Institut des Sciences et Techniques de l'Ingénieur d'Angers

RAPPORT DE PROJET DE 4 E ANNEE

Option Automatique et Génie Informatique

Projet Cartomatic

Reconnaissance de formes

Projet réalisé par :

Antoine Crinière
Mikaël Came
Christophe Rigaud

Projet encadré par :

Jean-Baptiste Fasquel

2009-2010

Remerciements

Nous remercions M. Jean Baptiste Fasquel pour sa disponibilité, ses indications, son aide et la précieuse documentation qu'il nous a fournies.

Nous remercierons également nos camarades pour l'aide et leurs points de vue qu'ils ont pu nous apporter et pour l'ambiance qui a régné tout au long du projet.

Merci à l'ISTIA et l'ensemble de son équipe pédagogique pour la disponibilité des locaux et matérielle.

Sommaire

REMERCIEMENTS **2**

SOMMAIRE **3**

INTRODUCTION **4**

1. PRESENTATION DU PROJET **5**

1.1. CONTEXTE 5
1.2. CONSIGNES DE PROJET 5
1.3. DIAGRAMMES 6
1.3.1. SADT *6*
1.3.2. GANTT *7*

2. TECHNOLOGIES ET METHODES **8**

2.1. TECHNOLOGIES UTILISEES POUR LA RECONNAISSANCE 8
2.2. TECHNOLOGIE UTILISEE POUR LA GESTION DE LA BASE DE DONNEES 10
2.3. TECHNOLOGIE UTILISEE POUR L'IHM 11
2.4. TECHNOLOGIE UTILISEE POUR LA CAPTURE D'IMAGE 11

3. DEROULEMENT DU PROJET **12**

3.1. AXES DU PROJET 12
3.2. PROTOTYPE DE L'APPLICATION 12
3.3. RESULTATS DU PROTOTYPAGE 15
3.4. DEVELOPPEMENT DU PROJET FINAL 16
3.4.2. L'intégration de OpenSURF *17*
3.4.3. Diagrammes UML du projet *18*
3.4.4. Fonctionnement de l'application *19*
3.5. RESULTATS DE L'APPLICATION 24
3.5.1. — Réglage du seuil de reconnaissance *24*
3.5.2. Rotation de l'objet *25*
3.6. CONCLUSIONS SUR LE PROJET FINAL 26

CONCLUSION **27**

BIBLIOGRAPHIE / WEBOGRAPHIE **28**

Introduction

Dans le cadre de la quatrième année d'école d'ingénieur, l'ISTIA propose de mener à bien un projet, durant le deuxième semestre, en équipes composées de plusieurs étudiants.

Parmi les différents sujets proposés, l'un d'entre eux a retenu plus particulièrement notre attention, le projet Cartomatic.

L'objectif de ce projet est de développer un système de reconnaissance d'objets par ordinateur robuste et fiable.

Ce projet fait appel à plusieurs de nos compétences acquises durant notre formation au sein de l'ISTIA parmi lesquelles:

La simulation et le test d'algorithmes sous SCILAB

Le développement C++ sous UNIX

Vision par ordinateur

La gestion de projet

De plus, nous avons vu en ce sujet, l'opportunité de s'intégrer au sein d'un projet important, de découvrir des technologies et méthodes nouvelles, ainsi que de développer notre sens pratique.

Et n'oublions pas que ce projet a été pour nous
un moyen d'étancher notre soif de
connaissances...

@ Toute l'équipe

1. Présentation du projet

1.1. Contexte

1.1.1. Qu'est-ce que le concours Cartomatic?

Ce projet a pour objectif de réaliser une plateforme multi robots distribuée pour l'exploration et la cartographie de bâtiment. Trois partenaires forment le consortium, le LISA à Angers (Laboratoire d'ingénierie des Systèmes Automatisés), l'équipe Maia du LORIA à Nancy et un partenaire industriel : la société Wany Robotics basée à Montpellier.

Ce projet s'articule autour de quatre grands points:

— La réalisation d'une plateforme multi robots, chaque robot disposant d'une base d'objets reconnaissables commune.

— La localisation de chaque robot dans l'espace.

— L'architecture et la stratégie de déploiement de la flotte de robots.

— La reconnaissance visuelle d'objet et l'apprentissage.

Notre équipe a pour but de développer le quatrième point sur la reconnaissance d'image ainsi que de constituer la base d'objets commune à chaque robot.

1.2. Consignes de projet

Pour ce qui est des consignes du projet, il nous a été demandé d'étudier la méthode SIFT (Scale Invariant Feature Transform) qui permet de caractériser les points importants d'une image (contour, angle, etc.).

Les librairies graphiques OpenCV et GTK seront utilisées pour gérer respectivement l'acquisition vidéo et l'interface graphique (IHM).

L'ensemble du projet a été développé sous un système UNIX.

1.3.Diagrammes

1.3.1. SADT

Voici la description fonctionnelle du programme suivant la méthode SADT (Structured Analysis Design Technique) :

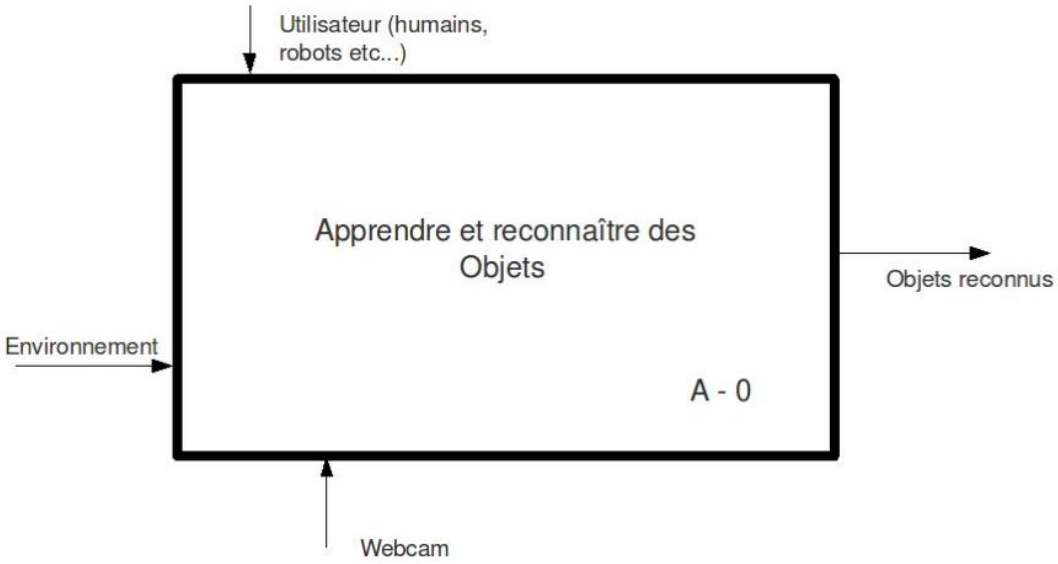


Figure 1 : SADT du projet de niveaux A -0

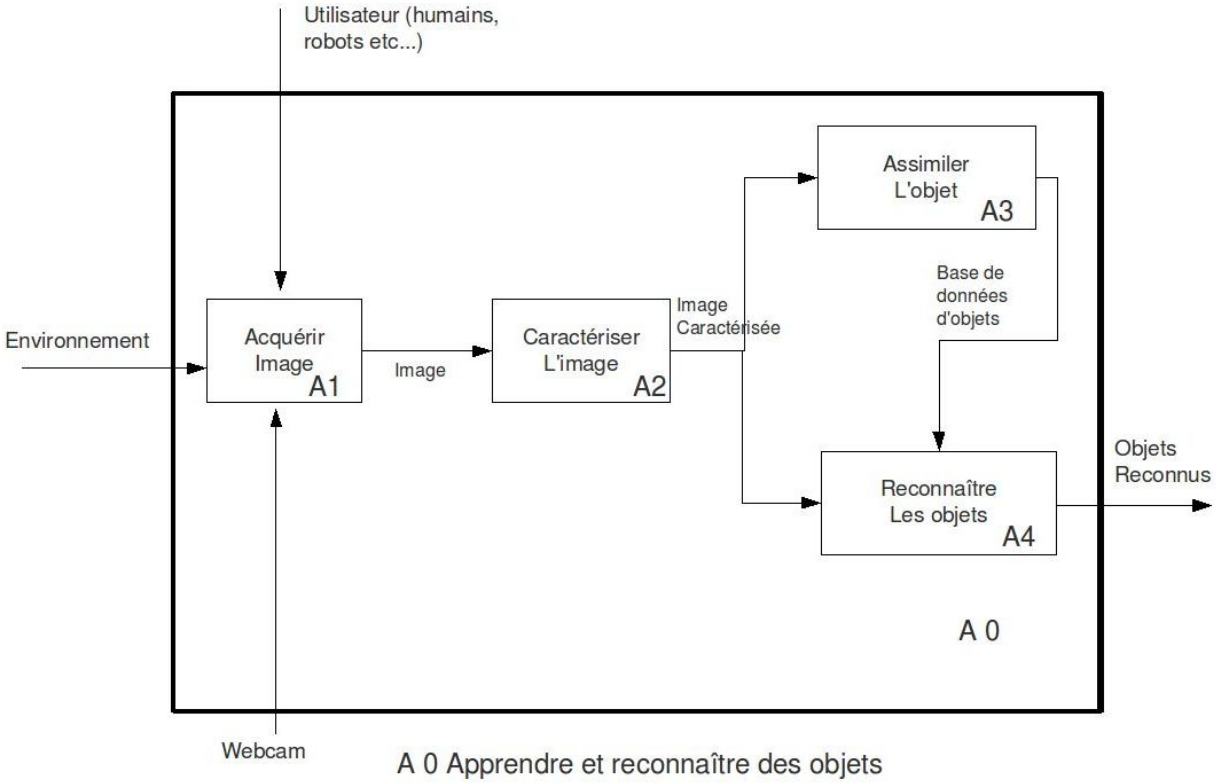


Figure 2 : SADT du projet de niveau A0

1.3.2. GANTT

Voici le déroulement du projet tel que nous l'avions planifié :

