GAY Simon

Rapport de stage 2010

ENSEEIHT

Master Informatique et Télécommunication
Parcours Système Informatique et Génie Logiciel

Année : 2010

Nom du Laboratoire :
AI Lab, université de Zurich
Directeur :
Pr. Rolf Pfeifer

Developmental robotics: towards the development of a body image and tool use

Auteur(s) :
Hugo Gravato Marques et Max Lungarella

Directeur de Recherche :
Pr. Rolf Pfeifer
Responsable du stage :
Dr Hugo Gravato Marques

Nom du Projet :
ECCERobot
Dirigé par :
Owen Holland

Résumé :

Construire l'image de son propre corps à partir de ses observation permet d'obtenir un modèle fiable et utilisable pour planifier ses actions. Le projet ECCERobot tente de copier le fonctionnement du corps humain, notamment sa structure musculaire. La structure du robot, dit "anthropomimétique", est trop complexe pour créer un modèle mathématique réaliste. Une partie du projet consiste donc à permettre au robot de construire l'image de son propre corps à l'aide de ses capteur et notamment de sa caméra. Le stage consiste à permettre au robot de détecter et segmenter automatiquement dans le champ de vision de la caméra les éléments faisant partie de son corps, afin de générer un modèle utilisable dans un simulateur.

Remerciements :

Je tiens à remercier toute l'équipe du laboratoire AI Lab pour leur accueil, l'aide et les conseils qu'ils m'ont apportés tout au long de mon stage.

Je tiens à remercier tout particulièrement le Dr Hugo Gravato Marquès, Juan Pablo Carbajal et Cristiano Alessandro pour l'encadrement, le soutient et l'expérience enrichissante et pleine d'intérêt qu'ils m'ont fait vivre durant ces six mois au sein du laboratoire.

Je tiens également à remercier mon tuteur de stage Jean-Christophe Buisson et le directeur du département informatique Gérard Padiou pour leurs conseils et pour le temps qu'ils m'ont consacré tout au long de cette période.

Sommaire

IV Les expériences

I Introduction

1 Le laboratoire AiLab

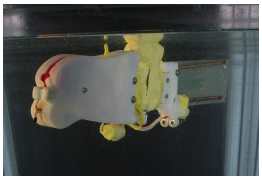
Fondé en 1987, le laboratoire AI Lab fait partie du département d'informatique de l'université de Zurich. Il est actuellement dirigé par le professeur Rolf Pfeifer et emplois actuellement 32 personnes dont 9 doctorants.

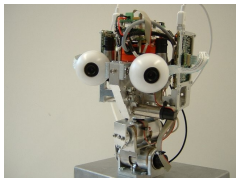
ai lab

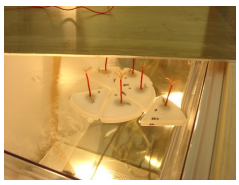
Les sujets de recherches concernent l'étude de l'intelligence sous toutes ses formes, principalement celle liée à l'incarnation (embodiment) dans un corps physique, qui considère que l'intelligence n'est pas seulement due à l'activité cérébrale, mais d'une interaction entre le cerveau, le corps et l'environnement. La plupart des travaux s'inspirent ainsi du vivant pour concevoir, par exemple, des robot capables de se déplacer en marchant, courant, nageant ou sautant avec un minimum d'actionneurs, de se diriger avec un minimum de capteurs et une puissance de calcul réduite, ou créer des capteurs plus performants.

Voici une liste non exhaustive des recherches actuellement en cours :
-"Anthropomimetic robotics", projet dans lequel s'inscrit mon stage, il consiste à construire un robot inspiré par l'anatomie humaine.
-"from locomotion to cognition", étude basée sur la nage de poissons et de quadrupèdes robotiques utilisant un minimum d'actionneur (le robot poisson utilise un unique actionneur pour se déplacer dans l'espace).
-"artificial octopus", construction et étude de la mobilité d'un poulpe artificiel. Il s'agit de recréer artificiellement la structure complexe des céphalopodes, le contrôle du corps (avec son nombre infini de degrés de liberté) et le comportement du poulpe réel.
-"Tribolon" , système d'auto-assemblage constitué de petit modules comportant un ou plusieurs aimants et un moteur engendrant des vibrations, posés sur la surface d'un liquide conducteur (pour l'alimentation). Ce système permet d'étudier l'assemblage de molécules au sein des cellules vivantes.
-"Dynamical Coupling in Motor-Sensory Function Substitution', conception d'une prothèse de main dont le système d'actionneur est inspiré d'une main humaine, contrôlée par le système nerveux du porteur et capable de lui fournir des informations tactiles.

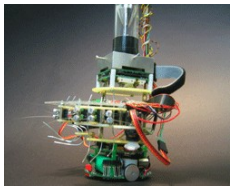
quelques robots :


WANDA


iCub


Tribolon


MiniDog


Artificial Mouse

La liste complète des travaux et des robot est disponible sur la page officielle de l'AI Lab :
http://ailab.ifi.uzh.ch/

2 ECCERobot

Les robot humanoïdes actuels copient la forme humaine, mais pas son fonctionnement, pourtant importante si l'on veut obtenir un comportement réaliste, en vue d'une interaction homme-machine.

Partant de ce constat, le projet de la conception d'un robot anthropomimétique a vu le jour, fondé en 2009 par EU's 7th Framework Programme[1], et soutenu par cinq partenaires : l'université de Sussex[2], le Arificial Intelligence Lab de l'Universitée de Zurich[3], la Elektrotehnicki Fakultet[4], le Robotics and Embedded Systems[5] et le Robot Studio[6].

ECCERobot version 1

Le robot ECCERobot-1 (pour *Embodied Cognition in a Compliantly Engineered Robot)* est un robot humanoïde (et anthropomimétique) constitué d'un torse, d'une tête et de deux bras.

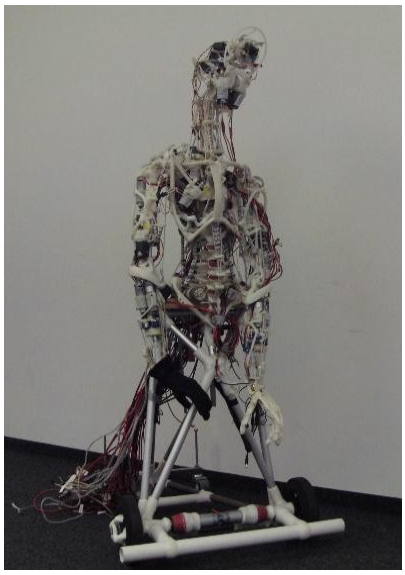
Il fut construit en 2007 dans le cadre du projet CRONOS[7] (Conscious Robot (Or Near Offer) System), basé sur l'étude de la conscience, et réutilisé pour le projet ECCERobot.

Sa structure a été construite à partir d'une matière plastique qui devient malléable à 60°C, et renforcés par des tubes métalliques (notamment au niveau des bras). Ses actionneurs sont constitués de moteurs électriques dont l'axe comporte une poulie autour desquelles viennent s'enrouler des câbles de spectra, reliés à la structure par des tendeurs. On simule ainsi l'élasticité des tendons biologiques. Ces muscles artificiels tentent de reproduire aussi fidèlement que possible la structure musculaire humaine, et confèrent au robot une structure souple.

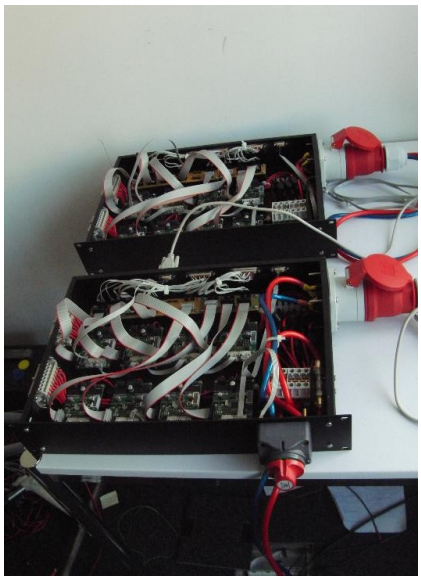
Chaque "muscle" comporte un capteur de rotation sur les poulies et sera prochainement équipé de capteur de pression permettant de mesurer la tension sur le câble.

Les actionneurs sont alimentes par des unités de contrôle électroniques (ECU) spécialement conçu pour ce projet. Chaque carte permet le contrôle de deux moteurs. Ces cartes communiquent avec un PC par le biais d'un bus CAN.

Le système visuel du robot est constitué d'une caméra firewire I-fire Unibrain d'une résolution de 640X480 pixels, et muni d'un objectif grand angle.



ECCERobot sur son support



Les cartes de puissance

Site officiel de ECCERobot :
http://eccerobot.org/

[7]Page du projet CRONOS :
http://www.cronosproject.net/

fondateur du projet :
[1]http://cordis.europa.eu/fp7/home_en.html

partenaires :
[2]http://www.sussex.ac.uk/
[3]http://ailab.ifi.uzh.ch/
[4]http://www.etf.rs/
[5]http://portal.mytum.de/welcome/
[6]http://www.therobotstudio.com/

3 problématique

La construction de l'image de son propre corps permet de planifier des taches dans un environnement et d'interagir avec lui. Des expériences récentes menée sur des singes montrent que ceux-ci sont capable de contrôler un bras mécanique connecté à leur cerveaux. Cette expérience montre ainsi qu'ils sont capable de corréler leur activité cérébrale aux mouvements du bras, donc d'intégrer celui-ci à l'image qu'ils ont de leur corps. Cette image peut aussi être modifiée dans le cas de la manipulation d'un outil, ce dernier s'intégrant dynamiquement à l'image[8].

Un robot intégrant cette faculté [8,9] pourrait utiliser un outil comme une extension de son corps, et pourrait plus facilement planifier des actions dans son environnement[10,11,12].

Le robot ECCERobot disposant d'articulation complexes (puisque basées sur celles d'un être humain) et d'actionneurs élastiques, il est très difficile de définir une modélisation fiable de son corps. Donner au robot la faculté de construire lui-même l'image de son corps permettrait d'obtenir un modèle fiable puisque basé sur ses propres observations.

Dans cette optique, le projet actuel en développement dans le contexte du projet ECCERobot consiste à permettre la construction d'une image du corps du robot en testant la corrélation entre les signaux envoyés aux moteurs et les informations issues des capteurs renseignant l'état des "muscles" artificiels (longueur et tension sur les tendons) et les informations issues de la caméra.

La partie correspondant à mon stage consiste à concevoir un algorithme de vision par ordinateur permettant la segmentation et le suivi de tout objet en mouvement dans le champs visuel du robot, et capable de fournir des informations sur les déplacements. Ceci permettant de déterminer si un objet fait partie ou non du corps du robot par comparaison avec les signaux envoyés aux moteurs.

Les spécifications du projet précisent que le robot doit être capable de distinguer ses deux bras, et de faire la différence entre ses bras et tout autre objet en mouvement. On ne tiendra pas compte des problèmes d'occlusion, tous les objets seront entièrement visible au cours des diverses expériences.

De nombreux travaux on été publiés en matière de vision par ordinateur [1], notamment des algorithmes de segmentation de l'image et de suivi. La segmentation repose dans la plupart des cas sur la densité [2], la couleur ou la texture, ou encore sur l'analyse de la trajectoire de points d'intérêt[3]. Quand au suivi, il repose sur une caractéristique de l'élément à suivre qui le différencie de l'arrière plan, un élément d'image [4], un contour [5], ou sur une forme ou un modèle prédéfini par l'utilisateur [6,7]. Malheureusement, aucun des algorithmes étudié ne répond à toutes les spécifications. C'est pour cela que j'ai décidé de concevoir un nouvel algorithme.

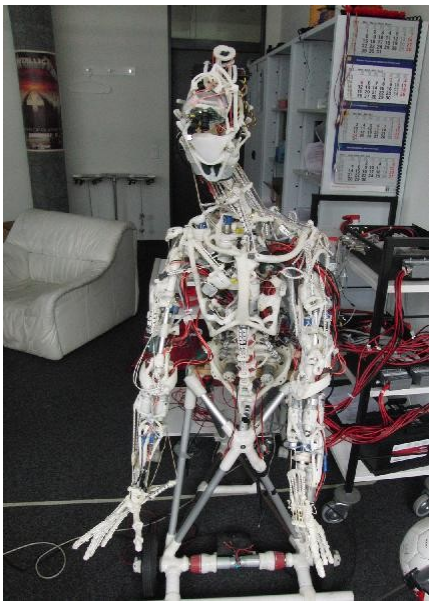


Figure 1 : il est très difficile de segmenter les éléments du robots : ou est la limite entre bras et avant-bras ? Les câbles rouges près des épaules sont-ils sur les bras ou derrière le robot?

Compte tenu de la constitution du robot (petites éléments de différentes couleurs, texture similaire sur toute sa surface), on ne peut utiliser d'algorithme de segmentation basé sur la couleur. La spécification du projet fait que l'on ne peut pas utiliser de reconnaissance de formes, n'importe quel objet mobile pouvant faire partie du robot (par exemple on peut fixer un outil sur sa main). La segmentation et le suivi se feront donc uniquement en étudiant le flot optique et le déplacement des objets.

II Les débuts du projet

La première partie du stage consistait principalement à trouver un algorithme permettant de segmenter les objets mobiles capable de conserver un historique des déplacements.

1 premières tentatives

Les premiers essais utilisaient un algorithme de flot optique de Horn et Schunk. L'idée consistait à détecter et isoler des rectangles contenant des pixels du flot optique non nuls (figure 2). Dans chaque rectangle, on créé un segment correspondant à la courbe de tendance des pixels non nul. Petit à petit, on construit le squelette du robot. Le problème est qu'il est très difficile (voire impossible) de segmenter deux éléments liés lorsqu'ils se déplacent simultanément.

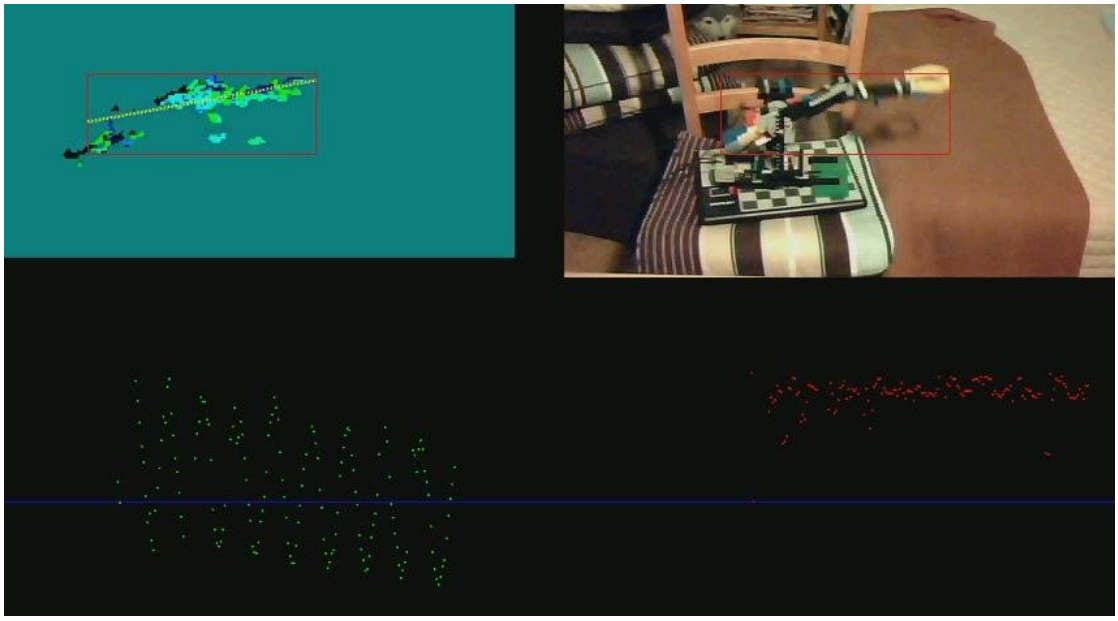


Figure 2 : L'une des premières tentatives. En haut : l'image et le flot optique
en bas : à gauche, l'inclinaison du bras, à droite la longueur.

L'idée suivante était l'utilisation d'un algorithme de substraction d'arrière plan adaptatif. On isole alors des patch correspondant aux objets pour les suivre. Malheureusement, l'utilisation d'un grand nombre de patch est trop lent pour du temps réel, et toute rotation de l'objet rendait les patchs inutilisables.

Mais l'idée de conserver une trace visuelle des objets fut réutilisée, cette fois sous la forme d'un ensemble de points d'intérêt suivant des éléments d'images à l'aide d'un algorithme de flot optique de Lucas-Kanade.

2 choix technologiques

Le langage C++

Le projet devait initialement être codé en langage Java. Mais l'utilisation de la librairie C++ OpenCV s'est avéré par la suite incontournable, notamment pour l'utilisation de l'algorithme de flot optique de Lucas-Kanade. Malgré l'existence de l'interface JNI (Java Native Interface) permettant d'utiliser du code C++ dans un programme Java, j'ai pris la décision de coder mon algorithme entièrement en C++. Outre le fait que ce langage soit plus rapide que le Java, donc mieux adapté à la vision par ordinateur, le projet global dans lequel s'inscrit mon application utilise ce langage, son intégration n'en sera que plus facile.

La librairie opencv

La librairie OpenCV (Open source Computer Vision) est une librairie C++ libre orienté sur la vision par ordinateur, et implémentant plus de 500 algorithme dans ce domaine, notamment des algorithmes de traitement d'image, de segmentation, de suivi, de reconnaissance et de flot optique. Cette librairie s'est donc imposée lorsque j'ai commencé à utiliser l'algorithme de Lucas-Kanade. L'exemple *lkdemo* fournie avec la librairie a d'ailleurs servie de modèle pour mon algorithme.

