# Contributions à la dextérité d'un système de réalité augmentée mobile appliqué à la maintenance industrielle

Jean-Yves Didier

*UNIVERSITÉ D'ÉVRY - VAL D'ESSONNE*
*ECOLE DOCTORALE SITEVRY*

# ***T H E S E***

pour obtenir le grade de

**DOCTEUR DE L'UNIVERSITÉ D'ÉVRY**

*Spécialité : ROBOTIQUE*

présentée et soutenue publiquement

par

Jean-Yves DIDIER

le 12 décembre 2005

***Titre :* Contributions à la dextérité d'un système de réalité augmentée mobile appliqué à la maintenance industrielle**

———

***Directeur de thèse : Malik MALLEM***

———

**JURY**

| | | |
|---|---|---|
| Mme. Marie-Odile Berger | , Chargé de recherche INRIA (LORIA), HDR | , Rapporteur |
| M. Jean-Marc Lavest | , Prof. Université Blaise-Pascal (Clermont-Ferrand) | , Rapporteur |
| M. Nassir Navab | , Prof. Technische Universität München | , Examinateur |
| M. Florent Chavand | , Prof. Université d'Evry | , Examinateur |
| M. Malik Mallem | , Prof. Université d'Evry | , Examinateur |
| M. David Roussel | , MdC IIE-CNAM | , Examinateur |
| M. Samir Otmane | , MdC Université d'Evry | , Examinateur |

# Table des matières

# Table des figures

# Remerciements

Je souhaite remercier les nombreuses personnes qui m'ont apporté leur soutien et leur aide au cours de ces années pendant lesquelles j'ai accompli ce long travail de fond qu'est une thèse.

Ma gratitude va en particulier à Malik Mallem qui a dirigé cette thèse et m'a judicieusement conseillé tout au long de ce travail, à David Roussel qui m'a encadré plus particulièrement sur l'aspect de la gestion des augmentations et de la méthode de pseudo-correction (post-rendering), à Samir Otmane qui m'a aidé sur la partie concernant l'architecture informatique et enfin à Fakhr-eddine Ababsa dont la participation à la partie localisation par la vision et à la partie concernant le filtrage et la prédiction est non des moindres.

Les partenaires de l'aventure que fût le projet AMRA, trouvent également leur place dans ces lignes, je pense en particulier pense à Christine Mégard, Christophe Leroux, Sylvie Naudet, Quoc-Cuong Pham et Arnaud Hocquard. Que les stagiaires qui ont également contribué directement ou indirectement à ce travail soient remerciés : en particulier Madjid Maidi qui est par la suite devenu un sympathique co-doctorant, mais aussi André Laurie, Jérôme Nancy, Yoann Petit et enfin Cyril Lanquetuit.

Je remercie également les membres du jury, en particulier Marie-Odile Berger et Jean-Marc Lavest pour avoir accepté de rapporter ce travail et apporter une contribution critique à celui-ci, ainsi que Nassir Navab qui a accepté de venir d'Allemagne en tant qu'examinateur.

Florent Chavand fera également l'objet de ma reconnaissance à double titre : en tant qu'examinateur faisant partie du jury et en tant qu'ancien directeur du laboratoire qui a mis à disposition sa structure pour m'accueillir lors du commencement de cette thèse. Son successeur à la tête du laboratoire, Étienne Colle, sera tout autant remercié.

De plus, comment ne pas citer aussi l'ensemble des doctorants pour l'ensemble des moments partagés en particulier Mr Merad (que les autres me pardonnent de ne pas tous les avoir expressément nommés) ainsi que le personnel technique et administratif que j'ai côtoyé pendant ces quelques années et qui a contribué à alléger mes charges en rapport direct ou très indirect avec cette thèse.

Enfin, je tiens à remercier ma famille pour le support et l'aide qu'elle m'a fournie, me laissant retrouver un havre de paix et de repos parfois nécessaire pendant les moments de tourmente que seul un doctorant peut connaître. La pensée finale sera pour Napassanan qui fût aimante, patiente et compréhensive au cours de ces dernières années.

*The most merciful thing in the world, I think, is the inability of the human mind to correlate all its contents. We live on a placid island of ignorance in the midst of blackseas of infinity, and it was not meant that we should voyage far. The sciences, each straining in its own direction, have hitherto harmed us little ; but some day the piecing together of dissociated knowledge will open up such terrifying vistas of reality, and of our frightful position therein, that we shall either go mad from the revelation or flee from the deadly light into the peace and safety of a new dark age.*
-H.P. Lovecraft, "The Call of Cthulhu"

# Introduction générale

En 1982, le public des salles de cinéma pu découvrir le long-métrage Tron. Ce film était l'un des premiers à mêler de façon intensive des images générées par ordinateur et des séquences filmées. Depuis cette époque, l'industrie cinématographique n'a eu de cesse d'employer et d'incruster des images de synthèse de plus en plus élaborée dans les séquences filmées, afin d'introduire des effets spéciaux supplémentaires qui sont parfois impossibles à produire dans la réalité. Cette incrustation d'images de synthèse qui n'ont aucune existence physique dans un scène réelle constitue en quelque sorte un apport à la réalité ou une "augmentation".

Ces techniques, dites de post-productions puisque employée après que les prises de vue d'images réelles aient été effectuées, constituent en quelque sorte l'illustration du concept de base de la réalité augmentée. La réalité augmentée consiste à enrichir, renforcer la réalité à l'aide d'informations supplémentaires pour mieux l'appréhender. La réalité augmentée caractérise tout système qui améliore la perception de l'environnement réel, généralement par superposition d'images de synthèse, d'objets virtuel, de symboles, de schémas ou d'information textuelle sur des images réelles. D'autres type d'association entre le réel et le virtuel sont possibles, notamment à l'aide de sons ou de retour d'efforts.

De plus, à la différence de la post-production, un système de réalité augmentée doit interagir en temps réel avec le monde physique. Dans le domaine des spectacles télévisés filmés en direct, les techniques de réalité augmentées commencent à être employées pour insérer des images publicitaires sur les terrains de sport ou même pour des jeux comme en témoignent les projets de recherche de la chaîne anglaise BBC par exemple [BBC, url].

Si l'augmentation de la réalité augmentée est aujourd'hui de plus en plus couramment employée dans le domaine des divertissements et du spectacle, il n'en demeure pas moins que ce thème de recherche ne reste pas limité à ce contexte. Ainsi, la réalité augmentée est également employée dans le domaine de la maintenance industrielle où elle sert souvent le double objectif de faciliter l'accès aux informations nécessaires à l'opération d'une tâche de maintenance pour l'opérateur expérimenté d'une part, et, d'autre part, elle peut constituer un outil de formation et d'apprentissage pour le mainteneur inexpérimenté.

Cette thèse porte sur la problématique posée par les systèmes de réalité augmentée (RA) mobile appliqués à la maintenance industrielle. L'objectif principal est d'apporter des contributions à la dextérité d'un système de RA mobile. En effet, les contraintes de flexibilité, de temps réel et de précision seront prises en compte tout le long du travail présenté.

Dans le chapitre 1, le domaine de recherche qu'est la réalité augmentée est introduit. Un ensemble de systèmes et de prototypes sont étudiés et analysés afin d'en dégager une architecture générale, ainsi que les verrous scientifiques et technologiques qui y sont associés tels que : la localisation du système par rapport à l'espace de travail et aux objets qui le compose, le lien sémantique entre ce qui est vu et la base de connaissance du système, l'hétérogénéité technologique des solutions disponibles et enfin la complexité et la variété des tâches de maintenance. Pour chacun de ces verrous nous proposerons nos contributions.

Ainsi, le chapitre 2 introduira notre architecture logicielle basée sur la programmation par composants. Elle permet de s'affranchir du verrou de l'hétérogénéité des solutions technologiques et des algorithmes de traitement spécifiques à ces dernières. Dans notre architecture, une application est décrite par un ensemble de composants mis en communication entre eux.

Sur cette architecture est bâti un système de localisation par la vision qui sera présenté au chapitre 3. Ce dernier, que nous avons automatisé au maximum, repose sur l'utilisation de cibles codées qui permettent de calculer en temps réel la pose de la caméra par rapport aux objets suivis. Il présente également l'avantage de pouvoir lier sémantiquement ce qui est vu par la caméra aux connaissances détenues par le système sur le monde réel. Ce sera pour nous l'occasion d'introduire de nouveaux algorithmes spécifiques dédiés au problème posé. Ce système présente donc l'avantage de résoudre deux verrous : le problème de sa localisation temps réel et le lien sémantique entre ce qui est vu et sa base de connaissances. Nous étudierons également les méthodes spécifiques introduites afin d'estimer leurs performances par rapport aux algorithmes et aux systèmes connus.

Une application de réalité augmentée ne serait pas complète sans un système de gestion des augmentations (dans notre cas il s'agira principalement des augmentations visuelles). Pour ce faire, des exemples de procédure de maintenance ont été étudiés afin de dégager les structures inhérentes à ces dernières que nous avons modélisées. Ceci nous permettra de nous affranchir des problèmes liés à la complexité et à la variété des tâches de maintenance. Ce système de gestion des augmentations a été employé dans le cadre du démonstrateur AMRA (Assistance à la Maintenance en Réalité Augmentée) développé en collaboration avec le CEA et Alstom, le demandeur du projet. Ce prototype a fait l'objet d'un projet RNTL de 2001 à 2004. Les composants de ce système seront détaillés au chapitre 4.

Enfin, dans le chapitre 5, nous verrons comment élargir le système proposé à la réalité augmentée en vision directe (dans ce cas le monde réel est directement vu par l'opérateur par le biais de casques de vision semi-transparents), cette dernière présentant des verrous spécifiques quant à la localisation spatiale mais également temporelle du système. Nous montrerons quel est l'ensemble de capteurs à employer dans ce cas précis. Nous exposerons les résultats obtenus en simulation en introduisant une nouvelle méthode de prédiction basée sur les filtres stochastiques dits de "Monte-Carlo". De plus, nous introduirons une architecture de pipeline graphique combinant de manière innovante les techniques de prédiction du point de vue et les méthodes dites de "post-rendering", ce qui apporte des éléments de solution aux verrous technologiques apportés par les systèmes de réalité augmentée en vision directe.

Nous allons à présent nous attacher à définir plus précisément le domaine de recherche qu'est la réalité augmentée.

# Chapitre 1

# Réalité augmentée : Tour d'horizon

Dans un premier temps, il convient de définir le domaine de recherche qu'est la réalité augmentée (RA). Nous verrons ainsi quelles sont les définitions qui lui sont associées, les diverses fonctionnalités apportées par la réalité augmentée ainsi que la taxonomie technique qui lui est associée. Ceci nous permettra de faire ressortir les divers verrous technologiques et scientifiques auxquels est confrontée la RA. Enfin, nous examinerons quelques systèmes en particulier afin de dégager les principales composantes d'un système de RA sur lesquelles seront effectuées nos développements ultérieurs. Au passage nous aurons restreint notre domaine d'application à la maintenance industrielle. De plus, les augmentations pouvant être de plusieurs natures, nous mettrons essentiellement l'accent sur les augmentations de type visuel.

## 1.1 Définitions et taxonomie

### 1.1.1 Définitions

L'émergence de la réalité augmentée en tant que domaine de recherche coïncide avec la banalisation des dispositifs de visualisation tête-haute dans les laboratoires de recherche en réalité virtuelle vers le début des années 1990.

Par voie de conséquence, l'une des premières définitions données à la réalité augmentée était la suivante : *une forme de réalité virtuelle où le dispositif de visualisation tête-haute de l'opérateur est semi-transparent, ce qui permet une vision nette du monde réel.*

Par la suite, cette définition a évolué pour sortir de ce cadre très restreint. Dans l'article de Milgram, la réalité augmentée a pour but *d'augmenter la rétroaction naturelle de l'opérateur avec le monde réel à l'aide d'indices virtuels* [Milgram et al., 1994]. Cette définition permet de sortir du cadre de l'utilisation des seuls dispositifs de visualisation tête-haute.

La définition donnée par Azuma demeure plus restrictive car elle introduit des conditions supplémentaires : *un système de réalité augmentée complète le monde réel avec des objets virtuels (générés par ordinateur) de telle sorte qu'ils semblent coexister dans le même espace que le monde réel* [Azuma et al., 2001]. Toujours selon lui, les systèmes de réalité augmentée possèdent les propriétés suivantes :

1. *combiner des objets réels et virtuels dans un environnement réel*,
2. *être temps-réel et interactifs*,
3. *recaler (aligner) les objets réels et virtuels*.

Nous verrons ultérieurement que le troisième point n'a pas nécessairement besoin d'être respecté pour être en présence d'un système de réalité augmentée. Les propriétés 1) et 3) pourraient être reformulées en termes d'associations entre le réel et le virtuel.

En définitive, la définition de Fuchs et Moreau [Fuchs et Moreau, 2004] semble la plus équilibrée : *La réalité augmentée regroupe l'ensemble des techniques permettant d'associer un monde réel avec un monde virtuel, spécialement en utilisant l'intégration d'Images Réelles (IR) avec des Entités Virtuelles (EV) : images de synthèse, objets virtuels, textes, symboles, schémas, graphiques, etc. D'autres types d'association entre mondes réels et virtuels sont possibles par le son ou par le retour d'effort.*

Au vu des nombreux critères qui constituent un système de réalité augmentée, il convient d'établir une classification des divers systèmes qui existent. Dans la littérature, nous trouvons essentiellement deux systèmes de classification :

- une taxonomie fonctionnelle, qui classifie les systèmes suivant la manière dont l'association entre le réel et le virtuel est traitée,
- une taxonomie technique, qui porte sur les différentes techniques d'affichage ou de vision du monde réel.

## 1.1.2 Taxonomie fonctionnelle

Cette classification, empruntée à Fuchs et Moreau dresse une liste de catégories fonctionnelles basée sur la nature des associations entre le réel et le virtuel, indépendamment du dispositif employé pour faire l'association [Fuchs et Moreau, 2004]. Cette taxonomie essentiellement concentrée sur les augmentations de type visuel est présentée selon un ordre croissant de la contribution des augmentations visuelles dans la scène réelle.

### 1.1.2.1 Fonctionnalité "réalité documentée" et "virtualité documentée"

Il s'agit de la fonctionnalité la plus simple à mettre en oeuvre : Le réel et le virtuel sont dans deux cadres d'affichage séparés, mais ayant un rapport d'identification entre eux. Les EV peuvent alors être simplement du texte décrivant l'objet visible (figure 1.1) ou le processus en cours. (Cette fonctionnalité ne respecte pas nécessairement la troisième propriété des systèmes de réalité augmentée suivant la définition d'Azuma).

De même, le cas de figure inverse peut se produire : un processus modélisé de manière virtuelle peut, à certains moments montrer des images réelles du processus en cours. Il s'agit alors de "virtualité documentée".

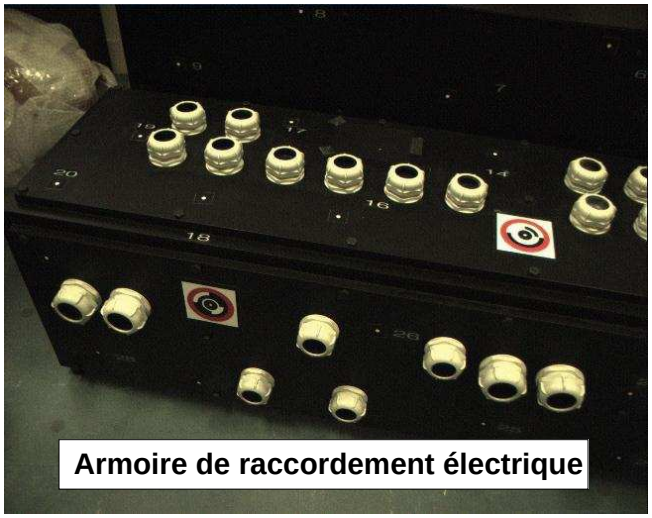



FIG. 1.1 – Réalité documentée

#### 1.1.2.2 Fonctionnalité “Réalité à compréhension ou à visibilité augmentée”

Dans le cadre de cette fonctionnalité, le réel et le virtuel sont dans un même espace d’affichage ce qui nécessite un recalage du virtuel sur le réel.

La “réalité à compréhension augmentée” (figure 1.2) consiste à incruster des informations sémantiques passives telle que la fonction, la nomenclature ou la référence des objets réels sur les images de la scène réelle. Il peut s’agir de titres, de légendes, ou de symboles qui ne sont pas nécessairement la contrepartie exacte des objets réels.

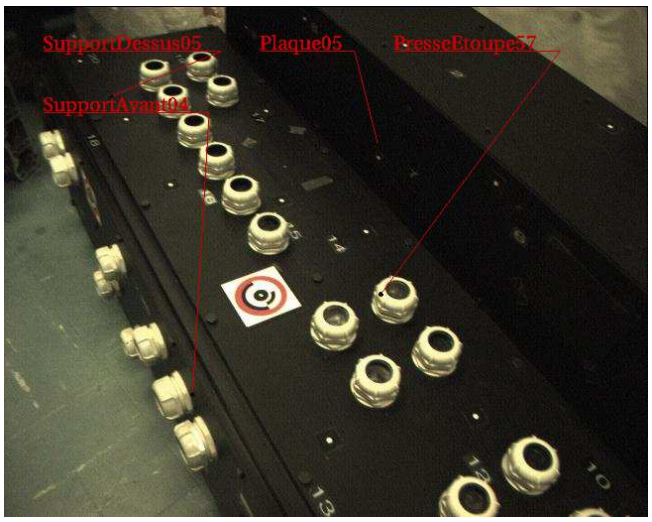



FIG. 1.2 – Réalité à compréhension augmentée

La “réalité à visibilité augmentée” (figure 1.3) consiste à sur-ligner les objets réels afin de mieux les voir. Ceci est généralement fait à l’aide d’EV qui sont des modèles en “fil de fer” ou en transparence des objets réels. Bien sûr, la mise en correspondance géométrique du réel et du virtuel s’avère là aussi cruciale lors de l’exploitation de cette fonctionnalité. La visibilité peut-être améliorée en permettant à l’opérateur de visualiser une partie de la scène réelle qui est cachée, comme par exemple l’intérieur d’un dispositif. On parlera alors de “vision en transparence”.

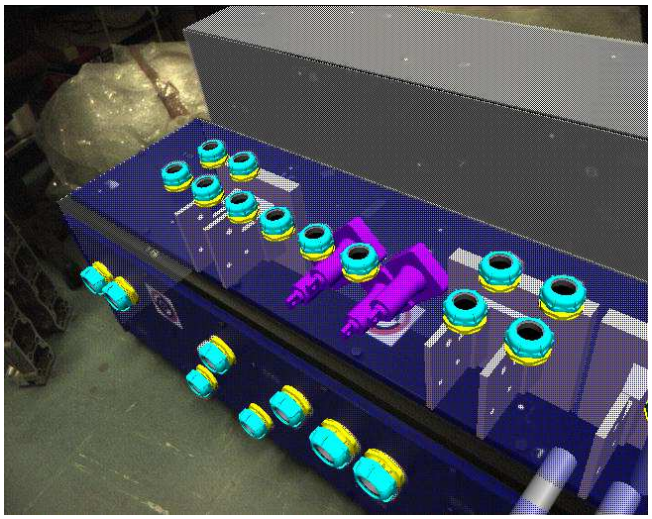


FIG. 1.3 – Réalité à visibilité augmentée

#### 1.1.2.3 Fonctionnalité "Association du réel et du virtuel"

De nouveaux objets virtuels sont ajoutés à la scène réelle dans le cadre de cette fonctionnalité. Suivant le niveau d'intégration des objets virtuels dans la scène réelle, nous nous retrouvons confrontés à deux cas :

- les objets virtuels sont incrustés ("overlay") devant les objets réels (figure 1.4), sans aucune occultation des objets virtuels par les objets réels. Il s'agit alors d'une association par superposition,
- les objets virtuels sont intégrés avec les objets réels qui peuvent alors occulter les EV. Il s'agit d'une association tridimensionnelle car elle nécessite de gérer les occultations réel/virtuel et inversement.

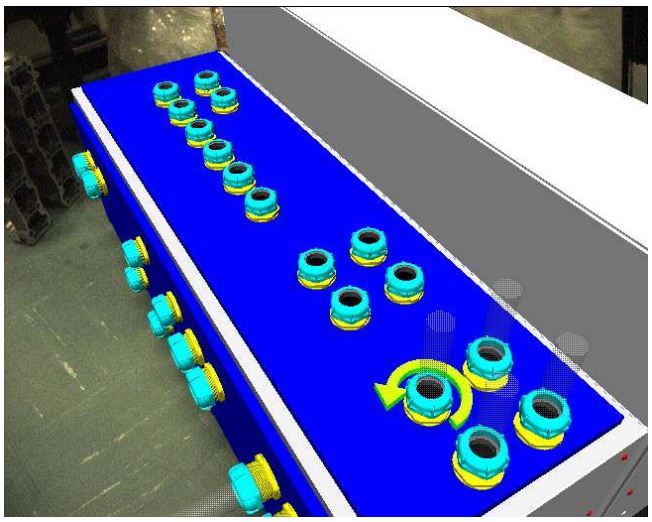


FIG. 1.4 – Association du réel et du virtuel

#### 1.1.2.4 Fonctionnalité "Association comportementale du réel et du virtuel"

Les EV ne se contentent pas seulement de simuler l'aspect d'objets réels mais ils possèdent également des propriétés physiques (masse, rugosité, élasticité, fluidité, etc.) utilisées pour les faire interagir de manière réaliste avec le monde réel.

#### 1.1.2.5 Fonctionnalité "Substitution du réel par le virtuel" ou "Réalité virtualisée"

Les propriétés de l'ensemble de la scène réelle sont connues et modélisées, dès lors, il est possible de substituer à la scène réelle une scène virtuelle équivalente. Ceci permet de passer du réel au virtuel et inversement pour faciliter la compréhension du monde réel. La scène virtuelle permet également de présenter la scène réelle sous un nouveau point de vue qui permet de mieux appréhender le déroulement du processus réel. D'autre part, si la scène réelle est instrumentée par un ensemble de capteurs (par exemple sur les articulations d'un robot), il est possible d'animer la scène virtuelle en fonction de ces informations.

Comme nous pouvons le remarquer, même si cette classification n'est pas totalement fermée aux augmentations d'un autre type que des augmentations visuelles, ce sont bien sur ces dernières que la discrimination des fonctionnalités est effectuée. Dans le même ordre d'idée, la taxonomie technique qui suit se concentre essentiellement sur l'augmentation visuelle de la réalité.

### 1.1.3 Taxonomie technique

Cette taxonomie a été introduite par Milgram [Milgram et al., 1994]. Elle permet d'introduire un certain nombre de notions attachées à la réalité augmentée. Elle commence par définir sept classes de systèmes de réalité augmentée, classifiés selon le type du dispositif d'affichage et la proportion respective du réel par rapport au virtuel.

Il convient tout d'abord d'établir une liste des dispositifs d'affichage possibles :

– les moniteurs vidéos,
– les casques de réalité virtuelle (casques de RV) qui se déclinent également en plusieurs types,

1. les casques de réalité virtuelle classiques (occlusifs car la réalité n'est pas perçue directement au travers de ces derniers),
2. les casques de réalité virtuelle avec retour vidéo (en anglais "Video see-through"), sur lesquels sont montés deux caméras qui fournissent la vision du monde réel à l'opérateur,
3. les casques de réalité virtuelle semi-transparents (en anglais "Optical see-through"), qui permettent de voir directement le monde réel et de manière simultanée les augmentations virtuelles par le biais d'un système optique composé de prismes ou de miroirs semi-réfléchissants.

La vision sera directe ( Réalité augmentée en vision directe - RAVD ) si l'opérateur voit directement les objets ou le monde réel. Elle sera indirecte ( Réalité augmentée en vision indirecte - RAVI ) si ce dernier la perçoit par le biais d'une caméra par exemple.

Le repère de travail du système est égocentrique s'il s'avère critique de localiser l'opérateur ou à tout le moins la direction de son regard. Dans le cas contraire, le repère sera exocentrique.

Enfin, la vision des objets réels et virtuels peut s'effectuer à l'échelle 1 :1 ou non. En l'occurrence, ce sera toujours le cas pour la réalité augmentée en vision directe (RAVD). Le tableau 1.1 présente les sept classes de dispositifs identifiées par Milgram.

| **Type de système Périphérique, Augmentation** | Monde ( réel/ virtuel ) | Vision ( directe/ indirecte ) | Repère ( Exocentrique/ Egocentrique ) | Vision à l'échelle 1 :1 ? |
|---|---|---|---|---|
| 1 - Moniteur vidéo, incrustations virtuelles | réel | indirecte | exocentrique | non |
| 2 - Casque de RV, incrustations virtuelles | réel | indirecte | égocentrique | non |
| 3 - Casque semi-transparent de RV, incrustations virtuelles | réel | directe | égocentrique | oui |
| 4 - Casque de RV avec retour vidéo, incrustations virtuelles | réel | indirecte | égocentrique | oui |
| 5 - Moniteur vidéo, monde virtuel avec incrustations vidéo | virtuel | indirecte | exocentrique | non |
| 6 - Casque de RV, monde virtuel avec incrustations vidéo | virtuel | indirecte | égocentrique | non |
| 7 - Monde virtuel avec interactions avec des objets réels | virtuel | direct, indirect | égocentrique | oui |

TAB. 1.1 – Classes des dispositifs d'affichage

Il est à noter que certains de ces systèmes sont plus proches des systèmes de RV purs que des systèmes de réalité augmentée. Cette taxonomie établit et classifie ces systèmes sur une échelle continue entre la réalité augmentée et la réalité virtuelle, échelle représentée à la figure 1.5 page suivante, qui est appelé réalité mixée.

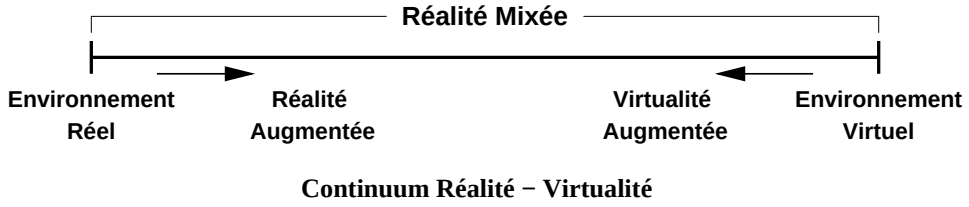



FIG. 1.5 – Représentation simplifié du continuum entre réalité et virtualité

Les systèmes sont ensuite classifiés suivant trois critères de discrimination :
– le degré de connaissance du monde,
– le degré de fidélité de représentation du monde,
– le degré d'immersion de l'opérateur par rapport aux informations affichées.

A chacun de ces critères correspond également une échelle de gradation établissant ainsi un système de classification à trois axes.

#### 1.1.3.1 Degré de connaissance du monde

Ce degré de connaissance traduit également le degré de modélisation du monde. Comme la figure 1.6 le montre, à une extrémité de l'axe se trouve un monde non modélisé (ce qui arrive dans le cas d'une téléopération à l'aide d'un simple retour vidéo) alors qu'à l'autre extrémité le monde est entièrement modélisé (ce qui est le cas des applications de réalité virtuelle).

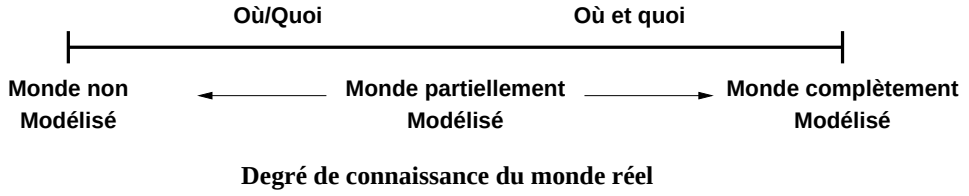



FIG. 1.6 – Degré de connaissance/modélisation du monde réel

Notre figure montre plusieurs cas qui peuvent se présenter sous la forme des mots interrogatifs "Où" et "Quoi". "Où" signifie que l'on a localisé le point de vue par rapport au monde réel, "Quoi" signifie que l'on connaît ce que l'on regarde, ce qui introduit un lien sémantique entre ce qui est regardé et ce qui va pouvoir être affiché. Nous notons qu'il est possible, dans le cas où le degré de connaissance du monde n'est pas très étendu, de ne pas pouvoir répondre au deux interrogations simultanément. Dans le cas où la réponse à la question "Où" n'est pas possible, le recalage du virtuel sur le réel ne sera pas possible, ce qui, une fois encore, n'est pas compatible avec la troisième caractéristique des systèmes de réalité augmentée énoncée par Azuma.

#### 1.1.3.2 Degré de fidélité de représentation du monde

Le degré de fidélité de représentation du monde va être tributaire de deux facteurs complémentaires :
– la modalité d'affichage : plus elle est en adéquation avec la vision humaine, plus le degré de fidélité augmente. Par exemple, la vision stéréoscopique sera plus en adéquation avec la vision humaine que la vision monoscopique,
– la fidélité du rendu : suivant que ce dernier soit réaliste ou non, les entités virtuelles semblent réelles ou non.

Ceci amène à la classification représentée à la figure 1.7 page suivante. Nous voyons comment chaque facteur contribue au degré général de fidélité de représentation du monde.

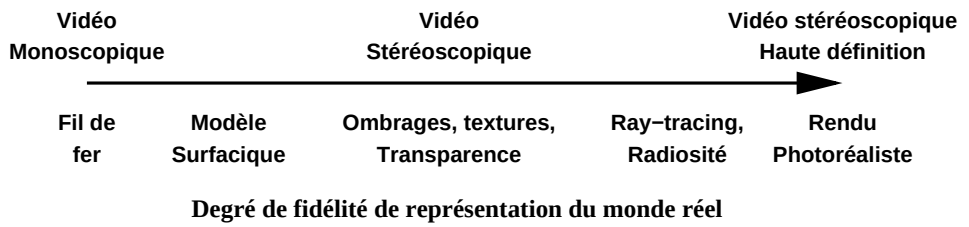



FIG. 1.7 – Degré de fidélité de représentation du monde suivant la technologie d'affichage et la qualité du rendu

#### 1.1.3.3 Degré d'immersion de l'opérateur

Le degré d'immersion de l'opérateur dans la scène et le degré de fidélité de représentation du monde ne sont pas totalement décorrélés. En effet, le degré d'immersion va dépendre également de la technologie employée pour réaliser l'affichage. Ainsi, sur cet axe, nous allons retrouver la modalité d'affichage qui sera cette fois mise en balance avec la technologie du dispositif d'affichage comme nous pouvons le constater dans la figure 1.8.

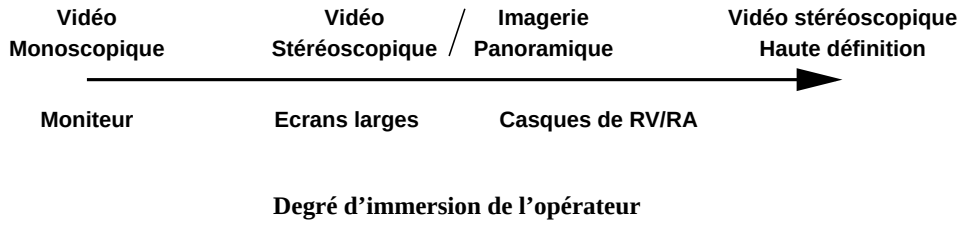



FIG. 1.8 – Degré d'immersion en fonction des technologies et modalités d'affichage

Après avoir présenté les définitions et terminologies et la taxonomie liée aux réalités virtuelles et augmentée, nous allons présenter la problématique générale de la réalité augmentée.

## 1.2 Verrous associés à la réalité augmentée

### 1.2.1 L'hétérogénéité des solutions technologiques

Si nous reprenons la définition donnée par Azuma (énoncée en page 17), nous devons combiner des objets réels et virtuels dans un environnement réel. Dans le cadre des augmentations visuelles, ceci implique de pouvoir visualiser les augmentations visuelles mixées avec la réalité. L'un des premiers point est de choisir le dispositif de visualisation en fonction de la tâche à accomplir. Les dispositifs d'affichage ont déjà été présentés dans le tableau 1.1 page 21 suivant les cas d'utilisation, chaque classe de dispositifs introduisant des contraintes propres.

De même, lorsque l'on veut effectuer le recalage des objets virtuels sur les objets réels, il est nécessaire de localiser le point de vue employé (qui est celui de la caméra dans le cas de la vision indirecte et celui de l'opérateur en vision directe). Pour ce faire, de nombreuses solutions technologiques peuvent être employées :

- les capteurs de position et d'orientation magnétique (type Polhemus ou Flock of Bird),
- les capteurs de position et d'orientation à ultrasons (type IS-600 de chez Intersense),
- les capteurs de position et d'orientation optique utilisant les infra-rouges (type ART),
- les caméras simples,
- les centrales inertielles composées de gyroscopes et d'accéléromètres,
- etc...

| Type de capteur | Transposition mesure → localisation capteur |
|---|---|
| Magnétique | directe |
| Ultrasons | directe |
| Optique type ART | directe |
| Caméras | traitement d'image + estimation de la pose |
| Gyroscopes | intégration simple |
| Accéléromètres | intégration double |

TAB. 1.2 – Transposition entre la mesure fournie par le capteur et la localisation de ce dernier en fonction de son type

Ici également, l'ensemble des solutions technologiques est très hétérogène [Welch et Foxlin, 2002]. Une fois les solutions technologiques choisies, ces dernières vont influencer la manière dont nous allons envisager le recalage des objets virtuels sur les objets réels. Or pour effectuer le recalage, nous avons besoin de localiser le point de vue du système. Comme nous le verrons, une architecture modulaire est souvent employée pour donner aux systèmes de RA une certaine flexibilité par rapport à l'hétérogénéité des solutions technologiques employées (chapitre 2).

### 1.2.2 Problème de la localisation temps réel du point de vue du système

Chaque type de capteur va nécessiter une phase de calibration (c'est à dire l'identification des paramètres caractérisants un modèle mathématique décrivant leur fonctionnement interne). A partir de cette phase, il est possible d'utiliser ce modèle et d'expliciter les relations entre ce dernier et la localisation spatiale du capteur. Cette étape est tributaire du type de capteur utilisé comme nous pouvons le voir dans le tableau 1.2. A cette transposition, il faut éventuellement ajouter le filtrage des données pour supprimer le bruit de mesure et les formules de changement de repère entre le repère local lié au capteur et le repère du point de vue. Ces problèmes sont détaillés, notamment pour le cas de l'utilisation d'une caméra en tant que capteur de localisation au chapitre 3.

Dans le cas particulier de la réalité augmentée en vision directe, il faudra également tenir compte de l'asynchronisme qui existe entre la vision du monde réel directement effectuée par l'opérateur et le système de réalité augmentée proprement dit. Ceci implique un recalage dynamique qui doit tenir compte des mouvements de l'opérateur. Nous détaillerons cette problématique et les solutions que nous pouvons y apporter au chapitre 5.

### 1.2.3 L'interaction entre le réel et le virtuel

L'interaction entre le réel et le virtuel regroupe plusieurs classes de problématiques : le lien sémantique entre objets réels et virtuels, les périphériques d'interaction et le lien entre les augmentations visuelles et la réalité. Le lien entre la réalité et les augmentations visuelles a été présenté en début de chapitre dans l'exposition de la taxonomie fonctionnelle (voir section 1.1.2 page 18) et sera développé plus précisément au chapitre 4.

Pour pouvoir interagir entre le réel et le virtuel, il faut établir un lien sémantique entre les objets réels et les entités virtuelles. Ceci peut s'effectuer de manière implicite ou explicite. L'association est implicite lorsque l'identification est faite à partir d'un objet lié au dispositif réel que l'on souhaite identifier. C'est le cas si l'on utilise un capteur magnétique accolé au dispositif ou encore une cible codée collée sur ce dernier. L'association sera explicite si le dispositif est reconnu directement, c'est à dire sans passer par la reconnaissance d'un intermédiaire en utilisant par exemple des opérateurs de traitement d'image dédiés à la reconnaissance de formes.

Lorsque le lien sémantique est créé, il est possible d'interagir par le biais d'un support physique qui peut-être un périphérique informatique classique tel que clavier, souris ou joystick ou des périphé-

riques plus spécifiques tels que les gants de réalité virtuelle permettant de mesurer les mouvements des doigts de la main ou les interfaces dites “tangibles” qui peuvent employer des cibles codées collées sur un support rigide ou au contraire des objets simples du monde réel utilisés en temps que métaphore d’objets virtuels. Ici encore, le problème de l’hétérogénéité des périphériques d’interaction est soulevée.

### 1.2.4 La contrainte du temps réel

Nous l’avons déjà vu à propos du problème de la localisation, les systèmes de réalité augmentée se doivent d’être temps réels. Ceci implique une optimisation de la chaîne de traitement des données, de l’acquisition de ces dernières par les capteurs jusqu’à l’affichage final de la scène virtuelle sur l’écran. Sachant que la précision de certains algorithmes est dépendante du nombre d’itérations à fournir, il devient alors critique de déterminer le bon compromis entre précision et stabilité d’une part et la vitesse d’exécution d’autre part. Cette contrainte intervient tout au long de la constitution des différents éléments du système de réalité augmentée.

Nous allons à présent étudier quelques systèmes de réalité augmentée pour voir quelles sont les solutions apportées à ces divers problèmes.

## 1.3 Étude de quelques systèmes de RA

Il n’est évidemment pas possible de présenter tous les systèmes de RA existants afin de déterminer dans chaque cas quelle a été la solution employée. Au contraire, nous n’allons étudier que quelques systèmes, tout d’abord le plus vieux d’entre tous qui pose les fondements des systèmes de réalité augmentée puis ensuite nous nous intéresserons à quelques systèmes en particulier pour illustrer dans quels cadre la réalité augmentée est employée. Puis nous nous intéresserons aux applications dans le domaine industriel de la maintenance et de l’assemblage/démontage qui est un des thèmes de cette thèse. Enfin, nous établirons une comparaison entre les divers systèmes cités.

### 1.3.1 Les projets en réalité augmentée génériques

#### Le système de Sutherland

L’un des premiers systèmes de réalité augmentée a vu le jour en 1968, il s’agit de l’oeuvre de Sutherland [Sutherland, 1968]. Souvent considéré comme étant l’un des travaux fondateurs de la réalité virtuelle, il s’agit aussi, sous plusieurs aspects d’un système de réalité augmentée. En effet, ce système est constitué :

- d’un casque de réalité virtuel semi-transparent à vision stéréoscopique. L’affichage est assuré par des tubes cathodiques miniaturisés,
- d’une accélération graphique matérielle modifiée pour obtenir des taux de rafraîchissement de l’image de l’ordre de 30 Hz,
- d’un système de suivi de la tête composé d’un bras mécanique lié au casque.

Ce sont les composants matériels d’un système de réalité augmentée. Même si l’application présentée était la visualisation de molécules en trois-dimensions, l’auteur précise que “*les miroirs semi-transparents des prismes au travers desquels l’utilisateur regarde lui permettent de voir simultanément les images générées par les tubes cathodiques et les objets de la pièces simultanément. Les objets affichés peuvent donc être détachés du monde physique ou coïncider avec des cartes, des bureaux, des murs ou les touches d’une machine à écrire.*”. Si, au vu des standards actuels, ce système emploie des technologies encombrantes, il n’en demeure pas moins un système complet renfermant les constituants essentiels d’un système de RA.

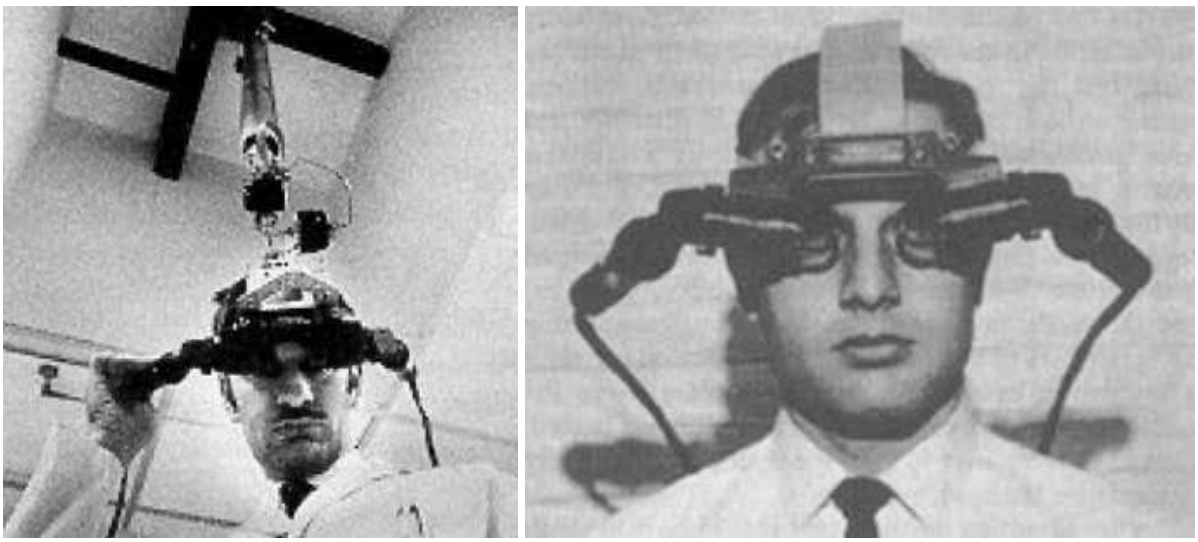


FIG. 1.9 – Le dispositif de Sutherland en 1968 comportant un casque de réalité virtuelle semi-transparent et un bras mécanique pour le suivi de la tête

**Le projet AR$^2$ Hockey**

Le projet AR$^2$ Hockey [Ohshira et al., 1998] présenté en 1998 par le Mixed Reality System Laboratory au Japon est une application de la RA dans le domaine ludique. Il s'agit d'un jeu de hockey sur table dans lequel le palet est remplacé par un palet virtuel que les joueurs se renvoient. L'équipement utilisé pour la manipulation est constitué de lunettes vidéos semi-transparentes, d'un capteur magnétique d'orientation et de position de type Polhemus pour suivre les mouvements de la tête de l'utilisateur et d'une caméra par paire de lunettes.

Trois types d'erreurs ont été identifiées pour effectuer le recalage :
- les erreurs statiques ou erreur de positionnement.
- les erreurs dynamiques ou erreurs du à un décalage dans le temps.
- les erreurs de rendu dues à une différence de qualité entre les images réelles et virtuelles.

Le recalage s'effectue sur le point de vue de la caméra et non pas sur le point de vue du joueur, l'hypothèse de travail étant que l'objet est suffisamment loin pour que le décalage ne se remarque pas. Pour la prédiction des mouvements de la tête de l'utilisateur, un algorithme du second ordre est utilisé :

$$\hat{p} = p_t + v_t \Delta t + \frac{1}{2} a_t \Delta t^2$$

$\hat{p}$ est la position prédite.
$p_t$ désigne le dernier enregistrement de position.
$v_t$ désigne la vitesse et $a_t$ désigne l'accélération.
$\Delta t$ désigne le temps entre la mesure de $p_t$ et l'affichage de l'image.

Comme les mesures effectuées ne comportent que la position, la vitesse et l'accélération sont calculées à l'aide des positions précédentes. Le système retient donc à chaque fois trois positions.

#### 1.3.1.1 Le projet CAMELOT

En 2000 fut présenté le projet CAMELOT [Broll et al., 2000] de l'Institut d'Informatique Appliquée qui est un département du Centre National Germanique de Recherche en Informatique. La réalité augmentée est utilisée pour créer une table de travail virtuelle. Les buts de ce projet étaient les suivants :
- augmenter l'espace de travail.
- cet espace est utilisé par plusieurs personnes qui travaillent en collaboration.

– l'interaction avec les objets virtuels doit être intuitive.
– l'application doit être capable de fonctionner partout.

Pour chaque participant, le matériel utilisé est une paire de vidéo-lunettes semi-transparentes Sony LDI-100 de résolution 800 x 600, un capteur inertiel Intersense IS 600 et deux caméras pour suivre le déplacement des objets réels.

La manipulation des objets virtuels se fait en bougeant des objets réels qui leurs sont associés (Par exemple, une tasse peut représenter un bâtiment). Cette association entre un objet réel et son pendant virtuel est appelé une unité d'interaction et illustre le principe des interfaces tangibles.

Ce projet a essentiellement évalué le matériel utilisé. Il en ressort que :

– Le champ de vision de 28° seulement perturbe la vue de l'utilisateur (par comparaison, la portion de l'espace vue en 3 dimensions par l'homme est d'approximativement 120°).
– Les lunettes semi-transparentes ne cachent pas réellement les objets réels quand on leur superpose des objets virtuels car, plus les objets virtuels sont sombres, plus ils apparaissent transparents.
– La précision des capteurs utilisés n'est pas suffisante pour ce type d'application selon les personnes qui l'ont programmée.

#### 1.3.1.2 La manipulation du laboratoire iMAGIS

Plus récemment, en 2001, Grasset et Gascuel ont présenté leurs travaux sur un environnement de réalité augmentée collaboratif. Cette manipulation, réalisée au laboratoire iMAGIS, met surtout l'accent sur la manipulation d'objets réels et virtuels [Grasset et Gascuel, 2001].

Le matériel utilisé pour la manipulation est une paire de lunettes semi-transparentes I-glasses. L'acquisition des positions et orientations se faisait avec un capteur magnétique de type Flock of Birds. D'autre part, afin de réduire les temps de latence et les bruits sur les mesures du capteur, un filtre de Kalman est employé.

Des efforts ont été portés sur la programmation de l'application afin de tenir compte de la diversité des périphériques d'entrée (souris, clavier, capteur de position et d'orientation...) et des périphériques de sortie (lunettes, écran). Le système développé est modulaire et aisément configurable à l'aide de fichiers XML. Chaque utilisateur possède ainsi un fichier qui définit ses préférences, le contexte d'utilisation de l'application ainsi que les paramètres de calibration des lunettes. Ceci permet d'apporter une certaine flexibilité face à l'hétérogénéité des capteurs employés.

### 1.3.2 Les projets de réalité augmentée dans le cadre de la maintenance et l'exécution de tâches

Ces projets peuvent se décomposer suivant deux grands axes avec une rupture chronologique : Avant 1998, les projets de ce type étaient de taille réduite et étaient surtout des manipulations de laboratoire. Après 1998, les systèmes employés ont une finalité tournée vers une application industrielle. Ce sont des projets de plus grande envergure qui associent souvent des consortiums composés d'industriels et de laboratoires de recherche.

#### 1.3.2.1 Avant 1998 : Les manipulations effectuées en laboratoire

Au début des années 1990, Steven Feiner travaillait sur le projet KARMA [Feiner et al., 1993] [Feiner et al., url] (Knowledge-based Augmented Reality for Maintenance Assistance). Expérimenté au Computer Science Department de l'université de Californie du Sud, ce projet avait pour but de guider un opérateur dans le cadre de la réalisation de tâches simples de maintenance d'une imprimante laser.

Le banc de test est constitué d'une imprimante, de trois capteurs à ultrasons (placés sur le couvercle de l'imprimante, le bac à papier et un casque lié à la tête de l'utilisateur), d'un casque de réalité

virtuelle semi-transparent développé par ce laboratoire et de plusieurs machines pour faire tourner l'application (une machine se chargeait de gérer l'affichage, une autre était employée pour l'acquisition des données venant des différents capteurs et d'autres machines se chargeaient du coeur de l'application).

FIG. 1.10 – Le banc de test utilisé pour KARMA

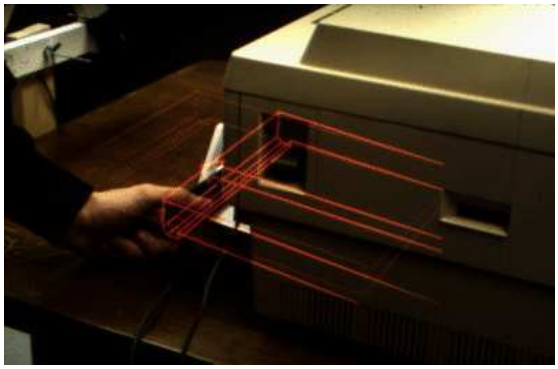


FIG. 1.11 – La réalité augmentée fournie par KARMA

Les efforts fournis dans ce projet ont plutôt été portés sur les modalités de représentation de la tâche à effectuer. Pour cela, un moteur graphique (IBIS pour Intent Based Illustration System) a été développé. Ce dernier se basait sur un système de règles qui définissent la manière d'effectuer une tâche ou d'accomplir un but.

Ces règles sont de deux types et correspondent à une séparation haut niveau / bas niveau :

– les règles générales spécifient les effets visuels à accomplir pour aider à atteindre son but.
– les règles de style donnent les différents moyens par lesquels les effets visuels sont exécutés.

Parallèlement, chacune de ces règles était divisée en méthodes et évaluateurs, les premières exposant les actions à effectuer, les dernières vérifiant si ces actions ont été effectuées ou non.

En 1996, les chercheurs du même laboratoire ont lancé deux projets concernant l'utilisation de la réalité augmentée dans le cadre du montage, de la maintenance et de l'inspection des bâtiments [Webster et al., 1996].

Le premier, appelé "architectural anatomy", leur permettait de voir les structures internes d'un bâtiment, telles que les colonnes de soutien ou les circuits électriques. Le casque de réalité virtuelle employé pour KARMA a été réutilisé. Les mouvements de la tête de l'opérateur étaient suivis par un capteur à ultrasons. L'intérêt de ce projet réside dans l'interaction avec les objets virtuels qui sont sélectionnables à la souris. Des informations supplémentaires étaient affichées à la sélection telles que les dimensions de la pièce observée ou les contraintes mécaniques qui s'exercent sur elle. Ces dernières étaient obtenues grâce au couplage de l'application avec des logiciels de calcul d'effort utilisés par les architectes.

Le deuxième projet visait à accélérer le montage des structures en aluminium et à éviter les erreurs qui peuvent entraîner l'effondrement de la structure. (Les différentes pièces utilisées étaient de formes peu différentes, ce qui est source de confusions). Le banc de test était constitué d'une structure aluminium en forme de diamant, d'une paire de lunettes semi-transparentes de marque Virtual I/O équipées d'un capteur d'orientation. Les mouvement de la tête et de la main droite étaient suivis par un radar optique de marque Origin Instruments. L'utilisateur était également équipé d'un lecteur de code barre pour différencier les pièces avant le montage.

Pour chaque pièce à monter, le système de réalité augmentée suivait ces quatre étapes :

1. Une bande audio donnait l'ordre à l'opérateur de se diriger à l'endroit où sont entreposées les pièces détachées et lui dit quelle est la pièce à prendre.


(a) La structure de test (b) Une vue en réalité augmentée

FIG. 1.12 – le projet de montage de structures en aluminium

2. La lecture du code barre collé sur la pièce permettait au système de confirmer que celle-ci est la bonne ou non.
3. L'ordre était donné à l'opérateur d'aller sur le site de montage. Une fois sur le site de montage :
   – Une vue de l'endroit où la pièce doit être montée était montrée à l'opérateur.
   – Un texte descriptif accompagnait cet affichage.
   – Des instructions verbales indiquait comment monter la pièce.
4. Le système vérifie que la pièce était bien montée en demandant à l'opérateur de placer sa main sur le code barre. Les mouvements de la main étant suivis par un capteur, le système pouvait ainsi déterminer si la pièce est bien montée.

#### 1.3.2.2 1998 : une année de transition

Cette année a marqué la fin des manipulations de laboratoire et le début de l'intérêt des industriels pour la réalité augmentée dans le cadre de l'aide au montage et à la maintenance.

Au Fraunhofer Institut (IGD), Reiners a travaillé sur la tâche d'insertion d'un verrou de portière d'une automobile [Reiners et al., 1998]. En effet, cette tâche nécessite des gestes spécifiques. La RA permet de mettre en situation un ouvrier sur le matériel réel afin de lui apprendre les gestes à effectuer. Le banc de test utilisait des lunettes immersives complétées de deux caméras qui lui restituent leur vision du monde réel. La portière utilisée était munie de cibles pour que le traitement des images fournisse la position de la tête de l'utilisateur par rapport au banc de test. Le système de réalité augmentée lui montre successivement la pièce à insérer, l'endroit où la pièce devait venir, la trajectoire d'insertion ainsi que le sens de vissage des divers écrous pour fixer la pièce. Le moteur graphique de rendu était développé à l'aide d'OpenSG (SG pour Scene Graph) qui est une sur-couche objet d'OpenGL et qui fut utilisé par la suite dans le projet Arvika (qui est décrit plus loin).

Dans le même temps, Boeing [Neunmann et Majoros, 1998] a utilisé la réalité augmentée dans le cadre du montage des faisceaux de câbles électriques dans les avions, ainsi que leurs connexions dans les armoires électriques. Le même prototype devait servir également à effectuer des travaux d'inspection et de maintenance dans ces mêmes armoires. Le dispositif employé était un casque de réalité virtuelle surmonté d'une caméra pour voir le monde réel. La scène est instrumentalisée à l'aide de cibles qui étaient localisées rapidement par traitement des images des caméras. Le système de

FIG. 1.13 – La manipulation de Reiners

réalité augmentée se contentait d'annoter les images avec des textes qui se modifient en même temps que l'environnement. Pour réaliser ceci, Boeing a développé son propre moteur d'affichage, basé sur OpenGL.

Un certain nombre d'études ergonomiques et psychophysiques ont été également effectuées sur ce projet. Elles ont permis de montrer que la réalité augmentée était d'une aide réelle dans ces tâches de maintenance car elle réduit le temps passé par l'opérateur à retrouver les éléments dont il a besoin pour accomplir une tâche. La preuve finale a été apportée lors du congrès IWAR'98 (International Workshop on Augmented Reality) par Curtis qui a montré qu'un opérateur ne connaissant rien à la tâche allait plus vite pour l'accomplir avec l'aide du système de réalité augmentée qu'un mainteneur chevronné [Curtis et al., 1998]. Ceci concernait la disposition des ensembles de câbles à disposer dans les armoires électriques des avions.

#### 1.3.2.3 Après 1998 : l'essor de la réalité augmentée dans l'industrie

A partir de 1999, nous assistons à la naissance de projets de grande envergure qui regroupent à la fois des laboratoires de recherche et des acteurs majeurs de l'industrie.

**Le projet ARVIKA et son successeur ARTESAS**

Ainsi, en 1999 a démarré ARVIKA [Arvika, url]. Ce projet de nature exploratoire qui a duré quatre années, visait à introduire la réalité augmentée dans le secteur industriel allemand. ARVIKA est dû à l'initiative du BMBF (Le ministère fédéral allemand de l'éducation et de la recherche). Ce projet avait pour coordinateur général l'industriel Siemens et regroupait un consortium allemand d'une vingtaine de partenaires parmi lesquels figuraient EADS, Ford, BMW, Framatome et l'IGD (Fraunhofer Institut). Le but était de développer les technologies utilisant la réalité augmentée pour les intégrer dans le cycle de vie complet des produits industriels : de la conception à la mise en service, en passant par la fabrication et la maintenance.

Une base technologique commune permettant d'utiliser la réalité augmentée a été développée et chacun des partenaires a conçu en parallèle sa propre application par dessus cette base. Le concept central du projet était en réalité le développement d'un serveur Web tourné vers la réalité augmentée, capable de s'adapter au contexte d'utilisation.

Peu après la fin du projet ARVIKA (en 2004), un nouveau projet du nom d'ARTESAS regroupant les mêmes acteurs a été démarré. Celui-ci, se basant sur l'expérience acquise au cours du projet précédent se concentre sur le suivi sans le recours de cibles codées adapté aux environnements industriels [Wuest et al., 2005] ainsi que sur les interfaces homme-machine plus ergonomiques et intuitives à employer qui sont les prochains enjeux des systèmes de réalité augmentée.

### Le projet Starmate

En mars 2000 a débuté le projet Starmate[Schwald et de Laval, 2003][Schwald et al., 2001] (d'une durée de trois ans). Il s'agissait ici d'un programme à l'échelle européenne où l'on retrouvait le Fraunhofer Institut (ZGDV) et EADS. Les autres partenaires étaient Thomson-CSF (devenu depuis Thales Optronique), Dune, Tecnatom et CS-SI. Le but de ce projet était d'assister un opérateur dans l'accomplissement des tâches de maintenance sur des systèmes mécaniques complexes. STARMATE devait s'acquitter de deux tâches complémentaires : aider l'utilisateur expérimenté dans des tâches de montage/démontage et de maintenance et former les opérateurs novices à ces mêmes tâches.

L'équipement que portait le mainteneur était le suivant :

- des lunettes semi-transparentes. Il s'agissait donc d'un système de réalité augmentée en vision directe.
- des écouteurs pour fournir des informations audio supplémentaires.
- un microphone connecté à un système de reconnaissance vocale.
- un système de pointage pour permettre à l'utilisateur de désigner les objets avec lesquels il travaille.
- un système de suivi de mouvements constitué de deux caméras pourvues de filtres à infrarouges.
- une unité de transmission des données entre l'utilisateur et le reste du système.

Les avancées dans ce projet concernaient pour la plupart l'architecture de la partie du système qui traite les informations. Une architecture modulaire a été définie qui se décompose de la manière suivante :

1. Un superviseur qui régule et coordonne les flux de données envoyés par les différents processus temps réel.
2. Un module d'interprétation des commandes qui traduit les commandes vocales et celles issues du système de pointage en commandes compréhensibles par le système.
3. Un module de restitution qui est le moteur graphique de l'application.
4. Un module de suivi des objets qui informe le système sur la position des divers éléments de la scène.
5. Un module de gestion de scénarios. La gestion d'un scénario est subdivisé en deux parties :
   - la procédure par défaut qui est constituée des différentes étapes du scénario. Chacune de ces étapes décrit les différentes augmentations requises par la manipulation.
   - les informations supplémentaires. Celles-ci décrivent les différentes augmentations qui peuvent être fournies à l'utilisateur en cas de demande de sa part.

### La maintenance de centrales nucléaires

En 2001, Dutoit [Dutoit et al., 2001] qui travaille alors à l'Institut pour l'informatique de la TUM (Technische Universität München) a exposé ses travaux concernant l'utilisation de la réalité augmentée dans le cadre de la maintenance des centrales nucléaires. Dans ce milieu, les procédures de maintenance sont codifiées de manière stricte et rigide. Chacune d'entre elle est détaillée dans les manuels de maintenance. Lorsqu'un opérateur travaille sur le site, son action est limitée au nombre des procédures décrites dans les manuels qu'il a emportés avec lui, ce qui n'est pas l'idéal dans le cas où il se trouve en face d'un problème inattendu.

Dans ce projet, la réalité augmentée est utilisée pour remplacer la volumineuse documentation que l'opérateur doit emporter. Le système embarqué est constitué de :

- un casque de réalité virtuelle.
- un écran tactile à cristaux liquides relié à l'unité de traitement citée plus bas.
- une interface vocale et un dispositif de pointage.
- une caméra pour permettre un retour visuel des dispositifs observé dans le casque.

– une unité de traitement embarquée qui communique via un réseau sans fil avec le reste du système.

Il est à noter que l'utilisateur peut retirer son casque à tout moment pour consulter l'écran à cristaux liquides. Les problèmes rencontrés ont été principalement le volume de la documentation à digitaliser et la relative autonomie dont devait disposer le système embarqué en raison des transmission impossibles dans certaines parties de la centrale. Le casque et l'écran ont des rôles complémentaires. Les informations sont affichées au fur et à mesure du déroulement des opérations dans le casque (dans le but de guider l'opérateur pas à pas). Sur l'écran, le mainteneur peut voir l'intégralité des opérations à effectuer lors de la procédure, celle qu'il est en train d'effectuer étant mise en évidence.

**Le projet AMRA**

Le projet AMRA [Didier et al., 2005b] est un projet RNTL (Réseau National des Technologies Logicielles) [RNTL, url] placé sous la tutelle du ministère de la Recherche. Ce dernier a commencé en 2002 et s'est achevé en mai 2004. Le projet était mené par un consortium de partenaires qui sont

– Alstom Transport, chef de file du projet et partenaire industriel apportant la problématique de travail,
– Le Commissariat à l'Énergie Atomique (CEA), plus particulièrement le Laboratoire d'Intégration des Systèmes et des Technologies (LIST), qui a développé un système de vision pour le projet, et qui a réalisé l'intégration finale du prototype,
– Le Laboratoire Systèmes Complexes (LSC) qui est notre laboratoire,
– Acti-CM, une start-up issue du CEA spécialisée dans la métrologie.

Le but était d'implémenter un système de Réalité Augmentée à usage mobile pour une utilisation en milieu industriel, et plus spécifiquement dans le domaine de la maintenance industrielle. Ce projet a poursuivi plusieurs objectifs :

– Fournir une aide contextuelle à des mainteneurs inexpérimentés, leur permettant d'être formés sur site,
– Apporter aux agents de maintenance une assistance permettant d'accéder sur leur poste de travail à des informations pertinentes (documentation de maintenance, modes opératoires, films de montage),
– Augmenter la disponibilité de l'information sur le lieu de maintenance en utilisant les techniques de Réalité Augmentée.

Le prototype AMRA est un système de réalité augmentée en vision indirecte constitué d'un tablette-PC (un ordinateur portable "allégé" pourvu d'un écran tactile ) pour la visualisation des informations, qui agit comme une fenêtre augmentée sur le monde réel, grâce à la caméra embarquée sur ce dernier. Ce type de système aborde plusieurs problématique : celle de l'informatique nomade (en anglais "mobile computing"), celle du recalage temps réel des entités virtuelles sur les images du monde réel, et enfin celle du développement d'une aide graphique contextuelle adaptée. Nous évoquerons plus longuement les développements que nous avons réalisé pour ce projet dans le chapitre 4.

### 1.3.3 Comparaison du matériel et des méthodes employées dans les différents projets

Nous allons comparer les divers systèmes présentés par rapport aux technologies employées pour la visualisation et le suivi du point de vue du système. De plus, nous verrons quels sont les types d'algorithme utilisés pour réaliser le filtrage des données et de quelle manière est assuré le recalage

du virtuel sur le réel.

Le tableau 1.3 nous montre le matériel employé dans les différents projets de réalité augmentée, notamment le type de capteur lié à la tête de l'observateur et les lunettes semi-transparentes utilisées.

| **Projet** | Type de capteur | | Dispositif d'affichage | Résolution |
|---|---|---|---|---|
| KARMA | ultrasons | 1 | casque monochrome monoscopique | 720 x 280 |
| Architectural anatomy | ultrasons | 1 | casque monochrome monoscopique | 720 x 280 |
| "Structures aluminium" | radar optique | 1 | i-glasses | 263 x 230 |
| "Verrou de portière" | caméra | 2 | casque de réalité virtuelle | |
| Boeing | caméra | 1 | casque de réalité virtuelle | |
| Starmate | caméras infrarouges<br>inertiel | 2<br>1 | lunettes semi-transparentes | |
| AR$^2$Hockey | caméra<br>magnétique | 1<br>1 | i-glasses | 263 x 230 |
| CAMELOT | caméra<br>inertiel | 2<br>1 | Sony LDI-100 | 800 x 600 |
| iMAGIS | magnétique | 1 | i-glasses | 263 x 230 |
| AMRA | caméra | 1 | Tablette-PC | 1024x768 |

TAB. 1.3 – Le matériel utilisé dans les différents projets

Le tableau 1.4 regroupe les informations relatives au type de recalage utilisé par chacune des applications de réalité augmentée. La colonne 'cibles' nous renseigne sur la manière dont le monde réel est équipé avec des cibles pour faciliter le calcul de la position et de l'orientation de la tête de l'observateur. Une technique courante est d'utiliser des cibles qui ont une forme, un motif ou une couleur spéciale en vue de traiter de façon rapide leur image prise par une caméra. La position et/ou l'orientation de ces cibles étant connue dans un repère donné associé au monde réel, cela apporte des informations supplémentaires sur la position et l'orientation de la caméra par rapport à ce même repère. Les capteurs utilisés peuvent fournir des mesures bruitées. Pour cela, des méthodes de filtrage peuvent être employées. C'est pourquoi, une colonne concernant le filtrage a été ajoutée.

Ce dernier tableau ne recèle que peu de renseignements sur le recalage dynamique dans ces projets. La documentation lue à ce sujet n'aborde pas le problème ou alors l'élude en déclarant que c'est un des problèmes difficiles à résoudre dans un système de réalité augmentée. D'autre part, les données provenant des mesures des différents capteurs sont rarement filtrées, ce qui peut affecter d'une manière importante des algorithmes de prédiction tels que celui du second ordre (Nous rappelons que le filtre de Kalman est non seulement une méthode de prédiction mais aussi de filtrage).

## 1.4 Conclusion

Ce chapitre est consacré à la définition de la réalité augmentée et à la présentation de sa taxonomie ainsi que des verrous scientifiques et technologiques correspondants d'une part. D'autre part, ce chapitre présente les principaux système de RA de la dernière décennie. L'étude de ces systèmes a permis d'en extraire les principales fonctionnalités à savoir l'architecture, le suivi temps réel, la prédiction du point de vue dans le cas de la vision directe ainsi que les modalités de présentation des

| **Projet** | Recalage | | Filtrage | Cibles |
|---|---|---|---|---|
| | statique | dynamique | | |
| KARMA | Partiel | Non | Non | Non |
| Architectural Anatomy | Partiel | Non | Non | Non |
| "Structures aluminium" | Oui | Non | Non | Non |
| "Verrou de portière" | Oui | | | Oui |
| Boeing | Oui | | | Oui |
| Starmate | Oui | | | |
| AR$^2$Hockey | Oui | Oui (second ordre) | Non | Oui |
| CAMELOT | Oui | | | Non |
| iMAGIS | Oui | | | Non |
| AMRA | Oui | | Non | Oui |

TAB. 1.4 – Le type de recalage utilisé pour chaque projet

informations visuelles. À ce tour d'horizon s'ajouteront la description de systèmes et solutions techniques et scientifiques correspondant aux différents verrous abordés par les chapitres qui constituent cette thèse.

Dans le chapitre suivant, nous présenterons les architectures des principaux systèmes de RA et celle de notre système qui s'appuie sur le paradigme de la programmation par composants qui constitue une des premières contributions de cette thèse.

# Chapitre 2

# Architecture logicielle des systèmes de réalité augmentée

Dans ce chapitre, nous aborderons des notions d'ordre général à propos des architectures logicielles employées dans les systèmes de réalité augmentée. Nous allons commencer par étudier les systèmes qui existent actuellement puis nous exposerons notre proposition d'architecture.

Au vu des différents systèmes présentés, lors du chapitre précédent, il apparaît que les solutions apportées pour chacune des applications de réalité augmentée (RA) jusqu'à présent dépendent étroitement de 3 facteurs :

- Les technologies de "tracking" disponibles,
- Les algorithmes et procédés de filtrage applicables aux systèmes de RA,
- Le domaine d'application.

Peu à peu est venu le besoin de construire des systèmes et de concevoir des architectures logicielles qui fournissent une couche d'abstraction entre les diverses technologies de tracking et les algorithmes de fusion et de traitement des données. Ceci est compréhensible étant donné que la réalité augmentée est un domaine de recherche assez jeune qui est tributaire de l'évolution des nouvelles technologies.

De plus, dans le but d'appliquer ces systèmes dans un milieu industriel, il devient nécessaire de pouvoir adapter facilement une application et de la faire évoluer simplement en vue de s'adapter aux contraintes du milieu. La notion de prototypage rapide d'applications intervient alors.

Nous allons donc, dans un premier temps, passer en revue les architectures logicielles existantes qui ont contribué à améliorer la flexibilité de ces systèmes face à la grande diversité des capteurs et des algorithmes existants. Puis, nous verrons comment la programmation orientée composants peut apporter une solution à ce problème. Enfin, nous terminerons en exposant notre proposition d'architecture logicielle qui sera ensuite employée au cours des diverses phases d'élaboration et de conceptions de nos systèmes de réalité augmentée employés dans le cadre de la maintenance industrielle.

## 2.1 Les architectures modulaires existantes

Le domaine de la réalité augmentée est riche en solutions architecturales développées dans le but de résoudre le problème de la modularité des applications développées, et de faciliter la réutilisation du code écrit. Ainsi, ces dernières années, environ une trentaine de solutions de ce type ont vu le jour [Endres et al., 2005]. Au lieu d'effectuer une liste exhaustive de chacune de ces architectures, nous avons préféré en sélectionner quelques unes parmi les plus représentatives. Nous comparerons également les diverses approches présentées à la section 2.1.7 page 38.