N°	Nom de la tâche	Durée	Début	Fin	Prédécesseurs	Noms ressources
1	Début du projet	0 jou	Mer 16/12/0	Mer 16/12/0		
2	Analyse de l'existant	3 jours	Mer 16/12/0	Mer 13/01/1	1	Antoine CRINIERE,Mikael CAME,Christophe RIGAUD
3	Conception de l'interface	3 jours	Mer 13/01/1	Mer 27/01/1	2	Christophe RIGAUD
4	Implémentation de la vidéo	2 jours	Mer 13/01/1	Mer 20/01/1	2	Mikael CAME
5	Implémentation du calcul SIFT	4 jours	Mer 13/01/1	Mer 03/02/1	2	Antoine CRINIERE
6	Réunion de lancement	0.25 jour	Mer 03/02/1	Mer 03/02/1	5,4,3	Jean-Baptiste FASQUEL,Antoine CRINIERE,Christophe RIGAUD,Mikae
7	Intéraction interface/vidéo	2 jours	Mer 03/02/1	Mer 24/02/1	6	Mikael CAME,Christophe RIGAUD
8	Interaction interface/calcul	3 jours	Mer 24/02/1	Mer 10/03/1	7,6	Antoine CRINIERE,Christophe RIGAUD
9	Optimisation du temps de calcu	2 jours	Mer 10/03/1	Mer 24/03/1	8	Mikael CAME,Antoine CRINIERE,Christophe RIGAUD
10	Ergonomie de l'interface	2.04 jour	Mer 24/03/1	Mer 31/03/1	8,9	Antoine CRINIERE,Christophe RIGAUD,Mikael CAME
11	Réunion de bilan	0.25 jou	Mer 31/03/1	Mer 31/03/1	10	Antoine CRINIERE,Christophe RIGAUD,Jean-Baptiste FASQUEL,Mikae
12	Fin du projet	0 jou	Mer 31/03/1	Mer 31/03/1	11	

Figure 3 : liste des tâches

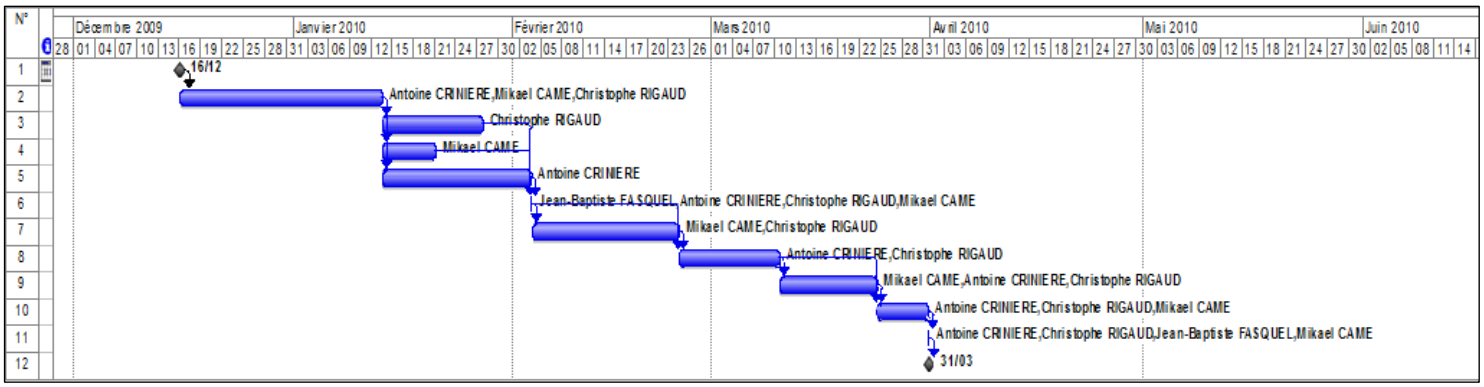


Figure 4 : diagramme de Gantt

2. Technologies et méthodes

2.1. Technologies utilisées pour la reconnaissance

2.1.1. SIFT

La méthode SIFT, développer en 1999 par David Lowe, est invariante, elle permet de décrire un point d'une image en fonction de sa place, de l'endroit de la prise de vue et des points qui l'entourent dans une « entité » appeler KEYPOINT (point clé) représenté comme ci-dessous.

- coordonnées x, y

- inclinaison et échelle rapport a la prise de vue

- un vecteur de 128 paramètres décrivant le point par rapport aux points environnants (descripteur)

Une fois que les « descripteurs » de chaque image (inconnues et connues) ont été générés, il suffit de comparer les points clés de deux images un à un, en calculant la distance euclidienne entre leurs descripteurs. Ensuite, on repère pour chaque vecteur de l'image inconnue, le vecteur de l'image connue avec lequel la distance euclidienne est la plus petite.

Si celle-ci est en dessous d'un certain seuil défini par l'utilisateur, alors on dit que les deux points « match"(correspondent).

On lie l'image inconnue à une image connue en fonction du nombre de keypoints qui sont semblables entre les deux images

2.1.2. ASIFT

Tandis que le SIFT utilise l'échelle, le zoom l'inclinaison et la rotation de l'image pour générer des keypoints, l'ASIFT, elle, prend en compte deux autres paramètres la rendant ainsi plus invariante que le SIFT.

Malheureusement, le code de cette méthode n'est pas public, elle appartient au CMAP de l'école polytechnique. Le code source est disponible, mais le Framework nécessaire à sa compilation est soumis à des ayants droit.

2.1.3. PCA-SIFT

Développé par Yan Ke, Rahul Sukthankar, PCA-SIFT consiste à utiliser et générer des descripteurs SIFT puis de faire tourner un algorithme d'optimisation le PCA. Le code source de cette méthode est disponible, du moins celui de la reconnaissance, par contre la manière dont les descripteurs SIFT sont transformés pour devenir des descripteurs PCA-SIFT est des plus obscure, et demande un temps de traitement assez conséquent.

Après plusieurs essais de cette méthode, il a été conjointement décidé d'utiliser une architecture logicielle similaire à celle du PCA-SIFT, mais en utilisant une autre manière de générer des descripteurs.

Deux choix étaient alors possibles, le premier était de revenir au SIFT, le second d'étudier une nouvelle méthode, OpenSURF.

2.1.4. Open SURF

La méthode OpenSURF est une amélioration de la méthode SIFT. Elle a été mise au point en 2006 par le laboratoire de vision par ordinateur de l'université des sciences et des technologies *ETH Zurich* (Suisse). L'acronyme SURF signifie *Speeded Up Robust Features*, soit en français une *méthode accélérée et robuste*.

Utilisant le principe de base du SIFT, la méthode Surf, elle, génère des descripteurs de 64 paramètres. Ils ont l'avantage d'être plus stables vis-à-vis des variations environnementales, que ceux générés par le SIFT; ce qui leurs confèrent une précision inégalée.

Un autre avantage de l'OpenSURF est l'ensemble des méthodes qui l'accompagne, facilitant ainsi la génération des descripteurs d'image, l'acquisition vidéo et son implémentation en général.

2.2.Technologie utilisée pour la gestion de la base de données

Comme vu précédemment l'un des objectifs de ce projet est de doté chaque robot d'une base de descripteurs d'images génériques et correspondant a des objets courants.

Il doit aussi être possible d'y ajouter des éléments au travers de la fonction d'apprentissage.

Pour ce faire, plusieurs technologies sont à notre disposition du simple fichier texte à la base SQL en passant par une gestion de fichiers XML.

Base de données SQL : Ce type de base, bien que complet et structuré, n'a pas réellement d'utilité vu le peu d'informations que l'on a à stocker. Une seule table serait nécessaire, une base de données relationnelle ne servirait à rien et ralentirait le programme.