Je liste ici les structures et fonctions de cette librairie utilisés dans le code et nécessaires pour le comprendre Pour plus de renseignement, se référer à la documentation officielle (http://opencv.willowgarage.com/documentation/index.html):

Les objets de la librairies OpenCV :

IplImage : structure OpenCV d'une image.
CvCapture : objet permettant de lire un flux vidéo.
CvPoint et *CvPoint2D32f* : structure d'un point définissant les coordonnées respectivement avec des entier et avec des réels.
CvSize : structure définissant la hauteur et la largeur d'une image : {int width, int height}

fonctions :

Voici les principales fonctions OpenCV utilisées par l'algorithme de suivi.

cvCalcOpticalFlowPyrLK(const CvArr* prev, const CvArr* curr, CvArr* prevPyr, CvArr* currPyr, const CvPoint2D32f* prevFeatures, CvPoint2D32f* currFeatures, int count, CvSize winSize, int level, char* status, float* track_error, CvTermCriteria criteria, int flags);

prev et *curr* : frames aux temps t et t-1,
prevPyr et *currPyr* : buffer des pyramides des deux frames ,
prevFeatures : vecteur de points dont on veux obtenir le flot optique,
currFeatures : vecteur de points résultats,
count : nombre de points,
winsize : taille de la fenêtre de recherche. CvSize est un objet OpenCV de signature (int x, int y)

level : niveau maximal pour la pyramide de recherche
status : vecteur qui indique pour chaque point si le suivi a réussi,
track_error : vecteur contenant l'erreur des patch d'image de chaque point
criteria : spécification des paramètres de recherche.
flags : autres paramètres

cette fonction implémente l'algorithme de recherche itérative de Lucas Kanade avec recherche pyramidale utilisé pour le suivi des points d'intérêt. L'algorithme de suivi repose en grande partie sur celui-ci.

cvGoodFeaturesToTrack(const CvArr*, image CvArr* eigImage, CvArr* tempImage, CvPoint2D32f* corners, int* cornerCount, double qualityLevel, double minDistance, const CvArr* mask=NULL, int blockSize=3, int useHarris=0, double k=0.04);

image : image en noir et blanc de référence
eigImage et tempImage : image tampon,
corners : liste de points dont les coordonnées sont celles des coins détectés,
cornerCount : pointeur du nombre de points détectés
qualityLevel : seuil minimum d'acceptabilité des coins,
minDistance : distance minimale entre deux points,
mask : (facultatif) utilisé pour spécifier une région d'intérêt
blockSize : taille des blocs
useHarris : défini si on utilise le détecteur de Harris
k : paramètre du detecteur de Harris.

détermine la position de n points d'intérêt facile à suivre, c'est à dire les coins, pour un algorithme de flot optique. Cette fonction est utilisé à l'initialisation pour positionner les points de suivi sur l'image.

cvFindCornerSubPix(const CvArr image, CvPoint2D32f* corners, int count, CvSize win, CvSize zero_zone, CvTermCriteria criteria)*;

permet de replacer un point sur le coin le plus proche de sa position initiale, dans une fenêtre de taille prédéfinie. Cette fonction est utilisée pour le replacement des points.

Autres fonctions utilisées :

cvCreateImage : permet d'initialiser une IplImage.
cvReleaseImage : relâche le pointeur d'une image.
cvNamedWindow : créé une fenêtre.
cvZero : efface une fenêtre.
cvQueryFrame : récupère une IplImage d'un flux vidéo.
cvRectangle : trace un rectangle sur une IplImage.
CvGet2D : récupère la valeur d'un pixel.

OpenRTM

RT middleware est une plateforme de développement orienté sur la robotique (RT signifiant "Robot Technology") dont le but est d'optimiser la programmation en permettant la création de modules que l'on peut interconnecter et réutiliser (RTM facilite la compatibilité entre les modules donc leur réutilisation dans des projets différents). On peut ainsi créer une bibliothèque de modules plus ou moins universels. Le projet ECCERobot utilisant cette plateforme, le code de mon application sera utilisé pour créer un module RTM une fois fonctionnel.

Page officielle du projet OpenRTM : http://www.openrtm.org/OpenRTM-aist/html-en/index.html

III L'algorithme de suivi

Après avoir trouvé une idée d'algorithme qui semblait répondre à tous les critère, la seconde partie du stage pouvait commencer : il s'agissait d'implémenter et d'optimiser cet algorithme. Celui-ci a cependant beaucoup évolué, en fonction des problèmes techniques, jusqu'à la version décrite ici.

1 Le principe

1-1 Le principe de suivi

Afin de conserver un historique des déplacements, il est important de suivre les objets mobiles, même lorsque ceux-ci se sont arrêtés.

Pour cela, on dispose des points d'intérêt sur l'image, puis, grâce à l'algorithme de flot optique de Lucas-Kanade, implémenté dans la fonction *cvCalcOpticalFlowPyrLK,* chaque point va suivre un élément d'image. En analysant les déplacements de cet ensemble de points, on peut segmenter les objets et les suivre. On défini un objet ainsi :

Définition 1 : On défini un objet comme un groupe de points qui se déplacent simultanément.

Soit Pt l'ensemble des points utilisés, pi un point de Pt et $v_{(pi)}$ la vitesse de pi :

$$\text{Objet} = \{ P \subset Pt, \exists i / pi \in P , v(pi)\neq 0 \iff \forall j / pj \in P , v(pj)\neq 0 \}$$

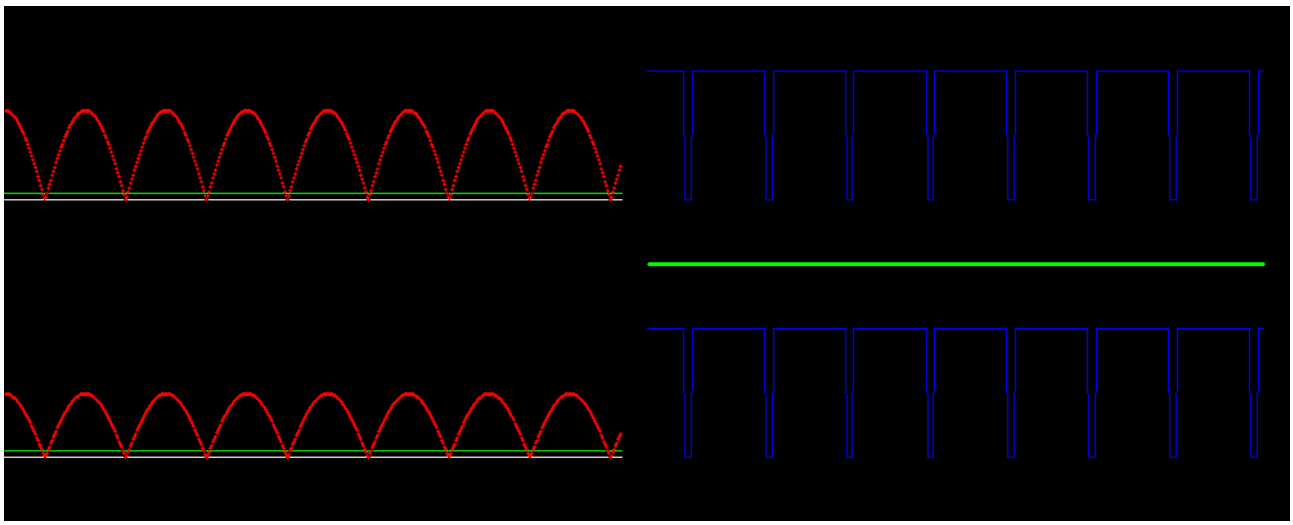
En effet, si un objet se déplace, s'arrête ou change de direction, tout les points placé sur l'image de celui-ci doivent faire de même.

Définition 2 : L'état d'un objet défini si celui-ci est mobile ou immobile.

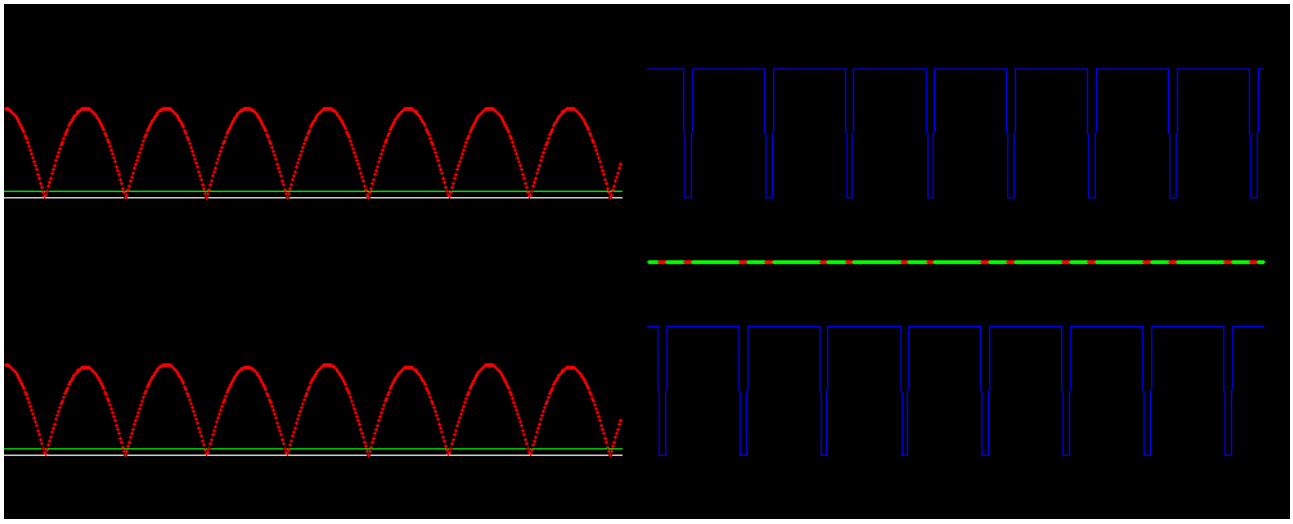
Il en découle les propriétés suivantes

Propriété 1 : un point qui se trouve dans un état différent d'un objet ne fait définitivement plus partie de cet objet.

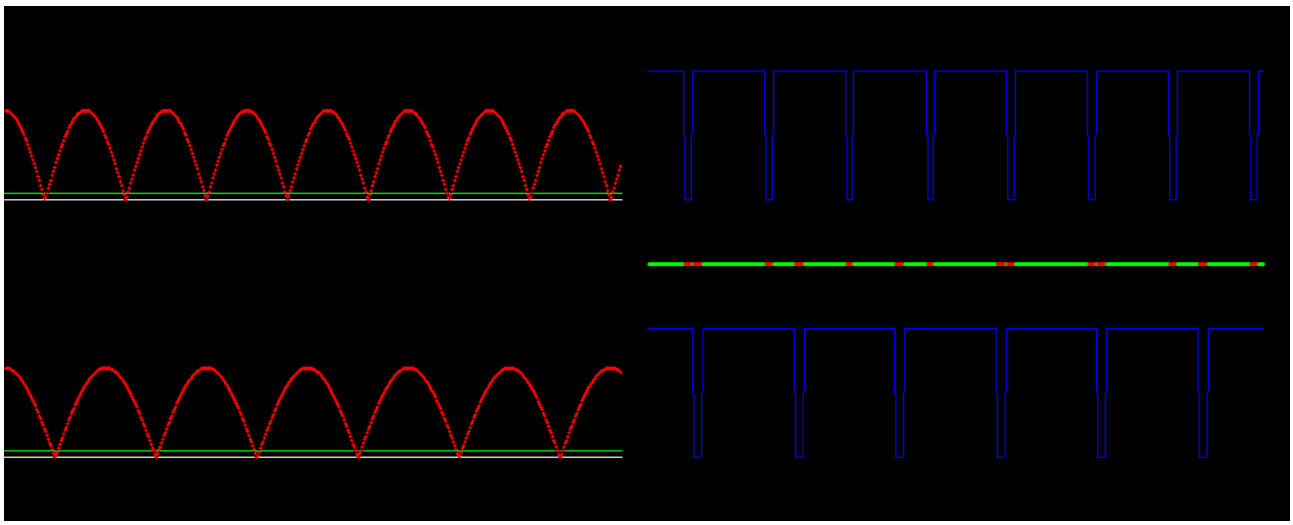
Propriété 2 : deux objets séparés se déplaçant simultanément seront considérés comme un unique objet.



a) deux points se déplaçant simultanément avec des vitesses différentes.



b) même période mais déphasés



c) différentes périodes

Figure 3 : illustration du principe : en rouge la norme de la vitesse de deux points, en bleu la courbe d'état mobile-immobile. La présence de parties rouge sur la courbe centrale indique que les points ne se déplacent pas simultanément. Les points du cas a) font partis du même objet.

Pour suivre un objet, on utilise alors la règle suivante : on compare le nombre de points mobiles et immobiles, pour déterminer si l'objet se déplace ou non. On replace alors tous les points qui ne sont pas dans le même état que l'objet dans une zone proche de celui-ci.

1-2 Gestion des points :

Les points sont gérés différemment si ils font partis de l'arrière plan ou d'un objet. Dans le cas de l'arrière plan, les points sont gérés de manière à optimiser leur répartition sur l'image pour segmenter les objets qui se déplacent. Pour cela, on défini une carte des densités de points. Si un mouvement est détecté, par comparaison de deux frames successives, et que la densité de point est faible, des points choisis parmi les points immobiles sont repositionnés dans cette zone. Ceci permet de s'assurer qu'un nombre suffisant de points se déplaceront pour qu'un objet sont détecté.

Afin d'éviter que les points ne soient «balayés» par le passage d'un objet, les points recouverts par un objet sont immobilisés, et ne reprennent le suivi de l'image qu'une fois qu'ils ne sont plus occultés.

Pour cela, on calcule la distance du point au centre de l'objet. Si celui-ci est dans l'ellipse de covariance des points d'au moins un objet, on verrouille le point sur sa position et on ne tient plus compte de son état. Lorsqu'il n'est plus dans aucune ellipse, le point reprend le suivi de l'élément d'image sur lequel il était lorsqu'il a été libéré.

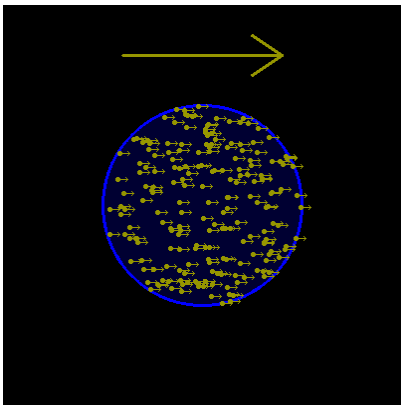
Les points appartenant à un objet sont gérés selon la règle décrite précédemment : une fois l'état de l'objet connu (mobile ou immobile), les points qui ne sont pas dans le même état sont replacés à proximité du centre de l'objet. Pour choisir la nouvelle position, chaque objet dispose d'une carte de densité indiquant la densité des points qui sont dans le même état que lui. Si l'objet est mobile, on replace aléatoirement le point dans une zone un peu plus grande que celle où la densité est considérée comme suffisante pour faire partie de l'objet. Ceci permet de couvrir petit à petit les régions de l'objet qui apparaissent (en cas de rotation, notamment), les points replacés sur l'arrière plan où un autre objet seront replacés à nouveau à l'étape suivante. Dans le cas contraire, où l'objet est immobile, la zone de replacement est seulement définie par la région de forte densité. Il est en effet difficile de déterminer si un point immobile de l'image fait parti de l'objet ou de l'arrière plan, on ne replacera donc les points que dans la zone où l'on est sûr que l'objet se trouve. Ceci nous amène à la propriété suivante :

Propriété 3 : lorsqu'un objet est immobile, la surface de l'objet virtuel qui le suit ne peut que diminuer.

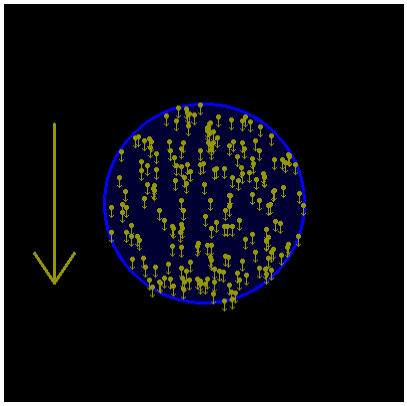
1-3 Déterminer les déplacements

La mesure des déplacements (en translation et en rotation) de l'objet, s'effectue en analysant le déplacement des points en fonction de leur position dans l'objet. Les résultats seront d'autant plus précis que les agents seront nombreux et uniformément répartis.

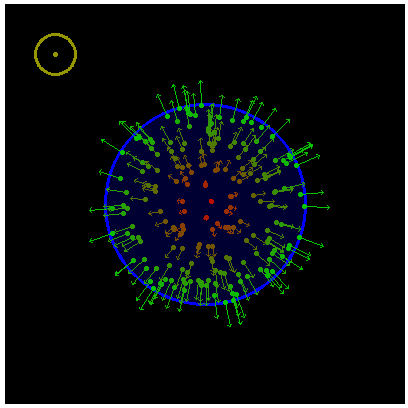
La figure 4 montrent le flot optique donné par une sphère lorsqu'elle se déplace en translation ou en rotation suivant les trois axes. En considérant que tout objet peut être assimilé à un ellipsoïde, on peut estimer les déplacement sur les trois axes.



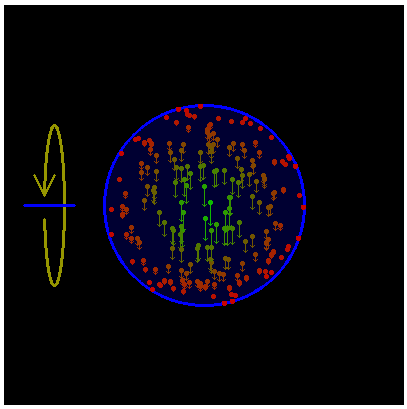
a) translation d'axe X



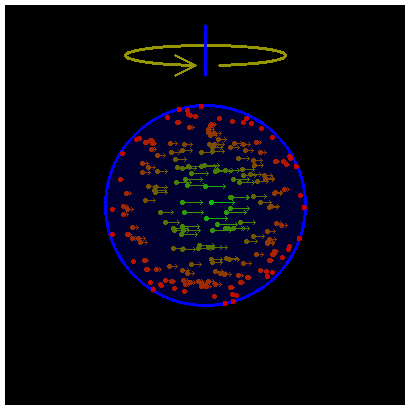
b) translation d'axe Y



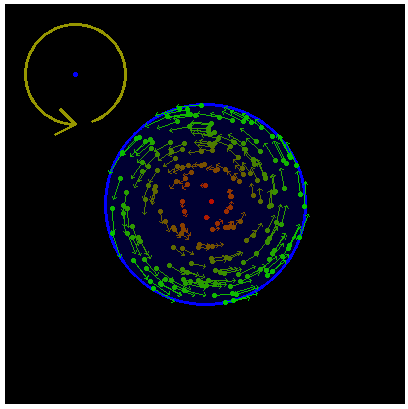
c) translation d'axe Z



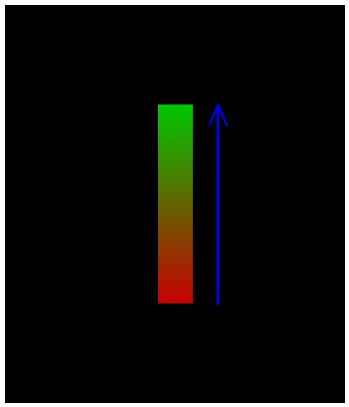
d) rotation d'axe X



e) rotation d'axe Y



f) rotation d'axe Z



g) échelle de couleur des vitesses

Figure 4 : flot optique d'une sphère selon son mouvement.

-Translation selon les axes X et Y :
on obtient la vitesse linéaire dans le plan en faisant la moyenne des vitesse des points. On ne peut utiliser la vitesse du centre de l'objet car sa forme peut être modifié, par une rotation par exemple.

$$Vx = \frac{1}{n} \cdot \sum_{i=0}^{n} Vx_i$$

$$Vy = \frac{1}{n} \cdot \sum_{i=0}^{n} Vy_i$$

-Rotation d'axe Z :
elle s'obtient en faisant la moyenne des vitesses tangentielles des points dans un référentiel polaire. La valeur obtenue ne sera précise que si la rotation est uniquement d'axe Z, autrement, on n'obtiendra qu'une approximation. On calcule la moyenne des composantes tangentielles des vitesses des points :

soit $[\ Vr\ \ V\theta\ ]^T = R(\theta) * [\ Vx\ \ Vy\ ]^T$

$$Rz = \frac{1}{n} . \alpha . \sum_{i=0}^{n} \frac{Vt_i}{di}$$

d_i distance au centre de l'objet

Afin d'obtenir une bonne précision, on ne prendra en compte que les points dont la distance au centre est supérieure à une limite d_{min}

-Rotations d'axes X et Y :
cette mesure ne sera précise que si l'objet a une forme d'ellipsoïde, elle permet cependant de détecter le sens de la rotation, même si l'objet a une faible épaisseur. On compare cette fois la vitesse en fonction de leur éloignement au centre de l'objet: en effet, si l'objet est arrondi, les points situé au centre se déplaceront à une vitesse différente des points périphériques (ces derniers seront même immobile en cas de rotation sans translation).