### 2.1.1 COTERIE

COTERIE (Columbia Object-oriented Testbed for Exploratory Research in Interactive Environments) était un projet de l'université de Columbia de 1996 à 1999. Ce système supporte la création d'environnements virtuels avec plusieurs utilisateurs simultanés qui peuvent interagir avec une palette hétérogène de dispositifs d'affichage et de périphériques d'entrée.

Cet environnement est développé sur la base d'un modèle de programmation orientée objet, modulaire et multi-processus. Il permet de communiquer de manière transparente via un modèle client-serveur et des objets distribués. Les applications développées sur cette plate-forme comprennent EMMIE un gestionnaire d'environnement fournissant des techniques d'interaction pour les environnements de RA [Butz et al., 1999] et MARS (Mobile Augmented Reality Systems) qui est la série des prototypes de RA développée par l'université de Columbia [Feiner et al., 1997].

COTERIE se base sur un graphe de scène distribué dont les propriétés sont localement modifiées pour chaque système client. Si cet environnement est particulièrement adapté au prototypage rapide d'applications collaboratives de réalité augmentée, sa faiblesse réside dans le fait que l'essentiel des informations passent par le graphe de scène dont on ne peut se dispenser. Le prototypage de l'application se fait au moyen d'un langage de scripts qui permet de modifier rapidement le comportement des objets du graphe de scène.

### 2.1.2 StudierStube

Développé par l'université de Vienne depuis 1997, la StudierStube [Schmalstieg et al., 2002] est l'architecture spécifique à la RA la plus ancienne encore en activité. Le principe est que chaque utilisateur dispose d'un espace de travail qui peut-être mis en commun avec d'autres, en partageant un graphe de scène commun. La synchronisation des graphes de scène est prise en charge par un protocole réseau robuste. Les espaces de travail peuvent se découvrir les uns les autres grâce à un gestionnaire de sessions centralisé.

Pour ce qui est de la gestion des capteurs et des périphériques d'entrée, ce projet s'appuie sur OpenTracker [Reitmayr et Schmalstieg, 2001], un système s'appuyant sur une description XML (eXtensible Markup Language) des flux de données, ce qui favorise une grande flexibilité et permet de gérer plusieurs capteurs de position et d'orientation simultanément. Le développeur peut programmer entièrement à l'aide de scripts OpenInventor en utilisant des noeuds pré-définis. Il peut également implémenter ses propres noeuds en C++.

Ce système convient pour les projets de réalité augmentée collaboratifs et distribués.

### 2.1.3 Tinmith

L'université d'Australie du Sud démarra en 1998 le projet TINMITH [Piekarski et Thomas, 2003] [Tinmith, url]. Ce dernier, axé sur la description des flux de données, est une librairie d'objets hiérarchisés calqué sur le modèle des systèmes de fichiers Unix. Cet ensemble de classes permet de gérer les flux de données issus des capteurs, les différentes opération de filtrage des données ainsi que le rendu final des applications. Cet ensemble d'objets écrits en C++ s'appuie sur un système de callback et un mécanisme de sérialisation du flux de donnée employant le XML. L'ensemble des communications entre les objets est géré par un graphe de flux de données.

Ce système, que nous avons déjà qualifié de hiérarchisé, est organisé suivant un paradigme de couches (figure 2.1 page suivante) que le flux de données doit traverser, de l'acquisition des données jusqu'à l'étape de rendu de la scène qui va mélanger réel et virtuel.

Cette architecture, bien que plus contraignante pour le développeur, est adaptée aux systèmes de réalité augmentée mobile, particulièrement pour les systèmes embarqués grâce à l'effort d'optimisation des briques élémentaires du système.

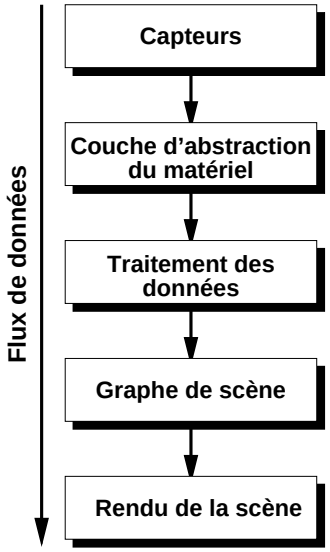



FIG. 2.1 – Flux de données et disposition de l'architecture en couches du système Tinmith

### 2.1.4 DWARF

Initié en 2000 par l'université de Münich, le projet DWARF [Bauer et al., 2001] [Dwarf, url] (pour Distributed Wearable Augmented Reality Framework) est un système de composants distribués entièrement décentralisé. Le système est constitué d'un ensemble de services basés sur CORBA (Common Object Request Broker Architecture - voir section 2.2.3.1 page 43 ) orientés vers une architecture générique pour la réalité augmentée.

Chaque service est décrit en utilisant des fichiers XML et sont indépendants les uns des autres, ce qui leur permet de fonctionner sur différentes machines d'un réseau. Les services sont sollicités dynamiquement à l'exécution, formant un graphe de flux de données distribué. Pour démarrer, le système ne nécessite qu'un composant pour gérer les services. Ce dernier tentera de découvrir si d'autres services sont présents sur le réseau ou non, ce qui permet d'intégrer de nouveaux services à l'exécution.

Dwarf est sans conteste l'un des systèmes les plus aboutis en matière de réalité augmentée distribuée. Toutefois, ce système présente certains inconvénients :

- la découverte des services se fait par diffusion sur le réseau complet des méta-données associées à ces services, ce qui entraîne une charge supplémentaire sur le réseau,
- DWARF est étendu en fonction des applications de réalité augmentée, et ne donne pas de modèle récurent aux applications de réalité augmentée,
- le système décrit une macro-architecture et ne peut pas traduire les détails de cette dernière. Ceci veut dire que les composants peuvent renfermer plusieurs sous-constituants différents dont l'un d'entre eux, s'il est modifié, entraîne la modification de l'ensemble du composant,
- ce système est dépendant, dans une large mesure, d'un environnement distribué, ce qui le rend difficile à employer pour des petites applications de réalité augmentée.

### 2.1.5 AMIRE

AMIRE [Abawi et al., 2004] [Amire, url] [Dörner et al., 2002] (pour Authoring MIxed REality) est un projet européen comprenant plusieurs partenaires dont Siemens et le Fraunhofer AGC qui s'est déroulé sur vingt-sept mois, de 2002 à 2004. Le but avoué de ce projet était de définir et de développer une architecture système qui permette de concevoir et d'implémenter rapidement des applications de réalité mixée sans connaître les détails des technologies sous-jacentes qui sont employées.

Le paradigme employé ici est également la programmation orientée composants à laquelle nous

consacrerons une section ultérieurement (cf section 2.2 page suivante). Il est à noter que cette architecture comporte une granularité à deux niveaux : les gemmes (qui sont des micro-composants) et les composants (qui sont une agrégation de plusieurs gemmes). Les deux classes de composants comportent chacune une interface de configuration, des "slots" d'entrée, des "slots" de sortie qui peuvent être reliés aux slots d'entrée d'autres composants. Par ailleurs, cette architecture possède une ouverture sur les systèmes distribués par le biais de l'utilisation de CORBA.

Enfin, ce système contient également un éditeur graphique d'applications. Ce dernier permet à un utilisateur de connecter les divers composants tout en étant guidé pas à pas pour suivre des règles génériques de conception des applications de réalité mixée. L'éditeur sauvegarde la configuration dans un fichier XML. Ce dernier est ensuite chargé par un gestionnaire de composants qui va lancer l'application. Nous noterons que ce système est celui qui est le plus proche en termes de concepts de celui que nous allons présenter ultérieurement.

### 2.1.6 DART

Plus récemment, en 2003, l'université de Columbia a démarré un autre projet appelé DART [MacIntyre et al., 2003] [MacIntyre et al., 2004] [Dart, url] (Designer's Augmented Reality Toolkit). Ce projet, orienté vers l'expérimentation et le prototypage rapide d'applications de réalité augmentée, s'appuie sur le moteur de Macromedia Director. Il emploie donc les paradigmes de base de Director qui font de lui un outil d'édition de contenus (il permet de paramétrer le comportement et les propriétés des objets virtuels).

Ce système inclut également un support pour les fonctions de bas niveau qui gèrent les capteurs, caméras et autres capteurs nécessaires. Cette édition présente l'avantage de se faire simplement à l'aide d'une interface graphique, ce qui permet de paramétrer en quelques clics une application de réalité augmentée.

Les développements nécessitent une connaissance du langage C++ et du langage de scripts de Director : LINGO.

### 2.1.7 Comparaison des systèmes

Les divers systèmes présentés possèdent leurs avantages et leurs inconvénients propres. Il convient d'établir un certain nombre de critères communs de classification de ces derniers. Puisque nous avons évoqué la modularité dont le système doit faire preuve pour accéder à un système de prototypage rapide d'applications, les critères que nous allons examiner seront en rapport avec ce concept.

Si nous parlons de modularité, cela veut dire que chaque système est composé de *modules*. Suivant les types de paradigmes utilisés lors de l'élaboration des diverses architectures présentées, cela va se traduire par d'autres termes : notamment *composants*, *services* ou, tout simplement, *objets*.

Ces différents modules possèdent une *granularité*, qui est déterminé par la taille des divers constituants du système. Une granularité fine signifiera qu' un module peut se contenter d'être l'implémentation d'un seul algorithme qui sera réutilisé en conjonction avec d'autres. A l'autre extrémité, nous trouvons les systèmes monolithiques. La granularité est importante car elle favorise la flexibilité des applications développées. D'un autre côté, une granularité fine peut parfois engendrer des inconvénients en termes de performances, en effet, si celle-ci est trop fine, le coût en temps de calcul de la communication entre les composants devient supérieur au coût en temps de calcul des actions effectuées par le composant en lui-même.

En termes de réutilisation du code développé, les divers modules programmés peuvent se comporter comme des boîtes noires, c'est à dire que leurs entrées/sorties sont connues, leur fonctionnement général également, mais il n'est pas nécessaire de savoir comment l'algorithme est implémenté au sein

du composant pour le réutiliser. Ceci veut dire que le système peut renfermer divers *niveaux d'abstraction* qui dépendent de l'utilisateur du système. Ainsi le développeur aura besoin de ses connaissances de programmation pour développer chaque composant. Toutefois, du point de vue d'un utilisateur final qui veut simplement réutiliser ces fonctionnalités, il est nécessaire de déterminer s'il doit connaître ces méthodes de programmation ou au contraire s'il peut créer une application (prototypage rapide) sans connaissances particulières .

Enfin, l'environnement dans lequel les modules sont destinés à être déployés va influer sur l'architecture du système. Ce dernier peut être *portable* et peut fonctionner sur de multiples plates-formes (au minimum Unix et Microsoft Windows) ou, au contraire, peut-être restreint à un système. D'autre part, les modules peuvent être *distribués* sur un réseau de machines ou au contraire situés sur un seul ordinateur. D'autres systèmes permettent également les deux, avec ou sans perte de *performances*. Ainsi, une architecture pensée pour un système distribué dès le départ aura le plus grand mal à fonctionner sur une seule machine sans pertes de performances. A l'inverse, une architecture pensée pour un système unique peut ne pas offrir d'extensions pour un système distribué. Elle sera alors *non distribuable*, par opposition à une architecture conçue pour un système unique mais qui s'avère *distribuable*. Le tableau 2.1 compare les systèmes que nous avons étudiés en fonction des caractéristiques que nous avons énoncé.

| Système | Granularité | Niveau d'abstraction | Portabilité | Type |
|---|---|---|---|---|
| COTERIE | Moyenne | Nécessite la connaissance d'un langage de scripts | oui | Distribué |
| StudierStube | Moyenne | Nécessite la connaissance d'un langage de scripts | oui | Distribué |
| TINMITH | Variable | Nécessite la connaissance du langage C++ | oui | Distribuable |
| DWARF | Grossière | Développeur/utilisateur séparés | oui | Distribué |
| AMIRE | Variable | Développeur/utilisateur séparés | non | Distribuable |
| DART | Grossière | Développeur/utilisateur séparés | non | Non Distribuable |

TAB. 2.1 – Tableau de comparaison des architectures modulaires de réalité augmentée

Par rapport à de telles architectures, nous allons voir comment nous allons *construire un système* qui sera *à granularité variable, à plusieurs niveaux d'abstraction, portable et conçue pour fonctionner sur un système unique sans pertes notables de performances tout en gardant une ouverture pour permettre au système d'être distribuable*. Ceci se fera grâce à la programmation orientée composants qu'il convient d'introduire afin de jeter les bases de notre système.

## 2.2 La programmation orientée composants

La programmation orientée composants (Component Based Software Engineering) est une technique de programmation qui est apparue dans les années 1990. Ce type de programmation, comme nous allons le voir, permet une grande modularité du code. L'idée de base est de travailler sur des entités nommées *composants* que nous allons définir à présent.

### 2.2.1 Définition d'un composant

Comme il existe plusieurs systèmes de composants, il y a plusieurs définitions possibles de ces derniers. Elles présentent de nombreuses similarités entre elles tout en divergeant sur certains points particuliers.

A titre d'exemple, voici deux définitions :

La première [D'Souza et Wills, 1998] est : *"Un composant est un module cohérent d'un programme qui peut être développé séparément et distribué en tant qu'unité indépendante. Il offre une interface par laquelle il peut être connecté, tout en restant inchangé, avec d'autres composants pour former un système de taille plus importante."*

La deuxième [Szyperski, 2002] est : *"Un composant est une unité de composition avec une interface spécifiée contractuellement et des dépendances contextuelles uniquement explicites. Un composant peut-être déployé de manière indépendante et est sujet à une composition avec d'autre composants."*

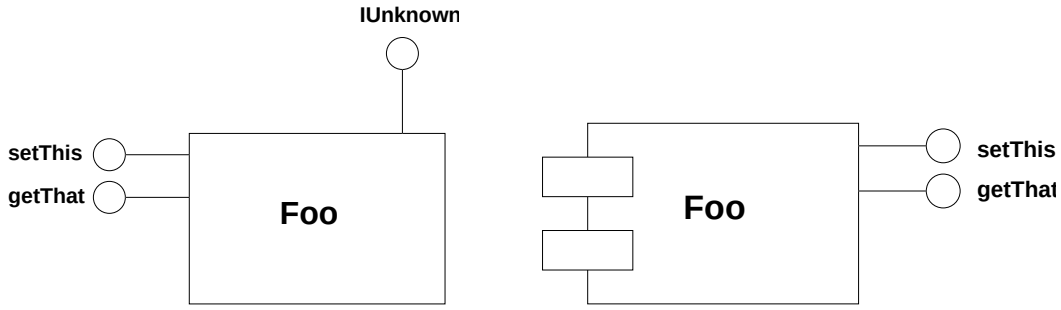
#### 2.2.1.1 Propriétés des composants

Si ces définitions sont légèrement différentes, en revanche, elle traduisent les mêmes propriétés [Szyperski, 2003] :

1. un composant renferme du code compilé,
2. un composant implémente une ou plusieurs interfaces définies,
3. un composant se comporte comme un boîte noire : les détails de son implémentation sont inconnus,
4. un composant doit pouvoir fournir une interface,
5. un composant est sujet à une composition avec d'autres composants.

#### 2.2.1.2 Représentations des composants

Suivant les systèmes de composants, ces derniers peuvent être représentés de manières différentes (nous verrons plus loin quelle représentation nous emploierons). La figure 2.2 nous montre deux représentations différentes des composants. L'une est empruntée à COM (Component Object Model - présenté en 2.2.3.2 page 43) , l'autre à UML (Unified Modelling Language). Dans les deux cas, les ronds donnent les interfaces implémentées par le composant qui est représenté par le rectangle. L'interface *IUnknown* est un trait caractéristique des composants COM.




(a) Composant COM (b) Composant UML

FIG. 2.2 – Conventions de représentation des composants en COM et en UML

Les composants, comme on le voit, sont des concepts de programmation d'un haut niveau d'abstraction. Leur mise en oeuvre fait appel à des techniques de base de programmation que nous allons présenter.

### 2.2.2 Concepts sous-jacents à la programmation par composants

La programmation par composants s'appuie sur un certain nombres de notions en informatique qu'il convient à présent de décrire. En effet, un composant est distribué sous forme de librairie dynamique. De plus, il doit être à tout moment possible de déterminer l'interface du composant, ce qui

peut-être effectué en employant les mécanismes d'introspection. Enfin, il convient de détailler les différents modes de communication entre les composants puisque ceux-ci sont destinés à être composés avec d'autres composants.

#### 2.2.2.1 Les librairies dynamique

Comme nous l'avons vu, un composant renferme du code compilé. Ceci veut dire que le code en question doit renfermer des points d'accès pour permettre l'utilisation du module à l'exécution de l'application : nous tenons la description du comportement d'une librairie dynamique.

L'un des corollaires à ce principe est le système de plugins qui permet, en employant des librairies dynamiques qui doivent respecter une certaine interface de chargement (les points d'entrée), de changer le comportement d'une application. Hormis cette interface qui doit être respectée, l'utilisateur est libre du choix du langage dans lequel il veut écrire ses librairies. Le mécanisme des plugins est donc potentiellement intéressant en ce sens qu'il permet l'ouverture d'un logiciel à de futures évolutions, comme de nouveaux algorithmes qui ne seront élaborés que dans le futur et donc non existants à l'heure de l'écriture du logiciel. Si ce système de plugin reste générique, c'est à dire que l'interface de chargement est la moins spécialisée possible, nous obtenons ainsi un système de chargement de composants.

#### 2.2.2.2 Le principe d'introspection

Ce principe est couramment employé dans certains langages de programmation orientés objet tels que SmallTalk [Goldberg et Robson, 1989], Java [Gosling, 1996] [Microsystems, 1997a], Objective C [Inc, 1993] et C# [Hejlsberg et Wiltamuth, 2001]. Ce type de mécanisme permet à un objet d'exporter lui-même sa propre interface (c'est à dire les méthodes et les membres qui le composent) pendant l'exécution du programme.

Dans le cadre de la programmation par composants, puisque ces derniers doivent fournir une interface, il est possible ainsi de faire en sorte que le composant auto-exporte sa propre interface. Toutefois, il est à noter que très souvent les systèmes de composants n'utilisent pas cette fonctionnalité mais préfèrent garder un fichier de définition de l'interface à côté du code compilé.

#### 2.2.2.3 La communication inter-composants

Les divers composants d'un système doivent être sujet à composition avec d'autres composants. Ceci se fait par le biais d'un mécanisme de communication. Les liens liant les composants s'appellent alors connections. Suivant le système de composants, nous pouvons avoir plusieurs type de liens. Metha et al. dressent une taxonomie complète de ces connecteurs [Mehta et al., 2000]. Les grandes familles en sont les suivantes :

- Les appels de procédure,
- Les évènements,
- Les connecteurs d'accès aux données,
- Les connecteurs de liens,
- Les flux de données,
- Les arbitres qui gèrent par exemple les accès concurrentiels,
- Les adaptateurs qui peuvent fournir des mécanismes de sérialisation (qui est l'encapsulation de données multiples dans un flux de données unique),

Parmi ces familles de connecteurs, nous n'en présenterons que deux car ce sont celles que nous avons employé par la suite.

**Les appels de procédure**

Les *appels de procédures* modélisent le flux de contrôle entre les composants via diverses techniques d'invocation. Ceci permet aussi le transfert de données entre les composants grâce aux paramètres passés. Parmi les exemples que l'on peut citer, nous trouvons les méthodes de la programmation orientées objets, le système fork et exec des environnements Unix, les invocations de fonctions de "callback" des systèmes basés évènements et les appels systèmes. Les appels de procédures sont aussi à la base de connecteurs composites tels que RPC (Remote Procedure Call) [Birell et Nelson, 1984].

Parmi ces mécanismes, nous citerons particulièrement le paradigme *signal-slot*. Ce dernier est un mécanisme très répandu dans le domaine de la conception des interfaces graphiques pour les utilisateurs. A notre connaissance, quatre librairies possèdent de tels mécanismes :

- QT [Qt, url] une librairie de programmation d'interfaces graphiques,
- libsigc++[Sigc++, url] une librairie issu de GTK qui est elle-même une librairie de programmation d'interfaces graphiques,
- sigslot [Sigslot, url], une librairie dédiée au mécanisme signal/slot,
- boost [Boost, url], une metalibrairie répertoriant un grand ensemble de fonctionnalités pour les programmeurs C++.

Les concepts derrière ce type de mécanisme sont assez simples en réalité : chaque objet possède des signaux et des slots. Un objets qui veut communiquer avec un autre émettra alors le contenu de sa communication via un signal qui appellera alors un slot qui recevra le même type de contenu que celui qui a été envoyé. De plus, un tel mécanisme rend possible la connexion et la déconnexion à la demande des objets. Il doit être éventuellement capable de pister et suivre les données transmises. Á un slot peuvent être connectés plusieurs signaux et, de même, un signal peut être connecté à plusieurs slots.

Un certain nombre de variations existent dans la mise en oeuvre de tels mécanismes. Certaines librairies effectuent la vérification des types transmis entre signaux et slots directement à la compilation (c'est le cas de sigslots et libstdc++) alors que d'autres, plus permissives, effectuent cette vérification à l'exécution (c'est le cas de QT) au prix d'un surcroît en temps de calcul.

Ces appels de procédure restent toutefois les plus rapides en terme de temps d'exécution. En revanche, leur utilisation ne permet pas toujours de visualiser la manière dont les données sont traitées. Pour cela, les communications à base de flux de données demeurent les plus adaptées.

**Les flux de données**

Les *flux* sont utilisés pour effectuer des transferts d'un large ensemble de données entre plusieurs processus autonomes. Ils sont également employés dans les systèmes client-serveur avec des protocoles de transfert de données pour retourner le résultat des calculs.

Les flux ont été employés dans des modèles formels d'architecture pour représenter des connecteurs avec des protocoles d'utilisation assez complexes. Les flux peuvent être combinés avec d'autres types de connecteurs pour fournir des connecteurs composites où les autres connecteurs sont encapsulés dans les flux. On parle à ce moment de *sérialisation*.

Les exemples classiques de flux sont les pipes Unix (cette architecture est aussi appelé architecture "*pipe-filters*"), les sockets de communication TCP/UDP et les protocoles client/serveur propriétaires.

Une discussion sur les avantages et les inconvénients de ce type d'architecture est détaillé à la section 2.3.4 page 49.

Nous venons de définir le concept de composant. Nous avons également examiné les mécanismes de programmation de base employés par ces entités. Nous allons maintenant examiner quelques systèmes de programmation par composants.

### 2.2.3 Principaux systèmes de composants

Parmi les systèmes de programmation par composants qui reviennent le plus souvent dans la littérature, nous pouvons en citer plus particulièrement quatre : CORBA, COM et ses dérivés, XPCOM et JavaBeans. Nous allons les présenter de manière succincte, en reprenant le modèle de la présentation de ces systèmes proposé par Cox et Song [Cox et Song, 2001].

#### 2.2.3.1 CORBA

CORBA (Common Object Request Broker Architecture) a été proposé par l'OMG (Object Management Group) en tant que standard pour la communication entre composants. Il s'agit d'une partie d'un modèle plus important, l'OMA (Object Management Architecture) qui définit à un haut niveau d'abstraction un contexte dans lequel un composant opère et interagit, ce qui inclut les services standards sur lesquels les composants peuvent se baser.

Un composant CORBA fournit ses fonctionnalités via une interface définie dans le langage IDL (Interface Definition Language) et n'interagit avec les autres composants que par le biais de cette interface. De plus, l'intégralité de l'architecture CORBA est orientée vers les invocations de méthodes d'objets distants, ce qui en fait un excellent choix pour les systèmes distribués.

#### 2.2.3.2 COM et ses dérivés

COM [Com, url] (Component Object Model), ses dérivés COM+, DCOM (la version distribuée de COM) et plus récemment .NET et ses extensions spécifiques telles que OLE ou ActiveX, sont des outils crées par Microsoft et, par voie de conséquence, implémentés seulement sur les plate-forme de l'éditeur de logiciels en question.

COM spécifie un standard pour les communications entre les composants qui consiste en des interfaces COM et un système pour enregistrer et passer des messages entre composants par le biais de ces dernières. Les interfaces COM sont définies en employant le langage MIDL (Microsoft Interface Definition Language).

COM définit les interfaces entrantes comme des interfaces recevant des appels venant des autres composants. De même, les interfaces sortantes sont des interfaces au moyens desquelles les autres composants sont invoqués. La communication se fait par le biais d'évènements.

#### 2.2.3.3 XPCOM

Le système XPCOM [Xpcom, url] (pour Cross-Platform Component Object Module) est le système de composants développé par la Mozilla Foundation. Celui-ci est utilisé dans les logiciels développés par l'organisation, notamment Mozilla, Firefox et Thunderbird.

Structurellement similaire à COM, XPCOM est pensé pour être utilisé essentiellement au niveau applicatif. Il est également, comme son nom l'indique, multi plate-forme, ce qui n'est pas le cas de COM.

XPCOM emploie également son propre langage de définition des interfaces : XPIDL qui est basé sur le langage IDL de CORBA et qui contient des extensions spécifiques relative au composants XPCOM.

#### 2.2.3.4 JavaBeans

Selon ses spécifications [Microsystems, 1997b], un Java Bean est *"un composant logiciel réutilisable qui peut être manipulé visuellement à l'aide d'un outil de développement"*. Un Java Bean peut-être un élément d'une interface graphique ou, par exemple, un composant assurant la connexion à une base de données.

Les composants Java Beans communiquent les uns les autres à l'aide d'évènements. L'interface entre les composants est donc assurée par le modèle de gestion des évènements du langage Java. En plus des objets Java classiques, Les composants possèdent des propriétés, qui peuvent être réglées par le programmeur, pour ajuster leur comportement. Les JavaBeans implémentent donc certaines interfaces spécifiques qui font d'eux des composants.

#### 2.2.3.5 Comparaison rapide de ces systèmes

Si l'on reprend les mêmes critères que ceux employés pour comparer les architectures génériques des systèmes de réalité augmentée (voir section 2.1.7 page 38), il s'avère rapidement que la granularité de chaque système de composants sera variable. En effet, celle-ci dépendra de ce qui est implémenté dans les composants. En revanche, nous rajouterons ici un critère de performances au niveau de l'invocation des méthodes des divers composants. Le tableau 2.2 synthétise ces résultats. Comme nous pouvons le remarquer, le système à base de JavaBeans offrait une solution avec des caractéristiques intéressantes par rapport au système que l'on envisage de produire. Toutefois, ses performances faibles liées à l'utilisation de la machine virtuelle Java et aux communications événementielles (nécessairement asynchrones) en font un mauvais candidat.

L'ensemble de ces systèmes permet de livrer un composant, c'est à dire une librairie compilée sans avoir à fournir le code source. Toutefois, il est nécessaire, si l'on veut intégrer ces objets dans le cadre d'un projet ou d'une application d'écrire ce que l'on appelle la *"glu logique"* qui va permettre de connecter ces divers composants. Il s'agit d'instructions écrite dans un langage de programmation ou de script détaillant la manière dont les composants sont liés les uns aux autres.

A l'exception des JavaBeans, l'écriture de cette glu logique nécessite la connaissance d'un langage de script ou de programmation, ce qui ne fournit pas un niveau d'abstraction suffisant pour un utilisateur final. Nous verrons ultérieurement que nous aurons à développer notre propre système de composants afin d'offrir une solution simple et satisfaisante à ce problème.

| Système | Niveau d'abstraction | Portabilité | Type | Performances |
|---|---|---|---|---|
| CORBA | Nécessite l'écriture de "glu logique" | oui | Distribué | + |
| COM | Nécessite l'écriture de "glu logique" | non | Distribuable | ++ |
| XPCOM | Nécessite l'écriture de "glu logique" | oui | Distribuable | ++ |
| JavaBeans | Développeur/utilisateur séparés | oui | Distribuable | - |

TAB. 2.2 – Tableau de comparaison des systèmes de programmation orientée composants

La programmation orientée composant présente certains avantages qui font qu'elle connaît un succès grandissant parmi les entreprises du secteur logiciel mais également dans les laboratoires universitaires. Nous allons examiner maintenant les raisons de cet engouement.

### 2.2.4 Avantages et inconvénient de la programmation orientée composants

#### 2.2.4.1 Avantages et inconvénients communément admis

Essentiellement utilisée par les entreprises de création de logiciels, la programmation orientée composants compte, parmi ses avantages communément admis :

1. **La centralisation des compétences :** L'équipe de développement peut-être divisée en sous-groupes, chacun se spécialisant dans le développement d'un composant,
2. **La sous-traitance :** Le développement d'un composant peut-être externalisé, à condition d'en avoir bien réalisé les spécifications au préalable,

3. **Facilité de mise à jour :** La modification d'un composant ne nécessite pas la recompilation du projet complet, voir pas de recompilation du tout,
4. **Facilité de livraison/déploiement :** Dans le cas d'une mise à jour, ou d'un correctif, la livraison en est facilitée, puisqu'il n'y a pas besoin de re-livrer l'intégralité du projet, mais seulement le composant modifié,
5. **Choix des langages de développement :** Il est possible, dans la plupart des cas, de développer les différents composants du logiciel dans des langages de programmation différents. Ainsi, un composant nécessitant une fonctionnalité particulière pourra profiter de la puissance d'un langage dans un domaine particulier, sans que cela n'influe le développement de l'ensemble du projet,
6. **Productivité :** La réutilisabilité d'un composant permet un gain de productivité non négligeable car elle diminue le temps de développement, d'autant plus que le composant est réutilisé souvent.

L'un des inconvénients essentiels de ce type de programmation est que le découpage fonctionnel d'une application en divers composants doit être mené de manière très rigoureuse, tout en pensant au fait que le composant doit être réutilisable, c'est à dire penser également à son utilisation future. Ceci peut amener un système développé en employant la méthode de programmation orientée composants à consommer sensiblement plus de ressources qu'une application programmée avec une méthode traditionnelle. En effet, le système de communication entre les composants est plus coûteux en temps de calcul qu'un simple appel de fonction.

#### 2.2.4.2 Intérêt pour le monde académique

Si la programmation orientée composant a gagné des secteurs entiers de l'industrie du logiciel, le monde académique n'est pas en reste par rapport à cette technique. A titre d'exemple, nous pouvons citer quatre projets qui s'appuient sur une architecture à base de composants.

**OSCAR**

OSCAR [Blum, 2001] (Operating System for the Control of Autonomous Robots) est une architecture développée par l'université de Munich, basée sur des composants pour la robotique mobile. Ce domaine de recherche présente quelques problématiques communes avec la réalité augmentée, notamment le problème de la localisation. Ainsi, les raisons avancées par l'auteur pour justifier son approche basée composants sont les suivantes :

- Les mesures acquises par les capteurs se font en parallèle et nécessitent plusieurs chaînes de traitement concurrentes,
- La modularité aide au développement en équipe et facilite les tests,
- L'utilisation des composants apporte robustesse, flexibilité et reconfigurabilité au système.

L'auteur développe ensuite un système pour la robotique mobile basé sur CORBA, ce qui lui permet de décentraliser certains calculs sur des machines distribuées.

**ORCA**

Plus récemment, toujours en robotique mobile, le système ORCA [Brooks et al., 2005] a été développé par l'université de Sydney. Les motivations derrière la création de ce système, basé ici aussi sur CORBA sont les suivantes :

1. *La complexité inhérente* de la robotique mobile. Celle-ci est due à l'interaction de capteurs hétérogènes, avec de nombreux algorithmes (un problème similaire jusque là à celui de la réalité augmentée mobile) et avec un certain nombre d'actionneurs.
2. *Un besoin de flexibilité*. Un système doit être suffisamment flexible pour pouvoir mener des expérimentations sur des algorithmes de fonctionnalités équivalentes sans que ceci ait des répercussions sur l'ensemble du système.
3. *Un environnement distribué*. Ceci pour pouvoir contrôler les robots à distance pendant les phases d'expérimentation.
4. *L'hétérogénéité des plate-formes et du matériel informatique*. Ceci est un problème récurrent en robotique mobile.

Par la suite, chaque composant de ce système est supposé être un processus autonome qui va interagir avec un autre ensemble de processus. Fait intéressant, la connectivité de chacun des composants est assurée par l'interprétation d'un fichier XML (eXtensible Markup Language) qui dresse la liste des connections à établir, fournissant ainsi une première couche d'abstraction par rapport à l'assemblage des composants au lieu de l'utilisation de "glu logique".

### DWARF

Nous avons déjà présenté précédemment le projet DWARF (cf section 2.1.4 page 37) mais nous allons néanmoins nous étendre sur les raisons du choix de la programmation orientée composants dans cette solution. Elle est justifiée par les concepteurs du système [Bauer et al., 2001] de la manière suivante : chaque démonstrateur de réalité augmentée développé jusqu'à l'époque de la création de leur système était centré sur une application précise, ce qui en faisait un système monolithique difficile à réemployer dans d'autres applications.

Toutefois ces différentes applications emploient les mêmes types d'algorithmes et effectuent des tâches très similaires. Elles ont en commun certains schémas de conception (appelés aussi "design patterns" ) qui posent dans les grandes lignes les architectures de ces applications. Ceci veut dire que si ces systèmes avaient été développés de manière générique et modulaire, ces derniers auraient pu être réemployés dans d'autre systèmes de réalité augmentée à moindre coût en termes d'effort et de temps de développement. Ceci se traduit par un temps plus grand consacré à la recherche de nouveaux algorithmes et de nouveaux concepts puisqu'un temps moindre est consacré au développement des prototypes et des démonstrateurs de ces technologies.

La conséquence directe de cette conclusion a débouché sur le développement du système DWARF qui est orienté composant. Par la suite, les auteurs se sont attaché à identifier ces schémas de conception [MacWilliams et al., 2004] communs aux applications de réalité augmentée afin de dégager les composants communs aux systèmes de réalité augmentée.

### AMIRE

Ce dernier système fut également sommairement présenté précédemment (cf section 2.1.5 page 37). Le choix de la programmation orientée composants a été réalisé ici également pour des raisons de réutilisation du code déjà développé dans d'autres applications de réalité augmentée.

L'autre raison est liée aux design patterns évoqué à propos de DWARF : les concepteurs du projet ont jugé bon d'assister le concepteur d'une application de réalité augmentée via une interface graphique de création des applications de réalité mixée. C'est pourquoi, un squelette d'application est fourni sur lequel l'utilisateur greffe les composants qui vont assurer le bon fonctionnement du système. Ceci permet, à une personne qui n'est pas experte dans le domaine de la réalité augmentée, de

réaliser sa propre application de réalité augmentée.

De l'ensemble des applications citées, il ressort que la programmation orientée composants présente des intérêts certains lors de la conception d'une architecture générique pour les systèmes de réalité augmentée.

Au travers de ce chapitre, nous avons étudié les diverses architectures modulaires qui peuvent exister au sein des systèmes de réalité augmentée existants. Ceci nous a permis de dégager des critères de comparaisons entre ces derniers pour jeter les bases de notre système. Celui-ci devra satisfaire les *critères 1) de granularité variable, 2) de multiplicité des niveaux d'abstraction, 3) de portabilité et 4) de pouvoir fonctionner sur une machine unique tout en restant distribuable*.

En étudiant la programmation orientée composants, il apparaît que celle-ci peut fournir une architecture générique capable d'apporter des solutions satisfaisant les critères énumérés. Nous allons à présent construire cette solution en nous attachant à déterminer quel type de composants nous allons employer et par le biais de quel type de communication ils vont être connectés.

## 2.3 Ébauche d'une première architecture adaptée aux applications de réalité augmentée

Dès 2002-2003 [Didier, 2003a] [Didier, 2003b], nous nous sommes attachés à jeter les bases d'une architecture qui mette en oeuvre les principes de modularité et d'interchangeabilité des composants. Celle-ci posait les bases et identifiait les composants nécessaires, de manière schématique, à un système de réalité augmentée.

### 2.3.1 Décomposition fonctionnelle d'un système de réalité augmentée

Ce découpage en grands domaines de fonctionnalités est représenté à la figure 2.3 page suivante. Basée sur la description des flux de données, cette architecture nous amène des capteurs jusqu'au moteur de rendu pour la réalité augmentée. Les divers composants fonctionnels à haut niveau sont les suivants :

- Les *capteurs*, qui ne font pas à proprement parler partie du système mais qui sont à la source des données nécessaires au bon fonctionnement du système,
- Le *Traitement des données capteurs*, qui centralise tous les traitements associés aux données issues des capteurs. Ces traitements concernent notamment l'estimation des paramètres de localisation nécessaire en vue de recaler le réel et le virtuel. Ils incluent également les primitives de traitement d'image, image qui pourra être restituée, dans le cas de la réalité augmentée en vision indirecte au moteur de rendu pour la RA,
- Les *évènements* utilisateurs déclenchés par l'utilisation d'une interface graphique ou par l'emploi des périphériques d'entrée courant d'un système informatique tels que la souris ou le clavier. Ces évènements peuvent agir sur le contexte d'exécution de l'application de réalité augmentée,
- Un *espace mémoire partagé*, qui est placé de manière à assurer le lien entre le rendu de l'application , considéré comme un processus à part et le (ou les) processus de traitement des données issues des capteurs,
- Le *moteur de rendu pour la RA*, qui centralise les problèmes liés à l'affichage des augmentations. Ce moteur peut aussi gérer des données multimédia, voire des graphes de scène, en fonction du contexte de l'application.

Toutefois, dans l'optique de résoudre le problème du traitement des données issues des capteurs, surtout si ces derniers fournissent des informations de nature hétérogène, nous avons poursuivi plus avant la décomposition fonctionnelle au niveau de ces traitements.

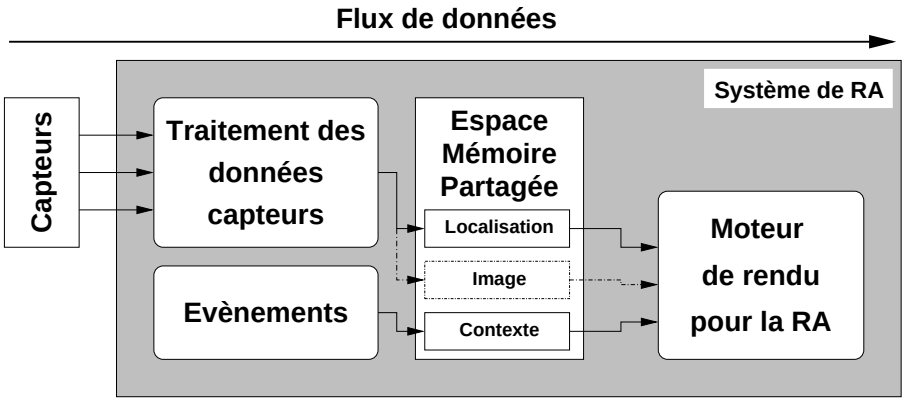



FIG. 2.3 – Architecture globale d'un système de réalité augmentée

### 2.3.2 Architecture du traitement des données capteurs

Cette architecture est conçue dans le cadre d'une application qui doit pouvoir fonctionner aussi bien en réalité augmentée en vision directe qu'en réalité augmentée en vision indirecte. Comme nous le voyons à la figure 2.4 page suivante, chacun des flux de données est horodaté lors de l'acquisition des mesures faites par les capteurs, puisque ceci est capital pour la localisation spatio-temporelle, problématique critique en réalité augmentée en vision directe.

Cette architecture de traitement des informations capteurs met en évidence deux types de composants distincts :

- les *collecteurs* qui se chargent de récupérer les données capteurs et de les horodater, il est à noter qu'il s'agit toujours d'un composant "tampon" entre les capteurs et les traitements qui sont effectués par la suite. En ceci, les collecteurs pourraient s'apparenter aux constituants de la couche d'abstraction matérielle de l'architecture du système Tinmith présenté précédemment (cf section 2.1.3 page 36),
- les unités de *traitements* qui modifient le flux de données afin de résoudre un problème donné, ici de localisation. Dans ces unités, nous pouvons subdiviser le comportement de ces dernières en deux catégories : les unités à traitement unique (**traitement1** sur la figure 2.4 page suivante), et les unités de traitement qui ont pour vocation la fusion des données issues des différents flux (ici **traitement2** et **traitement3** par exemple).

A l'issue des traitements, les résultats sont entreposés dans un espace mémoire partagé qui sert de tampon entre l'acquisition et le traitement des données et l'affichage qui est géré par un autre processus. Une architecture d'un tel type nous avait également orienté vers une première définition de notre système de composants que nous allons présenter.

### 2.3.3 Première architecture de communication entre composants

Cette première architecture était basée sur la constatation que les différents capteurs avaient des rythmes d'acquisition propres et que les traitements associés aux données capturées devaient être asynchrones. Ceci implique qu'il existe au minimum un processus par capteur ainsi qu'un processus général qui se charge de l'affichage.

Comme la première architecture fut développée sous un système de type Unix, une solution s'imposa naturellement pour la communication entre les processus : l'utilisation des pipes. Ceux-ci sont des canaux de communication entre les processus composés de deux descripteurs de fichiers, l'un servant à la lecture, l'autre à l'écriture. De plus, ce système permet de synchroniser à volonté les processus par rapport à l'arrivée de nouvelles données. Nous avons étendu ce concept à chaque unité de traitement pour la transformer en un processus indépendant.

Il s'agit là en réalité d'une architecture classique de type "pipes - filters" couramment répandue dans les systèmes de type Unix. Les composants sont plus spécifiquement nommés des *filtres*. Suivant

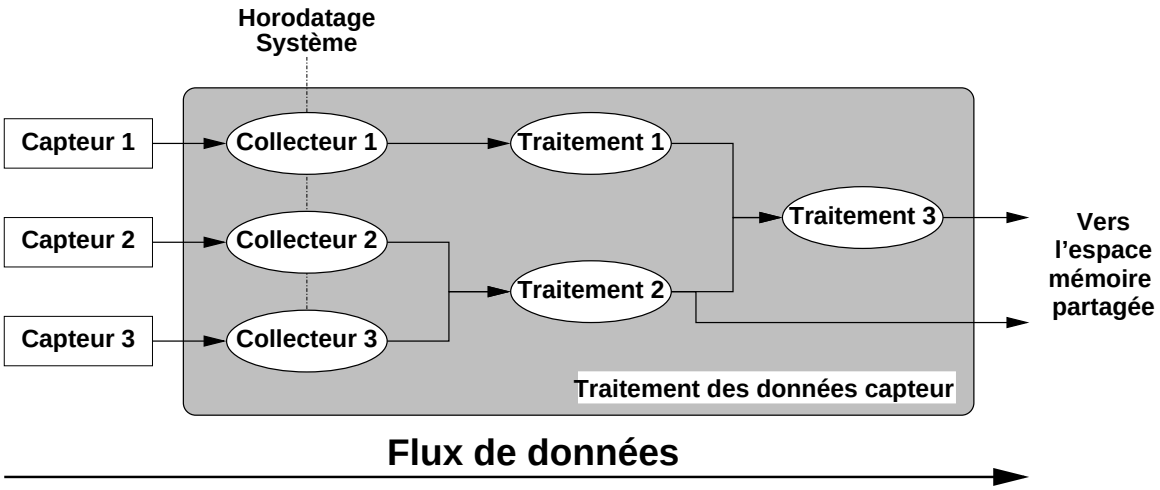



FIG. 2.4 – Architecture du composant de traitement des données capteurs

les préceptes de base de cette architecture, nous avons implémenté une première application.

Cette application était une simulation de système de réalité augmentée en vision directe. Elle s'attachait à mettre en évidence les problématiques liées au recalage dynamique dans le cas où les seules informations permettant d'estimer la localisation en position et en orientation d'un opérateur sont issues d'un capteur magnétique de type Polhemus [Didier, 2002]. Cette architecture était composée de trois composants et dotée, au demeurant, d'une structure linéaire. Elle comportait un collecteur qui était une émulation de capteur polhemus, une unité de traitement qui permettait de filtrer les données et enfin d'un système de mixage d'image virtuelle et d'image de synthèse représentant une scène réelle telle qu'une caméra aurait pu la percevoir en suivant une trajectoire pré-définie. Le composant de filtrage s'est décliné en trois versions différentes, renfermant chacune un modèle différent d'estimation de la position et de l'orientation de l'opérateur.

Cependant, nous allons le voir, une telle architecture n'est pas sans inconvénients, ce qui nous a incité à repenser notre modèle.

### 2.3.4 Discussion sur l'architecture "pipes - filters"

Les avantages et les inconvénients de cette architecture classique sont connus. Nous allons reprendre ces derniers, répertoriés par Garlan et Shaw [Garlan et Shaw, 1993]. Nous commencerons donc par citer les avantages d'une telle solution :

1. Cette architecture permet au concepteur de considérer l'intégralité du comportement du système comme étant la composition simple des comportements de chaque filtre,
2. Cette architecture autorise la réutilisation du code : tous les filtres peuvent être connectés, si les données transmises entre eux sont compatibles avec les traitements effectués,
3. Un système basé sur cette architecture peut-être facilement maintenu et amélioré : de nouveaux filtres peuvent être ajoutés au système existant et les anciens filtres peuvent être remplacés par des nouveaux plus performants,
4. Elle permet certains types d'analyses spécialisées, notamment concernant les inter-blocages (*deadlock*) de processus,
5. Enfin, cette architecture supporte de manière naturelle une exécution concurrentielle : chaque filtre peut être implémenté comme une tâche séparée et est donc potentiellement parallélisable.

Toutefois, une telle architecture comporte également des inconvénients :

1. Les architectures “pipes - filters” mènent souvent à une linéarisation du traitement des données. Dans ce cas, nous obtenons des branches composées de plusieurs traitements qui s’enchaînent. Les bénéfices d’une programmation concurrentielle sur ces traitements sont alors perdus et posent alors des problèmes de performances,
2. Puisque tous les filtres sont indépendants, le développeur doit concevoir ses filtres de manières à ce qu’ils transforment complètement les données d’entrées en données de sortie. En particulier, ce type de systèmes n’est pas adapté pour les applications interactives,
3. Cette architecture est difficile à mettre en oeuvre lorsqu’il faut établir une correspondance entre deux flux séparés mais pourtant en relation,
4. En fonction de certaines implémentations, il est nécessaire d’établir un plus petit dénominateur commun pour les données transmises, ce qui peut ajouter à la complexité de l’écriture des entrées/sorties de chaque filtre. Ceci peut s’avérer coûteux en termes de performances du système,
5. Enfin, cette architecture est très consommatrice d’espace mémoire puisque les données sont répliquées à l’entrée et à la sortie de chaque filtre.

Ainsi cette architecture recèle certains points faibles qui l’ont révélée comme étant inadaptée à un système générique de réalité augmentée, notamment lorsque nous avons commencé à réfléchir à l’utilisation du traitement de l’image pour obtenir les informations d’orientation et de position. En effet, il aurait fallu dupliquer l’image dans chaque filtre de la chaîne de traitement, ce qui aurait été désastreux pour les performances globales du système, tant en espace mémoire qu’en temps d’exécution, ce dernier point s’avérant capital pour un système devant traiter les images à leur cadence d’acquisition.

Toutefois, même si la nature des composants a été repensée depuis, il s’avère que la décomposition fonctionnelle reste valide et que certains des composants que nous avons été amené à développer s’inscrivent dans ces catégories. Nous allons donc présenter le nouveau modèle d’architecture basée composants que nous avons développé, en commençant par son constituant fondamental.

## 2.4 Système de composants pour la réalité augmentée

### 2.4.1 Le composant

Le composant est l’élément central de l’application. D’après les définitions que nous avons donné, ce dernier renferme du code compilé, doit pouvoir fournir une interface et est soumis à une ou plusieurs lois de composition entre les divers composants. De ces trois propriétés fondamentales découlent les autres. Ceci veut dire que l’on devra rapidement aborder l’aspect implémentation des composants. De plus, certains choix en matière de programmation doivent être effectués très tôt.

#### 2.4.1.1 Choix d’implémentation

**Choix du langage de programmation** La première décision concerne le langage d’implémentation. Nous avons opté pour le langage C++ qui permet de profiter des avantages de la programmation objet. De plus, comme les programmes écrits dans ce langage passent par une étape de compilation, ceci nous permet d’obtenir du *code compilé*, qui est la transposition d’un programme écrit en un langage compréhensible par la machine. Ce dernier fonctionne de manière native sur la plate-forme sur laquelle il a été compilé, ce qui est garant de bonnes performances en terme de temps d’exécution.

**Choix du mode de communication** C'est le mode de communication inter-composants qui détermine également les lois de composition que nous détaillerons ultérieurement. Nous avons opté pour le mécanisme signal-slot décrit en section 2.2.2.3 page 42. Celui-ci est en réalité un appel de procédure déguisé, ce qui garantit la synchronisation des traitements lorsqu'elle est nécessaire.

**Choix de la librairie** Comme nous ne souhaitions pas ré-implémenter un nouveau mécanisme signal/slot, nous avons choisi une des librairies parmi celles qui implémentent ce paradigme. Notre choix s'est porté sur la librairie QT. Cette dernière présente plusieurs avantages :

- Cette librairie est *portable*, et fournit une API (Application Programming Interface) qui permet de couvrir bon nombre des fonctionnalités de base des systèmes d'exploitation. A part pour quelques fonctionnalités bas niveau qui sont fortement liées au système d'exploitation, ceci garantit que le code écrit sur une plate-forme peut-être recompilé sur une autre plate-forme sans changements ou presque,
- Le mécanisme signal/slot implémenté permet la connection/déconnection à l'exécution,
- Le modèle d'objets implémenté par QT permet de développer certains objets capables d'auto-introspection, en particulier leur signaux et leurs slots. Or ces derniers constituent l'interface de notre composant. Ces objets seront donc *capables d'exporter leur propre interface*.

#### 2.4.1.2 Représentation d'un composant

En tenant compte des principes énoncés dans la section précédente, un composant est un objet qui contient des signaux, des slots ainsi que des mécanismes d'introspection permettant d'exporter l'interface de ces signaux et de ces slots. Par la suite, nous nous référerons parfois aux composants en employant le terme *objet*. Un objet peut donc être représenté sous la forme de la figure 2.5.

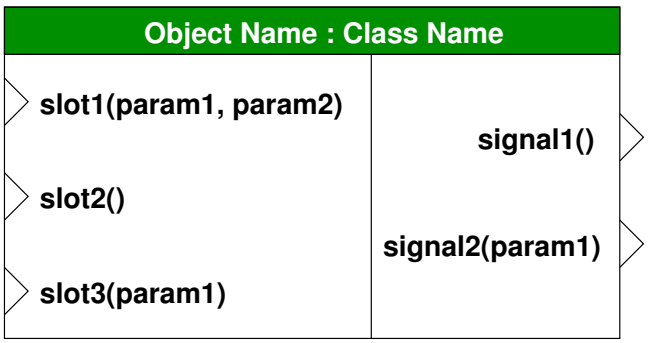



FIG. 2.5 – Vue d'un composant avec ses signaux et slots

Il convient de détailler le mécanisme d'implémentation des objets QT qui nous servirons de composants par la suite, ainsi que d'expliquer de manière plus concrète le mécanisme signal/slot sous jacent.

#### 2.4.1.3 Implémentation d'un composant

L'implémentation d'un objet (appelé aussi bloc atomique) repose sur un ensemble de contraintes relativement simples que nous allons énumérer au fur et à mesure. Celles-ci seront familières pour quiconque ayant déjà travaillé avec la librairie QT. A ces contraintes liées aux fonctions QT nous n'en rajouterons qu'une. Elle sera liée au constructeur. Voici, pour rappel les différents points qui doivent être respectés lors de l'implémentation d'un objet spécifique à la librairie QT.

### La déclaration générale de la classe

Les problèmes d'implémentation liés aux objets est en réalité limité à quelques contraintes sur l'interface employée pour décrire un objet, autrement dit : la déclaration de la classe.

Celle-ci va commencer comme suit :

```
class NewObject : public QObjectInheritor
```

Ici `NewObject` se réfère au nom de la nouvelle classe développée et `QObjectInheritor` se réfère à tout objet héritant de la classe `QObject` de l'API QT, y compris `QObject`. Ainsi, chaque composant doit impérativement hériter de la classe `QObject`, directement ou indirectement. Bien sûr il ne faudra pas oublier d'inclure auparavant les en-têtes nécessaires à la déclaration de la classe dont nous allons hériter.

### Contraintes sur le constructeur

Nous fixons dès à présent la forme du constructeur que nous allons employer pour chacune des classes destinées à être un composant. Le constructeur que chaque composant doit implémenter est le suivant :

```
public NewObject(QObject* parent, const char* name);
```

Un des constructeurs de la classe doit nécessairement posséder les deux premiers paramètres `QObject* parent` et `const char* name`. Si d'autre paramètres suivent, ils doivent impérativement avoir une valeur par défaut en raison du mode d'instanciation des objets qui est utilisé par la suite.

### Implémentation des signaux et des slots

Juste après la déclaration de la classe vient une ligne nécessaire pour pouvoir utiliser le mécanisme signal/slot et l'introspection qui est familière aux utilisateurs de QT [Qt, url] :

```
Q_OBJECT
```

Le bloc de déclaration des slots commence par :

```
public slots :
```

La déclaration d'un slot se fait sous la forme :

```
void nomSlot(typeParam1 param1, typeParam2 param2, ...);
```

La seule contrainte existant sur les slots est le type de retour de la fonction qui doit être le mot-clé "`void`"

De même pour le bloc de déclaration des signaux et la déclaration de ces derniers, nous avons :

```
signals :
void nomSignal(typeParam1 param1, typeParam2 param2, ...);
```

Pour le type de retour d'un signal, nous trouvons la même contrainte que pour les slots.
Les paramètres d'un signal donné permettent de passer des valeurs et des données à un slot qui aura le même ordre et le même type de paramètres.

Le reste de la déclaration de la classe ne change pas, ce qui veut dire que tous les concepts connus en C++ peuvent être exploités tels quels pour la déclaration de la classe. La dernière chose qu'il reste à savoir au niveau des objets concerne l'émission d'un signal dans le code de l'objet qui se fait de la manière suivante :

```
emit nomSignal(param1,param2, ...);
```

### Exemple d'implémentation

Au final, nous avons inclus dans ce document un exemple de déclaration de deux composants dans le listing 2.1. Ce court exemple répertorie les différents points que nous venons d'énoncer.

```
#include <qobject.h> // En-tête définissant la classe QObject
#include <qstring.h>

class Boucle: public QObject //Déclaration de classe, hérite de QObject
{
Q_OBJECT //Utilisé par QT
public:
     // Constructeur aux paramètres figés
     Boucle(QObject* parent = 0, const char* name = 0);

// Déclaration des slots
public slots:
     void setIterations(int n);

// Déclaration des signaux
signals:
     void newIteration(int i);
     void sendToken(QString s);
};


class DisplayInt: public QObject
{
Q_OBJECT
public:
     DisplayInt(QObject* parent = 0, const char* name = 0);

public slots:
     void display(int i);
};
```

Listing 2.1 – Exemple de déclaration de classes représentant des composants

### Mécanisme signal/slot

Lorsque l'on utilise un objet QT, le mécanisme signal/slot permet la connection et la déconnection de ces derniers à la volée. La seule contrainte à la connection est que la signature d'un signal doit correspondre à celle du slot auquel on va faire la connection. Cette signature est en réalité la liste des types des paramètres passés.

Une connection s'effectue de la manière suivante, via la méthode de classe implémentée dans `QObject` :

```
connect(objet1, SIGNAL(signal(...)),
        objet2, SLOT(slot(...)));
```

La déconnection s'effectue de la même manière :

```
disconnect(objet1, SIGNAL(signal(...)),
           objet2, SLOT(slot(...)));
```

`SIGNAL` et `SLOT` sont des macros définies dans l'API QT. Elles prennent pour paramètre une chaîne de caractère donnant le nom et les types des paramètres des signaux et des slots. `objet1` et `objet2` sont des pointeurs vers des objets qui héritent de `QObject`.

En résumé, si l'on reprend l'exemple déclarant deux composants, si l'on instancie un objet `boucle` de la classe `Boucle` et un objet `displayint` de la classe `DisplayInt` et si l'on veut connecter le signal `void newIteration(int i)` de l'objet `boucle` au slot `void display(int i)` de l'objet `displayint`, on écrira :

```
connect(boucle,SIGNAL(newIteration(int)),
        displayint,SLOT(display(int)));
```

Le mécanisme signal/slot sera également mis à profit pour pouvoir configurer et initialiser nos composants.

#### 2.4.1.4 Mécanismes d'initialisation

Chaque composant possède à présent des slots. Ces slots peuvent être utilisés pour configurer et initialiser le composant. Les slots qui seront considérés comme étant des slots d'initialisation auront une signature particulière. En effet, pour des raisons de simplicité, ils ne doivent comporter qu'un seul paramètre dont le type fait partie d'une liste pré-définie. Ces types sont les suivants : `void` (vide) , `int` (entier sur 32 bits), `long` (entier sur 64 bits), `float` (flottant simple précision) , `double` (flottant double précision), `QString` (chaîne de caractères) et enfin `QObject*` (pointeur sur un objet QT).

Par la suite, nous distinguerons deux types d'initialisation :

– les *initialisations de pré-connection*, qui sont réalisées avant que les composants soient mis en relation les uns avec les autres,
– les *initialisations de post-connection*, qui sont réalisées après que les composants soient mis en relation les uns avec les autres, ce qui permet aux initialisations de se propager d'un composant à un autre.

La figure 2.6 page ci-contre détaille plus précisément le mécanisme de propagation des initialisations par rapport au type d'initialisation réalisée.

Les composants, une fois crées et compilés sont stockés dans des librairies dynamiques. Toutefois, il faut pouvoir ré-exploiter les différents composants entreposés dans ces librairies lors d'un chargement de ces dernières à l'exécution. Pour faciliter cette récupération, ces librairies implémentent un mécanisme particulier.

#### 2.4.1.5 Librairie dynamique

Une librairie dynamique employée comme un plugin se doit d'avoir des points d'entrée qui permettent à cette dernière de communiquer avec le programme principal. Ces points d'entrée sont figés définitivement.

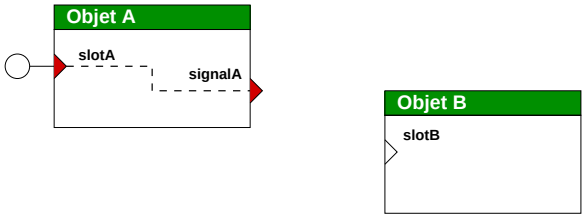
Dans le cas qui nous intéresse, seulement 3 points d'entrée sont à définir pour pouvoir exploiter les objets stockés au sein de cette dernière. Ils sont :

1. une méthode décrivant les objets entreposés dans la librairie,
2. une méthode visant à créer ou instancier un objet de la librairie à la demande,
3. une méthode visant à détruire un objet de la librairie.

Pour faciliter la programmation de ces points d'entrée, nous avons créé un ensemble de trois macros spécifiques déclarées dans un fichier d'en-tête particulier. Le nom du fichier d'en-tête est `metalibrarytoolkit.h`.

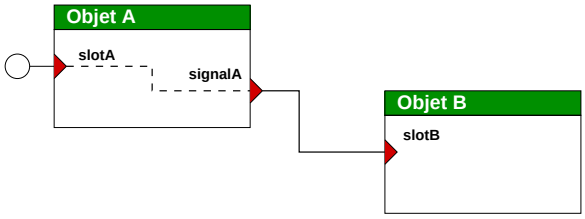
Les macros quant à elles sont :

– `METALIB_BEGIN` qui signale le début d'une librairie,




Nous supposons qu'un objet A contenant un slot *slotA* qui active également un signal *signalA* et qu'un objet B ayant un slot *slotB* sont présent sur la feuille. Les connections entre composants ne sont pas encore réalisées. Si l'on initialise *slotA*, celui-ci va activer *signalA*, qui, n'étant connecté à aucun slot, ne fait rien.

(a) Initialisation de pré-connection




Les connections entre composants sont dorénavant réalisées. Si l'on initialise *slotA*, celui-ci va activer *signalA*. Ce dernier a été connecté à *slotB*, qui est alors à sont tour activé. Les initialisations se propagent bien le long des connexions.

(b) Initialisation de post-connection

FIG. 2.6 – Initialisations et mécanisme de propagation des initialisations

– `METALIB_END` qui signale la fin de la librairie,
– `METALIB_OBJECT(ClassName)` avec `ClassName` le nom d'une des classes développées.

Au final, l'implémentation d'une librairie employant les deux composants dont la déclaration est transcrite dans le listing 2.1 page 53 va receler les lignes écrites dans le listing 2.2.

```
#include "../include/metalibrarytoolkit.h"
#include "boucle.h"

METALIB_BEGIN
METALIB_OBJECT(Boucle)
METALIB_OBJECT(DisplayInt)
METALIB_END
```

Listing 2.2 – Exemple d'implémentation d'une librairie

A présent, nous avons défini l'intégralité de nos composants, de leur interface à la manière dont ils sont stockés et récupérables dans le code compilé. Nous allons maintenant nous intéresser aux lois de composition de nos composants.

## 2.4.2 Loi de composition des composants

Avant de définir les lois de composition auxquelles seront soumis nos composants, il convient d'exposer le concept auquel ces lois sont rattachées : la feuille.

### 2.4.2.1 Le concept de feuille

Le mécanisme de communication entre les objets s'effectue par des connections entre les signaux des uns et les slots des autres. Nous appelons une feuille (ou "sheet") un tel ensemble d'objets et de connections. Une feuille représentée graphiquement pourra donc avoir l'aspect de la figure 2.7.

Toutefois, décrire seulement une liste d'objets et de connections entre ces derniers ne suffit pas dans le cadre d'une application. En effet, un certain nombre de composants, s'ils ont été écrits de

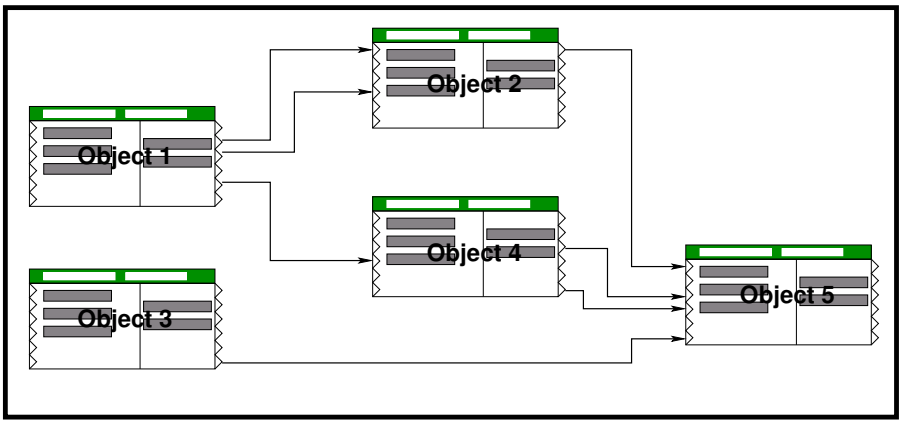



FIG. 2.7 – Feuille représentant des objets connectés entre eux

manière générique, vont nécessiter une étape d'initialisation. Notre feuille décrit également ces initialisations qui peuvent être des pré-connections ou des post-connection, comme nous l'avons déjà écrit plus haut.

Puisque le concept de feuille est maintenant présenté, nous allons exposer les deux lois de composition que nous employons pour nos composants : une loi de composition explicite, et une loi de composition implicite.

#### 2.4.2.2 Loi de composition explicite

La loi de composition explicite est la plus simple puisqu'elle est directement liée au mécanisme signal/slot. Ainsi, une feuille avec son ensemble de connections signal/slots entre composants est le résultat d'une composition explicite. Cette agrégation décrit, en temps que tel, une application puisque nous disposons aussi des mécanismes d'initialisation et configuration des composants.

Il est toutefois possible de faire également des agrégations locales : c'est ce que l'on appelle les macro-composants ou macro-blocs. Le concept de feuille peut-être employé pour décrire ces derniers. Un macro-bloc sera une entité qui, vue de l'extérieur, apparaît comme un objet avec ses signaux et ses slots propres mais qui, à l'intérieur, est en réalité un ensemble de composants communiquant entre eux. Il s'agit effectivement d'une structure de feuille telle que définie précédemment, la différence étant que l'on doit définir les entrées/sorties du macro-bloc par rapport aux composants internes. Cette notion se rapproche de celle des composants du système AMIRE (cf section 2.1.5 page 37).

Une autre application de ces macro-blocs, qui disposent eux aussi de mécanismes d'initialisation, est qu'ils permettent de circonscrire les effets d'une initialisation de post-connection (donc susceptible de se propager le long des connections) au sein du macro-bloc, et donc aux composants qui le constituent.

Toutefois, par rapport au mécanisme d'initialisation, il existe une autre loi de composition qui, par opposition à la précédente, est implicite.

#### 2.4.2.3 Loi de composition implicite

Cette loi de composition fait appel au mécanisme d'initialisation. Elle emploie un slot ayant un paramètre de type pointeur sur un objet de classe `QObject`.

En effet, il est possible, par ce moyen, de donner à un composant un autre composant avec lequel travailler de concert, puisque les composants développés héritent de la classe `QObject`. La possibilité est offerte, de part le modèle d'objets de la librairie QT, de déterminer de quelle sous-classe de `QObject` (via la méthode `bool inherits(QString name)` implémentée dans `QObject`)

l'objet hérite. Ceci permet de vérifier l'implémentation d'interfaces particulières.

Une feuille en tant que telle modélise un état statique d'une application. En effet, si nous nous restreignons à la feuille, les composants n'ont qu'un cycle de vie et la configuration dans laquelle ils travaillent de concert reste figée. Or, les composants ne sont pas forcément nécessaires du début à la fin de l'exécution d'une application, mais peuvent se cantonner à un cycle de vie plus court. Il devient alors nécessaire d'imaginer des mécanismes supplémentaire en vue de représenter de façon adéquate cet aspect dynamique d'une application.

#### 2.4.2.4 Extension du mécanisme de composition

Une feuille n'est finalement qu'une sorte d'état représentant une application à un moment donné. Pour décrire une application dans sa totalité, un ensemble de plusieurs feuilles est nécessaire. Sachant qu'une seule feuille est opérationnelle à un instant donné, il est nécessaire d'y adjoindre un mécanisme de gestion des feuilles. Puisqu'une feuille correspond à un état donné de l'application, le mécanisme de gestion est en réalité un automate (ou "state machine"). Une représentation de notre application correspond alors à la figure 2.8.

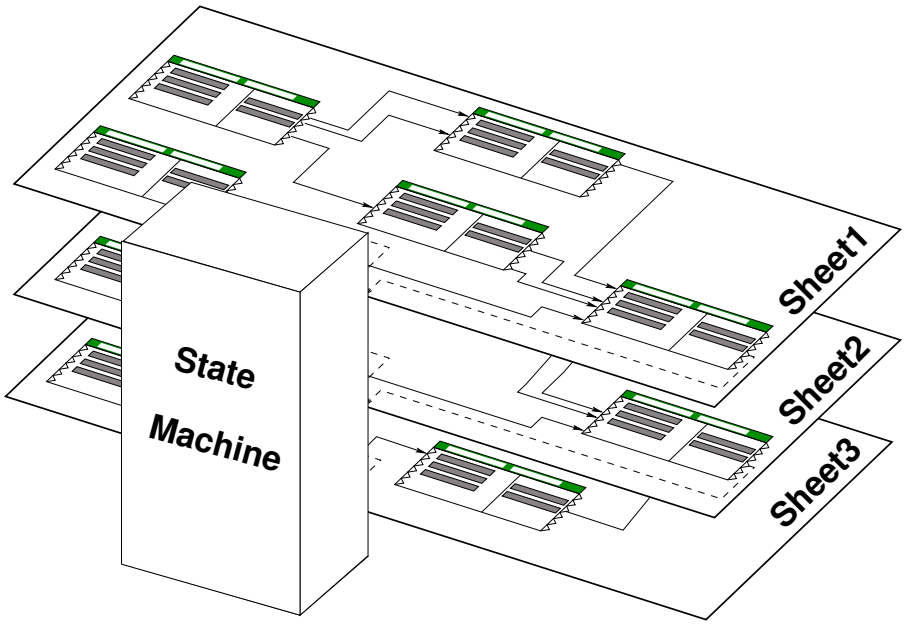



FIG. 2.8 – Vue d'une application : un automate gérant plusieurs feuilles

**Description de l'automate**

L'automate employé ici est similaire en de nombreux points à la description de l'automate de Moore[Moore, 1956] largement employé dans le domaine de la conception de circuit dans les puces électroniques et les micro-contrôleurs. En effet, nous employons un automate à états finis régis par un ensemble de transitions. Les actions de l'application sont fonction de l'état dans lequel l'automate se trouve puisqu'à cet état va correspondre une feuille donnée décrivant les interactions entre les objets. L'automate n'étant ici qu'un moyen de décrire des états et des transitions, plutôt que d'utiliser un formalisme contraignant, nous avons préféré décrire un automate comme étant un ensemble de transitions, chaque transition étant constituée d'un état initial, d'un jeton ("token") de transition et d'un état final. Ainsi le changement d'état est activé quand l'automate reçoit un jeton de la part de

la feuille courante. Enfin, notre automate doit pouvoir contenir un état final, qui lorsqu'il est atteint permet de terminer et d'arrêter proprement l'application.