Fichiers textes : un fichier texte répertoriant les différents descripteurs d'images (en indiquant leurs chemins) peut être une bonne solution lorsque l'on a peu de fichiers à traiter, car les temps d'accès sont assez longs (recherche de caractères, découpage du fichier, etc.).

Fichiers XML : De même utilité que le fichier texte répertoriant les descripteurs, le XML a un avantage de taille, la rapidité. Rapidité d'accès et rapidité de recherche, de plus il n'est plus nécessaire de « découper » le fichier.

2.3.Technologie utilisée pour l'IHM

La méthode OpenSURF est une amélioration de la méthode SIFT. Elle a été mise au point en 2006 par le laboratoire de vision par ordinateur de l'université des sciences et des technologies *ETH Zurich* (Suisse). L'acronyme SURF signifie *Speeded Up Robust Features*, soit en français une *méthode accélérée et robuste*.

Utilisant le principe de base de la méthode SIFT, la méthode SURF, elle, génère des descripteurs de 64 paramètres. Ils ont l'avantage d'être plus stables vis-à-vis des variations environnementales, que ceux générés par la SIFT; ce qui leur confère une précision inégalée.

Un autre avantage de l'OpenSURF est l'ensemble des méthodes qui l'accompagne, facilitant ainsi la génération des descripteurs d'image, l'acquisition vidéo et son implémentation en général. Pour retrouver l'intégral du projet OpenSURF, veuillez vous référer à la bibliographie.

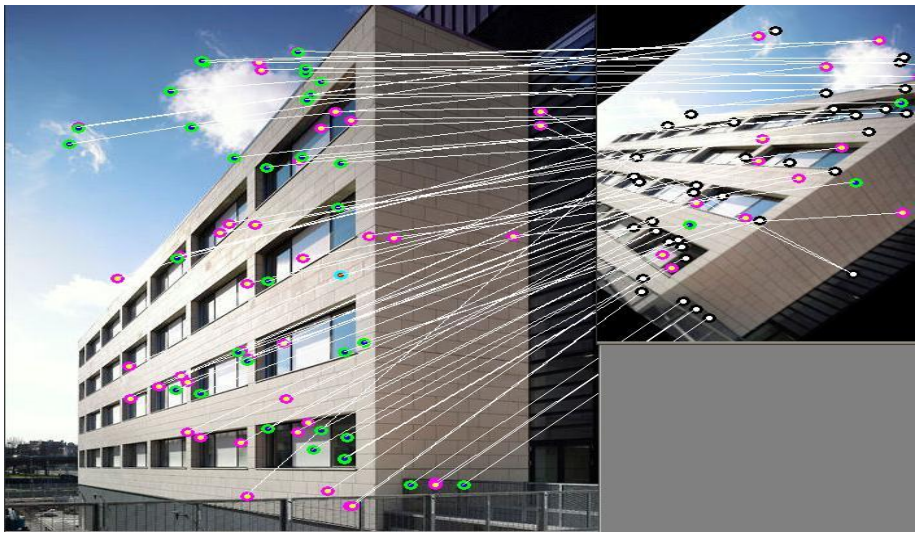


Figure 5 : points similaires retrouvés avec OpenSURF entre deux images

2.4.Technologie utilisée pour la capture d'image

Pour réaliser la capture d'image à partir d'une caméra, nous avons utilisé la librairie OpenCV (Open Computer Vision). Cette bibliothèque libre, développée par Intel, est disponible sur la plateforme Windows et Linux.

Elle est dédiée au traitement d'images. Son haut niveau de codage offre une interface utilisateur ainsi qu'une multitude d'algorithmes. Ce qui permet de faire des applications de manipulations d'images très simplement et donc de s'affranchir du côté technique au niveau du matériel.

Elle permet dans notre cas de créer une fenêtre contenant le flux vidéo d'une caméra connectée à l'ordinateur et d'en extraire une image (png).

3. Déroulement du projet

3.1.Axes du projet

Après avoir étudié le cahier des charges, nous avons planifié la réalisation de notre projet.

Nous avons choisi de réaliser dans un premier temps un prototype, afin de souligner les problèmes éventuels et pour bien maitriser l'ensemble des fonctionnalités, puis dans un deuxième temps, une seconde version qui corrige les défauts du prototype et dans laquelle on va ajouté des fonctionnalités.

Pour gagner du temps lors de la réalisation du prototype, nous avons listé toutes les fonctions nécessaires à l'application, nous les avons ensuite les réalisées et les testées séparément. De cette manière, nous avons pu travailler chacun de notre côté et nous étions surs de ne rien oublier lorsque du regroupement de toutes les parties.

Nous avons utilisé la même méthode pour la réalisation de la version finale, mais en faisant la modélisation UML du programme, afin d'avoir un programme final bien structuré.

3.2.Prototype de l'application

3.2.1. Simulation de l'algorithme

Afin de développer un système de reconnaissance fiable, la première étape fut de simuler celui-ci sous SCILAB.

Comme il a été expliqué précédemment (3.1.1) la reconnaissance s'appuie sur deux points.

La recherche de la distance euclidienne minimum entre un descripteur d'une image inconnue et l'ensemble des descripteurs de chaque image contenue dans la base.

Puis la comparaison de cette distance avec un seuil, dit « seuil de reconnaissance », c'est ce seuil qui définit si un descripteur a une réelle correspondance dans l'image connue.

Dans le cas ou par exemple, deux descripteurs de l'image inconnue correspondent à un seul et même descripteur d'une image connue.

Pour cette première approche, il a été décidé d'utiliser, pour générer les descripteurs de chaque image, l'algorithme SIFT.

Pour cela nous nous sommes rendus sur la page web de David Lowe afin de télécharger un exécutable qui permet de générer les descripteurs, d'une image passée en paramètre, et de les enregistrer dans fichier texte portant l'extension « .key ».

Une fois ces fichiers de descripteurs générés, on les utilise dans un script SCILAB qui va ouvrir deux d'entre eux et faire tourner un algorithme des plus proches voisins entre un descripteur de la première image et l'ensemble de ceux de la deuxième. Puis recommencer avec les autres descripteurs de la première image.

Voici les premiers résultats :

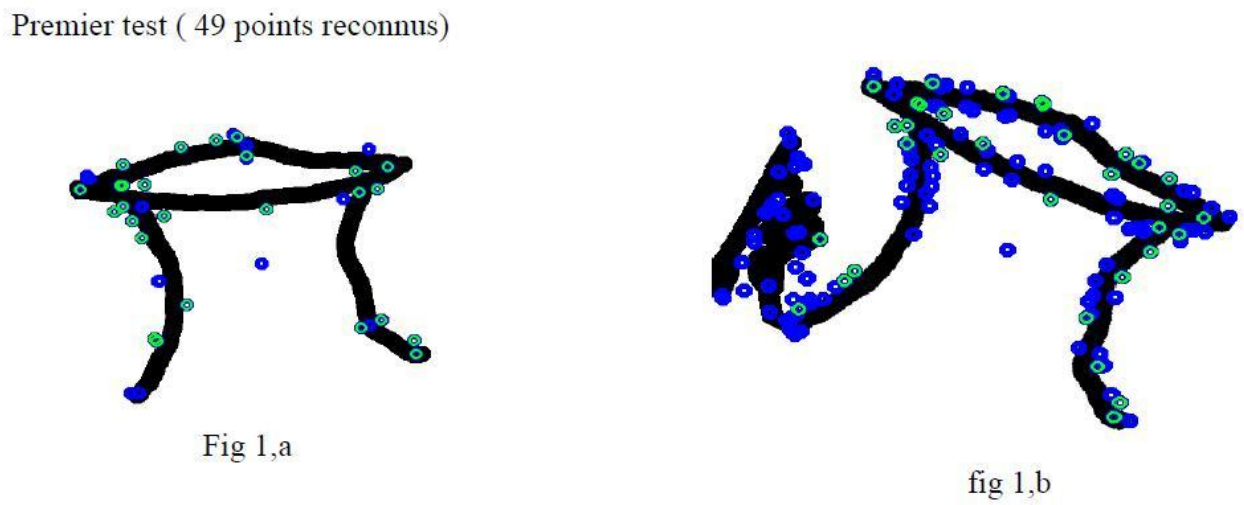


Figure 6 : premiers résultats

Les points bleus représentent l'ensemble des descripteurs de chaque image. Les verts sont eux ceux reconnus entre les deux images.

Il est indiqué 49 points reconnus, en réalité il y en a un peut moins, car pour cette simulation nous n'avons pas défini de seuil de reconnaissance (pour pouvoir exporter les coordonnées des points reconnus) donc deux points de la première image peuvent correspondre a un même point de la deuxième image.

Mais cette simulation nous donne quand même de nombreuses informations quant à la puissance de l'algorithme SIFT et ses conditions d'utilisation c'est d'ailleurs cette version qui a été traduite en C++ et implémentée dans la version 1 de l'application.

3.2.2. Test d'acquisition vidéo

L'objectif de l'acquisition est d'enregistrer une photo au format png, à partir de la webcam.