On peut approximer les vitesses de rotation par :

$$Rx = \frac{1}{n} . \beta . \left(\sum_{i=0}^{n} Vx_i . \frac{di}{r_{(\theta)}} - \sum_{i=0}^{n} Vx_i . (1 - \frac{di}{r_{(\theta)}}) \right)$$

$$Ry = \frac{1}{n} . \beta . \left(\sum_{i=0}^{n} Vy_i . \frac{di}{r_{(\theta)}} - \sum_{i=0}^{n} Vy_i . (1 - \frac{di}{r_{(\theta)}}) \right)$$

θ angle du point en coordonnées polaires en prenant les axes de l'ellipse pour référentiel.
$r_{(\theta)}$ distance du centre à l'ellipse en fonction de θ.

-Translation d'axe Z :
Là encore, on ne pourra pas obtenir de valeur précise, seulement une approximation. On calcule la moyenne des composantes axiales des vitesses des points.

$$Vz = \frac{1}{n} . \gamma . \sum_{i=0}^{n} \frac{Vr_i}{di}$$

1-4 Les limites de l'algorithme

Le suivi des points s'effectuant par l'algorithme de Lucas-Kanade, on ne pourra pas détecter les déplacements si l'objet est de couleur unie, ou si la vitesse devient si importante que le point à suivre se retrouve en dehors de la zone de recherche.

Pour le calcul des déplacements, on considère que chaque objet est assimilable à un ellipsoïde. La précision des résultats dépend donc de la forme de l'objet et de sa ressemblance avec un ellipsoïde.

1-5 Principe de la segmentation

Maintenant que nous savons comment suivre un objet, voyons maintenant comment les segmenter.

Il était prévu dans un premier temps d'utiliser un maillage, chaque point étant relié au point le plus proche dans chacune des quatre directions haut, bas, droite et gauche. Si deux points liés se déplacent à des vitesses trop différentes, le lien est rompu. Lorsqu'un maillage se détache du maillage principal, on créé un nouvel objet (figure 5). Bien qu'efficace, la recherche des points voisins et le parcours du maillage à chaque étape ralentissait le système (retard supplémentaire de plus d'une seconde), il fut remplacé par un principe plus simple et surtout beaucoup plus rapide : parmi les points mobiles, on peut déterminer ceux appartenant à un nouvel objet lorsque celui change d'état. Ainsi, si un nombre suffisant de points passe de l'état immobile à mobile en même temps, on créé un nouvel objet à partir de ces points. Ce système n'est bien entendu pas exempt de défaut : deux objets non liés mais se déplaçant simultanément seront considérés comme un unique objet, et un nouvel objet ne pourra être détecté que lorsqu'il change de direction (pas de mouvement ou de rotation continue). Mais compte tenu des spécifications du projet, ces inconvénients ne sont pas très gênants (on utilise seulement des mouvements périodiques, et les objets ne sont jamais totalement synchronisés).

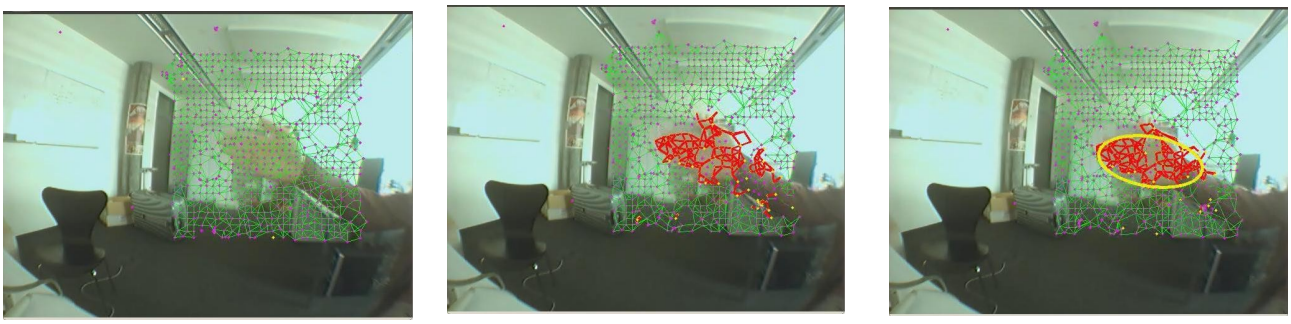


Figure 5 : La première méthode employée utilisant un maillage. En rouge les sous-maillages séparés du principal, efficace mais trop lent...

Malgré cela, deux objets séparés mais considérés comme un seul et même objet ne serait pas problématique puisque, rappelons-le, le but du projet est de lier les signaux de commandes aux déplacements vues par la caméra.

2 Implémentation

Cet algorithme de suivi est implémenté à l'aide d'un système multi-agents comprenant deux types d'agents : des agents simples et très nombreux (moutons) dont le but est de suivre un point donné en filtrant les mouvements parasites et mesurer vitesse, position et états de celui-ci , et des agent plus complexes et peu nombreux (bergers), attribués à chaque objet et dont le but est de tirer des résultats statistiques des nombreux agents moutons de leur objet. Ces bergers doivent aussi contrôler leur groupe et repositionner les moutons en cas de valeurs erronées, ou simplement pour mieux les répartir.

2-1 Les cartes

Les cartes sont des matrices de tailles (largeur/10)X(hauteur/10) , dont chaque élément contient une caractéristique d'une cellule de taille 10X10 pixels de l'image. Ces cartes permettent de simplifier les calculs de l'algorithme en considérant cette hypothèse : les propriétés des éléments d'une même cellule sont considérées comme identiques.

Les cartes de l'arrière plan :

objectMap : contient le numéro de l'objet occupant les cellules, -1 si la cellule est vide. Elle permet d'éviter que les objets se chevauchent et de dessiner les surfaces des objets sur l'image.
motionMap : utilisé pour replacer les points de l'arrière plan, les cellules valent 0 si du mouvement est détecté, que les objets ne recouvrent pas la cellule et que celle-ci ne contient pas d'agents, la valeur est supérieure à 0 sinon.
motionMap2 : utilisé par les objets, défini si du mouvement a été détecté dans une cellule.

les cartes des objets :

aMap2 : défini le nombre d'agents de l'objet par cellules.
aMap3 : défini les cellules où les points peuvent être replacé, c'est à dire celles contenant des agents dans le même état que l'objet, et dans le cas où celui-ci est mobile, les voisines des cellules précédentes, afin d'augmenter la surface. La valeur d'une cellule est supérieure à 0 si la cellule fait partie de la zone de replacement.

2-2 Les classes

Les classes principales :

Classe Agent (moutons) :
chaque mouton se voit attribuer un point. Comme ceux-ci sont rangé dans une liste afin de pouvoir être utilisé avec la fonction du flot optique (voir partie II-2), on donne simplement l'indice du point dans la liste.

attributs :

```
float x,y;          // coordonnées actuelles du mouton ( t )
float x0,y0;        // coordonnées de l'étape précédente ( t-1 )
float xo,yo;        // coordonnées de l'avant-dernière étape ( t-2 )

float x1,y1;        // coordonnées actuelles du point
float x2,y2;        // coordonnées de l'étape précédentes

int number;         // position du point dans la liste
int position;       // position du mouton dans sa liste (liste du background ou d'un objet)

float dx ,dy ;      // vecteurs vitesse actuelle sur les axes x et y
float dx2,dy2;      // vecteurs de la vitesse à l'étape précédente
float s;            // norme de la vitesse
```

```
bool move,move0,move1;   // état de mouvements aux frames t, t-1, t-2
bool locked;             // verrouillage du point
bool init1,init2;        // agent replacé récemment
bool stop0,stop1,stop2;  // arrêt ou changement de direction aux frames t, t-1, t-2
                         // ces variables ne sont pas les opposés des variables move, en cas de
                         // changement de direction, move et stop auront pour valeur true.
```

fonctions:

```
// constructeur
   Agent();

// initialisation du mouton
   void setAgent(int num, CvPoint2D32f point);
```

num correspond au numéro de l'agent, permettant de récupérer son point dans la liste *point* et d'initialiser un pointeur sur celui-ci.

```
// étape d'un mouton de l'arrière plan, détermine la vitesse du point en filtrant les parasites
   bool step(int width, int heigth, int **agentMap, CvPoint2D32f* points,IplImage* grey);

// étape d'un mouton d'un objet, détermine la vitesse du point en filtrant les parasites
   bool stepObj(int width, int heigth, int **aMap, CvPoint2D32f* points, float x, float y, IplImage*
grey);

// replace un mouton sur une position favorable au suivi proche de (i,j)
  void replace(float i, float j, int** aMap, CvPoint2D32f* points, IplImage* grey);

// dessine un point sur l'image en paramètre (img) à la position du mouton
  void drawAgent(IplImage* img);
```

Filtrage des mouvements parasites :

L'algorithme du flot optique peut générer des résultats erronés, notamment si l'élément à suivre ne contient que peu de détails, ou si un objet se déplace trop rapidement. Le point peut alors osciller entre deux éléments d'image similaires d'une frame à l'autre. Le filtre permet de réduire ces mouvements parasites.

Principe du filtre : la vitesse est calculée en comparant les positions de l'agent à la frame courante et celle de l'avant dernière frame, afin de doubler la précision, et d'éliminer les oscillations.

Le filtre est aussi chargé de replacer le point si celui-ci sort de l'image, au centre de l'image dans le cas d'un point de l'arrière plan (événement rare donc le replacement est peu important), où à proximité du centre d'un objet (dans ce cas on cherche un élément facile à suivre à l'aide de la fonction *cvFindCornerSubPix*).

Déterminer l'état de l'agent :

On détermine l'état de l'agent en comparant le vitesse obtenue à un seuil, pour éliminer d'éventuels parasites. Pour pouvoir être considéré comme mobile ou immobile, l'agent doit rester dans un même état sur deux frames consécutives. Dans le cas d'un changement d'état, les points ne

seront pas replacés.

A cause du filtre, les points d'un même objet ne changeront pas d'état exactement en même temps, mais parfois avec une frame de décalage, c'est pourquoi on conserve l'état des agents des trois dernières frames.

Pour détecter un changement de direction, on étudie l'accélération :
si le segment [($Vxt\text{-}1_i$,$Vyt\text{-}1_i$), (Vxt_i,Vyt_i)] coupe le cercle de centre (0,0) et de rayon r_{min}, on considère qu'il y a eu changement de direction. Lorsque cet événement a lieu, on considère que l'agent est arrêté.

Gestion des points de l'arrière plan :

Le replacement des points sur les régions en mouvement :

Afin de simplifier les calculs, on utilise en guise de carte de densité la carte motionMap2 décrite précédemment. On compte le nombre de points dans chaque cellules. Si on détecte du mouvement dans une cellule, par comparaison de deux frames successives, et que la cellule est vide, un point parmi les points immobiles est placé dans cette cellule à l'aide de la fonction *cvFindCornerSubPix*.

Création d'un nouvel objet :

On compte le nombre de points qui sont passés de l'état immobile à mobile à la frame courante ou la frame précédente. Si ce nombre est supérieure à un seuil définissant le minimum de points pour permettre de déterminer translations et rotations, on défini un nouvel objet avec l'ensemble de ces points.

Si les conditions sont réunies, la fonction *setObject* retire les agents satisfaisant les conditions citées précédemment de la liste des agents de l'arrière-plan et constitue la liste d'agents de l'objet, puis initialise les variables de celui-ci.

Classe Object (bergers) :

attributs :

```
Agent* sheep;                  // liste des agents (moutons) de l'objet
int nbAgents;                  // nombre d'agents

int *score;                    // poids de chaque agents

float x,y,z;                   // position du centre de l'objet aux points instants t et t-1
float x0,y0,z0;

float rx,ry,rz;                // position angulaire, rx et ry obtenues par intégration de drx et
                               // dry, rz angle de l'ellipse de covariance

float sx,sy,sz;                // taille de l'objet, sz n'étant pas utilisé

float dx,dy,dz,drx,dry,drz;    // vitesses linéaires et angulaires de l'objet

bool move0,move1,move2;        // objet en mouvement aux instant t, t-1 et t-2
```

```
int **aMap,**aMap2,**aMap3;   // cartes de densité des agents

float rotZ;                   // angle d'axe z obtenu par intégration de drz

int number;                   // numéro de l'objet

Covariance *cov1c, *cov1p;    // objets permettant de définir la matrice de covariance
                              // de différents groupes de points de l'objet
Covariance *cov0, *cov4;

Ellipse* ellips0,*ellips4 ;   // ellipses de covariance utilitaires

Ellipse* ellips1p,*ellips1c;  // ellipse de covariance des points périphériques et centraux,
```

fonctions :

```
// constructeur
   Object();

// initialisation de l'objet
   void setObject(IplImage *img,int*** agentMap,Agent* agentList,int*counter,int n);

// calcule l'état des agents
   void compute(CvPoint2D32f* points, IplImage * image, IplImage *grey);

// dessine l'objet
   void drawObject(IplImage *img);

// calcule les nouvelles coordonnées et vitesses de l'objet et dessine celui-ci
   void drawObject(IplImage *img,IplImage* vect,IplImage* vect2,IplImage *vect3,IplImage
*vect4, CvPoint2D32f* points, IplImage *grey, int** objectMap, float *errors,int**
motionMap,int*** textureMap);
```

Gestion des points d'un objet :

Chaque berger utilise une carte de densité de ses propres points, définissant les propriétés de cellules de 10X10 pixels. Chaque cellule note le nombre de points qu'elle contient. Afin de simplifier les calculs, on considère la propriété simplificatrice suivante: tous les points d'une même cellule font partis d'un même objet. Ainsi, si une cellule contient au moins un point qui se déplace comme la majorité, les autres points ne seront pas déplacé. On améliore ainsi la résistance aux bruits.

Un point peut être replacé si l'une de ces trois conditions est remplie :

si le point est immobile et la majorité des points se déplace

$$p \not\Subset Pt \;,\; card(P) > \tfrac{1}{2} * card(Pt) \Rightarrow p^{t+1} = replace(p^t)$$

si le point se déplace et la majorité des points est immobile

$p \in Pm$, $card(P) < \frac{1}{2} * card(Pt) \Rightarrow p^{t+1} = replace(p^t)$

si la densité de points dans une cellule est trop importante

$aMap2(E(x/10), (E(y/10))) > d_{max} \Rightarrow p^{t+1} = replace(p^t)$

Cette dernière condition est particulièrement importante pour les rotations d'axes X et Y. En effet, pendant la rotations, les points de la face visible s'accumulent sur le bord de l'objet, tandis que la face qui apparaît ne comporte aucun point. Cette troisième condition permet de replacer les points autour de la dernière position connue de l'objet. Ceux qui seront replacé sur l'objet continueront le suivi, ceux qui seront replacé sur l'arrière plan seront replacés aux étapes suivantes (conditions 1 et 2).

On notera que pour déterminer l'état de l'objet, on n'utilise pas la majorité comme seuil. En effet, lorsqu'un objet se déplace, il y a en permanence des points replacés en dehors de celui-ci, ce qui peut fausser le résultat. Ce seuil ne doit pas être trop faible, car si l'objet est immobile, et qu'un autre objet passe à proximité, récupérant quelques points de l'objet virtuel (le berger), ce dernier pourrait détecter un état mobile et suivre le second objet. Les résultats expérimentaux montre qu'un seuil de 60% de points immobiles est un bon compromis. Pour éviter une oscillation entre les états mobiles et immobiles, notamment lors d'une transition, on utilisera pour la condition 2 un second seuil de 80% de points immobiles. Celui-ci est élevé pour éviter qu'un objet de petite dimension ne soit perdu par son berger. Entre ces deux seuils, les conditions 1 et 2 ne sont pas utilisées.

On défini les zones de replacement des points de la manière suivante :

dans le cas où l'objet est immobile, cette zone est défini en replissant la carte aMap3 avec les conditions suivantes : les cellules ont une valeur non nulle si elles contiennent au moins un point immobile ou qui a changé de direction sur les frames t, t-1 et t-2.

Dans le cas où l'objet est mobile, on remplis la carte aMap3 avec ces conditions : une cellule a une valeur non nulle si elle contient au moins un agent qui s'est déplacé aux frames t, t-1 ou t-2, et si on y a détecté du mouvement (on s'aide de la carte motionMap2), ou si la cellule est voisine d'une cellule satisfaisant ces conditions. On permet ainsi d'éviter de disperser les points que l'on replace sur des surfaces qui ne peuvent appartenir à l'objet (puisque immobiles) et on défini une zone de recherche permettant de déterminer le contour de l'objet après une rotation.

Les conditions 1 et 2 peuvent alors être redéfinies par cette règle unique :

un point est replacé si il est dans une cellule de aMap3 de valeur nulle.

$aMap3(E(x/10), E(y/10)) = 0 \Rightarrow p^{t+1} = replace(p^t)$

Dans le cas où l'état n'est pas défini (entre les deux seuils), la troisième règle utilise la carte aMap2, dont les cellules ont une valeur non nulle si elles contiennent au moins un agent.

Si un point doit être replacé, on choisi une cellule de valeur non nulle au hasard, puis une position au hasard dans la cellule. La fonction *cvFindCornerSubPix* permet de placer le point sur un élément de l'image proche de cette position facile à suivre.

les autres classes de l'application :

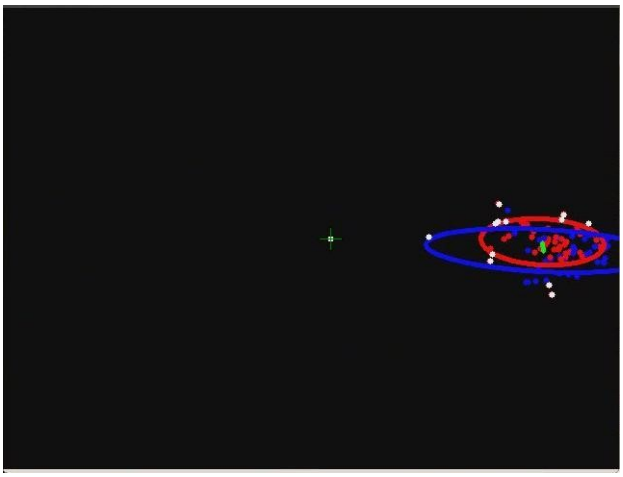
Covariance : défini et calcule la matrice de covariance d'un ensemble de valeurs, en l'occurrence, les coordonnées des agents, et de retourner les résultat sous forme d'une *Ellipse*.
Ellipse : simple ensemble de valeurs permettant de stocker les position du centre, la longueur des deux axes et l'orientation de l'ellipse définissant un objet.