Évidemment, le principe des feuilles décrit jusqu'alors doit être modifié pour permettre la communication avec l'automate. Ceci est fait en autorisant à un objet particulier par feuille à communiquer avec l'automate. Cet objet envoie les jetons à l'automate qui activeront les changements de feuille le cas échéant. Ces changements de feuille permettront également de gérer le cycle de vie des composants. Par ailleurs, nous adjoignons à ces derniers une propriété qui est la persistance. Les composants seront dits "persistants" si leur cycle de vie dans l'application est le même que celui de l'application. Dans le cas contraire, ils seront dits "non persistants". Nous allons à présent détailler le mécanisme de passage d'un état de l'automate à un autre.

**Mécanisme de passage d'un état à un autre**

Comme nous l'avons expliqué précédemment, à chaque état de l'application correspond une feuille. Ceci implique qu'*à tout moment, il n'y a qu'une seule feuille active qui donne la configuration dans laquelle les composants oeuvrent de concert*. Il convient toutefois de décrire précisément le mécanisme de passage d'un état à un autre, ceci aidant à la compréhension du fonctionnement des applications telles que nous les envisageons.

En supposant qu'une feuille est déjà active et que cette dernière envoie un jeton à l'automate, si ce jeton et cette feuille correspondent à une transition, alors le changement d'état est activé. Celui-ci s'effectue en 5 étapes :

1. *Déconnections :* Il s'agit de la déconnection de l'ensemble des différents liens signaux/slots décrit par l'état initial. A la fin de cette étape, toute communication entre objets est rendue impossible,
2. *Changement de la feuille courante :* La feuille courante change et devient celle décrite par l'état final de la transition qui a été activée,
3. *Initialisations pré-connection :* Avant de connecter les différents liens signaux/slots de la feuille courante, cette étape permet d'initialiser les objets qui seront connectés entre eux. Durant cette phase certains objets dits non persistants sont également instanciés ou détruits,
4. *Connections :* L'ensemble des différents liens signaux/slots décrit dans la feuille courante sont activés. A la fin de cette étape, les objets communiquent à nouveau entre eux,
5. *Initialisations post-connections :* A présent un certain nombre d'initialisations qui doivent se propager au travers de la feuille par le biais des communications signaux/slots sont effectuées.

A l'issue de ce cycle, la feuille courante a donc changé, ce qui a aussi transformé au passage les divers liens de communication existants entre les objets composant l'application. La gestion de plusieurs feuille permet également de gérer le cycle de vie (au sens de cycle de vie dans l'application : de l'instanciation à la destruction) des composants puisque certains ne seront pas utilisés sur toutes les feuilles. Ainsi, dans la liste des initialisations que renferment chaque feuille, il existe des initialisations particulières qui permettent d'instancier et de détruire des composants suivant les besoins. Ceci permet également une économie certaine de la mémoire utilisée par l'application en cours.

Nous venons de préciser les bases de notre système de composants ainsi que les diverses lois de compositions qui entrent en jeu. Au passage, nous nous sommes assurés de la portabilité du système en employant la librairie QT qui possède cette caractéristique. Ce dernier ne fait appel à aucun mécanisme de distribution dans sa version actuelle, toutefois, il est possible de créer un composant qui fera l'interface avec d'autres systèmes localisés sur un ensemble de machines en réseau ce qui permet une extension vers les systèmes distribués. Enfin, tant les macros-blocs que nous avons défini que

la nature même de nos composants permettent d'assurer une granularité variable de ces derniers. Le dernier objectif que nous nous sommes fixé pour ce système est l'élaboration de plusieurs niveaux d'abstraction qui permettent de détacher le rôle du concepteur final de l'application de celui du développeur de composants. Nous allons exposer les mécanismes permettant d'accéder à de tels niveaux d'abstraction.

## 2.5 Niveaux d'abstraction par rapport aux composants

Un niveau d'abstraction intermédiaire peut-être constitué d'un langage de script très simple et interprété permettant de lier les composants ensemble. Toutefois, celui-ci nécessite la connaissance de la syntaxe du langage de script, ce qui ne fournit pas un niveau d'abstraction total par rapport à la programmation pour un utilisateur final. Si nous voulons nous affranchir de la programmation, il nous faudra malgré tout passer par une phase où un fichier renfermant les connections et les diverses initialisations sera interprété par un gestionnaire de composants pour lancer une application. Une des solutions est d'employer le langage de balises XML[Yergeau et al., 2004] (pour eXtensible Markup Language).

### 2.5.1 De l'intérêt d'utiliser XML

XML est devenu un standard pour bon nombre d'applications Web ainsi que les gros systèmes logiciels. XML est un langage de balises qui permet de définir soit même sa propre hiérarchie de balises à l'aide de ce qui est appelé une DTD (pour Document Type Definition). Avec une DTD appropriée, les outils XML standards peuvent être utilisés pour éditer, faire des vérifications de type, interpréter et transformer n'importe quel fichier XML. L'utilisation d'une telle technologie possède plusieurs avantages :

- Un éditeur graphique de DTD peut être employé pour concevoir et maintenir une DTD,
- Une API d'interprétation des fichiers XML permet de construire directement dans la mémoire la structure correspondante au fichier en mémoire, tout en faisant des vérifications automatiques de la syntaxe et de la cohérence des informations,
- La même API permet de manipuler ces structures à l'exécution tout en conservant la cohérence de ces dernières. La structure modifiée peut-être à nouveau enregistrée dans un fichier tout en assurant sa validité par rapport à la DTD,
- Un éditeur graphique peut-être employé pour aider l'utilisateur à concevoir une application sans que ce dernier ait à connaître la syntaxe XML (l'utilisation d'un tel éditeur permet donc de passer à un niveau d'abstraction supplémentaire, précisément celui auquel nous voulons arriver),
- Il est possible de générer de manière automatique de la documentation à partir des fichiers XML en employant le XSL (pour eXtensible Style Language)

De tels avantages ont grandement contribué à la popularité du langage XML. Ce dernier est déjà utilisé dans bon nombre d'applications de réalité augmentée qui veulent employer un système de configuration générique. C'est le cas de DWARF, AMIRE et OpenTracker de la StudierStube. Enfin, la librairie QT contient justement les primitives permettant de manipuler les fichiers XML.

A présent, il convient de spécifier le système de balises que nous allons employer pour décrire les relations entre les composants et la structure de l'application en elle-même.

## 2.5.2 Définition d'un système de balises XML pour la description des applications et des macro-blocs

### 2.5.2.1 Description d'une application

Un document décrivant une application est composée de 5 blocs qui sont :

1. `defines` qui contient une liste de constantes pré-définies [1],
2. `libraries` qui liste les librairies à charger au démarrage,
3. `objects` qui donne la liste des objets (composants) employés dans l'application,
4. `sheets` qui décrit les différentes feuilles composant l'application,
5. `statemachine` qui donne l'automate associé à l'application.

Bien sûr, chacune de ces sections renferme plusieurs autres balises. La hiérarchie de ces balises est représentée dans la figure 2.9. Nous allons à présent donner la liste de toutes les balises avec leur attributs et leur significations.

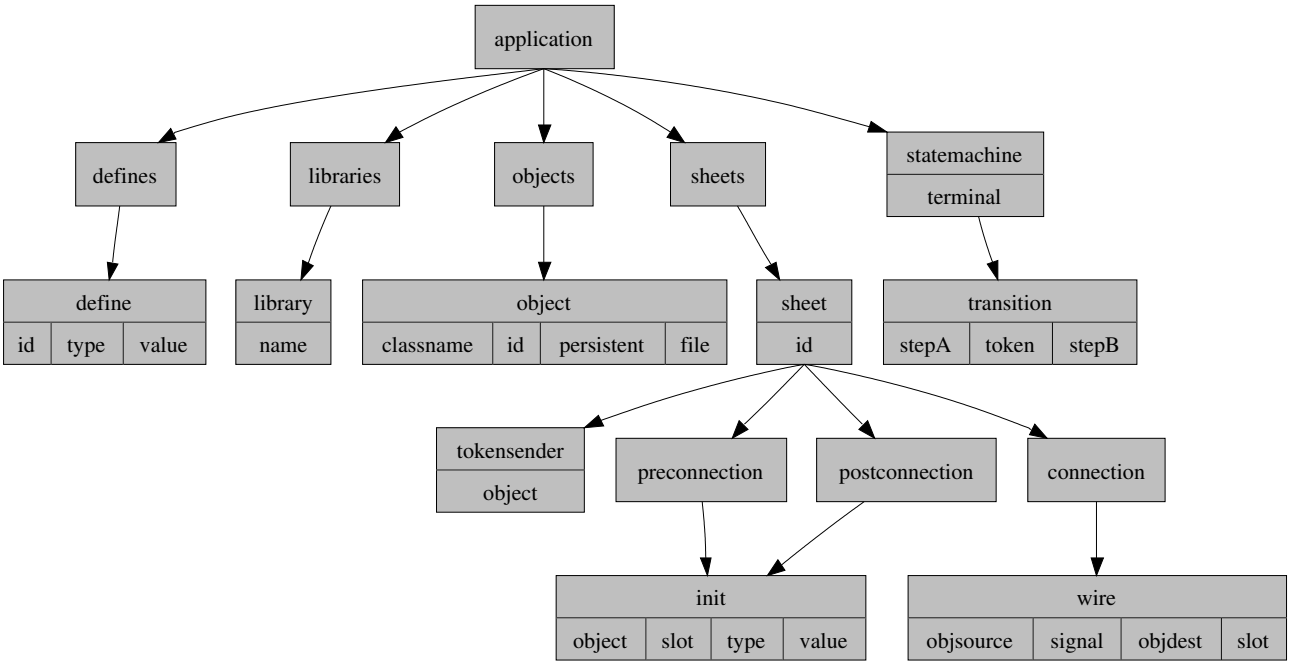



FIG. 2.9 – Organisation des différentes balises d'un fichier XML décrivant une application.

#### Jeu de balises XML pour décrire une application

La table 2.3 page suivante présente la liste de toutes les balises XML employées pour décrire une application. A chaque balise est associée la fonction qui lui correspond par rapport aux concepts que nous avons énoncé précédemment.

#### Jeu d'attributs des balises

Certaines balises du document comportent des attributs permettant de modifier les propriétés de ces dernières. La table 2.4 page 62 dresse la liste de chacun de ces attributs par balise et détaille la modification de propriétés engendrée.

[1] L'utilité de cette section est justifiée dans la section 2.5.2.3 page 63

| Balise | Description |
|---|---|
| `application` | Balise maîtresse du document |
| `defines` | Liste des constantes pré-définies |
| `define` | Une des constantes pré-définies |
| `libraries` | Liste des librairies à charger au démarrage |
| `library` | Une des librairies à charger |
| `objects` | Liste des objets instanciés pour l'application |
| `object` | Un des objets à instancier |
| `sheets` | Liste des feuilles composant l'application |
| `sheet` | Une des feuilles composant l'application |
| `tokensender` | Indique l'objet de la feuille dialoguant avec l'automate |
| `preconnection` | Liste des initialisations de pré-connection |
| `connection` | Liste des connections inter-objets |
| `postconnection` | Liste des initialisations de post-connection |
| `init` | Une des initialisations de composants |
| `wire` | Une des connections entre les composants |
| `statemachine` | Description des transitions constituant l'automate |
| `transition` | Description d'une des transition de l'automate |

TAB. 2.3 – Jeu de balises XML décrivant une application

A cette table, il nous faut ajouter la liste des classes d'équivalence des attributs employés, principalement pour les identifiants. En effet, certains attributs de balises font référence à des identifiants déclarés en tant qu'attributs d'autres balises. Nous allons les classer en deux catégories : les attributs qui génèrent l'identifiant et les attributs qui exploitent l'identifiant généré. La liste de ces équivalences est donnée dans la table 2.6 page 64.

Maintenant que nous avons exposé la manière dont une application est formalisée dans un document XML, nous allons voir la formalisation spécifique aux macro-blocs.

#### 2.5.2.2 Description d'un macro-bloc

Un document décrivant un macro-bloc est composée de 5 blocs qui sont :

1. `defines` qui contient une liste de constantes pré-définies,
2. `libraries` qui liste les librairies à charger au démarrage de l'application,
3. `objects` qui donne la liste des objets employés dans le macro-bloc,
4. `sheet` qui décrit la feuille composant le macro-bloc,
5. `signals` qui donne la liste des signaux sortant du macro-bloc,
6. `slots` qui donne la liste des slots entrant du macro-bloc.

Bien sûr, chacune de ces sections renferme plusieurs autres balises. La hiérarchie de ces balises est représentée dans la figure 2.10 page suivante. Si certaines parties de ce diagramme sont similaires avec la figure 2.9 page précédente ( comme c'est le cas pour les ensembles de balises dépendant de `defines`, `libraries`, `objects`) , certaines modifications sont en revanche apportées pour caractériser les macro-blocs que nous allons maintenant indiquer.

**Changements par rapport à la formalisation d'une application**

| Balise | Attribut | Description |
|---|---|---|
| `statemachine` | `terminal` | Spécifie le nom de l'état terminal de l'automate |
| `define` | `id`<br>`type`<br>`value` | Donne le nom de la constante pré-définie<br>Type de la constante pré-définie<br>Valeur de la constante pré-définie |
| `library` | `name` | Nom ou chemin relatif de la librairie par rapport au fichier XML |
| `object` | `classname`<br>`id`<br>`persistent`<br>`file` | Nom de la classe de l'objet<br>Nom ou identifiant de l'objet<br>Indique si le cycle de vie de l'objet est celui de l'application<br>Si cet attribut est présent, alors l'objet est un macro-bloc dont le fichier XML associé est pointé par la valeur de cet attribut. |
| `sheet` | `id` | Nom ou identifiant de la feuille |
| `transition` | `stepA`<br>`stepB`<br>`token` | Identifiant de feuille (état initial de la transition)<br>Identifiant de feuille (état final de la transition)<br>Jeton qui provoque le passage de l'état `stepA` à l'état `stepB` |
| `tokensender` | `object` | Identifiant de l'objet servant d'intermédiaire entre la feuille et l'automate |
| `init` | `object`<br>`slot`<br>`type`<br>`value` | Identifiant de l'objet à initialiser<br>Nom du signal par lequel se fait l'initialisation<br>Type du paramètre d'initialisation<br>Valeur du paramètre d'initialisation |
| `wire` | `objsource`<br>`signal`<br>`objdest`<br>`slot` | Identifiant de l'objet émetteur<br>Nom du signal de l'objet émetteur<br>Identifiant de l'objet récepteur<br>Nom du slot de l'objet récepteur |

TAB. 2.4 – Liste des balises et de leurs attributs respectifs

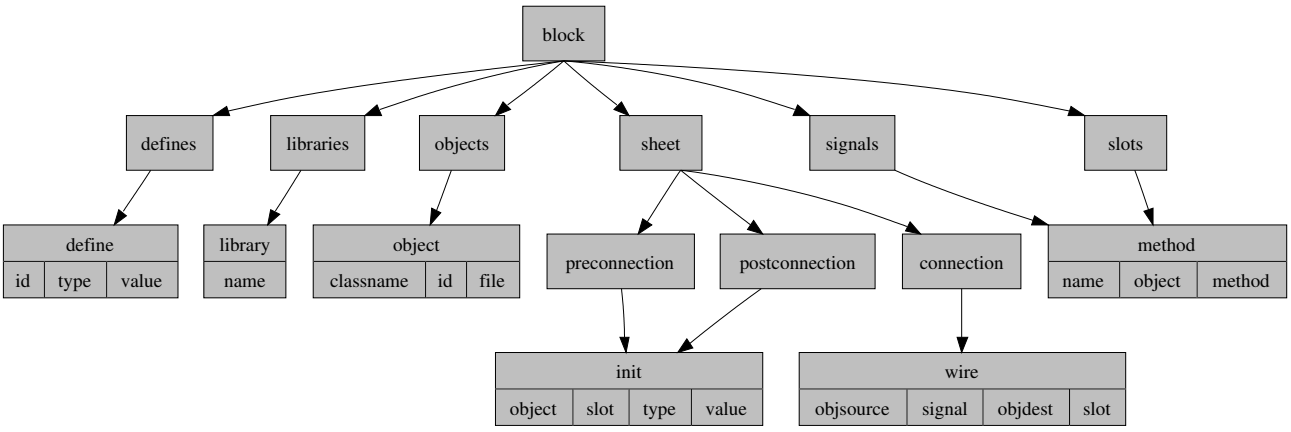



FIG. 2.10 – Organisation des différentes balises d'un fichier XML décrivant un macro-bloc.

Un macro-bloc a, comme nous l'avons vu, la même structure interne qu'une feuille. Nous conservons d'ailleurs les mêmes mécanismes de pré-connection, de connection et de post-connection. Simplement, là où pour une application nous pouvons avoir plusieurs feuilles, les macro-blocs n'en contiennent qu'une seule. De plus, nous avons besoin de décrire les entrées/sorties de notre macro-bloc de manière à ce que ce dernier soit perçu comme une boîte noire. Les changements principaux sont donc les suivants :

- Changement de la balise `application` maîtresse du document en la balise `block`,
- Ajout des balises `signals` et `slots` qui décrivent les entrées/sorties du macro-bloc,
- Suppression de la balise `sheets` puisqu'il n'y a plus qu'une seule feuille,
- Suppression de l'attribut `persistent` des balises `object` puisque ces dernier vont hériter cet attribut directement du macro-bloc,
- Suppression de l'attribut `id` de la balise `sheet` puisque cette dernière est unique dans un macro-bloc,
- Suppression de la balise `tokensender` associée à la balise `sheet` puisque le macro-bloc est destiné à être intégré dans une feuille,
- Ajout de la balise `method` qui permet de faire le lien entre une entrée/sortie du macro-bloc et une entrée/sortie d'un objet à l'intérieur du macro-bloc.

Au final, nous obtenons le tableau 2.5 qui recense les nouveaux attributs pour les balises qui ont changé.

| Balise | Attribut | Description |
|---|---|---|
| `sheet` | | Tout attribut a disparu. |
| `object` | `classname` | Nom de la classe de l'objet |
| | `id` | Nom ou identifiant de l'objet |
| | `file` | Si cet attribut est présent, alors l'objet est un macro-bloc dont le fichier XML associé est pointé par la valeur de cet attribut. |
| `method` | `name` | Nom de l'entrée/sortie. Cette chaîne de caractères est structurée de la même manière que le nom d'un signal ou d'un slot d'un objet. |
| | `object` | Identifiant de l'objet interne au macro-bloc qui se chargera de l'entrée/-sortie réelle. |
| | `method` | Nom du signal ou du slot de l'objet `object` qui fait l'entrée/sortie réelle. |

TAB. 2.5 – Liste des balises et de leurs attributs respectifs spécifiques aux macro-blocs

Que le fichier soit un macro-bloc ou une application, nous allons examiner en détail le mécanisme des initialisations à l'intérieur des feuilles qui est commun aux deux et dont certaines particularités ne peuvent ressortir en énonçant uniquement la formalisation XML.

#### 2.5.2.3 Spécificités du mécanisme d'initialisation

Le mécanisme d'initialisation (les balises de type `init`) a un comportement spécifique en fonction des valeurs des différents attributs. Il convient donc de détailler quelque peu ces comportements. Outre l'utilisation normale d'une initialisation, il est possible d'utiliser ce mécanisme en vue d'établir des compositions implicites. De plus, il permet d'instancier et de détruire les objets non-persistants.

**Dans le cadre d'une initialisation normale**
L'attribut `object` a pour valeur un identifiant d'objet. L'attribut `type` peut prendre plusieurs valeurs : `bool, integer, float, double, string, void, define`. Si le type est `define`,

les attributs `type` et `value` sont substitué par les valeurs des attributs `type` et `value` de la balise `define` dont l'identifiant `id` est donné par l'attribut `value` avant modification. Dans le cas normal, la valeur stockées dans l'attribut `value` est passée à l'objet identifié par `object` via le slot défini par `slot`.

**Dans le cadre d'une composition implicite**
L'attribut `object` a pour valeur un identifiant d'objet. L'attribut `type` prend la valeur `object`. Le pointeur se référant à l'objet dont l'identifiant est donné par l'attribut `value` est passé en paramètre au slot identifié par `slot` de l'objet identifié par `object`.

L'objet identifié par l'attribut `value` est alors composé implicitement avec le composant identifié par l'attribut `object`.

**Dans le cadre d'une instanciation et d'une destruction d'un objet non-persistant**
L'attribut `object` a pour valeur le mot-clé `this`. L'attribut `type` doit nécessairement prendre la valeur `object`. A partir de là, deux comportements sont possibles suivant la valeur de l'attribut `slot` :

- Si `slot` a pour valeur `instanciate`, alors l'objet qui aura pour identifiant la valeur de l'attribut `value` sera instancié dans le cas où il est non persistant et où il n'est pas déjà instancié,
- Si `slot` a pour valeur `destroy`, alors l'objet qui aura pour identifiant la valeur de l'attribut `value` sera détruit dans le cas où il est non persistant et où une instance de l'objet existe déjà.

| Attribut générateur | Attributs équivalents |
|---|---|
| `object :id` | `tokensender :object, init :object, wire :objsource, wire :objdest , method :object` |
| `sheet :id` | `statemachine :terminal, transition :stepA , transition :stepB` |

TAB. 2.6 – Table d'équivalence des attributs

A l'issue de cette formalisation, nous savons comment décrire le fonctionnement général d'une application grâce à un document XML.

Une fois les objets élémentaires et les briques développées, il suffit d'écrire les documents XML associés aux macros-blocs et à l'application dont nous pouvons trouver des exemples à l'annexe B page 185. Ensuite, ces fichiers seront lus et interprétés par un moteur d'exécution (dit *runtime*) qui se chargera de l'exécution de l'application décrite. Dans le souci de présenter une vision complète du système, il convient d'étudier la manière dont ce programme gère et interprète les fichiers XML.

### 2.5.3 Présentation du moteur d'exécution (runtime)

Dans cette section, nous allons présenter de manière succincte l'ensemble des classes développées pour interpréter et exécuter les fichiers de description d'une application. Nous lierons un certain nombre de classes avec les données stockées dans les fichiers XML. Puis nous montrerons comment s'effectue la séquence d'import et d'exécution de l'application. Enfin, nous nous intéresserons à l'interprétation des constantes prédéfinies et des macro-blocs.

#### 2.5.3.1 Diagramme des classes

L'ensemble des classes développées se subdivise en deux familles de classes :

– Les classes qui traduisent une balise XML et les attributs de cette dernière,

– les classes d'interprétation et d'exécution qui vont instancier les précédentes puis les exploiter.

Le diagramme des classes qui indique également à quelle famille chacune appartient est présenté dans la figure 2.11. Une correspondance est établie entre les balises élémentaires et les classes développées au sein de la table 2.7. Certaines de ses classes s'autoréférencent dans un dictionnaire statique en fonction de leur identifiant. Cela permettra de les retrouver directement en employant leur identifiant comme clé de recherche. C'est le cas des classes `MetaObject`, `MetaDefine`, `MetaIO` et `MetaSheet`. Les entrées dans ce dictionnaire sont automatiquement effacées dès que l'objet est détruit.

Nous allons à présent voir comment ces classes oeuvrent ensemble lors de la séquence d'import et d'exécution de l'application.

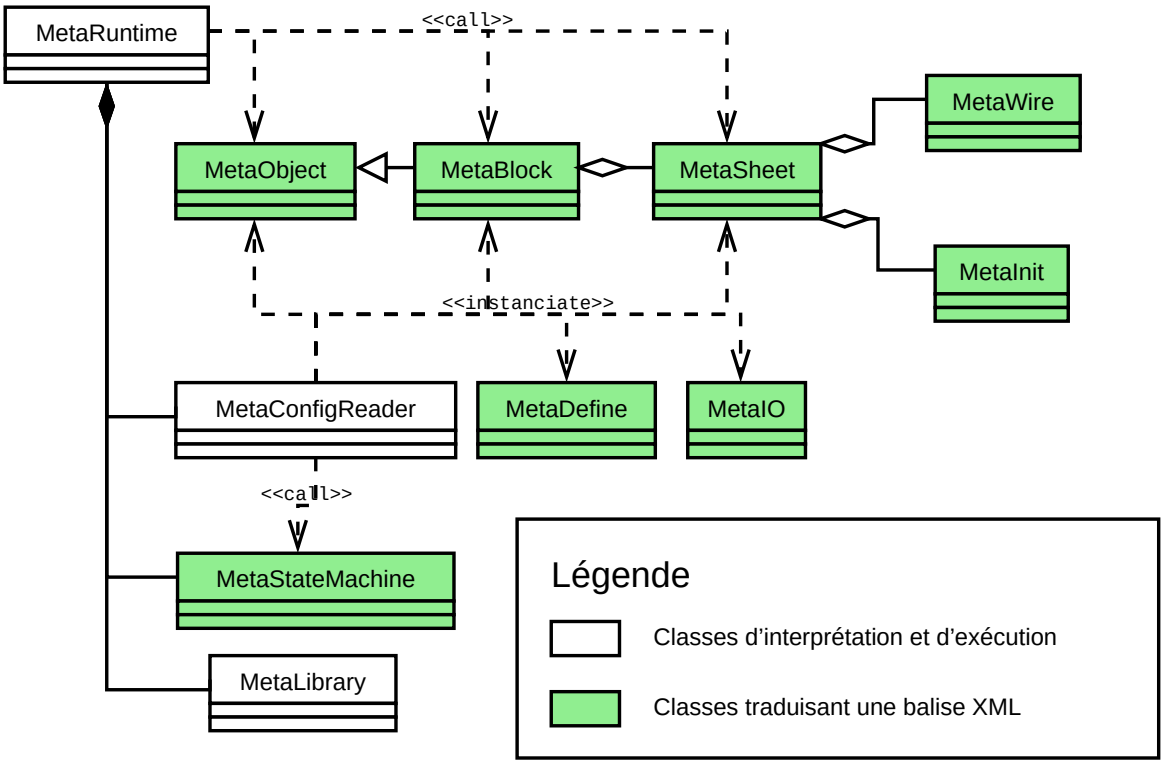



FIG. 2.11 – Diagramme des classes développées pour le moteur d'interprétation et d'exécution

| Balise XML | Classe |
|---|---|
| object | MetaObject (objet simple) |
| object | MetaBlock (macro-bloc) |
| define | MetaDefine |
| sheet | MetaSheet |
| wire | MetaWire |
| init | MetaInit |
| statemachine | MetaStateMachine |
| method | MetaIO |

TAB. 2.7 – Correspondance classes / balises XML

#### 2.5.3.2 Séquence d'actions d'import et d'exécution

La séquence d'import et d'exécution d'une application est la suivante :

1. Chargement du fichier XML décrivant l'application,
2. Interprétation de la section `libraries`,
3. Interprétation de la section `objects`,
4. Interprétation de la section `sheets`,
5. Interprétation de la section `statemachine`,
6. Lancement de l'application.

A l'issue du chargement du fichier XML décrivant l'application, nous obtenons une structure d'arbre composée des types élémentaires utilisés par la librairie QT pour manipuler les fichiers XML. Il faut donc traduire cette structure dans des types qui reflètent le plus fidèlement possible les données que nous souhaitons exploiter. C'est à ce moment qu'interviennent les 4 étapes suivantes.

L'interprétation de la section `libraries` est sans doute la moins complexe de toutes puisqu'elle se contente de stocker les chemins d'accès aux librairies en vue d'un chargement ultérieur. Les 3 étapes suivantes vont faire l'objet d'une étude plus détaillée.

##### Interprétation de la section `objects`

Cette interprétation est menée en plusieurs temps :

1. Instanciation des balises `object` en classes `MetaObject` ou `MetaBlock`,
2. Chargement des librairies contenant les classes compilées,
3. Si certains objets sont de type "persistent" alors ils sont instanciés.

Le mécanisme spécifique de lecture des objets de type `MetaBlock` est expliqué section 2.5.3.3 page ci-contre. Quand au procédé d'instanciation, il permet de faire le lien entre les objets de type `MetaObject` et les véritables entités sur lesquelles s'effectuent les connections/déconnections et initialisations, à savoir des objets dont la classe hérite de `QObject`.

Une fois la section `objects` interprétée, nous pouvons passer à l'interprétation de la section `sheets` qui contient la description des liens entre les objets.

##### Interprétation de la section `sheets`

Il s'agit de la section qui est la plus lourde en opérations d'initialisation, essentiellement parce que la feuille est le concept le plus complexe. Cette séquence est composée des étapes suivantes :

1. Interprétation de la section `defines`
2. Interprétation de chaque sous-section `sheet` qui comprend :
   (a) L'interprétation des parties `preconnection` et `postconnection`,
   (b) L'interprétation de la partie `connection`

L'interprétation de la section `defines` traduit les balises `define` en objets `MetaDefine`. De même, chaque sous-section `sheet` est instanciée en sa contrepartie `MetaSheet` qui stockera les données associées aux balises `wire` dans des objets `MetaWire`. Au passage, les mécanismes liés aux macro-blocs modifie quelque peu la complexité de la transformation comme le montre la section 2.5.3.3 page suivante.

Les interprétations des initialisations des parties `preconnection` et `postconnection` se traduiront par la production d'objets `MetaInit`. Le mécanisme de substitution des constantes pré-définies intervient à ce niveau. En effet, si la balise `init` contient un attribut `type` qui a pour valeur `define` alors, on va rechercher l'objet `MetaDefine` qui a pour identifiant la valeur de l'attribut `value` de notre balise `init`. Cet objet nous fournit un nouveau couple de valeurs (`type, value`) que l'on substitue à l'ancien dans la balise `init`. Ceci en fait un mécanisme de substitution proche de ceux employés par le préprocesseur C ou C++ lorsqu'ils traitent la directive `#define`.

Nous allons à présent nous intéresser aux problèmes posés par les macro-blocs et la solution technique apportée dans le moteur d'exécution.

#### 2.5.3.3 Exploitation des macros-blocs

Un macro-bloc, nous le rappelons est un regroupement de plusieurs composants qui communiquent entre eux à l'intérieur du macro-bloc. Vu de l'extérieur, un macro-bloc doit ressembler à un composant, c'est à dire qu'il ne doit rien laisser percer de sa structure interne. Ceci implique que ce sont ses points d'entrée/sortie qui sont connectés ou initialisés.

##### Le problème de la récursivité

Le problème auquel l'on devra faire face en employant les macro-blocs est le problème de la récursivité. En effet, un tel système de regroupement des composants en macro-blocs prend encore plus de valeur si on autorise les groupements de groupements, et ainsi de suite. Nous souhaitons donc inclure des macro-blocs à l'intérieur de macro-blocs. Pour cela, il faut introduire des mécanismes qui permettent cette récursivité.

L'autre problème qui est posé est la manière dont sont référencés les objets. Il n'existe qu'un seul dictionnaire pour l'ensemble des objets élémentaires. Il faut donc s'assurer que les noms d'objets définis à l'intérieur des macro-blocs (que l'utilisateur du macro-bloc n'est pas censé connaître) ne vont pas écraser la définition d'autres objets définis dans le corps principal de l'application. Ce raisonnement est valable également pour les constantes pré-définies.

Par conséquent, nous avons introduit une notion similaire aux `namespace` en C++ pour référencer les objets contenus dans les macro-blocs. Cet adressage se fait d'une manière relativement simple. Si on prend l'exemple de la figure 2.12 page suivante, nous avons trois objets à enregistrer :

– l'objet A de la feuille,
– le bloc B de la feuille,
– l'objet A du bloc B.

Sachant que l'on ne sait pas à l'avance dans quelle feuille va s'insérer le bloc `B`, nous ne pouvons pas nommer nos objets de manière à éviter d'avoir deux fois les mêmes noms. Le référencement se fera de la manière suivante pour les identifiants : comme l'objet A du bloc B est dans B, on va le nommer `B::A`. Si nous avons des blocs à l'intérieur des blocs, nous pouvons étendre cette convention de nommage récursivement.

##### Conséquences au niveau de la transcription des balises en type élémentaires

Cette solution va être mise en oeuvre à plusieurs niveaux différents. Le premier concerne la lecture de l'application principale, notamment l'interprétation de la section `sheets`. La deuxième concerne la lecture de la feuille associée à un macro-bloc. Pour ceci, il convient également d'examiner la manière dont est interprété le fichier XML contenant un macro-bloc.

### Interprétation d'un fichier XML de description de macro-blocs

Cette interprétation se fait en plusieurs étapes qui sont les suivantes :

1. Chargement du fichier XML décrivant le macro-bloc,
2. Interprétation de la section `libraries`,
3. Interprétation de la section `objects`,
4. Interprétation de la section `signals`,
5. Interprétation de la section `slots`,
6. Interprétation de la section `defines`,
7. Interprétation de la section `sheet`.

Beaucoup d'étapes sont similaires à celles présentées dans la section 2.5.3.2 page 66 concernant la lecture et l'interprétation des fichiers qui décrivent une application. Aussi, nous ne présenterons que celles qui diffèrent des autres, à savoir les interprétations des sections `sheet`, `signals` et `slots`.

L'interprétation des sections `signals` et `slots` va simplement transcrire le contenu des balises `method` dans sa contrepartie, l'objet `MetaIO`. Celui-ci stockera le nom du signal ou du slot contenu dans `name` préfixé à l'aide du namespace associé au bloc ainsi que le nom de l'objet (qui peut lui-même être un macro-bloc) sur lequel est effectué la connection en interne qui sera donc aussi préfixé avec le namespace.

Une fois cette étape réalisée, on va interpréter la section `sheet`. Dans le cas où les objets référencés dans les balises `init` et `wire` sont des macro-blocs, on va substituer récursivement les couples (objet, signal/slot) jusqu'à ce que l'objet du couple soit un objet élémentaire et non pas un macro-bloc.

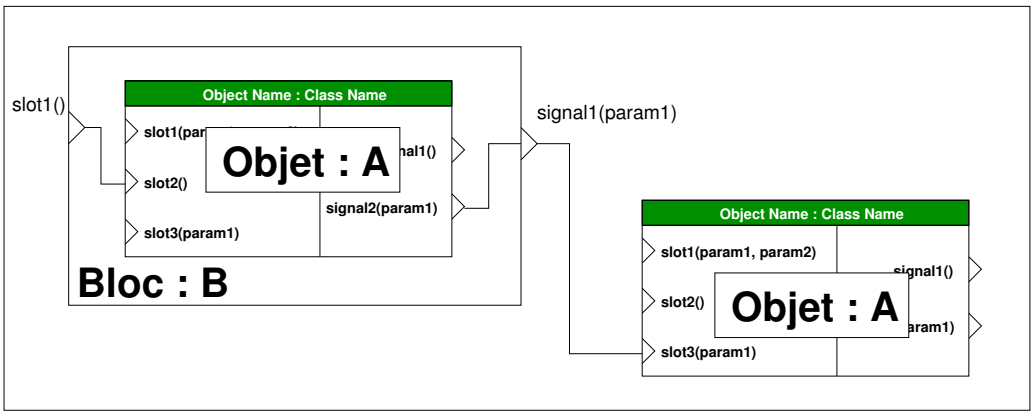



FIG. 2.12 – Feuille liant un objet et un bloc dans lequel est encapsulé un objet

### Mécanisme de lecture des feuilles de l'application principale

Le mécanisme d'interprétation des feuilles dans l'application principale va également effectuer les opérations de substitution des paramètres dans les balises `init` et `wire`. Ainsi si l'on reprend l'exemple de la figure 2.12, si dans le document XML nous trouvons :

```
<init object="B" slot="slot1()" ... />
```

Ceci sera interprété comme une initialisation sur le slot `slot2()` de l'objet `B::A`.
De même si nous avons la connection :

```
<wire objsource="B" signal="signal1(param1)" objdest="A" slot="slot3(param1
```

Celle-ci sera interprétée directement comme une connection entre le signal `signal2(param1)` de l'objet `B::A` et le slot `slot3(param1)` de l'objet `A`.

Enfin, si nous avions eu un ensemble d'objets `A` et `C` à l'intérieur du bloc `B` ce qui serait écrit à l'intérieur du fichier de description du bloc par :

```
<wire objsource="A" signal="..." objdest="C" slot="..."/>
```

Ceci est interprété au final comme une connection directe entre les objets `B::A` et `B::C`.

Pour terminer ce tour d'horizon sur les macro-blocs, nous décrirons également le mécanisme d'instanciation de ces derniers.

**Instanciation des macro-blocs**
La séquence d'actions qui permettent l'instanciation des macro-blocs est la suivante :

1. Instanciation des objets (ceci concerne aussi les macro-blocs) interne au bloc,
2. Initialisations de pré-connection de la feuille associée au bloc,
3. Connection des divers objets du bloc,
4. Initialisations de post-connection de la feuille associée au bloc.

Un tel mécanisme garantit que l'instanciation d'un macro-bloc dans un autre macro-bloc ne perturbera pas le comportement du macro-bloc de plus haut niveau.

Par contre, dans le cas de l'instanciation d'un macro-bloc (qui n'a pas été déclaré comme étant persistant) lors d'une initialisation de post-connection, les initialisations de post-connection situées au niveau de ce macro-bloc risquent d'influer sur le comportement des objets liés à ce dernier. C'est pourquoi, cette possibilité est à éviter.

Nous venons de présenter la première version du moteur d'exécution dédié au prototypage rapide d'application. Après avoir présenté les concepts de base, nous avons formalisé le fonctionnement d'une application, ce qui a abouti à un document XML décrivant les diverses relations entre les constituants de l'application. Nous avons expliqué comment réaliser une implémentation des composants. Enfin, comme nous avons introduit le concept de macro-bloc, nous avons présenté les mécanismes de gestion spécifiques introduit par ce concept supplémentaire. Nous avons ainsi obtenu un niveau d'abstraction intermédiaire où la connaissance du langage de programmation n'est plus nécessitée par l'utilisateur.

Afin d'obtenir un niveau d'abstraction total qui sépare la conception de l'application du développement des composants, nous avons créé un éditeur graphique capable d'écrire et de manipuler les fichiers XML dont nous venons de détailler la spécification. Ceci permet à l'utilisateur final de manipuler des métaphores graphique au lieu des fichiers XML.

### 2.5.4 Éditeur graphique

L'éditeur graphique que nous avons développé respecte les concepts qui ont été exposés jusqu'à présent. Nous nous contenterons d'esquisser les fonctionnalités de ce dernier ainsi que la manière dont les composants sont gérés à travers cette interface graphique. Il permet d'éditer les feuilles qui composent l'application (figure 2.13 page 71) ainsi que l'automate qui gère le passage d'une feuille (qui est également un état de l'application) à une autre (figure 2.15 page 72).

La seule chose dont l'éditeur ait besoin pour mener à bien ce travail d'édition et pour connaître les interfaces des composants, est l'ensemble des librairies compilées qui renferme les objets. Ainsi, il n'est pas nécessaire de connaître le code source des composants pour pouvoir éditer une application. L'aspect final de ces derniers change sensiblement par rapport à la représentation que nous leur avions donnée initialement, tout en gardant les grand principes comme le montre la figure 2.14 page suivante.

Cet éditeur possède également son propre format de fichiers XML (qui possède par voie de conséquence une autre DTD - cf annexe A.4 page 182) pour stocker les propriétés graphiques associées au entités graphiques manipulées. Ceci permet, lors de la sauvegarde du projet, de sauver également le positionnement des composants et des états de l'automate pour une édition ultérieure. Ainsi, deux fichiers sont en réalité sauvegardés : un qui centralise les propriétés graphique qui sont superflues du point de vue du moteur d'exécution et un qui renferme les informations que nous avons détaillé précédemment (cf section 2.5.2 page 60).

L'utilisation de cet éditeur graphique nous permet d'atteindre le niveau d'abstraction escompté pour notre système de programmation orientée composants puisque la manipulation de cette application ne nécessite pas de connaissances particulières, ni en programmation, ni en XML.

## 2.6 Conclusion

Dans ce chapitre, nous venons de présenter un système de composants et ses infrastructures logicielles qui nous permettent de profiter pleinement de la programmation orientée composants pour effectuer du prototypage rapide d'applications de réalité augmentée.

Ce système est à *granularité variable* de part le système adopté qui garantit des composants initiaux de taille variable. Au besoin, ces composants peuvent être agrégés en macro-blocs, ce qui permet de rajouter une gradation supplémentaire au niveau de granularité.

Le développement des composants s'appuyant sur la librairie multi-plateforme QT, il nous est possible de fournir un système *portable* d'une plate-forme à une autre.

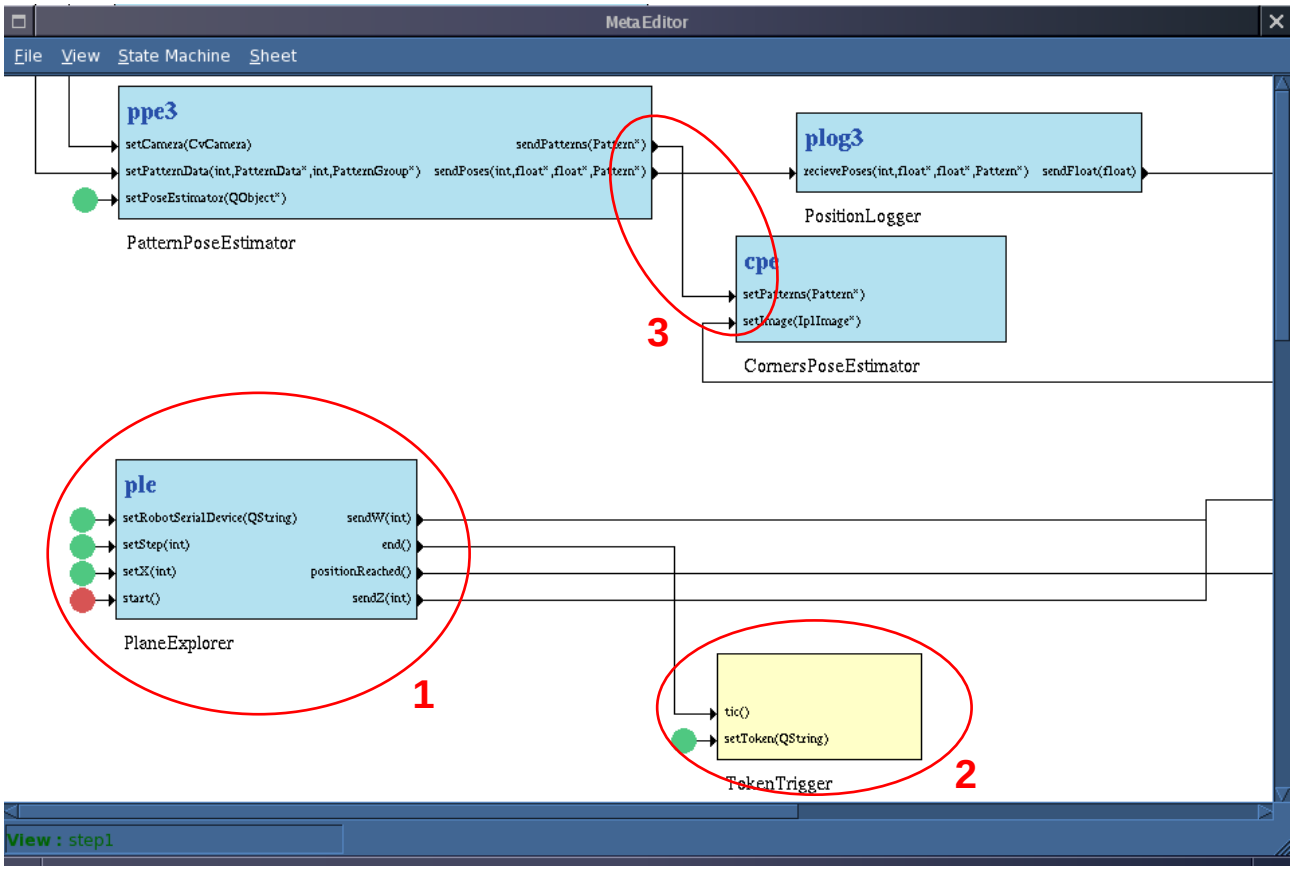
Ce système est optimisé pour les communications inter-composants au sein d'un seul système grâce au mécanisme signal-slot adopté qui est équivalent aux appels de procédures classiques. Toutefois, il est possible de l'*ouvrir vers les systèmes distribués* avec la création de composants spécifiques d'interfaçage avec le réseau.

De plus, grâce à l'utilisation d'un automate, il est possible de finement *contrôler le cycle de vie des divers composants* ainsi que les différentes configurations des connections entre ces derniers, ce qui permet d'ajouter à la *flexibilité* du système.

Enfin, les utilisations conjointes d'un fichier constitué d'un système de balises XML spécifiques et d'un éditeur graphique spécialisé nous permettent de concevoir les applications de réalité augmentée suivant des *niveaux d'abstraction différents*. Ceci offre la possibilité à un utilisateur qui ne possède pas les compétences requises en programmation de créer malgré tout une application de réalité augmentée.

Dans cette architecture générique basée composants, nous avons également dégagé un certain nombre de fonctionnalités propres aux systèmes de réalité augmentée, tant au niveau du traitement des données issues des capteurs qu'au processus final de fusion du monde réel avec les entités virtuelles. Il nous suffira, à présent, de développer le ou les ensembles de composants oeuvrant de concert pour réaliser chacune de ces tâches particulières.

Parmi les fonctionnalités que notre système doit couvrir plus particulièrement, nous comptons l'acquisition des données, le filtrage, le traitement et la fusion des diverses données issues des capteurs qui donneront plusieurs exemples d'application de notre architecture (sections 3.4.1.2 page 98 et 3.4.2.6 page 102), le mixage des informations capteur avec les informations virtuelles contextuelles, les moyens d'interactions physiques entre l'utilisateur et le système de réalité augmentée, le




**Légende**

1. Composant standard (en bleu), les disques représentent les initialisations,
2. Composant autorisé à communiquer avec l'automate (en jaune),
3. Connection entre 2 objets

FIG. 2.13 – Interface graphique en mode d'édition des feuilles

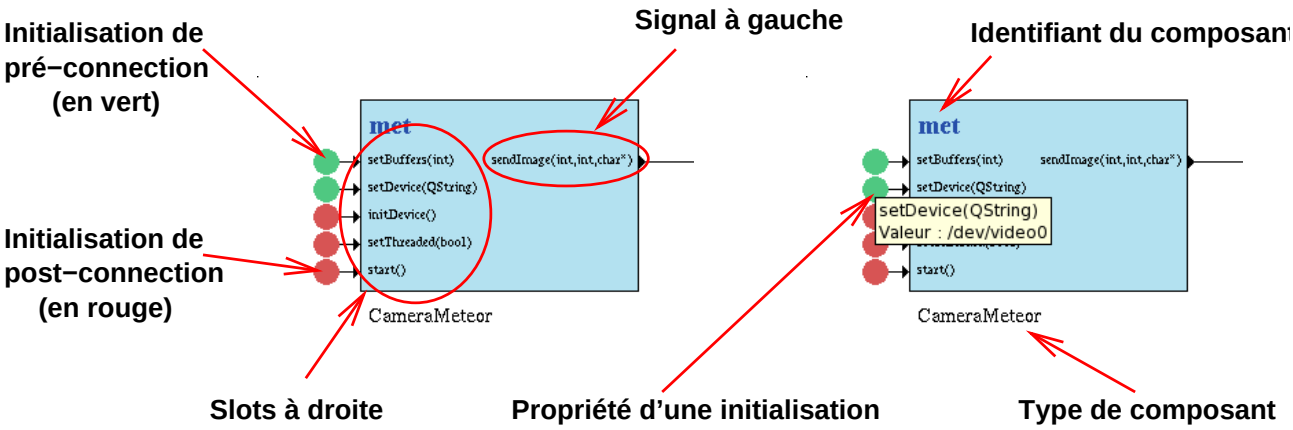
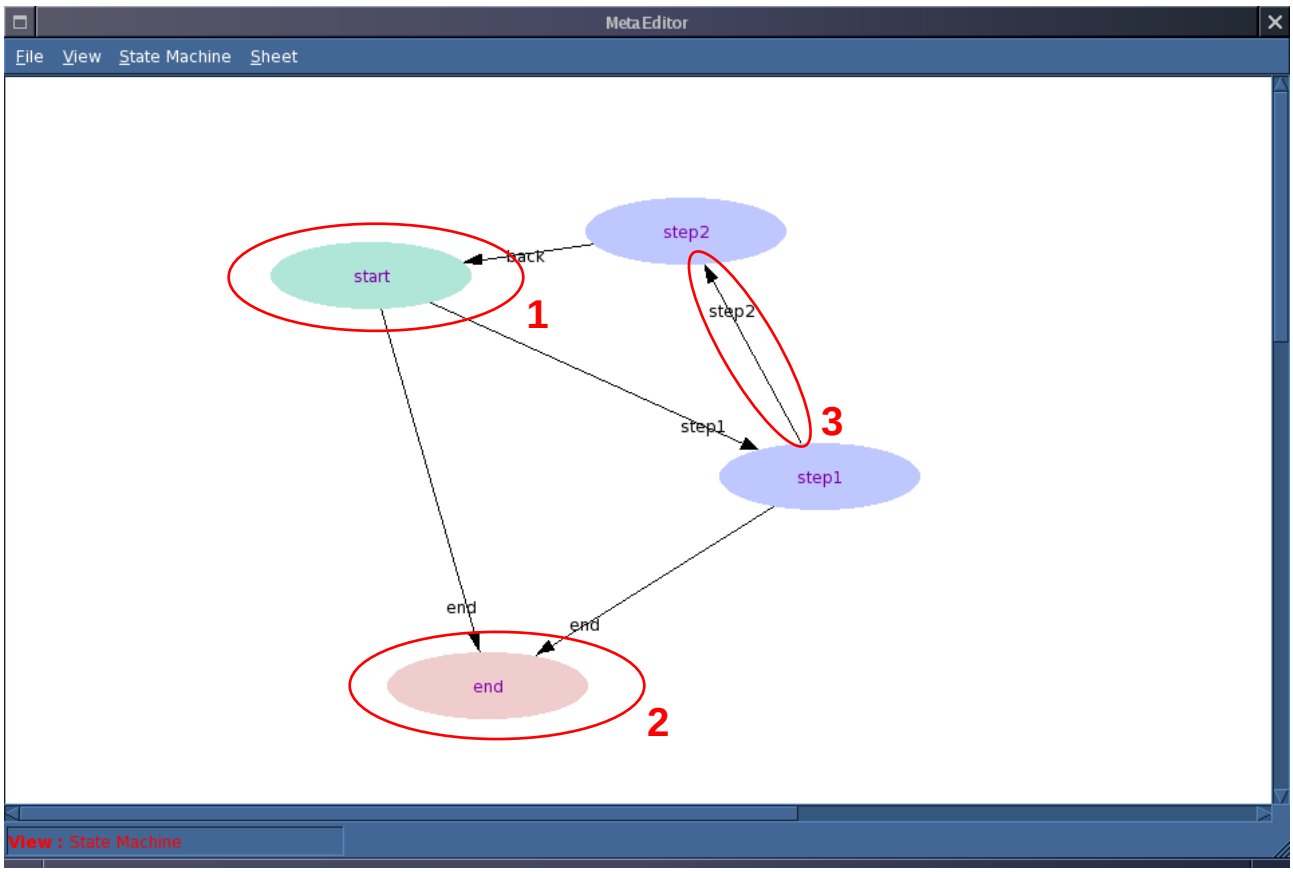



FIG. 2.14 – Détails d'un composant, avec ou sans affichage des propriétés d'une initialisation




**Légende**

1. État initial de l'automate (en vert),
2. État final de l'automate (en rose),
3. Transition avec son jeton

FIG. 2.15 – Interface graphique en mode d'édition de l'automate

système d'augmentation qui doit dépendre du contexte et des divers médias à présenter au cours de la procédure qui seront également développés sous forme de composants au chapitre 4.

Nous allons à présent aborder le problème de la localisation par la vision dans lequel nous verrons deux applications complètes à notre architecture.

# Chapitre 3

# Système de localisation par la vision

Comme nous l'avons vu, les systèmes de réalité augmentée mobile doivent employer divers capteurs pour résoudre le problème de la localisation du point de vue. Dans le cadre de la réalité augmentée en vision indirecte, ce point de vue sera celui du capteur d'acquisition des images, en l'occurrence une caméra.

Or, nous pouvons utiliser directement la caméra pour effectuer la localisation du point de vue. Pour cela, nous nous attacherons à examiner les outils théoriques et mathématiques à notre disposition pour modéliser le capteur et estimer sa localisation dans l'espace de travail (dans ce cas on parle d'estimation de la pose). De plus, nous examinerons les divers systèmes qui permettent la localisation du point de vue par la vision.

Enfin, nous verrons quelles sont les solutions apportées dans le cadre de notre démonstrateur qui apporte ses propres réponses au problème de la localisation par la vision tout en proposant quelques algorithmes originaux permettant le calcul de la pose et l'échantillonnage d'une zone rectangulaire.

## 3.1 Outils théoriques pour la localisation par vision en réalité augmentée

Les outils théoriques pour les systèmes de vision introduits s'attachent aux deux catégories de problèmes rencontrés lorsque l'on souhaite effectuer une localisation à l'aide d'une caméra :

- le problème de la calibration, qui permet de modéliser et de déterminer les paramètres du capteur, c'est à dire déterminer la relation existant entre l'image fournie par la caméra et l'espace qui est ainsi photographié,
- le problème de l'estimation de la pose de la caméra, qui permet de déterminer la position et l'orientation de la caméra par rapport à un repère attaché à l'espace de travail,

Nous allons à présent survoler ces deux volets pour lesquels de nombreuses réponses ont été apportées au cours des dernières décennies.

### 3.1.1 Calibration d'une caméra

#### 3.1.1.1 Modélisation du capteur

Avant de calibrer la caméra, il est nécessaire de la modéliser. Un modèle de caméra représente le processus de formation des images. Il permet d'établir la relation analytique entre les coordonnées d'un point dans l'espace objet, et celle du point correspondant dans le plan image au moyen des paramètres choisis. Plus précisément, il s'agit de déterminer la matrice de projection qui transforme un point de l'espace 3D en un point du plan que forme l'image.

Nous verrons tout d'abord le modèle du sténopé employé de manière classique auquel nous rajouterons ultérieurement les formulations traduisant les déformations radiales et tangentielles de l'image.

**Modèle du sténopé**

Cette modélisation, composée uniquement de transformations linéaires appelé sténopé linéaire ou parfois DLT (pour Direct Linear Transformation) a été formulée en 1971 par Abdel-Aziz et Karara [Abdel-Aziz et Karara, 1971].

Pour exprimer cette relation, il est nécessaire de définir plusieurs repères.

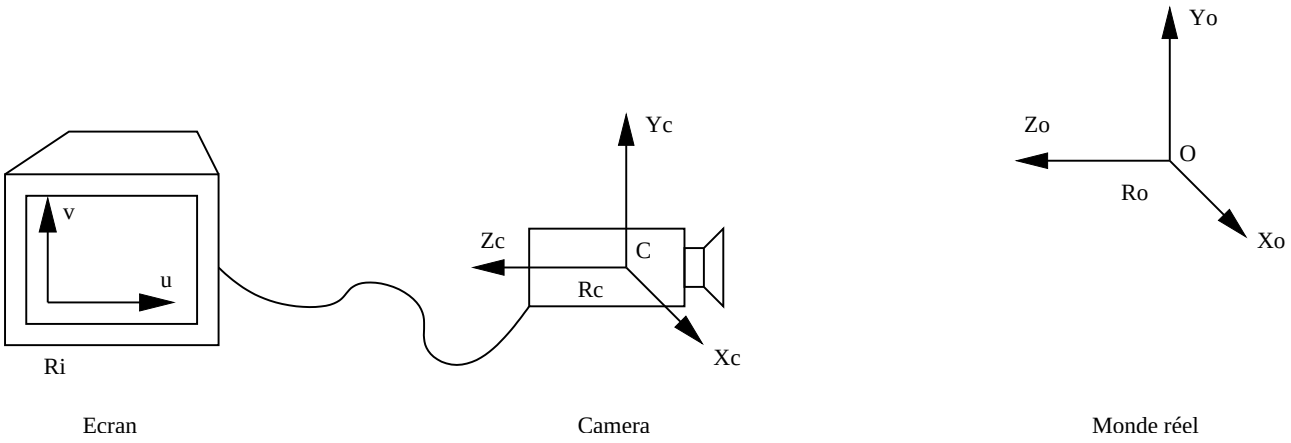



FIG. 3.1 – Les différents repères employés pour la calibration de caméra

Le modèle de projection qui permet de passer d'un point dans le repère monde $(R_o)$ au repère de visualisation $(R_i)$ peut s'écrire sous la forme :

$$\begin{pmatrix} su \\ sv \\ s \end{pmatrix} = \begin{pmatrix} k_u & s_{uv} & u_0 \\ 0 & k_v & v_0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} & & & t_x \\ & R_{C/O} & & t_y \\ & & & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{pmatrix} \tag{3.1}$$

Qui s'écrit souvent, plus simplement :

$$\underset{(p_i)}{\begin{pmatrix} su \\ sv \\ s \end{pmatrix}} = \underset{(M_{int})}{\begin{pmatrix} k_u f & s_{uv} f & u_0 \\ 0 & k_v f & v_0 \\ 0 & 0 & 1 \end{pmatrix}} \cdot \underset{(M_{ext})}{\begin{pmatrix} & & & t_x \\ & R_{C/0} & & t_y \\ & & & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}} \cdot \underset{(P_i)}{\begin{pmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{pmatrix}} \tag{3.2}$$

Avec :

- $p_i$ coordonnées de l'image $(R_i)$.
- $M_{int}$ matrice des paramètres intrinsèques de la caméra. Elle traduit la mise à l'échelle de l'image, la translation d'origine (dans le plan image) et la projection perspective ( $Z_c = f$ ). La plupart des modèles considèrent que le paramètre $s_{uv}$ est nul car ce dernier traduit la non-orthogonalité des cellules photo-réceptrices dans la matrice CCD ou CMOS de la caméra,
- $M_{ext}$ matrice de passage du repère monde $((R_0))$ au repère caméra $((R_c))$, ce que l'on appelle aussi les paramètres extrinsèques.
- $P_i$ coordonnées de l'objet dans le repère monde $((R_0))$.

Un certain nombre de variables sont ici introduites. Parmi celles-ci, il faut distinguer les paramètres intrinsèques des paramètres extrinsèques. Les paramètres extrinsèques sont liés à la transformation du repère monde au repère caméra. Ceux-ci sont :

- $R_{C/O}$ qui est la matrice de rotation entre le repère $(R_0)$ et le repère $(R_c)$,
- les coordonnées du centre $C$ dans le repère $(R_0)$ : $t_x, t_y$ et $t_z$. Nous utiliserons également $t = (t_x, t_y, t_z)$,

Les paramètres intrinsèques liés à la formation de l'image dans la caméra sont :

- la focale de l'objectif utilisé : $f$.
- les facteurs d'échelle suivant les deux axes de l'image : $k_u$ et $k_v$.
- la translation d'origine de l'image : $u_0$ et $v_0$.

L'équation 3.2 peut aussi s'écrire après produit et normalisation :

$$\begin{pmatrix} su \\ sv \\ s \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & 1 \end{pmatrix} \cdot \begin{pmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{pmatrix} \tag{3.3}$$

**Distortions optiques**

Les propriétés optiques des lentilles des caméras font que parfois l'image récupérée présente des distortions. Si l'on prend $x$ et $y$ les coordonnées réelles et normalisées d'un point dans l'image et $\hat{x}$ et $\hat{y}$ les coordonnées idéales de ce point, c'est à dire sans distortion, nous obtenons :

$$\hat{x} = x + x[k_1 r^2 + k_2 r^4] + ... \tag{3.4}$$
$$\hat{y} = y + y[k_1 r^2 + k_2 r^4] + ... \tag{3.5}$$

Avec $k_1$ et $k_2$ les coefficients de distortion radiale et $r^2 = x^2 + y^2$. Il est à noter que l'équation complète de distortion comporte d'autres termes qui prennent en compte la distortion tangentielle. Toutefois, le gros des imprécisions, quand on veut tenir compte des distortions dues à l'optique viennent de la distortion radiale [Tsai, 1987]. Si nous souhaitons exprimer ces distortions dans l'image réelle, alors nous devons introduire $\hat{u}$ et $\hat{v}$ qui seront les coordonnées idéales du point dans l'image et $u$ et $v$ les coordonnées réelles mesurées sur l'image. Comme nous avons les correspondances $u = u_0 + k_u x + s_{uv} y$ et $v = v_0 + k_v y$, nous pouvons écrire :

$$\hat{u} = u + (u - u_0)[k_1 r^2 + k_2 r^4] + ... \tag{3.6}$$
$$\hat{v} = v + (v - v_0)[k_1 r^2 + k_2 r^4] + ... \tag{3.7}$$

Suivant le type d'application visée, la prise en compte des distortions optiques peut être ignorée.

La modélisation mathématique du processus de formation des images étant réalisée, nous allons voir comment estimer les paramètres qui régissent ces équations. Cette étape s'appelle la calibration.

### 3.1.1.2 Méthodes de calibration

La calibration de la caméra consiste à estimer les paramètres intrinsèques et extrinsèques de la caméra, donc à déterminer les 11 paramètres $m_{11} \cdots m_{33}$. Pour cela, des méthodes d'optimisation sont utilisées. Elles consistent à estimer les paramètres ci-dessus à partir d'un échantillonnage constitué de points mesurés dans le repère monde $(X_i, Y_i, Z_i)$ et des points correspondants dans l'image $(u_i, v_i)$. La première méthode que nous allons présenter n'estime pas les distortions optiques, à l'inverse de la seconde.

**La méthode des moindres carrés**

C'est une méthode employée pour effectuer la minimisation d'erreurs pour la calibration. L'équation 3.3 devient :

$$\begin{pmatrix} X_i & Y_i & Z_i & 1 & 0 & 0 & 0 & 0 & -u.X_i & -u.Y_i & -u.Z_i \\ 0 & 0 & 0 & 0 & X_i & Y_i & Z_i & 1 & -v.X_i & -v.Y_i & -v.Z_i \end{pmatrix} * C = \begin{pmatrix} u \\ v \end{pmatrix} \quad (3.8)$$

Avec $C = (m_{11}, m_{12}, m_{13}, m_{14}, m_{21}, m_{22}, m_{23}, m_{24}, m_{31}, m_{32}, m_{33})^T$.

La connaissance d'un point du monde réel de coordonnées $(X_0, Y_0, Z_0)$ et de ses coordonnées dans l'image $(u, v)$ nous donne 2 équations pour 11 inconnues. Il faut donc utiliser 6 points différents au minimum pour que le système soit déterminable. Ces points doivent respecter les conditions suivantes :

– ils ne doivent pas être coplanaires ni alignés,
– ils ne doivent pas être confondus,

$N$ points ($N \geq 6$) sont utilisés pour résoudre le système : $[A] \cdot C = B$. Le vecteur C qui minimise la distance euclidienne entre $[A] \cdot C$ et $B$ est alors, d'après la méthode des moindres carrés : $C = A^T(AA^T)^{-1}B$. La matrice C étant déterminée, la calibration de la caméra est effectuée. En effet, C est la matrice de projection qui transforme un point de l'espace 3D en un point du plan que forme l'image.

De nombreuses autres méthodes différentes existent pour calibrer une caméra. En effet, la méthode présentée estime de manière globale l'ensemble des paramètres intrinsèques et extrinsèques. Certaines applications nécessitent que ces estimations soient faites pour chacun des paramètres, afin de connaître la valeur réelle des paramètres intrinsèques et extrinsèques. Nous pouvons citer par exemple la méthode de Faugeras-Toscani [Faugeras et Toscani, 1986] qui introduit des contraintes sur les coefficients de la matrice à utiliser en vue de simplifier la résolution du problème. Nous allons à présent détailler la méthode de calibration de Zhang que nous utiliserons ultérieurement dans notre système.

**La méthode de Zhang**

Cette méthode [Zhang, 1999] est basée sur l'homographie que l'on peut observer entre un objet plan dans l'espace et son image. Ainsi, en reprenant l'équation 3.2 page 74 et en supposant que le plan contenant les points employés pour la calibration est d'équation $Z = 0$ dans le repère monde, nous obtenons :

$$\begin{pmatrix} su \\ sv \\ s \end{pmatrix} = M_{int} \begin{bmatrix} r_1 & r_2 & r_3 & t \end{bmatrix} \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} = M_{int} \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \quad (3.9)$$

Où $r1$, $r2$ et $r3$ désignent les colonnes de la matrice de rotation $R_{0/C}$. Un point $P$ du plan et son image $p$ sont donc liés par l'homographie $H$ suivant la relation :

$$p = HP \text{ avec } H = \lambda M_{int} \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \quad (3.10)$$

$\lambda$ vaut ici $\frac{1}{s}$ et est un facteur d'échelle arbitraire. Le modèle de caméra employé par Zhang prend également en compte le paramètre $s_{uv}$ qui peut être non nul. Si un ensemble de points $P_i$ et leurs images $p_i$ sont donnés, cette homographie peut-être déterminée avec la méthode des moindres carrés que nous avons exposé plus haut ou à l'aide de méthodes d'optimisation non-linéaires.

Si nous reprenons l'équation 3.10, en notant $h_1$, $h_2$ et $h_3$ les colonnes de $H$ et en traduisant que $r1$ et $r2$ font partie d'un repère orthonormé, nous obtenons :

$$h_1^T M_{int}^{-1^T} M_{int}^{-1} h_2 = 0 \quad (3.11)$$

$$h_1^T M_{int}^{-1^T} M_{int}^{-1} h_1 = h_2^T M_{int}^{-1^T} M_{int}^{-1} h_2 \quad (3.12)$$

Ces contraintes peuvent être réécrites en posant :

$$B = M_{int}^{-1^T} M_{int}^{-1} = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{12} & B_{22} & B_{23} \\ B_{13} & B_{23} & B_{33} \end{bmatrix} \tag{3.13}$$

$$= \begin{bmatrix} \frac{1}{k_u^2} & \frac{-s_{uv}}{k_u^2 k_v} & \frac{v_0 s_{uv} - u_0 k_v}{k_u^2 k_v} \\ \frac{-s_{uv}}{k_u^2 k_v} & \frac{s_{uv}^2}{k_u^2 k_v^2} + \frac{1}{k_v^2} & -\frac{s_{uv}(v_0 s_{uv} - u_0 k_v)}{k_u^2 k_v^2} - \frac{v_0}{k_v^2} \\ \frac{v_0 s_{uv} - u_0 k_v}{k_u^2 k_v} & -\frac{s_{uv}(v_0 s_{uv} - u_0 k_v)}{k_u^2 k_v^2} - \frac{v_0}{k_v^2} & \frac{(v_0 s_{uv} - u_0 k_v)^2}{k_u^2 k_v^2} + \frac{v_0^2}{k_v^2} + 1 \end{bmatrix} \tag{3.14}$$

Nous pouvons poser le vecteur :

$$b = \begin{bmatrix} B_{11} & B_{12} & B_{22} & B_{13} & B_{23} & B_{33} \end{bmatrix}^T \tag{3.15}$$

De même, la $i$ème colonne de la matrice $H$ s'écrira $h_i = \begin{bmatrix} h_{i1} & h_{i2} & h_{i3} \end{bmatrix}^T$. Nous pouvons alors écrire :

$$h_i^T B h_j = v_{ij}^T b \tag{3.16}$$

avec

$$v_{ij} = \begin{bmatrix} h_{i1}h_{j1} & h_{i1}h_{j2} + h_{i2}h_{j1} & h_{i2}h_{j2} & h_{i3}h_{j1} + h_{i1}h_{j3} & h_{i3}h_{j2} + h_{i2}h_{j3} & h_{i3}h_{j3} \end{bmatrix}^T \tag{3.17}$$

Les deux équations 3.11 et 3.12 page ci-contre peuvent donc s'écrire :

$$\begin{bmatrix} v_{12}^T \\ (v_{11} - v_{22})^T \end{bmatrix} b = 0 \tag{3.18}$$

Si $n$ plans sont observés, nous pouvons regrouper $n$ équations du même type que l'équation 3.18. Nous avons donc :

$$Vb = 0 \tag{3.19}$$

avec $V$ une matrice de taille $2n \times 6$. Si $n \geq 3$, alors nous pouvons déterminer une solution à cette équation qui sera le vecteur propre associé à la plus petite valeur propre de la matrice $V^T V$. Nous obtenons ainsi une estimation de $b$ qui nous permet de calculer les paramètres intrinsèques. Ces derniers seront de la forme :

$$\begin{aligned} v_0 &= (B_{12}B_{13} - B_{11}B_{23})/(B_{11}B_{22} - B_{12}^2) \\ \lambda &= B_{33} - [B_{13}^2 + v_0(B_{12}B_{13} - B_{11}B_{23})]/B_{11} \\ k_u &= \sqrt{\lambda/B_{11}} \\ k_v &= \sqrt{\lambda B_{11}/(B_{11}B_{22} - B_{12}^2)} \\ s_{uv} &= -B_{12}k_u^2 k_v/\lambda \\ u_0 &= s_{uv}/k_v - B_{13}k_u^2/\lambda \end{aligned}$$

Une fois la matrice $M_{int}$ connue, il est possible de calculer les paramètres extrinsèques de la caméra :

$$\begin{aligned} r_1 &= kM_{int}^{-1}h_1 \\ r_2 &= kM_{int}^{-1}h_2 \\ r_3 &= r_1 \wedge r_2 \\ t &= kM_{int}^{-1}h_3 \\ \text{avec } k &= \frac{1}{||M_{int}^{-1}h_1||} = \frac{1}{||M_{int}^{-1}h_2||} \end{aligned}$$

Dans quelques cas dégénérés cette méthode de résolution échoue : il s'agit de configurations où les plans observés d'une image sur l'autre sont parallèles. Dans la pratique, il suffit de changer l'orientation du plan d'une image sur l'autre pour se retrouver dans des conditions permettant la calibration de la caméra.

