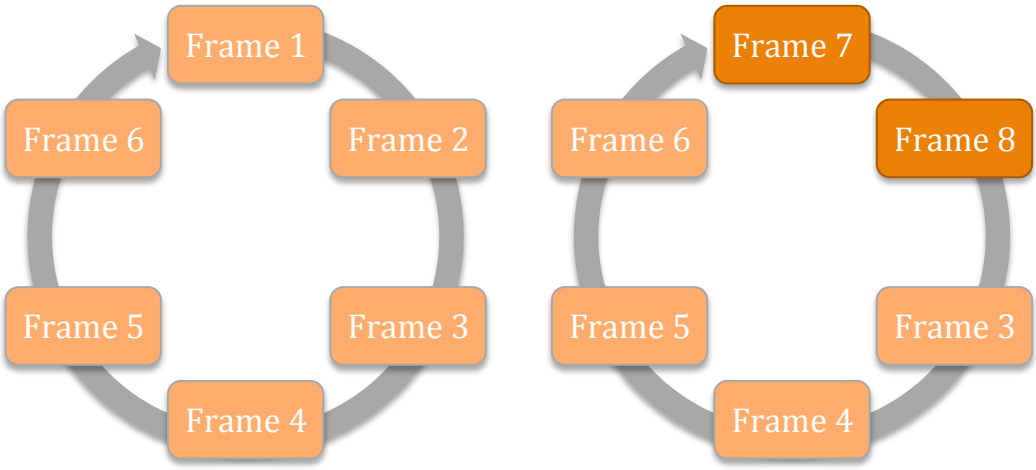


Figure 15 Principe de fonctionnement de la queue de traitement circulaire

Lors de l'ajout d'une frame dans la queue de traitement, deux cas de figure se présente :

- La file est vide ou n'est pas totalement remplie :
  Dans ce cas, les frames sont stockées les unes à la suite des autres sans perte de frames précédentes.

- La file est pleine :
  Dans ce cas, La première frame entrée dans la file est effacée et remplacée par la nouvelle frame.

La liste suite donc un procédé « FIFO » (First In First Out) ce qui permet de toujours posséder une référence vers une ancienne frame pour le calcul de la position.

1.3. Synchronisations des captures de mouvements de l'appareil

Un point important des fusions des différentes parties du projet est la synchronisation des acquisitions matérielles avec les acquisitions des frames. Cela permet d'obtenir les bonnes valeurs en sortie des périphériques de mouvements.

La Figure 16 explique la synchronisation qui est réalisée entre les différents composants matériels :

Le thread principal de l'application est utilisé pour effectuer les opérations de configuration logicielles des périphériques :

- Initialisation de la queue de traitement GCD du capteur CCD,

- Initialisations des acquisitions des capteurs de mouvements (programmation DMA).

Une fois ces étapes réalisées, l'application se met en attente de la première interruption générée par le capteur CCD. Elle sera récupérée par le composant logiciel GCD puis transmise au thread de gestion de la caméra qui effectuera trois opérations :

- Création d'une nouvelle frame et insertion dans la file de traitement circulaire,
- Lancement d'un timer sur le thread principal (inutilisé) afin de permettre les acquisitions des valeurs de sorties du gyroscope et de l'accéléromètre,
- Calcul de la position relative de la frame précédente.