Le code utilisant la librairie opencv remplit bien sa fonction. Lors d'un clic sur le bouton capture de l'IHM une fenêtre avec le flux vidéo s'ouvrait, puis, avec un appui sur la touche espace, la photo était prise et enregistrée sur le disque dur au format pgm. Ensuite, le convertisseur d'image 'convert' faisait la conversion au bon format.

Mais il y avait quelques problèmes à résoudre.

— Il fallait permettre la sélection de la caméra, car celle par défaut n'est pas forcément celle connectée en USB.

— On voulait avoir, pour les tests, un retour sur la caractérisation, en affichant les points en direct pendant l'acquisition de l'image. De cette manière, on peut savoir si les points capturés caractérisent uniquement l'objet ou si d'autres éléments bruitent la capture.

3.2.3. Première IHM

Dans un premier temps, nous avons recherché les fonctions principales que devais remplir l'Interface Homme Machine; tout en gardant à l'esprit qu'elle n'était qu'une aide au développement car elle ne serait pas nécessaire au système embarqué. Nous l'avons ainsi imaginée avec une zone de contrôle et une zone d'affichage d'image :

Figure 7 : première version de l'IHM

Le projet avançant, nous nous sommes orienté vers le multifenêtrage pour simplifier l'affichage avec les différentes librairies utilisées, notamment GTK et OpenCV. La fenêtre contenant les boutons de contrôles étant codés avec GTK tandis que l'affichage des images et de la vidéo était codé avec OpenCV.

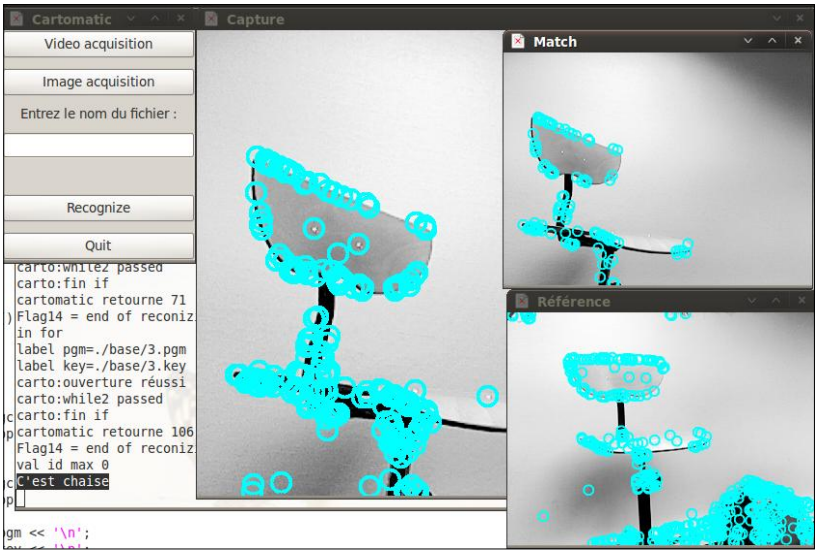


Figure 8 : IHM multifenêtrée

3.3.Résultats du prototypage

Bien que cette première version fonctionne correctement, ses points forts ne sont pas nombreux.

3.3.1. Points forts

Les technologies sont maitrisées, l'algorithme de reconnaissance (plus proches voisins) a fait ses preuves. L'IHM et l'acquisition vidéo fonctionnent et permettent une génération rapide des descripteurs.

3.3.2. Points faibles

L'algorithme SIFT est certes puissant, mais les descripteurs de 128 paramètres qu'il génère obligent à avoir un seuil de reconnaissance élevé, ce qui autorise de « fausses » reconnaissances.

De plus, l'application gère très mal les accès à la mémoire, à chaque reconnaissance les fichiers textes regroupant les descripteurs de chaque image sont ouverts puis lus. L'opération est ainsi répétée pour chaque fichier présent ce qui ralentit considérablement l'application.

3.4.Développement du projet final

3.4.1. Améliorations de la base de données

Les problèmes d'accès mémoire nous ont obligés à revoir notre base de données et l'utilisation de celle-ci.

Pour cela, nous avons décidé de charger la base entière en mémoire au démarrage de l'application, ce qui permet un accès aux descripteurs plus rapide.

Pour ce qui est du chargement de la base en lui même, nous avons décidé d'utiliser un fichier XML auquel nous accédons grâce à la librairie TiniXML.

Voilà maintenant à quoi ressemble la base :

BASE.xml

```
<BASE nb="19">
    <fichier nom="./base/1.xml" nature="chaise noire" />
    <fichier nom="./base/2.xml" nature="armoire a outils" />
    <fichier nom="./base/3.xml" nature="archiveur en carton" />
    <fichier nom="./base/4.xml" nature="livre rouge" />
    <fichier nom="./base/5.xml" nature="stylos" />
    <fichier nom="./base/6.xml" nature="camera de surveillance" />
    <fichier nom="./base/7.xml" nature="triangle" />
    <fichier nom="./base/8.xml" nature="rond" />
    <fichier nom="./base/9.xml" nature="carre" />
    <fichier nom="./base/10.xml" nature="livre jaune" />
    <fichier nom="./base/11.xml" nature="carre2" />
    <fichier nom="./base/12.xml" nature="agenda" />
    <fichier nom="./base/13.xml" nature="croix2" />
    <fichier nom="./base/14.xml" nature="croix3" />
    <fichier nom="./base/15.xml" nature="chris" />
    <fichier nom="./base/16.xml" nature="cr11" />
    <fichier nom="./base/17.xml" nature="rond" />
    <fichier nom="./base/18.xml" nature="carre3" />
<fichier nom="./base/155.xml" nature="agenda inconnue" />
</BASE>
```

Comme vous pouvez le voir, la base est définie par un paramètre nommé « NB » qui nous renseigne sur le nombre de fichiers présents (ce qui accélère encore le traitement), puis chaque fichier est décrit par le chemin du fichier, ainsi que la nature de l'image qu'il représente.

3.4.2. L'intégration de OpenSURF

Avant d'intégrer la bibliothèque OpenSURF au projet (et donc remplacer la bibliothèque SIFT), nous l'avons testé longuement afin de sélectionner les éléments qui pourraient nous être utiles ainsi que les modifications en découlant.

Pour commencer, nous avons étudié le fonctionnement de chaque scénario proposé dans le projet OpenSURF. Ces scénarios incluaient :

- le calcul des points clefs d'une image existante ;
- la superposition des points clefs trouvés sur une image en cours d'acquisition (webcam…)
- la recherche d'un objet dans une image
- l'affichage d'objet mobile
- l'affichage des points qui similaires dans deux images et leurs correspondances

Nous avons finalement réutilisé le premier scénario, plus précisément la méthode d'extraction des points clefs et des descripteurs d'une image. Cette méthode convertit une image 32bits (au format png ou pgm) en une image 32bits monochrome; puis y applique la méthode de calcul de la matrice de covariance (Hessian) pour obtenir les descripteurs de l'image. Le détail de la méthode implémentée est en annexe 3 p.33.

Cette méthode est utilisée lors de l'acquisition vidéo, de l'apprentissage ou encore lors de la génération manuelle de descripteurs (keypoint generator).