Il est également possible d'améliorer le résultat de cette calibration en employant des méthodes d'optimisation non-linéaires telles que celle de Levenberg-Marquart [More, 1977]. On initialise alors l'algorithme avec les résultats précédents puis on minimise itérativement le critère :

$$\sum_{i=1}^{n}\sum_{j=1}^{m} \|m_{ij} - \hat{m}(M_{int}, R_i, t_i, M_j)\|^2 \tag{3.20}$$

Où $\hat{m}(M_{int}, R_i, t_i, M_j)$ est la projection du point $M_j$ dans l'image $i$ selon l'équation 3.10 page 76.

Pour tenir compte des distortions optiques, il suffit d'introduire également les équations 3.6 et 3.7 page 75 dans la quantité que l'on cherche à minimiser. Ceci revient à écrire l'équation :

$$\sum_{i=1}^{n}\sum_{j=1}^{m} \|m_{ij} - \hat{m}(M_{int}, k_1, k_2, R_i, t_i, M_j)\|^2 \tag{3.21}$$

où $k_1$ et $k_2$ peuvent prendre pour valeur $0$ à l'initialisation de l'algorithme.

Nous utiliserons à la section 3.4.1 page 95 la méthode de calibration de Zhang car c'est une méthode dont l'utilisation est flexible qui est également exacte dans ces calculs.

Une fois la caméra calibrée, nous supposons que les paramètres intrinsèques de cette dernière resteront fixes au cours du temps alors que les paramètres extrinsèques de cette dernière (la position et l'orientation) sont par définition variables si l'on utilise un dispositif mobile. Il existe des méthodes permettant d'estimer ces paramètres, connaissant les paramètres intrinsèques. On parle alors de méthodes d'estimation de la pose.

## 3.1.2 Estimation de la pose de la caméra

### 3.1.2.1 Introduction aux méthodes de calcul de la pose

Le calcul de la pose s'appuie sur l'extraction de primitives géométrique qui vont permettre d'appairer des points 2D extraits de l'image (et donc exprimé dans le repère image) avec les points 3D connus de l'objet (exprimé dans le repère associé à l'espace de travail). Par la suite, lorsque nous évoquerons les conditions d'application d'un algorithme de pose, nous parlerons de points en lieu et place de couples de points 2D-3D.

Il existe plusieurs types d'algorithmes dédiés à l'estimation de la pose d'une caméra. On peut distinguer trois grandes familles :

- les méthodes analytiques reposant sur un faible nombre de points,
- les méthodes numériques d'optimisation par minimisation d'un critère d'erreur,
- les méthodes itératives, sous ensemble important de la famille précédente.

Nous ne pourrons évoquer de manière exhaustive l'ensemble des méthodes proposées dans la littérature. Toutefois, nous présenterons celle qui nous ont paru le plus représentatives et qui peuvent se calculer en temps réel, ce facteur étant déterminant pour les systèmes de réalité augmentée.

### 3.1.2.2 Méthodes analytiques

Ce sont des méthodes qui sont utilisées sur un faible nombre de points et qui admettent un ensemble fini de solutions. Leur complexité algorithmique est généralement faible d'où un temps de calcul court lorsque l'algorithme est implémenté. L'estimation de la pose faite par ces méthodes est

*théoriquement précis*. Toutefois, ce calcul est tributaire de l'extraction des points de l'image que l'on veut appairer avec les points de l'objet en trois dimensions. Suivant la qualité de l'image acquise et des traitements effectués, ces méthodes peuvent en définitive produire des résultats imprécis.

De nombreuses méthodes de calcul analytique ont été proposées ces vingt dernières années. Ainsi, en 1981, Fischler et Bolles ont proposé plusieurs méthodes de résolution et de calcul de la pose [Fischler et Bolles, 1981]. Dans l'annexe de leur article sur la méthode RANSAC (RAndom SAmple Consensus), ils donnent deux solutions analytiques différentes dont une méthode pour trois points comportant la résolution des racines de deux équations quadratiques et une méthode pour quatre points non coplanaires qui comporte une dizaine d'étapes de calcul.

Hung et al ont de leur côté proposé en 1985 une méthode permettant d'estimer la pose d'une cible à l'aide de 4 points coplanaires non alignés [Hung et al., 1985].

Quelques années plus tard, en 1989, Dhome a introduit une méthode permettant de calculer la pose d'un objet en utilisant trois arêtes de ce dernier. Un polynôme de degré 8 est alors résolu et les solutions proposées triées en fonction de la validité de la configuration des arêtes [Dhome et al., 1989].

Enfin, en 1992, Dementhon et Davis ont proposé une méthode permettant de calculer la pose d'une caméra à l'aide de trois points seulement [DeMenthon et Davis, 1992].

#### 3.1.2.3 Méthodes d'optimisation par minimisation d'un critère d'erreur

Comme nous l'avons vu, les méthodes analytiques sont tributaires de la qualité de l'image acquise. La solution pour tenter de minimiser l'erreur est donc d'effectuer le calcul sur un ensemble plus large de points. Toutefois, les méthodes analytiques ne se basant que sur un nombre réduit et fini de points, il faut s'appuyer sur des méthodes qui estiment la pose en minimisant un critère d'erreur. Parmi celles-ci, nous retrouverons une formulation dite des "moindres carrés" dans laquelle les paramètres intrinsèques ne font plus partie des inconnues à estimer à la différence de la méthode de calibration présentée en 3.1.1.2 page 75 [Loukil, 1993].

#### 3.1.2.4 Méthodes itératives de calcul de la pose

Ces méthodes constituent un sous-ensemble important des méthodes précédentes. La pose est estimée une première fois (parfois par un algorithme analytique) puis un algorithme itératif affine au fur et à mesure l'estimation de la pose en se basant sur la minimisation d'un critère donné. Ces méthodes permettent d'obtenir une excellente précision dans la localisation de la caméra par rapport à son environnement. Toutefois, chaque itération effectuée entraîne un sur-coût en temps de calcul pour estimer la pose. Il faut en tenir compte afin d'optimiser le nombre d'itérations à effectuer et les opérations réalisées au cours de chaque itération afin de réduire ce sur-coût.

Fishler et Bolles ont ainsi introduit un algorithme basé sur leur résolution analytique du problème des trois points pour l'étendre à quatre points non coplanaires [Fischler et Bolles, 1981].

En 1995, Dementhon et Davis ont également proposé une méthode itérative (nommée POSIT) pour calculer la pose d'un objet. L'ensemble des points utilisés ne doit pas être coplanaire. De plus, il doit au minimum comporter quatre points [Dementhon et Davis, 1995].

Enfin, nous mentionnerons également l'algorithme de l'itération othogonale développé en 2000 qui fera l'objet d'une étude détaillée.

#### 3.1.2.5 L'itération orthogonale

L'algorithme d'itération orthogonale introduite par Lu et al. permet de calculer la pose d'une caméra dont on connaît les paramètres intrinsèques [Lu et al., 2000]. Il utilise une formulation qui optimise les critères de convergence concernant la matrice de rotation, à l'inverse des algorithmes classiques à vocation générale.

**Notations**
Considérons un ensemble de points de coordonnées $p_i$ dans le repère monde, $q_i$ dans le repère caméra et ayant $v_i$ de coordonnées $(u_i, v_i, 1)$ pour une image dans le plan image normalisé lié au repère caméra. En notant

$$R = \begin{pmatrix} r_1^t & r_2^t & r_3^t \end{pmatrix} \text{ et } t = \begin{pmatrix} t_x & t_y & t_z \end{pmatrix}$$

les paramètres extrinsèques de la caméra, on a alors les relations suivantes :

$$u_i = \frac{r_1 p_i + t_x}{r_3 p_i + t_z}$$
$$v_i = \frac{r_2 p_i + t_y}{r_3 p_i + t_z}$$

et

$$q_i = R p_i + t$$

Si on parvient à déterminer les $q_i$ à partir des points images, on obtiendra $R$ et $t$ en minimisant le critère d'erreur quadratique suivant :

$$\min_{R,t} \sum_i \|R p_i + t - q_i\|^2, \text{ avec } R^t R = I \tag{3.22}$$

**Reformulation du problème**
Un tel problème de minimisation sous contrainte peut être résolu en utilisant les quaternions ou la Décomposition en Valeurs Singulières (SVD). La méthode SVD procède ainsi : On considère $\bar{p} = \frac{1}{n}\sum_i p_i$ et $\bar{q} = \frac{1}{n}\sum_i q_i$ les barycentres respectifs de $p_i$ et $q_i$ puis on définit

$$p'_i = p_i - \bar{p}, q'_i = q_i - \bar{q}$$

et

$$M = \sum_i q'_i p'^t_i$$

$\frac{1}{n}M$ représente la matrice de covariance entre $p_i$ et $q_i$. On montre alors que la solution du problème (3.22) est donnée par

$$R^* = arg\, max_R(R^t M) \tag{3.23}$$
$$t^* = \bar{q} - R^* \bar{p} \tag{3.24}$$

Soit $(U, \sigma, V)$ une SVD de M, c'est à dire $U^t M V = \sigma$ La solution de (3.23) est alors

$$R^* = V U^t$$

L'algorithme porte le nom d'itération orthogonal car il s'appuie sur l'identité entre la projection orthogonale de $q_i = R p_i + t$ sur le rayon optique passant par $v_i$ et C le centre optique de la caméra d'une part et $q_i$ d'autre part,puisque $v_i$ est l'image de $q_i$ (voir figure 3.2 page suivante). La matrice de projection sur ce rayon optique étant

$$V_i = \frac{v_i v_i^t}{v_i^t v_i}$$

Nous obtenons

$$R p_i + t = V_i(R p_i + t)$$

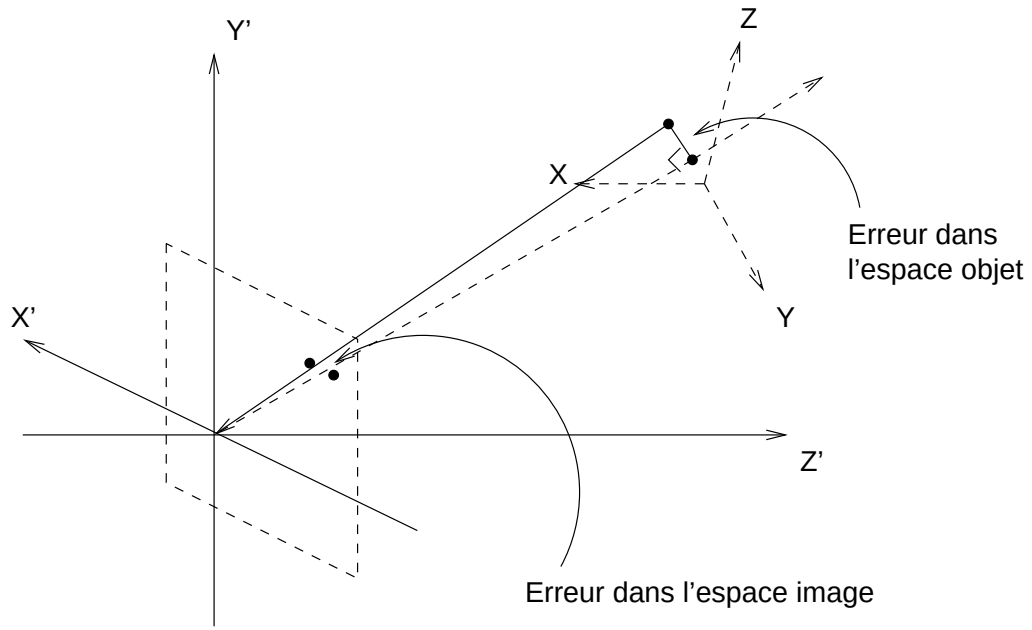



FIG. 3.2 – Projection sur le rayon optique passant par $v_i$ et C

l'erreur sur la matrice de projection se traduit donc par une erreur $e_i = (I - V_i)(Rp_i + t)$ dans l'espace objet. Nous allons estimer $R$ et $t$ qui minimisent la somme des carrés de ces erreurs :

$$E = \sum_i \|ei\|^2 = \sum_i \|(I - V_i)(Rp_i + t)\|^2 \tag{3.25}$$

Les informations données par les $v_i$ sont entièrement contenues dans les $V_i$. Pour $R$ donné, nous pourrons écrire la relation :

$$t(R) = \frac{1}{n}(I - \frac{1}{n}\sum_i V_i)^{-1} \sum_i (V_i - I)Rp_i \tag{3.26}$$

On peut montrer que $(I - \frac{1}{n}\sum_i V_i)$ est définie positive, et donc (3.26) a bien un sens. On définit alors

$$q_i(R) = V_i(Rp_i + t(R)) \text{ et } \bar{q}(R) = \frac{1}{n}\sum_i q_i(R)$$

L'équation (3.25) peut alors être réécrite sous la forme

$$E(R) = \sum_i \|Rp_i + t(R) - q_i(R)\|^2$$

**Résolution itérative**

On remarque alors que cette équation ressemble fortement à (3.22) mais on ne peut pas résoudre directement $R$ en utilisant

$$M(R) = \sum_i q'_i(R)p'^t_i \text{ où } p'_i = p_i - \bar{p} \text{ et } q'_i(R) = q_i(R) - \bar{q}(R)$$

car $M$ dépend ici de R. En revanche on peut alors calculer $R$ itérativement de la façon suivante : Si on connaît $R^{(k)}$, on peut alors calculer $t^{(k)} = t(R^{(k)})$ et $q_i^{(k)} = R^{(k)}p_i + t^{(k)}$ et

$$R^{(k+1)} = arg\,min_R \sum_i \|Rp_i + t - V_i q_i^{(k)}\| \tag{3.27}$$

$$R^{(k+1)} = arg\,max_R(R^t M(R^{(k)})) \tag{3.28}$$

Les $V_i q_i^{(k)}$ sont alors traité comme les $q_i$ dans (3.27). On obtient ainsi $R^{(k+1)}$ grâce à la SVD de $M(R^{(k)})$, puis on posera $t^{(k+1)} = t(R^{(k+1)})$
Une solution $R^*$ au problème de l'estimation de la pose en utilisant l'algorithme d'itération orthogonal est donc un point fixe de la suite définie par (3.28) et vérifie

$$R^* = arg\,min_R \sum_i \|Rp_i + t - V_i(R^* p_i + t(R^*))\|$$

**Initialisation de l'algorithme**
Il est important de noter que s'il a été prouvé que l'algorithme d'itération orthogonal convergeait, rien n'assure que la pose vers laquelle il converge soit celle correspondante à la pose réelle. C'est pourquoi il est très important d'initialiser l'algorithme avec une matrice $R^{(0)}$ proche de la bonne solution.

L'initialisation est généralement faite en appliquant un modèle de perspective faible (en anglais *weak perspective*). Ce dernier approxime un objet comme étant plan et parallèle au plan image. La rotation pour initialiser l'algorithme est alors calculée.

Comme nous le verrons ultérieurement, le choix d'une bonne matrice de rotation à l'initialisation conditionne les performances de l'itération orthogonale.

#### 3.1.2.6 Comparatif des algorithmes de calcul de pose

Comme nous pouvons le constater, les algorithmes de calcul de la pose sont nombreux et leurs conditions d'application varient. Afin de comparer rapidement leurs caractéristiques, nous avons dressé le tableau 3.1. La grande famille à laquelle appartient chacune des méthodes est indiquée ainsi que les conditions nécessaires pour pouvoir les employer.

| Nom | Année | Type | Conditions d'application |
|---|---|---|---|
| Fischler&Bolles a | 1981 | Analytique | 3 points |
| Fischler&Bolles b | 1981 | Itérative | 4 points |
| Fischler&Bolles c | 1981 | Analytique | 4 points non coplanaires |
| Hung&al | 1985 | Analytique | 4 points coplanaires |
| Dhome&al | 1989 | Analytique | 3 arêtes |
| Dementhon&Davis | 1992 | Analytique | 3 points |
| Moindres carrés | | Minimisation de critères d'erreur | 6 points au minimum |
| POSIT | 1995 | Itérative | min 4 points non coplanaires |
| Itération orthogonale | 2000 | Itérative | min 3 points |

TAB. 3.1 – Tableau récapitulatif des méthodes d'estimation de pose présentées

Dans cette section, nous venons d'introduire les différents algorithmes nécessaires pour élaborer un système de localisation par la vision. Ceux-ci vont de la modélisation du capteur en passant par l'estimation des paramètres du modèle jusqu'à l'exploitation de ces derniers pour retrouver la position et l'orientation du capteur, donc du système par rapport à un repère lié à un espace de travail donné.

Nous allons, dans la section suivante, présenter quelques systèmes de localisation par la vision mettant en oeuvre ces outils théoriques.

## 3.2 Systèmes de localisation par la vision en réalité augmentée

La localisation de la position et de l'orientation de la caméra en temps réel peut s'effectuer au moyen de plusieurs méthodes. Toutes celles que nous allons citer s'appuient sur la connaissance des paramètres intrinsèques de la caméra, puisque nous supposons que seuls les paramètres extrinsèques qui constituent la pose de la caméra sont susceptibles de varier au cours du temps dans le cadre d'une utilisation normale de la caméra.

Toutefois, cette localisation nécessite de pouvoir appairer des points de l'image avec des points de l'espace représenté dans l'image. Il nous faut donc extraire des primitives de cette image. La nature de ces primitives peut être très différente suivant les traitements que l'on veut effectuer ou le but recherché. Nous distinguons trois grandes familles de méthodes d'extraction des primitives de localisation :

- Celles qui sont basées sur la recherche d'une cible connue parfois aussi appelée amer qui a été mise en place dans la scène,
- Celles qui sont basées sur l'extraction de primitives géométriques particulières intrinsèques aux objets dont on veut assurer le suivi,
- Celles qui sont basées sur l'extraction de primitives géométriques de la scène qui n'ont aucun lien avec la scène à suivre.

Nous ne traiterons pas de la dernière catégorie car elle ne permet pas de fournir d'informations sémantiques au système de vision concernant ce qui est vu. En effet, ces méthodes vont suivre des primitives caractéristiques dans l'image sans établir de mise en correspondance entre ce qui est vu et la base de connaissances du système. Aussi, nous allons nous étendre sur les deux premières familles de systèmes de vision pour la réalité augmentée.

### 3.2.1 Les méthodes par recherche et extraction de cibles codées

Ces méthodes permettent de résoudre dans le même temps le lien sémantique et le problème de la pose de la caméra par rapport à l'objet observé. Le principe de ces méthodes est le suivant : le monde réel est instrumenté en employant des amers (ou cibles) au graphisme particulier dont la géométrie est connue. Au lieu de chercher à reconnaître directement les objets du monde réel, il suffit de reconnaître les cibles qui sont accolées à ces derniers. Leur géométrie étant connue, il est alors possible d'optimiser les traitements d'image à employer pour les reconnaître. De plus, ces cibles peuvent renfermer un motif particulier ou un code qui permet de faire le lien sémantique entre ce qui est observé et la connaissance à priori que le système a du monde qu'il observe.

De nombreux systèmes de cible codée existent, aussi, nous n'allons présenter que quelques uns parmi ceux-ci, c'est à dire ceux qui sont le plus couramment utilisés.

#### 3.2.1.1 Cybercode

Ce système de localisation à l'aide d'une cible unique est l'un des plus anciens. Proposé par Rekimoto et al. en 1998, il fut dans un premier temps appelé Matrix [Rekimoto, 1998] puis il fut renommé ultérieurement en Cybercode [Rekimoto et Ayatsuka, 2000].

Les traitements employés pour détecter ce type de cible et estimer la pose de la caméra (voir figure 3.3 page suivante) sont les suivants :

**a)** L'image est binarisée,

**b)** Une analyse des composantes connexes pour déterminer les zones noires de l'image est réalisée pour trouver la barre située sous le code,

**c)** A partir de la localisation de la barre, les quatre coins du code sont retrouvés. En utilisant ces quatre coins, le système estime les paramètres dus aux distortions causées par la caméra ou l'inclinaison de la cible,

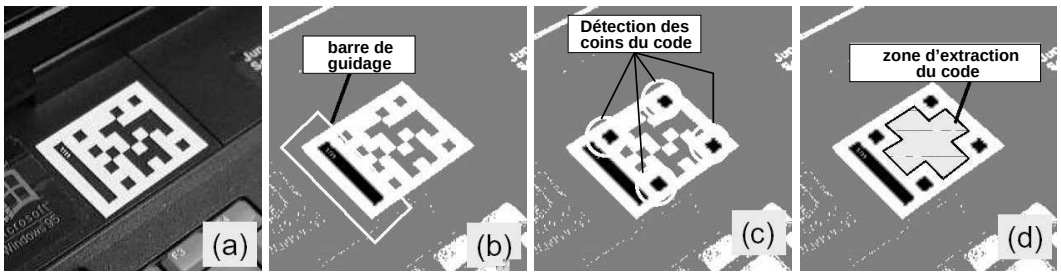



FIG. 3.3 – Les différentes étapes de reconnaissance d'un cybercode

**d)** En utilisant ces paramètres, l'image est renormalisée pour pouvoir extraire le code constitué de 33 bits. Si celui-ci est valide alors on pourra affiner l'estimation de la pose.

Il est à noter que l'étape 3 estime à chaque fois les 8 paramètres de l'homographie qui amène le plan de la cible codée dans le plan image en résolvant un système de 8 équations à 8 inconnues.

La pose est ensuite déterminée en s'appuyant sur les contraintes liant les quatre coins dans l'image et leurs coordonnées dans le monde réel. Ceci est effectué en minimisant un critère portant sur la normale au plan grâce à l'algorithme du simplexe. Les vecteurs représentant les arêtes de la cible sont alors recalculés et les paramètres de la pose de la caméra sont estimés à partir de ces derniers. Il y a 33 bits d'information dans le code porté par la cible, ce qui fait environ 8 milliard de possibilités.

#### 3.2.1.2 ARToolkit

Le système ARToolkit [Kato et al., 2000] [Kato et Billinghurst, 1999] est une librairie qui permet de rapidement créer des applications de réalité augmentée. Cette dernière contient également un système de cibles carrées comportant une bordure noire et un motif interne comme on peut le constater sur la figure 3.4.

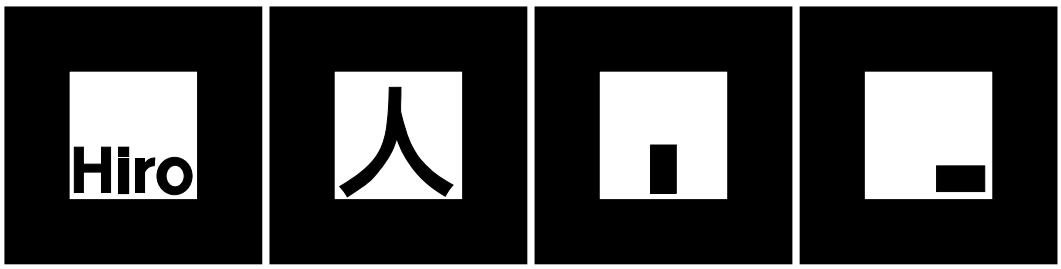



FIG. 3.4 – Différentes cibles employées par le système ARToolkit

Le processus pour extraire et identifier le code est ici similaire à celui du système précédent, à savoir :

1. L'image est binarisée,
2. Une analyse des composantes connexes de l'image est effectuée, les quadrilatères sont retenus,
3. L'homographie entre le plan de la cible et le plan image est calculée en vue de normaliser l'image à l'intérieur de la cible,
4. Le code est reconnu en calculant le coefficient de corrélation entre l'image normalisée et une banque d'images normalisées de référence,
5. Ensuite, les auteurs proposent leur propre algorithme itératif pour calculer la pose de la caméra par rapport à la cible.

Ce système présente des libertés en termes de motifs susceptibles d'être employés dans le système de cibles. En effet, il est possible de faire apprendre au système les motifs employés. Toutefois, plus

il y a de motifs stockés, plus le temps moyen de reconnaissance d'une cible s'allonge, la comparaison par corrélation nécessitant de parcourir l'ensemble des motifs afin de trouver le candidat le plus ressemblant.

Dans le souci d'augmenter le taux de reconnaissance des cibles, certaines variations sur le motif des cibles furent introduites, basées sur la formulation du calcul de corrélation entre le motif extrait et les motifs de référence appris par le système [Owen et al., 2002]. Toutefois, les performances de ce système, qui a le mérite d'être distribué gratuitement [Artoolkit, url], peuvent être grandement améliorées, c'est pourquoi, la plupart des laboratoires de recherche ont développé depuis leurs propres systèmes ressemblant fortement à ce dernier. Ainsi, Zhang et Al. compareront pas moins de 4 systèmes équivalents dont l'ARToolkit [Zhang et al., 2002]. D'autres systèmes similaires existent depuis qui cherchent à améliorer le taux de détection tel que ARTag [Fiala, 2005].

#### 3.2.1.3 InterSense

Durant l'année 2002, la société InterSense [Naimark et Foxlin, 2002] a également développé son propre système de cibles codées. Celui-ci s'appuie sur des cibles circulaires comme nous pouvons le voir à la figure 3.5. Bien qu'il ne s'agisse pas du premier du genre (Cho et Al. [Cho et Neumann, 1998] ont développé un système similaire en 1998), il est de loin le plus aboutit puisque les traitements sont actuellement embarqués dans l'électronique d'un système hybride comprenant une caméra et une centrale inertielle. Les codes sont stockés sur 15 bits ce qui fait en tout 32768 possibilités.

FIG. 3.5 – Quelques cibles codées du système d'InterSense

L'algorithme de reconnaissance des cibles, directement câblé à la suite du capteur CCD, comporte les étapes de traitement suivantes :

1. Un réhaussement du contraste,
2. Une détection des arêtes en employant un filtre de Sobel de noyau de taille $3 \times 3$,
3. Une binarisation suivie d'une érosion de l'image, les zones blanches de l'image contiennent des cibles potentielles qui sont filtrées en fonction de leur forme,
4. Le code de la cible est lu en s'aidant des points blancs au centre de la cible et des deux points noirs situés dans les quadrants. Cet ensemble de trois points forme un repère qui permet d'extraire le code.

Comme les autres systèmes de cible circulaires, il est nécessaire d'avoir plusieurs amers dans le champ de vision pour calculer la pose de la caméra, une cible donnant un point particulier. Dans le système présenté, un minimum de quatre cibles de ce type est nécessaire.

Les cibles présentent l'inconvénient de devoir être fixées sur les objets à suivre. De plus, les systèmes présentés utilisent des cibles planes, ce qui suppose que l'objet possède des surfaces planes sur lesquelles les cibles peuvent être collées. Enfin, si les cibles sont occultées, le système ne peut plus estimer la pose de la caméra. Ceci implique que les cibles doivent être placées de manière à ce qu'au moins une de celles-ci soit toujours visible par la caméra. Maintenant que nous avons exposé les

systèmes de réalité augmentée fonctionnant à l'aide de cibles codées, nous allons présenter quelques algorithmes qui extraient des primitives géométriques d'une image et permettent de s'affranchir potentiellement des inconvénients liés aux cibles codées.

### 3.2.2 Les méthodes d'extraction de primitives géométriques

Les primitives géométriques extraites peuvent être de plusieurs natures : il peut s'agir de points, de lignes, de cercles, etc. Ces méthodes présentent l'avantage de pouvoir se passer de cible. En revanche, il leur est nécessaire de s'appuyer sur un modèle précis des objets de la scène qui sont suivis. De plus, ces méthodes, pour être employées en temps réel, c'est à dire à la cadence d'acquisition des images vidéo, doivent inclure des méthodes permettant d'accélérer les traitements. Elles vont donc, d'une manière générale, incorporer un modèle d'évolution de la scène d'une image à une autre ou à tout le moins accélérer les traitements nécessités par les algorithmes itératifs de calcul de la pose.

Parmi ces méthodes, nous pouvons compter celle de Gennery [Gennery, 1982] [Gennery, 1992] qui permet de suivre des arêtes mises en évidence grâce à un filtre de Sobel. Pour accélérer la détection de ces primitives d'une image sur l'autre, un modèle extrapolant la vitesse d'une image sur l'autre est appliqué. La recherche de la nouvelle arête se fait à plus ou moins 5 pixels de l'endroit prédit. Dans le même ordre d'idée, Lowe [Lowe, 1987] [Lowe, 1992] propose un système fonctionnant sur un autre formalisme de l'extrapolation de la vitesse de déplacement des arêtes dans l'image. Harris [Harris et Stennet, 1990] utilise la reprojection des contours du modèle sur l'image afin de localiser la zone où les contours de l'objet réel doivent être recherchés. Ces algorithmes génériques, vieux de plus de dix ans sont encore beaucoup utilisés aujourd'hui en raison de leur simplicité de mise en oeuvre.

Quelques algorithmes plus spécifiques ont été développés depuis. Certains sont axés sur l'asservissement d'un modèle virtuel sur un modèle réel en détectant les arêtes saillantes (ici des segments de droite) d'un modèle [Drummond et Cipolla, 2002], d'autres le font avec des formes plus variées telles que des points, des lignes et des ellipses [Comport et al., 2003] [Marchand et al., 1999]. Enfin, certaines méthodes existent pour suivre des objets texturés desquels on a extrait des zones d'intérêt (patch) que l'on retrouve par homographie d'une image sur l'autre [Vacchetti et al., 2004]. La même chose peut être effectuée sur des plans texturés [Vigueras et al., 2003]. Les auteurs de cette dernière méthode ont en réalité plusieurs modèles de prédiction choisis en fonction du mouvement effectué par la caméra.

Toutefois, la plupart de ces techniques ne sont pas complètement automatisées. Une intervention humaine est souvent nécessaire pour donner les zones d'intérêt de l'image dans lesquelles les primitives se trouvent. De même, l'appariement entre ces primitives extraites et celles du modèle sont souvent faites manuellement. Enfin, l'un des problèmes récurrent de ces méthodes est de trouver à l'initialisation un bon algorithme de reconnaissance capable de distinguer l'objet observé dans la base de données de modèles répertoriés.

C'est pourquoi, la plupart des systèmes de réalité augmentée opérationnels et automatiques se basent sur un système de cibles codées pour pouvoir déterminer ce qui est suivi et où ce dernier est placé par rapport à la caméra ou par rapport à l'espace de travail. Ces systèmes, en termes de cadence de traitement des images vidéos ne sont pas nécessairement plus rapide mais il présentent l'avantage d'être entièrement automatisés, ce qui n'est pas encore le cas des autres méthodes. Dans la suite de nos travaux, nous allons nous intéresser à un tel système de cibles codées. Toutefois, dans l'immédiat, nous allons introduire quelques algorithmes spécifiques que nous avons développés et que nous avons incorporé dans notre système de localisation par la vision.

## 3.3 Introduction à de nouveaux algorithmes dédiés aux système de localisation par la vision basés sur des cibles carrées

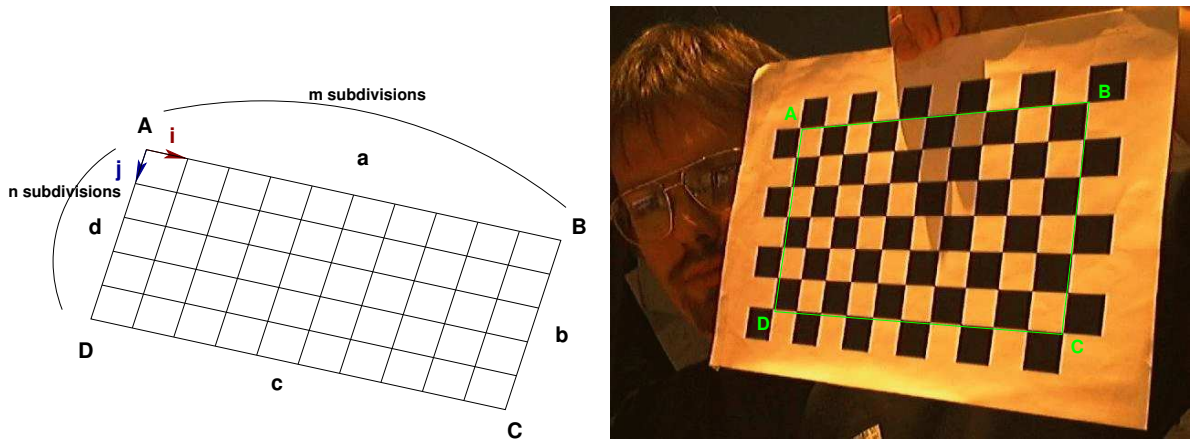
Dans cette section, nous allons introduire deux algorithmes spécifiques que nous avons développés pour notre système de localisation par la vision. Il s'agit d'une partie des contributions significatives apportées par ce système. Le premier algorithme permet l'échantillonnage d'une zone rectangulaire sans connaître les paramètres intrinsèques d'une caméra. Il sera appliqué dans le cadre d'une procédure de calibration automatique (voir section 3.4.1 page 95) des paramètres intrinsèques de la caméra et dans le décodage des cibles (voir section 3.4.2.3 page 99). Le deuxième algorithme calcule la pose d'une cible carrée de manière analytique en se basant sur les contraintes apportées par les renseignements concernant la géométrie de cette cible. Il sera employé ultérieurement pour retrouver la pose de la caméra par rapport à une cible codée.

### 3.3.1 Échantillonnage d'une zone rectangulaire

Le but de cette manipulation est d'échantillonner une portion rectangulaire de l'espace dans une image acquise par la caméra. Ce rectangle est dans une position et une orientation quelconque dans l'espace par rapport au capteur, ce qui engendre des déformations perspectives. Cet algorithme sera par la suite employé à plusieurs reprises dans notre système, comme nous le verrons en section 3.4.1 page 95.

#### 3.3.1.1 Notations

Soient $A, B, C$ et $D$ les sommets du rectangle. Nous appellerons $a, b, c$ et $d$ les longueurs des côtés $AB, BC, CD$ et $DA$ comme cela est montré sur la figure 3.6(a). Nous supposons que l'échantillonnage de la zone rectangulaire se fera en employant une subdivision de taille $m \times n$. Cette zone rectangulaire étant plane, nous choisissons $(A, \vec{i}, \vec{j})$ comme repère avec comme index : $i \in [[0..m]]$ et $j \in [[0..n]]$.




(a) Conventions de notation (b) Image sur laquelle les coins sont détectés

FIG. 3.6 – Échantillonnage d'une zone rectangulaire

#### 3.3.1.2 Formulation du problème

Sachant que l'objet de travail est un rectangle et qu'une image de ce dernier a été prise sur laquelle les coins ont été détectés, comment échantillonner de manière efficace la zone rectangulaire ?

La solution classique est de calculer l'homographie entre le plan constitué par la zone rectangulaire et son image (voir équation 3.9 page 76 présentée à propos de la méthode de calibration de Zhang). Ceci suppose la résolution d'un système comportant au minimum huit inconnues, résolution qui sera linéaire ou non suivant la formulation exacte des équations. En définitive, cette méthode comporte aussi un calcul masqué des paramètres intrinsèques de la caméra. Les points 2D échantillonnant la zone rectangulaire sont alors recalculés à l'aide de cette homographie.

La résolution à l'aide du calcul de l'homographie reste une méthode générique qui n'exploite pas l'information concernant le caractère rectangulaire de la zone à échantillonner. Nous allons voir que nous pouvons calculer les points 2D qui nous intéressent simplement en connaissant les coordonnées dans l'image des coins de la zone rectangulaire dans l'image.

#### 3.3.1.3 Première méthode : l'échantillonnage pondéré simple

Cette première méthode se base sur une formule barycentrique simple. Elle ne nécessite que la connaissance des coordonnées des points $A, B, C,$ et $D$ dans l'image qui sont $(u_A, v_A)$, $(u_B, v_B)$, $(u_C, v_C)$ et $(u_D, v_D)$. Nous notons $M_{ij}$ le point échantillonné à l'aide des index $i$ et $j$ qui aura pour coordonnées $(u_{ij}, v_{ij})$. A l'aide de la relation barycentrique simple, nous avons :

$$\forall i \in [[0..m]], \forall j \in [[0..n]]$$

$$u_{ij} = \frac{1}{mn}\left[u_A(m-i)(n-j) + u_B i(n-j) + u_C ij + u_D(m-i)j\right] \quad (3.29)$$

$$v_{ij} = \frac{1}{mn}\left[v_A(m-i)(n-j) + v_B i(n-j) + v_C ij + v_D(m-i)j\right] \quad (3.30)$$

L'application de cet algorithme sur une zone rectangulaire nous donne le résultat présenté à la figure 3.7(a) page suivante. Comme on peut le remarquer sur le damier, l'échantillonnage réalisé est très pauvre puisqu'il ne coïncide pas avec les coins du damier, ce qui est notamment visible de manière flagrante pour la deuxième et la pénultième colonne. Ceci s'explique à cause des problèmes de projection perspective qui résultent de la prise de l'image qui conserve les alignement mais non pas les distances. Une modification de cette méthode d'échantillonnage s'impose donc.

#### 3.3.1.4 Deuxième méthode : l'échantillonnage "perspectif"

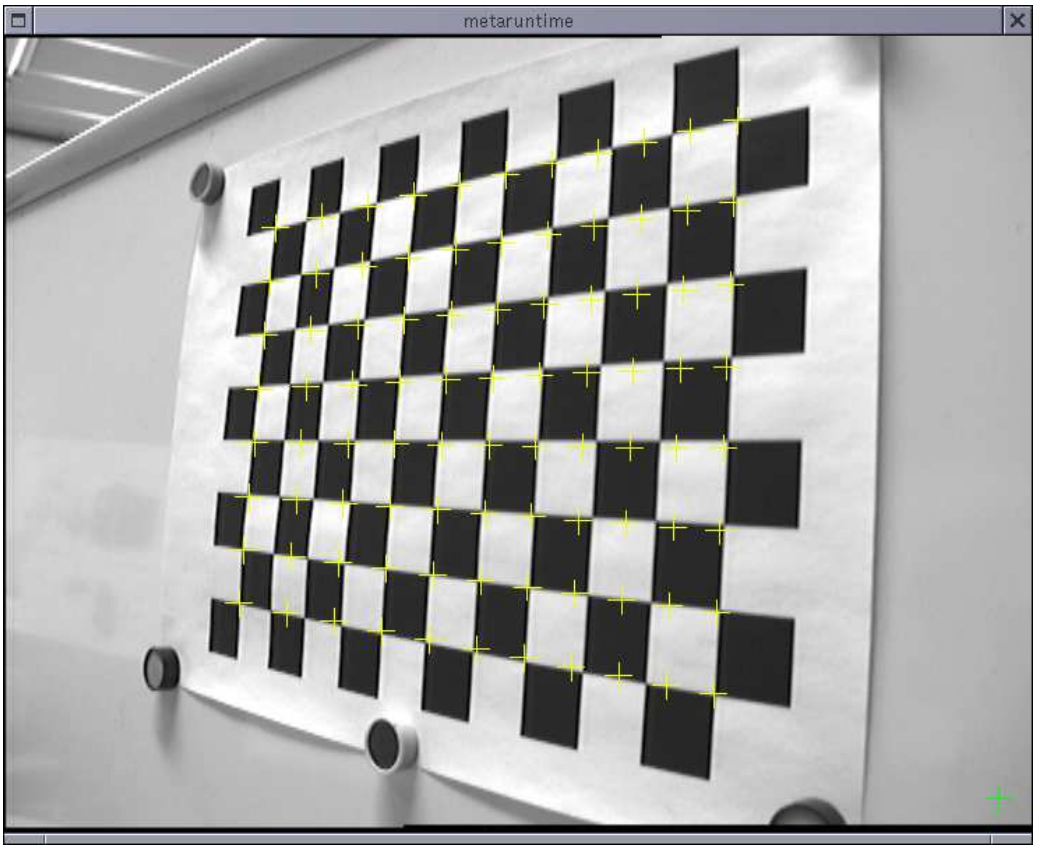
Si le problème de la projection perspective entre en ligne de compte, il nous faut une estimation de la profondeur des 4 sommets du rectangle par rapport à la caméra. Comme nous ne connaissons pas les paramètres intrinsèques à cette dernière, il nous faut nous appuyer sur une autre caractéristique connue. En l'occurrence, notre objet est rectangulaire. A l'aide de la caractérisation du processus de projection de l'image, nous allons voir comment nous pouvons obtenir une méthode d'échantillonnage beaucoup plus raffinée que la précédente.

**Modèle du sténopé linéaire**

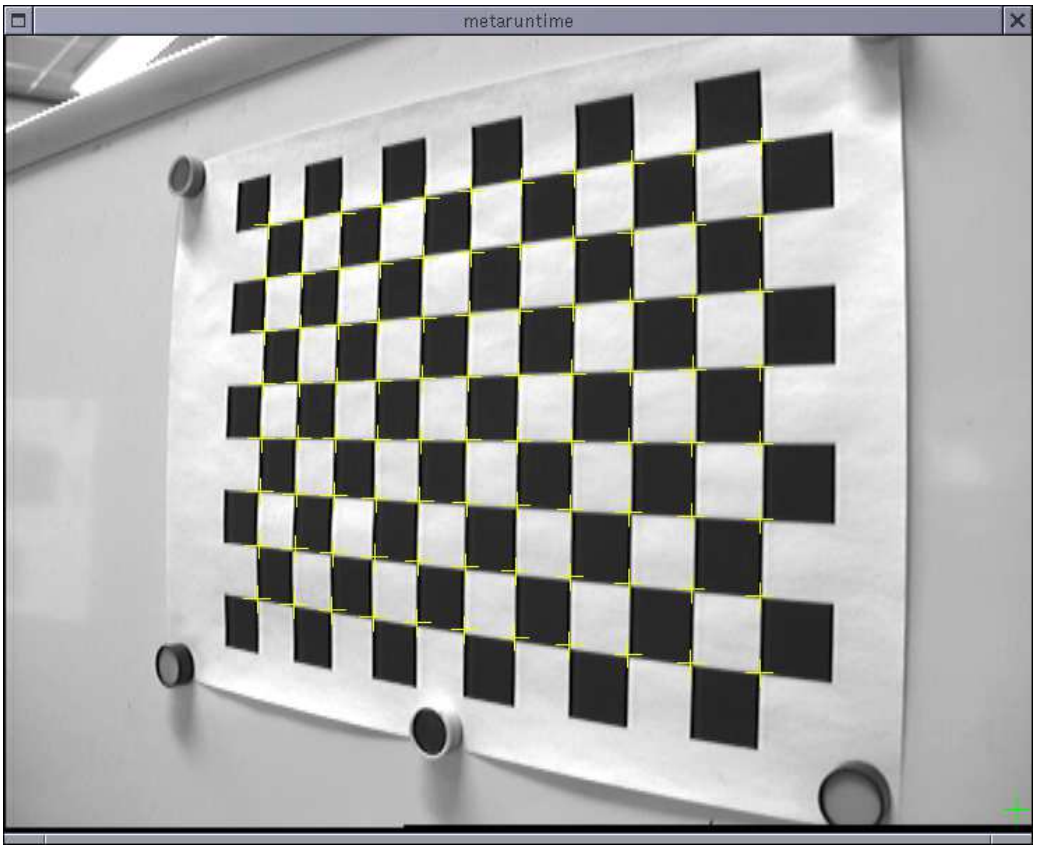
Il s'agit du modèle qui traduit la transformation de projection perspective effectuée par la caméra. Nous ne tiendrons pas compte des distortions radiales et tangentielles de l'objectif de la caméra. Nous allons utiliser une présentation différente de la modélisation que nous avons donnés à la section 3.1.1.1 page 74. Dans son modèle le plus simple, c'est à dire en se plaçant dans le repère caméra, la relation qui lie un point $m$ du plan image de coordonnées $(x, y)$ à un point $M$ dans l'espace de coordonnées $(X, Y, Z)$ est de la forme :

$$x = \frac{f}{Z}X \quad (3.31)$$

$$y = \frac{f}{Z}Y \quad (3.32)$$




(a) Algorithme barycentrique simple




(b) Algorithme barycentrique perspectif

FIG. 3.7 – Application des algorithmes d'échantillonnage (en rouge apparaissent les points échantillonnés)

Une fois le point projeté, il nous faut encore appliquer la transformation affine qui permet de passer du point $(x, y)$ du plan image au repère image réel de la caméra (repère $(u, v)$) :

$$u = k_u x + u_0 \tag{3.33}$$
$$v = k_v y + v_0 \tag{3.34}$$

D'où nous pouvons directement déduire la relation :

$$fX = \frac{Z}{k_u}(u - u_0) \tag{3.35}$$
$$fY = \frac{Z}{k_v}(v - v_0) \tag{3.36}$$

Nous allons utiliser ce modèle afin d'affecter des profondeurs arbitraires à nos points $A, B, C$ et $D$.

**Affectation de profondeurs arbitraires**

Comme nous avons un rectangle, nous pouvons écrire la propriété suivante :

$$\overrightarrow{AB} = \overrightarrow{DC}$$

Ce qui se traduit par :

$$\begin{pmatrix} X_B - X_A \\ Y_B - Y_A \\ Z_B - Z_A \end{pmatrix} = \begin{pmatrix} X_C - X_D \\ Y_C - Y_D \\ Z_C - Z_D \end{pmatrix} \tag{3.37}$$

Nous multiplions les équations concernant les coordonnées $X$ et $Y$ par $f$, ce qui donne :

$$\begin{pmatrix} fX_B - fX_A \\ fY_B - fY_A \\ Z_B - Z_A \end{pmatrix} = \begin{pmatrix} fX_C - fX_D \\ fY_C - fY_D \\ Z_C - Z_D \end{pmatrix} \tag{3.38}$$

Il suffit alors de réinjecter les relations empruntées au sténopé linéaire :

$$\begin{pmatrix} \frac{Z_B}{k_u}(u_B - u_0) - \frac{Z_A}{k_u}(u_A - u_0) \\ \frac{Z_B}{k_v}(v_B - v_0) - \frac{Z_A}{k_v}(v_A - v_0) \\ Z_B - Z_A \end{pmatrix} = \begin{pmatrix} \frac{Z_C}{k_u}(u_C - u_0) - \frac{Z_D}{k_u}(u_D - u_0) \\ \frac{Z_C}{k_v}(v_C - v_0) - \frac{Z_D}{k_v}(v_D - v_0) \\ Z_C - Z_D \end{pmatrix} \tag{3.39}$$

Si nous prenons la première composante de la dernière égalité, nous remarquons que nous pouvons la simplifier par $k_u$. En arrangeant un peu les termes nous obtenons :

$$u_B Z_B - u_A Z_A - u_0(Z_B - Z_A) = u_C Z_C - u_D Z_D - u_0(Z_C - Z_D) \tag{3.40}$$

Or, d'après l'égalité $Z_B - Z_A = Z_C - Z_D$ que nous avons par ailleurs, les termes en $u_0$ se simplifient de chaque côté. Nous pouvons faire de même avec la deuxième composante de l'égalité. Au final, l'équation 3.39 se résume à :

$$\begin{pmatrix} u_B Z_B - u_A Z_A \\ v_B Z_B - v_A Z_A \\ Z_B - Z_A \end{pmatrix} = \begin{pmatrix} u_C Z_C - u_D Z_D \\ v_C Z_C - vD Z_D \\ Z_C - Z_D \end{pmatrix} \tag{3.41}$$

Nous pouvons fixer arbitrairement $Z_A$ à 1, pour obtenir $Z_B, Z_C$ et $Z_D$, il nous faut alors résoudre le système suivant :

$$\begin{pmatrix} u_B & -u_C & u_D \\ v_B & -v_C & v_D \\ -1 & 1 & -1 \end{pmatrix} \begin{pmatrix} Z_B \\ Z_C \\ Z_D \end{pmatrix} = \begin{pmatrix} v_A \\ u_A \\ -1 \end{pmatrix} \tag{3.42}$$

$$M \begin{pmatrix} Z_B \\ Z_C \\ Z_D \end{pmatrix} = \begin{pmatrix} v_A \\ u_A \\ -1 \end{pmatrix} \tag{3.43}$$

Si on résout le système, nous obtenons alors les profondeurs relatives suivantes :

---

$$Z_B = \frac{1}{\det M}[u_A(v_C - v_D) + v_A(u_D - u_C) - (u_C v_D - u_D v_C)] \tag{3.44}$$

$$Z_C = \frac{1}{\det M}[u_A(v_B - v_D) + v_A(u_D - u_B) + (u_D v_B - u_B v_D)] \tag{3.45}$$

$$Z_D = \frac{1}{\det M}[u_A(v_B - v_C) + v_A(u_C - u_B) - (u_B v_C - u_C v_B)] \tag{3.46}$$

$$\det M = (u_C v_D - v_C u_D) + (u_D v_B - u_B v_D) + (u_B v_C - u_C v_B) \tag{3.47}$$

$$\tag{3.48}$$

---

Ce calcul peut s'assimiler à celui qui est fait pour déterminer les homographies. En effet, il s'agit de l'étape préliminaire qui nous permet de calculer les points d'échantillonnage par la suite. Il est à noter qu'au lieu de résoudre un système d'équations à huit inconnues, il s'agit ici d'un système d'équation à trois inconnues dont la solution est simple à écrire.

**Calcul des coordonnées des points d'échantillonnage dans l'image**

Nous reprenons le point $M_{ij}$ de coordonnées $(u_{ij}, v_{ij})$ dans l'image et de coordonnées $(X_{ij}, Y_{ij}, Z_{ij})$ dans l'espace. Nous appliquons la formule barycentrique simple aux *coordonnées dans l'espace*, ce qui va donner :

$$X_{ij} = \frac{1}{mn}[X_A(m-i)(n-j) + X_B i(n-j) + X_C ij + X_D(m-i)j] \tag{3.49}$$

$$Y_{ij} = \frac{1}{mn}[Y_A(m-i)(n-j) + Y_B i(n-j) + Y_C ij + Y_D(m-i)j] \tag{3.50}$$

$$Z_{ij} = \frac{1}{mn}[Z_A(m-i)(n-j) + Z_B i(n-j) + Z_C ij + Z_D(m-i)j] \tag{3.51}$$

En reprenant ces équations et en exploitant les équations 3.35 et 3.36, puis en effectuant des simplifications qui sont du même ordre que celles des équations 3.39 à 3.41 nous obtenons :

$$u_{ij} Z_{ij} \frac{1}{f} = \frac{1}{f}\frac{1}{mn}[u_A Z_A(m-i)(n-j) + u_B Z_B i(n-j) + u_C Z_C ij + u_D Z_D(m-i)j] \tag{3.52}$$

$$v_{ij} Z_{ij} \frac{1}{f} = \frac{1}{f}\frac{1}{mn}[v_A Z_A(m-i)(n-j) + v_B Z_B i(n-j) + v_C Z_C ij + v_D Z_D(m-i)j] \tag{3.53}$$

$$Z_{ij} = \frac{1}{mn}[Z_A(m-i)(n-j) + Z_B i(n-j) + Z_C ij + Z_D(m-i)j] \tag{3.54}$$

Ce qui, après simplification et réécriture, donne :

---

$$u_{ij} = \frac{1}{mnZ_{ij}}\left[u_A Z_A(m-i)(n-j) + u_B Z_B i(n-j) + u_C Z_C ij + u_D Z_D(m-i)j\right] \quad (3.55)$$

$$v_{ij} = \frac{1}{mnZ_{ij}}\left[v_A Z_A(m-i)(n-j) + v_B Z_B i(n-j) + v_C Z_C ij + v_D Z_D(m-i)j\right] \quad (3.56)$$

$$Z_{ij} = \frac{1}{mn}\left[Z_A(m-i)(n-j) + Z_B i(n-j) + Z_C ij + Z_D(m-i)j\right] \quad (3.57)$$

---

Le résultat de l'application de cet algorithme en employant la première méthode d'estimation de la profondeur sur une zone rectangulaire est visible sur la figure 3.7(b) page 89. On remarque rapidement que les points résultants de l'échantillonnage sont très proche des coins du damier, ce qui montre l'amélioration de l'échantillonnage réalisé. Cet échantillonnage, on le rappelle, utilise une déformations perspective alors que les paramètres intrinsèques de la caméra ainsi que la position et l'orientation de la zone rectangulaire dans l'espace sont inconnus. Nous verrons dans le chapitre suivant quels sont les applications de cet algorithme original.

## 3.3.2 Calcul de la pose d'une cible carrée

Nous calculerons la pose de la caméra à l'aide des coordonnées des quatre coins de ce dernier. Comme tout calcul de pose, cela présuppose que la caméra a été calibrée et donc que ses paramètres intrinsèques sont connus. Nous reprendrons en partie les notations introduites dans la section précédente. En effet, un carré n'est qu'un rectangle particulier. Nous reprenons donc l'ordre des points tels que nous l'avions donné à la figure 3.6(a) page 87.

De même, nous avions introduit, dans les équations 3.44 à 3.47 page précédente, le calcul de profondeurs arbitraires données à chacun des sommets du rectangle. Ces profondeurs sont en réalité déterminées à un facteur d'échelle près par rapport aux profondeurs réelles. Nous notons $Z'_A$, $Z'_B$, $Z'_C$ et $Z'_D$ ces dernières. Il s'agit des profondeurs liées aux coins de la cible dans le repère de la caméra. L'une des premières étapes de calcul est la détermination de ces profondeurs réelles.

### 3.3.2.1 Calcul des profondeurs réelles

Nous notons $r1$ le rapport $Z_A/Z_C = Z'_A/Z'_C$. De même $r2$ est le rapport $Z_B/Z_D = Z'_B/Z'_D$. Au vu des équations 3.44 à 3.47 page précédente, les valeurs de ces rapports sont connus. Les paramètres intrinsèques de la caméra sont donnés, dans notre problème par la matrice :

$$M_{int} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

Ceci présuppose que, dans le cas où nous souhaitons tenir compte de la distortion radiale, celle-ci a déjà été compensée. Le raisonnement que nous tiendrons se fera sur les points $A$ et $C$, le raisonnement sur $B$ et $D$ se faisant de la même manière.

A l'aide de cette matrice et des relations du modèle du sténopé, nous pouvons écrire les relations :

$$X_A = \frac{Z'_A}{\alpha_u}(u_A - u_0) \qquad Y_A = \frac{Z'_A}{\alpha_v}(v_A - v_0)$$
$$X_C = \frac{Z'_C}{\alpha_u}(u_C - u_0) \qquad Y_C = \frac{Z'_C}{\alpha_v}(v_C - v_0)$$

Il suffit à présent, de calculer la norme de $\|\overrightarrow{AC}\|$ et de dégager les inconnues du problème, à savoir ici $Z'_A$.

$$\|\overrightarrow{AC}\| = \sqrt{(X_C - X_A)^2 + (Y_C - Y_A)^2 + (Z'_C - Z'_A)^2}$$

Or nous connaissons la géométrie de la cible (à tout le moins la taille de son côté) donc nous connaissons la valeur de cette norme. De même, les inconnues de cette équation sont $Z'_C$ et $Z'_A$ or nous connaissons la valeur du rapport de ces deux valeurs. En conclusion, nous pouvons écrire :

$$Z'_A = \frac{-\|\overrightarrow{AC}\|}{\sqrt{(r_1-1)^2 + \left(\frac{r_1(u_C-u_0)-(u_A-u_0)}{\alpha_u}\right)^2 \left(\frac{r_1(v_C-v_0)-(v_A-v_0)}{\alpha_v}\right)^2}}$$

Une fois $Z'_A$ calculé, nous pouvons obtenir $Z'_C$. De même, nous pouvons tenir le même raisonnement pour $Z'_C$ et $Z'_D$.

#### 3.3.2.2 Calcul de la pose

Une fois ces profondeurs connues, nous pouvons déterminer la translation et la rotation associée à notre cible dans le repère de la caméra. L'origine du repère de la cible est placé en son centre, à l'intersection des diagonales du carré. Enfin, le repère associé sera constitué de vecteurs colinéaires à $\overrightarrow{AB}$, $\overrightarrow{AD}$ et $\overrightarrow{AB} \wedge \overrightarrow{AD}$, ce qui nous donne également les composantes de la matrice de rotation. Pour une meilleure stabilité de la rotation, nous prendrons en réalité comme vecteurs : $\overrightarrow{AC}+\overrightarrow{DB}$,$\overrightarrow{AC}-\overrightarrow{DB}$ , $(\overrightarrow{AC}+\overrightarrow{DB}) \wedge (\overrightarrow{AC}-\overrightarrow{DB})$. Cette seconde formulation, permet de faire intervenir les positions des 4 coins de la cible dans le calcul de la rotation. La pose de la caméra par rapport à la cible se fait en calculant l'inverse de la transformation trouvée.

#### 3.3.2.3 Récapitulatif complet de la méthode

En résumé, pour calculer la pose de la caméra, les données utilisées sont :
- les paramètres intrinsèques de la caméra,
- les coordonnées des 4 coins de la cible dans l'image,
- la mesure réelle d'un des côtés du carré.

Nous noterons $O$ le centre de la cible et $R_{3\times3}$ la matrice qui décrit la rotation entre le repère de la caméra et le repère de la cible. Ses lignes seront notées $r_{1*}$, $r_{2*}$, $r_{3*}$. Les étapes de calcul sont alors résumées à la figure 3.9 page suivante. Nous avons également inclus quelques exemples d'augmentation obtenus à l'aide de cet algorithme de calcul de la pose à la figure 3.8. Une étude détaillée de cet algorithme sera présentée dans la section 3.5 page 104.

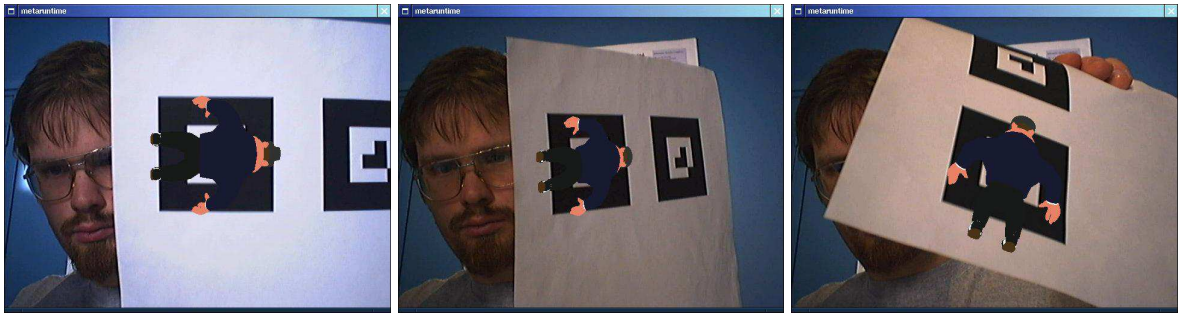


FIG. 3.8 – Exemples d'augmentation

Dans cette section, nous avons introduit deux nouveaux algorithmes dédiés aux systèmes de cible codées. L'un, est utilisé dans le cadre de l'échantillonnage de zone rectangulaire de l'espace. L'autre, est employé pour calculer la position et l'orientation d'une cible carrée par rapport à un repère lié à la caméra. Dans la section suivante, nous allons voir de quelle manière ces algorithmes ainsi que les connaissances théoriques présentées à la section 3.1 page 73 sont employées dans le cadre du développement de notre système de localisation par la vision.

$$
\begin{aligned}
Z_A &= 1 \\
Z_B &= \frac{1}{\det M}[u_A(v_C - v_D) + v_A(u_D - u_C) - (u_C v_D - u_D v_C)] \\
Z_C &= \frac{1}{\det M}[u_A(v_B - v_D) + v_A(u_D - u_B) + (u_D v_B - u_B v_D)] \\
Z_D &= \frac{1}{\det M}[u_A(v_B - v_C) + v_A(u_C - u_B) - (u_B v_C - u_C v_B)] \\
\det M &= (u_C v_D - v_C u_D) + (u_D v_B - u_B v_D) + (u_B v_C - u_C v_B) \\
r_1 &= \frac{Z_A}{Z_C} = \frac{Z'_A}{Z'_C} \\
r_2 &= \frac{Z_B}{Z_D} = \frac{Z'_B}{Z'_D} \\
Z'_A &= \frac{-\|\overrightarrow{AC}\|}{\sqrt{(r_1-1)^2 + \left(\frac{r_1(u_C-u_0)-(u_A-u_0)}{\alpha_u}\right)^2 \left(\frac{r_1(v_C-v_0)-(v_A-v_0)}{\alpha_v}\right)^2}} \\
Z'_B &= \frac{-\|\overrightarrow{BD}\|}{\sqrt{(r_1-1)^2 + \left(\frac{r_1(u_D-u_0)-(u_B-u_0)}{\alpha_u}\right)^2 \left(\frac{r_1(v_D-v_0)-(v_B-v_0)}{\alpha_v}\right)^2}} \\
& Z'_C = \tfrac{Z'_A}{r_1} \qquad Z'_D = \tfrac{Z'_B}{r_2} \\
& X_A = \tfrac{Z'_A}{\alpha_u}(u_A - u_0) \quad Y_A = \tfrac{Z'_A}{\alpha_v}(v_A - v_0) \\
& X_B = \tfrac{Z'_B}{\alpha_u}(u_B - u_0) \quad Y_B = \tfrac{Z'_B}{\alpha_v}(v_B - v_0) \\
& X_C = \tfrac{Z'_C}{\alpha_u}(u_C - u_0) \quad Y_C = \tfrac{Z'_C}{\alpha_v}(v_C - v_0) \\
& X_D = \tfrac{Z'_D}{\alpha_u}(u_D - u_0) \quad Y_A = \tfrac{Z'_D}{\alpha_v}(v_D - v_0) \\
X_O &= (X_A + X_B + X_C + X_D)/4 \\
Y_O &= (Y_A + Y_B + Y_C + Y_D)/4 \\
Z_O &= (Z'_A + Z'_B + Z'_C + Z'_D)/4 \\
r_{1*} &= \frac{\overrightarrow{AC} + \overrightarrow{DB}}{\|\overrightarrow{AC} + \overrightarrow{DB}\|} \\
r_{2*} &= \frac{\overrightarrow{AC} - \overrightarrow{DB}}{\|\overrightarrow{AC} - \overrightarrow{DB}\|} \\
r_{3*} &= r_{1*} \wedge r_{2*}
\end{aligned}
$$

FIG. 3.9 – Estimation complète de la position et de l'orientation de la cible par rapport au repère métrique lié à la caméra

## 3.4 Réalisation du système de localisation par la vision

Dans cette section, nous construirons notre propre système de localisation par la vision à base de cibles codées.

La première étape consiste à calibrer la caméra, étape pour laquelle nous proposons une méthode semi-automatique dans laquelle l'intervention de l'homme est relativement minime. Cette procédure de calibration utilise la méthode de calibration de Zhang ainsi que notre algorithme d'échantillonnage de surfaces rectangulaires. Puis, nous exposerons sommairement les composants développés et intégrés à l'architecture décrite au chapitre 2.4 page 50 pour cette première tâche ainsi que la manière dont ils collaborent.

Ensuite, nous évoquerons le système de localisation en lui-même. Nous exposerons la structure des cibles employées ainsi que les traitements nécessaires à la reconnaissance de ces dernières. Dans ce cadre, nous appliquerons à nouveau notre algorithme d'échantillonnage perspectif. De plus, nous utiliserons notre algorithme de calcul de la pose. Enfin, nous examinerons également les composants développés pour mettre en oeuvre le système de localisation par la vision.

### 3.4.1 Méthode de calibration semi-automatique de la caméra

Pour effectuer la calibration, nous utilisons un damier rectangulaire constitué de cases carrées noires et blanches. L'utilisateur positionne la caméra de manière à voir le damier dans son intégralité tout en couvrant au maximum la surface de l'image. Lorsqu'il estime que la prise de vue est bonne, il le signale au système de calibration qui va détecter les coins du damier. S'il parvient à le faire, alors le système calcule également les coordonnées 3D associées au coins détectés dans l'image. Il prévient l'utilisateur que la détection s'est bien déroulée (voir figure 3.10 page suivante). Dans ce cas, l'utilisateur est invité à prendre une autre vue du damier sous un angle différent. Cette étape doit être répétée au minimum deux fois pour satisfaire aux conditions d'application de la méthode de calibration de Zhang. Dans le cas où une détection s'est mal déroulée, l'utilisateur est simplement invité à reprendre une vue du damier. Quand le système obtient un nombre d'images suffisant (dans notre cas au minimum 3 prises sous un angle différent), il identifie les paramètres intrinsèques de la caméra à l'aide de la méthode de Zhang présentée en section 3.1.1.2 page 76. Le capteur est alors calibré en prenant en compte les paramètres de distortion radiale afin de stabiliser la valeur des paramètres intrinsèques de la caméra [Vigueras-Gomez et al., 2005].

Pour mener à bien cette tâche, nous avons utilisé la librairie OpenCV, qui est une librairie de traitement d'images. Cette dernière possède notamment, parmi ses fonctionnalités, une implémentation de l'algorithme de calibration de Zhang ainsi que des fonctions pour détecter les coins internes d'un damier. Nous allons voir comment nous exploitons ces fonctions dans le cadre de notre application, ainsi que les composants qui en découlent et la manière de les faire collaborer entre eux.

#### 3.4.1.1 Réordonnancement automatique des coins d'un échiquier

Comme nous l'avons évoqué, la librairie OpenCV comporte une fonction de détection automatique de damiers. Or, cette fonction échoue à réordonnancer les coins détectés une fois sur deux environ. De ce fait, il est difficile d'attribuer de manière automatique une coordonnée 3D à l'un des coins détectés.

Pour résoudre ce problème, il nous faut d'abord trouver les coins extérieurs de l'échiquier puis ensuite réordonnancer les points détectés.