2-3 déterminer les mouvements

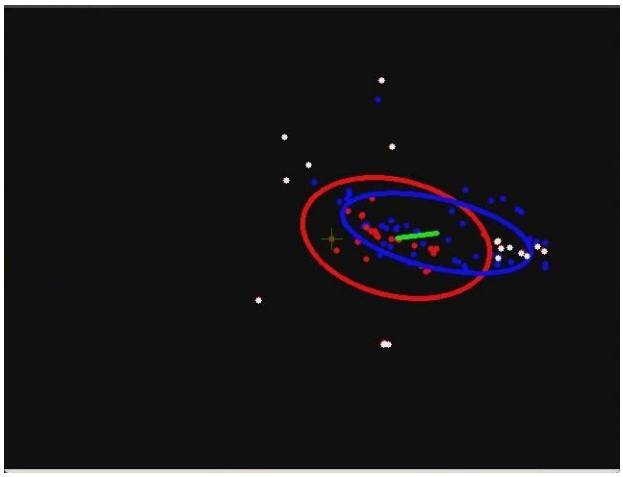
Les valeurs de Vx, Vy, Vz et Rz s'obtiennent d'après les formules décrite précédemment. On notera simplement que pour Vz et Rz, on ne tient compte de la vitesse des points que si la distance au centre est supérieure à un certain seuil, et la vitesse calculée si le nombre de points utilisé est suffisant.

Pour Rx et Rz, on utilise une simplification : pour chaque point, on calcule la distance au centre, puis on sépare les points en deux groupes : les points situés au centre et ceux situés en périphérie. Les vitesses angulaires Rx et Ry sont obtenues en comparant les vitesses moyennes de ces deux groupes.
On ne peut bien entendu pas obtenir des valeurs précises, on peut en revanche obtenir le sens et les variations de la rotation.

Les captures de la figure 6, obtenues pendant un test, montrent la différence entre une rotation et une translation :
chaque point représente l'extrémité du vecteur vitesse d'un agent (le centre (0, 0) du graphe est en vert).
Les points de la périphérie de l'objet sont en rouge, les points du centre en bleu. Les ellipse représentent les ellipse de covariance de leur groupe de points respectifs. Le segment vert montre la différence des moyennes de ces deux groupes.



a) Mouvement de translation :
l'ensemble des points se déplace uniformément .



b) mouvement de rotation :
Les vitesses des points périphériques couvrent une plage plus vaste contenant la valeur nulle

Figure 6 : comparaison de répartition des vitesses d'une translation et d'une rotation

2-4 représenter les objets :

L'affichage a été organisé pour afficher les données les plus pertinentes. Il se compose de deux parties : l'affichage principal et l'affichage "objet".

L'affichage principal affiche les images du flux vidéo, auxquelles on ajoute les aires d'influence des objets. On représente ces aires en ajoutant à l'image des carrés de dimension 10X10 pixels aux positions définies par les cellules de la carte *objectsMap* de valeur différentes de -1.

La couleur des carrés dépend de la valeur de la cellule correspondante, donc du numéro de l'objet.

On défini les couleurs suivantes :
objet 0 : bleu
objet 1 : vert
objet 2 : rouge
objet 3 : mauve
objet 4 : bleu clair
autres : jaune

On termine cet afficheur avec l'ellipse de covariance de l'aire d'influence de chaque objet, permettant de vérifier que l'objet virtuel suit bien l'objet mobile.

afin de pouvoir étudier et vérifier le suivi des vitesses et position des objets, j'ai créé un système d'affichage intuitif utilisant un système de cardans permettant de suivre qualitativement l'évolution des valeurs de Rz, Rx, Ry, et Vz (figure 7). Le centre du cardan donne la position (X, Y) de l'objet.

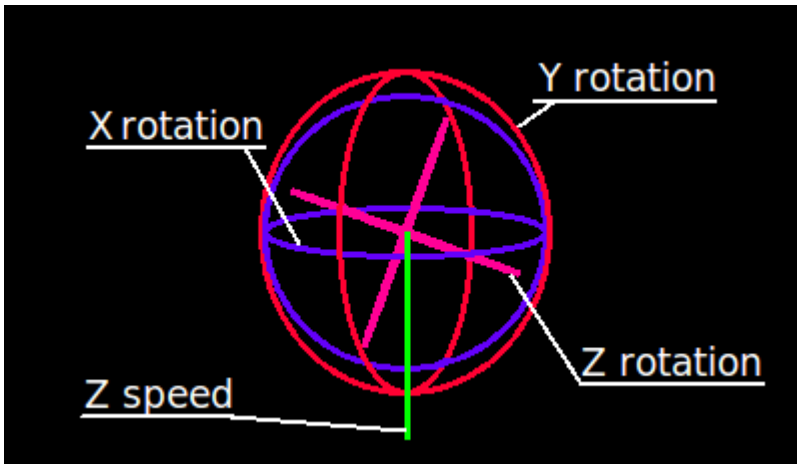


Figure 7 : le cardan

Les trois cardans tournent autours de leur axe respectif, conformément à la figure, l'axe vert est le vecteur Vz, positif si dans la partie supérieure, négatif sinon.

L'afficheur Objet apparaît dans une seconde fenêtre : pour chaque objet, on affiche son ellipse de covariance, en blanc, ses points, jaune si ils sont mobiles, violets si ils sont immobiles, et le cardan.

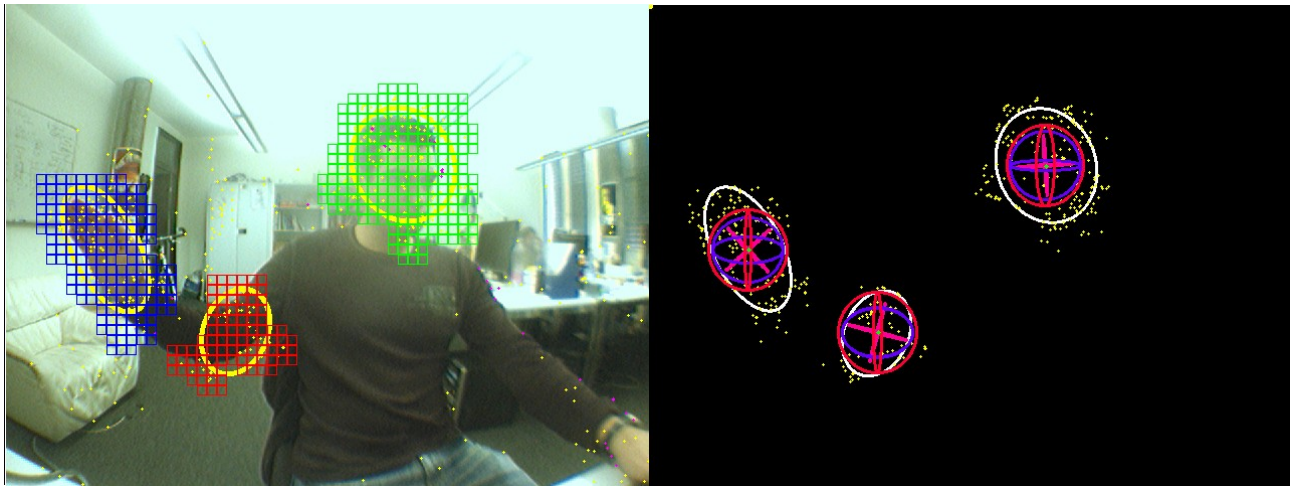


Figure 8 : à gauche l'affichage principal, à droite l'affichage "objets"

Ces deux systèmes d'affichage permettent de suivre visuellement le bon fonctionnement de l'algorithme pendant son exécution.

D'autres afficheurs ont été utilisés pendant la mise au point de l'algorithme, notamment l'affichage des vecteurs vitesses, visible sur la figure 6.

2-5 La lecture d'un flux vidéo :

OpenCV permet d'utiliser un flux provenant d'une caméra ou d'un fichier vidéo au format AVI .

On commence par créer l' objet OpenCV CvCapture, définissant le flot que l'on veut lire, à l'aide de la fonction correspondante :

cvCreateCameraCapture(nb) permet de définir un flot provenant d'une caméra USB . nb correspond au numéro de la caméra. si on ne le connait pas, on passe la valeur -1, la fonction choisira la première caméra trouvée.

cvCaptureFromAVI(path) permet d'obtenir un flot provenant d'une vidéo au format .avi . Le chemin doit être de type char* (et non un String). Cette fonction a été utilisé pour les premières experiences utilisant des vidéos.

Enfin, cvCaptureFromCAM(CV_CAP_ANY) permet de définir un flot provenant d'une caméra firewire. C'est cette fonction qui sera utilisé pour le test utilisant le robot.

Pour lire le flot et obtenir une image, on utilise la fonction *cvQueryFrame(capture)*. Celle-ci retourne une image sous la forme d'une IplImage.

L'inconvénient du flot de la caméra firewire est que les images sont stockées dans un buffer. La lecture du buffer étant plus lente que le nombre de nouvelles frames par secondes, la file d'attente s'allonge, occupant inutilement la mémoire et accumulant du retard à l'affichage.

Pour contourner le problème, on créé un thread qui va lire les images à intervalle régulier, et les copier dans une IplImage partagée. On permet ainsi au thread principal de toujours avoir la dernière image du flux vidéo.

2-6 Intégration dans un module RTM :

Une fois l'application fonctionnelle, le code a été intégré dans le module *MotionBasedTracking*. Pour cela, on ajoute à l'interface de la fonction principale des ports de communication ainsi que les *consumer* nécessaire à la lecture des données, et un port de sortie. On initialise ces ports dans le constructeur du composant.

Le composant dispose d'une sortie (appelée *dataPort*) permettant de communiquer les déplacements des objets.

L'envoi s'effectue à chaque frame selon la séquence suivante :

nombre d'objets détectés, puis pour chaque objet, la séquence : positions X, Y, Z, angles Rx, Ry, Rz, taille des axes X, Y, Z (toujours égal à 0), vitesses sur les axes X, Y, Z, vitesses angulaires sur les axes X, Y, Z.

le vecteur comporte ainsi $1+Nb_{Obj} * 15$ éléments qui seront enregistrés par un module dédié à la lecture.

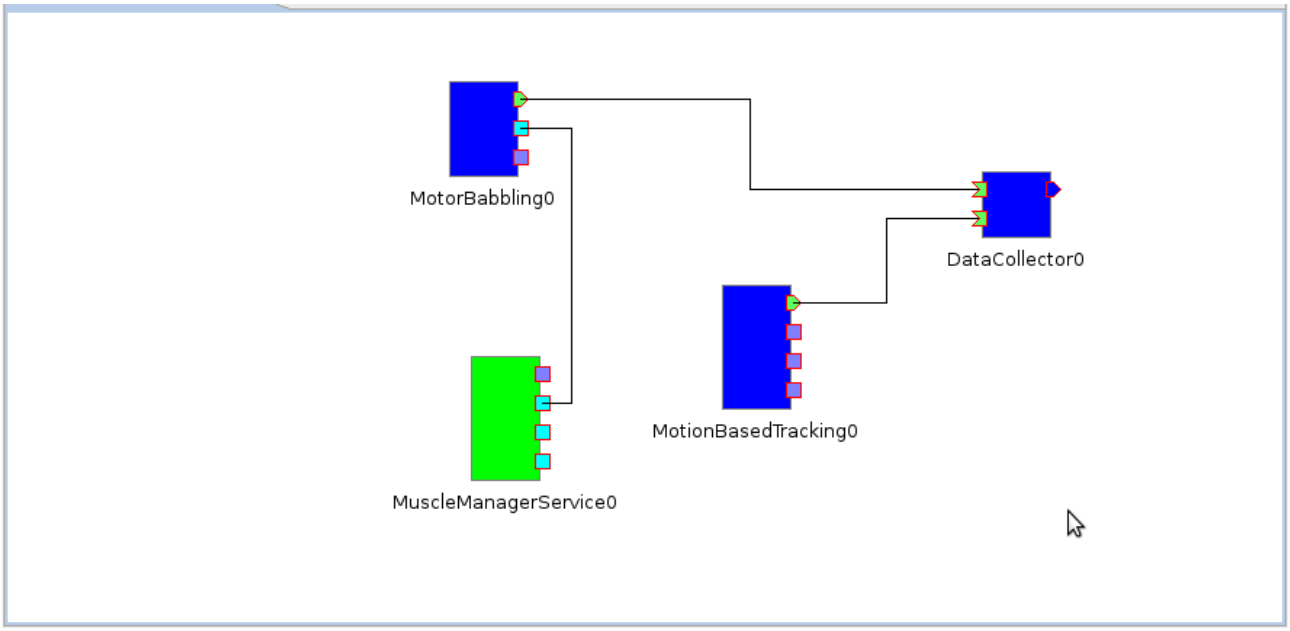


Figure 9 : le réseau RT complet.

Le module dispose de trois ports de communication non utilisés par l'algorithme. Celui-ci devait initialement contrôler directement les moteurs, cette tâche fut par la suite confié à un autre module : le composant *MotorBabbling*. Il est en effet important que l'envoi des commande et le suivi soient indépendant pour que les résultats ne soient pas influencés par les commandes.

Le réseau RT complet présenté figure 9 comprend quatre modules :

DataCollector : collecte les données envoyés sur ses deux entrées et les sauvegarde dans un fichier .log .
MotionBasedTracking : le composant de l'algorithme.
MotorBabbling : modue utilisé pour commander les moteurs. Il transmet la valeur des signaux à l'interface et au module DataCollector.
MuscleManagerService : interface permettant de piloter les moteurs, c'est lui qui envois les commandes sur le bus CAN .

IV Les expériences

1 Les premiers tests de l'algorithme

1-1 expérience 1, test de la segmentation

cette première expérience a pour but de tester l'efficacité de la segmentation de plusieurs objets, leur suivi et la détection des mouvements en translation et en rotation sur trois axes.

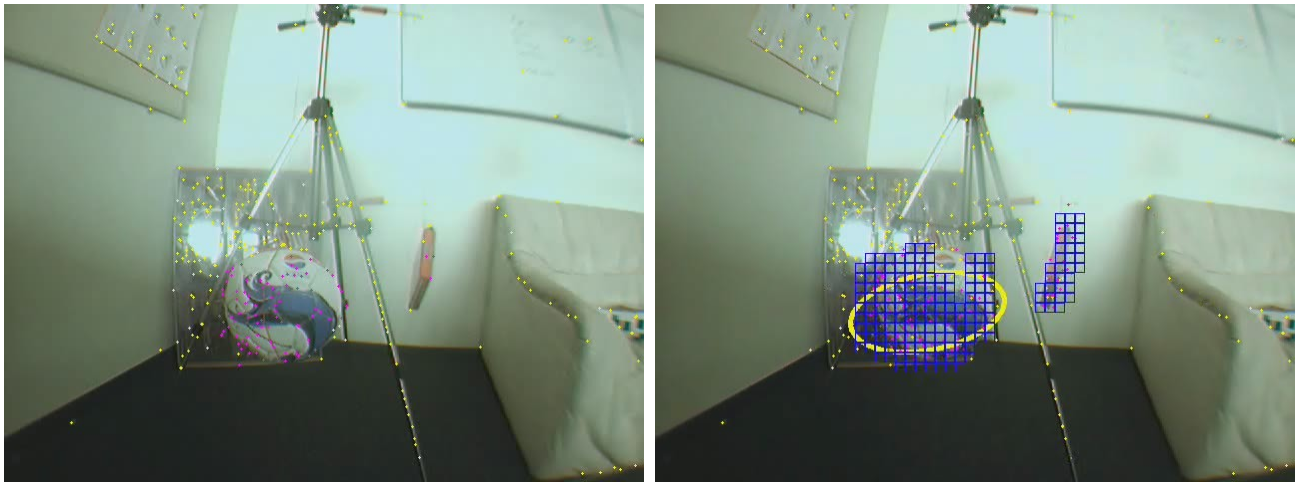
Conditions de l'expérience : deux objets, un ballon et une boite en carton, sont suspendus par une corde à un support, devant un arrière plan encombré. On donne à ces deux objets un mouvement dans des plans différents.

On enregistre cinq séquences vidéo d'une minute environ, que l'on utilise à la place du flux vidéo provenant de la webcam :
-Pour la première, on donne au ballon un mouvement dans le plan xOz, et xOy pour la boite. Les deux objets ne sont pas synchronisés.
-Les conditions sont identiques dans la seconde vidéo, à la différence que les objets sont presque synchronisés, au moins sur les trois premiers balancements .
-pour la troisième, on donne au ballon un mouvement de rotation d'axe y.
-pour la quatrième, la rotation du ballon est plus rapide, tandis que ses oscillations ont une faible amplitude.
-pour la dernière vidéo, on donne au ballon un mouvement dans le plan xOy, avec une amplitude suffisante pour masquer partiellement la boite à chaque oscillation.