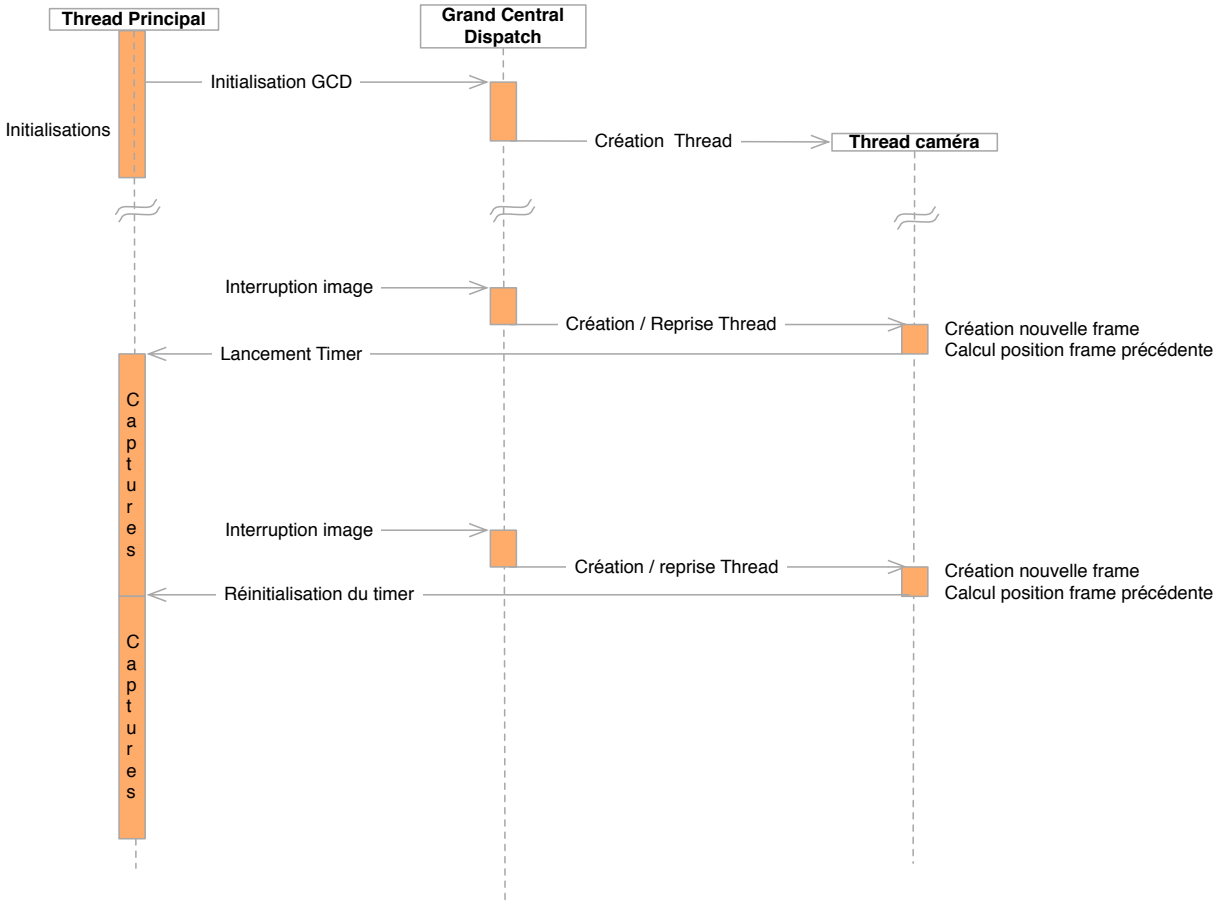


Figure 16 Diagramme de séquence de la synchronisation

Le timer créé par le thread de la caméra possède un intervalle de $0{,}01s$ permettant ainsi d'obtenir quatre acquisitions d'accéléromètre pour chaque image reçue ($\frac{1}{25}s$). La réinitialisation du timer permet la « resynchronisation » de celui-ci avec les interruptions images.