**Détection des coins extérieurs**

L'ensemble de points 2D trouvés s'inscrit à peu de choses près dans un quadrilatère. En réalité, à

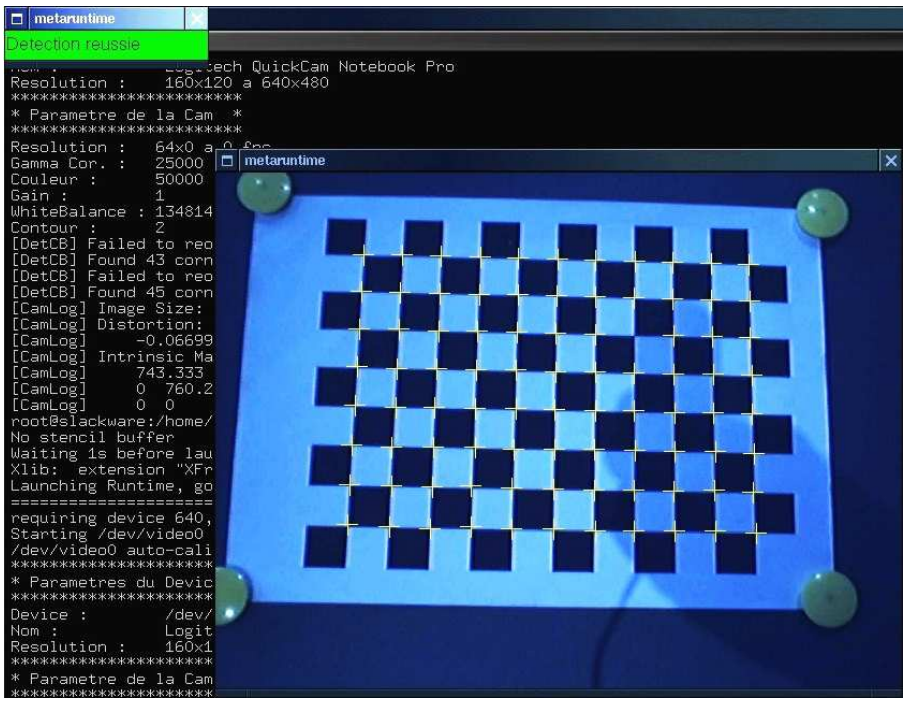


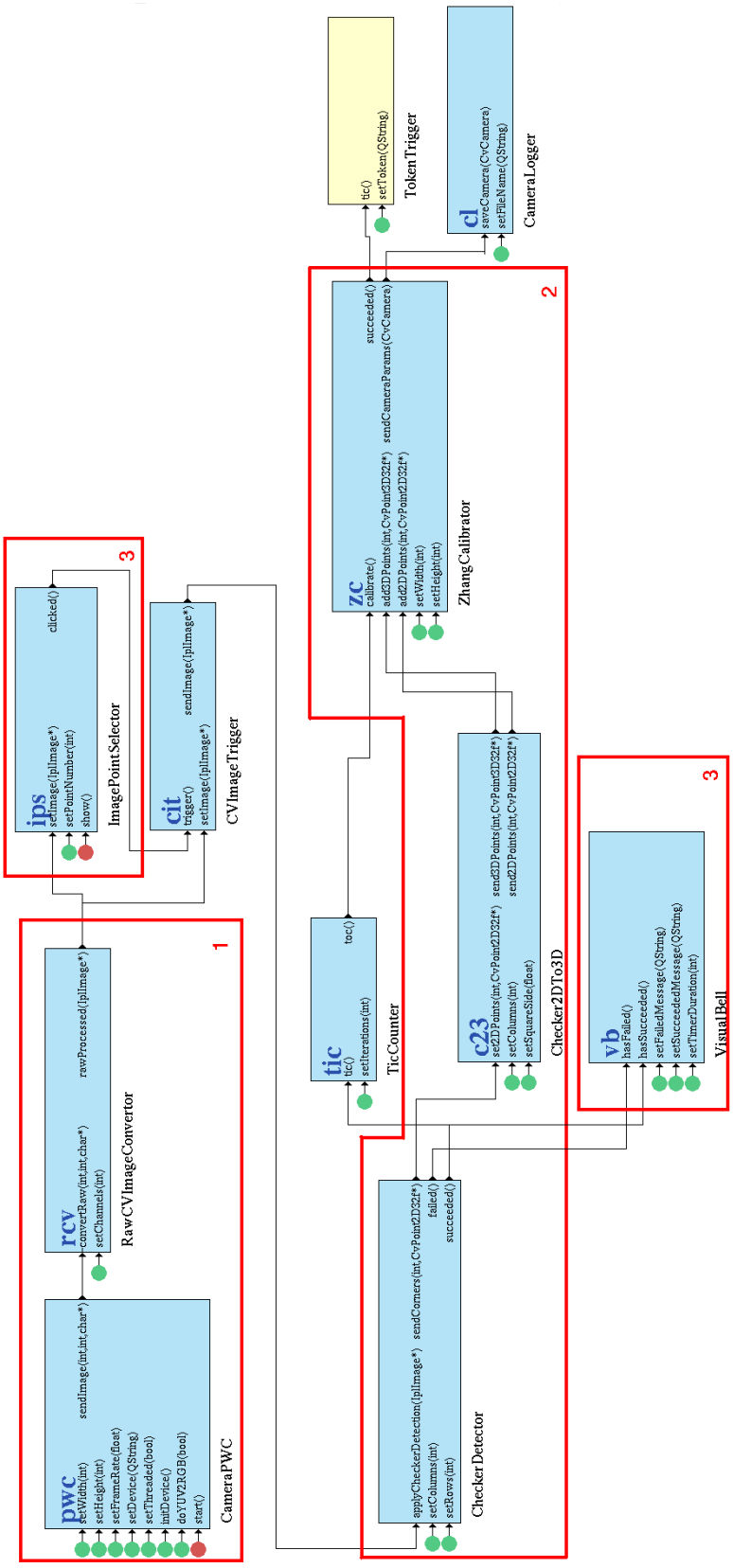

FIG. 3.10 – Exemple de détection des coins intérieurs de l'échiquier (en haut à gauche le système signale à l'utilisateur que la prise de vue a été réussie)

l'aide d'OpenCV, nous pouvons trouver les points situés sur l'enveloppe convexe de cet ensemble de points. Le parcours du contour trouvé en calculant le changement de direction nécessaire pour passer d'un point à un autre nous permet de sélectionner les coins extérieurs puisque le changement de direction sera nettement plus important que pour les autres points.

Il suffit ensuite d'identifier ces 4 points avec nos points $A, B, C$ et $D$, ce qui est fait en prenant par convention le point $A$ en haut à gauche, le point $B$ en haut à droite, le point $C$ en bas à droite et le point $D$ dans le dernier quartier restant. Ceci nous permet d'imposer une contrainte d'ordre sur les coins restant à ordonner. Ceci présuppose que le damier reste dans le même sens, c'est à dire que la caméra ne tourne que faiblement autour de la normale principale au plan image.

**Réordonnancement des points**

C'est à ce niveau que va intervenir une première fois notre algorithme d'échantillonnage introduit dans la section 3.3.1 page 87. A ce stade, nous avons détecté l'ensemble des coins intérieur du damier et déterminé les 4 coins de la zone rectangulaire dans laquelle ils s'inscrivent. Une fois les 4 coins trouvés, nous pouvons calculer les points d'échantillonnage de notre grille (les $m_{ij}$) sur l'image et prendre le coin intérieur du damier qui est le plus proche de $m_{ij}$. Nous prendrons comme système de coordonnées dans l'espace 3D $(A, \vec{i}, \vec{j}, \vec{i} \wedge \vec{j})$, $\vec{i}$ étant colinéaire à $\overrightarrow{AB}$ et $\vec{j}$ étant colinéaire à $\overrightarrow{AD}$. A un coin intérieur du damier, on associera les coordonnées 3D $(X_{ij} = k.m.i, Y_{ij} = k.n.j, Z_{ij} = 0)$ avec $k$ la longueur d'un côté d'un carré élémentaire constituant le damier. Nous obtenons ainsi une coordonnée 3D qui est associée à chacun des points 2D ainsi réordonnancés. Bien entendu, cette méthode ne s'applique que si les composantes de déformation radiales et tangentielles de la lentille (voir section 3.1.1.1 page 75 à ce sujet) restent dans des proportions raisonnables. Elle peut devenir inopérante en cas d'utilisation de lentilles de type "fish-eye" qui possèdent un grand angle d'ouverture et donc de fortes déformations radiales.




**Légende**

1. Composants gérant l'acquisition de l'image,
2. Composants dédiés au problème de la calibration,
3. Composants de l'interface utilisateur.

FIG. 3.11 – Feuille regroupant les composants développés dans le cadre d'une calibration semi-automatique

#### 3.4.1.2 Application à l'architecture par composants

La figure 3.11 page précédente donne la représentation d'une feuille regroupant les composants intervenant dans la procédure semi-automatique de calibration de la caméra. Sur cette dernière, quatre types de composants sont employés :

1. Les composants associés à l'acquisition d'image,

   **CameraPWC** est un module d'acquisition des images à partir d'une webcam dans un format brut (nous verrons que nous avons également des composants d'acquisition pour d'autres types de caméra),

   **RawCVImageConvertor** convertit les images dans un format brut vers un format compatible avec la librairie OpenCV.

2. Les composants spécifiques au système de calibration,

   **CheckerDetector** est le détecteur de damiers,

   **Checker2Dto3D** est le module de réordonnancement des points qui affecte des coordonnées 3D aux coins du damier détecté,

   **ZhangCalibrator** contient la méthode de calibration de Zhang.

3. Les composants de l'interface utilisateur,

   **ImagePointSelector** affiche le flux vidéo et permet à l'utilisateur de sélectionner l'image qu'il souhaite utiliser pour la prise de vue,

   **VisualBell** affiche un message à l'utilisateur lui indiquant si l'extraction des points pour la calibration s'est bien déroulée ou non.

4. Les composants annexes,

   **CVImageTrigger** envoie à la demande une image présentée en entrée sur sa sortie. C'est ce qui est fait quand l'utilisateur sélectionne une image,

   **TicCounter** compte ici le nombre de prises de vues réussies, lorsque le nombre voulu est atteint, c'est lui qui va déclencher l'opération de calibration proprement dite en envoyant un signal au composant **ZhangCalibrator**.

   **TokenTrigger** indique à l'automate que l'opération de calibration est finie et qu'il peut réarranger les composants pour passer à l'état suivant de l'application,

   **CameraLogger** va entreposer les paramètres de calibration de la caméra dans un fichier pour une utilisation ultérieure.

Le composant **TokenTrigger** peut signaler à l'automate qui gère l'application de passer à une autre feuille avec un autre arrangement de composants. Ce pourrait être, par exemple, une feuille représentant une démonstration de réalité augmentée s'appuyant sur les cibles codées telle que nous allons la décrire à présent.

### 3.4.2 Système de localisation à l'aide de cibles codées

Nous présentons ici le système de cibles codées que nous employons. Nous commencerons par décrire leur caractéristique géométrique, puis comment nous les repérons dans l'image, déterminons le code et calculons la pose de la cible. De plus, nous montrerons comment nous pouvons accélérer la vitesse de détection des cibles en tirant parti de la cohérence temporelle des images d'un flux vidéo. Enfin, nous montrerons quels sont les composants développés pour notre architecture générique.

#### 3.4.2.1 Métrique des cibles

Nos cibles ont une géométrie identique à celle employées dans la librairie ARToolkit. Elles sont carrées, avec un pourtour noir d'une épaisseur de taille équivalente à un quart du côté de la cible. A l'intérieur, une autre zone quadrillée suivant une grille de taille 4x4 donne le code de la cible comme nous pouvons le constater sur la figure 3.12.

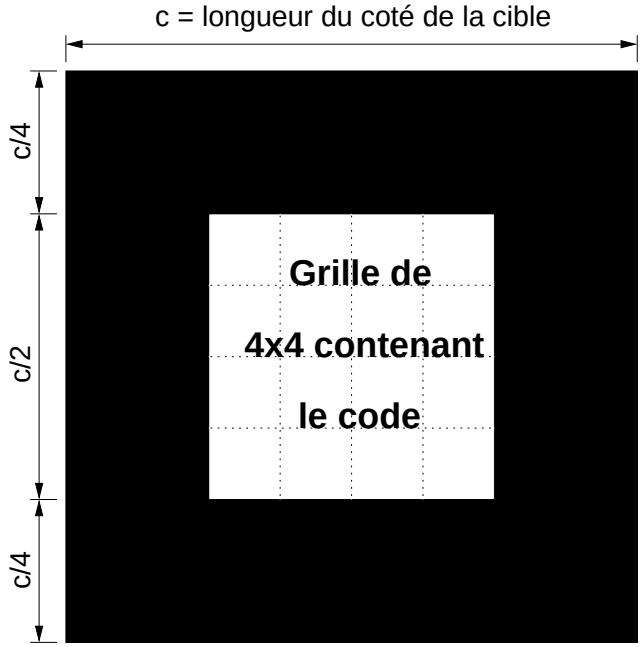



FIG. 3.12 – Métrique des cibles codées employées

La zone centrale renferme donc 16 carrés qui peuvent être soit noir, soit blanc. Nous choisissons de manière logique une représentation binaire du code de la cible qui est $blanc = 1$ et $noir = 0$. Nous obtenons un code sur 16 bits pour chacune des cibles.

#### 3.4.2.2 Contraintes liées aux cibles codées

En théorie, notre code sur 16 bits devrait donner $2^{16}$ possibilités soit 65536 cibles différentes. Toutefois, le système de cibles doit respecter une contrainte forte, qui est de permettre la détection de l'orientation de la cible, ce qui veut dire que chaque cible qui est tournée d'un quart de tour et qui, à la phase d'identification, possède un code différent est en réalité une autre instance de la même cible. L'instance principale de la cible sera celle qui aura le code le plus petit. Nous appelons ces ensembles de 4 codes des *classes d'équivalence* telle que celle présentée en exemple à la figure 3.13 page suivante.

En conséquence, cela divise par 4 le nombre de classes d'équivalence que l'on peut attribuer par rapport au nombre de code possibles. De plus, les cibles ne doivent pas présenter de symétrie centrale car nous ne pourrions pas distinguer de code plus petit qu'un autre et donc connaître l'orientation de la cible. La figure 3.14 page suivante montre quelques unes des cibles que nous ne pouvons pas employer dans notre système.

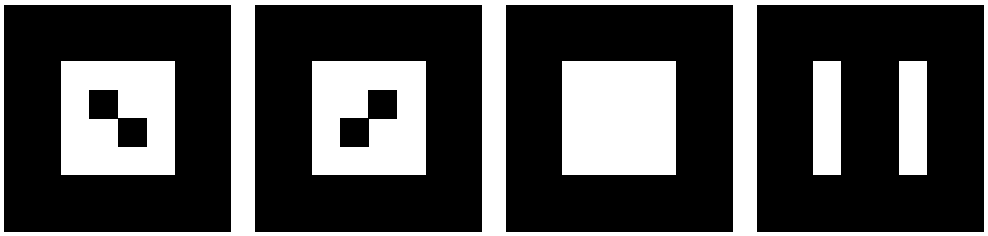
En définitive, nous obtenons 16320 classes d'équivalence en utilisant un algorithme de force brute qui vérifie pour chaque code de 1 à 655535 si ce dernier est ne présente pas d'invariance de rotation.

#### 3.4.2.3 Méthode d'extraction des cibles dans l'image

La méthode d'extraction des cibles est similaire à celle des autres systèmes cités dans la section 3.2.1 page 83. Nous effectuons les traitements suivants :

(a) code 56831 (b) code 62463 (c) code 65467 (d) code 65487

FIG. 3.13 – Exemple de classe d'équivalence : la classe d'équivalence 56831



(a) code 64479 (b) code 64959 (c) code 65535 (d) code 39321

FIG. 3.14 – Exemples de codes rejetés

1. Binarisation de l'image,
2. Extraction des contours,
3. Approximation polygonale des contours détectés,
4. Pour chaque approximation polygonale qui se révèle être un quadrilatère : discrimination et identification de la cible, le cas échéant.

Les trois permières étapes sont des traitements fournis dans la librairie OpenCV. L'étape de discrimination et d'identification de la cible s'effectue de la manière suivante : le quadrilatère détecté est supposé être le contour extérieur de la cible. Les 4 coins de ce quadrilatère sont donc les 4 coins extérieurs de notre cible. Si les sommets du quadrilatère ne sont pas ordonnés dans le sens horaire, nous les réordonnons dans le bon sens.

Nos cibles constituent des ensembles de 64 cellules carrées (16 pour le code et 48 pour la bordure). Pour déterminer le code, nous devons échantilloner au centre des cellules et non pas sur le bord de ces dernières. Aussi, à partir des 4 coins extérieurs, nous appliquons une première fois notre algorithme d'échantillonnage pour déterminer les centres des 4 cellules qui vont servir de points externes à la grille d'échantillonnage du code de la cible.

Ensuite, nous calculons, en employant l'algorithme d'échantillonnage prenant en compte les déformations dues à la projection perspective évoqué en section 3.3.1 page 87 une grille de 8x8 qui va venir sur le quadrilatère. Sur ces 64 points, les 48 du pourtour doivent être noirs alors que les 16 du centre fournissent un code valide comme le montre la figure 3.15 page ci-contre. Si ces conditions ne sont pas respectées, alors le quadrilatère est rejeté. Dans l'autre cas, la cible est reconnue.

#### 3.4.2.4 Calcul de la pose

A l'issue de l'application de la méthode de reconnaissance de la cible, si celle-ci contient un code valide, nous calculons la position et l'orientation de la cible par rapport au repère de la caméra. La méthode employée est celle décrite précédemment à la section 3.3.2 page 92. Une évaluation en détail de cette méthode est proposée en 3.5.2 page 108.

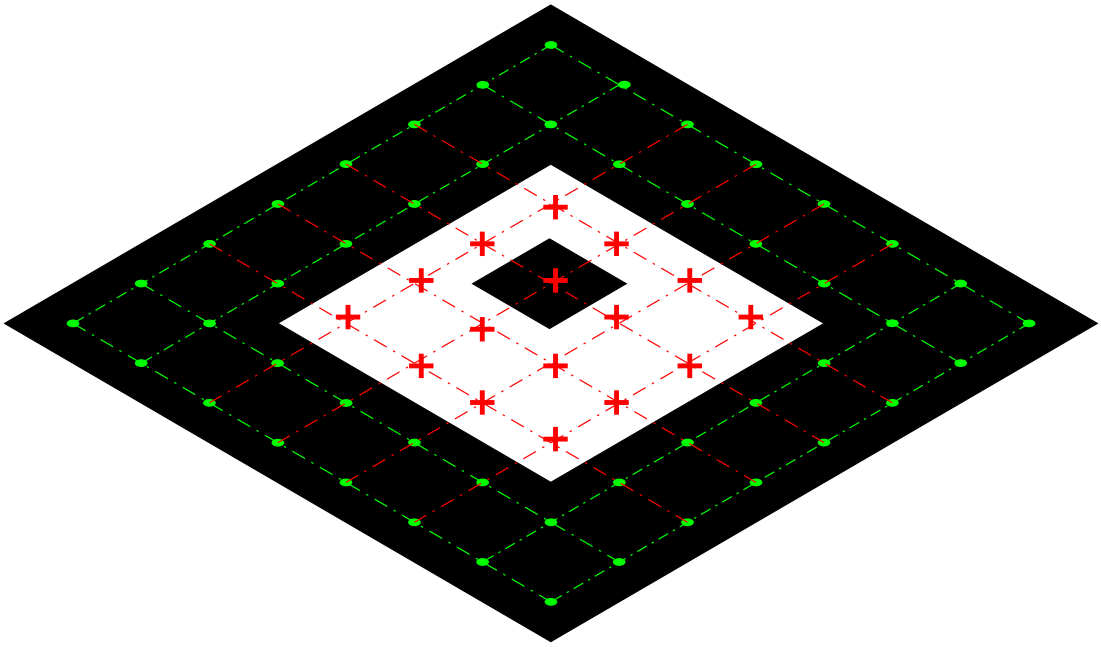


FIG. 3.15 – Grille de discrimination. Les croix rouges sont les points dédiés à la reconnaissance du code. En vert, les points dédiés à la discrimination de la cible.

#### 3.4.2.5 Méthodes de suivi des cibles

L'ensemble des méthodes d'extraction des cibles dans l'image en font une opération coûteuse en termes de temps de calculs. Il est possible d'accélérer ces traitements en tirant partie de la cohérence temporelle qui lie deux images d'une séquence vidéo. Nous avons implémenté deux méthodes permettant de suivre une cible d'une image sur l'autre (voir figure 3.16 page suivante).

La première, la plus simple, suppose que les variations d'une image sur l'autre sont faibles et donc que les coins de la cible à détecter sur la nouvelle image sont proches de l'endroit où ils étaient dans l'image précédente. Une fenêtre de recherche du coin d'une largeur de 20 pixels est utilisée mais cette méthode de suivi décroche très souvent.

La deuxième, plus complexe, se base sur le suivi des arêtes. Nous reprenons les coins détectés sur l'image suivante, ce qui nous donne un emplacement supposé de chaque arête sur la nouvelle image (en magenta sur la figure 3.16). Puis, pour chacune des arêtes, nous faisons un calcul du gradient suivant des directions perpendiculaires aux arêtes supposées, en partant de l'intérieur vers l'extérieur. Ces recherches sur des segments perpendiculaires sont effectuées à respectivement 1/4 et 3/4 de la distance du segment d'origine. La longueur du segment perpendiculaire est de 1/5 de la taille apparente des côtés de la cible sur l'image précédente, de part et d'autre de l'arête supposées. Si dans ce calcul du gradient photométrique nous obtenons une forte variation positive, alors il s'agit probablement de l'endroit où passe la nouvelle arête. Les nouvelles zones de recherche sont alors initialisées à l'intersection des 4 droites des nouvelles arêtes détectées (en brun sur la figure3.16).

Une fois les zones de recherche déterminées, nous effectuons une détection locale des coins puis nous réappliquons l'algorithme de discrimination et d'identification des cibles pour s'assurer que nous ne faisons par d'erreur. Dans le cas d'un décrochage, un traitement global pour détecter la cible sur l'image est à nouveau appliqué. Dans les premiers tests que nous avons effectués sur une machine pourvue d'un Pentium III cadencé à 1,1GHz nous avons observé que l'acquisition, le traitement, la localisation et l'affichage final de l'augmentation se faisait à une cadence de 12 images par seconde. Cette cadence passe à 18 images par secondes lorsque le suivi de cibles est activé. Cette première série de tests nous a permis aussi de choisir notre algorithme de suivi, le deuxième s'avérant plus

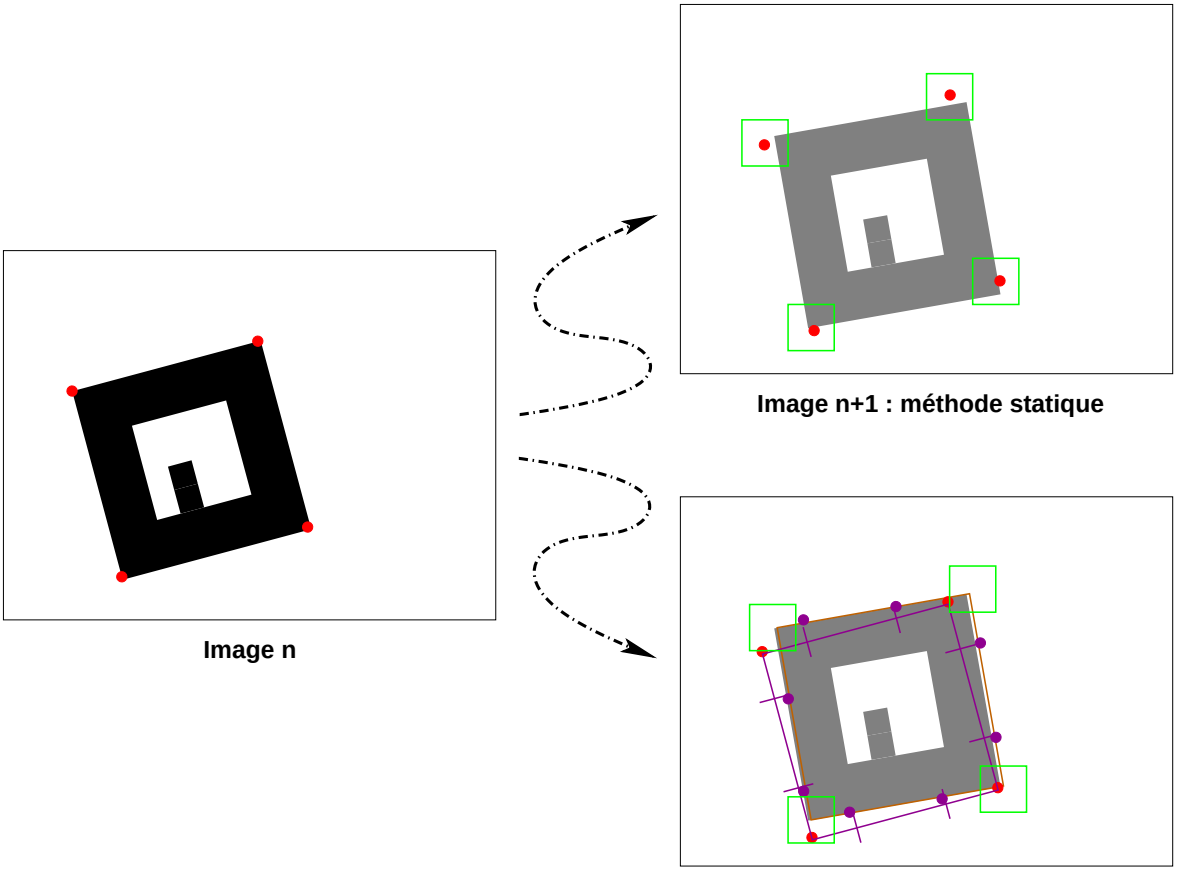



FIG. 3.16 – Méthodes de suivi des cibles d'une image sur l'autre d'une séquence vidéo. Les rectangles en vert indiquent les zones de recherche des coins de la nouvelle cible. Les points en rouge indiquent l'ancienne position des coins de la cible.

robuste.

#### 3.4.2.6 Application à l'architecture par composants

Comme dans le cas de la calibration semi-automatique, nous pouvons décomposer chacune des fonctionnalités en un ensemble de composants. La décomposition employée pour le système de localisation par la vision utilise certaines spécificités de l'architecture qui permettent de laisser le système ouvert à d'autres algorithmes de suivi de la cible d'une image à l'autre ainsi que d'autres méthodes de calcul de la pose de la caméra.

La figure 3.17 page ci-contre représente la feuille qui met en évidence les relations liant les divers composants. Ceux-ci, ici encore, appartiennent à quatre grandes familles de composants :

1. Les composants gérant l'acquisition de l'image,

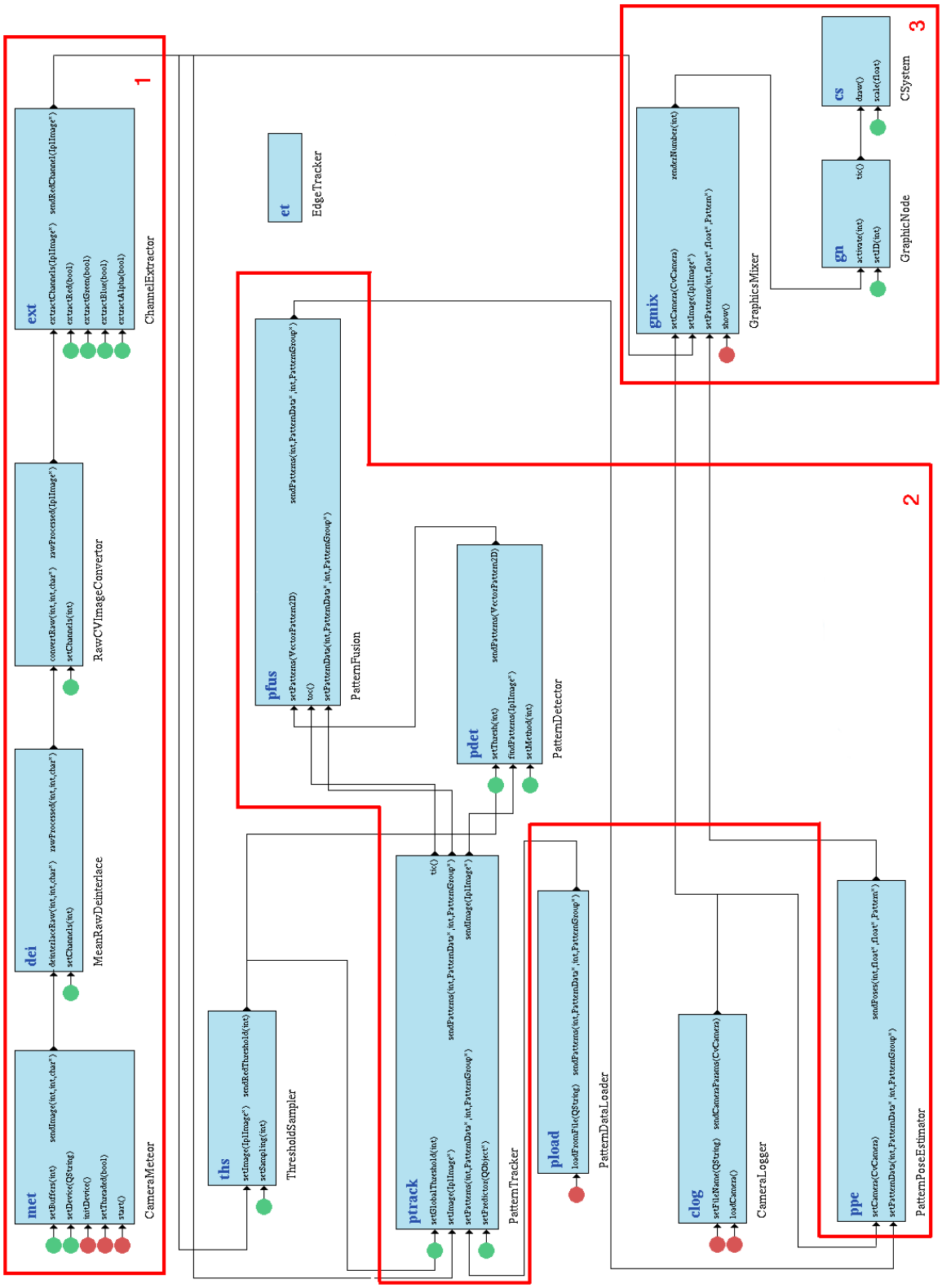
   **CameraMeteor** est un module d'acquisition des images à partir des cartes MeteorII du fabricant Matrox,

   **MeanRawDeinterlace** est un filtre de désentrelacement des images,

   **RawCVImageConvertor** convertit le format d'image au format de la librairie OpenCV,

   **ChannelExtractor** extrait une composante particulière du canal vidéo (pour des contraintes techniques liés au matériel employé [1]).

[1] La carte d'acquisition employée est une Meteor II/MC qui nous permet de multiplexer les signaux de différentes caméras filmant en noir et blanc au sein d'une seule image couleur.




**Légende**

1. Composants gérant l'acquisition de l'image,
2. Composants spécifiques au système de localisation par cible,
3. Composants gérant l'affichage des augmentations.

FIG. 3.17 – Détails architecturaux du système de localisation par la vision employant des cibles codées

2. Les composants spécifiques au système de localisation par cible,
   **PatternDetector** détecte les cibles par traitement d'image suivant la méthode exposée précédemment,
   **PatternTracker** contient la méthode générale permettant de suivre une cible d'une image sur l'autre,
   **EdgeTracker** est l'algorithme de suivi basé sur les lignes, il est composé implicitement (voir section 2.4.2.3 page 56) avec **PatternTracker**,
   **PatternFusion** fusionne les données issue de la détection avec celles issues du suivi,
   **PatternPoseEstimator** contient notre algorithme de calcul de la pose de cibles carrées. Son comportement peut également être changé par composition implicite pour obtenir l'estimation de la pose par d'autres méthodes de calcul.
3. Les composants gérant l'affichage des augmentations,
   **GraphicsMixer** est le composant d'affichage simultané du flux vidéo issu de la caméra et des augmentations à superposer,
   **GraphicNode** un composant activé suivant le code de la cible détectée. Son activation déclenche l'affichage d'une augmentation correspondant au code de la cible, ici l'augmentation est contenue dans le composant **CSystem**,
   **CSystem** est la représentation d'un repère en trois dimensions superposé à la cible.
4. Les composants annexes,
   **ThresholdSampler** est un composant de réglage automatique du seuil de binarisation nécessaire pour détecter la cible,
   **PatternDataLoader** charge des paramètres associés aux cibles qui seront employées dans l'application (taille de la cible, position et orientation de cette dernière par rapport au repère de l'objet à laquelle elle est associée, etc. ),
   **CameraLogger** utilisé ici pour charger les paramètres intrinsèques associés à la caméra.

Il est à noter que dans cet exemple nous n'avons pas utilisé les mêmes composants pour l'acquisition que dans l'exemple de calibration présenté auparavant. Il s'agit en réalité de l'architecture du système de vision pour une autre machine dont les périphériques d'acquisition des images ne sont pas les mêmes. Toutefois, grâce à notre architecture, nous pouvons transposer cette application d'un système à un autre sans difficultés, les entrées/sorties des composants du système d'acquisition étant les mêmes, ils sont interchangeables. Ceci illustre la modularité et l'adaptabilité de notre architecture en fonction des capteurs employés.

Nous venons de proposer, pour notre système de localisation par la vision, une méthode de calibration semi-automatique du capteur et un système de cibles codées reposant sur des algorithmes spécifiquement développés. Ce système tire partie des avantages offerts par l'architecture orientée composants que nous avions introduite. Elle permet également de laisser notre système ouvert, c'est à dire que l'on peut modifier la manière dont le suivi des cibles est effectué d'une image sur l'autre ainsi que les algorithmes de calcul de la pose utilisés. Nous allons exploiter ces propriétés dans la section suivante pour comparer et analyser les performances de notre système de cibles par rapport à des algorithmes existants.

## 3.5 Résultats expérimentaux

Cette section concerne l'évaluation détaillée de notre système de localisation par la vision. Une première évaluation en simulation portera sur les capacités de reconnaissance du système en fonction de l'inclinaison de la cible et la hauteur apparente de cette dernière dans l'image. Ensuite, nous

comparerons sur la base de données réelles les performances de l'algorithme de pose que nous avons introduit par rapport à deux autres algorithmes de référence : l'itération orthogonale qui a déjà été employée par Ababsa pour calculer la pose d'une cible carrée [Ababsa et Mallem, 2004] et la méthode des moindres carrés. Cette comparaison nous amènera à proposer une méthode hybride d'estimation de la pose. Enfin, nous donnerons quelques éléments de comparaison entre notre système et celui proposé dans la librairie ARToolkit.

## 3.5.1 Étude préliminaire des capacités de détection des cibles codées

Dans un premier temps, nous évaluerons les capacités de détection et de reconnaissance des cibles de notre système. Nous comparerons également trois méthodes différentes d'échantillonnage pour retrouver le code de la cible. Ces méthodes sont :

**Méthode 0** L'algorithme d'échantillonnage pondéré simple présenté dans la section 3.3.1.3 page 88,

**Méthode 1** Une méthode d'échantillonnage basée sur le calcul des profondeurs de manière empirique en se basant sur les longueurs apparentes des côtés de la cible dans l'image. En reprenant les notations employées à la figure 3.6(a) page 87, nous attribuons les profondeurs arbitraires suivantes : $Z_A = 1$, $Z_B = d/b$, $Z_C = (da)/(bc)$ et $Z_D = a/c$,

**Méthode 2** L'algorithme d'échantillonnage "perspectif" introduit dans la section 3.3.1.4.

Nous allons à présent donner la liste des paramètres retenus pour étudier la capacité de détection des cibles. Puis, nous présenterons le banc de simulation employé avant d'exposer les résultats et la conclusion de cette première étude.

### 3.5.1.1 Paramétrisation du problème

A présent que nous avons 3 méthodes différentes pour calculer la grille d'échantillonnage à appliquer, il nous faut les évaluer les unes par rapport aux autres. De plus, il est important de dégager les paramètres qui entrent en ligne de compte pour cette évaluation.

Par exemple, l'algorithme employé ne sera que peu affecté par une translation de la cible qui serait contenue dans un plan parallèle au plan image. Par contre, la hauteur apparente de la cible joue un rôle crucial dans l'évaluation de ces algorithmes (plus la cible semble éloignée, plus la détection est difficile). De même, l'orientation de la cible en roulis et tangage est importante alors que les résultats de l'algorithme vont être peu influencés par l'angle de lacet.

Pour nos tests, nous allons donc retenir trois paramètres qui sont par ailleurs représentés à la figure 3.18 page suivante :

- la hauteur apparente notée $\gamma$ qui est égale au quotient de la tangente de la hauteur de la cible et de la distance entre le plan parallèle au plan image contenant le centre de la cible et la caméra et la tangente de l'angle d'ouverture vertical de la caméra,
- l'angle $\theta$ qui est la rotation autour de l'axe $Ox$ qui amène $Oz$ en $Oz'$,
- l'angle $\theta$ qui est la rotation autour de l'axe $Oz$ qui amène $Oy$ en $Oy'$.

Nos paramètres étant retenus, nous pouvons démarrer le processus expérimental.

### 3.5.1.2 Évaluation de la reconnaissance des cibles par simulation

Nous avons décidé d'estimer les capacités de reconnaissance des cibles en simulation. Pour cela, nous avons généré des images de synthèse (comme celles de la figure 3.19 page suivante) représentant une cible pour diverses valeurs du paramètre $\gamma$ : $1, 5$, $2$, $3$, $4$, $5$ et $10$. Pour chacune de ces valeurs, un balayage exhaustif de degré en degré est effectué pour les angles de roulis $\phi$ et de tangage $\theta$. Il est à noter que nous exploitons aussi les propriétés de symétrie pour ne balayer qu'une partie des valeurs des angles de roulis et de tangage. Ces derniers seront compris entre $0^o$ et $90^o$ dans nos tests.

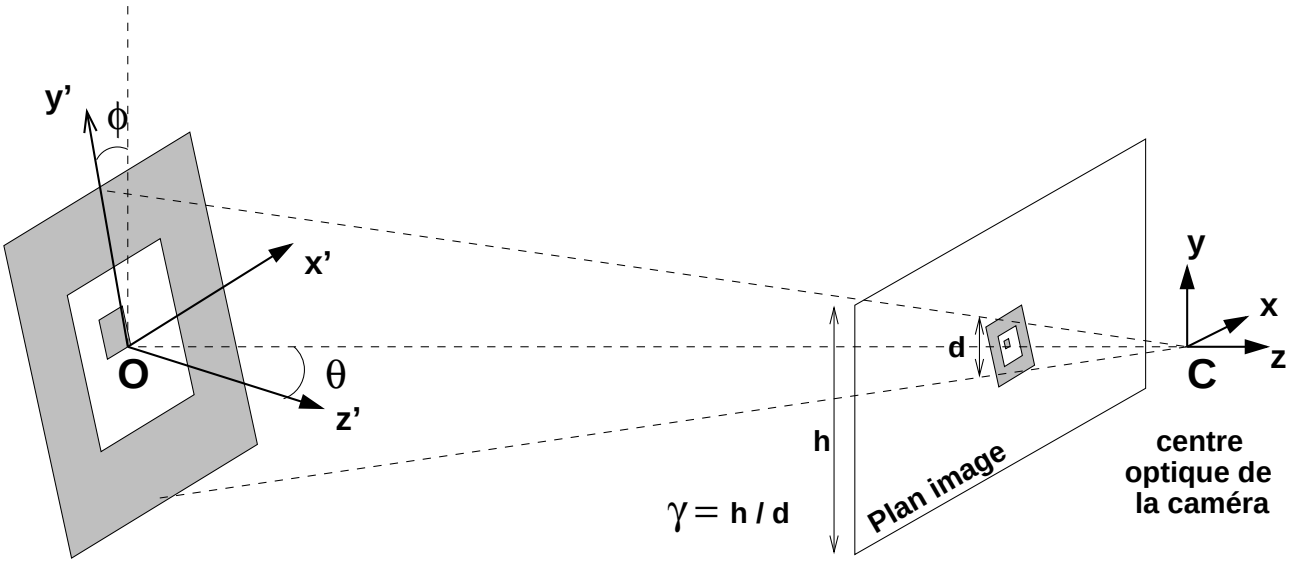
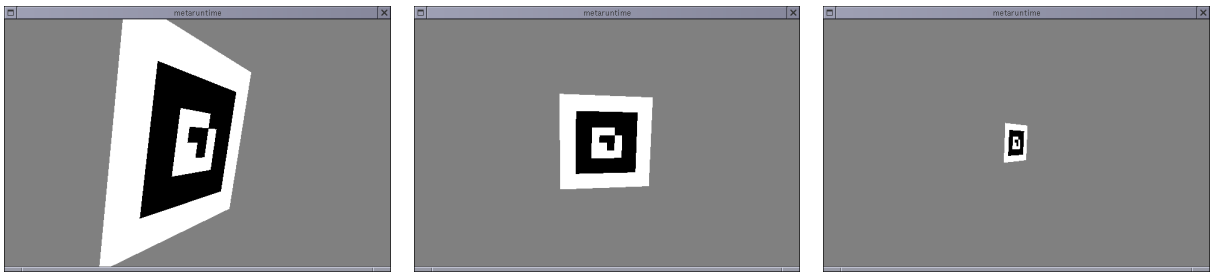



FIG. 3.18 – Paramètres employés pour évaluer la capacité du système à détecter les cibles codées



(a) Pour $\gamma = 2$ (b) Pour $\gamma = 4$ (c) Pour $\gamma = 10$

FIG. 3.19 – Exemples d'images de synthèse employées pour tester les performances en reconnaissance

S'il est vrai que l'utilisation d'images de synthèse nous place dans les meilleures conditions possibles pour la détection des cibles, cela nous permet aussi de connaître les capacités et les limites de détection intrinsèques de notre système sans perturbations comme les problèmes de variation de la luminosité, des ombres portées sur la cible ou même le bruit de mesure du à l'acquisition de l'image par un capteur physique. Ceci nous permet également d'automatiser le processus et parcourir de manière exhaustive tous les cas possibles.

#### 3.5.1.3 Résultats expérimentaux

A l'issue de nos tests, nous avons tracé pour chacune des valeurs du paramètre $\gamma$, les capacités des trois algorithmes d'échantillonnage à restituer le bon code pour la cible, pour chaque valeur des angles de roulis et de tangage. Nous avons obtenu la figure 3.20 page ci-contre.

Nous constatons, au premier abord, que les capacités des trois algorithmes sont assez inégales, notre méthode (la méthode 2), qui prend en compte les effets de perspective donnant les meilleurs résultats. Toutefois, nous remarquons que, plus nous éloignons la cible de la caméra, (ce qui est traduit par une hausse du paramètre $\gamma$), les divers algorithmes tendent vers les mêmes performances. Ceci est expliqué par le fait que les effets de perspective sont moins marqués sur une cible très éloignée.

Pour la méthode 2), la cible est détectée pour une inclinaison de $80^o$ lorsque la cible est proche et que cette inclinaison maximale décroît lentement jusqu'à $60^o$ pour une hauteur apparente de cible 10 fois moins grande que la hauteur de l'image. Ceci nous permet de connaître les limites de cet algorithme en matière de reconnaissance des codes des cibles.

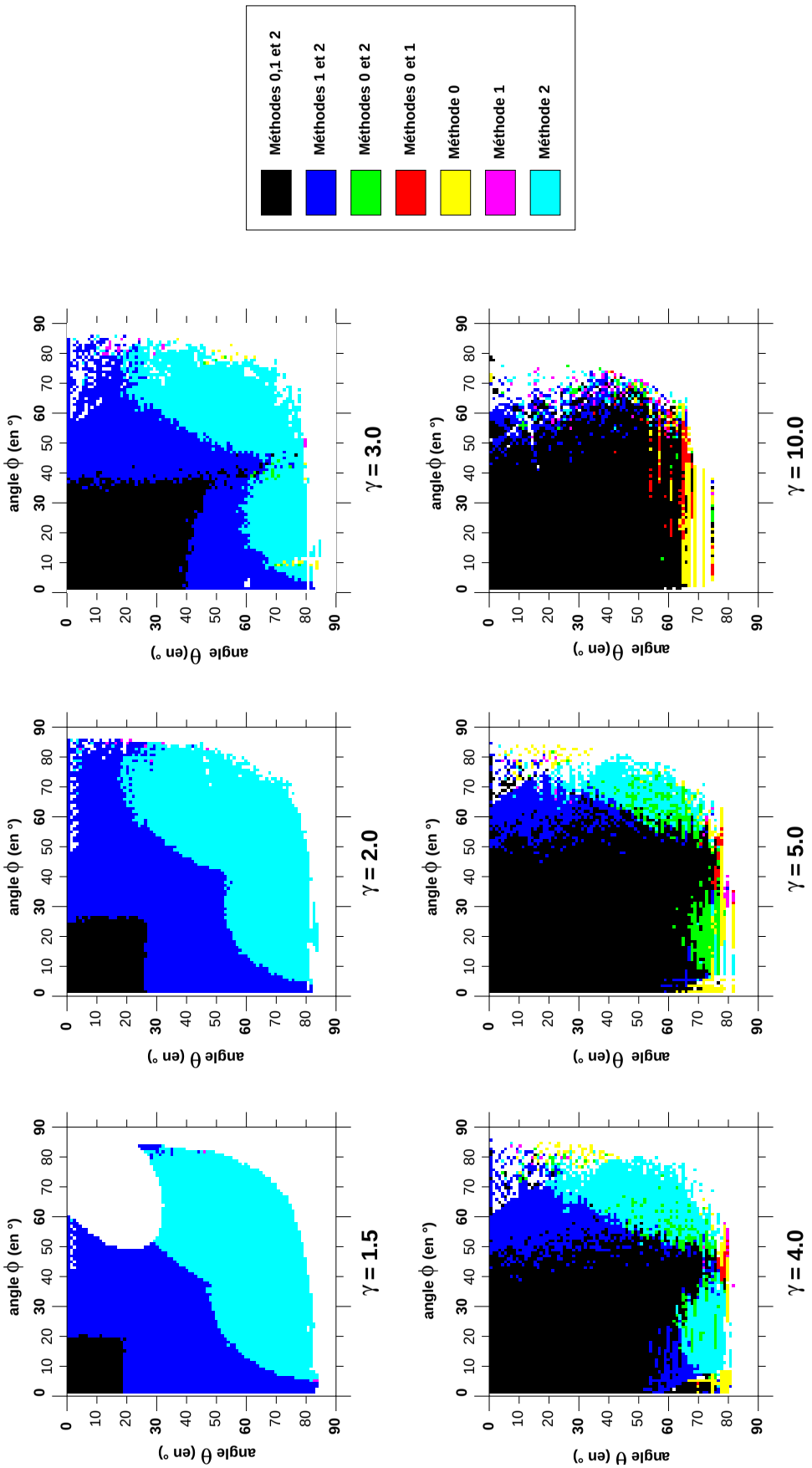



FIG. 3.20 – Capacité de reconnaissance suivant les 3 méthodes d'échantillonnage pour différentes valeurs de $\gamma$. Un point coloré indique que la cible a été reconnue par au moins un algorithme

Suivant la distance nominale à laquelle le système opère par rapport aux cibles, il est possible de sélectionner le type d'algorithme employé pour l'échantillonnage. Ainsi il serait possible de substituer à notre algorithme d'échantillonnage perspectif, le simple algorithme d'échantillonnage pondéré (décrit à la section 3.3.1.3 page 88) qui présente l'avantage d'être moins complexe, donc moins coûteux en temps de calcul.

Nous allons à présent examiner les performances de notre algorithme de calcul de la pose.

## 3.5.2 Évaluation des algorithmes de calcul de la pose par des données réelles

Afin de connaître les qualités et les défauts intrinsèques de notre algorithme spécifique de calcul de la pose (introduit dans la section 3.3.2 page 92), nous l'avons comparé avec deux autres algorithmes de référence. Le premier est l'itération orthogonale (IO) présentée en section 3.1.2.5 page 79. Celui-ci est, pour la première image, initialisé à l'aide de l'approximation de perspective faible. D'une image sur l'autre, la pose de l'image d'avant est reprise pour initialiser la pose de l'itération orthogonale sur l'image suivante. Le deuxième est la méthode des moindres carrés (MMC). Or, cette dernière doit employer un nombre de points supérieurs ou égal à 6 pour pouvoir effectuer une estimation de la pose de la cible. Nous avons donc, pour cette méthode, utilisé les coins internes à la cible dessinés par le code. Cette technique, dans le cadre de nos expériences avec nos cibles nous a permis d'utiliser en moyenne une quinzaine de points (le chiffre exact varie entre 12 et 25 suivant la cible) pour cet algorithme.

Nous avons ensuite établi une base de comparaison des algorithmes suivant plusieurs critères :
- Le temps d'exécution,
- L'erreur de reconstruction sur la cible,
- L'erreur de généralisation : pour une pose calculée sur une cible, de combien est l'erreur de reconstruction sur des cibles voisines ?
- L'erreur d'appréciation des distances métriques de chacun des algorithmes.

Les tests ont été menés sur deux types de configurations différentes :

**Configuration 1** : Xeon cadencé à 2,7 GHz, caméra USB de type CMOS,

**Configuration 2** : Pentium III cadencé à 1,1GHz, caméra Sony XC-555P utilisée avec une carte d'acquisition vidéo de type Matrox MeteorII.

### 3.5.2.1 Erreur de reconstruction

Pour cette expérimentation, nous promenons notre caméra à la main au dessus d'une cible parmi celles qui sont représentées à la figure 3.22 page 110. Lors de ce mouvement, nous calculons la pose de la caméra ainsi que l'erreur de reconstruction.

Le jeu de tests est constitué d'un ensemble de 685 poses calculées par chacun des 3 algorithmes. Ce jeu de valeurs a été obtenu sur la configuration 2. Pour chacun des calculs de pose, nous avons reprojeté le modèle de la cible sur le plan image et mesuré l'écart en pixels entre le coin réel de la cible et le coin reprojeté comme nous pouvons le voir à la figure 3.21 page suivante.

Les résultats obtenus sont regroupés dans le tableau 3.2 page ci-contre. Dans les valeurs données, certaines ont été retirées lorsqu'elles biaisaient les statistiques retournées (retrait des "outliers").

Nous remarquons que, conformément à nos attentes, l'algorithme spécifique développé est plus rapide que les autres méthodes employées. Si la méthode des moindres carrés semble dans notre cas plus lente, c'est aussi parce qu'elle nécessite la détection de coins supplémentaires, ce qui implique l'application d'opérateurs de traitement d'image auxiliaires, ce qui augmente d'autant le temps d'exécution de l'algorithme.

| **Type d'algorithme** | spécifique | IO | MMC |
|---|---|---|---|
| **Évaluation temporelle** | | | |
| Valeurs écartées ("outliers") | 6/685 | 47/685 | 0/685 |
| Temps d'exécution moyen ($\mu$s) | 22.30486 | 630.0094 | 15645.724 |
| Écart type sur temps d'exécution moyen | 12.944281 | 115.0231 | 4242.0378 |
| **Évaluation portant sur l'erreur de reconstruction** | | | |
| Valeurs écartées | 6/685 | 0/685 | 36/685 |
| Erreur de reconstruction moyenne (pixel) | 0.4415440 | 0.636916 | 1.1188554 |
| Écart type sur l'erreur de reconstruction moyenne | 0.3560174 | 0.4193855 | 0.8092120 |

TAB. 3.2 – Erreur de reconstruction moyenne sur 685 poses calculées

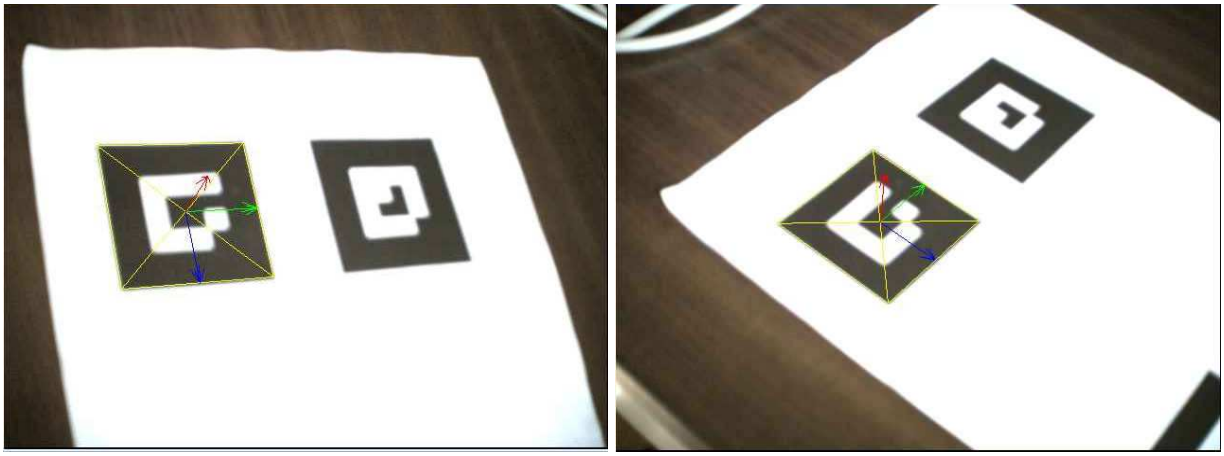


FIG. 3.21 – Exemples de recalage d'incrustations virtuelles sur des cibles codées

Du point de vue de l'erreur de reconstruction, notre algorithme spécifique fournit également des résultats meilleurs que les méthodes génériques. Nous verrons que tel n'est pas le cas lorsque l'on aborde le problème des erreurs de généralisation.

### 3.5.2.2 Erreur de généralisation

Pour cette série de tests, les valeurs ont été obtenues à l'aide de la configuration 1. Dans un premier temps, nous allons utiliser une feuille de papier sur laquelle sont imprimées quatre cibles de 5 centimètres de côté comme montré sur la figure 3.22 page suivante. Sur ces quatre cibles, une sert au calcul de la pose, les trois autres servent à calculer l'erreur de généralisation, c'est à dire l'erreur entre les reprojections de ces cibles et les coins réellement détectés. Pour ces tests, la caméra est également promenée à la main au dessus des cibles.

Lors de nos premiers tests, nous nous sommes aperçus que l'initialisation en utilisant un modèle de perspective faible n'assurait pas nécessairement la convergence de l'itération orthogonale vers la pose réelle de la cible. Il peut lui arriver de converger vers un autre minimum local. Ceci a des répercussions sur le temps de calcul nécessaire à cet algorithme pour converger. Nous avons donc préféré, pour l'initialisation de l'itération orthogonale à la première image, employer notre algorithme spécifique de calcul de la pose. D'autre part, comme nous pouvons le constater dans le tableau 3.3 page 112, suivant la nature du mouvement effectué, l'itération orthogonale donne des résultats disparates. De plus, nous avons fait varier la valeur du critère de convergence (le paramètre $\epsilon$ : plus celui-ci est petit plus le nombre d'itérations pour converger sera important) pour l'itération orthogonale afin d'ajuster le compromis temps de calcul/précision de la pose calculée imposé par ce type d'algorithme.

De ces valeurs, il ressort que c'est la méthode des moindres carrés qui procure les meilleurs

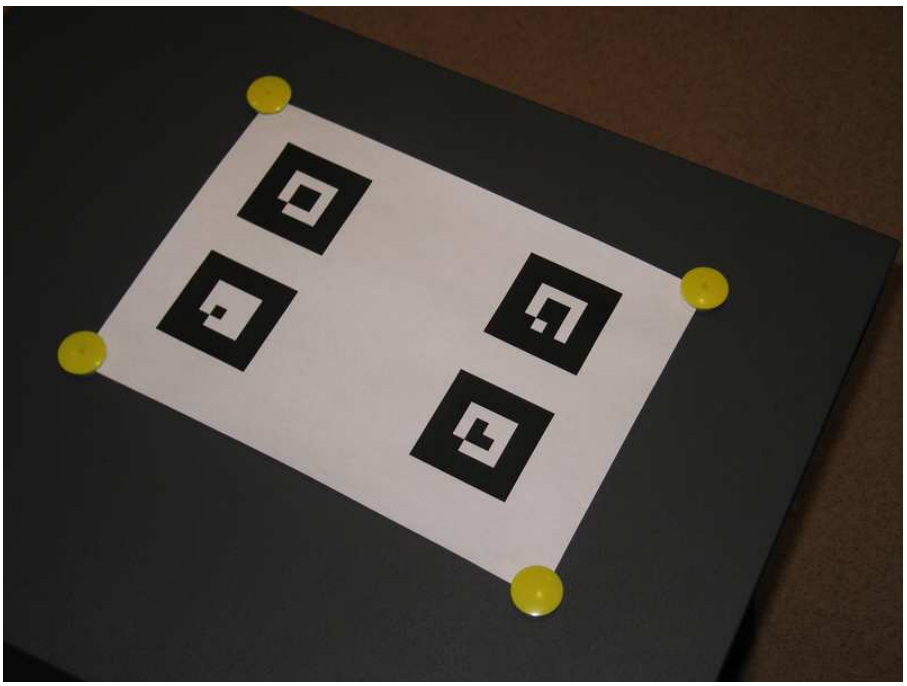


FIG. 3.22 – Jeu de cibles employé pour tester l'erreur de généralisation de chacun des algorithmes

résultats en termes d'erreur de généralisation. Si nous regardons les performances de l'itération orthogonale, il s'avère que la variation du paramètre $\epsilon$ n'influe pas sur l'erreur de généralisation. En revanche, elle augmente le nombre d'itérations nécessaires pour obtenir le même résultat. Sur cette base, pour les mesures suivantes, nous fixerons $\epsilon = 10^{-5}$. Un autre point important, si l'on considère cette étude, est le mauvais comportement de l'itération orthogonale lorsque l'on effectue des mouvements rapides (c'est à dire que l'algorithme de suivi des cibles décroche et que la détection de la cible doit se faire en utilisant toute l'image). En effet, cet algorithme donne des résultats pire que ceux de notre algorithme spécifique. Ceci s'explique par le fait que les mouvements rapides induisent des variations rapides de la pose d'une image sur l'autre. Or, l'itération orthogonale est ici initialisée à l'aide de la pose précédente. Le problème que nous avions lors de nos premiers tests ressurgit alors : l'itération orthogonale converge vers un minimum local, ce qui explique ces différences notables de performances.

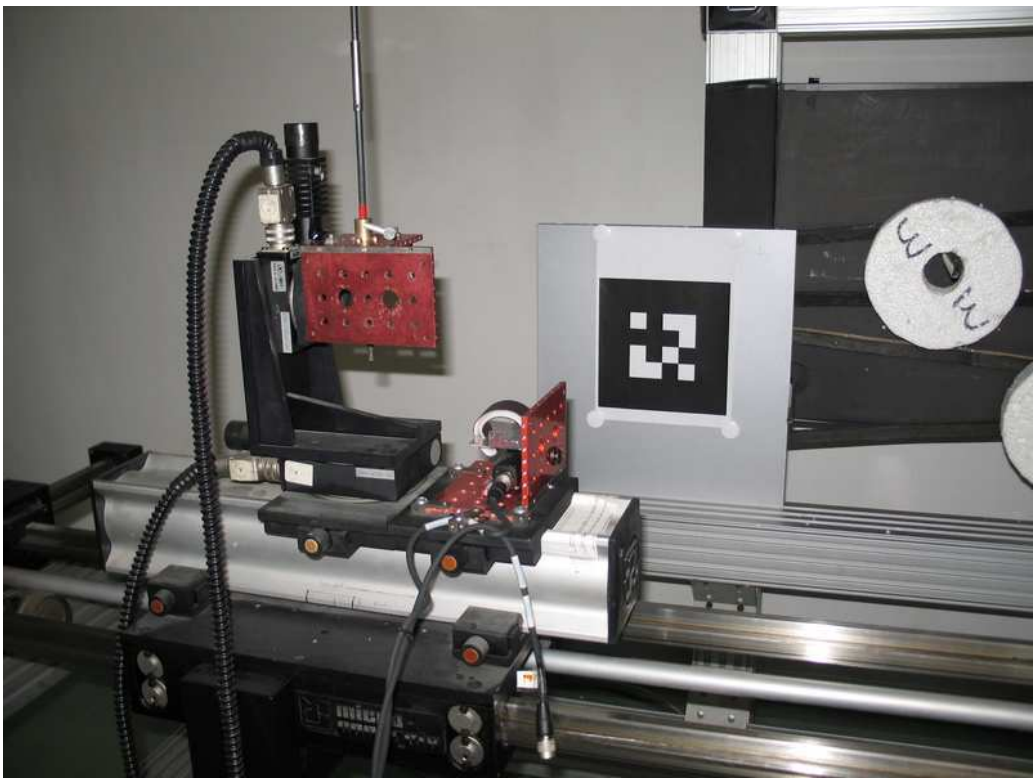
A présent, nous allons observer comment ces algorithmes se comportent pour évaluer les distances.

#### 3.5.2.3 Erreur d'évaluation des distances caméra-cible

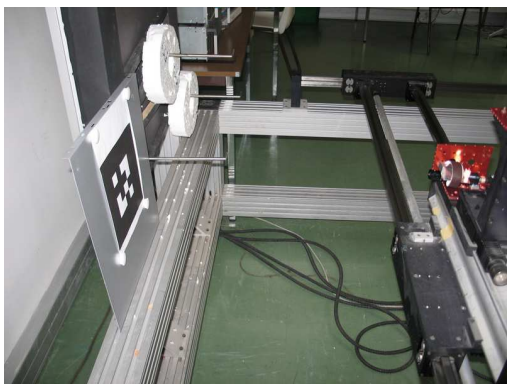
Pour apprécier les erreurs d'évaluation des distances caméra-cible faites par les divers algorithmes, nous avons couplé la configuration 2 avec un banc robotisé de calibration capable de se déplacer sur deux axes de translation. La caméra est placée sur le banc de calibration alors que la cible (mesurant 20 centimètres de côté) est fixée sur une partie immobile du banc. Le montage est représenté figure 3.23 page suivante. Ce banc de calibration nous permet d'être précis au dixième de millimètre. Ainsi, en commandant ce dernier, nous connaissons la position réelle de la caméra par rapport à la cible que nous pouvons donc comparer aux poses estimées par nos trois algorithmes.

Par le biais de ce banc, nous allons échantillonner une partie de l'espace pour laquelle nous relèverons les poses calculées. Nous obtenons ainsi 1491 positions pour lesquelles chaque algorithme a fournit une pose valide correspondante. Nous calculons alors la distance entre le centre optique de la caméra et celui de la cible que nous comparons. Si l'on trace le relevé des distances, nous obtenons la figure 3.24 page 113.

Dans un deuxième temps, nous traçons l'erreur d'estimation de la distance par rapport à la distance



(a)



(b)

(c)

FIG. 3.23 – Banc de test employé pour comparer les évaluations de distance faites par les divers algorithmes de calcul de la pose

(a) mouvement lent, $\epsilon = 10^{-5}$, Nombre de valeurs :181

| **Type d'algorithme** | spécifique | IO | MMC |
|---|---|---|---|
| erreur moyenne ($pixel$) | 22.418 | 16.889 | 12.989 |
| erreur moyenne ($pixel^2$) | 502.611 | 285.257 | 168.715 |
| écart type sur erreur ($pixel^2$) | 530.916 | 141.591 | 72.0089 |
| temps moyen ($\mu s$) | 7.39227 | 704.722 | 4723.99 |
| écart type sur temps ($\mu s$) | 0.571657 | 74.4174 | 861.484 |

(b) mouvement rapide, $\epsilon = 10^{-5}$, Nombre de valeurs :258

| **Type d'algorithme** | spécifique | IO | MMC |
|---|---|---|---|
| erreur moyenne ($pixel$) | 19.314 | 45.851 | 15.952 |
| erreur moyenne ($pixel^2$) | 373.062 | 2102.36 | 254.476 |
| écart type sur erreur ($pixel^2$) | 1018.7 | 2082.18 | 836.652 |
| temps moyen ($\mu s$) | 7.80233 | 707.74 | 4041.64 |
| écart type sur temps ($\mu s$) | 4.46211 | 52.9335 | 1006.79 |

(c) mouvement lent, $\epsilon = 10^{-8}$, Nombre de valeurs :352

| **Type d'algorithme** | spécifique | IO | MMC |
|---|---|---|---|
| erreur moyenne ($pixel$) | 25.066 | 20.352 | 14.658 |
| erreur moyenne ($pixel^2$) | 628.325 | 414.238 | 214.884 |
| écart type sur erreur ($pixel^2$) | 406.936 | 150.725 | 90.9749 |
| temps moyen ($\mu s$) | 7.4517 | 1448.26 | 4617.22 |
| écart type sur temps ($\mu s$) | 1.13215 | 886.581 | 917.024 |

(d) mouvement rapide, $\epsilon = 10^{-8}$, Nombre de valeurs :232

| **Type d'algorithme** | spécifique | IO | MMC |
|---|---|---|---|
| erreur moyenne ($pixel$) | 30.645 | 43.782 | 17.806 |
| erreur moyenne ($pixel^2$) | 939.151 | 1916.89 | 317.066 |
| écart type sur erreur ($pixel^2$) | 1087.16 | 1948.37 | 846.038 |
| temps moyen ($\mu s$) | 7.33621 | 1444.73 | 4269.99 |
| écart type sur temps ($\mu s$) | 0.593702 | 782.532 | 1048.03 |

TAB. 3.3 – Erreur de généralisation suivant différents paramètres pour l'algorithme d'itération orthogonale

réelle pour chacun des trois algorithmes (voir figure 3.26(a) page 114). De ce graphe, nous pouvons tirer les premières conclusions :

- L'algorithme spécifique et l'itération orthogonale présentent sensiblement la même erreur d'estimation des distances,
- Ces derniers ont tendance à surestimer la distance,
- Pour ceux-ci l'erreur semble augmenter linéairement avec la distance,
- La méthode des moindres carrés a tendance à sous-estimer les distances,
- Pour ce dernier, l'erreur semble augmenter quadratiquement avec la distance.

À la première lecture, le comportement de notre algorithme spécifique pourrait sembler contradictoire : nettement en deçà des autres algorithmes pour l'erreur de généralisation, l'appréciation des distances semble être du même ordre que l'itération orthogonale. Comme l'appréciation des distances est liée à la position uniquement, nous pouvons en déduire que notre algorithme spécifique donne des résultats satisfaisants en position mais que l'erreur est plus conséquente pour l'orientation.