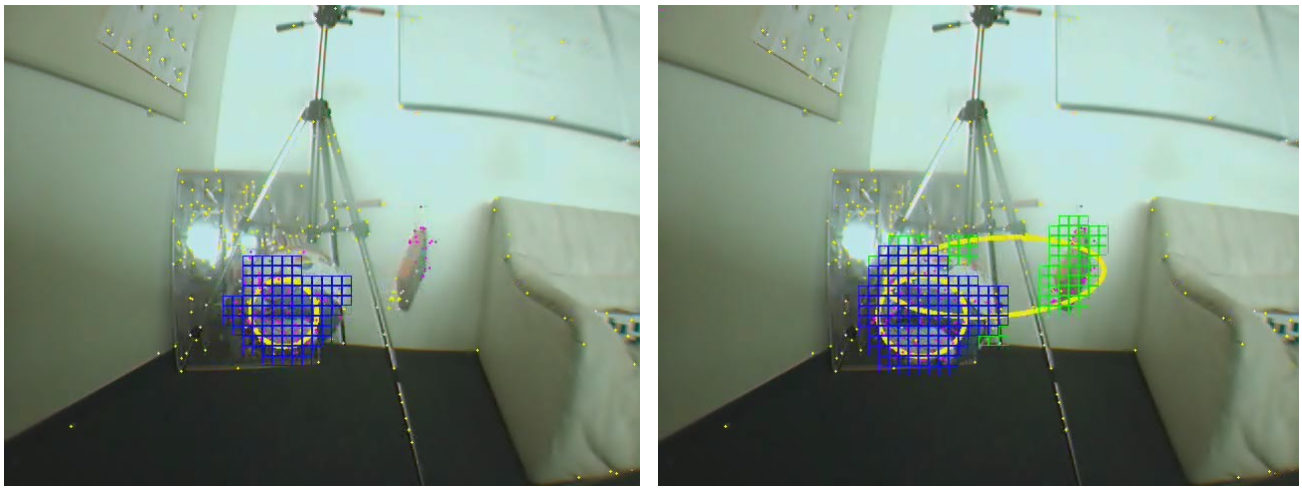
Observations :

pour les vidéos 1 à 3 :

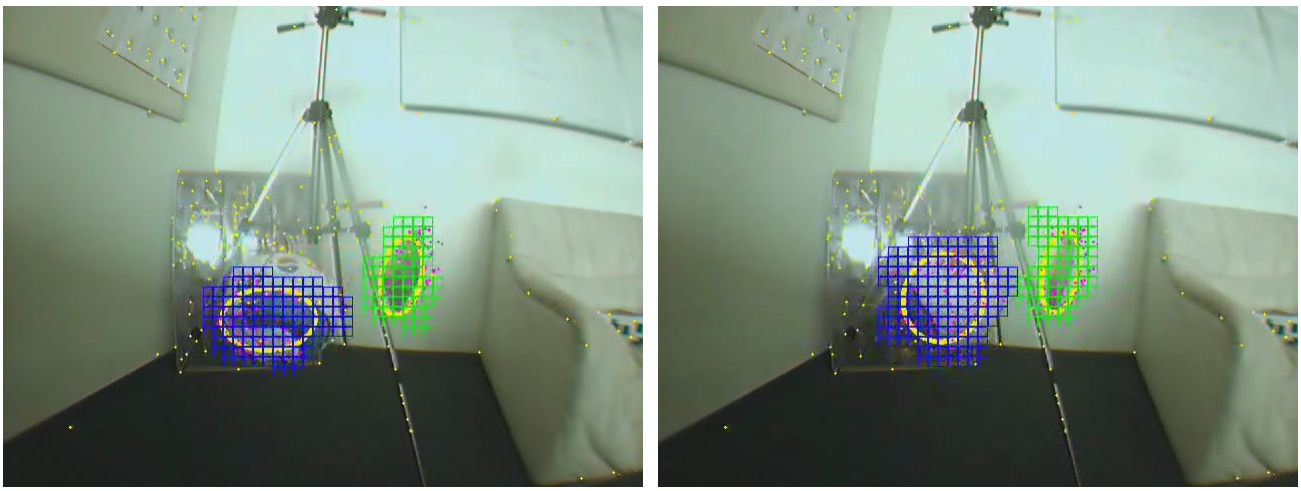
La séquence obtenue est à peu près la même pour les trois premières vidéo. Je décris ici une séquence de la première vidéo (figure 10).
Peu après l'initialisation, les deux objets sont détectés (frame 7). Leur changement de direction ayant eu lieu au même moment, ils sont considérés comme un seul et unique objet.
Mais n'étant pas exactement synchronisé, les points n'agissant pas comme la majorité sont replacés (frame 18). Cette étape dure plusieurs secondes dans le cas de la vidéo 2, les objets étant presque synchronisés. Le ballon comportant plus de points et couvrant une plus grande surface, ceux situés sur la boite sont progressivement replacé sur le ballon (frame 20),
La boite n'étant plus couverte par l'objet, des points de l'arrière plan sont replacés sur celle-ci. Un nouvel objet est défini (frame 28). Les points situés sur la partie non couverte du ballon sont rapidement replacés sur la boite (frame 33).



Frame 0 frame 7



Frame 20 frame 28



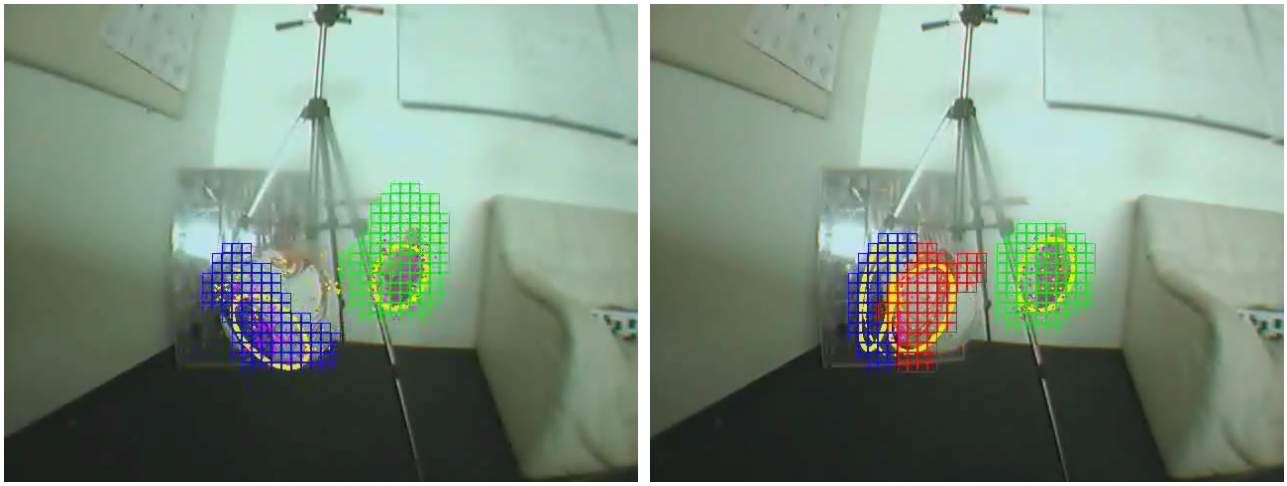
Frame 33 frame 201

Figure 10 : séquence de la première vidéo

Les deux objets sont ensuite correctement suivi. On notera toutefois que lors de la rotation du ballon dans les vidéos 3, les points ont parfois du mal à "accrocher" la surface blanche du ballon, ce

qui explique que l'objet virtuel soit décalé par rapport à la position du ballon (figure 11a). Parfois, l'objet virtuel est suffisamment décalé pour qu'un second objet soit créé sur le bord du ballon (figure 11b).

Les vidéos 4 et 5 montrent les limites de l'algorithme :
pour la vidéo 4, la vitesse de rotation du ballon étant importante, l'objet virtuel ne peut pas explorer la surface et est parfois éjecté de sa surface, les points se plaçant alors sur l'arrière plan. Un second objet virtuel est ensuite détecté et suit le ballon. Si le premier objet virtuel n'a pas été éjecté hors de la surface, on peut alors voir une très forte corrélation entre les deux objets virtuels du ballon. La détection et le suivi de la boite ne pose, en revanche, pas de problème.



a) sur la vidéo 4, l'ellipse est décalée

b) sur la vidéo 5, l'ellipse est tellement décalée qu'un second objet (en rouge) est détectée sur la surface laissée libre.

Figure 11 : problèmes engendrés par les rotation

Sur la vidéo 5, le ballon est parfaitement détecté et suivi. Mais comme il recouvre partiellement la boite (figure 12), certains points se retrouvent sur celle-ci, jusqu'à ce que l'un de ces deux objets change de direction. La présence de points sur la boite empêche la segmentation de celle-ci. La boite n'est jamais segmenté.

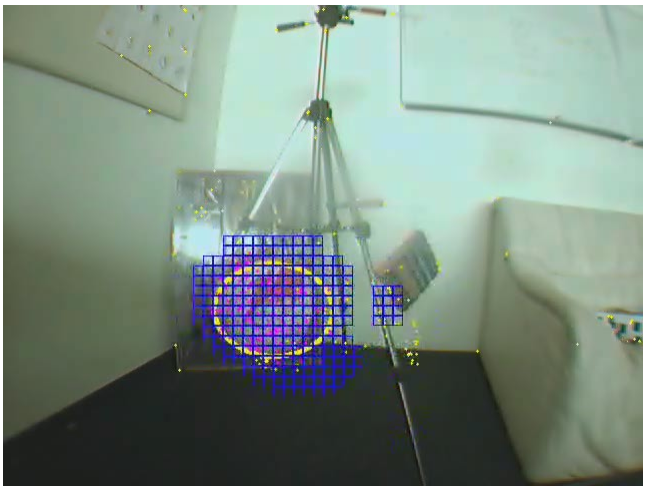
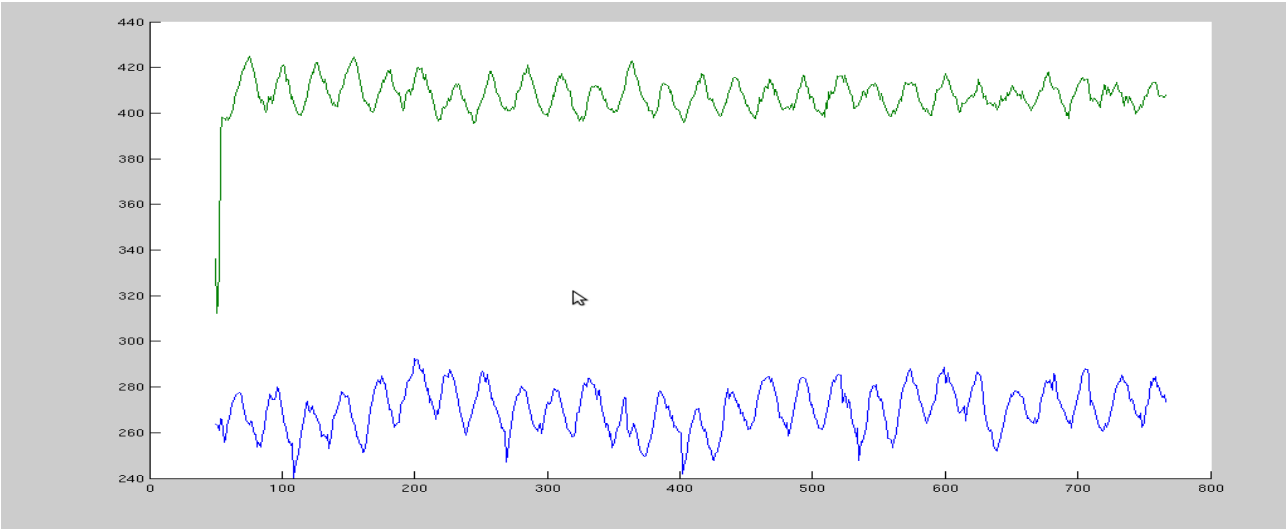


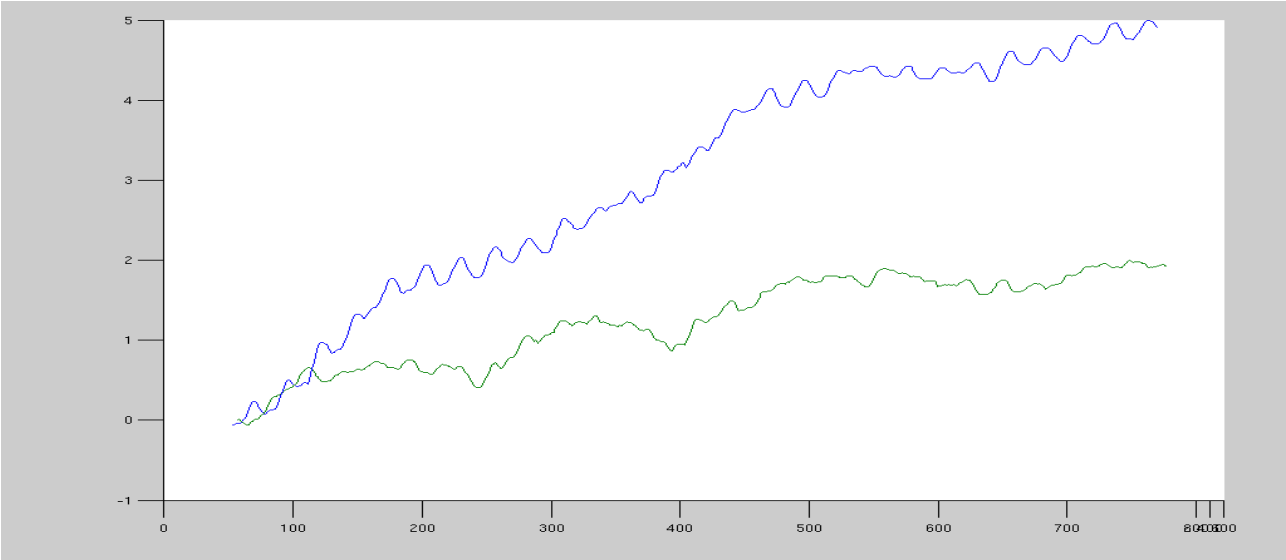
Figure 12 : à chaque oscillation, une partie des points se retrouvent sur la boîte

On trace ensuite sous MatLab les valeurs de X et Z (figure 13) pour chaque objet (les variations sur l'axe Y étant trop faibles pour être exploitable). On obtient une sinusoïde sur l'axe X. Les variations étant dues au problèmes "d'accrochage" des surfaces blanches.

La valeur de Z étant obtenue en intégrant Vz, il est normal que la valeur moyenne soit différente de la position neutre. On reconnaît néanmoins la sinusoïde. Les variations de la courbe du second objet sont dues à la rotation de la boite, bien plus large qu'épaisse.



a) position sur l'axe X



b) position sur l'axe Z. On identifie facilement la rotation de la boite

Figure 13 : affichage des positions sur X et Z du ballon (bleu) et de la boîte (vert)

Malgré ces défauts, l'algorithme montre qu'il est parfaitement capable de segmenter et suivre plusieurs objets à la fois, tout en fournissant des données suffisamment précises sur les déplacements pour pouvoir être identifiés.

1-2 expérience 2, suivi de plusieurs objets

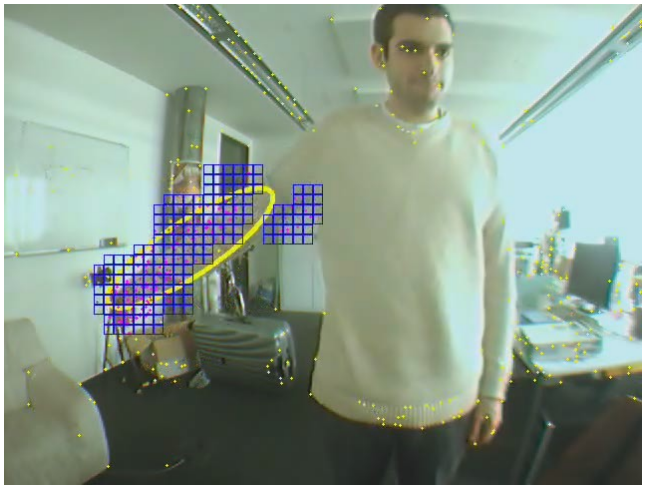
Cette expériences a pour but de montrer la segmentation et le suivi d'éléments liées et de caractéristiques visuelles (couleur, textures...) identiques.

Conditions : Je me tiens au centre de la pièce, immobile, portant un pull de couleur unie. Je bouge alternativement mon bras entier et mon avant-bras seul. On enregistre trois séquences à peu près identiques.

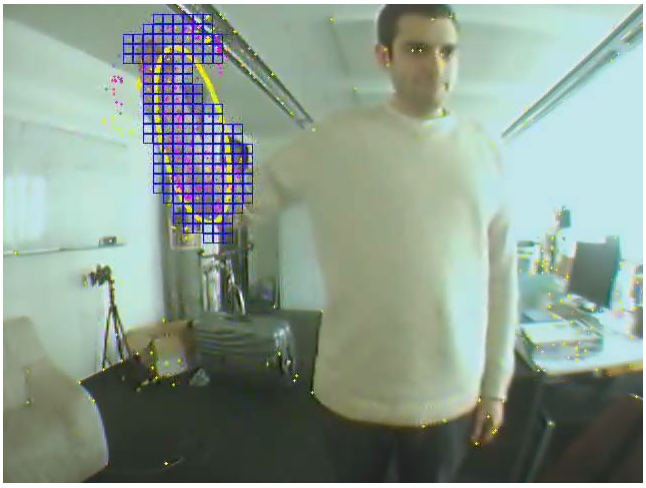
Observations : compte tenu de la part du hasard dans la gestion des points, l'apparition et la répartition des objets peut varier, bien que le suivi des objets soit toujours efficace. Voici les observations des deux cas les plus fréquents (figures 14 et 15):

cas 1 : après l'initialisation, mon bras est détecté (frame 16). La taille de l'objet virtuel augmente pour le couvrir entièrement. Seule mon épaule, pas assez rapide, est considérée comme immobile. Puis je bouge mon avant-bras seul. Mon bras, immobile et ne comportant qu'une minorité de points, est éliminé de l'objet (frame 100). Ce dernier couvre alors uniquement mon avant-bras, et continue à le suivre (frame 212) jusqu'à la fin de la vidéo.

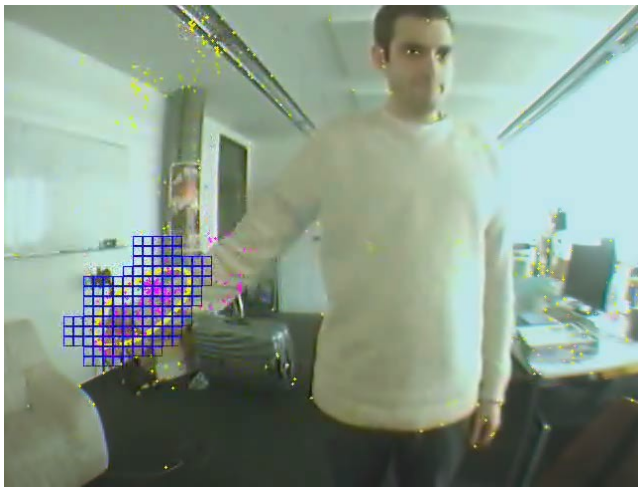
Frame 5



frame 16



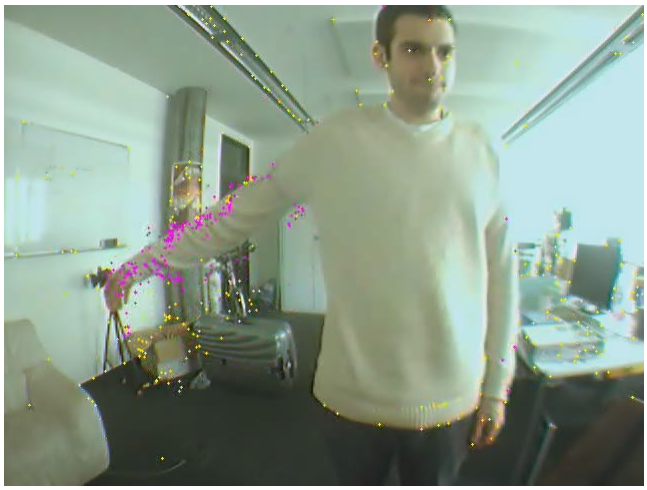
frame 100



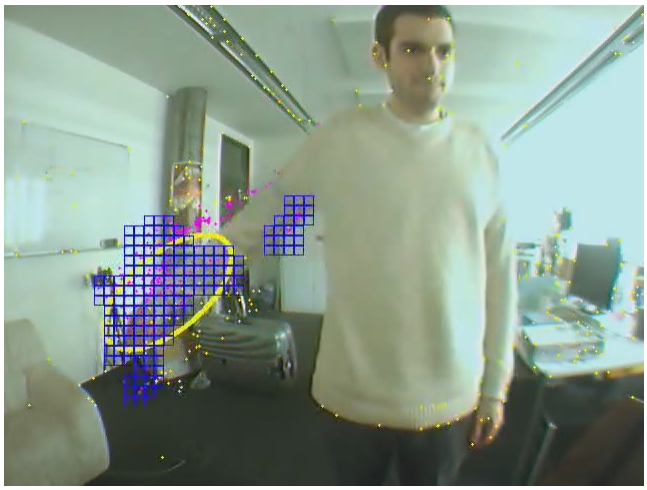
frame 212

Figure 14 : séquence 1 de la seconde expérience

Cas 2 : Comme pour le cas 1, mon bras est détecté (frame 26). Mais cette fois, la surface de l'objet est plus importante et couvre mon épaule et une partie de mon corps, le tissu bougeant en effet avec mon bras. Lorsque je bouge mon avant bras, c'est celui-ci qui est exclu de l'objet (frame 149), celui-ci étant immobile. La différence avec le premier cas est qu'un nombre suffisant de points ont été replacés sur mon épaule, qui, lorsque je ne bouge plus que mon avant-bras, deviennent immobile, permettant ainsi d'obtenir une majorité de points immobiles. Mon avant-bras, qui continue de bouger, est détecté et capturé par un autre objet (frame 172). Les deux objets suivent alors les deux parties de mon bras, sans modifier leur position, jusqu'à la fin de la vidéo et ce, malgré le fait qu'il n'y ait pas de différence visible (même texture et couleur).