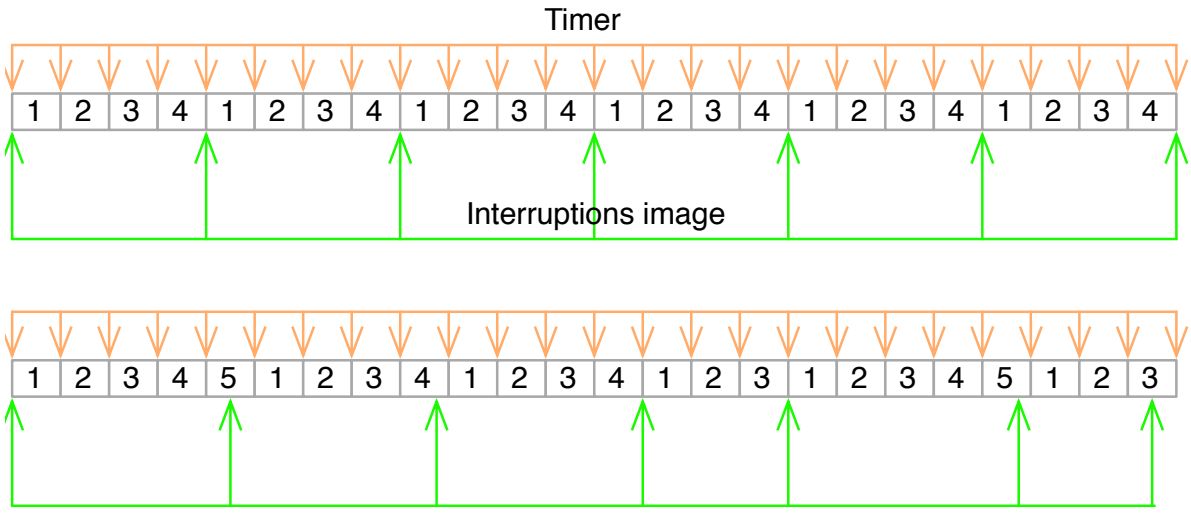


Figure 17 Illustration de la synchronisation

La figure ci-dessus, permet l'illustration de la synchronisation réalisée et la compare à ce qu'elle aurait été sans la réinitialisation du timer. Sans cette étape, les acquisitions ne seraient pas accordées et le nombre de captures aurait été très variant selon les frames.