Toutefois, comme dans notre expérimentation nous n'avions pas tenu compte de la distortion radiale de l'image dû au système optique de la caméra, nous allons mettre en évidence l'influence

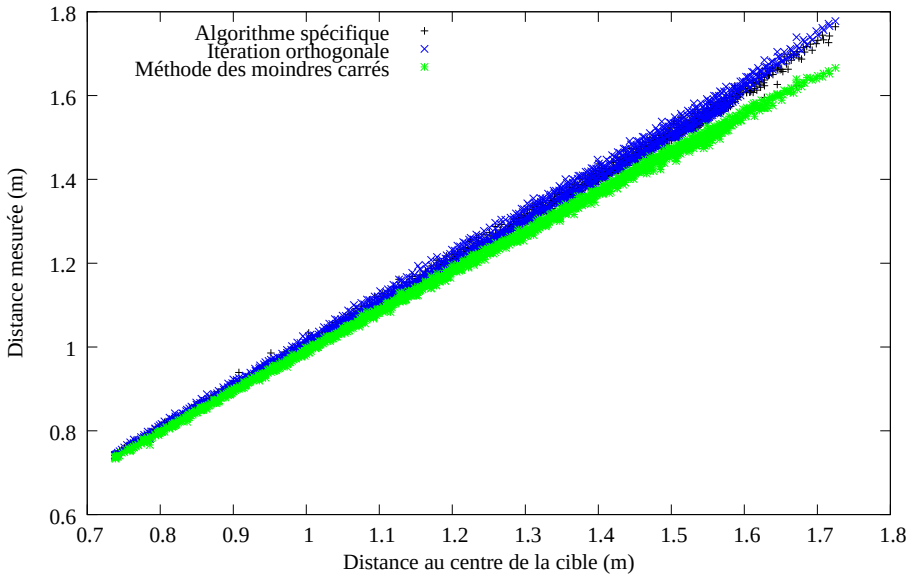



FIG. 3.24 – Évaluation des distances en fonction de la distance réelle (1491 valeurs)

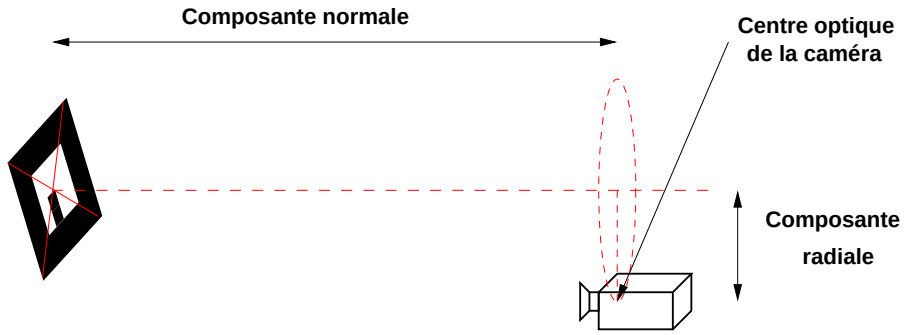
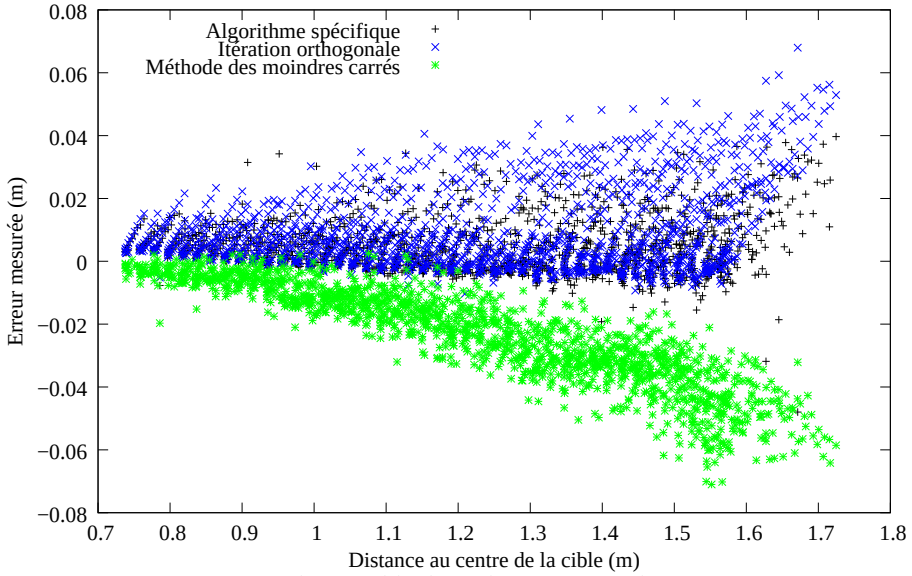



FIG. 3.25 – Décomposition de la distance estimée en composantes radiales et normales

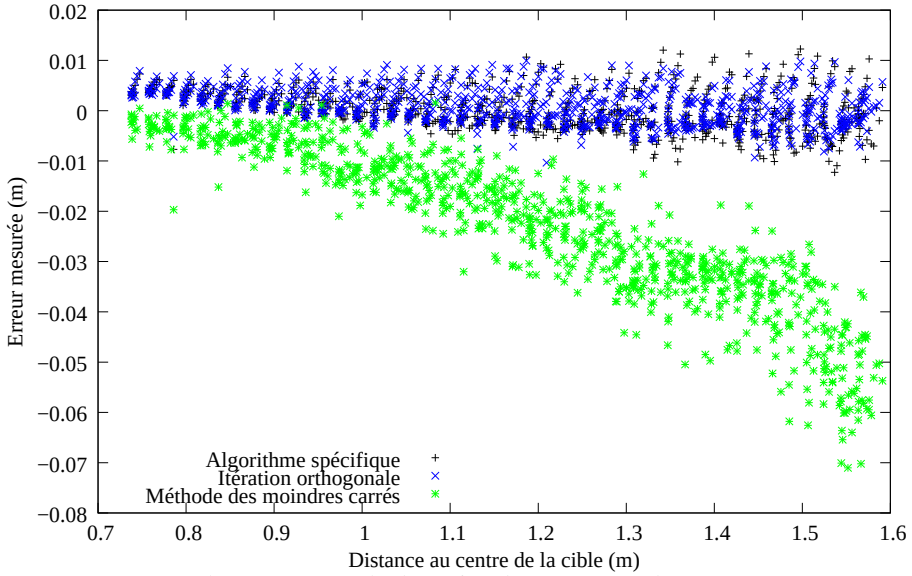
de cette dernière sur l'évaluation des distances. Nous allons décomposer cette erreur d'évaluation en fonction de la composante normale et de la composante radiale représentés dans la figure 3.25. Comme le plan image de la caméra et le plan de la cible sont quasiment coplanaires dans cette expérience, l'angle entre la normale à la cible et la droite passant par le centre optique de la caméra et le centre de la cible est pratiquement le même que celui qu'il y a entre l'axe optique de la caméra et la droite précédente. Pour chacun des algorithmes, nous traçons cette décomposition. Ces diagrammes sont exposés à la figure 3.27 page 115.

Sur ces trois diagrammes, nous observons de manière évidente l'augmentation de l'erreur d'appréciation avec l'augmentation de la composante radiale. Nous avions pensé qu'elle était négligeable sur notre caméra alors que la compensation de la distortion radiale s'avère capitale pour estimer précisément les distances. Nous notons au passage que la méthode des moindres carrés ne semble pas affectée par la distortion radiale. En réalité, la routine programmée dans la librairie OpenCV compense la distortion radiale pour peu que nous l'ayons calculé lors de la calibration de la caméra, opération que nous avions effectuée. Nous pouvons également constater que l'algorithme spécifique et la méthode des moindres carrés sont plus sensibles au bruits de mesure que l'itération orthogonale. Enfin, tant que la composante radiale est négligeable, l' algorithme spécifique et l'itération orthogonale ont une erreur d'appréciation des distances proches de 0 alors que celle de la méthode des moindres carrés croit de manière plus rapide avec la distance.

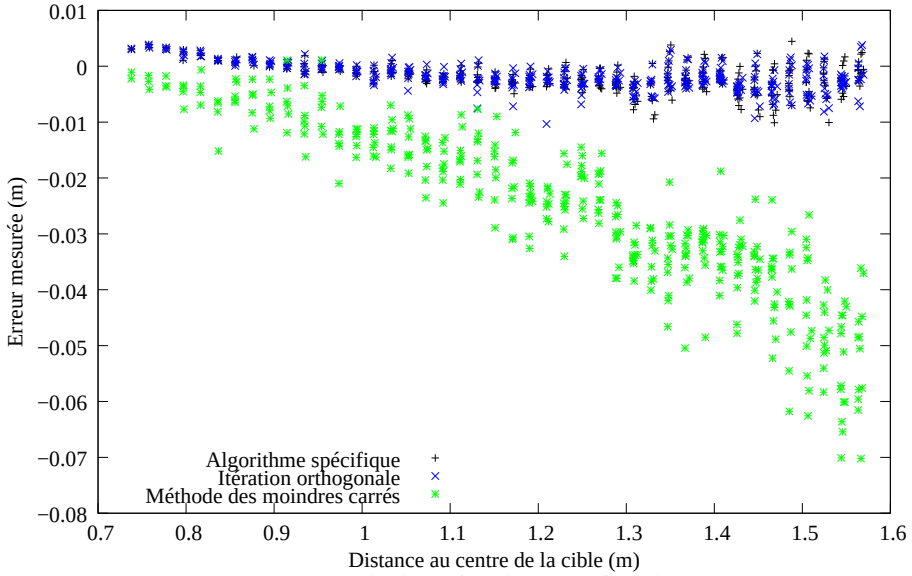
Ainsi, si l'insertion d'objets virtuels s'effectue sur la partie de l'image qui ne contient que la cible qui a servi au calcul de la pose de la caméra (comme c'est le cas des interfaces tangibles




(a) Sur l'ensemble des valeurs (1491 valeurs)

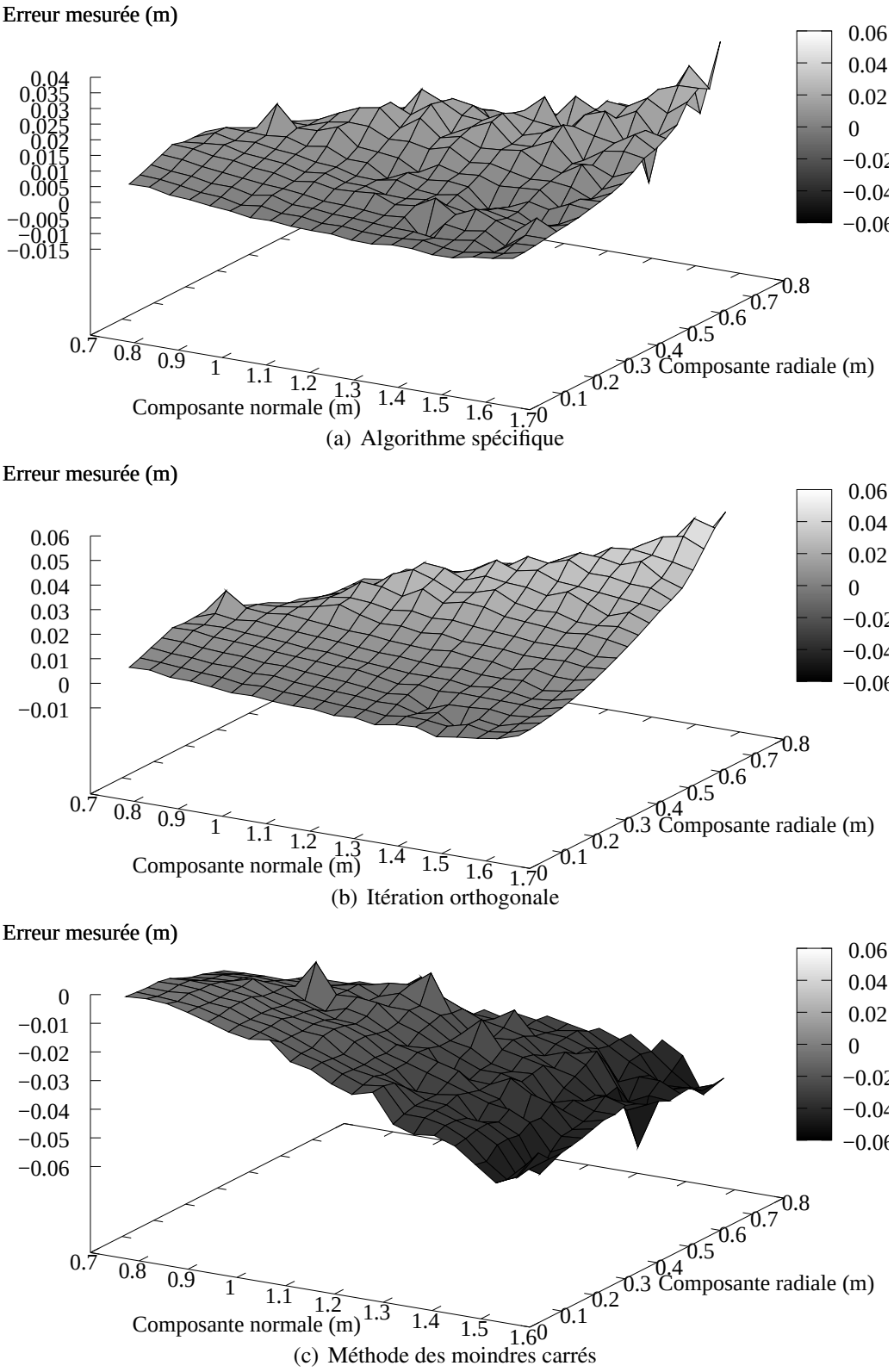



(b) Pour un angle de moins de $22^o$ (873 valeurs)




(c) Pour un angle de moins de $11^o$ (361 valeurs)

FIG. 3.26 – Erreurs d'évaluation des distances en fonction de la distance réelle




(a) Algorithme spécifique

(b) Itération orthogonale

(c) Méthode des moindres carrés

FIG. 3.27 – Erreurs d'évaluation des distances en fonction des composantes normales et radiales

[Kato et al., 2000]) alors l'estimation de la pose peut s'effectuer sans tenir compte des distortions radiales. En effet, l'erreur de reconstruction sera voisine de celle de la cible. En revanche, si le système doit effectuer une localisation précise (le critère d'erreur d'appréciation des distances devient alors prépondérant) ou incruster des objets dans une partie de l'image qui ne contient pas la cible codée, alors le calcul de la pose de la caméra doit prendre en compte les distortions radiales induites par l'optique de la caméra.

Pour évaluer de manière plus précise l'erreur d'appréciation de distance faite par chacun de nos algorithmes, nous nous restreignons aux mesures prises dans un cône de $22^o$ autour de la normale à la cible, puis nous faisons de même autour de $11^o$. Les tracés résultants peuvent être vus à la figure 3.26 page 114. Nous constatons que l'erreur moyenne en fonction de la distance varie très peu entre les deux sous-échantillons de mesure aussi nous prendrons celui qui est le plus peuplé, à savoir les mesures prises dans un cône de $22^o$ autour de la normale à la cible. Nous effectuons une régression linéaire sur les valeurs issues de l'algorithme spécifique et l'itération orthogonale et une régression quadratique sur la méthode des moindres carrés. Nous obtenons les résultats suivants :

```
======================          ========================
Valeur des paramètres           Erreur asymptotique
======================          ========================

Algorithme spécifique           f1(x) = a1*x + b1
======================          ========================
a1              = 0.99426       +/- 0.0005321    (0.05352%)
b1              = 0.0068773     +/- 0.0006487    (9.432%)

Itération orthogonale           f2(x) = a2*x + b2
======================          ========================
a2              = 0.994286      +/- 0.0004808    (0.04835%)
b2              = 0.00763455    +/- 0.0005861    (7.677%)

Méthode des moindres carrés     f3(x) = a3*x*x + b3*x + c3
======================          ========================
a3              = -0.0446704    +/- 0.003791     (8.486%)
b3              = 1.04552       +/- 0.008957     (0.8567%)
c3              = -0.0115026    +/- 0.005135     (44.64%)
```

Pour l'itération orthogonale et l'algorithme spécifique, nous avons en moyenne 6 ‰d'erreur (soit une erreur moyenne de 12 mm à 2 m de distance) alors que pour la méthode des moindres carrés nous obtenons environ 70 mm d'erreur à 2 m (soit 3,5 %).

Sur l'ensemble de ces études, si nous voulons comparer de manière qualitatives les capacités de chacun des algorithmes, nous avons dressé le tableau 3.4 page suivante qui récapitule l'ensemble de nos conclusions. Concernant l'erreur pixellique de généralisation, nous avons mis *+ ou - -* concernant l'itération orthogonale puisque ces performances dépendent de la vitesse du mouvement. À la lecture de ce tableau, il semblerait que l'itération orthogonale soit le meilleur compromis. Toutefois, nous allons voir dans quelle mesure il est possible d'améliorer cet algorithme, et plus spécifiquement sa vitesse de convergence ainsi que sa sensibilité au mouvements rapides.

#### 3.5.2.4 Vers une solution hybride

L'algorithme de l'itération orthogonale, tel que nous l'avons employé, utilise, pour initialiser la pose à partir d'une image donnée, la pose calculée à l'image précédente. Or, notre algorithme spéci-

| **Algorithme** | spéc. | IO | MMC |
|---|---|---|---|
| Vitesse d'exécution | + + | = | - - |
| Erreur pixellique de reprojection | + | + | - |
| Erreur pixellique de généralisation | - | + ou - - | + |
| Erreur d'estimation des distances | + | + | - - |
| Sensibilité aux bruits de mesure | - | + | - |

TAB. 3.4 – Comparaison qualitative des 3 algorithmes de calcul de pose

fique peut être utilisé pour calculer cette pose à l'initialisation.

Nous proposons donc une nouvelle méthode de calcul itératif de la pose dont la pose initiale est calculée par notre algorithme spécifique et les itérations suivantes sont effectuées à l'aide de l'itération orthogonale.

Comme les deux algorithmes ont approximativement le même taux d'erreur dans l'estimation des distances, nous allons étudier l'erreur de généralisation de cet algorithme modifié. Nous adoptons dans un premier temps le même protocole expérimental que précédemment (voir section 3.5.2.2 page 109) pour calculer cette erreur. Nous obtenons les erreurs consignées dans le tableau 3.5.

(a) mouvement lent, $\epsilon = 10^{-5}$, Nombre de valeurs :104

| **Type d'algorithme** | spécifique | IO | MMC |
|---|---|---|---|
| erreur moyenne ($pixel$) | 20.939 | 20.786 | 15.542 |
| erreur moyenne ($pixel^2$) | 438.445 | 432.068 | 241.564 |
| écart type sur erreur ($pixel^2$) | 534.333 | 510.921 | 533.454 |
| temps moyen ($\mu s$) | 7.11371 | 66.4144 | 4110.67 |
| écart type sur temps ($\mu s$) | 1.85254 | 64.2013 | 832.214 |

(b) mouvement rapide, $\epsilon = 10^{-5}$, Nombre de valeurs :77

| **Type d'algorithme** | spécifique | IO | MMC |
|---|---|---|---|
| erreur moyenne ($pixel$) | 28.636 | 27.210 | 24.555 |
| erreur moyenne ($pixel^2$) | 820.072 | 740.405 | 602.97 |
| écart type sur erreur ($pixel^2$) | 2280.69 | 2248.52 | 2260.96 |
| temps moyen ($\mu s$) | 6.97403 | 122.992 | 3794.52 |
| écart type sur temps ($\mu s$) | 0.602463 | 113.896 | 860.237 |

TAB. 3.5 – Nouvelle série de tests avec l'itération orthogonale modifiée

Nous constatons que la nouvelle version de l'itération orthogonale s'est affranchie des problème de dégradation des performances lorsque les mouvements dans l'image sont plus rapides.

Nous avons alors changé notre protocole expérimental pour voir évoluer l'erreur de généralisation en fonction de la distance. Nous avons utilisé deux cibles de 5 cm de côté que nous avons progressivement éloignées l'une de l'autre. La première cible sert au calcul de la pose, la seconde à calculer l'erreur de généralisation. La figure 3.28 page suivante montre les résultats obtenus où nous remarquons que si les performances en termes d'erreur de généralisation sont légèrement plus faibles pour l'itération orthogonale, elles restent néanmoins très proches de celles de la méthode des moindres carrés.

Enfin, nous avons relevé le temps mis par les deux versions de l'itération orthogonale (la version classique avec initialisation de type "weak perspective" et notre version modifiée) ainsi que le nombre d'itération nécessaires pour converger vers la solution (voir résultats table 3.6 page suivante). Nous constatons que nous parvenons à faire converger de manière plus rapide l'itération orthogonale environ 10 fois plus rapidement.

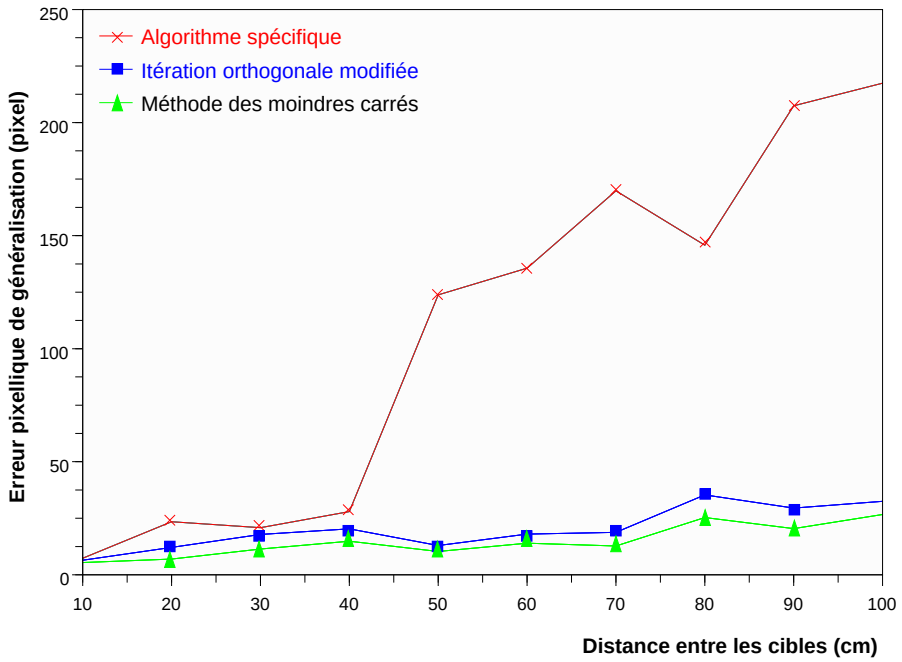



FIG. 3.28 – Courbe représentant l'erreur pixellique de reprojection en fonction de la distance de l'objet par rapport à la cible sur laquelle est calculé la pose

| **Pose d'initialisation** | temps init ($\mu s$) | temps pose ($\mu s$) | nb iterations |
|---|---|---|---|
| Weak perspective | 29.3669 | 425.751 | 34.0296 |
| Algorithme spécifique | 8.30769 | 90.0414 | 2.60947 |

TAB. 3.6 – Améliorations en temps de calcul - ($epsilon = 10^{-5}$, Nombre de valeurs :169)

En résumé, cette méthode présente le meilleur compromis en terme de temps de calcul, d'erreur de généralisation et d'erreur sur l'estimation des distances parmi les algorithmes que nous avons employé pour calculer la pose des cibles carrées.

Nous allons à présent apporter des éléments de comparaison avec un autre système existant : l'ARToolkit.

### 3.5.3 Éléments de comparaison avec l'ARToolkit

Nous n'avons pu comparer directement les performances de l'ARToolkit, notamment en temps d'exécution ou en terme d'erreurs d'appréciation des distances et d'erreur de généralisation car nous ne pouvions pas directement l'intégrer dans notre architecture par composants. Toutefois, nous avons pu tester un exemple complet de l'ARToolkit sur la configuration 2. Cet exemple incluait : l'acquisition, la reconnaissance de la cible, le calcul de la pose et le dessin final des augmentations. Ce dernier permettait de traiter les images à la cadence de 10 à 15 images par seconde. À titre de comparaison, notre système, dans des conditions similaires, effectue les traitements à environ 12-13 images par secondes lorsque le mode de suivi des cibles n'est pas activé, puis ensuite à 17-18 images par secondes avec le suivi. Nous notons que notre système inclut également une étape de désentrelacement de l'image afin d'améliorer la qualité du rendu de l'image final qui n'était pas réalisée avec l'ARToolkit.

Sur le plan de la capacité de l'ARToolkit à détecter de manière précise les coins de la cible, nous pouvons nous référer à l'article de Zhang [Zhang et al., 2002] qui établit une comparaison entre ce système et quelques autres systèmes similaires développés par des laboratoires allemands. Le dé-

tecteur de coin qui a servi de référence est celui qui est implémenté dans la librairie OpenCV que, précisément, nous utilisons. Cette étude a montré que l'ARToolkit avait des performances moindres que les autres systèmes de cible et que OpenCV sur ce chapitre notamment en termes de détection des coins de la cible. Comme la pose est calculée sur les positions de ces coins extraits de l'image, ceci influence grandement la précision du recalage fournit par l'ARToolkit.

Une autre expérience sur la capacité d'évaluation des distances de cette librairie a été menée en 2002 par Malbezin [Malbezin et al., 2002]. Cette étude met en avant le fait que le pourcentage d'erreur augmente avec la distance. Pour une cible de 20 cm de côté, l'erreur atteint entre 6 et 8% à une distance de 93,5 cm. Compte tenu des paramètres utilisés pour leur caméra et des nôtres, nous pouvons comparer avec l'erreur fournie par notre système à environ 1,15m où nous aurions, si nous utilisions l'ARToolkit, une erreur moyenne d'environ 8 cm. Or nous pouvons constater, en regardant les résultats de la figure 3.26 page 114, que l'erreur moyenne se situe bien en deçà de celle de l'ARToolkit puisque inférieure au centimètre, même si elle ne corrige pas les erreurs dues à la distortion radiale induite par l'optique de la caméra.

Enfin, si théoriquement l'ARToolkit permet une variété infinie de code pour ses cibles, le temps d'identification moyen d'une cible est proportionnel au nombre de cibles répertoriées par le système. Ainsi, pour un grand nombre de code utilisés par une application, l'ARToolkit offrira des performances médiocres, une cible détectée étant comparée par corrélation avec l'ensemble des cibles répertoriées par la librairie. De notre côté, si le nombre de cibles codées est limité à 16320 et si le choix du motif interne de la cible n'est pas totalement libre, le temps d'identification sera constant, le code de la cible étant directement lu sur l'image.

## 3.6 Conclusion

Ainsi, dans ce chapitre, nous avons présenté deux algorithmes spécifiques destiné à résoudre le problème de la localisation par la vision. L'un est employé pour l'échantillonnage de zone rectangulaire de l'espace à partir de leur image en deux dimensions, l'autre est utilisé afin de déterminer la pose de la caméra par rapport à des cibles codées.

Nous avons mis en oeuvre ces algorithmes dans notre architecture de programmation par composants, qui a été mise en oeuvre pour automatiser le système de vision. Ainsi, nous avons développé une procédure de calibration semi-automatique de la caméra, puis le système de vision employant les cibles codées. Ce dernier est un système ouvert dans le sens qu'il permet de modifier à volonté le type d'algorithme utilisé pour le calcul de la pose ainsi que le modèle utilisé pour prédire l'emplacement d'une cible sur une image donnée à partir de la position précédente.

Enfin, nous avons évalué les performances globales de notre système de localisation tout en les comparant avec des algorithmes connus. Cette étude a porté sur des critères de performances qui sont : le temps d'exécution, l'erreur de reconstruction, l'erreur de généralisation, l'erreur d'estimation des distances. Celle-ci nous a permis de proposer en définitive un système hybride de calcul de la pose reposant sur l'algorithme spécifique que nous avions introduit et l'itération orthogonale. De plus, elle met en évidence l'importance critique de corriger la distortion radiale due à l'optique de la caméra pour améliorer les performances du système. Enfin, nous avons fourni des éléments de comparaison entre notre système et celui qui reste le plus populaire au sein de la communauté de réalité augmentée à savoir l'ARToolkit. Ceci nous permet d'affirmer que notre système apporte des solutions intéressantes pour effectuer de la localisation par la vision à l'aide de cibles codées.

Toutefois, notre système de réalité augmentée ne serait pas complet sans un véritable système de gestion des augmentations. Ce dernier va faire l'objet de la partie suivante.

# Chapitre 4

# Système de gestion des augmentations

Dans ce chapitre, nous présenterons les composants spécifiques que nous avons développé afin de gérer les augmentations de notre système de réalité augmentée. Ces développements s'inscrivent dans le cadre du projet RNTL (Réseau National des Technologies Logicielles) AMRA (Assistance à la Maintenance en Réalité Augmentée) que nous présenterons brièvement.

Parmi les développements spécifiques du LSC sur ce projet figurent la modélisation d'une procédure de maintenance classique. Pour ce faire, des exemples de procédure de maintenance ont été étudiés afin de dégager les structures inhérentes à ces dernières que nous avons modélisées.

Ceci nous a permis de nous affranchir des problèmes liés à la complexité et à la variété des tâches de maintenance. Les procédures sont mises sous un format électronique qui exploite les technologies XML (pour eXtensible Markup Language). Leur contenu est lié à des documents multimédia (sons, images, vidéos, modèles 3D, etc ) qui sont ensuite exploités par le système de gestion des augmentations.

Les modèles 3D de leur côté, sont instrumentés par des animations décrivant les étapes de la procédure de maintenance.

L'ensemble des composants développés forme ainsi un moteur multimédia pour la réalité augmentée.

## 4.1 Le projet AMRA

Le projet AMRA est un projet RNTL (Réseau National des Technologies Logicielles) [RNTL, url] placé sous la tutelle du ministère de la Recherche. Ce dernier a commencé en 2002 et s'est achevé en mai 2004. Le projet était mené par le consortium de partenaires suivant :

- Alstom Transport, coordinateur du projet et partenaire industriel apportant la problématique de travail,
- Le Commissariat à l'Énergie Atomique (CEA), plus particulièrement le Laboratoire d'Intégration des Systèmes et des Technologies (LIST), qui a développé un système de vision pour le projet, et qui a réalisé l'intégration finale du prototype,
- Le Laboratoire Systèmes Complexes (LSC) qui est notre laboratoire,
- Acti-CM, une start-up issue du CEA spécialisée dans la métrologie.

Le but était d'implémenter un système de Réalité Augmentée à usage mobile pour une utilisation en milieu industriel, et plus spécifiquement dans le domaine de la maintenance industrielle. Ce projet a poursuivi plusieurs objectifs :

- Fournir une aide contextuelle à des mainteneurs inexpérimentés, leur permettant d'être formés sur site,
- Apporter aux agents de maintenance une assistance permettant d'accéder sur leur poste de travail à des informations pertinentes (documentation de maintenance, modes opératoires, films

de montage),
– Augmenter la disponibilité de l'information sur le lieu de maintenance en utilisant des techniques de Réalité Augmentée.

Le prototype AMRA est un système de réalité augmentée en vision indirecte constitué d'un tablette-PC (un ordinateur portable "allégé" pourvu d'un écran tactile - voir figure 4.2 page suivante) pour la visualisation des informations, qui agit comme une fenêtre augmentée sur le monde réel, grâce à la caméra embarquée sur ce dernier. Ce type de système aborde plusieurs problématiques : celle de l'informatique nomade (en anglais "mobile computing"), celle du recalage temps réel des entités virtuelles sur les images du monde réel, et enfin celle du développement d'une aide graphique contextuelle adaptée.

Sur cette base, un système de réalité augmentée en vision indirecte a été développé dans lequel quelques-uns de nos composants ont été incorporés à l'aide, non pas de notre architecture incluant des librairies dynamique, mais en utilisant de la "glu logique". Ces composants sont un moteur multimédia pour la réalité augmentée, un composant de mixage des images réelles avec des entités virtuelles, ainsi qu'un composant de chargement et d'interprétation des procédures de maintenance structurées stockées sous forme de fichiers XML. Nous verrons comment ces derniers sont employés dans l'architecture générale du prototype que nous allons présenter.

## 4.1.1 Architecture générale du prototype

### 4.1.1.1 Le matériel employé

Le premier choix architectural concerne le type de matériel et de périphériques utilisés pour notre application de réalité augmentée. Au départ, nous nous étions orientés vers des lunettes semi-transparentes, en vue d'augmenter la réalité en vision directe. Toutefois, ce choix a été abandonné pour plusieurs raisons :
– Dans le cas de lunettes semi-transparentes, deux options s'offrent à nous. Nous pouvons choisir des systèmes au coût faible mais au champ de vision très restreint (moins de $30^o$ en diagonale dans la plupart des cas) ou nous pouvons employer des systèmes plus coûteux qui sont malheureusement trop encombrants pour le type d'application envisagée,
– Les positions des opérateurs de maintenance (voir figure 4.1 page ci-contre) ne leur permettent parfois pas d'utiliser et de porter confortablement les lunettes semi-transparentes et les accessoires qui lui sont associés (batterie, unité de calcul portable, etc.) ,
– Finalement, une étude de terrain nous a révélé que les mainteneurs étaient réfractaires au port de lunettes semi-transparentes.

Paradoxalement, les mainteneurs se sont trouvés prêt à accepter un système de type tablette-PC qui nécessite leur deux mains, mais qu'ils peuvent poser à un endroit de leur atelier une fois qu'ils ont accédé aux renseignements recherchés. Nous avons donc employé un tablette-PC qui tient le rôle de fenêtre augmentée sur l'espace de travail. Ce dernier est pourvu d'une caméra (modèle Basler A100) qui fait également office de capteur de position et d'orientation grâce à l'instrumentation du site avec des cibles codées qui sont différentes du système que nous avons employé pour nos propres besoins (voir chapitre 3 page 73 à ce sujet). Le système de vision développé par le CEA est également un système de recalage par cible codée.

À l'époque où le projet a commencé, les tablette-PC n'étaient pas pourvus d'une puissance suffisante pour effectuer du traitement d'images et dans le même temps mixer les images réelles avec des modèles virtuels. C'est pourquoi, une partie des traitements a été délocalisée sur une machine distante qui prend en charge le traitement des images afin de trouver la pose de la caméra. Ceci implique l'utilisation d'une architecture distribuée que nous allons exposer à présent.

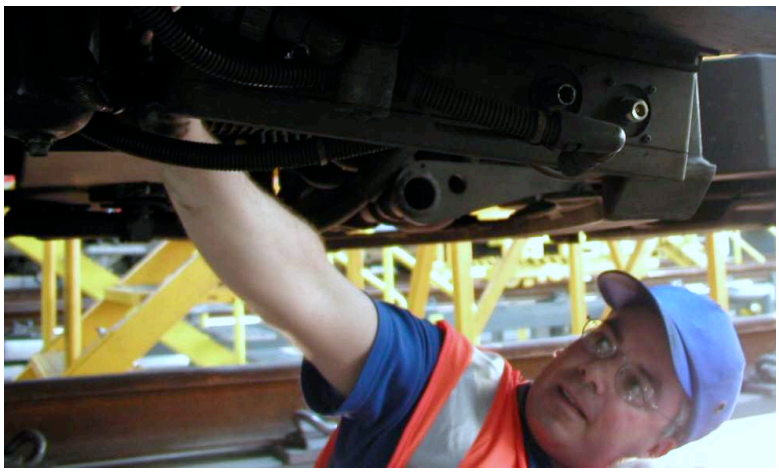


FIG. 4.1 – Position de l'opérateur pendant une des phases de maintenance

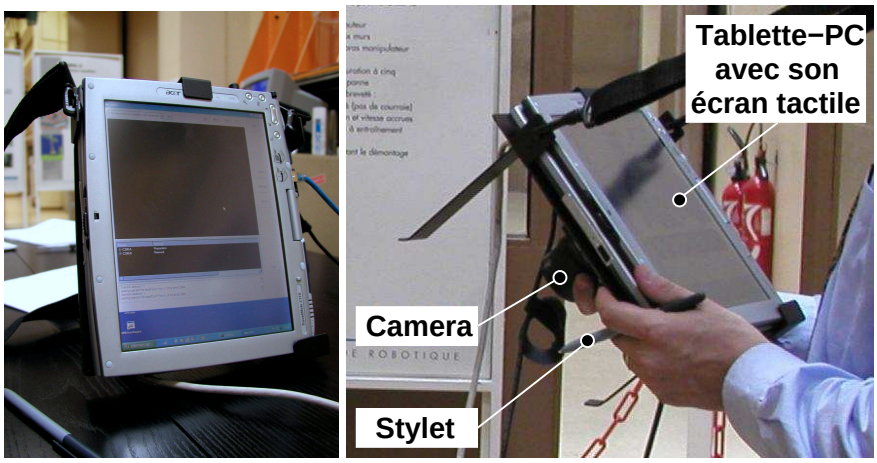



FIG. 4.2 – Tablette-PC et ce dernier équipé de sa caméra

#### 4.1.1.2 Architecture logicielle générale

Nous avons employé une architecture distribuée telle que représentée à la figure 4.3 page suivante. Elle nous permet de partager les ressources et les flux de données entre l'unité mobile (le tablette-PC) et l'ordinateur distant qui fait le lien avec la base de données de maintenance et les algorithmes lourds de traitement d'image pour le recalage. En effet, la base de données de maintenance est centralisée et partagée entre tous les mainteneurs.

Au niveau des flux de données, la caméra est attachée au tablette-PC et elle est connectée par un câble IEEE1394 (firewire - qui sera remplacé plus tard par une liaison radio haute fréquence avec le PC distant pour assurer la mobilité complète du tablette-PC). Enfin, le tablette-PC et l'ordinateur distant communiquent par liaison WiFi en utilisant le protocole IEEE802.11g. Le flux vidéo qui transite entre les deux machines est compressé en raison de la bande passante limitée de ce type de réseau.

#### 4.1.1.3 La tâche dévolue au LSC

Le LSC, dans cette architecture, a développé le moteur multimédia pour la réalité augmentée. Il nous a également fallu créer un nouveau format de document électronique pour les procédures de maintenance ainsi que des modèles 3D adaptés à l'usage qui peut en être fait en réalité augmentée.

Les composants développés pour ce projet par le LSC devaient s'acquitter des tâches suivantes qui ont été définies lors de l'analyse du projet :

– afficher le flux vidéo représentant l'environnement de travail,

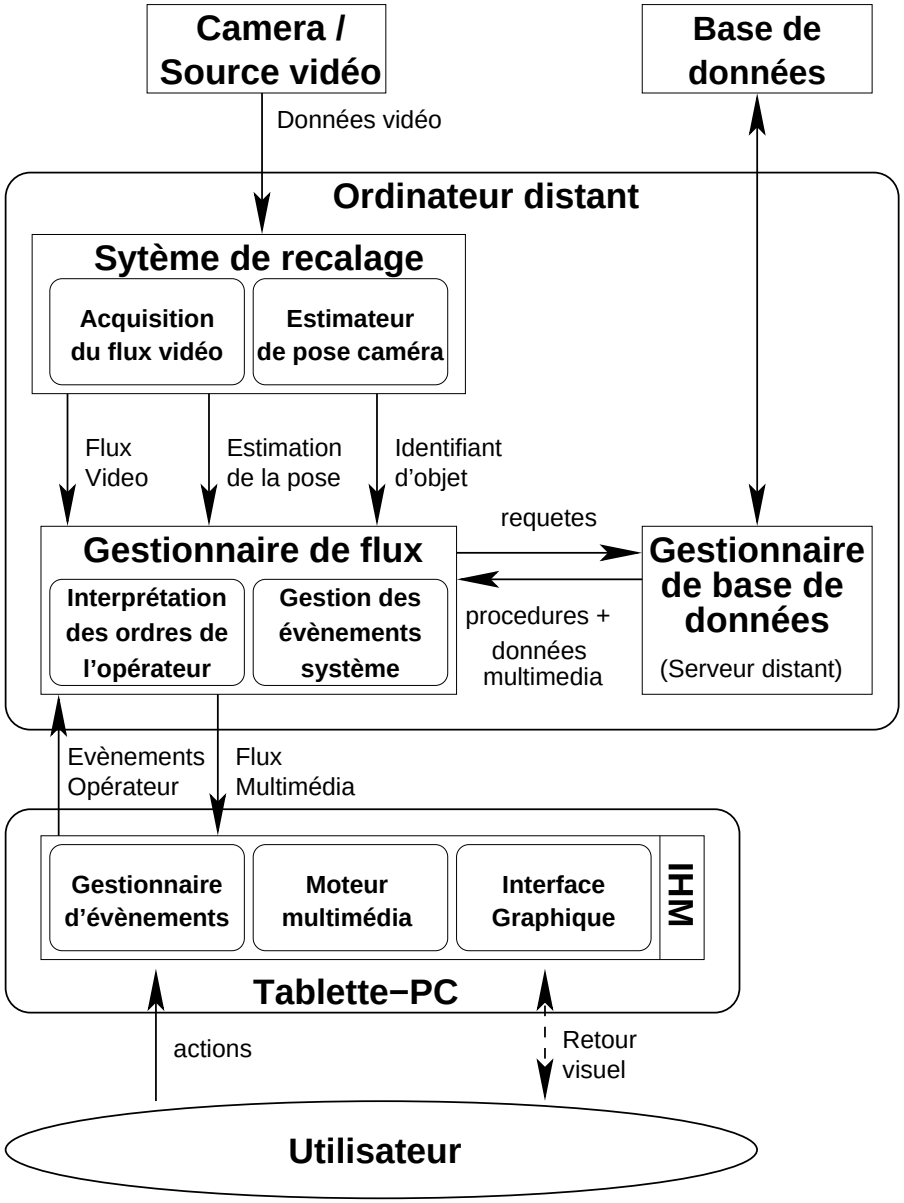



FIG. 4.3 – Vue globale de l'architecture logicielle du prototype

- afficher un modèle 3D virtuel du support de maintenance en surimpression sur le flux vidéo ou bien seul avec des aides de navigations adaptées,
- afficher le modèle virtuel recalé sur le flux vidéo ou non,
- fournir des mécanismes d'interfaçage avec une procédure de maintenance sous forme numérique,
- désigner sur la maquette numérique un élément du support de maintenance.

L'une des tâches globales du prototype AMRA est de fournir une aide contextuelle pendant l'exécution d'une procédure de maintenance. Pour ce faire, une description numérique d'une procédure de maintenance a été conçue au travers d'un document structuré (utilisant le formalisme XML). Un tel document XML permet de stocker, outre le texte des différentes étapes d'une procédure, les prérequis, l'outillage nécessaire, ainsi que des liens vers les différents médias attachés à cette étape d'une procédure (images, sons, vidéo, modèles 3D animés).

A présent, nous allons voir comment nous avons transcrit les procédures de maintenance sous un format numérique.

## 4.2 Traduction des procédures de maintenance sous forme numérique

La numérisation des procédures de maintenance répond au besoin de fournir un accès facilité aux documentations techniques concernant le dispositif à maintenir. En effet, cette documentation est traditionnellement imprimée dans des manuels qui tendent à devenir encombrants et peu faciles à consulter comme cela a été rappelé par Ventura [Ventura, 1988].

Un avantage supplémentaire à l'utilisation des procédures de maintenance numériques est qu'elles peuvent nous permettre d'inclure des nouveaux médias qui ne pouvaient pas être exploités dans le cadre des manuels traditionnels. Ainsi, nous pouvons ajouter des vidéos, des commentaires sonores ou même des modèles en trois dimensions avec lesquels il est possible d'interagir. De plus, dans le cadre de notre projet, la réalité augmentée doit également servir de support en tant que nouveau média à ces procédures numériques.

Nous avons donc analysé dans un premier temps les procédures de maintenance qui nous ont été fournies par Alstom afin d'en dégager les caractéristiques communes.

### 4.2.1 Analyse d'une procédure de maintenance

Une procédure de maintenance est caractérisée par un titre se référant à la tâche à effectuer et comporte une référence non ambigüe utilisée pour le classement des procédures.

Les procédures sont composées des éléments suivants :
- des consignes de sécurité ou de références à des consignes de sécurité,
- des outils à employer,
- des consommables et des pièces détachées utilisées,
- des directives de maintenance proprement dites.

Les directives de maintenance sont organisées hiérarchiquement de manière arborescente. Chaque directive peut être détaillée par une liste de sous-directives, elles-même redétaillées. Si une étape est détaillée par des sous-étapes, alors elle contient un titre générique résumant les sous-étapes. Chaque étape peut contenir également des messages d'avertissement ou des consignes de sécurité qui s'appliquent à chacune de ses sous-étapes également.

Les procédures contiennent également des photographies qui aident les mainteneurs à visualiser la pièce ou les éléments qui sont mentionnés dans la procédure. Le texte de la procédure fait référence aux pièces désignées sur l'image par des numéros. Un certain nombre d'étapes peuvent faire référence à la même image.

Toutefois, dans le cadre d'une application de réalité augmentée, il est possible également "d'augmenter" la procédure papier. Le passage du document à sa contrepartie numérique ouvre des possibilités dans le cadre de l'exploitation de certains médias supplémentaires qui ne pouvaient pas être employés dans le cadre classique d'une procédure imprimée dans un manuel technique.

#### 4.2.1.1 Apports du passage à une procédure de maintenance numérique

Au départ, les seuls médias disponibles sur une procédure classique sont du texte ainsi que des graphiques en deux dimensions ou des photographies.

L'intégration d'une version numérique de la procédure de maintenance dans un système informatique permet de compléter la documentation en fournissant d'autres types d'informations bien connues dans les applications de type multi-médias, à savoir :
- de l'hypertexte. Comparable au texte affiché dans un navigateur, celui-ci comporte des hyperliens et des références croisées vers d'autre médias et procédures.

– des images ou des photographies “augmentées”. Le passage de la souris ou de quelque autre dispositif de pointage déclenche un évènement dans l'application qui provoque l'affichage de médias supplémentaires ou modifie l'image en question.
– des sons ou commentaires audio.
– des animations en deux dimensions ou un flux vidéo combiné ou non avec du son.
– des modèles en trois dimensions animés ou non. Ceci permet d'aider à la visualisation et à connaître la situation relative d'une pièce par rapport à une autre sur le dispositif de maintenance.

Si nous effectuons une transition des manuels techniques vers des procédures de maintenance numérique, il est également possible de prévoir l'utilisation de ces dernières en réalité augmentée. Nous allons voir dans quelle mesure ceci est possible.

#### 4.2.1.2 Apports d'une application en réalité augmentée exploitant une procédure de maintenance numérique

L'apport de la réalité augmentée consiste à mettre en adéquation le monde réel avec les modèles virtuels présentés à l'opérateur. Une fois la position du mainteneur déterminée par le système informatique, l'opérateur peut faire interagir le monde réel et les modèles virtuels dans le but de faciliter et d'améliorer son apprentissage de la procédure de maintenance.

Dans ce cadre, une fois l'opérateur situé par rapport au monde réel, nous pouvons superposer ou non les modèles CAO (Conception Assistée par Ordinateur) des pièces sur la vue du monde réel obtenue par la caméra.

L'analyse des procédures faite, nous pouvons alors dégager une structuration sous forme de balises XML d'une procédure de maintenance, au travers de la définition d'une DTD (Document Type Definition - le document qui décrit la hiérarchie de balises XML d'un fichier) pour notre document.

### 4.2.2 Modèle XML d'une procédure de maintenance

Par rapport à la structure d'une procédure de maintenance, le choix d'une transposition numérique sous format XML nous a semblé adapté. En effet, une procédure de maintenance est structurée de manière arborescente, ce qui est précisément le cas des balises qui sont employées dans le format XML. De plus, l'idée n'est pas nouvelle en elle-même puisque le département de la défense américain a spécifié une DTD pour les manuels techniques qui sont appelés IETM [Ietm, 2000] (Interactive Electronic Technical Manual). Cette spécification n'est toutefois pas adaptée à notre cas car elle n'est pas prévue pour être utilisée dans le cadre de la réalité augmentée. C'est pourquoi, nous avons développé notre propre version de manuel électronique adapté à la RA dont nous allons détailler le modèle.

Nous avons décomposé notre document XML en trois grandes parties qui correspondent à l'analyse des procédures de maintenance.

– `pre-requirements` qui décrit ce dont les opérateurs vont avoir besoin pour effectuer l'opération de maintenance prévue,
– `available-media` qui dresse la liste des médias qui vont être employés dans la procédure de maintenance en plus du texte de la procédure en elle-même,
– `phases` qui recense l'ensemble des étapes de la procédure de maintenance. Pour chacune de ces étapes, nous trouvons des notes explicatives, des recommandations de sécurité et la liste des médias en rapport avec l'étape.

La DTD décrivant l'arborescence des balises XML du document se trouve dans l'annexe A.5 page 182. La figure 4.4 représente cette organisation hiérarchique dont nous allons préciser le rôle des différents constituants.

Nous allons maintenant présenter le système de balises implémenté dans la DTD.

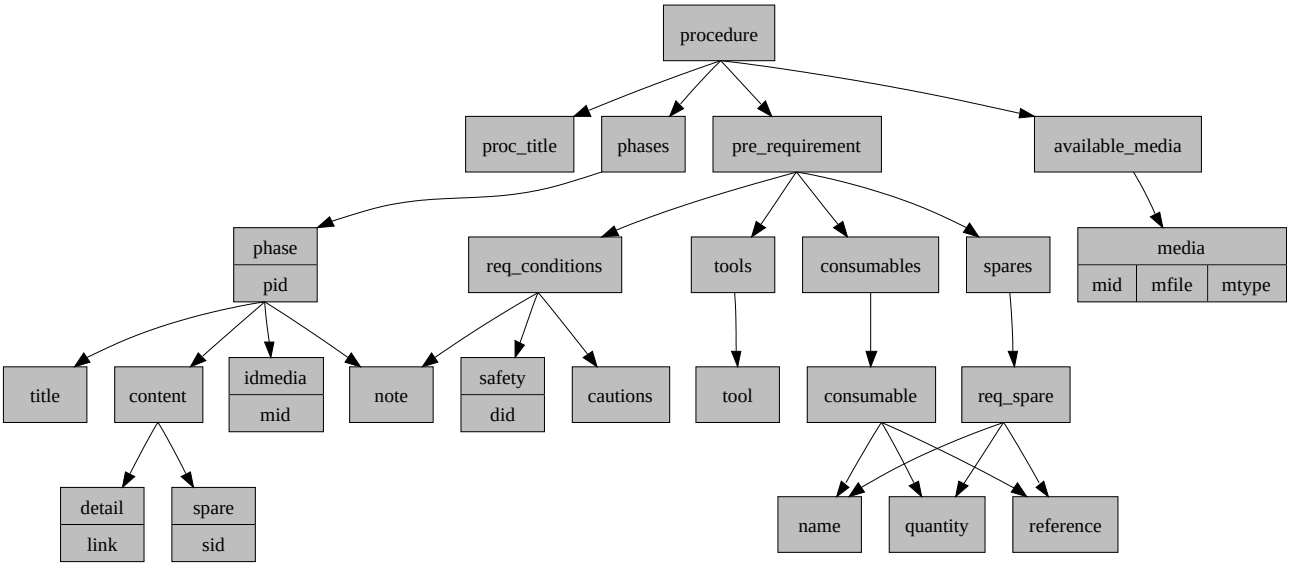



FIG. 4.4 – Diagramme d'agencement des balises dans notre proposition de structuration d'une procédure de maintenance

#### 4.2.2.1 Jeu de balises

Nous allons donc nommer toutes les balises que nous avons défini dans une procédure de maintenance formatée en XML telle que nous la proposons. Le tableau 4.1 page suivante récapitule l'ensemble de ces balises. Certaines de ces balises contiennent des attributs. C'est notamment le cas pour les balises `media, safety, idmedia, spare` et `detail` qui sont exposées dans le tableau 4.2 page 129. Beaucoup sont des identifiants et donc se doivent d'être unique dans le document pour les distinguer les uns des autres.

Le type de média, quant à lui, est un type MIME[MIME, url] (Multipurpose Internet Mail Extensions) dont les spécifications peuvent être trouvées dans les RFC[RFC, url] (Requests For Comments - documents qui font office de norme pour les protocoles utilisés sur Internet) 2045 à 2049 et dont la liste complète est enregistrée auprès de l'IANA (Internet Assigned Numbers Authority - L'organisme chargé, entre autres, des attributions des adresses Internet dans le monde). Ainsi, le type MIME associé à un fichier VRML (voir section 4.3.2 page 130 pour une description de ce format) sera `x-world/x-vrml` et celui associé à un fichier XML sera `text/xml`.

La procédure de maintenance sera lue par un composant qui va gérer cette dernière ainsi que les médias qui lui sont associés pour illustrer les étapes que le mainteneur doit effectuer pour la procédure en cours. Parmi ces types de média se trouve celui qui concerne les modèles 3D que nous allons recaler sur le dispositif de maintenance physique. Ce dernier fera l'objet d'une étude détaillée dans la section suivante avant que nous présentions le moteur multimédia qui implémente cette structuration.

## 4.3 Augmentations 3D animées

Dans notre cas, les modèles 3D sont employés en tant qu'assistance graphique pour illustrer une étape du scénario de maintenance. Il nous faudra donc les concevoir en conséquence.

Les modèles 3D ( tel que celui représenté à la figure 4.5(a) page 129 ) que nous employons sont les contreparties virtuelles du dispositif physique à maintenir 4.5(b) page 129. Nous avons choisi comme format d'importation le standard VRML97[X3D, url] puisqu'il s'agit d'un format d'objets 3D largement répandu que la plupart des modeleurs et des outils de CAO (Conception Assistée par Ordinateur) savent importer et exporter ce format qui a actuellement tendance à être remplacé par son

| Balise | Contenu entre balise ouvrante et balise fermante |
|---|---|
| procedure | corps du document de la procédure de maintenance |
| proctitle | nom de la procédure |
| pre_requirement | pré-requis qui doivent être portés à la connaissance du mainteneur avant que ce dernier commence le travail de maintenance : outils à employer, pièces détachées, précautions à prendre, etc ...) |
| req_conditions | toutes les notes, précautions à prendre et avertissement destinés au mainteneur pour assurer sa protection personnelle lors de la tâche de maintenance |
| safety | précautions à prendre pour la sécurité personnelle du mainteneur |
| caution | avertissements destinés au mainteneur |
| note | note à caractère informatif |
| tools | liste des outils |
| tool | désignation d'un outil |
| spares | liste des pièces détachées |
| req_spare | désignation d'une pièce détachée (boulon, écrou, etc ...) |
| consumables | liste des consommables |
| consumable | désignation d'un consommable (huile, téflon, etc ...) |
| name | nom de l'élément (pièce détachée ou consommable) |
| reference | identifiant de l'élément |
| quantity | quantité requise de l'élément |
| phases | liste des actions à effectuer |
| phase | action à effectuer (peut elle-même contenir plusieurs phases) |
| title | libellé de l'action |
| content | contenu de la phase d'action |
| spare | référence à une pièce du support de maintenance |
| available_media | liste des médias à pré-charger pour la procédure en cours |
| idmedia | référence à un média pré-chargé |
| media | description d'un média associé à la procédure de maintenance |
| detail | lien vers une autre procédure de maintenance |

TAB. 4.1 – Jeu d'attributs de balises employées dans la procédure de maintenance

successeur et équivalent qui se conforme strictement au standard XML : X3D[X3D, url]. La migration de VRML à X3D ne devrait donc pas poser de problèmes à l'avenir dans la mesure où X3D maintient la même structure de graphe de scène ainsi que les mêmes noeuds.

VRML et son successeur présentent également l'avantage de permettre de rajouter des comportements interactifs aux modèles 3D. Ceci permet de les animer en vue d'illustrer des étapes de maintenance telles que l'assemblage ou le désassemblage. Enfin, ce format permet de créer soi-même ses propres noeuds (appelés PROTO) si l'on a besoin d'un comportement personnalisé.

Nos modèles furent directement importés à partir de ceux utilisés pendant la phase de conception du dispositif à maintenir. Toutefois, la puissance de calcul embarquée par le tablette-PC étant faible, nous avons du réduire la complexité des modèles CAO employés. Ceci a été fait de manière partiellement automatique et dans le même temps partiellement manuelle jusqu'à ce que nous obtenions un bon compromis entre la complexité graphique et la vitesse d'exécution de l'application.

Ces modèles ont connus un autre changement important car nous devions illustrer les différentes étapes des procédures à l'aide d'animations des modèles en trois dimensions. Ceci nous a amené à traduire le texte des procédures de maintenance en animations graphiques. Nous allons à présent décrire cette étape.

| Balise | Attribut | Descriptif de l'attribut | Utilisation |
|---|---|---|---|
| media | mid | identifiant de média | obligatoire |
| | file | fichier ou URL contenant le média | obligatoire |
| | mtype | type de média | obligatoire |
| detail | link | lien vers un identifiant de procédure | obligatoire |
| safety | did | identifiant de consigne de sécurité | facultatif |
| phase | pid | identifiant de phase | obligatoire |
| idmedia | mid | identifiant de média | obligatoire |
| spare | sid | identifiant de pièce | obligatoire |

TAB. 4.2 – Liste des attributs employés par le jeu de balises XML d'une procédure de maintenance

(a) Modèle 3D d'un transformateur électrique

(b) Dispositif réel sur lequel vont être ajoutées les augmentations

FIG. 4.5 – Modèle 3D figurant un transformateur électrique et sa contrepartie réelle

### 4.3.1 Des procédures aux animations

Afin de compléter notre modèle 3D avec des animations, nous avons étudié plusieurs procédures de maintenance afin de dégager les actions couramment utilisées. Puis, nous les avons traduites sous forme d'animations réutilisables au travers des PROTOs VRML (voir § 4.3.2).

Afin de montrer ce qui est réellement fait, nous allons nous appuyer sur un exemple de de procédure de maintenance employée dans le cadre du projet AMRA. Cette opération s'effectue sur un transformateur électrique situé sous le plancher des wagons de passagers :

---

B1 ···
B2 Disconnect the electrical connections on the 'A' side as follows :
10 Remove the two self-lock nuts (14), bolts (1) and four washers (2) and disconnect the cable (13) from the LV mount on the transformer. Discard the self-lock nuts (14) and bolts (1).
20 Remove the self-lock nut (4) and the washer (5) and disconnect the cable (7) from the stud (3). Discard the self- lock nut (4).
B3 ···

---

Ce qui se traduit en français par :

---

B1 ···
B2 Déconnecter les connections électriques sur côté 'A' comme suit :
10 Retirer les deux écrous auto-bloquant (14), les boulons (1) et les quatre rondelles (2) et déconnecter le câble (13) de la monture LV sur le transformateur. Jeter les écrous auto-bloquant (14) et les boulons (1).
20 Retirer l'écrou auto-bloquant (4) et la rondelle (5) et déconnecter le câble (7) du plot (13).

Jeter l'écrou auto-bloquant (4).
B3 ...

---

Comme nous pouvons le voir dans cet extrait de procédure, ici l'action principale consiste à dévisser des boulons et des presses-étoupes et à les retirer des plaques, câbles et connecteurs maintenus par ces écrous. Le principe est de créer un modèle générique d'animation pour ces actions de vissage/dévissage puisque par essence, ce sont des opérations similaires.

Toutefois, avant d'exposer la manière dont nous avons intégré ces animations dans les prototypes VRML qui permettent d'illustrer les étapes de la procédure de maintenance, il convient d'exposer les principales caractéristiques du format VRML/X3D afin de voir comment nous allons exploiter ce dernier.

### 4.3.2 Brève introduction au format VRML/X3D

Le format VRML (et par extension son successeur X3D) décrit un graphe de scène. Ce graphe est constitué de divers noeuds ( *nodes*) qui forment une structure arborescente.

A chaque noeud est associé une fonctionnalité générale. Certains vont par exemple décrire les propriétés de la caméra, d'autres des lumières de la scène, certains vont donner la géométrie des objets. D'autres vont regrouper plusieurs noeuds. C'est le cas par exemple des noeuds de type "*Group*" ou encore "*Transform*" (ce dernier renferme une transformation géométrique qui sera appliqué à l'ensemble du sous-graphe de scène qui lui est rattaché). D'autres noeuds servent à calculer des interpolations ("*interpolators*") en temps, en translation, en rotation, etc ... Enfin, certains sont déclenché de manière interactive dès qu'un évènement se produit ("*sensors*"). Cet évènement peut-être un clic de souris, le passage à proximité du noeud lors de l'exploration de la scène, etc ...

Chaque noeud renferme des propriétés : ce sont les champs ("*fields*"). Dans le cas d'une transformation, nous trouverons ainsi des propriétés telles que la rotation, la translation et le facteur d'échelle part exemple. Les noeuds peuvent recevoir ou émettre des évènements ("*events*") susceptibles de modifier les champs internes au noeud. Ces évènements peuvent être de plusieurs types, selon qu'il s'agisse d'évènements à l'entrée qui vont déclencher un comportement particulier ("*eventIn*") du noeud ou des évènements déclenchés par un comportement d'un noeud ("*eventOut*"). Enfin, il existe des champs d'un type particulier appelé "*exposedField*". Ces derniers avec les "*eventIn*" et les "*eventOut*" sont les champs qui peuvent être modifiés directement de l'extérieur d'un noeud par d'autres noeuds par un langage de script par exemple.

Entre deux champs peut exister ce qui est appelé une route. Elle relie un *eventOut* à un *eventIn*. Ceci permet aux évènements de se propager ainsi d'un noeud à un autre.

Enfin, il est possible de fabriquer ses propres noeuds VRML et d'ajuster leur comportement à l'aide de scripts rédigés en Javascript ou VRMLScript par exemple. Ces noeuds faits sur mesure sont appelés "*PROTOS*". Il posséderont dans le fichier VRML une description avec la liste de leur propriétés. Plus loin dans le fichier, et plus précisément dans le graphe de scène, nous trouverons des instances de ces PROTOS où les valeurs des différents champs seront initialisées.

Sur nos modèles VRML, nous allons donc concevoir de nouveaux noeuds (donc des "*PROTOS*") dont les champs et les évènements vont refléter les divers aspects des opérations de montage/démontage comme des animations de dévissage, de retrait de pièces, ou d'objets devenant transparents (pour simuler temporairement une vue sur l'intérieur du dispositif).

Toutefois, ces prototypes doivent être suffisamment génériques pour être transposables à toutes les parties du modèle. Ils seront employés comme des noeuds de groupage VRML tels que "*Group*" ou "*Transform*" qui peuvent contenir toute sorte de géométrie (et de manière plus générale toute sorte de noeuds). Pour parachever le tout, nous pourrons lier les instances de prototypes entre elles afin de créer des animations concurrentielles et séquentielles.

Nous allons à présent détailler la conception des prototypes utilisés.

#### 4.3.2.1 La conception de nouveaux noeuds VRML pour la maintenance assistée en réalité augmentée

Comme nous l'avons dit plus tôt, les opérations de maintenance sont principalement constituées d'opérations de montage/démontage enchaînées suivant une séquence spécifique. Nos prototypes doivent donc fournir une simulation visuelle de l'opération de maintenance en animant certaines parties du modèle. Mais les animations en elle-même sont insuffisantes. Nous devons également capturer l'attention de l'opérateur sur une action spécifique. En d'autres termes, aux animations nous devons associer un indice visuel (éventuellement animé) qui renforce symboliquement l'action qui est effectuée.

Par exemple, le dévissage d'un écrou sera renforcé par une flèche tournante qui indiquera le sens de rotation. De même le retrait de ce dernier sera renforcé par une flèche indiquant la direction de retrait. Ces indices visuels seront bien sûr mis en évidence par une couleur vive (dans notre cas du vert fluorescent).

Nous allons maintenant nous focaliser sur le développement du prototype de dévissage "*Unscrewer*". Ce prototype peut-être considéré comme représentatif de l'ensemble des protos développés dans la mesure où ils partagent le même principe sous-jacent.

#### 4.3.2.2 Le prototype de dévissage "*Unscrewer*"

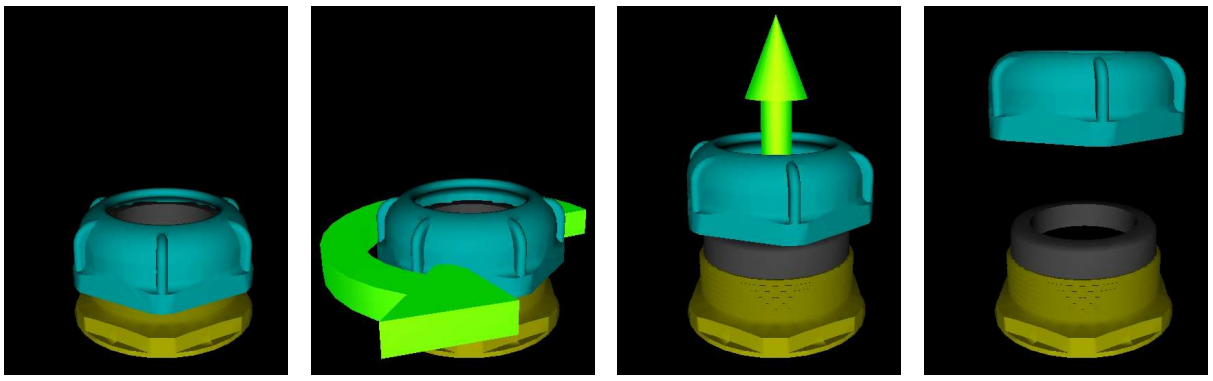
Ce prototype est employé à chaque fois que nous avons besoin de dévisser et de retirer une partie de mécanisme. La géométrie des pièces à dévisser est contenue dans le champ *children* du prototype ce qui nous permet de dévisser n'importe quel groupe de pièces. Nous pouvons voir l'architecture complète du prototype à la figure 4.7 page 133. Ce prototype se compose des étapes suivantes (voir aussi la figure 4.6 page suivante) :

- L'animation est activée lorsque le prototype reçoit un évènement routé sur un évènement d'entrée nommé "*startTime*". Le durée de l'animation exprimée en secondes est spécifiée dans le champ "*unscrewTime*" et le champ "*removePercent*" indique le pourcentage de l'animation qui est consacré à l'action de retirer l'objet.
- La partie mécanique est d'abord dévissée par un nombre de tours fixés par le le champ "*nbRounds*" avec un pas de filetage (champ "*stepSize*"). Ces paramètres permettent un dévissage partiel ou, au contraire, complet. Le dévissage est également mis en évidence par un indice visuel qui tournera dans le même sens que ce dernier qui est paramétré par le champ "*hinturl1*".
- Une fois le dévissage terminé, la partie dévissée est retirée, c'est à dire déplacée à une certaine distance ( paramétrée par le champ "*removeDist*"). Le retrait est lui aussi indiqué par un indice visuel indiqué par le champ "*hinturl2*".
- À la fin de chaque étape, l'indice visuel qui lui correspondait (et qui apparaissait au début de celle-ci) disparaît,
- Quand toutes les animations sont finies, le prototype émet un nouvel évènement (évènement de sortie "*stopTime*" qui peut-être utilisé pour déclencher une autre animation basée sur un prototype de même facture. Ceci permet d'enchaîner les animations de manière séquentielle ou concurrentielle.

#### 4.3.2.3 Augmenter les étapes de la procédure de maintenance

Les étapes de la procédure sont instanciées dans le modèle virtuel en liant les instances de différents prototypes comme nous l'avons expliqué précédemment.

Des noeuds de type "*TouchSensors*" sont insérés dans le modèle virtuel dans le but de déclencher un prototype semblable à celui que nous venons de décrire. Suivant le type d'utilisation du modèle



(a) État initial (b) Étape de dévissage (c) Étape de retrait de la pièce (d) État final

FIG. 4.6 – Étapes de dévissage et de retrait d'une partie mécanique

virtuel, ce "*sensor*" sera activé manuellement ou automatiquement par passage d'une étape de la procédure de maintenance à une autre. La figure 4.9 page 134 présente un extrait du graphe de scène et des routes associées à une sous-étape de la procédure de maintenance : 6 boulons doivent être dévissés avant de retirer une plaque. La séquence d'animation (représentée en partie à la figure 4.8 page 134) emploie des routes entre les champs *stopTime* des noeuds dont l'animation se finit et les champs *startTime* des noeuds décrivant les animations suivantes :

1. Le premier boulon est dévissé et le plaques couvrant le transformateur deviennent transparentes pour permettre de voir temporairement à l'intérieur du dispositif à maintenir (figure 4.8(a)),
2. Le deuxième boulon est alors dévissé,
3. La même action est appliquée au troisième boulon,
4. Enfin, les trois derniers boulons sont dévissés simultanément (figure 4.8(b)),
5. Quand le retrait des plaques est effectué, les plaques couvrantes redeviennent opaques (figure 4.8(c)).

En définitive, nos prototypes sont relativement simples mais il ont l'avantage d'être suffisamment génériques pour pouvoir être appliquées sur tout type de démontage quel que soit la maquette virtuelle. Dans le cadre du projet AMRA qui était un projet exploratoire nous avons fourni la preuve de faisabilité en instrumentant la maquette virtuelle manuellement. Toutefois, parmi les perspectives nous pouvons envisager une intégration de ces protos VRML dans un outils de génération automatique de modèles instrumentés directement à partir d'un modèle spécifiquement préparé pour l'augmentation et de la procédure de maintenance électronique.

Les modèles 3D présentés ainsi que les procédures de maintenance s'inscrivent dans un ensemble de documents multimédia qui sont employés par les différents composants développés par le LSC dans le cadre du projet AMRA. Ces derniers forment un moteur multimédia que nous allons à présent détailler.

## 4.4 Moteur multimédia pour la réalité augmentée

Dans le projet AMRA, nous avons mis l'accent sur l'aide que nous voulions apporter au mainteneur par le biais d'informations contextuelles. En conséquence, un moteur multimédia a été développé qui permet d'apporter une aide contextuelle à l'opérateur à chaque étape de la procédure en utilisant un ou plusieurs médias tels que des photos, de la vidéo, de la vidéo augmentée ou des modèles 3D seuls.