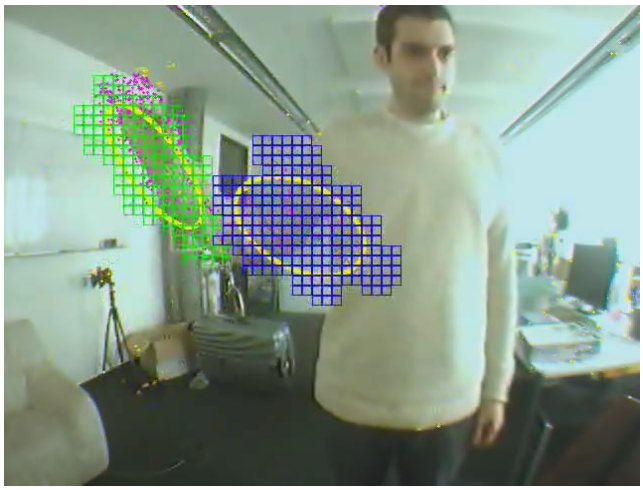


Frame 18

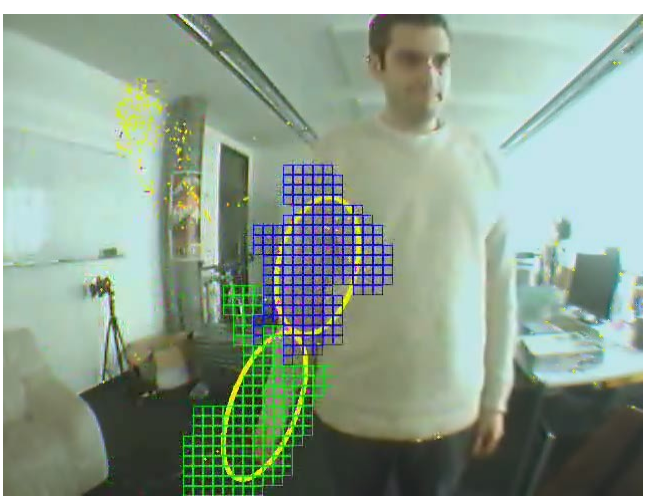


frame 26

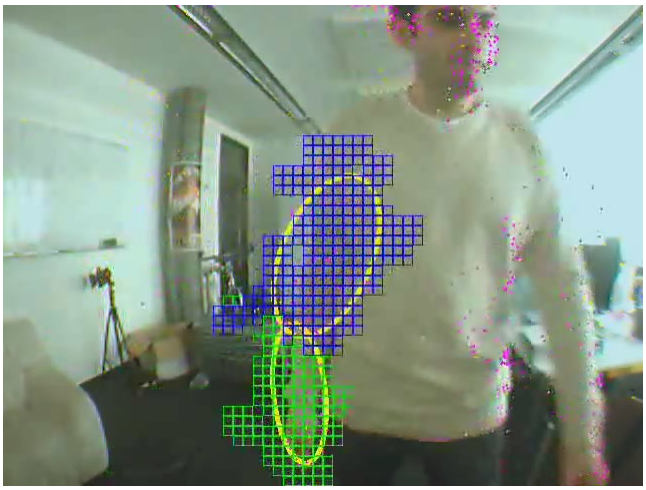
Frame 149



frame 172



Frame 236



frame 255

Figure 15 : séquence 2 de la seconde expérience

La trajectoire des deux objets dans le plan xOy est tracé sous MatLab (figure 16), on reconnaît les mouvement de mon bras après que les deux objets ait été créé.

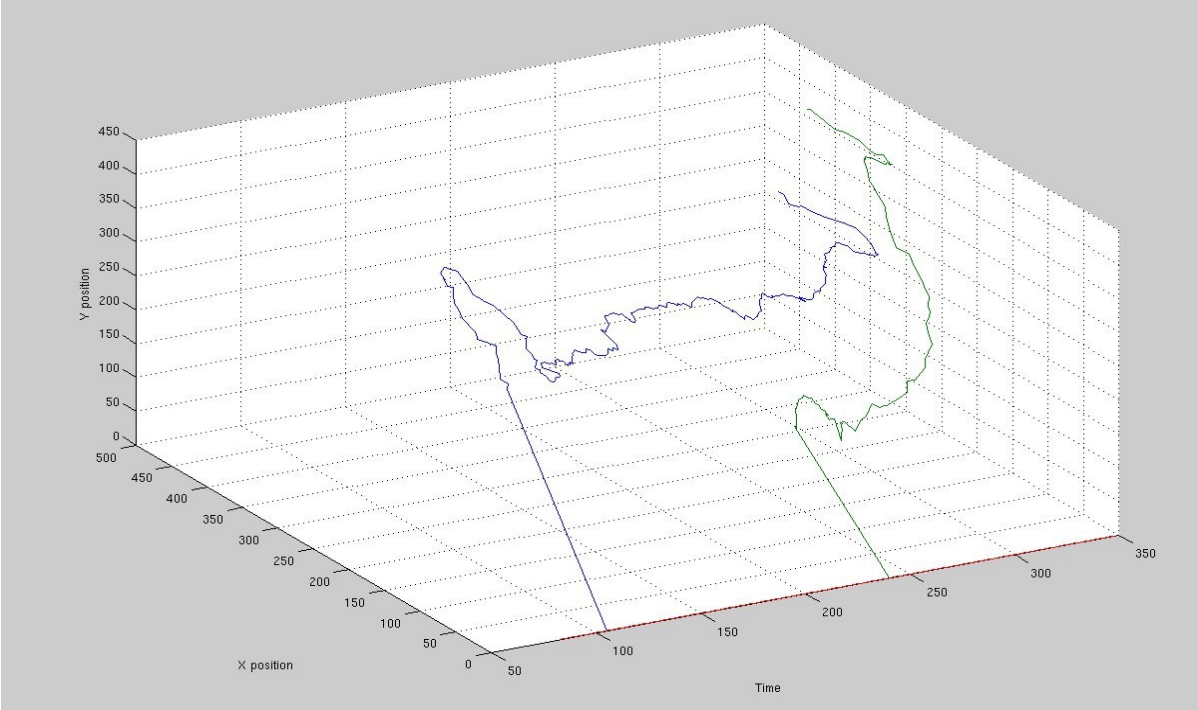


Figure 16 : trajectoires dans le plan xOy.
(L'axe des Y est inversé)

Cette expérience montre que l'algorithme est parfaitement capable de segmenter et suivre des éléments indifféremment des paramètres visuels, et de suivre leurs objets respectifs même lorsqu'ils se déplacent simultanément.

2 Expériences avec ECCERobot

2-1validation du concept

Ces tests ont permis de valider le fonctionnement de l'algorithme sur le robot. Ils ont en outre permis de définir le réglage optimal de la caméra, avec le logiciel Coriander (la caméra étant trop sensible à la lumière naturelle), mais aussi mis en évidence des problèmes inhérents à la structure souple du robot : la tête, et donc la caméra, de celui-ci se met à osciller lorsque le bras accélère ou décélère brutalement. La totalité de l'image est alors considérée comme un objet.

Pour remédier à ce problème, j'ai du implémenter un système pour éviter que l'ensemble de l'image ne soit détectée comme un objet :
lorsqu'un objet se déplace, des points sont replacés sur sa surface. Il en résulte une forte densité sur la surface couverte par ces points. Par contre, quand la caméra se déplace, la totalité des points changeront d'état. Mais la surface couvrira la totalité de l'image, d'où une densité moyenne faible. On ajoute donc une condition sur la densité pour déterminer si on créé un nouvel objet.

Pour ce test, le caméra est dirigée vers le bas, le robot regarde en direction de son bras gauche. On envois au bras une séquence de commandes permettant de lever, puis de baisser l'avant bras (figure 17). On répète cette séquence après un délai défini au hasard. On enregistre à la fois les signaux envoyés et ceux transmis par le module vision.

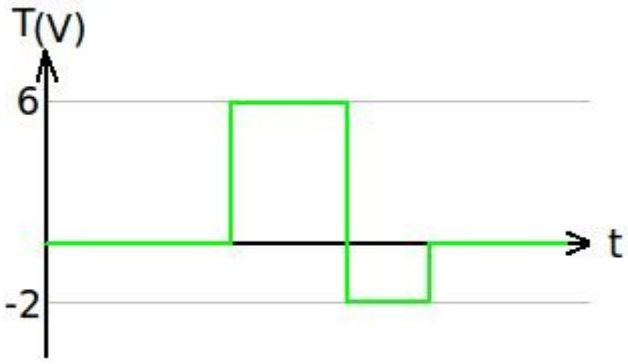


Figure 17 : le signal envoyé au bras
L'asymétrie permet de compenser le poids de l'avant bras.

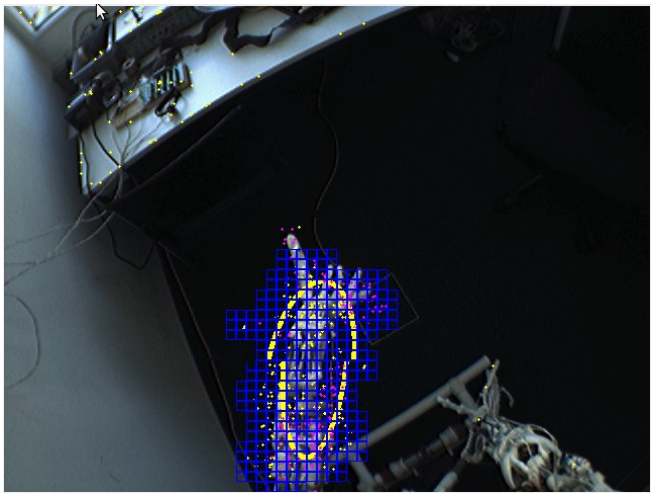
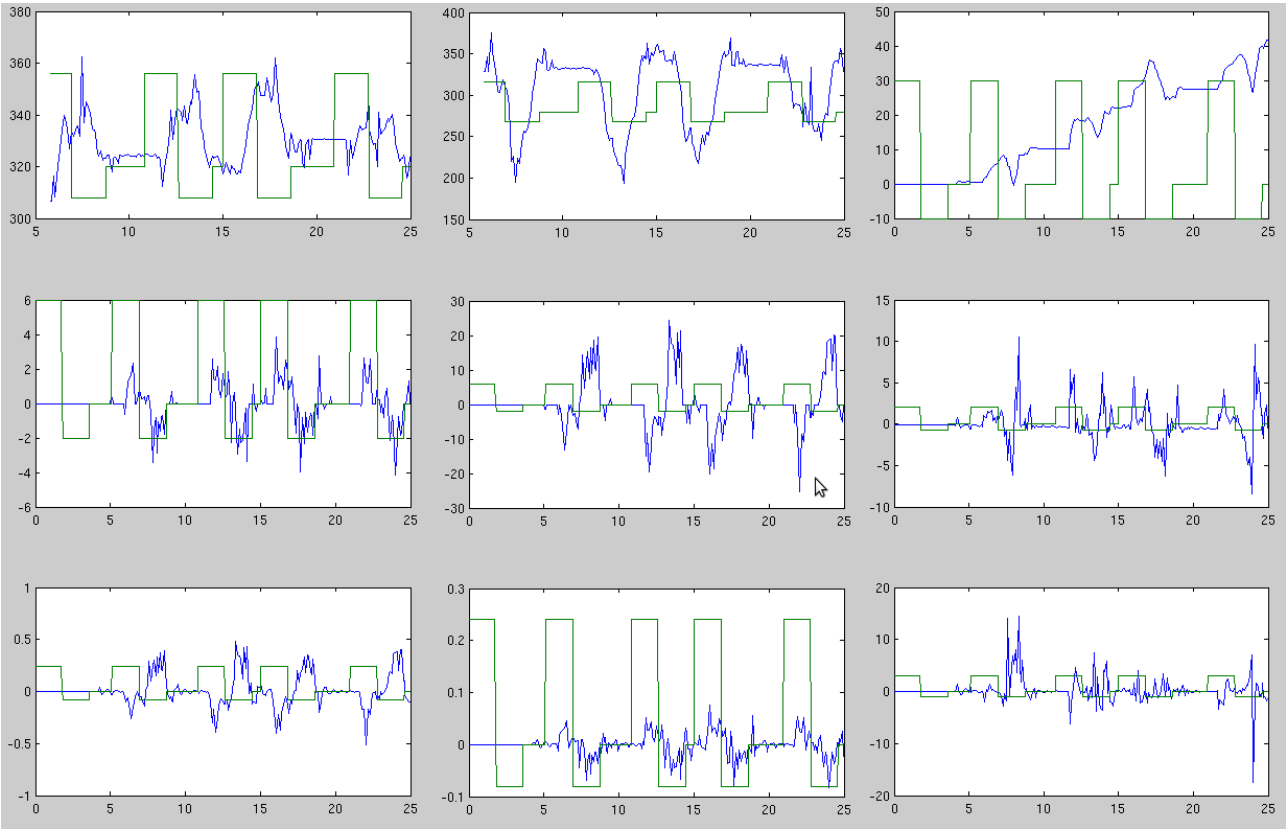


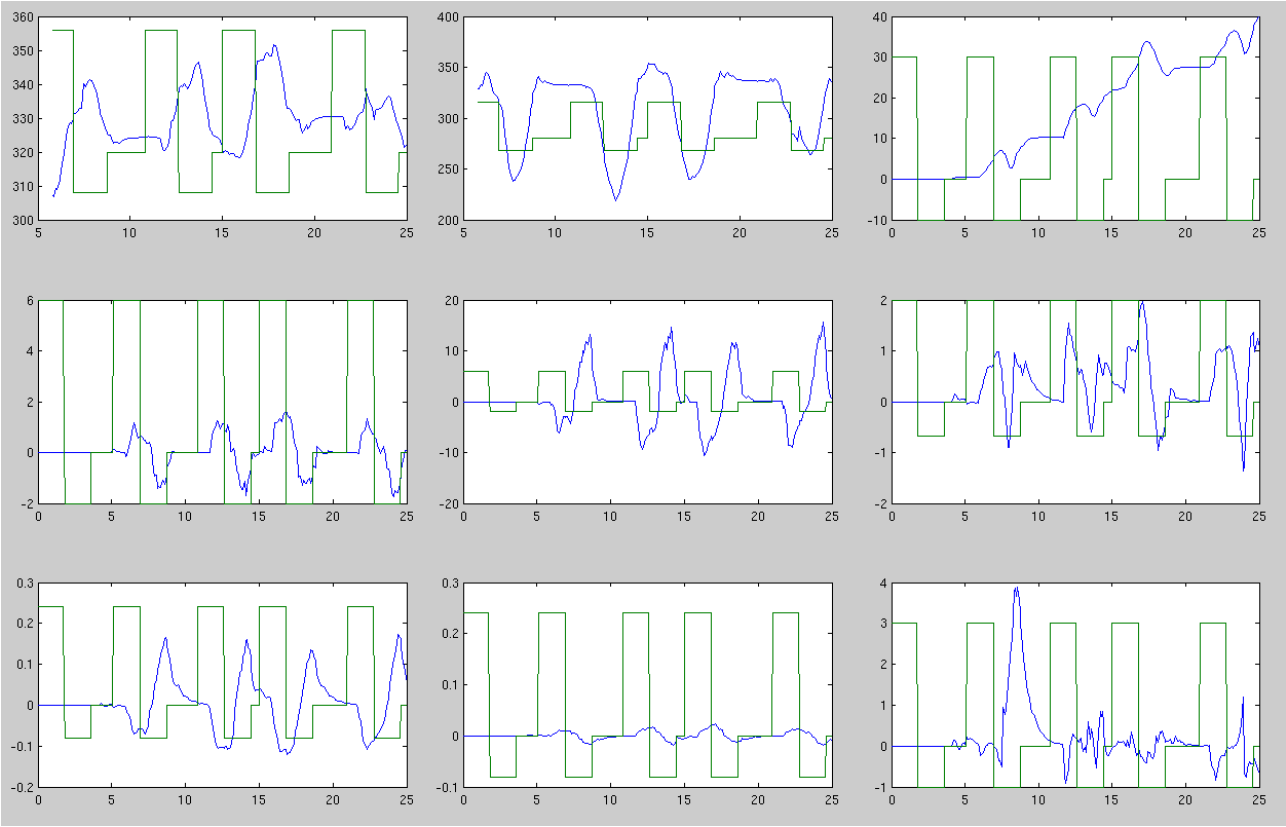
Figure 18 : le robot voit son propre bras bouger (la capture d'image s'est produite pendant un changement de frame, d'où la discontinuité)

On compare ensuite les signaux (figure 19): on remarque dans un premier temps que les signaux issus du module vision présentent une composante haute fréquence lors des changements de direction du bras. Ces perturbations sont dues aux oscillations de la tête. On utilise un filtre passe-bas pour les supprimer.

L'échelle des signaux de commande a été changée pour pouvoir être plus facilement comparé aux signaux du module vision. On utilisera cependant la même échelle pour les positions, vitesses linéaires et angulaires des axes X et Y afin de pouvoir comparer l'amplitude des mouvements sur ces deux axes.



a) les signaux issues du module vision



b) signaux filtrés

Figure 19 : les résultats obtenus lors des tests. Les colonnes représentent les axes X, Y et Z, Les lignes la position, la vitesse linéaire et la vitesse angulaire.

On s'aperçoit que les signaux de sortie suivent les signaux de commande (excepté pour Vrz, dont l'amplitude est faible), chaque courbe reproduit, à chaque envois de la commande, le même signal, indépendamment du délai entre deux commandes.

Ceci montre qu'il y a une forte corrélation entre les signaux de commande et les signaux de sortie, l'objet (le bras) peut donc être considéré comme une partie du robot.

L'algorithme montre donc qu'il est suffisamment efficace pour détecter et suivre les objets mobile, mais surtout il fournit des informations fiables et exploitables sur les déplacements de celui-ci.

2-2 segmentation des éléments du corps du robot

L'algorithme va enfin être utilisé dans les conditions pour lequel il a été conçu.

Pour cette expérience, on reprend la configuration des tests. Mais cette fois, les deux bras vont bouger aléatoirement en alternance. On ajoute dans le champs de vision un pendule constitué du ballon et de son support, utilisé lors des tests de l'algorithme. Ce pendule représente un objet ne faisant pas partie du corps du robot. Divers objets ont été disposé sur le sol afin d'éviter un fond de couleur uniforme (figure 20).