Pour assurer la compatibilité avec le reste du programme, l'application convertie, le tableau de descripteurs crée par la méthode d'extraction (OpenSURF), en un fichier XML qui pourra être comparé ou intégré à la base de données d'images.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<key nb_coord="21" nb_desc="64">
        <coord x="280.359" y="34.0111">
                <desc val="0.00899623"/>
                … (x 64)
                <desc val="0.0145478"/>
        </coord>
        <coord x="296.622" y="42.1445">
                <desc val="0.0115397"/>
                … (x 64)
                <desc val="0.0178634"/>
        </coord>
        ...(x 21 ici)
</key>
```

Figure 9 : structure d'un fichier XML généré à la suite d'une extraction de points clefs

3.4.3. Diagrammes UML du projet

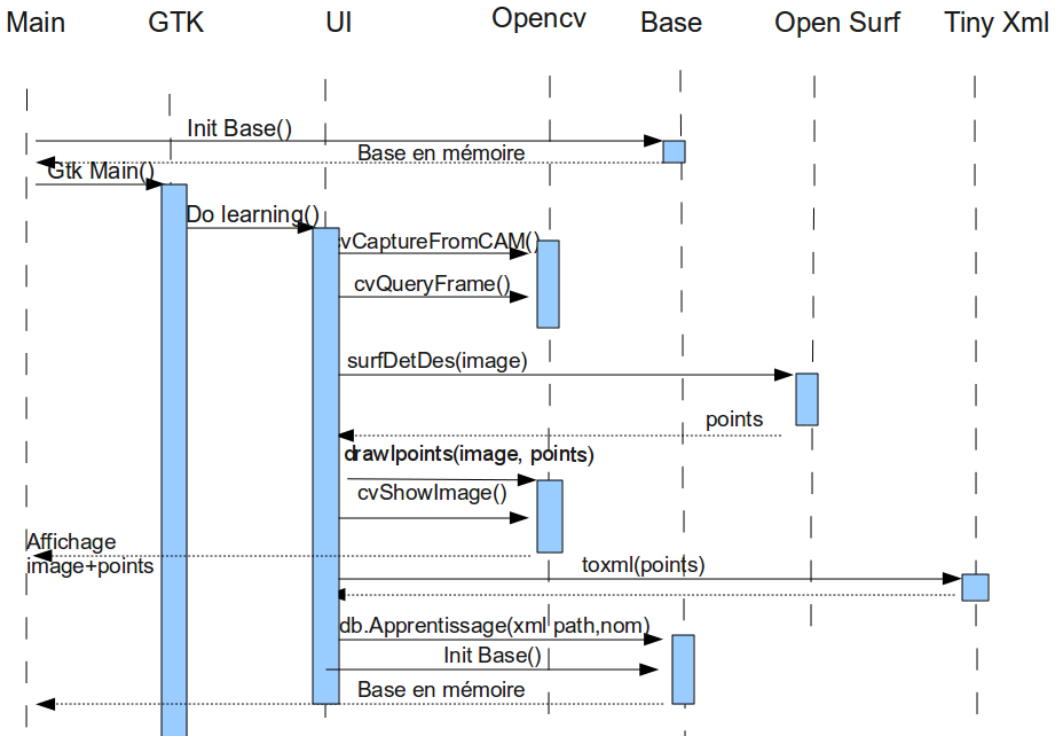


Figure 10 : processus d'apprentissage

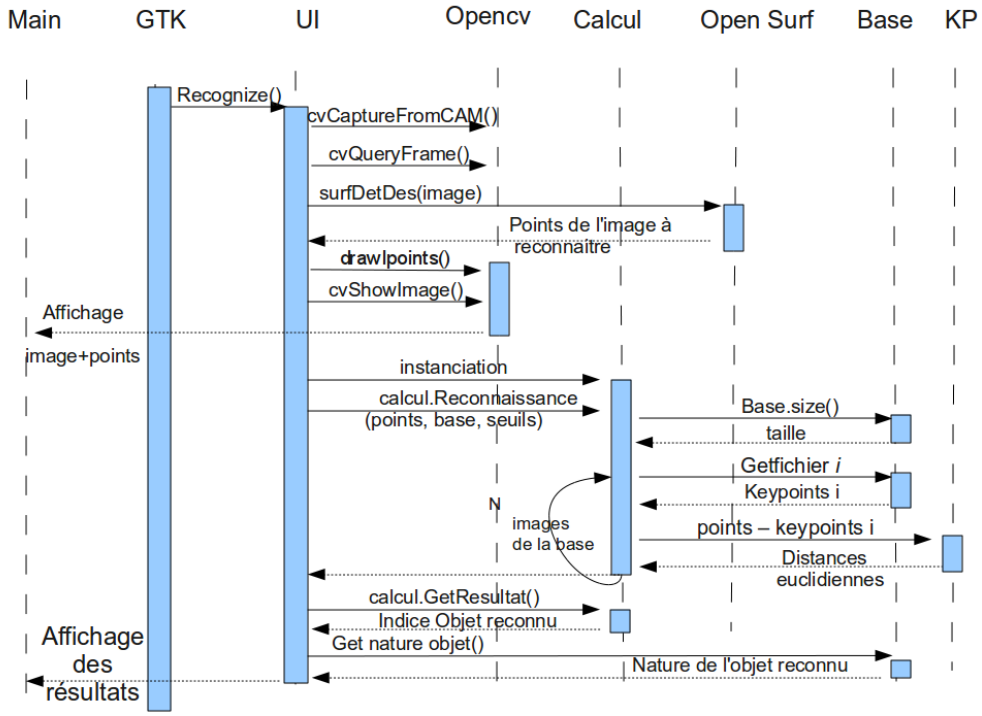


Figure 11 : processus de reconnaissance

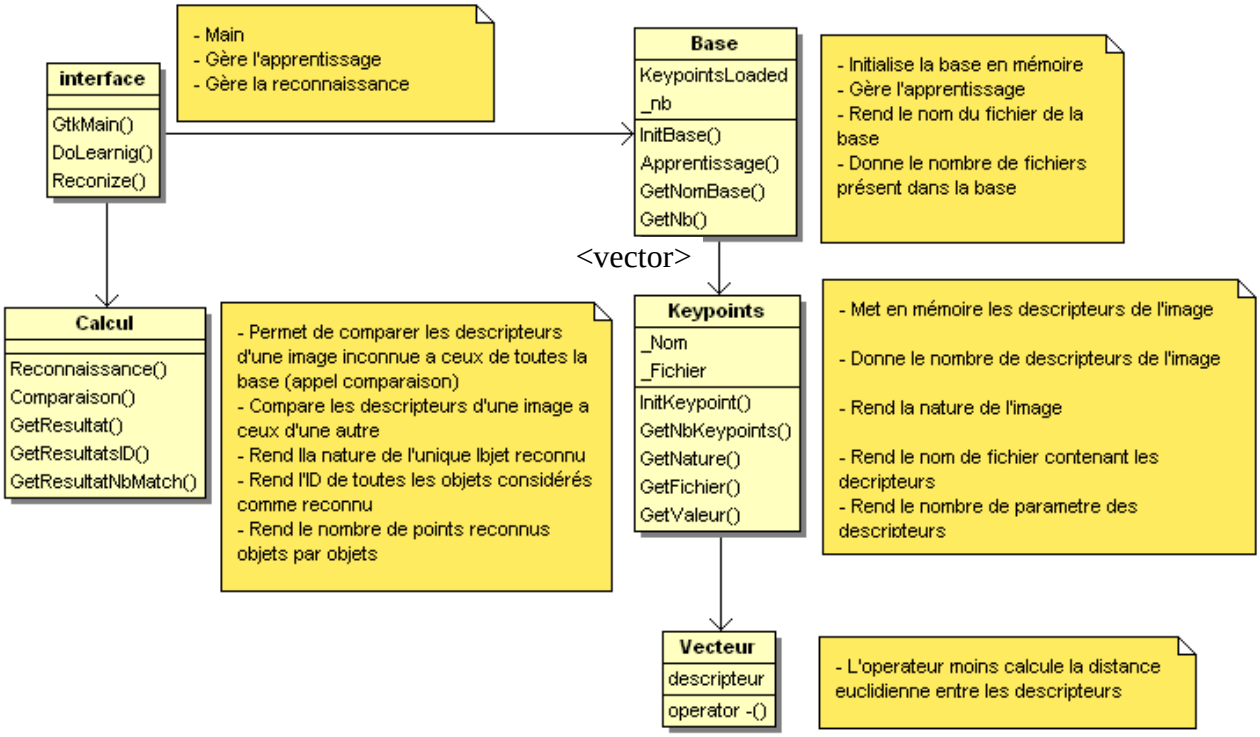


Figure 12 : diagramme de classes UML

3.4.4. Fonctionnement de l'application

3.4.4.1. Pré requis à la compilation

Pour pouvoir compiler l'application en C++, avec CodeBlocks ou par makefile, nous avions besoin du compilateur g++. Nous avons donc utilisé le gestionnaire de paquet Synaptic présent dans la distribution Ubuntu 9.10 – the Karmic Koala pour les télécharger.

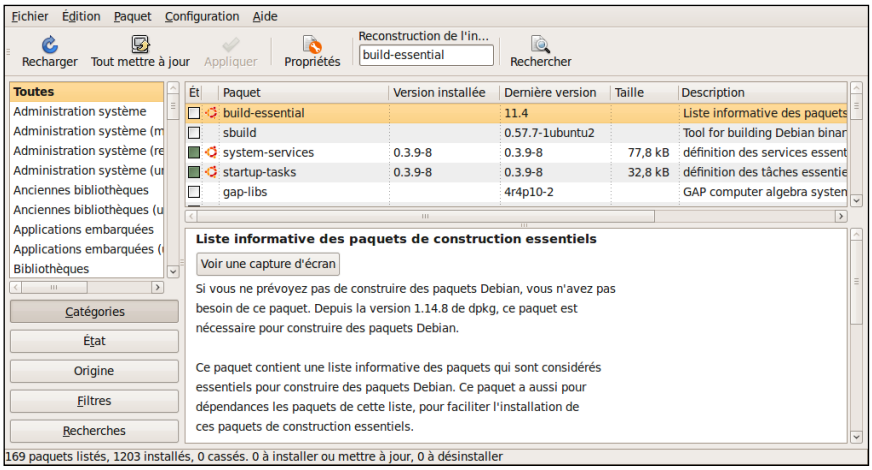


Figure 13 : le gestionnaire de paquet Synaptic

Au préalable, il a fallu configurer le terminal (utilisé en fond de tâche pour télécharger les paquets) pour le proxy de l'université. Pour ce faire, il suffit d'entrer dans le terminal :