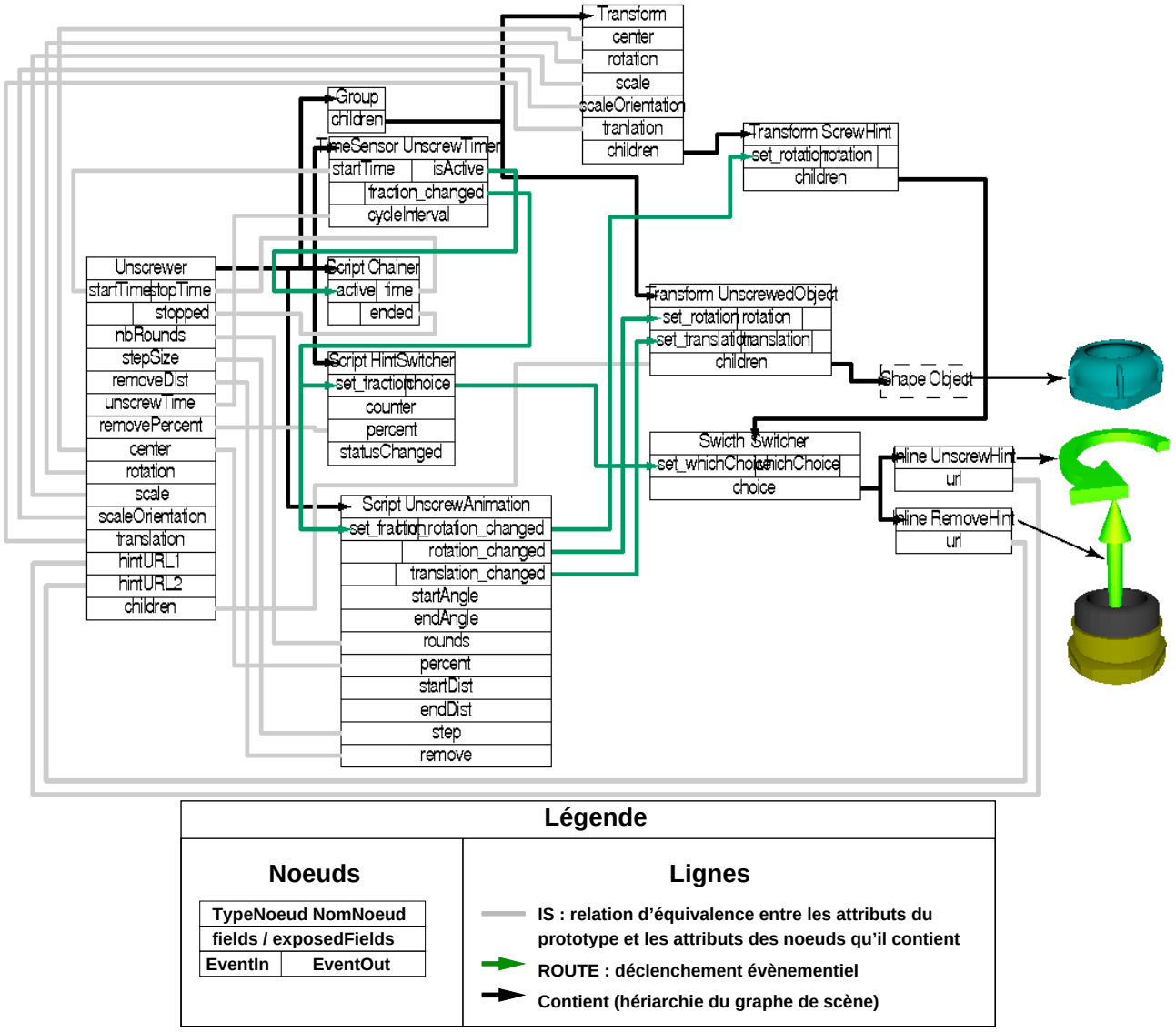



FIG. 4.7 – Architecture du prototype de dévissage *Unscrewer*

La particularité essentielle de ce moteur multimédia tient au fait que la réalité augmentée introduit un certain nombre de médias. L'originalité essentielle par rapport aux moteurs multimédia classiques est surtout la mise en correspondance entre le monde réel et les informations virtuelles.

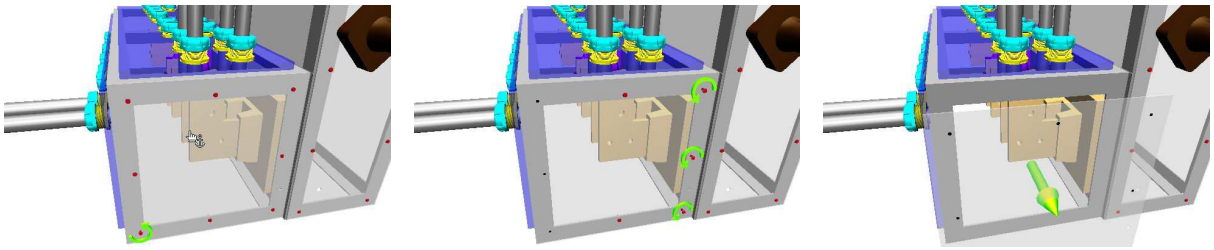
Nous avons représenté sur la figure 4.10 page 136 les différentes sources d'interaction entre les contenus multimédias, le suivi d'une procédure de maintenance, les évènements opérateur et les informations de recalage qui doivent interagir avec le moteur multimédia.

Un certain nombre de choix reposant sur des standards d'une part et sur des librairies multiplateformes d'autre part on été effectués lors du développement des classes constituant ce moteur multimédia.

### 4.4.1 Standards supportés

Le prototype AMRA étant une application destinée au monde industriel, nous avons du développer nos composants en vue de respecter un certain nombre de standards au niveau du format des médias employés. Parmi ces standards, nous comptons le format de document XML utilisé pour les procédures de maintenance qui a fait l'objet d'une étude détaillée à la section 4.2.2 page 126.

De plus, pour ce qui est de l'affichage d'images, un certain nombre de standards sont également



(a) Démontage du premier boulon (b) Démontage des trois derniers boulons (c) Retrait de la plaque

FIG. 4.8 – Etapes d'un enchaînement d'animations

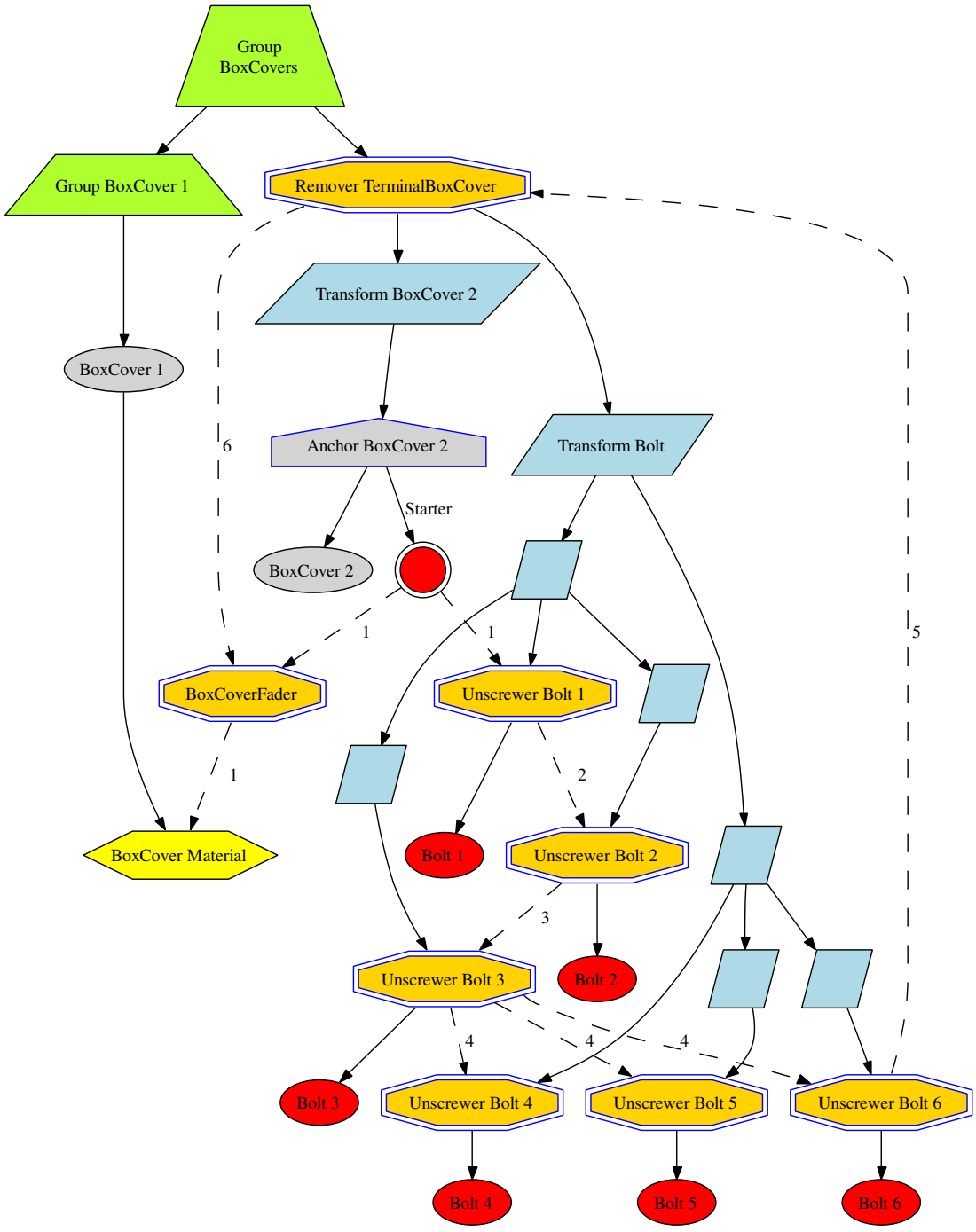



FIG. 4.9 – Graphe de scène (trait plein) et graphe des routes (en pointillés) associés à une simulation d'une étape d'une procédure de maintenance. Les octogones sont les instances des *PROTOs* employés

supportés : PNG, BMP, XBM, XPM, PNM, et JPEG. Les différents formats PNM sont : PBM (P1 ou P4), PGM (P2 ou P5), et PPM (P3 ou P6).

Comme nous venons de le présenter dans la section précédente, pour le choix du format des augmentations de type 3D utilisées, nous avons opté pour un format ouvert et largement répandu en matière de description de scène virtuelle : VRML 97 (Virtual Reality Modeling Langage). Ce format permet également de rendre interactif les scènes 3D décrites par le biais de langages de script. Le langage de script choisi est également très répandu : il s'agit de Javascript. Dans les annexes C page 189, une étude complémentaire donne le taux exact de conformance aux standards de la solution développée, tant pour VRML que pour Javascript. Elle donne aussi les solutions que nous avons adoptées pour les trois types de limitations que nous avons rencontrées, à savoir :

- les limitations sur l'interprétation du format VRML,
- les limitations sur l'interprétation des scripts Javascript,
- les limitations sur l'implémentation du standard VRML en Javascript.

Plus particulièrement, pour le format VRML, les difficultés que nous avons rencontré sont dues à l'implémentation du standards VRML par la librairie que nous avons choisie qui sont :

- Un problème d'import des noeuds de type Switch de la spécification VRML,
- Une implémentation déficiente du mot clé *IS* de la relation d'équivalence dans la spécification des PROTOs VRML,
- Une restriction des possibilités de routage dans un fichier VRML,
- Une restriction sur l'implémentation des protos externes.

L'un des avantages des standards est qu'il existe des librairies permettant de les exploiter qu'il suffit d'interfacer avec nos propres applications. Nous allons maintenant voir celles que nous avons employées dans le cadre de notre moteur multimédia.

### 4.4.2 Librairies employées

Pour permettre de gérer l'ensemble de médias cités, ce moteur doit implémenter plusieurs particularités. Il utilise plusieurs interfaces de programmation (API - Application Programming Interface) qui permettent de lier des librairies à nos composants. Les composants du moteur multimédia pour AMRA exploitent les fonctionnalités de trois librairies. Ces trois librairies présentent l'avantage d'être multiplateforme ce qui offre à nos composants l'avantage d'être portables d'une plate-forme à une autre. Notre moteur, écrit entièrement en C++, nous permet de lire et d'interpréter les fichiers XML décrivant les procédures de maintenance grâce à la librairie QT, qui nous permet également de lire et d'afficher plusieurs types de formats d'images. OpenInventor a été employé pour permettre la lecture des fichiers VRML/X3D qui décrivent les modèles 3D. Enfin, pour exécuter le code javascript employé pour animer les modèles 3D, nous utilisons la librairie NJS (l'ancien moteur Javascript de navigateur Netscape) qui nous permet d'interpréter le code Javascript utilisé pour les animations.

Nous avons représenté à la figure 4.10 page suivante le moteur multimédia ainsi que les différents éléments qui interagissent avec lui.

Des médias supplémentaires peuvent être pris en compte grâce à un système de plug-ins qui permet d'ajouter des gestionnaires de médias spécifiques pour chaque nouveau type de média. Ces plug-ins sont enregistrés durant l'initialisation de l'application.

Comme nous l'avons écrit, les médias sont gérés par un ensemble de composants qui constituent le moteur multimédia en lui-même. Nous allons donc présenter les composants développés à cette occasion.

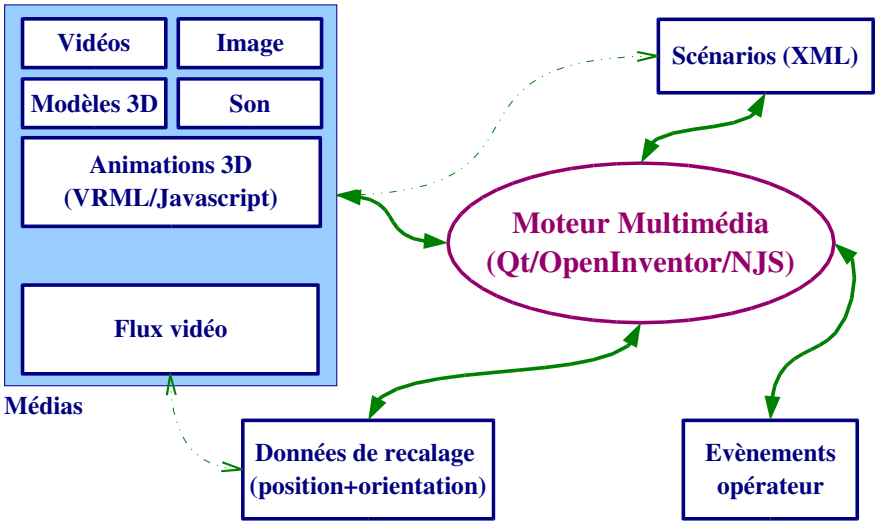


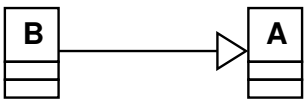

FIG. 4.10 – Moteur multimédia pour la réalité augmentée

### 4.4.3 Composants développés

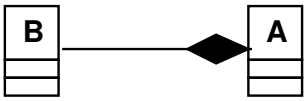
Avant de rentrer dans les détails, nous allons présenter le diagramme des classes développées. Toutefois, ce diagramme étant réalisé avec la norme UML, nous allons expliquer les éléments nécessaires à la compréhension de ce dernier.

#### 4.4.3.1 Lecture des relations UML

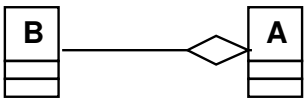
Les rectangles représentent les classes et les segments reliant les rectangles représentent des relations entre les classes. Ces relations peuvent être de plusieurs type. Nous montrons ici celles qui seront employées dans notre diagramme.




Se lit “la classe B hérite des propriétés de la classe A”.




Se lit “un objet de la classe B est un constituant de la classe A”. Ici, la destruction d’un objet de la classe A *entraîne la destruction* de son constituant de classe B.




Se lit “un objet de la classe B est un constituant de la classe A”. Ici, la destruction d’un objet de la classe A *n’entraîne pas la destruction* de son constituant de classe B.

#### 4.4.3.2 Diagramme des classes

A présent que le principe du diagramme est expliqué, nous pouvons visualiser l’architecture générale qui est exposée dans la figure 4.11 page suivante.

En quelques mots, les traitements et opérations pour lesquelles ces classes ont été conçues sont expliquées dans le tableau 4.3 page 138.

Parmi ces classes, certaines sont transformées en composants tel que nous les avons définis dans le chapitre 2 (plus précisément à la section 2.4 page 50) alors que d’autres sont utilisées uniquement pour assurer la cohérence des données ou l’encapsulation de ces dernières.

Parmi ces classes, les composants tels que nous les avons définis sont : *RAXml*, *VRMLImageManager*, *ImageMediaManager* et enfin *RAQtWidget*. Ce sont eux qui, une fois instanciés mettent en place le système de gestion des augmentation en fonction d’une procédure de maintenance. Toutefois, il convient d’aborder la manière dont le lien est fait entre la procédure de maintenance et les divers

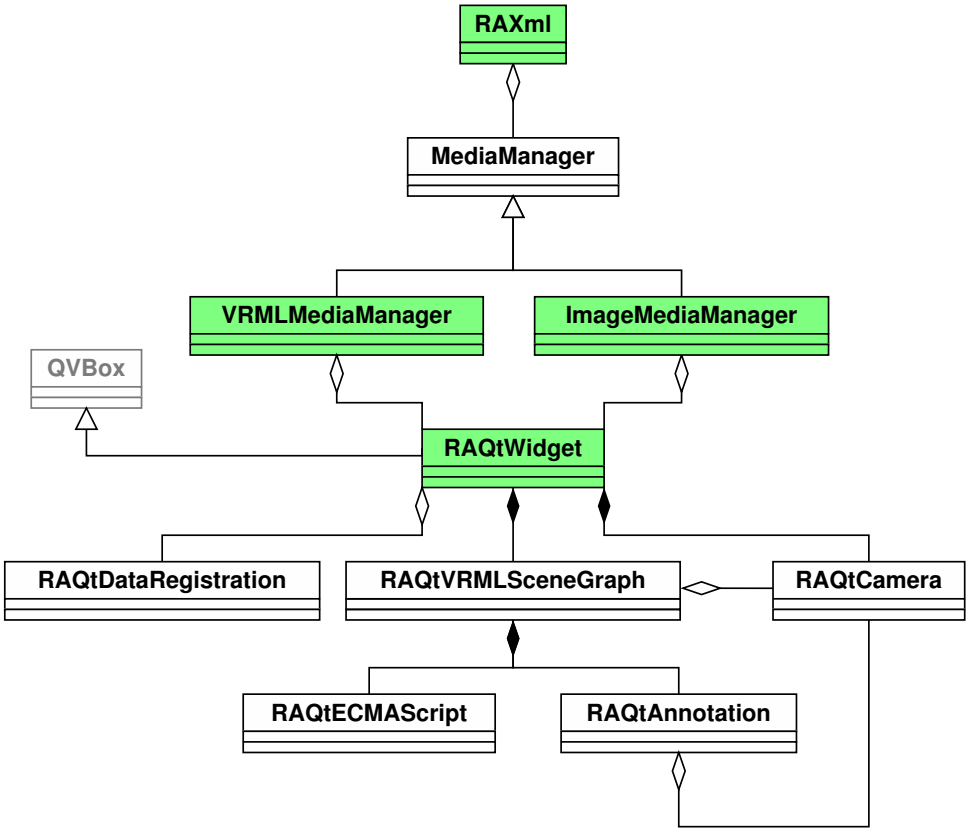



FIG. 4.11 – Diagramme UML des classes développées par le LSC. En vert se trouvent les classes qui sont également des composants.

médias employés. Pour cela, nous allons examiner le fonctionnement du composant *RAXml* dans le détail.

#### 4.4.3.3 Lien entre les procédures de maintenance et le moteur multimédia

Comme nous l'avons vu précédemment, chaque procédure de maintenance contient une liste de documents multimédia qui l'accompagne. Chaque média est répertorié avec son type MIME (voir section 4.2.2 page 126). A chaque type va correspondre un gestionnaire de média associé qui doit être enregistré avant l'utilisation des procédures de maintenance. Ces composants doivent hériter de la classe *MediaManager* que nous avons présentée dans notre diagramme de classes. Ainsi, si un média d'un certain type doit être "joué", le gestionnaire de média correspondant est appelé. La classe `MediaManager` est une classe abstraite qui doit être réimplémentée pour chaque type de médias appelé à être géré par l'interface d'AMRA.

Cette classe met également en place des mécanismes pour faciliter l'accès aux médias qui doivent être lus. L'un des principes important est le préchargement ou 'prefetch' des médias. Chaque média doit être enregistré par un identifiant lors de cette phase. Il sera alors possible de mettre le média en question en mémoire ou de mettre en place les composants ou objets qui faciliteront sa lecture ultérieurement. L'activation du média s'effectue plus tard lorsqu'une étape de la procédure de maintenance à laquelle il est lié est activée, le média étant nommé par un identifiant.

Toute liberté est donnée au développeur pour effectivement précharger le média ou non. Toutefois, il devra nécessairement passer par la phase d'enregistrement du média auprès du gestionnaire.

Deux exemples de gestionnaires de médias sont apportés dans l'architecture du système AMRA. Il s'agit des composants :

– *ImageMediaManager* : un gestionnaire d'images qui affiche des images simples,
– *VRMLMediaManager* : un gestionnaire d'augmentations 3D qui permet d'afficher les modèles

| Classe | Description sommaire |
|---|---|
| *RAxml* | gère les procédures de maintenance XML |
| *MediaManager* | interface générique de gestion d'un type de média |
| *VRMLMediaManager* | gestionnaire de médias au format VRML/X3D |
| *ImageMediaManager* | gestionnaire de médias de type image |
| *RAQtWidget* | widget de mixage réel/virtuel prenant également en entrée le flux vidéo capturé par la caméra du monde réel |
| *RAQtDataRegistration* | classe d'encapsulation des données de recalage |
| *RAQtVRMLSceneGraph* | classe gérant les graphes de scènes VRML |
| *RAQtCamera* | classe gérant les caméras réelles pour les faire coïncider avec les caméras de la scène virtuelle |
| *RAQtECMAScript* | moteur d'interprétation des scripts Javascript inclus dans les fichiers VRML |
| *RAQtAnnotation* | classe gérant le graphe de scène associé à une annotation du graphe VRML |

TAB. 4.3 – Description sommaire des classes développées

VRML spécialement conçus pour notre application.

Ces derniers sont une sur-couche du composant *RAQtWidget* qui effectue le mixage entre les images réelles prise par la caméra et les données multimédia graphiques, notamment les modèles 3D, qui nécessitent des moyens d'interaction particuliers. Nous allons à présent voir comment l'utilisateur peut interagir avec ses derniers grâce au composant développé.

### 4.4.4 Manipulation des modèles 3D

Plusieurs modes d'interaction sont possibles en utilisant les modèles 3D fournis à l'opérateur. Nous les avons représentés à la figure 4.12 page 140. Ce dernier peut interagir de manière passive avec le modèle virtuel en laissant la procédure de maintenance lancer les animations dès qu'il passe d'une étape de la procédure de maintenance à une autre. L'opérateur peut également interagir directement avec le modèle, soit en mode non recalé pour voir le détail du modèle 3D, soit en cliquant dessus pour lancer les animations correspondant aux différentes étapes de la procédure, ou enfin en l'utilisant pour demander des informations au système sur la pièce qu'il aura désigné.

Dans ce cas, le modèle 3D doit être recalé sur le monde réel. Nous pouvons alors, à l'aide d'un rayon virtuel projeté dans l'espace virtuel entre le point de vue de la caméra et un point désigné par l'utilisateur sur le plan image, désigner une partie du modèle virtuel en détectant l'intersection de ce rayon avec les pièces rencontrées. Ceci nous permet de retrouver l'identifiant de la pièce correspondante sur le modèle virtuel et de l'afficher, voir de faire une requête à la base de données concernant cette pièce pour obtenir des renseignements complémentaires. Le système permet également d'afficher le nom des pièces ainsi désignées sous forme d'annotations. Cette fonction peut-être employée tant sur le modèle virtuel que sur la vue réelle de l'image comme nous pouvons le voir sur la figure 4.12(a) page 140, si tant est qu'un modèle virtuel (visible ou non) soit recalé.

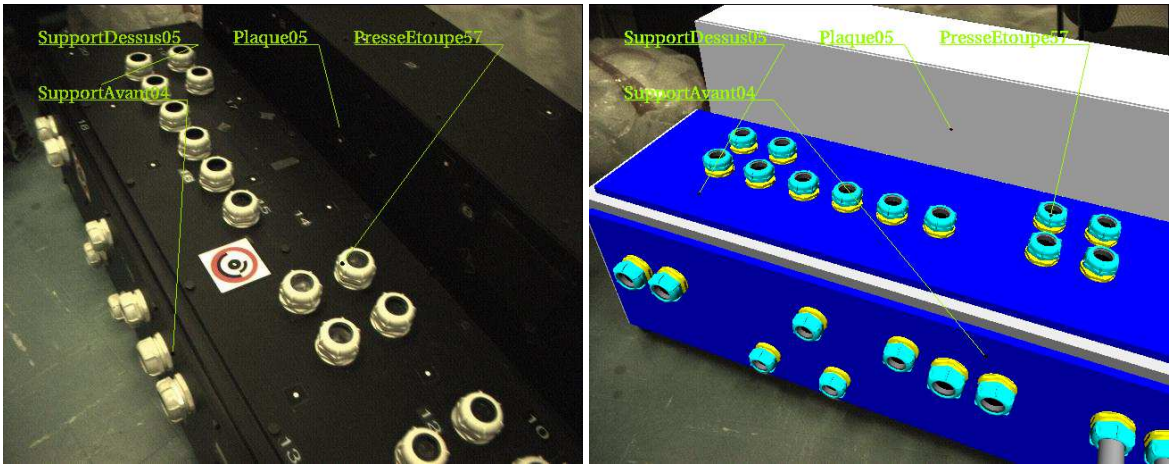
Afin de pouvoir employer les modèles 3D avec la librairie OpenInventor, nous avons créé un graphe de scène compatible avec les augmentations VRML et les augmentations supplémentaires telles que les annotations. La structure de graphe de scène employée est représentée à la figure 4.13 page 141 Cette figure donne la structure des graphes de scène OpenInventor tel que le widget de mixage les exploite. Pour chaque noeud, son nom est spécifié ainsi que son type qui correspond à un type de noeud OpenInventor. Outre la structure du graphe de scène principal qui est employée dans le composant *RAQtWidget*, nous retrouvons ici la structure qui est lié au graphe de scène VRML lorsqu'il

est chargé à partir d'un fichier et dont la structure est contenue dans la classe *RAQtVRMLSceneGraph*. L'image de ce graphe de scène principal est une des images du flux vidéo acquis par la caméra. Nous retrouvons également le graphe de scène qui est créé à chaque fois qu'une annotation apparaît à l'écran suite à la demande de l'opérateur , tel qu'il est défini dans la classe *RAQtAnnotation*.

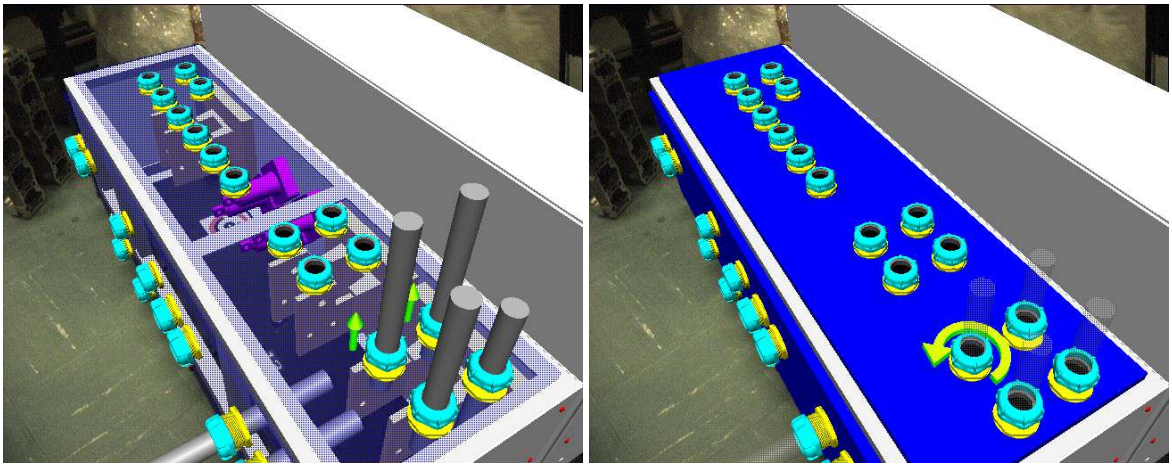
## 4.5 Conclusion

Nous avons présenté l'ensemble des fonctionnalités développées par le LSC au cours du projet de recherche AMRA. Ces fonctionnalités, qui font que le système AMRA intègre en réalité un moteur multimédia pour la réalité augmentée, couvrent divers domaines : tant celui de l'exploitation d'un scénario de maintenance décrit dans un format XML, que d'une interface générique de gestion des médias attachés à cette dernière, ainsi la représentation visuelle des informations contenues dans ces médias tout en les rendant cohérentes par recalage avec l'environnement réel de l'opérateur en train d'effectuer des opérations de maintenance. Ceci nous a également permis de développer les composants nécessaires à notre système de réalité augmentée pour permettre la visualisation et le mixage d'un flux vidéo capturant le monde réel avec les entités virtuelles.

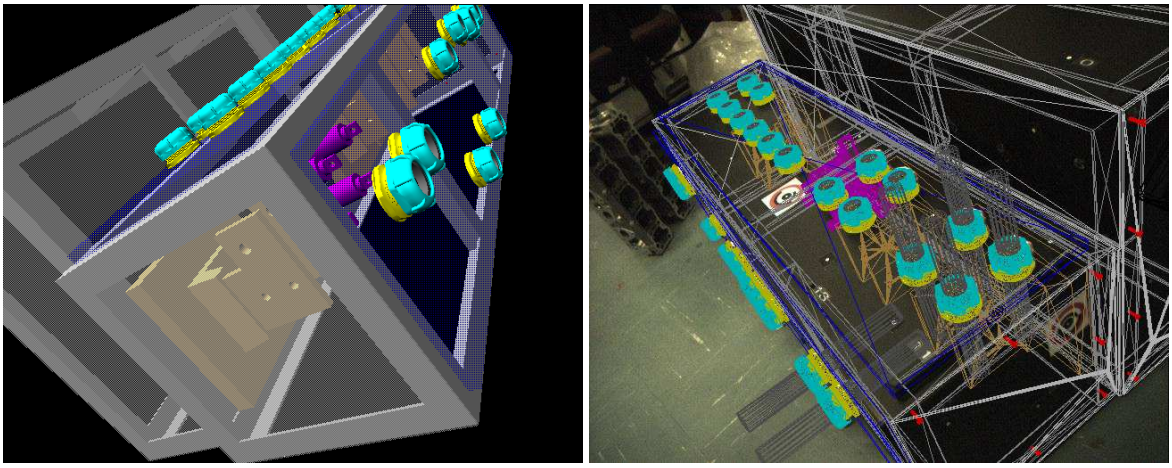
Nous allons à présent voir comment nous pouvons partiellement modifier ce système général pour pouvoir le transposer dans le cadre de la problématique en réalité augmentée en vision directe, ce qui nécessite la résolution de problèmes spécifiques. L'idée est en effet de fournir à terme pour le mainteneur une assistance main-libre, ce qui, dans notre cas, implique d'utiliser un casque de réalité virtuelle semi-trasnparent à la place du tablette-PC.




(a) Retrouver les noms d'une partie du modèle



(b) Animer une étape de la procédure de maintenance



(c) Autres modes d'exploration du modèle virtuel

FIG. 4.12 – Différents modes d'interaction avec le modèle 3D.

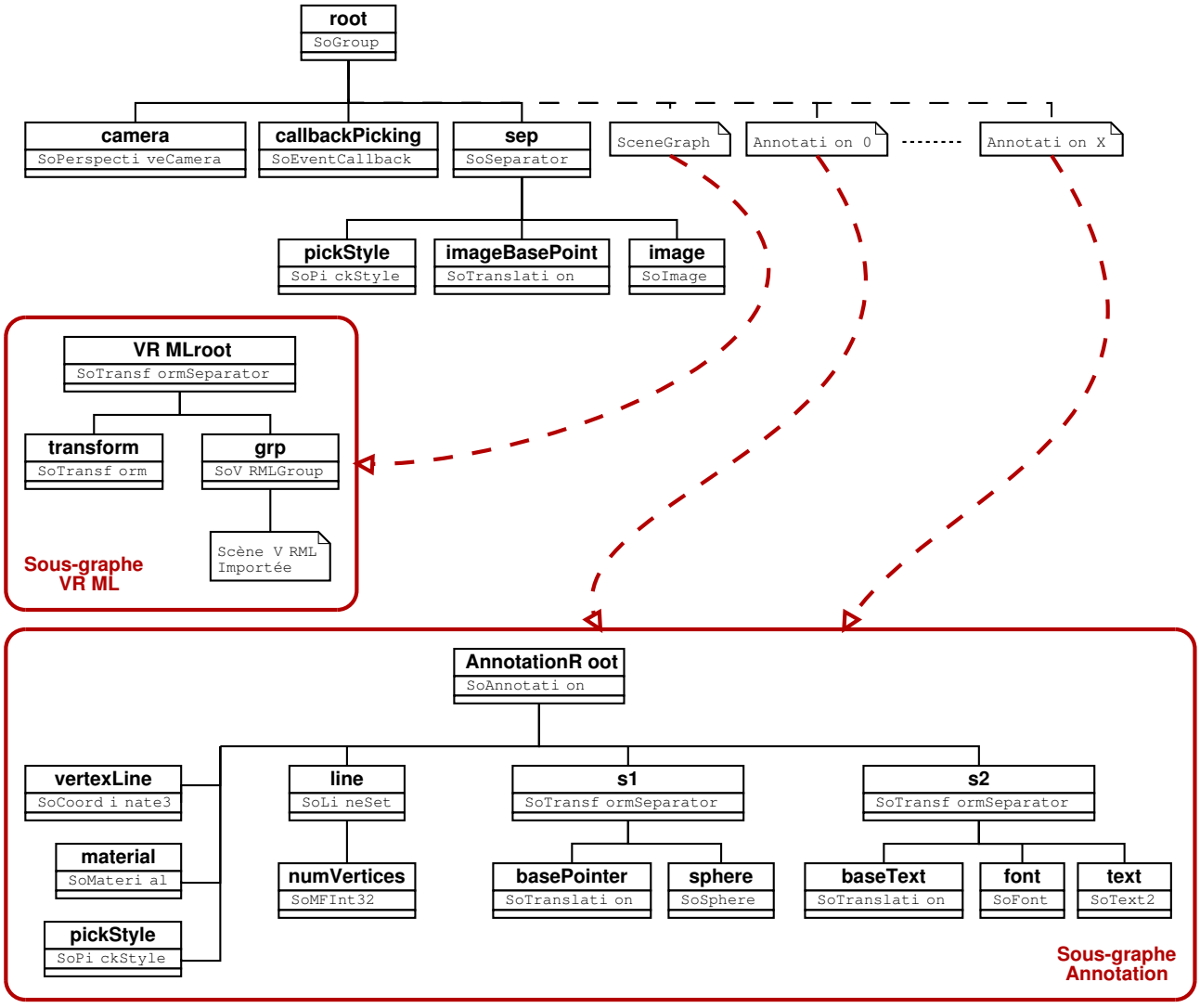



FIG. 4.13 – Graphe de scène général employé

# Chapitre 5

# Techniques particulières à la réalité augmentée en vision directe

Dans le cadre de la réalité augmentée en vision directe, la vue de l'opérateur est directement augmentée par un dispositif semi-transparent qui affiche les entités virtuelles et qui est placé entre lui et les objets observés. Ce peut-être une plaque de verre fixe qui reprojette les informations par holographie (une tel système est utilisé par [Olwal et al., 2005]) ou un casque de réalité virtuelle pourvu d'un système optique semi-transparent ("*optical see-through head mounted display*"). Par rapport au travaux que nous avons effectué, nous nous intéressons surtout à la deuxième catégorie de dispositifs.

À la différence de la vision indirecte où la représentation du monde réel est acquise par une caméra, la vision directe nécessite de synchroniser les entités virtuelles générées par rapport à la vision de l'utilisateur. Dans le cadre de cette synchronisation, un facteur important entre en jeu : la latence. Il s'agit du temps compris entre l'acquisition des données nécessaires à localiser l'opérateur (pour savoir où ce dernier regarde) et l'affichage effectif des entités virtuelles sur l'écran. Si cette latence n'est pas compensée, il se produit un décalage visuel entre les entités virtuelles et les objets réels, les premières semblant poursuivre les seconds sans jamais toutefois les rattraper lorsque la tête de l'opérateur est en mouvement.

Pour bien comprendre le problème de la latence, nous allons commencer par l'étudier et analyser les sources de latence, puis nous verrons comment cette latence est habituellement compensée dans la communauté de réalité augmentée en employant des méthodes de prédiction de point de vue similaires au problème du filtrage des données. Nous verrons également que la communauté de réalité virtuelle emploie d'autres méthodes de compensation de la latence basée sur les méthodes dites de "*post-rendering*". Après cette phase préliminaire, nous expliciterons plus précisément le problème du filtrage prédictif par le biais de deux filtres : le filtre de Kalman et sa version étendue qui sont largement utilisés, puis nous présenterons le filtre particulaire que nous comparerons en simulation avec les approches classiques. Enfin, nous montrerons comment nous pouvons combiner les approches différentes de la réalité augmentée et de la réalité virtuelle pour compenser efficacement la latence.

## 5.1 Le problème de la latence

Ce cas est le plus simple pour synchroniser les informations de localisation et l'affichage des augmentations. Il ne compense pas la latence mais donne les moyens de comprendre et d'analyser les sources de latence. Une analyse complète de ces dernière a par ailleurs été faite par Holloway [Holloway, 1995]. Nous nous contenterons d'une version simplifiée de son modèle pour aider à la compréhension du problème de la latence.

Si nous prenons un capteur de localisation quelconque, avant que nous puissions afficher les entités virtuelles, il nous faut acquérir les données par le biais du capteur, transmettre ces dernières du capteur

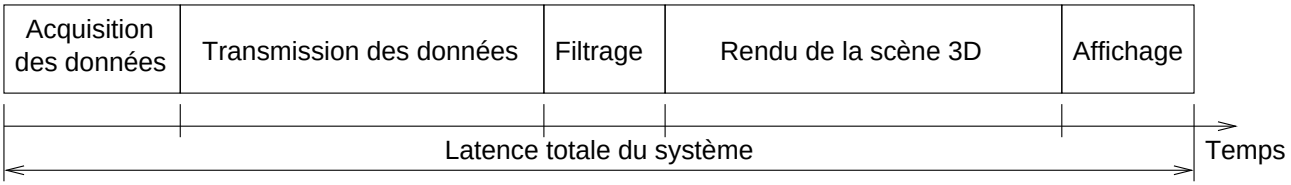



FIG. 5.1 – Décomposition du temps de latence global du système

au système informatique, calculer le point de vue de l'opérateur à partir de ces données, effectuer le rendu de la scène virtuelle en 3 dimensions et enfin l'afficher comme ceci est représenté sur la figure 5.1.

Dans le cadre du système de localisation que nous avons développé au chapitre 3.4 page 95, en sachant que l'acquisition de l'image, suivie de la localisation et du rendu de la scène s'effectue à la cadence de 18 images par secondes environ, le temps de latence global du système est d'au mieux 55 millisecondes. Sachant que le mouvement de la tête d'un opérateur peut atteindre en moyenne 2 radians par seconde, il est possible d'obtenir un décalage angulaire d'environ $20^o$ entre l'objet visualisé et l'image affichée, ce qui représente environ 10 centimètres d'erreur à 1 mètre de distance. Il devient donc capital de compenser la latence dans le cadre d'un système de réalité augmentée en vision directe si l'on souhaite conserver l'alignement de la scène virtuelle avec l'environnement réel observé par l'opérateur quel que soient les mouvements de celui-ci. Avant de proposer nos propres solutions à ce problème, nous allons voir quelles sont les méthodes de compensation existantes dans la littérature.

## 5.2 État de l'art sur la compensation de la latence

Deux approches différentes existent pour compenser la latence globale du système. La première consiste à effectuer une prédiction du point de vue à priori avant d'effectuer le rendu de la scène. La deuxième consiste à effectuer une modification à postériori de la scène dont on a effectué le rendu.

### 5.2.1 Les méthodes de prédiction du point de vue

Comme nous l'avons précisé précédemment, la latence provoque un décalage entre le réel et le virtuel lorsque la tête de l'utilisateur est en mouvement. L'une des solutions est d'inclure des algorithmes de prédiction des mouvements du point de vue, c'est à dire de la caméra ou de la tête de l'utilisateur.

Une des méthodes envisagées est celle de Albrecht, qui a été utilisée sur un casque de réalité virtuelle muni d'un capteur magnétique Polhemus [Albrecht, 1989]. Ce système prédit uniquement l'orientation de la tête. L'idée est de conserver les N mesures les plus récentes et de leur affecter des poids à chacun. La somme de ces valeurs est effectuée, ce qui nous donne une prédiction de la future position. L'intervalle de prédiction est fixé à 100 ms et suppose que les relevés du capteur sont espacés régulièrement dans le temps.

En 1991, Liang utilise un filtre de Kalman pour prédire l'orientation et la position de la tête de l'utilisateur [Liang et al., 1991]. Le modèle employé suppose que les rotations de la tête ne sont que peu fréquentes et que les translations se font pour la plupart dans la direction du regard. L'accélération de la tête est un modèle de Gauss-Markov de la forme $A = -\beta V + Kw(t)$. $A$ et $V$ représentent l'accélération et la vitesse. $\beta$ et $K$ sont des constantes et $w(t)$ représente une variable qui suit une distribution de loi normale $N(0, 1)$ qui représente les variations de la vitesse au cours du temps. Les données de leur capteur Polhemus sont au préalable lissées à l'aide d'un filtre passe-bas. Ces données

sont supposées transmises à intervalle de temps constant.

La méthode mise au point par Azuma en 1995 se base sur l'utilisation de capteurs de position et d'orientation ainsi que de capteurs inertiels [Azuma, 1995]. Il utilise plusieurs filtres de Kalman (la formulation détaillée de l'utilisation de ce filtre se trouve à la section 5.3 page 147 ) non seulement pour prédire le mouvement de la tête de l'utilisateur mais aussi pour prédire le temps entre chaque relevé des capteurs.

Chaque translation en $x, y$ et $z$ est traitée séparément par un filtre de Kalman pour lequel l'état est constitué de la position, de la vitesse et de l'accélération instantannées grâce à l'utilisation des capteurs inertiels. Le vecteur de mesure est constitué de la position et de l'accélération.

Pour l'orientation, un filtre de Kalman étendu est utilisé qui a pour vecteur d'état un quaternion représentant l'orientation, les vitesses angulaires et les accélérations angulaires. Le vecteur de mesure est constitué d'un quaternion et des vitesses angulaires.

Ce modèle a prouvé que l'utilisation d'un capteur inertiel aide grandement à la prédiction puisque ce dernier fournit des informations supplémentaires quant à l'accélération instantanée du mouvement. L'erreur moyenne de recalage en $mm$ pour les positions et en $^\circ$ pour les orientations est divisée par un facteur 2, voire 3 par rapport aux méthodes précédentes.

Depuis, les travaux se poursuivent sur l'utilisation des filtres permettant de prédire les mouvements de la tête de l'opérateur. Nous pouvons citer les travaux de Chai qui utilisent une autre équation d'état basée sur les angles d'euler qui introduit moins de non-linéarités [Chai et al., 1999]. Parallèlement, des capteurs hybrides employant la vision et une centrale inertielle ont été également conçus, dans ce cas précis, non pour prédire le mouvement, mais pour continuer à localiser le capteur hybride, même lorsque la partie localisation par la vision a décroché en cas de mouvements de trop forte amplitude. Nous pouvons citer le capteur hybride développé par You et Neumann [You et Neumann, 2001] qui pour la partie extrait des primitives géométriques de l'image observée ainsi que celui fabriqué et commercialisé par Intersense [Foxlin et Naimark, 2003] qui utilise le système de cibles présenté en section 3.2.1.3 page 85.

Toutefois, la prédiction du mouvement de l'opérateur, si elle permet de compenser la latence n'assure pas un recalage parfait au vu des résultats que nous avons obtenus et dont nous discuterons en section 5.4 page 152. Nous allons à présent examiner d'autres méthodes de compensation basées sur un paradigme différent : les méthodes de *post-rendering*.

### 5.2.2 Les méthodes de *post-rendering*

Si la communauté de réalité augmentée s'appuie sur des algorithmes de prédiction pour réduire la latence, la communauté de réalité virtuelle a développé un ensemble de techniques propres, en particulier pour les casques de réalité virtuelle (ou Head Mounted Display - HMD). Ces méthodes sont nommées méthodes de *post-rendering* : elle consistent, une fois que le rendu de la scène virtuelle a été effectué, à appliquer des corrections sur l'image générée pour prendre en compte des données issues des capteurs qui seraient devenues disponibles après le début du rendu de la scène.

Les premières méthodes employées furent des méthodes effectuant des corrections en deux dimensions uniquement. Parmi les travaux remarquables, nous pouvons citer le dispositif de recalcul d'adresse développé par Regan et Pose [Regan et Pose, 1994] utilisé pour compenser les mouvements en rotation. Plus récemment, Kijima et al. ont conçu un HMD-réflexe qui a des moyens de compenser le mouvement directement au niveau du matériel en se servant d'une centrale inertielle [Kijima et Ojika, 2002]. Pour permettre cette compensation, ils effectuent le rendu de la scène virtuelle sur une image de dimensions plus grandes que celle qui peut être affichée dans le HMD. Ensuite, la compensation appliquée consiste à calculer une sous-fenêtre de visualisation dans cette

grande image qui sera affichée dans le casque. Cette méthode, basée sur la proposition de So et Griffin se fonde sur l'hypothèse que le mouvement de rotation est le plus important à compenser [So et Griffin, 1992]. Cette compensation va s'effectuer en calculant les translations horizontales et verticales ( $\Delta x$ et $\Delta y$) de la sous-fenêtre par rapport à l'image d'origine en fonction des angles de rotation ( $\Delta\theta$ et $\Delta\phi$ estimés ) comme nous pouvons le voir à la figure 5.2.

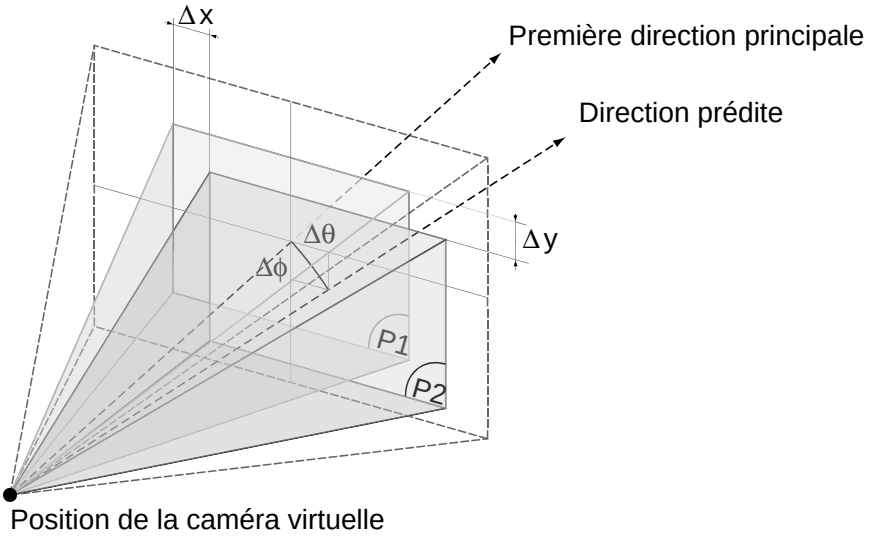



FIG. 5.2 – Méthode de translation d'une sous-fenêtre en fonction du mouvement de rotation

Toutefois, même implémentée directement au niveau matériel, cette méthode présente un inconvénient majeur puisqu'elle introduit des altérations majeures dans l'image. En effet, elle ne compense que deux degrés de liberté par rapport aux six degrés de liberté de mouvement d'un point de vue et elle ne peut donc pas compenser les déformations perspectives.

D'autres méthodes de post-rendering permettent de compenser directement les six degrés de liberté. Ces dernières utilisent l'image générée ainsi que la carte des profondeurs (*Z-buffer*) qui lui est associée [Mark et al., 1997] [Popescu et al., 2000]. À l'aide de la carte des profondeurs, il est possible de garder l'information 3D relative à l'image générée, qui sera ensuite corrigée pour appliquer la transformation correspondant à la compensation du mouvement. Ces méthodes nécessitent de remailler l'image d'origine partiellement sous peine de créer des trous ou des effets de gommages sur l'image corrigée. Ces méthodes restent toutefois très gourmandes en temps de calcul, voire parfois plus longues à appliquer qu'il ne faut de temps pour effectuer le rendu de la scène et donc nous ne pouvons pas les employer pour compenser la latence.

L'ensemble des méthodes présentées mettent en avant le compromis nécessaire entre les déformations de l'image ou le temps de calcul, comme nous allons le voir ultérieurement, nous avons appliqué notre propre algorithme de *post-rendering* qui compense parfaitement les mouvements de rotation et partiellement les mouvements de translation, au prix, de certaines déformations de l'image (voir section 5.5 page 156).

Avant d'introduire les méthodes particulières que nous avons développées dans le but de compenser la latence, nous allons d'abord nous intéresser au problème du filtrage, notamment à la formulation du filtrage de Kalman et du filtrage de Kalman étendu, ainsi que le filtrage particulaire qui seront employés dans notre méthode de prédiction. Nous verrons également comment appliquer ces filtres à un modèle des mouvements de la tête de l'opérateur.

## 5.3 Filtrage et prédiction des données

Nous allons présenter les équations généralisées qui régissent le filtre de Kalman. Dans le cas ou le système observé est régi par une équation linéaire et dans le cas ou l'équation devient non linéaire. Comme nous l'avons vu précédemment, le filtre de Kalman est employé par de nombreux systèmes afin de prédire le point de vue de l'utilisateur (section 5.2.1 page 144)) dans le but de compenser la latence.

### 5.3.1 Le filtre de Kalman discret

Ce dernier sert à estimer l'état $x \in \mathcal{R}^n$ d'un système qui est soumis à une équation différentielle stochastique linéaire avec une mesure $z \in \mathcal{R}^m$ [Welsh et Bishop, 1995]. Ce qui nous donne à chaque pas $k$ les équations d'état et de mesure suivantes :

$$x_k = Ax_{k-1} + Bu_k + v_{k-1} \text{ (équation d'état)} \quad (5.1)$$
$$z_k = Hx_k + e_k \text{ (équation de mesure)} \quad (5.2)$$

$u_k$ est une consigne appliquée en entrée au système. Les variables aléatoires $v_k$ et $e_k$ représentent respectivement le bruit du système et de la mesure. Ils sont supposés indépendants l'un par rapport à l'autre. De plus, ce sont des bruits blancs dont, par définition, la distribution suit une loi normale de moyenne nulle et de matrice de covariance non nulle (notée $N(0, X)$), ce qui nous donne, pour les densités de probabilité des variables $v$ et $e$ notées $p(v)$ et $p(e)$ :

$$p(v) \approx N(0, Q) \quad (5.3)$$
$$p(e) \approx N(0, R) \quad (5.4)$$

$A$ $(n \times n)$ est la matrice de prédiction qui relie l'état $x_{k-1}$ à l'état $x_k$.
$B$ $(n \times 1)$ relie $x_k$ à un éventuel signal de contrôle (ou consigne) $u_k$. C'est le vecteur de consigne.
$H$ $(n \times m)$ est l'équation de mesure qui relie $x_k$ et $z_k$.
$Q$ et $R$ sont des matrices de covariance associées aux bruits sur le processus à estimer et aux bruits sur la mesure.
Les matrices $A, B, H, Q$ et $R$ changent en fonction de l'évolution du processus.

On définit $\tilde{\hat{x}_k}$ l'état estimé à priori pour l'étape $k$ (ou estimateur à priori), la connaissance du système étant donnée avant l'étape $k$. Et $\hat{x}_k$ l'état estimé à posteriori (ou estimateur à posteriori) de $x_k$, connaissant la mesure $z_k$.

Nous pouvons introduire alors les matrices de covariance des erreurs à priori et à posteriori qui sont :

$$\tilde{P_k} = E[(x_k - \tilde{\hat{x}_k})(x_k - \tilde{\hat{x}_k})^T] \quad (5.5)$$
$$P_k = E[(x_k - \hat{x}_k)(x_k - \hat{x}_k)^T] \quad (5.6)$$

$E(X)$ désigne ici l'espérance mathématique de la variable $X$.

#### 5.3.1.1 Algorithme

Le filtre de Kalman fournit une solution récursive qui calcule, après chaque mesure, l'estimation optimale du vecteur d'état $x$ comme une mise à jour courante à partir de la nouvelle mesure seulement. Ce filtre sert également à faire de la prédiction dans la mesure où on peut calculer l'estimateur à priori de $x$.

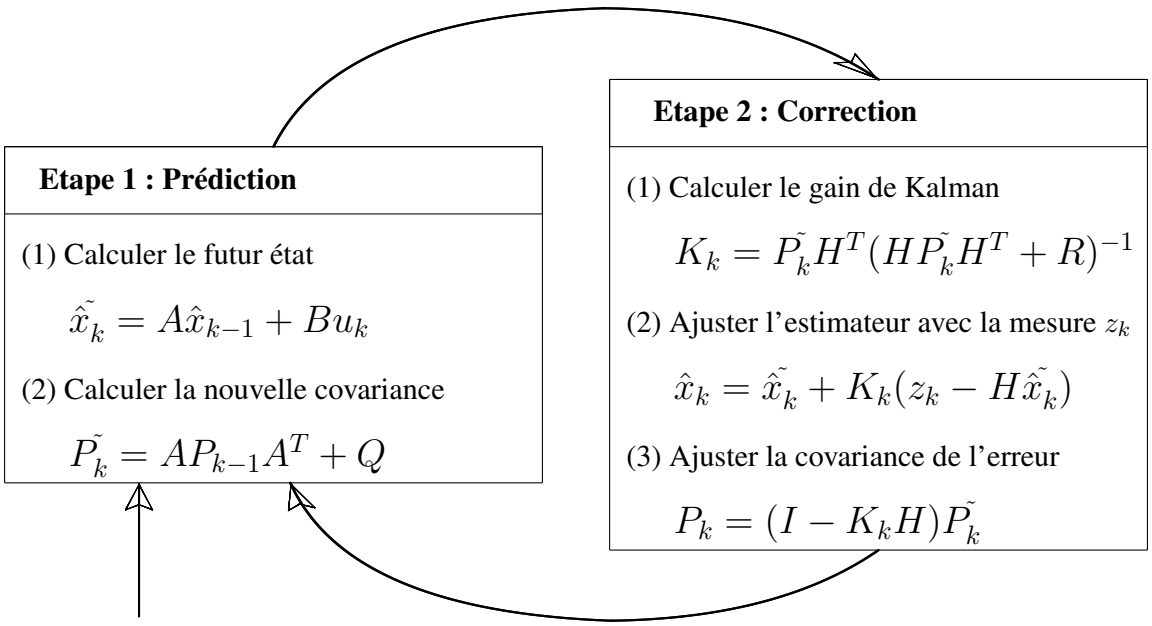



FIG. 5.3 – L'algorithme du filtre de Kalman linéaire

Le filtre est régit par un ensemble d'équations qui se divisent en deux groupes : les équations de prédiction et les équations de correction. La figure 5.3 page suivante montre le fonctionnement du filtre.

L'étape 0 constitue l'initialisation des paramètres du filtre avant de commencer à faire de la prédiction. Cette étape ne s'exécute qu'une seule fois.

L'étape 1 se charge d'estimer le futur état du système, connaissant l'état précédent.

L'étape 2 estime l'état réel du système à partir du vecteur de mesures.

On passe de l'étape 1 à l'étape 2 et inversement aussi longtemps que l'on a besoin d'estimer l'état du système.

### 5.3.2 Le filtre de Kalman étendu

Le filtre que nous avons présenté est utilisable dans le cas d'une équation stochastique linéaire uniquement. Heureusement, il est possible en modifiant ce filtre de l'appliquer à des cas non linéaires (lorsque l'équation d'état ou de mesure introduit des non-linéarités) : il s'agit du filtre de Kalman étendu. $x_k$ et $z_k$ sont alors régis par des lois de la forme :

$$x_k = f(x_{k-1}, u_k, v_{k-1}) \quad (5.7)$$
$$z_k = h(x_k, e_k) \quad (5.8)$$

Où $f$ et $h$ désignent des fonctions non linéaires. L'étape suivante consiste à linéariser ces équations ce qui nous donne :

$$x_k \approx \tilde{x}_k + A(x_{k-1} - \hat{x}_{k-1}) + Vv_{k-1} \quad (5.9)$$
$$z_k \approx \tilde{z}_k + H(x_k - \hat{x}_k) + Ee_k \quad (5.10)$$

Avec A, W, H et V qui sont des matrices Jacobiennes définies par :

$$A_{[i,j]} = \frac{\partial f_{[i]}}{\partial x_{[j]}}(\hat{x}_k, u_k, 0) \quad (5.11)$$
$$V_{[i,j]} = \frac{\partial f_{[i]}}{\partial v_{[j]}}(\hat{x}_k, u_k, 0) \quad (5.12)$$

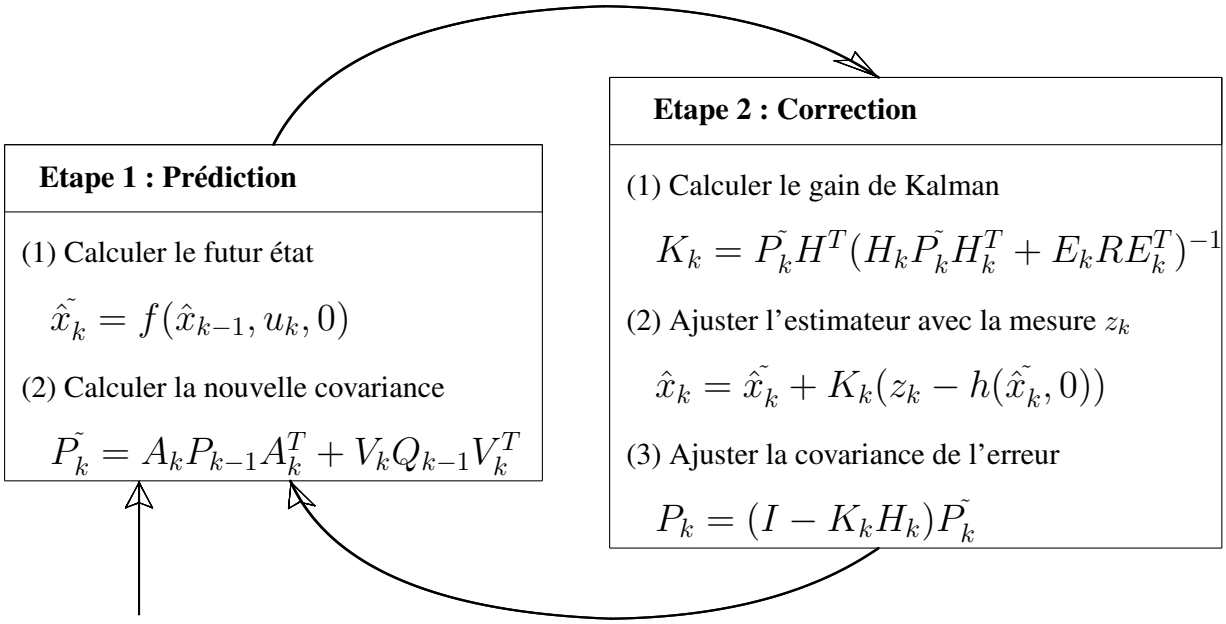



FIG. 5.4 – L'algorithme du filtre de Kalman étendu

$$H_{[i,j]} = \frac{\partial h_{[i]}}{\partial x_{[j]}}(\tilde{x}_k, 0) \tag{5.13}$$

$$E_{[i,j]} = \frac{\partial h_{[i]}}{\partial e_{[j]}}(\tilde{x}_k, 0) \tag{5.14}$$

On peut donc définir l'algorithme du filtre de Kalman étendu. Les étapes sont les mêmes que pour le filtre de Kalman linéaire. Le changement se situe dans les formules employées. La figure 5.4 présente le fonctionnement de ce filtre.

Plus récemment, un autre algorithme de filtrage a fait son apparition dans les problèmes de suivi en temps d'objet par le biais des ondes radars par exemple [Gordon et al., 1993]. Il s'agit du filtre particulaire dont nous allons exposer le fonctionnement dans le but d'établir ultérieurement une comparaison avec le filtre de Kalman.

### 5.3.3 Le filtre particulaire

Un filtre est alors généralement utilisé pour fusionner les données issues d'un système de localisation et celles issues de la centrale inertielle. La démarche classique est d'employer un filtre de Kalman étendu à cet effet [Chai et al., 1999]. Ce filtre se base sur le principe de la linéarisation des équations de mesure et d'état en employant les séries de Taylor. Mais la représentation sous forme de séries peut mener à une représentation dégradée des non-linéarités des fonctions concernées. De plus, lorsque l'on se trouve confronté à des modèles non-linéaires dans l'équation d'état et l'équation de mesure du filtre ou lorsque le bruit est supposé non gaussien, le filtre de Kalman peut calculer une solution non optimale. De plus, l'étape de linéarisation peut, sous certaines conditions, poser des problèmes d'instabilité ou de convergence du filtre de Kalman étendu.

Les méthodes séquentielles de Monte-Carlo [Doucet et al., 2001] [Gordon et al., 1993], appelées aussi filtres particulaires fournissent une solution générale aux problèmes où la linéarisation et le modèle de bruit gaussien mènent à de piètres performances car ces hypothèse ne traduisent pas assez fidèlement le processus étudié.

Dans cette section, nous allons étudier l'application d'un filtre particulaire classique de type Bayesian bootstrap [Gordon, 1997]. Cette méthode permet une représentation complète de la distribution à

posteriori des états, ce qui facilite le calcul de n'importe quelle estimation statistique. De plus, il peut traiter de manière efficace avec n'importe quelle non linéarité ainsi que n'importe quelle distribution statistique.

Notre but est de comparer, tant en simulation que sur une tâche réelle, les résultats obtenus par un estimateur à base de filtre particulaire par rapport à ceux donnés par un estimateur employant le filtre de Kalman étendu. Des mouvements typiques de la tête sont considérés et l'erreur de prédiction des deux estimateurs sera examinée (voir section 5.4 page 152).

Le filtrage particulaire est un domaine d'investigation qui a récemment progressé grâce à l'augmentation de la puissance de calcul des ordinateurs. Ces méthodes ont littéralement connues une renaissance après l'article fondateur de Gordon [Gordon, 1997], qui montrait que les méthodes de filtrages particulaires pouvaient être utilisées de manière pratique pour résoudre le problème de l'estimation optimale. Les équations d'état et de mesure sont alors présentées sous la forme :

$$x_{k+1} = f(x_{k-1}, v_k) \tag{5.15}$$
$$z_k = h(x_k) + e_k \tag{5.16}$$

Le filtre particulaire fournit une solution qui est une approximation bayésienne au problème récursif discrétisé en temps en mettant à jour une description approximative de la densité probabiliste du vecteur d'état postérieure au filtrage. Nous notons $x_k \in \mathcal{R}^N$ l'état du système observé et $Z_k = \{y_i\}_{i=0}^t$ l'ensemble des observations réalisées jusqu'au temps présent. En supposant que le bruit de processus $v_k$ est indépendant du vecteur d'état et que le bruit de mesure est $e_k$ avec pour densités probabilistes respectives $p_{v_k}$ et $p_{e_k}$. L'incertitude initiale sur le vecteur d'état $x_0$ est décrite par la densité $p_{x_0}$. Le filtre particulaire fait l'approximation de la densité probabiliste $p(x_k|Z_k)$ par un ensemble important de $N$ particules noté $\{x_k^{(i)}\}_{i=1}^N$ où chaque particule est pondérée par une valeur $w_k^{(i)}$ de manière à ce que la somme globale soit égale à un.

L'emplacement et le poids de chaque particule reflète la valeur de la densité probabiliste du vecteur d'état de la portion de l'espace considéré. Le filtre particulaire met à jour la localisation des particules et leur poids correspondant de manière récursive à chaque nouvelle observation effectuée. La prédiction non linéaire de la densité $p(x_k|Z_{k+1})$ et la densité de filtrage $p(x_k|Z_k)$ sont données par :

$$p(x_k|Z_{k-1}) = \int_{\mathcal{R}^N} p(x_k|x_{k-1})p(x_{k-1}|Z_{k-1})dx_{k-1} \tag{5.17}$$
$$P(x_k|Z_{k-1}) \propto P(z_k|x_k)P(x_k|Z_{k-1}) \tag{5.18}$$

L'idée est donc d'approximer $P(x_k|Z_{k-1})$ par :

$$P(x_k|Z_{k-1}) \approx \frac{1}{N}\sum_{i=1}^{N} \delta(x_k - x_k^{(i)}) \tag{5.19}$$

où $\delta$ est une fonction discrète de Dirac. Si l'on insère l'équation 5.19 dans l'équation 5.18, nous simplifions l'écriture de la densité. Ceci peut-être effectué en utilisant le bootstrap Bayesien appelé aussi algorithme de rééchantillonnage par importance dont les étapes sont les suivantes :

---

1. Générer, pour $k = O$, $N$ échantillons $\{x_0^{(i)}\}_{i=1}^N$ de la distribution initiale $P(x_0)$.
2. Calculer les poids $w_k^{(i)} = p(z_k|x_k^{(i)})$ et renormaliser en appliquant $\widetilde{w}_k^{(i)} = w_k^{(i)} / \sum_{j=1}^N w_k^{(j)}, i = 1...N$
3. Générer un nouvel ensemble $\{x_k^{(i*)}\}_{i=1}^N$ par rééchantillonnage en remplaçant $N$ fois les particules de $\{x_0^{(i)}\}_{i=1}^N$ où $P_r(x_k^{(i*)} = x_k^{(j)}) = \widetilde{w}_k^{(j)}$.

4. Prédire l'évolution des particules : $x_{k+1}^{(i)} = f(x_k^{(i*)}, v_k), i = 1, ...., N$ en leur affectant à chacune un niveau de bruit particulier suivant la densité $p(v_k)$
5. Augmenter $k$ et réitérer à partir de l'étape 2.

---

Ce modèle décrit l'évolution des particules. Lorsque nous voulons estimer la valeur fournie par ce filtre, il nous suffit de calculer la grandeur :

$$\hat{x}_k^N = \sum_{i=1}^{N} w_k^{(i)} x_k^{(i)} \tag{5.20}$$

Le filtre particulaire se décompose donc en deux parties : une partie "évolution" ou "prédiction" des particules reposant sur l'équation d'état et une autre qui est associée, au regard de la mesure, à une pondération bayésienne à chaque particule. À un fort coefficient de pondération associé correspond une particule représentant un état fortement vraisemblable.

Maintenant que nous avons exposé le fonctionnement des filtres de Kalman et du filtre particulaire, nous pouvons voir comment nous allons les appliquer aux mouvements de la tête de l'opérateur. Pour cela, nous allons nous attacher à décrire l'équation de mesure et l'équation de prédiction qui seront communes aux deux filtres et qui déterminent l'application de ces derniers à notre problème.

### 5.3.4 Modélisation du mouvement de la tête et dynamique du système

Nous allons employer pour notre système de réalité augmentée en vision directe un ensemble de capteurs hybrides composés d'un capteur de position et d'orientation (le système de localisation par la vision détaillé dans le chapitre 3.4 page 95) et une centrale inertielle qui fournit les accélérations linéaires et les vitesses angulaires de rotation. Les différents repères utilisés dans notre étude, comme nous pouvons le voir à la figure 5.5, sont $\{W\}$, le repère de référence associé à l'espace de travail, $\{C\}$ le repère associé à la caméra et $\{I\}$ le repère associé à la centrale inertielle. Comme la centrale inertielle est supposée liée à la caméra de manière rigide, la transformation entre les repères liés à ces capteurs est supposée connue.