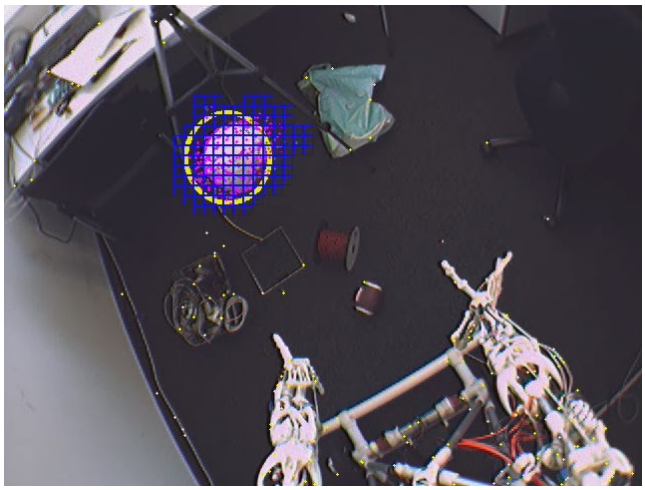
Figure 20 : la scène de l'expérience

Les séquences obtenues diffèrent d'un test à l'autre. L'ordre de détection des objets, notamment, peut varier en fonction de la séquence du mouvement des bras. Cependant, l'algorithme fonctionne quelle que soit la séquence. On notera toutefois que dans certains cas, les mouvements de la tête provoque la détection d'un nouvel objet dans l'arrière plan.

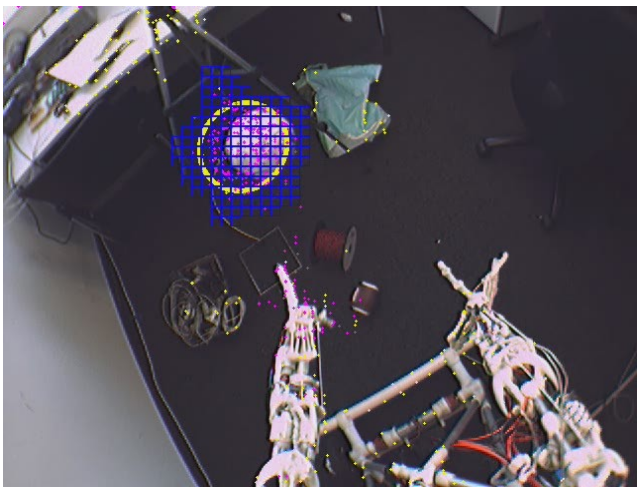
La séquence présenté est celle du dernier test effectué (figure 21):

On initialise l'algorithme (frame 0). Quelques secondes plus tard, le ballon, qui oscille, et donc change fréquemment de direction, est détecté et segmenté (frame 46). Peu de temps après, les bras commencent à bouger à tour de rôle (frame 85). Le système de replacement des points fait que les points replacés sur un bras lorsque celui-ci bouge seront à nouveau déplacés sur l'autre bras lorsque ce dernier bougera à son tour. Il est dans ces conditions difficiles d'obtenir un nombre suffisant de points mobiles pour segmenter un objet. Cependant, et malgré un accrochage du premier objet sur le bras (frame 208), le bras gauche finira par être détecté et segmenté à son tour (frame 266). Puis le bras droit sera détecté (frame 501). On notera qu'à cause des oscillations de la tête, quelques points situés sur le coin supérieure gauche de l'image ont été considérés comme faisant partie de l'objet, ce qui explique la forme allongé de l'ellipse. Ces points seront éliminés lors du prochain mouvement du bras (frame 501). Les trois objets virtuels continuent ensuite leurs suivi jusqu'à la fin de l'envoi des signaux (frames 996).

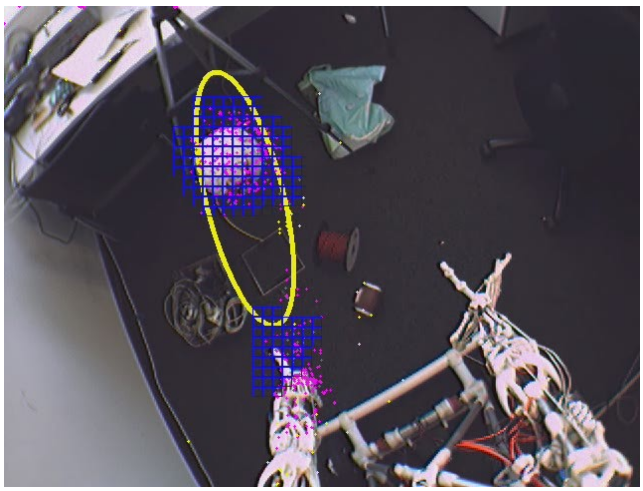
Frame 0



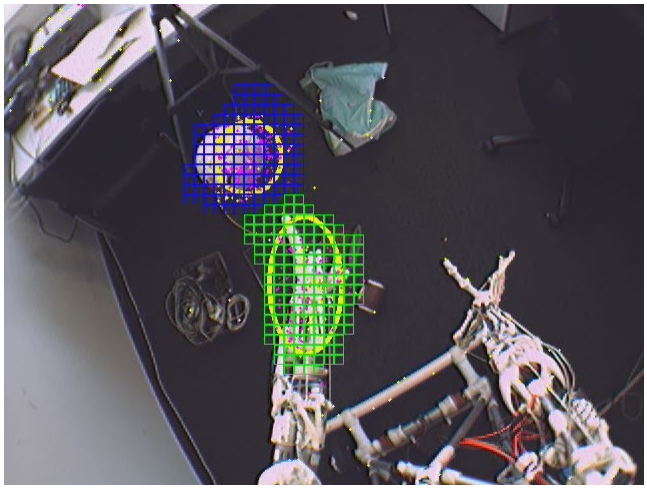
frame 46



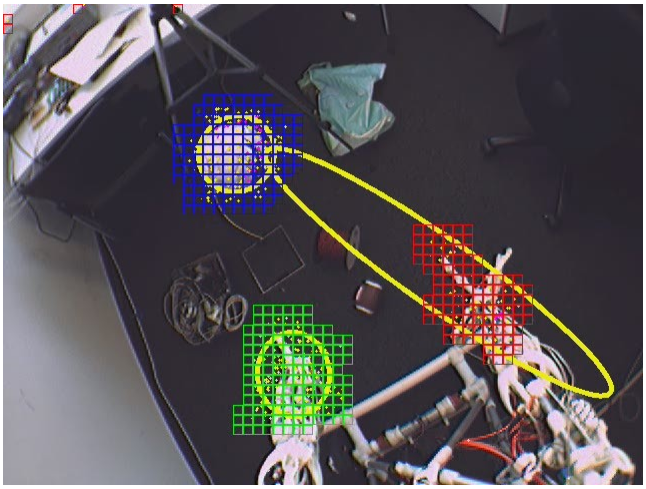
frame 85



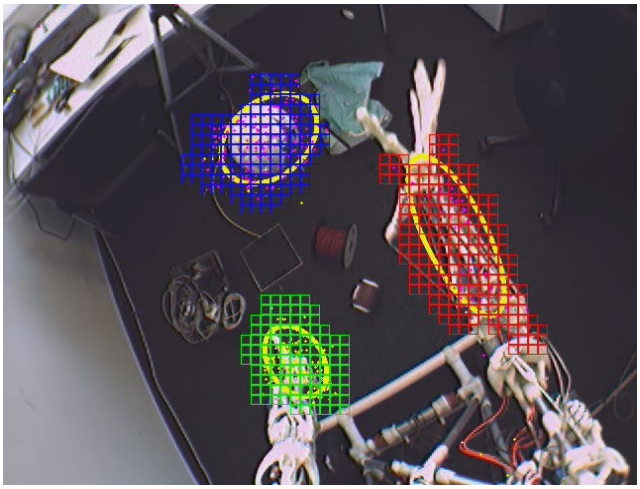
frame 208



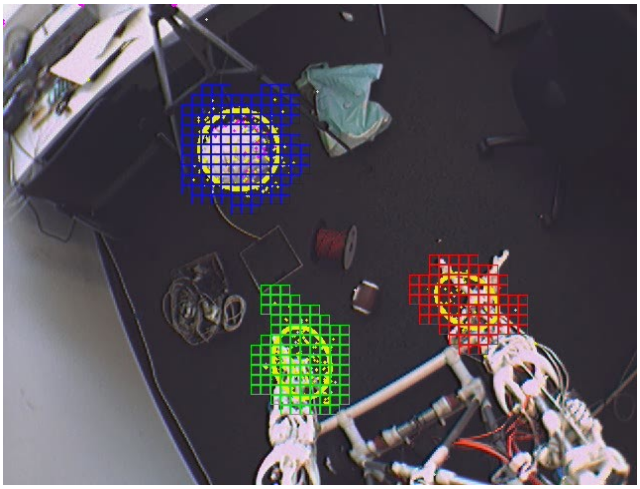
frame 266



frame 419



frame 501

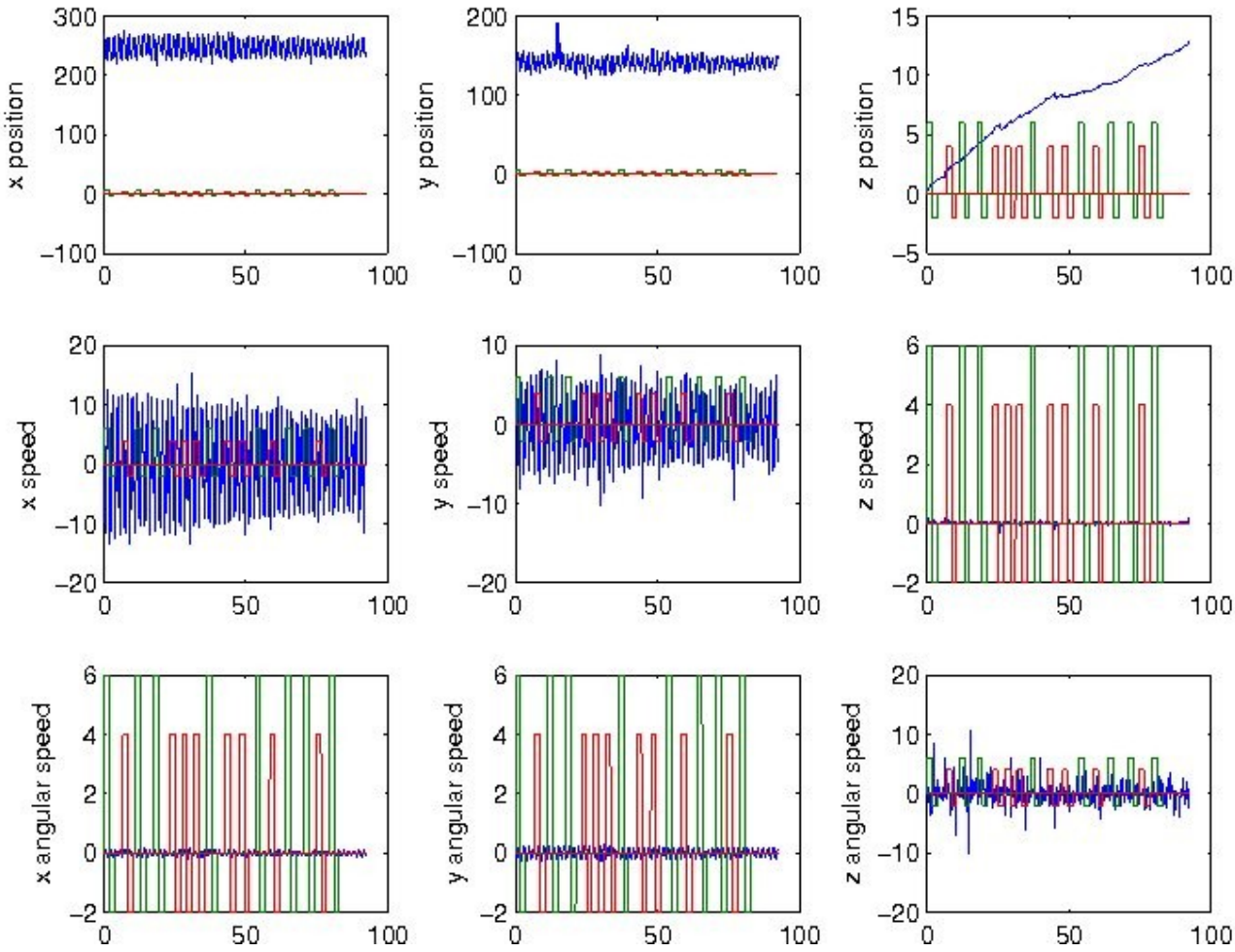


frame 996

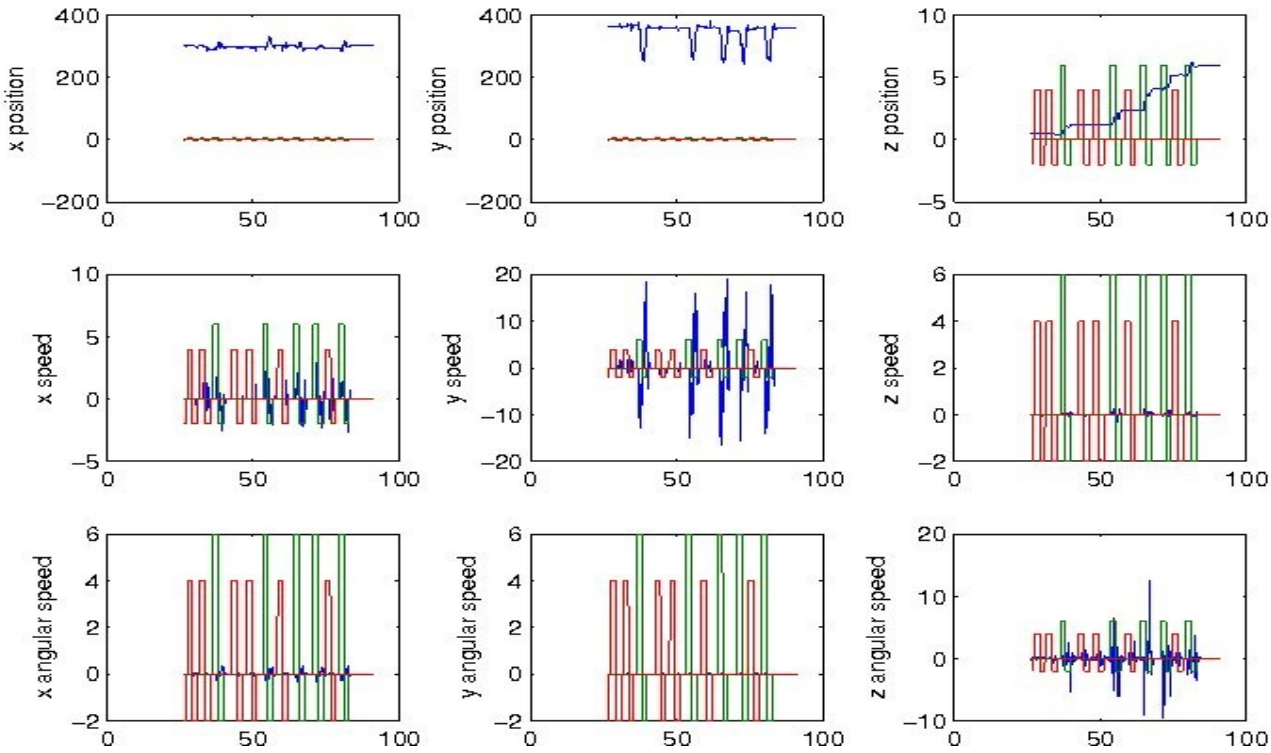
Figure 21 : séquence de l'expérience

Pendant l'expérience, on enregistre les signaux issue du module de commande et du module vision afin de les comparer.

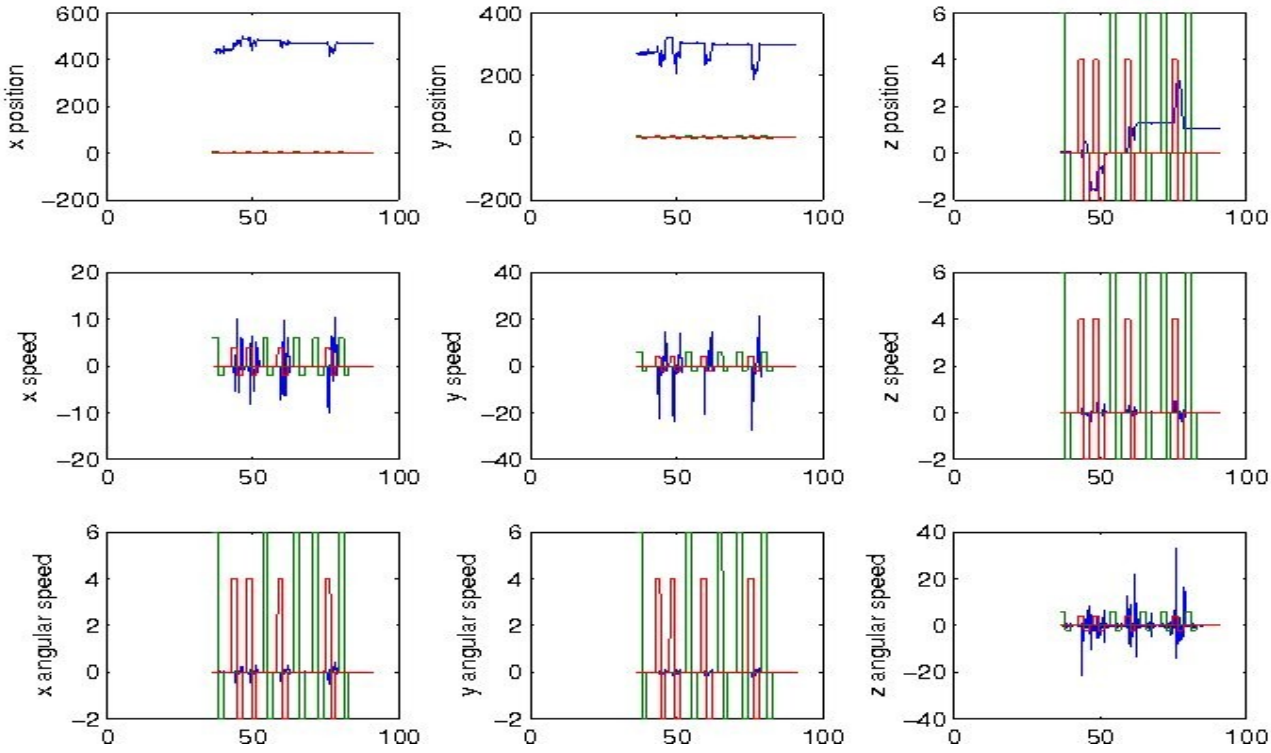
Ces enregistrements sont ensuite affichés sous MatLab (figure 22). La **figure** affiche les différentes courbes des trois objets comparés aux signaux des moteurs. L'association entre les signaux d'entrée et de sortie ne présentent aucune ambiguïté : on reconnaît facilement à quel bras correspond chaque objet. On notera toutefois la présence sur les courbes de signaux de faible amplitude que l'on peut associé au signal d'entré opposé. Ils sont dues aux oscillations de la tête provoqués par le déplacement des bras : lorsque l'un d'entre eux bouge, la caméra verra l'autre osciller. Ces signaux parasites seront cependant supprimés par le filtre passe-bas.



a) Les courbes du premier objet : pas de corrélation



b) courbes de l'objet 2, forte corrélation avec le signal 1 (en vert)



c) courbes de l'objet 3, forte corrélation avec le signal 2 (en rouge)

Figure 22 : courbes obtenues pour chaque objet

Dans un souci de lisibilité, les courbes affichés par la suite seront celles des signaux filtrés.