1.4. Séquence de l'application

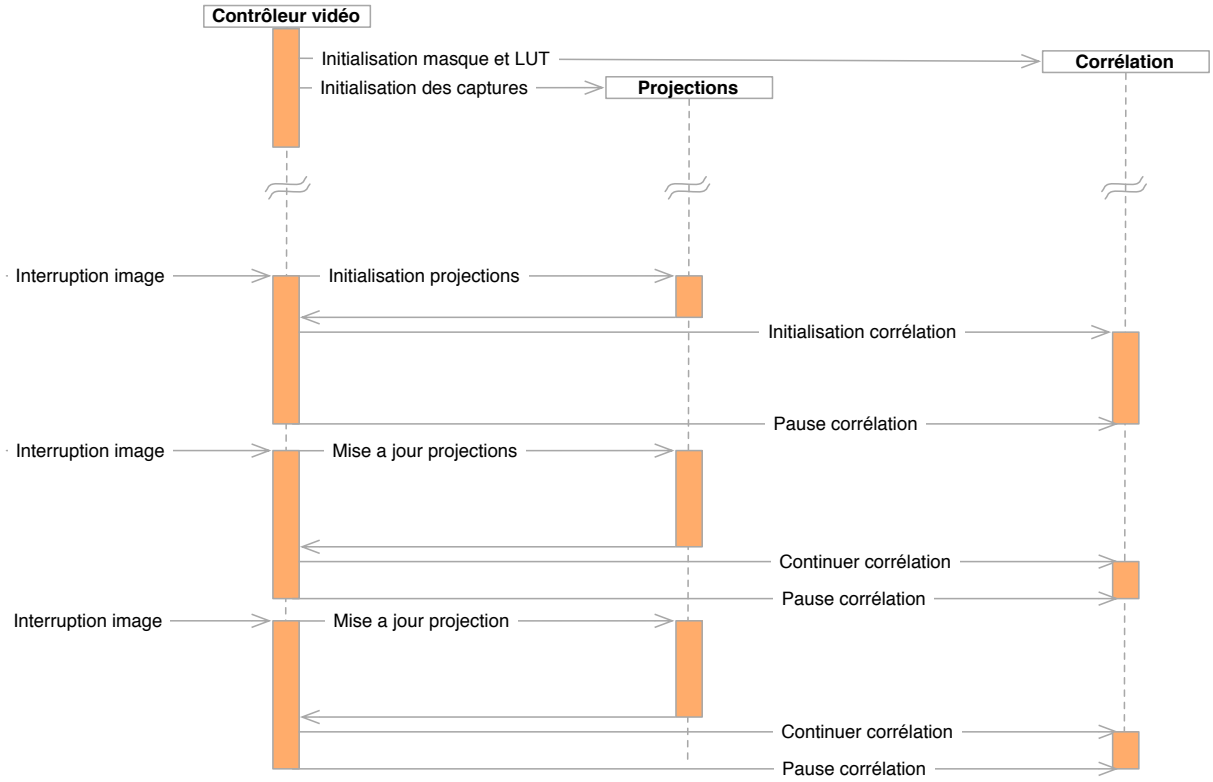


Figure 18 Diagramme de séquence de l'application

La figure ci-dessus représente les interactions entre les différents composants logiciels de l'application. L'application commencera donc par initialiser tous les composants. Puis à chaque interruption d'image, l'application fera les étapes suivantes :

- Calcul de la position de la frame par rapport à la précédente,
- Projections des points d'intérêts sur la nouvelle frame,
- Reprise de la corrélation à partir des nouveaux points.

Conclusion

Ce projet m'a permis de mettre en œuvre mes connaissances en matière de développement iOS mais m'a aussi permis d'acquérir de plus amples connaissances sur l'ordonnancement et la gestion mémoire sous iOS.

Par manque de temps ce projet n'est pas terminé. Les parties concernant les acquisitions, les synchronisations sont terminées mais la partie concernant les projections orthogonales n'est pas entièrement terminée.

Bibliographie

Clubic. (2011, Janvier 01). *La réalité augmentée : 10 application iPhone pour se repérer*. Consulté le Septembre 29, 2011, sur clubic: http://www.clubic.com/article-324608-3-realite-augmentee-applications-iphone-3g.html

Stallings, W. *Operating Systems – Internal and Design Principles.* ?: ?

Tanenbaum, A. S. (2001). *Modern Operating Systems.* ?: Prentice Hall PTR.

Wikipedia. (s.d.). *Modèle-Vue-Contrôleur*. Consulté le 08 16, 2011, sur Wikipédia: http://fr.wikipedia.org/wiki/Mod%C3%A8le-Vue-Contr%C3%B4leur

Wikitude (Écrivain). (2009). *Wikitude Augmented Reality* [Film].

Résumé :

Ce rapport est la finalisation du projet de fin d 'études numéro 6 de l'année 2011-2012. Il porte sur la conception d'une approche permettant le support d'algorithmes de traitements d'images « temps réel » sous systèmes mobiles. L'étude s'est déroulée en trois parties distinctes. Dans un premier temps, elle s'est orientée sur les composants systèmes de l'iPad ainsi que leurs modes d'acquisition. Puis, il a été étudié les méthodes de projections afin de permettre le report de points d'une frame à une autre. Le rapport fini par l'explication de l'algorithme de traitement d'images utilisé pour l'illustration de la méthode développée.

Mots Clefs : Accéléromètre, Gyroscope, Caméra, Traitement d'images, Temps réel, Ordonnancement, Projections orthogonales, iPad, iOS, Objective-C

Abstract :

This report is the ending of the sixth end of study project of the 2011-2012 school year. It's about the conception of a method allowing real time support of image processing algorithms on smartphones. The study of iPad accelerometer and gyroscope is first explained in this report. Then, it describes how projections are performed in order to project points from one frame to an other. To finish with the explanations on which image-processing algorithm is used.

Keywords : Accelerometer, Gyroscope, Camera, Image processing, Real time, Scheduler, Arithmetic projections, iPad, iOS, objective-C