```
export "http_proxy=http://eproxy.uang:3128"
```

Les paquets (bibliothèques) nécessaires à la compilation du projet sont :

La base pour la compilation sous Ubuntu :	build-essential et g++
Pour GTK :	libgtk2.0-dev
Pour OpenCV	apt://libcv1 et apt://libcv-dev
Pour trouver les paquets utiles à la compilation :	apt-cache showsrc
Pour rechercher les dépendances :	sudo apt-get build-dep

L'utilisation de ces paquets est détaillée en suivant les liens de la webographie p. 28.

3.4.4.2. La compilation

Pour compiler le programme, nous avons utilisé un fichier makefile pour l'Interface Homme Machine ainsi que CodeBlocks pour les différentes méthodes de calcul utilisées pour la reconnaissance d'objets. Cela nous a permis de bien comprendre les différentes étapes de la compilation.

Un exemple de ligne de commande constituée par le makefile :

```
gcc $(pkg-config –libs –cflags gtk+-2.0) main.c -o nom_de_lexecutable
```

Le makefile complet et commenté est disponible à l'annexe 2 p.32.

3.4.4.3. L'interface

Nous avons conçu l'application pour être utilisées de deux manières bien distinctes :

Par l'intermédiaire d'une interface pour pouvoir fiabiliser son fonctionnement aisément en ajustant les différents paramètres lors des campagnes d'essais.

Sans interface, c.-à-d. directement commander par un microcontrôleur sur un système embarqué (autonome et mémoire limitée)

Pour le premier cas, **l'interface**, nous avons fait le choix de la développer en anglais pour standardiser le nom des fonctions et faciliter la compréhension par une personne étrangère.

Voici l'apparence de l'interface finale :

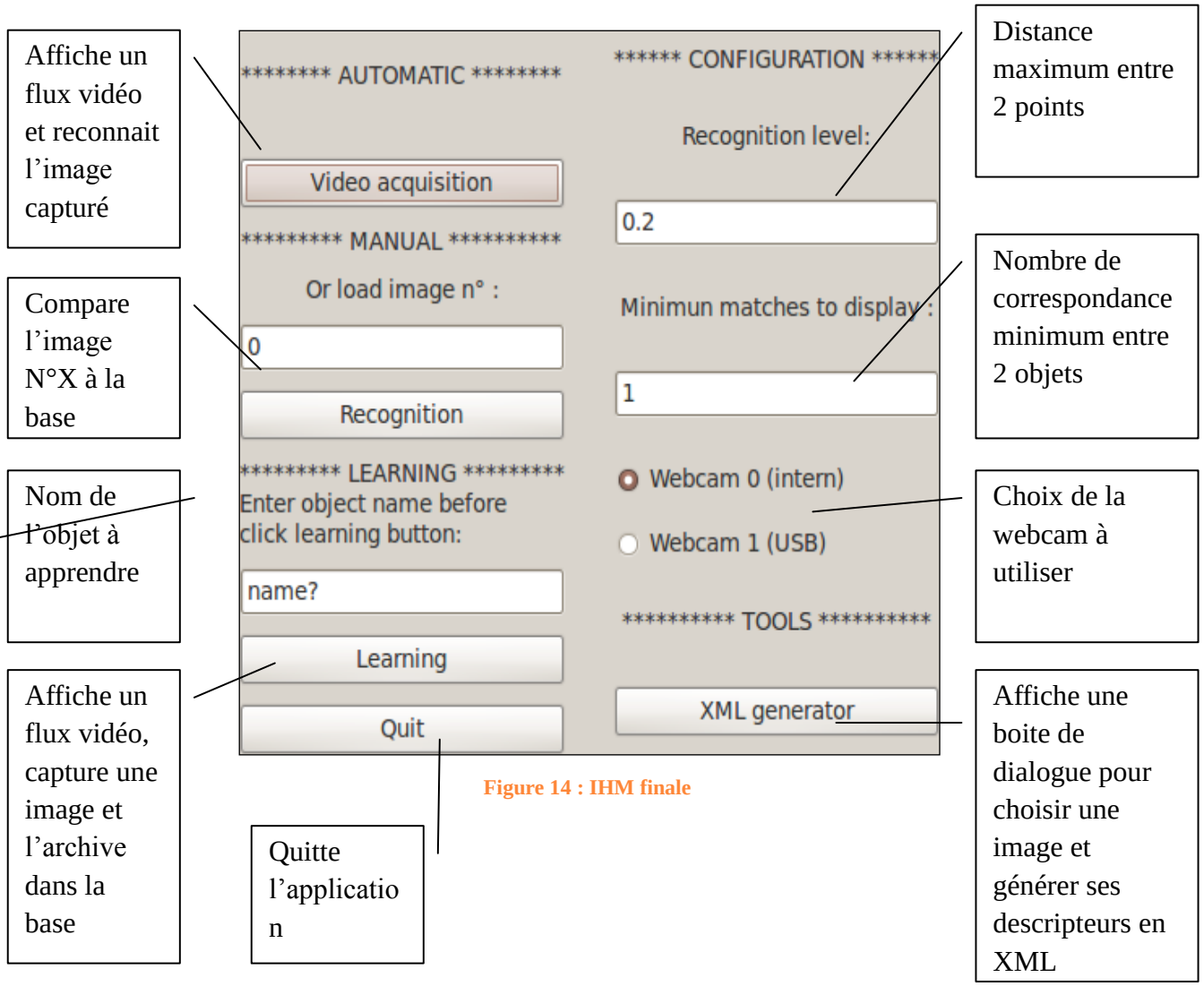


Figure 14 : IHM finale

Description des cinq zones (séparée par des ***titres***) :

- Automatic (automatique) : pour effectuer une reconnaissance automatique d'un objet quelconque. Cela consiste à afficher l'image de la webcam sélectionnée, faire une capture d'image et retrouver les images les plus ressemblantes parmi celles contenues dans la base de données d'images.
- Manual (Manuel) : pour la reconnaissance manuelle d'objets déjà présents dans une image.
- Learning (apprentissage) : contient les éléments nécessaires à l'apprentissage d'un objet (ou forme) à savoir son nom et son acquisition par la webcam.
- Configuration (configuration) : regroupe les principaux paramètres qui peuvent varier en fonction du milieu et de la machine où l'on souhaite utiliser l'application.
- Tools (outils) : permet d'accéder à des fonctions secondaires telles que la génération manuelle de fichier XML contenant les points intéressants (et leurs descripteurs) d'un fichier.

Concernant le second cas, **l'application embarquée**, il suffit de remplacer la fonction principale (main) par des appels directs aux fonctions techniques (métier) telles que la reconnaissance ou l'apprentissage.

3.4.4.4. Le suivi par la console

Lorsque l'application s'exécute, des informations sur son activité apparaissent dans la console (terminal). Cela permet de suivre le déroulement des actions en détail ; très utiles lors du débogage ou lors d'une exécution sur une machine lente.

Voici un exemple d'information que l'on peut apercevoir dans la console :