Corrélation des signaux :

On teste ensuite la corrélation des signaux obtenus, afin d'établir le lien entre les commandes moteur et les objets. On utilise la fonction MatLab *xcorr*, fournissant le coefficient de corrélation maximum entre f1(t) et f2(t+dt) et le retard dt correspondant. La figure montre la courbe coef(dt) obtenue pour le signal dY de l'objet 2, le signal ayant obtenu le meilleur coefficient de corrélation. La valeur maximale en valeur absolue est atteinte pour un retard de 17 mesures.

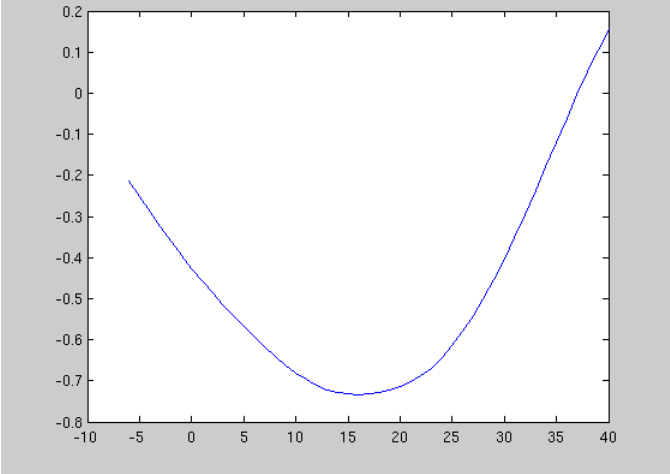


Figure 23 : valeur du coefficient de corrélation du signal dY de l'objet 2 en fonction du retard.

L'affichage montre les valeurs des coefficients (en valeur absolue) pour chaque signal. Ces signaux seront affichés dans l'ordre suivant : position, position angulaire, taille, vitesse, vitesse de rotation, chacune dans ses trois composantes x, y et z.

Les résultats obtenus montrent clairement la relation entre les signaux moteurs et le mouvement des objets :

Pour le premier objet (figure 24), la corrélation est très faible, signe que l'objet n'est lié à aucune commande moteur. On notera toutefois que certaines valeurs ont un coefficient relativement élevé, notamment celles concernant les vitesses et positions sur l'axe Z, ainsi que la taille : l'objet oscillant dans le plan xOy, les variations de la position et des vitesses sur Z et la taille ne varient que très peu. Les perturbations engendrées par les oscillations de la tête (quand les bras bougent) modifient donc les signaux d'une manière non négligeable, ce qui explique ces coefficients relativement élevés.

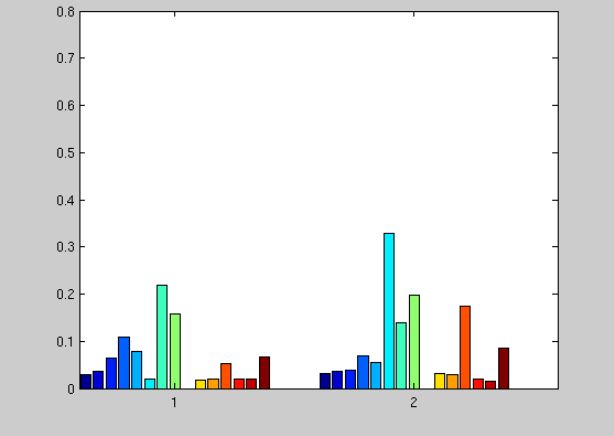


Figure 24: coefficients de corrélation de l'objet 1

Dans le cas du second objet (figure 25), la relation avec la commande 1 (bras gauche) est clairement établie : les coefficients des signaux issues des vitesses linéaires et angulaire sur les axes X et Y dépassent le seuil de 0.5, ceux de la position sur Y et la vitesse sur l'axe Z sont proches de 0.4 . Si ces coefficients sont relativement faibles, c'est que, bien que ces signaux soit synchronisés avec les mouvements du bras, ils sont différents des signaux moteurs.

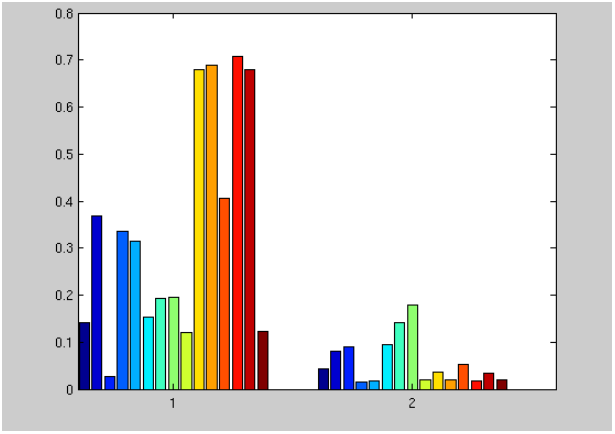


Figure 25 : corrélation de l'objet 2

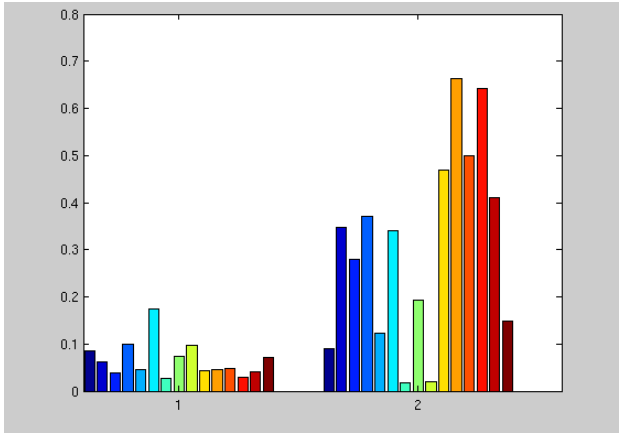


Figure 26 : corrélation de l'objet 3

Le troisième objet (figure 26) est, lui, lié à la commande moteur 2 : comme pour le deuxième objet, ses coefficients pour les vitesses et vitesses de rotation sont élevés pour cette commande. Tout comme pour celui-ci, les positions et positions angulaires restent faibles (autours de 0.3).

La figure 27 montre un comparatif des trois objets : l'ordre des signaux est le même, mais dans un souci de lisibilité, les composantes sur les axes sont répartis sur l'axe X du graphe.

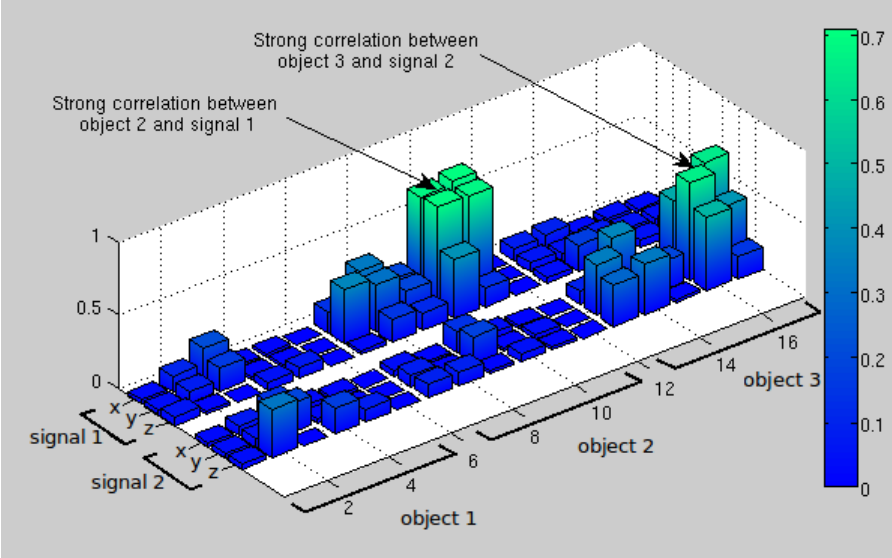


Figure 27 : comparatif des signaux de corrélation des trois objets.

L'algorithme montre ainsi qu'il est capable de fournir des informations fiables sur les objets en mouvements, permettant de définir si ils font partis du corps du robot, et si oui, quelle est l'influence d'une commande sur ces objets.

2-3 utilisation d'un outil

Dans cette expérience, on va fixer des objets, représentant un outil, au bras du robot. Le but est d'étudier le comportement de l'algorithme lorsque la forme d'un objet change. Le premier est une tige courte à laquelle on a fixé une balle en mousse à l'une des extrémités. Le bras est ainsi "allongé" de 20cm environ. Pour le second, on allonge la tige, pour une longueur de 50cm. Les éléments sont fixés au bras à l'aide de bandes Velcro, permettant de les fixer et les retirer rapidement. On conserve le pendule de l'expérience précédente pour représenter un objet de l'arrière plan.

L'expérience se déroule de la manière suivante : on envois la commande au bras droit douze fois. Puis on fixe le premier élément au bras. On relance un cycle de douze commandes. On allonge ensuite le premier élément, avant de lancer une dernière séquence de douze commandes.

Afin de pouvoir changer les éléments sans perturber le système par les manipulations, j'ai implémenté un mode "pause" pendant lequel aucun nouvel objet ne peut être crée.

Observations :

Le pendule est, comme dans l'expérience précédente, rapidement détecté et segmenté (frame 24). Le bras est détecté après deux mouvements, entaché de nombreux points parasites sur l'arrière plan (frame 126) qui seront éliminés au mouvement suivant (155) . On notera que malgré la présence du détecteur de mouvement, deux objets sont crées, regroupant différents points de l'arrière plan. Le quatrième (bleu clair) verra ses points se regrouper sur le bord supérieur de l'image, l'autre, regroupant des points de deux endroits opposés sur l'image, et qui se déplacent en même temps

(quand la caméra bouge) mettra plus de temps à regrouper ses points.

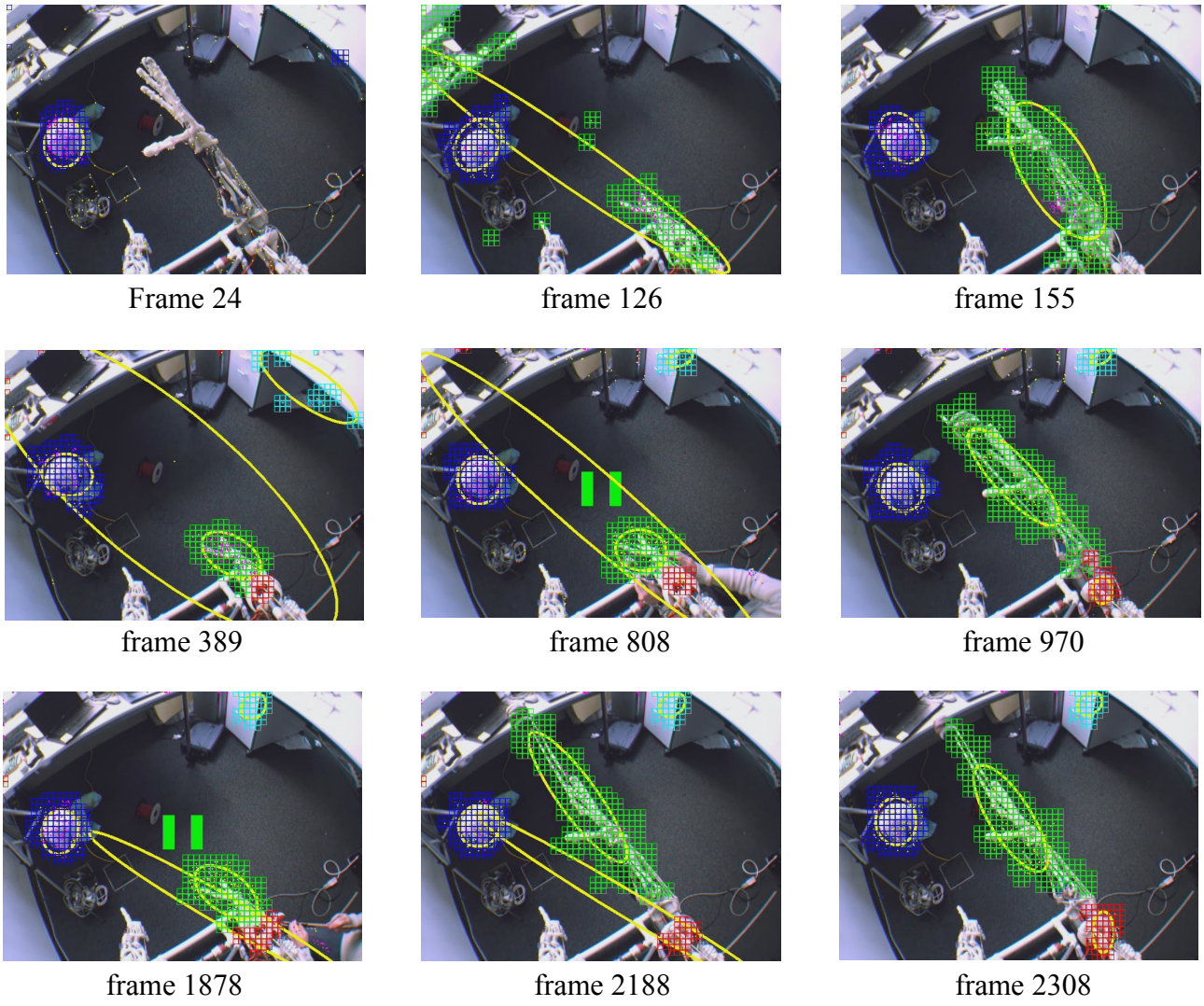


Figure 28 : séquence de l'expérience sur l'utilisation d'un outil

De la frame 764 à 894, on passe en mode "pause" pour placer le premier élément sur le bras du robot, puis on relance une séquence de douze commandes. Au mouvement suivant, l'objet virtuel suivant le bras recouvre la balle, montrant que celle-ci est bien considérée comme une partie du bras. On place ensuite la rallonge de l'élément (frame 1598 à 2019). Là encore, malgré la longueur, l'élément est bien considéré comme une partie du bras (frame 2188). On notera que le troisième objet (rouge) finira par suivre le bras (frame 2308).

L'image de l'objet est donc modifiée par les changements de forme de celui-ci. Un robot utilisant cet algorithme peut donc voir l'image de son propre corps si celui-ci utilise un outil, et l'utiliser comme une extension de son propre corps.

CONCLUSION

L'algorithme de segmentation et de suivi basé sur le mouvement a donc tenu ses objectifs, mais va aussi plus loin: il est non seulement capable de segmenter les objets faisant partis du corps du robot et de les suivre, mais est aussi capable de fournir des informations relativement fiable sur les déplacements dans l'espace de ceux-ci, ce qui permettra de reconstituer, dans un avenir proche, un modèle réaliste du robot. Le dr Hugo Gravato Marquès compte en effet adapter le code pour obtenir une modélisation du corps entier de ECCERobot.

L'algorithme, bien que répondant aux impératif du sujet du stage, est cependant loin d'être terminé et peut encore être amélioré : la gestion des occlusions peut être implémenté, par exemple en fixant les points sur l'objet virtuel lorsque deux objets sont proches, ou que l'un d'eux s'approche du bord de l'image. Le système peu être fiabilisé si on ajoute une mémoire aux objets virtuels, stockant des informations sur l'objet suivi (couleur, texture..) permettant de retrouver l'objet suivi si il venait à le perdre. L'algorithme peut être combiné avec l'algorithme de construction de chaine cinématique, ce qui fiabiliserait le suivi ainsi que les données fournies (les informations fournies par un objet peuvent être exploitées par les autres objets de sa chaine). Pour aller plus loin, il doit être possible d'utiliser le mouvement des points (comme expliqué en III-1.3) pour reconstituer l'objet en trois dimensions.

Concernant le stage lui-même, ce fut une expérience des plus intéressantes : j'ai en effet pu participer à toutes les étapes d'une expérimentation, de l'idée jusqu'à l'analyse des résultats. Les chercheurs avec qui j'ai travaillé m'ont transmis leur expérience et leur savoir-faire, qui devraient m'être utile pour ma future carrière.

Références liées à la vision par ordinateur :

[1] *Object Tracking: A Survey* , Alper Yilmaz , Omar Javed and Mubarak Shah, in *ACM Computing Surveys (CSUR)*, Volume 38 , Issue 4, Article No.: 13, 2006.

[2] *Density-Based Clustering for Real-Time Stream Data* , Yixin Chen and Li Tu, in *International Conference on Knowledge Discovery and Data Mining,* SESSION: Research track papers, Pages: 133 – 142, San Jose, California, USA , 2007.

[3] *Automatic Kinematic Chain Building from Feature Trajectories of Articulated Objects* , Jingyu Yan and Marc Pollefeys, in *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Volume 1, Pages: 712 – 719, 2006.

[4] Hyperplane Approximation for Template Matching , Frederic Jurie and Michel Dhome, in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Volume 24 , Issue 7, Pages: 996 – 1000, July 2002.

[5] Condensation- conditionnal density propagation for visual tracking , in Michael Isard and Andrew Blake, Int. J. Computer Vision, Volume 29, Issue 1,Pages: 5 – 28, 1998.

[6] *Efficient model-based 3D tracking of deformable objects* , Enrique Munoz , José M. Buenaposada and Luis Baumela, in *Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV'05)*, Volume1, Pages: 877 – 882, 2005.

[7] *Suivi de nuages de points d'intérets par filtrage particulaire : application au suivi de véhicules* , Thierry château and J.T. Lapresté, in *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Volume 24 , Issue 7, Pages: 996 – 1000, July 2002.

références liées à l'étude de l'image du corps :

[8] *Using Probabilistic Reasoning Over Time to Self-Recognize* , Kevin Gold and Brian Scassellati, in *Robotics and Autonomous Systems* , Volume 57, Issue 4, Pages: 384 - 392, April 2009.

[9] *What Can I Control?: The Development of Visual Categories for a Robot's Body and the World that it Influences ,* Charles C. Kemp and Aaron Edsinger , in Proc. Fifth Int. Conf. on Development and Learning (ICDL), 2006 .

[10] *Towards Manipulation-Driven Vision* , Paul M. Fitzpatrick and Giorgio Metta, in IEEE/RSJ Conference on Intelligent Robots and Systems, 2002.

[11] *Body schema in robotics: a review* , Matej Hoffmann, Hugo Gravato Marques, Alejandro, Hernandez Arieta, Hidenobu Sumioka, Max Lungarella, Rolf Pfeifer, in *IEEE Transactions on Autonomous Mental Development*, in press, 2010.

[12] *Sensorymotor coordination in a "baby" robot : learning about objects through grasping* , Lorenzo Natale, Francesco Orabona, Giorgio Metta and Giulio Sandini, in *Progress in brain research*, Volume 164, 2007.