```
make&&make run
make:
Nothing to be done for `all'.
./cartomatic
Chargement de 1.xml
…
La base s'est chargée correctement.
Acquisition avec la camera n°0...
Calcul des descripteurs...
Construction du fichier 0.xml contenant les descripteurs du snapshot...
Le snapshot (0.png) a été sauvegardé dans le dossier snap.
Reconnaissance en cours au seuil de 0.2...
Comparing with base\2.xml => match=3 => matchMax=3
...
0 armoire a outils (./base/2.xml)
```

Figure 7 : suivi console

La première ligne *make&&make run* est interprétée par le fichier *makefile* qui va compiler l'ensemble des fichiers du projet (si ils ont été modifiés) puis lancer l'application. Ensuite il y a une liste d'information qui correspond à toutes les actions nécessaires à l'accomplissement des actions demandées par l'utilisateur. Il est donc plus aisé de localiser une éventuelle erreur ou bien de comprendre qu'est ce qu'est en train de faire le programme s'il parait inactif.

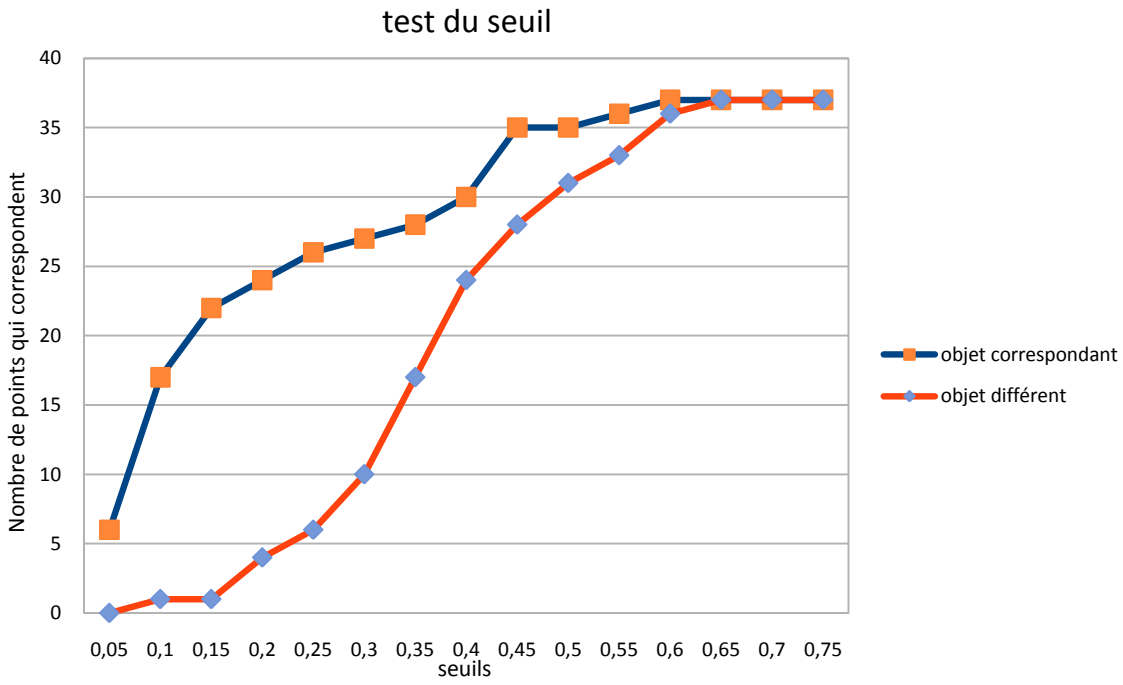
3.5. Résultats de l'application

3.5.1. – Réglage du seuil de reconnaissance

Ce test consiste à calibrer le seuil de tolérance, en testant plusieurs valeurs de seuils. On retiendra la valeur qui arrive à écarter les mauvais objets, sans que ce seuil soit trop restrictif. On fait les tests avec la même image, en la comparant avec dans la base, un objet identique et un objet différent.

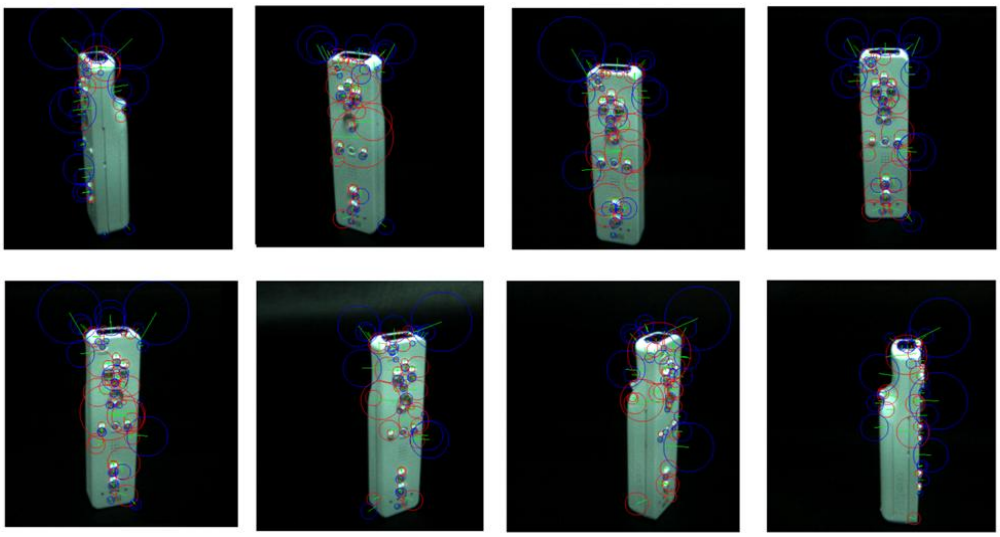
La valeur retenue à l'issue de ce test est 0,2.

Voici les résultats :



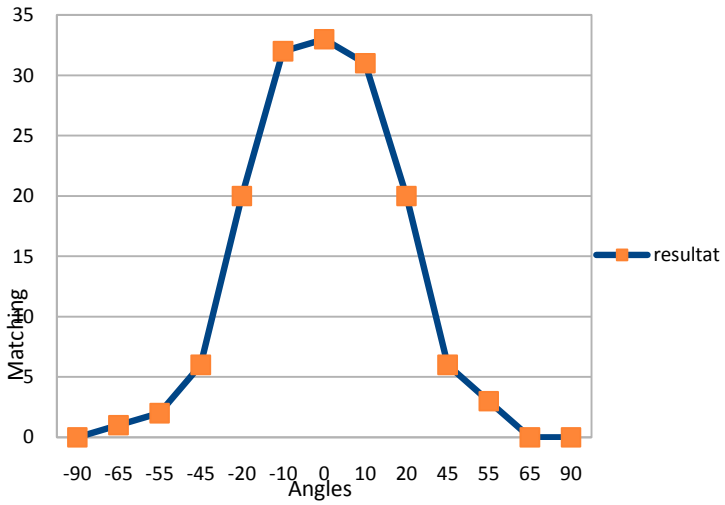
3.5.2. Rotation de l'objet

Le logiciel de reconnaissance doit pouvoir reconnaitre des objets avec un certain angle par rapport à l'image dans la base de données.



Ce test consiste à voir l'effet de la rotation sur le nombre de points qui sont mis en correspondance (matching). Nous avons enregistré l'image d'une Wiimote de face et nous l'avons comparé avec des images de Wiimote avec différents angles.

Voici les résultats :



Ce test montre bien que le logiciel reconnait des objets, même si l'objet est tourné par rapport à l'image dans la base de données. Au-delà de 45° degrés le « matching » est très bas, car on ne peut plus voir la face caractérisée. Pour cela, il faut enregistrer dans la base un même objet plusieurs fois avec des angles différents.

3.6.Conclusions sur le projet final

Grâce à ces tests, nous pouvons enfin conclure sur le résultat obtenu par l'application.

Un objet appris par l'application lui permet bien ensuite de le découvrir dans un environnement, avec une photo de celui-ci. Si plusieurs objets sont présents dans l'image de l'environnement, l'application les repère et les signale également.

Notre programme réalise donc le cahier des charges initial. L'implantation sur le robot sera facile il suffit d'automatiser son utilisation.

Le seul frein sera peut-être au niveau de l'utilisation du processeur, car le traitement de l'image et la reconnaissance demandent des ressources de calculs importantes. Si les ressources sont trop limitées, on peut imaginer l'ajout d'un processeur (ou ordinateur embarqué) dédié uniquement à cette tâche.

L'apprentissage est la partie la plus délicate de l'utilisation du logiciel. En effet si les objets sont mal caractérisés, alors il sera difficile de les repérer ensuite. La difficulté vient du fait que lors de l'apprentissage, c'est tout les points de l'image prise qui sont sauvegardés, ainsi un objet, une forme ou encore des couleurs dans le fond peuvent perturber la caractérisation de l'objet. Pour nos tests nous avons utilisé un fond noir ou blanc pour réaliser l'apprentissage. De plus, l'objet à apprendre était en gros plan afin de réduire le bruit possible.

On pourrait imaginer, pour une version future du logiciel, la possibilité de sélectionner les points caractéristiques grâce à l'IHM, en sélectionnant la zone où se trouve l'objet et en supprimant les points à l'extérieur de celle-ci.

Notre application est la partie vision d'une application de cartographie implémentée dans un robot, mais elle pourrait être utilisée dans d'autres domaines, d'autres contextes, ce qui ouvrirait d'autres possibilités.

Une implémentation sur un téléphone portable et grâce à une photo prise d'un bâtiment historique par exemple, pourrait donner accès à des informations sur ce lieu. Dans un magasin, l'application pourrait permettre d'écouter de la musique ou de voir la bande-annonce d'un film juste en prenant une photo de la couverture d'un CD ou d'un DVD.

On peut encore imaginer d'autres implémentations, dans une caisse enregistreuse de magasin par exemple, où l'application permettrait de reconnaitre rapidement les articles qui passent sur le convoyeur sans utiliser de codes-barres.

Conclusion

Ce projet a été pour nous l'occasion de voir à quoi ressemble un processus de recherche technologique et comment travailler en équipe.

Le partage des tâches, l'entraide, l'apprentissage et la satisfaction du travail réalisé ont été les maitres mots durant ce projet.

Appliqué aux domaines de la sécurité, culturel ou purement commercial, le traitement et la caractérisation d'une image sont des technologies en vogue en ce moment.

La découverte de ces technologies a été pour nous un point important de notre formation, c'est pour cela que nous remercions, de nouveau, M. Fasquel de nous avoir guidés tout au long de ce projet.

Bibliographie / webographie

Compilation sous Ubuntu

http://doc.ubuntu-fr.org/tutoriel/compilation

Récupération de paquet avec aptitude

http://manpages.ubuntu.com/manpages/jaunty/fr/man8/aptitude.8.html

Conversion entre les différents formats des parquets disponibles (.deb, .rpm, .tgz, .slp et .pkg)

http://doc.ubuntu-fr.org/alien

Utilisation du logiciel de creation d'interface GTK : Glade

Lien : http://glade.gnome.org/

http://tadeboro.blogspot.com/2009/09/glade3-tutorial-2-constructing.html

Installation des paquets pour l'utilisation GTK sous Ubuntu

http://www.siteduzero.com/tutoriel-3-2995-installer-gtk-sous-linux.html

Installation des paquets pour l'utilisation d'OpenCV sous Ubuntu

http://doc.ubuntu-fr.org/opencv

Développement avec GTK

http://gtk.developpez.com/cours/gtk2/Gtk.Les.images.php

http://www.gtk-fr.org/wakka.php?wiki=LeTutorial

http://library.gnome.org/devel/gtk/unstable/index.html

Générateur de documentation de code source : Doxygen

http://www.stack.nl/~dimitri/doxygen/

La méthode OpenSURF

http://www.vision.ee.ethz.ch/~surf/

La bibliothèque SIFT, ASIFT et PCA-SIFT

http://www.cs.ubc.ca/~lowe/home.html

http://www.cmap.polytechnique.fr/~yu/research/ASIFT/demo.html

[Ke2004] Y. Ke, R. Sukthankar, *PCA-SIFT: a more distinctive representation for local image descriptors*, in: Proceedings of the IEEE International Conference on, Computer Vision and Pattern Recognition, 2004.
[Lowe2004] D.G. Lowe, *Distinctive image features from scale-invariant keypoints*, International Journal of Computer Vision 60 (2) (2004) 91–110.
[Lowe1999] David G. Lowe, *Object recognition from local scale-invariant features*, International Conference on Computer Vision, 1999.
[Leaodeanu2005] M. Leordeanu and M. Herbert, A spectral technique for correspondence problems using pairwise constraints, in: Proceedings of the IEEE International Conference on Computer Vision, 2005.

[Yan Ke, Rahul Sukthankar]PCA-SIFT: A More Distinctive Representation for Local Image Descriptors

OpenCv :

http://www2.ulg.ac.be/telecom/assistant/tni/opencv.html

Learning OpenCV (Broché)

de Gary R. Bradski (Auteur), Adrian Kaehler (Auteur)

ANNEXES

Annexe 1 : code GTK de la première IHM

Exemple de code:

```
#include <stdlib.h>
#include <gtk/gtk.h>

int main(int argc, char **argv)
{
   GtkWidget *pWindow;
   GtkWidget *pVBox;
   GtkWidget *pImage;
   GtkWidget *pQuitImage;
   GtkWidget *pQuitBtn;

   gtk_init(&argc, &argv);

   pWindow = gtk_window_new(GTK_WINDOW_TOPLEVEL);
   gtk_window_set_default_size(GTK_WINDOW(pWindow), 320, 200);
   gtk_window_set_title(GTK_WINDOW(pWindow), "GtkImage");
   g_signal_connect(G_OBJECT(pWindow), "destroy", G_CALLBACK(gtk_main_quit), NULL);

   pVBox = gtk_vbox_new(FALSE, 0);
   gtk_container_add(GTK_CONTAINER(pWindow), pVBox);

   /* Chargement d'une image a partir d'un fichier */
   pImage = gtk_image_new_from_file("./gtk.png");
   gtk_box_pack_start(GTK_BOX(pVBox), pImage, FALSE, FALSE, 5);

   pQuitBtn = gtk_button_new();
   gtk_box_pack_start(GTK_BOX(pVBox), pQuitBtn, TRUE, FALSE, 5);
   g_signal_connect(G_OBJECT(pQuitBtn), "clicked", G_CALLBACK(gtk_main_quit), NULL);

   /* Chargement d'une image a partir d'un GtkStockItem */
   pQuitImage = gtk_image_new_from_stock(GTK_STOCK_QUIT, GTK_ICON_SIZE_LARGE_TOOLBAR);
   gtk_container_add(GTK_CONTAINER(pQuitBtn), pQuitImage);

   gtk_widget_show_all(pWindow);

   gtk_main();
```

Annexe 2 : le makefile

```
# Cartomatic makefile
# Indicates compilator to use
CC      = g++
# Specifies compilator options
CFLAGS  = `pkg-config gtkmm-2.4 --cflags opencv` -lcv -lhighgui -DDEBUG -D LINUX
LDLIBS  = `pkg-config gtkmm-2.4 --libs opencv`
# File extension suffixes .cpp, .o
SUFFIXES = .cpp .o
.SUFFIXES: $(SUFFIXES)
# Name of the main program
PROG  = cartomatic
# Object files .o necessary to build the main program
OBJS  =  tinyxml.o tinystr.o tinyxmlerror.o tinyxmlparser.o surfclass.o main.o
fasthessian.o integral.o surf.o utils.o ipoint.o base.o keypoints.o vecteur.o calcul.o
# Preprocessor directives
all: $(PROG)
# Compilation and link
$(PROG): $(OBJS)
        $(CC) $(LDFLAGS) -o $(PROG) $(OBJS) $(LDLIBS)
.cpp.o:
        $(CC)   $(CFLAGS) -c $< -o $@
dist:
        bash makedistlinux
clean:
        rm -rf *.o
mrproper:clean
        rm -rf $(PROG)
run:    ./$(PROG)
```

Annexe 3 : la méthode de reconnaissance de points intéressants et de descripteurs

```
//! Library function builds vector of described interest points

inline void surfDetDes(IplImage *img,                 /* image to find Ipoints in */

                       std::vector<Ipoint> &ipts,     /* reference to vector of Ipoints */

                       bool upright = false,          /* run in rotation invariant mode? */

                       int octaves = OCTAVES,         /* number of octaves to calculate */

                       int intervals = INTERVALS,     /* number of intervals per octave */

                       int init_sample = INIT_SAMPLE, /* initial sampling step */

                       float thres = THRES  ){        /* blob response threshold */

 // Create integral-image representation of the image

 IplImage *int_img = Integral(img);

 // Create Fast Hessian Object

 FastHessian fh(int_img, ipts, octaves, intervals, init_sample, thres);

 // Extract interest points and store in vector ipts

 fh.getIpoints();

 // Create Surf Descriptor Object
```

Projet réalisé par : **Antoine Crinière, Mikael Came et Christophe Rigaud**
Projet encadré par : **Jean-Baptiste Fasquel**

RESUME

Ce rapport présente une méthode de reconnaissance d'images pour cartographier une pièce en utilisant SURF (Speeded Up Robust Feature) un détecteur et descripteur d'images, efficace. L'équipe de travail est constituée de quatre étudiants de 4ème année à l'ISTIA; le travail été essentiellement basé sur l'application (logiciel) de cartographie qui sera embarquée dans un robot ultérieurement. Ce projet a duré 80 heures.
Sa fonction principale est de reconnaître des objets simples dans une pièce. Mais avant cela, il doit apprendre une base de données d'objets en image : c'est la deuxième fonction principale. L'apprentissage est composé de l'acquisition d'une image, du calcul de ses points intéressants, de la saisie de son nom et de sa sauvegarde.
La reconnaissance comporte l'affichage d'une vidéo en temps réel, la capture d'un instantané et la comparaison avec la base de données d'images pour vérifier si quelque chose y ressemble. S'il l'image comporte quelque chose de commun avec la base d'images, les objets reconnus sont affichés avec leur taux de correspondance respectif. Sinon, l'application informe l'utilisateur qu'aucun objet n'a été reconnu et propose de l'apprendre.
Les points d'intérêts sont représentés avec les descripteurs SURF. Une recherche symétrique sur chaque descripteur est effectuée pour détecter un ensemble de paires de points intéressants. Les résultats expérimentaux illustrent une bonne précision et une meilleure efficacité avec la méthode SURF en comparaison avec d'autres techniques de reconnaissance (SIFT: Scale Invariant Feature Transform).

MOTS-CLES : reconnaissance, image, apprentissage, vidéo, Automatique, robot

SUMMARY

This report presents an image matching method based on room mapping and SURF (Speeded Up Robust Feature) a robust image detector and descriptor. The work team is composed of 4th year students; working essentially on the mapping application that will be embedded into a robot. The project duration is 80 hours. The main function is to recognize basic objects in a room. Before that, object data base learning is necessary, that's the second main function.
The learning included a snapshot, a calculus of interesting points, a content name request and a backup.
Recognition is composed of the display of a live video, the capture a snapshot and the comparison of all the images in the database to check if something matches. If there is a match, the recognized objects are displayed with their matching rates. Else, the application informs the user no match is found and proposes to learn it.
Interesting points are represented with SURF descriptors. One to one symmetric search is performed on descriptors to select a set of matched interesting points. Experimental results illustrate high accuracy and superior computational efficiency of proposed approach in comparison with other matching techniques (SIFT: Scale Invariant Feature Transform).

KEYWORDS: recognition, image, learning, video, automatic control, robot