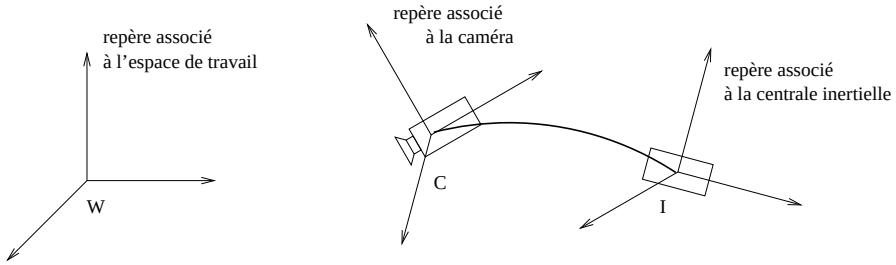



FIG. 5.5 – Repères liés au différents systèmes de coordonnées employés par les capteurs

Nous utilisons, dans le cadre de notre étude, le modèle de mouvement de la tête proposé par Chai et al [Chai et al., 2002][Chai et al., 1999], où l'état décrivant le mouvement de la tête à un instant donné est représenté par un vecteur de taille $15 \times 1$ :

$$X_{head} = (\theta, \omega, x, \dot{x}, \ddot{x})$$

où $\theta$ est l'orientation du repère caméra par rapport au repère monde en utilisant les angles d'Euler, $\omega$ est la vitesse angulaire et $x, \dot{x}, \ddot{x}$ sont les positions, les vitesses et les accélérations linéaires du repère lié à la caméra par rapport au repère monde. En fonction de ce vecteur d'état, l'équation discrétisée

en temps de l'évolution de ce modèle est donnée par :

$$\begin{bmatrix} \theta_{k+1} \\ \omega_{k+1} \\ x_{k+1} \\ \overset{\bullet}{x}_{k+1} \\ \overset{\bullet\bullet}{x}_{k+1} \end{bmatrix} = \begin{bmatrix} \theta_k + \Delta t.W(\theta).\omega_k + v_k^1 \\ \omega_k + v_k^2 \\ x_k + \Delta t.\ \overset{\bullet}{x}_k + \frac{1}{2}\Delta t^2.\ \overset{\bullet\bullet}{x}_k + v_k^3 \\ \overset{\bullet}{x}_k + \Delta t.\ \overset{\bullet\bullet}{x}_k + v_k^4 \\ \overset{\bullet\bullet}{x}_k + v_k^5 \end{bmatrix} \tag{5.21}$$

Dans cette équation $\Delta t$ est la période d'échantillonnage entre deux mesures, $W(\theta)$ est la matrice de rotation liée à la vitesse angulaire et les $v_k^i, i \in [1..5]$ sont les distributions modélisant l'incertitude de l'état. Aux capteurs sont associées des équations de mesure qui convertissent les données fournies par ces derniers dans les mêmes grandeurs que celles utilisées dans l'équation d'état. Puisque l'objet de cette étude est la comparaison des performances d'un filtre particulaire avec un estimateur de Kalman étendu, nous allons supposer que les capteurs mesurent directement les mêmes grandeurs que celles de l'équation d'état. Le système de vision nous donnera la pose de la caméra $(x_m, \theta_m)$dans le repère associé à l'espace de travail. La centrale inertielle donnera les vitesses angulaires $(\omega_m)$ par le biais de ses gyroscopes et les accélérations $(\overset{\bullet\bullet}{x}_m)$ par le biais des accéléromètres. Ces données issues des capteurs peuvent être représentées par le vecteur de mesure suivant :

$$z_m = [\theta_m, \omega_m, x_m, \overset{\bullet\bullet}{x}_m] \tag{5.22}$$

L'équation de mesures sera donc :

$$Z_k = H.X_k + e_k \tag{5.23}$$

Avec

$$H = \begin{bmatrix} I_{3\times3} & 0 & 0 & 0 & 0 \\ 0 & I_{3\times3} & 0 & 0 & 0 \\ 0 & 0 & I_{3\times3} & 0 & 0 \\ 0 & 0 & 0 & I_{3\times3} & 0 \end{bmatrix} \tag{5.24}$$

$I_{3\times3}$ étant la matrice identité de taille $3 \times 3$ et $e_k$ traduisant l'incertitude de la mesure par rapport aux bruits.

Nos équations d'état et de mesure étant définies, nous pouvons appliquer les filtres particulaires ou de Kalman étendu pour estimer l'orientation et la position de la tête de l'opérateur grâce aux mesures fournies par le système de localisation par la vision couplé à une centrale inertielle. Nous allons à présent exposer les résultats obtenus en appliquant les deux algorithmes de filtrage sur les valeurs d'une simulation que nous avons effectuée. Nous en conclurons le type de filtrage nous devons employer.

## 5.4 Application du filtre particulaire à la prédiction du mouvement de la tête

A notre connaissance, le filtre particulaire n'avait pas été employé dans le cadre de la prédiction du mouvement de la tête. Nous avons voulu voir quel était le comportement d'un tel filtre face à une base de comparaison à savoir le filtre de Kalman étendu qui est classiquement utilisé dans le cadre de la prédiction du point de vue en réalité augmentée [Ababsa et al., 2003].

Afin d'évaluer les performances de notre filtre particulaire, nous l'avons d'abord essayé sur un jeu de données en simulation. Une position et une orientation virtuelle de la caméra est décrite (voir

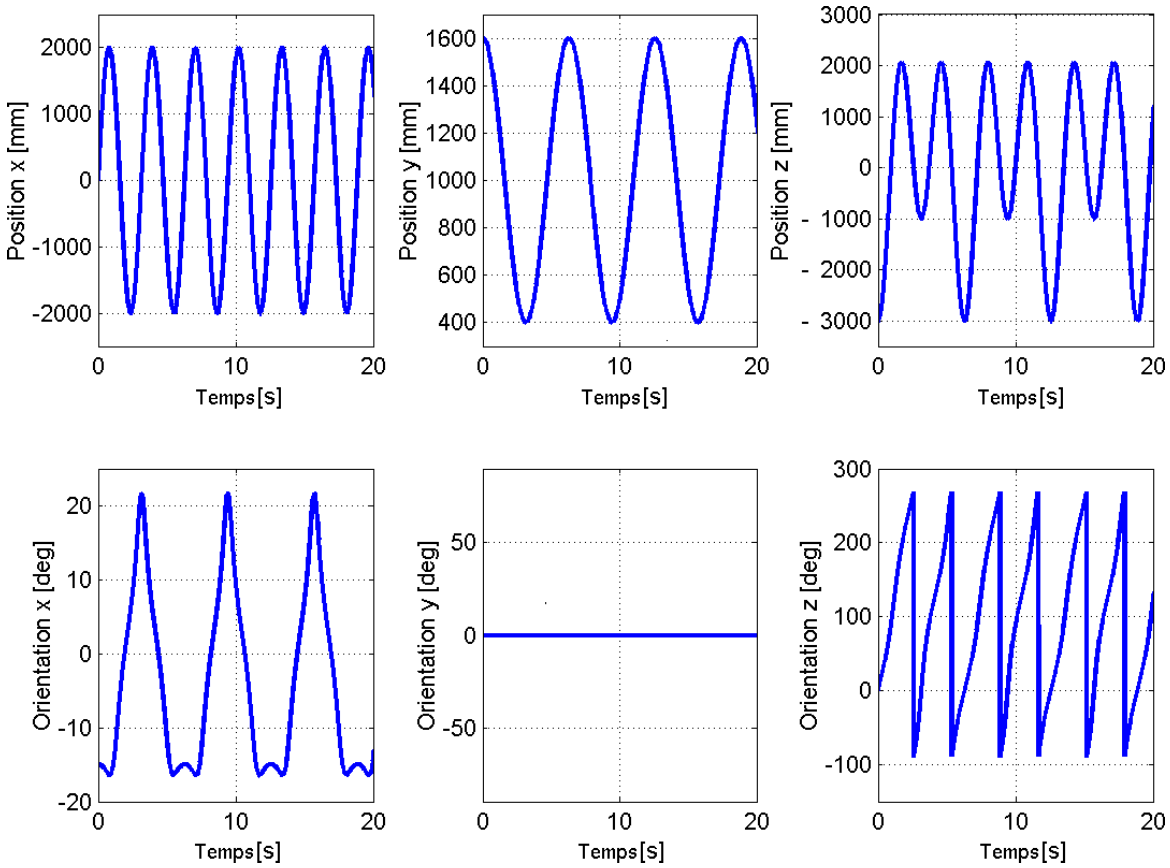



FIG. 5.6 – Jeu de données virtuelles servant à la simulation

figure 5.6 page suivante), de même pour les données issues des accéléromètres et des gyroscopes. A ces mesures, nous ajoutons un bruit blanc gaussien d'une variance de 0.01 $rad/s$ pour les vitesses angulaires, 1300 $rad/s^2$ pour les accéléromètres et 0.016 $m$ pour le système de localisation par la vision, en se basant sur les spécifications techniques de chaque capteur. De plus, le bruit de mesure et l'incertitude des états, respectivement, $v_k$ et $e_k$ sont considérés comme étant gaussiens, c'est à dire que $v_k \in N(0, Q)$ et $e_k \in N(0, R)$, $N$ étant une distribution normale et $Q$ et $R$ étant des matrices de covariance. Dans notre expérience, nous avons fixé le pas d'échantillonnage à $\Delta t = 15.6ms$ ce qui donne une fréquence de 60 Hz environ.

Les matrices de covariances du filtres sont choisies empiriquement pour optimiser les performances du filtre. Ceci concerne $R$ la matrice de covariance sur le bruit de mesure, $Q$ la matrice de covariance décrivant l'incertitude sur la valeur d'un état interne et la matrice d'erreur sur la première estimation de la pose $P_0$. Nous leur attribuons les valeurs suivantes :

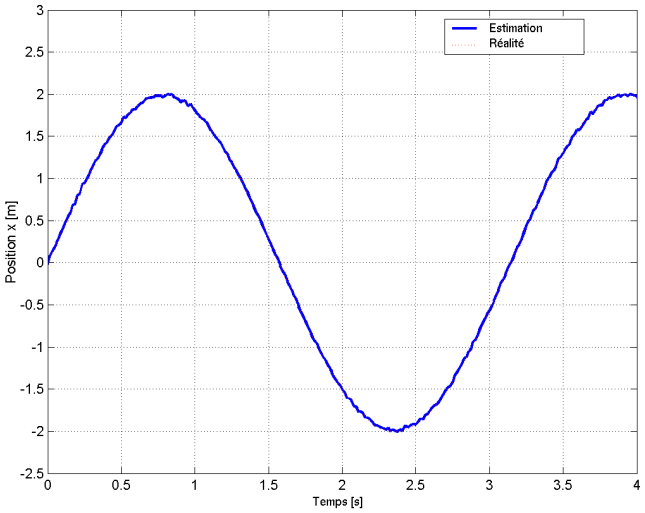
$$P_0 = diag([\ 100I_{3\times3} \quad 10I_{3\times3} \quad 100I_{3\times3} \quad 10^4I_{3\times3} \quad 10^6I_{3\times3}\ ]) \tag{5.25}$$

$$Q = diag([\ 0.01I_{3\times3} \quad 0.1I_{3\times3} \quad 10I_{3\times3} \quad 10^5I_{3\times3} \quad 10^6I_{3\times3}\ ]) \tag{5.26}$$

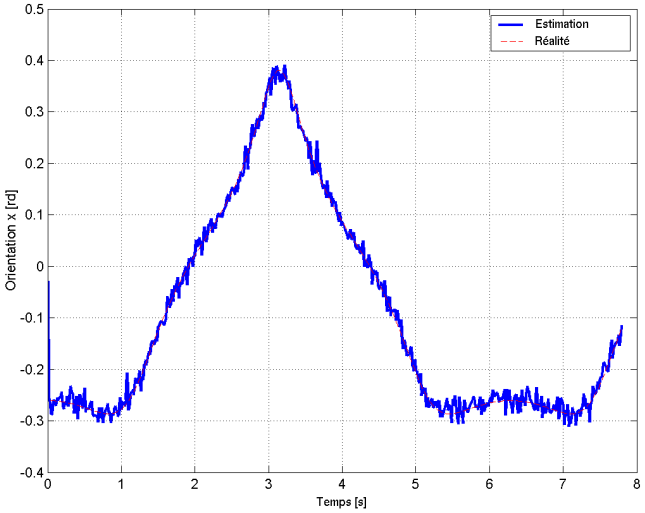
$$R = diag([\ 0.001I_{3\times3} \quad 0.01I_{3\times3} \quad I_{3\times3} \quad 100I_{3\times3} \quad 10^4I_{3\times3}\ ]) \tag{5.27}$$

Dans un premier temps, nous comparons le filtre particulaire (appelé aussi Sampling Importance Resampling ou SIR) avec le filtre de Kalman étendu (Extended Kalman Filter - EKF) en employant les équations 5.21 et 5.23 page ci-contre. Nous n'incluons pas ici les équations du filtre de Kalman étendu puisque ce dernier a déjà fait l'objet d'une présentation. Dans la figure 5.7 page suivante sont représentés les résultats de la prédiction du mouvement de la tête à l'aide du filtre particulaire. La position réelle et l'orientation sont représentées par des lignes en pointillés alors que l'estimation est représentée par des lignes continues.

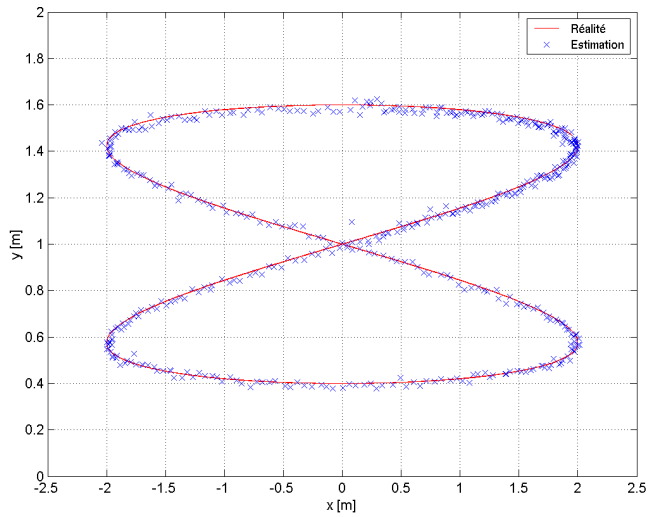
Le figure 5.7(a) représente la position estimée suivant la coordonnée x du repère monde et la figure 5.7(b) fait de même pour l'orientation autour de ce même axe. Le filtre utilisé ici comporte $N = 100$




(a) Position x

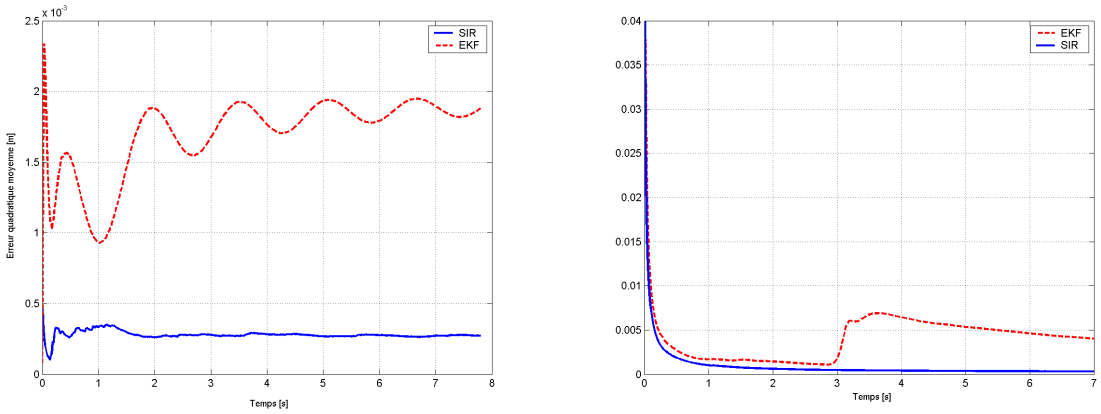



(b) Orientation x




(c) Vue de la trajectoire dans l'espace dans le repère monde

FIG. 5.7 – Estimation de la pose de la caméra par le filtre particulaire par rapport à la pose théorique




(a) Position x (b) Orientation x

FIG. 5.8 – Erreur quadratique moyenne de prédiction pour le filtre particulaire (SIR) et le filtre de Kalman étendu

| Estimation | Translation (m) | Orientation (rad) | Orientation ($^o$) |
|---|---|---|---|
| Particulaire | 0.01642 | 0.0003 | 0.0171 |
| Kalman étendu | 0.01986 | 0.0049 | 0.2807 |

TAB. 5.1 – Erreur quadratique moyenne pour le filtre particulaire et le filtre de Kalman étendu

particules. Il apparaît que le filtre particulaire réussit à suivre fidèlement la trajectoire, ce qui implique un modèle de prédiction assez robuste. La figure 5.7(c) représente la trajectoire simulée dans le repère monde projetée sur le plan $XY$. La ligne continue donne la représentation réelle alors que les croix représentent les positions prédites.

Afin d'évaluer les performances du filtre, nous calculons l'erreur quadratique moyenne entre les positions réelles et les positions estimées. Cette erreur est donnée par l'équation :

$$\epsilon(n) \quad = \quad \frac{1}{n}\sum_{k=1}^{n} \|x(k) - \hat{x}(k)\|^2 \tag{5.28}$$

Où $x(k)$ est un des points de la trajectoire réelle et $\hat{x}(k)$ est le même point de la trajectoire lorsqu'il est prédit.

A la figure 5.8, les erreurs quadratiques moyennes au cours du temps sont représentées pour le filtre particulaire et le filtre de Kalman étendu. Les figures 5.8(a) et 5.8(b) montrent l'erreur quadratique moyenne de la position et de l'orientation de la caméra suivant l'axe x du repère monde. Nous notons que, dans tous les cas, avec des conditions d'application similaires, le filtre particulaire donne une meilleure estimation que le filtre de Kalman étendu alors que nous sommes ici dans des hypothèses favorables à ce dernier (bruit gaussien).

Dans le tableau 5.1, nous avons résumé les résultats obtenus en moyenne par les deux méthodes :

Nous avons ensuite étudié l'influence du nombre de particules sur les performances du filtre. Ces résultats sont présentés à la figure 5.9 page suivante. Comme nous pouvions nous y attendre, plus le nombre de particules augmente, plus l'erreur quadratique moyenne diminue. Toutefois, plus nous calculons de résultats sur un ensemble de particules, plus le temps de calcul pour établir une estimation va linéairement augmenter. Il est donc essentiel de trouver le nombre de particules optimal à employer en fonction du problème posé. Ce nombre doit être adapté aux contraintes liées au temps réel tout en assurant les meilleures performances possibles. Nous avons donc effectué plusieurs expériences en faisant varier le nombre de particules. La valeur optimale est d'environ une centaine de particules pour le problème posé.

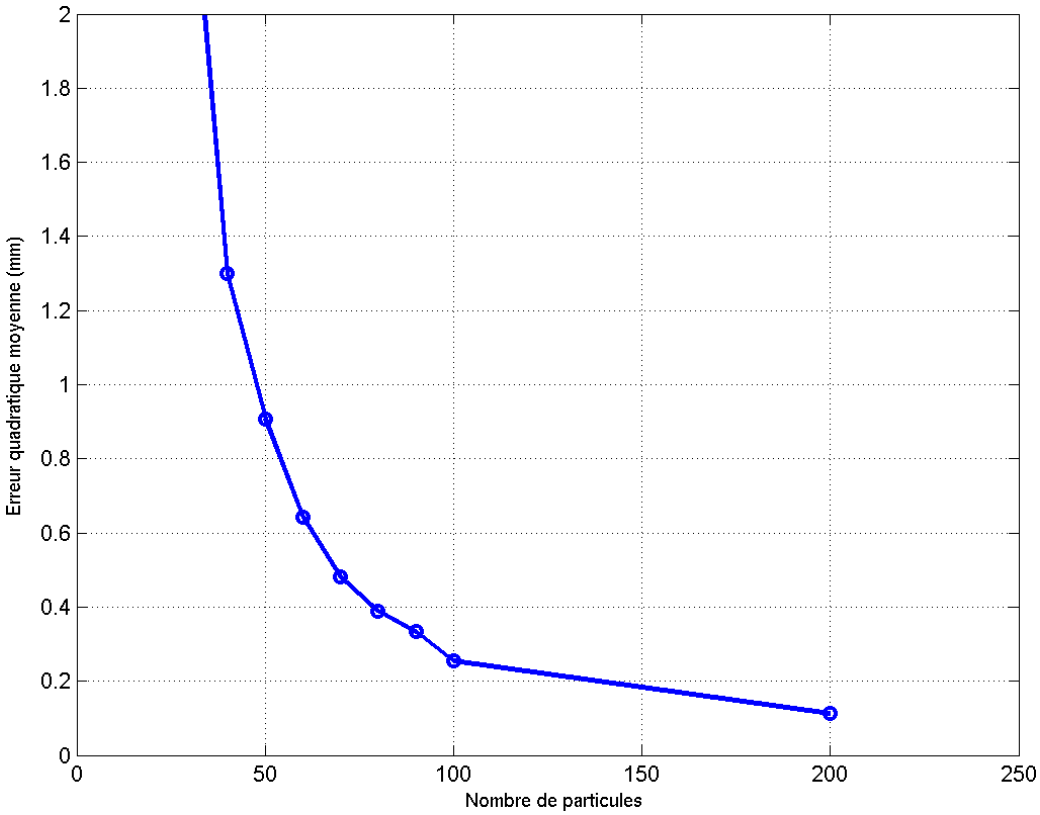


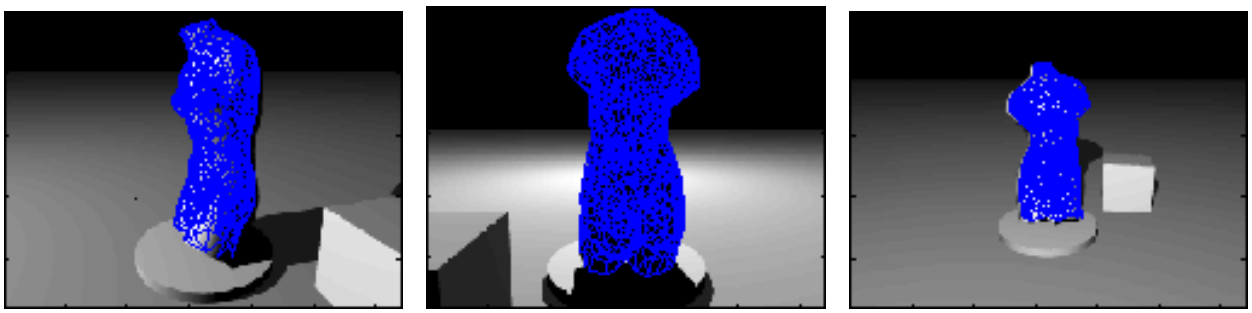

FIG. 5.9 – Effet du nombre de particules sur les performances du filtre

Si nous voulons résumer et comparer les deux filtres, nous pouvons récapituler leurs caractéristiques par rapport à divers critères de comparaison comme nous l'avons fait dans le tableau 5.2 page suivante. Ainsi, même si le filtre de Kalman étendu est en théorie optimal et son exécution rapide, il n'en demeure pas moins que le filtre particulaire reste plus pratique à mettre en place, puisqu'il est robuste par rapport aux conditions initiales, sa vitesse de convergence vers la solution est supérieure et il permet de modéliser les non-linéarités qui peuvent intervenir dans les équations du filtre.

| Filtre | Kalman étendu | Particulaire |
|---|---|---|
| Théorie | Optimal | Sous-optimal |
| Initialisation | Sensible | Robuste |
| Temps de calcul | Rapide | Dépend du nombre de particules |
| Convergence | + | ++ |
| Modèle de bruit | Blanc gaussien | libre |
| Non-linéarité | Linéarisation obligatoire | oui |

TAB. 5.2 – Comparaison du filtre de Kalman étendu et du filtre particulaire

Dans le but de contrôler l'efficacité de notre approche, nous avons simulé un système de réalité augmentée qui emploierait notre système de filtre particulaire. Nous avons généré une série d'images de synthèses prises par une caméra en mouvement. Pendant ce mouvement, la caméra observe un buste de statue (la *Vénus*) qui sera notre objet de référence. La *Vénus* est immobile par rapport au repère monde. Quand la pose de la caméra est prédite par le filtre particulaire, nous reprojetons le modèle filaire de la Vénus sur cette dernière, en tenant compte du retard que devrait avoir les mesures du capteur. La figure 5.10 page ci-contre montre le résultat visuel obtenu. Du point de vue qualitatif, les résultats donnés par le filtre particulaire permettent un recalage visuellement correct, mais qui reste perceptible. Ceci indique que nous pouvons améliorer la qualité du recalage. Pour se faire, nous allons coupler cette méthode de prédiction avec la méthode de *post-rendering* que nous avons développé.



(a) t = 0.78 s (b) t = 4.68 s (c) t = 6.4 s

FIG. 5.10 – Simulation du système de réalité augmentée

## 5.5 Mécanisme de double prédiction

Nous allons combiner la technique précédente de prédiction du point de vue de l'opérateur qui est l'approche classique en réalité augmentée avec une méthode de post-rendering qui est une approche employée en réalité virtuelle [Didier et al., 2004] [Didier et al., 2005a]. Dans le cas où nous couplons une centrale inertielle avec une caméra, il nous est possible d'obtenir des mesures de la centrale à une vitesse de 60 à 100Hz suivant le mode dans lequel nous l'utilisons. Si nous supposons que l'acquisition des données et le rendu de la scène s'effectuent à l'aide de processus exécutés en parallèle, il est donc possible d'acquérir un ou plusieurs jeux de données pendant que l'on effectue le rendu des augmentations virtuelles. Ceci veut dire que nous n'exploitons pas la totalité des données fournies par les capteurs pour effectuer la prédiction. Il est possible, en employant une méthode de post-rendering (à la condition que cette dernière ait un temps d'exécution assez bref en regard du temps nécessité par le rendu de la scène) d'effectuer une prédiction de dernière minute avec les données nouvellement acquises afin de transformer l'image rendue pour que le recalage soit encore amélioré (voir figure 5.11).

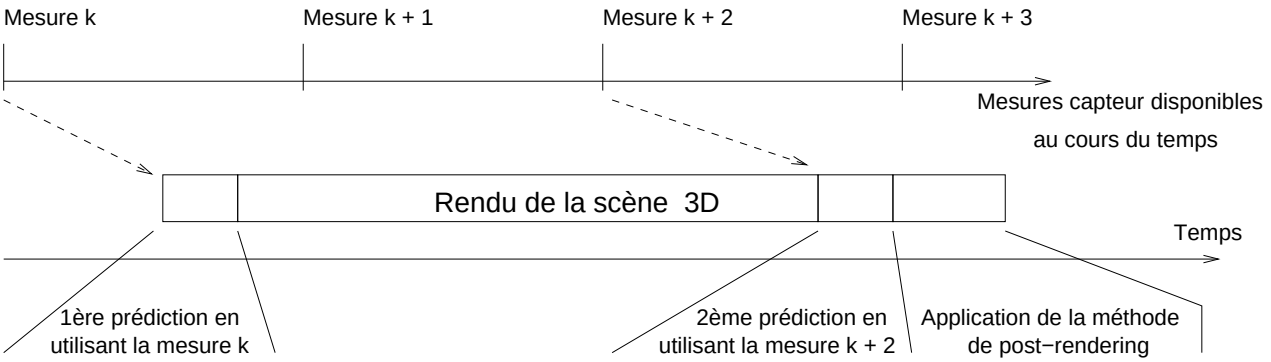



FIG. 5.11 – Chronogramme du mécanisme de double prédiction

Par la suite, nous verrons que la méthode proposée est :

- suffisamment rapide pour être employée en tant qu'algorithme de compensation de la latence,
- capable de compenser de manière idéale les mouvement de rotation,
- réduit les erreurs de recalage dans les conditions d'utilisation normales de cet algorithme, c'est à dire pour compenser la latence dans le cadre de mouvements effectués par un opérateur humain.

Nous allons décrire la méthode de manière générale, puis nous étudierons son comportement théorique avant d'implémenter cette dernière en simulation afin de mesurer plus précisément le gain apporté par cette méthode.

### 5.5.1 Méthode de post-rendering appliquée

Dans cette partie, nous supposons que la prédiction du point de vue et donc le recalage sont déjà effectués. Le modèle de caméra employé sera celui du sténopé. Nous allons commencer par introduire l'architecture de notre solution puis les notations employées pour modéliser le problème. Dans ce qui suit, nous aurons deux caméras, donc nous dupliquerons les notations employées pour chaque caméra.

#### 5.5.1.1 Algorithme général

L'idée principale de cette architecture est de pouvoir prendre en compte les mesures des capteurs qui seraient arrivées pendant que l'on effectue le rendu de la scène virtuelle.

Ceci peut-être effectué en quelques étapes seulement :

1. Faire une première prédiction du point de vue avec le dernier ensemble de mesures données par les capteurs,
2. Effectuer le rendu de la scène virtuelle selon le premier point de vue prédit. Ceci sera fait dans une image qui sera de dimensions plus grande que celle qui sera réellement affichée,
3. Si pendant le rendu, de nouvelles données sont acquises :
   (a) Faire une seconde prédiction avec le nouveau jeu de données,
   (b) Calculer la correction à effectuer par rapport à la première prédiction,
   (c) Appliquer la pseudo-correction à la scène virtuelle déjà rendue,
4. Afficher l'image finale.

Cette méthode n'a bien sûr d'intérêt que si la méthode utilisée pour corriger l'image s'effectue en un temps relativement court comparativement au rendu de la scène.

#### 5.5.1.2 Architecture du 'pipeline' graphique

Graphiquement, l'architecture d'un tel chemin de traitement des données peut-être synthétisé de la manière suivante :

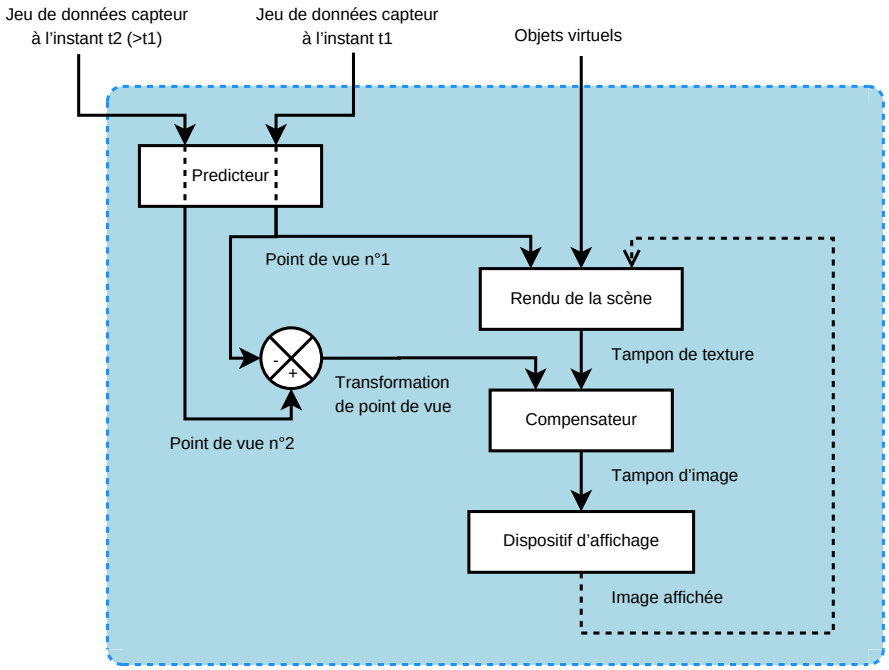



FIG. 5.12 – Architecture du 'pipeline' graphique

Une telle architecture comporte :

– un Prédicteur qui va donner la position et l'orientation du point de vue pour un instant donné, en fonction des dernières données acquises par les capteurs,
– un "Scene Renderer" qui va calculer les images en fonction du point de vue prédit précédemment et des augmentations qui seront affichées,
– un Compensateur, qui utilise la méthode que nous allons présenter dans les paragraphes suivants. Ses entrées sont l'image déjà calculée et la correction de point de vue qui a été effectuée.
– un Dispositif d'affichage, qui affichera l'image finale obtenue après compensation.

Nous allons à présent détailler le fonctionnement du Compensateur.

#### 5.5.1.3 Algorithme de compensation

L'idée principale est la suivante : Chaque point de vue prédit définit une pyramide de vue dans le monde réel dont le sommet serait la position de ce dernier. Sur la première pyramide, chaque plan de vue perpendiculaire à la direction du regard est une projection d'une partie du monde réel tel que perçu de ce point de vue. Il s'agit en réalité d'un plan image. Le second point de vue va définir une deuxième pyramide. L'intersection entre la deuxième pyramide et un plan image de la première va nous donner un quadrilatère comme le montre la figure 5.13.

Nous allons employer les notations suivantes pour la suite :
– $\mathcal{R}_1 = (C_1, \vec{x_{c1}}, \vec{y_{c1}}, \vec{z_{c1}})$ est le repère lié au premier point de vue prédit,
– $\mathcal{R}_2 = (C_2, \vec{x_{c2}}, \vec{y_{c2}}, \vec{z_{c2}})$ est le repère lié au second point de vue prédit,
– $\mathcal{R}_I = (I, \vec{x_i}, \vec{y_i}, \vec{z_i})$ est le repère lié au plan image $\mathcal{P}_1$,
– $\mathcal{P}_1$ est un plan image attaché à la première pyramide situé à une distance $\delta$ de son sommet $C_1$ dans la direction $-\vec{z_{C_1}}$. $\delta$ est homogène à une distance focale que nous pouvons choisir de manière arbitraire. Nous discuterons plus tard sur le choix de la valeur de $\delta$.
– $\Delta\mathcal{T}$ est la transformation qui permet de passer du repère $\mathcal{R}_1$ au repère $\mathcal{R}_2$.

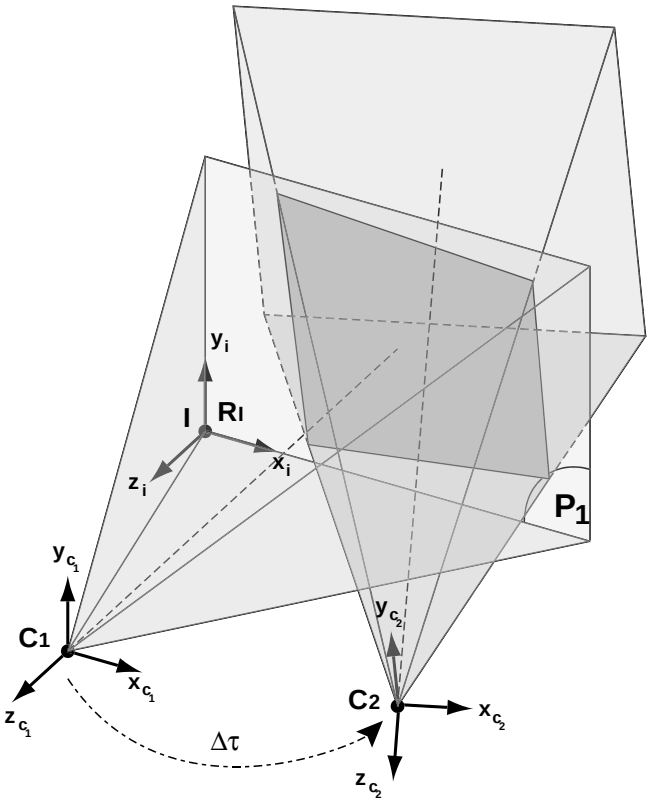



FIG. 5.13 – Pyramides de vision et notations

Il nous suffit de considérer la première image générée comme étant une image du plan $\mathcal{P}_1$. Nous

avons alors à appliquer la transformation appropriée pour afficher ce plan comme s'il était vu par le point de vue $C_2$.

Cette méthode va compenser les trois degrés de libertés liés aux mouvements de rotation et va également améliorer (de manière imparfaite) le recalage en translation. Il convient dès lors d'étudier les gains réels apportés par cette méthode.

### 5.5.2 Erreur théorique introduite par cette méthode

Dans le cadre de cette étude, prenons un sommet v de l'objet à représenter dont les coordonnées exprimées dans le repère caméra ($\mathcal{R}_1$) seront $(x, y, z)$.

La méthode décrite ci-dessus est équivalente à projeter les sommets des objets de l'espace caméra sur un plan, d'appliquer à ce dernier la transformation $\Delta\mathcal{T}$ et enfin de le reprojeter sur un plan image attaché à la seconde pyramide ($\mathcal{P}_2$) comme nous pouvons le voir à la figure 5.13.

$\Delta\mathcal{T}$ peut-être représenté sous la forme d'une matrice de transformation appliquée sur des coordonnées homogènes :

$$\Delta\mathcal{T} = \begin{bmatrix} R_{3\times3} & T_{3\times1} \\ 0_{1\times3} & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.29}$$

Nous définissons aussi $r_k, k \in \{1, 2, 3\}$ comme les vecteurs lignes $(r_{k1}, r_{k2}, r_{k3})$. Nous devons donc évaluer et calculer la différence des deux expressions suivantes : $P_2\Delta\mathcal{T}P_1\mathrm{v}$ (qui représente les coordonnées dans le plan image $P_2$ du sommet $v$ en appliquant notre méthode de post-rendering) et $P_2\Delta\mathcal{T}\mathrm{v}$ (qui représente les coordonnées réelles du sommet $v$ dans le plan image $P_2$) où $P_2$ est une matrice représentant les paramètres intrinsèques d'une caméra et $P_1$ une matrice représentant une projection perspective sur un plan. Il en découle :

$$P_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/\delta & 0 \end{bmatrix}, P_2 = \begin{bmatrix} \alpha_u & 0 & u_0 & 0 \\ 0 & \alpha_v & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Avec $\delta$, la distance entre l'origine du repère et le plan de projection et $\alpha_u$, $u_0$, $\alpha_v$, $v_0$ les paramètres intrinsèques de la caméra décrivant la projection dans un modèle standard de sténopé linéaire. [Tsai, 1987].

Après quelques calculs selon ces équations, nous parvenons à l'expression de l'erreur en $pixel^2$ pour chaque sommet de l'objet projeté dans l'espace caméra entre une vraie projection et notre méthode de post-rendering :

$$\epsilon(\mathrm{v}) = \alpha_u^2 \left[ \frac{r_1 \bullet \mathrm{v} + t_1}{r_3 \bullet \mathrm{v} + t_3} - \frac{r_1 \bullet \mathrm{v}' + t_1}{r_3 \bullet \mathrm{v}' + t_3} \right]^2 +$$

$$\alpha_v^2 \left[ \frac{r_2 \bullet \mathrm{v} + t_2}{r_3 \bullet \mathrm{v} + t_3} - \frac{r_2 \bullet \mathrm{v}' + t_2}{r_3 \bullet \mathrm{v}' + t_3} \right]^2 \tag{5.30}$$

Où $\mathrm{v}' = \frac{\delta}{z}\mathrm{v}$ et '$\bullet$' est le produit scalaire de deux vecteurs.

A présent, nous allons regarder les conséquences que l'introduction de cette erreur a sur certains types de mouvements :

#### 5.5.2.1 Les mouvements de rotation pure

Nous remarquons que si $t_1$, $t_2$, $t_3$ sont tous égaux à $0$, alors l'erreur en $pixel^2$ $\epsilon(\mathrm{v})$ est aussi égale à $0$. Ceci signifie que *lorsqu'il n'y a pas de correction à effectuer en translation, alors cette méthode n'introduit aucune erreur*. Elle est donc envisageable pour compenser des mouvements de rotation pure. Ce résultat est intéressant puisque ce sont précisément les mouvements de rotation qui contribuent le plus aux erreurs de recalage entre les objets réels et les entités virtuelles [Liang et al., 1991].

Nous allons voir à présent ce qu'il en est pour la compensation de la translation et surtout comment s'assurer que la compensation introduite ne déforme pas trop les objets représentés. Ceci débouche sur une discussion autour de la valeur du paramètre $\delta$.

#### 5.5.2.2 Du choix du paramètre $\delta$

Comme nous avons pu le constater, l'expression de l'erreur $\epsilon(\mathrm{v})$ dépend de la valeur du paramètre $\delta$. Le problème du choix de cette valeur se pose donc, si l'on veut minimiser l'erreur de recalage. Une réécriture de l'équation 5.30 mène à l'expression :

$$\epsilon(\mathrm{v}) \quad = \quad K(\mathrm{v},\delta)[\alpha_u^2 A(\mathrm{v})^2 + \alpha_v^2 B(\mathrm{v})^2] \tag{5.31}$$

avec

$$K(\mathrm{v},\delta) = \left(1 - \frac{\delta}{z}\right)^2 \frac{1}{(r_3 \bullet \mathrm{v} + t_3)(r_3 \bullet \mathrm{v}' + t_3)} \tag{5.32}$$

$$A(\mathrm{v}) = t_3(r_1 \bullet \mathrm{v}) - t_1(r_3 \bullet \mathrm{v}) \tag{5.33}$$

$$B(\mathrm{v}) = t_3(r_2 \bullet \mathrm{v}) - t_2(r_3 \bullet \mathrm{v}) \tag{5.34}$$

Si nous regardons de près ces expressions, nous voyons que le seul terme dépendant de $\delta$ est $K(\mathrm{v},\delta)$. En réalité, l'erreur est nulle lorsque $(1-\delta/z)^2$ vaut $0$, en d'autres termes quand $\delta = z$. Ceci signifie que l'erreur est nulle lorsque le sommet $v$ est situé sur le plan $\mathcal{P}_1$ défini par le paramètre $\delta$.

Pour des raisons évidentes, tous les sommets d'un modèle virtuel ne peuvent se situer sur un unique plan perpendiculaire à la direction du regard d'une caméra. Une solution est de calculer la boîte englobante de la scène virtuelle à afficher et de calculer la distance entre le plan perpendiculaire à la direction $\vec{z_{C_1}}$ passant par le centre de la boîte englobante et la position du premier point de vue prédit. Cette distance sera notre valeur arbitraire pour le paramètre $\delta$ car elle représente une valeur moyenne pour ce paramètre.

Nous allons à présent étudier en simulation le comportement de cet algorithme.

### 5.5.3 Protocole expérimental

Tous d'abord, nous décrirons notre banc de test pour cet algorithme. Le matériel sur lequel a été implémenté cet algorithme est un PC utilisant un processeur Intel® Xeon™cadencé à 2.4GHz. La carte graphique, quant à elle, est une Nvidia® GeForce FX 5200.

Pour la partie logicielle, le rendu graphique bas niveau a été effectué en employant la librairie graphique OpenGL, et plus particulièrement les extensions `WGL_ARB_pbuffer` (qui permet d'effectuer le rendu directement dans la mémoire de la carte graphique) et `WGL_ARB_render_texture` (qui permet de lier une texture à une zone mémoire) pour améliorer les performances de notre technique de compensation [1].

[1] A l'époque de l'implémentation, l'extension `EXT_framebuffer_object` (extension sortie le 17 janvier 2005) n'existait pas. Cette dernière extension présente aussi l'avantage de s'affranchir des problèmes de plate-forme de développement et de la dépendance au système de fenêtrage. En effet, l'extension `WGL_ARB_render_texture` ne possédait pas de contrepartie sur les systèmes de type Unix. La nouvelle implémentation portable de notre algorithme donne des performances en tous points similaires.

#### 5.5.3.1 Algorithme principal du banc de simulation

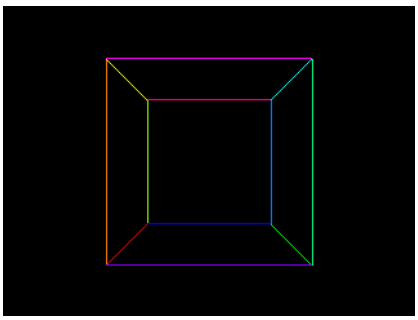
Les hypothèses de départ sont que la seconde prédiction va donner la position exacte de l'objet. Nous allons mesurer l'erreur entre une image de référence (telle qu'elle devrait être calculée par le système s'il n'y avait pas de temps de latence) et :

1. une image sur laquelle aucune correction n'est appliquée, c'est à dire, une image rendue en fonction du premier point de vue prédit,
2. une image sur laquelle notre algorithme de compensation a été appliqué.

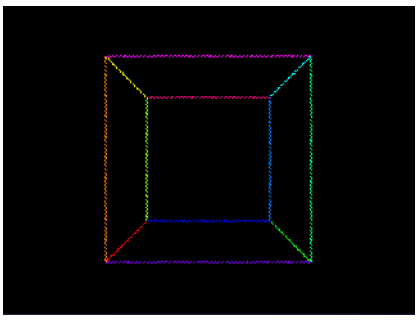
Nous allons alors comparer les erreurs obtenues suivant que notre algorithme soit employé ou non. Les erreurs seront exprimées en tant qu'erreurs moyennes en pixels.

Voici le protocole qui a été suivi pour évaluer cette algorithme :

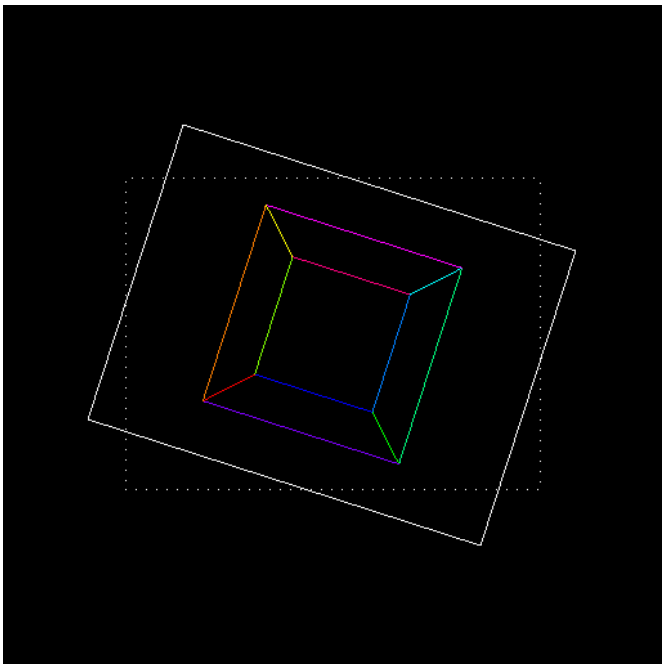
1. Effectuer le rendu de l'image de référence par rapport au deuxième point de vue de référence $C_2$ que l'on choisi arbitrairement,
2. Calculer un jeu de transformations $\Delta T$,
3. Pour chaque $\Delta T$ :
   (a) Effectuer le rendu de la même scène suivant le point de vue $C_1$ qui est lié au point de vue $C_2$ par la relation : $C_2 = \Delta T C_1$. Cette étape utilise les extensions OpenGL citées plus haut en vue d'optimiser les performances. En effet, le temps pris pour effectuer cette étape est chronométré,
   (b) Calculer l'erreur en pixels entre une scène rendue avec le point de vue $C_1$ et l'image de référence,
   (c) Calculer l'erreur en pixels entre la même scène sur laquelle on a appliqué la compensation et l'image de référence,



a) Image de référence



c) Image après compensation



b) Texture générée après perturbation

FIG. 5.14 – Exemple d'images du banc de test de la méthode de post-rendering

La figure 5.14 donne un exemple des images employées pour notre banc de test de la méthode de post-rendering. Ici, l'image de référence et l'image compensée sont de taille $320 \times 240$ pixels alors

que la texture générée est de taille $512 \times 512$ pixels. La compensation testée dans le cas présenté est la rotation autour de l'axe $\vec{z}$.

Nous avons effectué quatre séries de tests en utilisant ce banc de simulation. Le choix que nous avons fait est d'évaluer chaque degré de liberté de mouvement séparément. Deux séries de tests concernaient les mouvements de rotation autour des axes $(C_1, \vec{z})$ et $(C_1, \vec{y})$. Les deux derniers concernaient des déplacements en translation suivant les axes $(C_1, \vec{z})$ et $(C_1, \vec{x})$. Les unités choisies pour les rotations sont les degrés et pour la translation, ils seront exprimés dans la même unité que le paramètre $\delta$ qui par défaut seront appelés unité de translation.

Tout d'abord, pour prouver l'assertion d'utilisabilité de la compensation, il nous faut évaluer également les performances en temps d'exécution de l'algorithme.

#### 5.5.3.2 Temps d'exécution nécessité par l'algorithme de compensation

Puisque le calcul en lui-même est simple, le problème principal est d'accélérer le rendu de la texture nécessaire à l'algorithme de compensation qui est une goulet d'étranglement connu en terme de performances [Render to Texture, url]. Ceci implique le rendu direct de notre texture dans la mémoire de la carte graphique, sans passer par la copie des pixels issus du tampon image utilisé pour effectuer le rendu. Ceci est réalisé à l'aide de l'extension `WGL_ARB_render_texture`.

Le temps nécessité pour calculer une texture de taille 1024x1024 pixels$^2$ dans le but de l'afficher au travers d'un dispositif d'affichage de taille 800x600 pixels$^2$ est généralement en dessous de 4000 $\mu$secondes pour la majorité des compensations effectuées, à l'exception d'un déplacement qui amènerait le nouveau point de vue beaucoup plus près de la scène virtuelle que le précédent.

Les temps d'exécution pour chacune des séries de test sont tracés sur la figure 5.15 page suivante.

Nous pouvons dès lors valider l'hypothèse concernant le temps d'exécution de notre algorithme de compensation. Celui-ci s'effectue bien en un temps relativement court (environ 4ms) que l'on peut borner. Ceci permet également de prévoir le délai qui se rajoute à la distance de prédiction (chaque opération supplémentaire introduisant un nouveau délai ou temps de latence).

A présent, nous allons évaluer dans quelle mesure notre méthode de compensation permet de réduire les erreurs de recalage.

#### 5.5.3.3 La compensation de l'erreur de prédiction

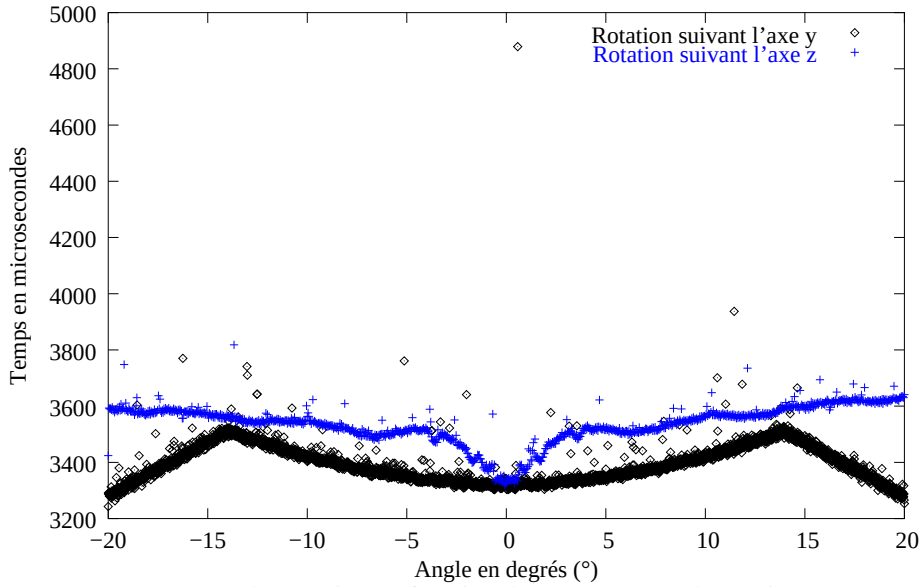
Dans un premier temps, nous évaluerons cette erreur sur les mouvements de rotations, puis sur les mouvements de translation.

**Mouvements de rotation pure** La figure 5.16 page 165 montre les erreurs obtenues pour les rotations effectuées autour des axes $\vec{z_i}$ et $\vec{x_i}$ (les erreurs obtenues autour de l'axe $\vec{y_i}$ ne sont pas représentées, les résultats étant les même que pour l'axe $\vec{x_i}$). Nous remarquons que l'erreur en pixels après compensation est de moins d'un pixel pour des angles variant entre -20$^o$ et +20$^o$, ce qui montre que notre algorithme corrige les erreurs de prédiction en rotation de manière presque parfaite.

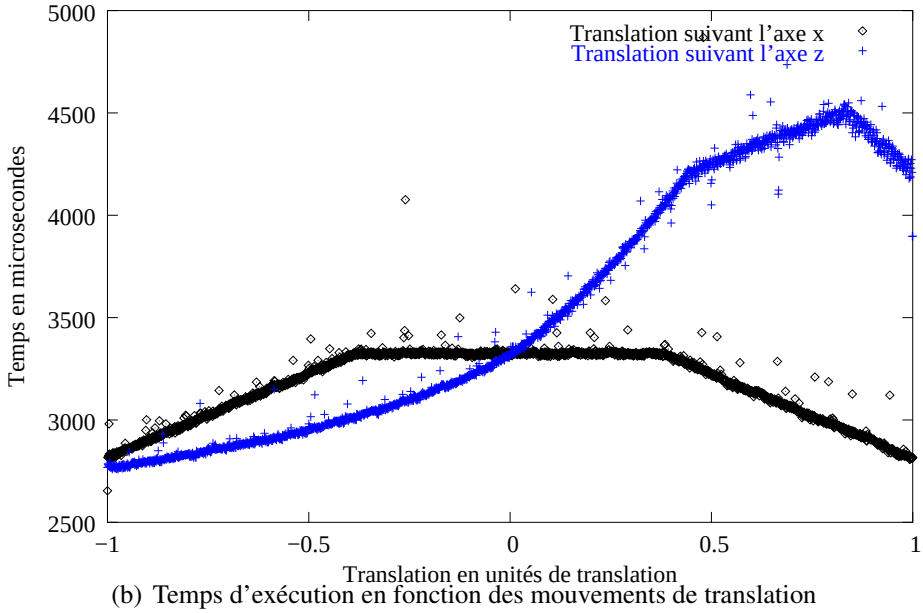
Une étude plus poussée des valeurs fournies nous a permis de voir que notre méthode n'apporte plus de bénéfice pour des angles très faibles de l'ordre de 0.04$^o$. Cependant, même dans ce cas de figure, l'erreur introduite est inférieure à 0.2 pixels. Nous pouvons donc considérer que la méthode peut-être employée pour corriger les erreurs de prédiction en rotation.

*Ceci prouve que notre méthode peut corriger de manière presque parfaite les erreurs de recalage pour des mouvements de rotation pure allant jusqu'à 20 degrés. De tels angles sont obtenus pour une cadence d'image d'environ 30 images par seconde pour des mouvements de la tête d'un opérateur humain.*

Nous allons à présent évaluer les erreurs obtenues pour des mouvements de translation.




(a) Temps d'exécution en fonction des mouvements de rotation




(b) Temps d'exécution en fonction des mouvements de translation

FIG. 5.15 – Temps d'exécution de l'algorithme de compensation pour chacune des séries de test.

**Mouvements de translation pure** Nous savons que les erreurs qui ne seront pas compensées de manière idéale par notre algorithme sont causées par des mouvements de translation. L'objet que nous avons choisi pour nos tests est inclus dans une boîte englobante cubique ayant pour taille de côté une unité de translation. La distance entre le centre de la boîte englobante et le sommet de la première pyramide est égal à 2 unités de translation, ce qui veut dire que pour notre série de tests le paramètre $\delta$ (notre distance focale arbitraire) aura pour valeur 2.

Comme précédemment, nous traçons les résultats obtenus pour des translations le long des axes $\vec{x_i}$ (les résultats sont similaires pour l'axe $\vec{y_i}$) et $\vec{z_i}$ en jouant sur les valeurs des paramètres $t_1$ et $t_3$ de la matrice représentée à l'équation 5.29 page 160. Dans ce cas, les erreurs introduites sont pour la plupart dues aux déformations perspective, puisque notre méthode assimile un objet à son plan de projection. Les erreurs calculées introduites par la figure 5.17 page 166 sont bien évidemment plus importantes que celles dues aux mouvements de rotation mais notre méthode parvient toujours à réduire les erreurs de recalage dans la plage des mouvements de translation que nous avons appliqué d'un facteur 10 environ.

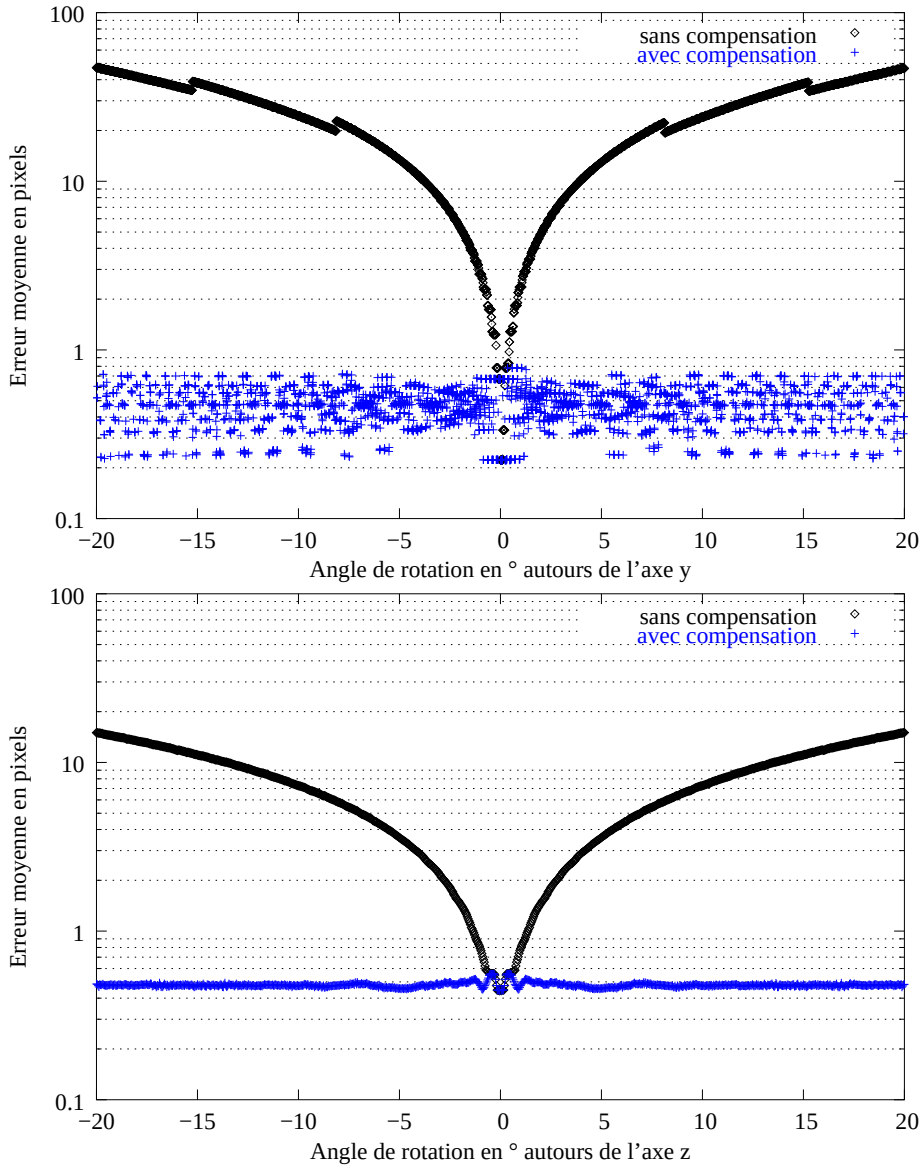



FIG. 5.16 – Erreur Moyenne en pixel (échelle logarithmique) calculée pour les mouvements de rotation

Comme nous pouvons le constater, notre méthode ne présente pas d'avantage réel pour les petites translations. Toutefois, comme cette erreur est toujours moins importante que s'il n'y avait pas de correction du tout, elle peut être effectuée pour tout type de translation le long de l'axe $\vec{x_i}$.

Les translations le long de l'axe $\vec{z_i}$ introduisent des résultats plus intéressant car la figure 5.17 page 166 montre que les performances de l'algorithme diminuent quand le deuxième point de vue prédit s'approche trop près de l'objet dont nous effectuons le rendu. Dans le cas présent, une translation de une unité de translation correspond à un rapprochement d'une distance de $\delta/2$ de l'objet. Nous pouvons donc conclure que s'approcher à une distance de $\delta/2$ de l'objet durant la correction de la prédiction mène à une perte de performances dues aux importantes déformations introduites par la projection perspective.

L'ensemble de cette étude sur les mouvements unitaires montre que nous pouvons employer notre algorithme dans une majorité des cas. D'autre part, nous savons également quelles sont les limites potentielles de cet algorithme, ce qui nous permet de choisir la meilleure alternative dans un cas défavorable à l'emploi de cet algorithme.

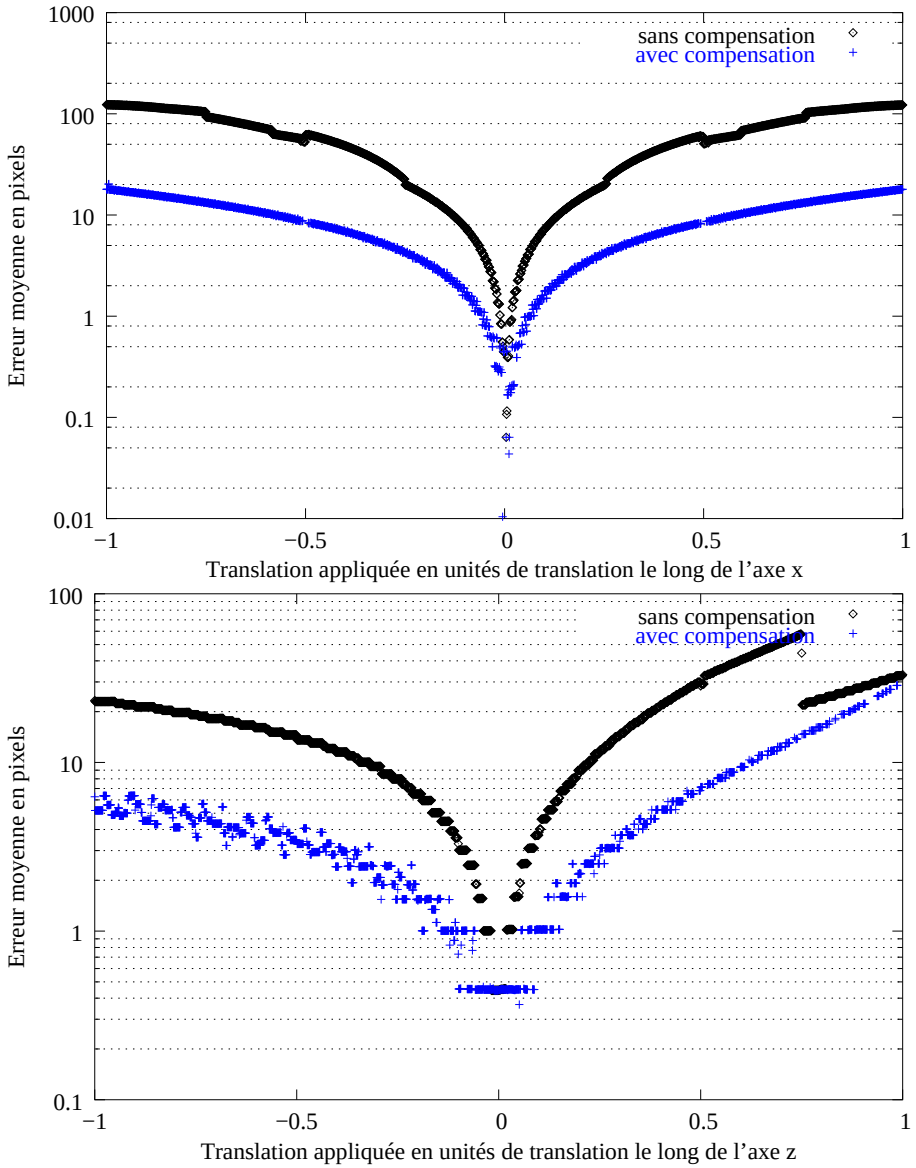



FIG. 5.17 – Erreur moyenne en pixels calculée pour des mouvements de translation

## 5.6 Conclusion

Nous avons présenté un système de recalage dynamique pour la réalité augmentée qui permet de suivre et d'estimer efficacement la tête d'un utilisateur. Cette approche est basée sur l'utilisation d'un filtre particulaire pour réaliser la fusion entre les données issues d'un système de localisation par la vision et les informations cinématiques issues d'une centrale inertielle. Nous avons comparé le filtre particulaire avec les approches classiquement utilisées telles le filtre de Kalman étendu qui a été appliqué dans les mêmes conditions expérimentales. Les erreurs quadratiques moyennes observées sur les prédictions de ces deux filtres montrent que le filtre particulaire se comporte de meilleure manière que le filtre de Kalman étendu.

Nous avons évalué, en simulation, notre filtre particulaires dans des conditions diverses, ce qui nous a permis de parvenir à la conclusion que ce dernier permet un suivi stable et robuste avec une erreur de projection de quelques pixels au maximum.

De plus, cette méthode de prédiction, combinée avec une méthode particulière de post-rendering que nous avons développé permet de compenser de manière idéale les mouvement de rotation et de réduire les erreurs de recalage dans les conditions d'utilisation normales de cet algorithme, c'est à dire pour compenser la latence dans le cadre de mouvements effectués par un opérateur humain.

Les travaux restant à effectuer sur le filtre particulaire consistent à évaluer la manière de le configurer pour parvenir aux performances optimales de ce dernier. Dans le cas idéal, l'ajustement manuel de ces paramètres devrait être remplacé par une procédure automatique qui garantirait l'obtention des conditions optimales. Une autre des options qui pourraient s'offrir à nous serait de changer le modèle en fonction des différents types de mouvements effectués par l'opérateur ainsi que d'ajuster le fonctionnement du filtre en conséquence.

L'étape suivante est l'intégration de ces composants dans un système de réalité augmentée en vision directe sur un casque de réalité virtuelle semi-transparent. Pour cela, il faudra résoudre les problèmes de calibration du point de vue de l'utilisateur par rapport au capteur hybride ainsi que la calibration interne de ce capteur, c'est à dire les relations de passage entre les repères propres à la centrale inertielle et ceux propres à la caméra.

# Conclusion générale

Dans cette thèse, nous nous sommes intéressés à l'ingénierie et aux méthodes de conception des systèmes de réalité augmentée appliqués à la maintenance industrielle. Ce champ de recherche comporte plusieurs verrous technologiques et scientifiques que nous avons identifiés et qui sont la localisation du système par rapport à l'espace de travail et aux objets qui le compose, le lien sémantique entre ce qui est vu et la base de connaissances du système, l'hétérogénéité technologique des solutions disponibles et enfin la complexité et la variété des tâches de maintenance.

Pour nous affranchir du verrou de l'hétérogénéité des solutions technologiques disponibles, nous avons jeté les bases d'une architecture flexible et modulaire s'appuyant sur la programmation orientée composants. Selon cette approche, une application de réalité augmentée est décrite par un ensemble de feuilles qui sont constituées de composants communiquant entre eux. A chaque état de l'application correspond une feuille active. Le changement de feuilles est alors orchestré par un automate. Cette architecture innovante permet de gérer de manière élégante et flexible le cycle de vie des composants au sein d'une application. De plus, le modèle introduit satisfait aux contraintes de temps réel imposées par les systèmes de réalité augmentée.

Sur cette architecture nous avons bâti un système de localisation par la vision qui repose sur l'utilisation de cibles codées. Ce système qui est hautement automatisé, nous permet de localiser en temps réel le système par rapport à la cible. Le code renfermé par cette dernière nous fournit également le lien sémantique entre ce qui est vu par la caméra et la base de connaissances du système. Ceci nous a permis d'introduire des algorithmes spécifiques pour améliorer la précision et la rapidité de la localisation en combinant une méthode analytique directe et une méthode itérative de calcul de la pose qui est l'itération orthogonale. L'approche proposée peut être améliorée en compensant les problèmes de distortion radiale de l'image dues à l'optique de la caméra car ces dernières sont une source d'erreur clairement identifiées concernant l'estimation des distances faites par le système.

De plus, nous avons développé des composants pour notre architecture qui nous permettent de gérer les augmentations. Ces composants forment un véritable moteur multimédia pour la réalité augmentée. Dans le cadre de la maintenance industrielle, les augmentations sont conditionnées par l'étape de l'opération de maintenance qui est réalisée. Nous avons donc étudié la structure interne de ces procédures que nous avons transformées et mises dans un format numérique qui exploite le standard XML. Leur contenu est alors lié à des documents multimédias (sons, images, vidéos, modèles 3D, etc) qui sont ensuite exploités par le système de gestion des augmentations. Les modèles 3D sont de leur côté instrumentés par des animations décrivant les étapes de la procédure de maintenance.

Enfin, nous avons souhaité élargir le système pour le transformer en un système de réalité augmentée en vision directe. Ceci permet de libérer les mains de l'opérateur. Dans le travail préparatoire que nous avons effectué, nous nous sommes attachés aux problèmes spécifiques présentés par ce genre de systèmes en terme de localisation spatiale et temporelle à cause de la latence. Nous présentons une approche qui combine les méthodes classiques issues de la communauté de réalité augmentée qui consistent à prédire le point de vue de l'utilisateur et les méthodes de la communauté de réa-

lité virtuelle qui utilisent les méthodes de post-rendering. Nous avons ainsi développé notre méthode qui s'appuie sur une prédiction réalisée à l'aide d'un filtre particulaire en lieu et place de l'approche classique qui emploie une filtre de Kalman étendu. Cette prédiction fournit des estimations d'une meilleure qualité et d'une plus grande robustesse au prix d'un surcôut en temps de calcul dont il faut tenir compte. Enfin, nous utilisons notre propre méthode de "post-rendering" qui permet, même si elle introduit des déformations perspectives dans l'image finale, de compenser la latence totale du système.

En résumé, les contributions énoncées ci-dessus permettent à notre système d'être flexible, temps-réel et orienté tâche, ce qui contribue à sa dextérité.

# Perspectives

Cette thèse présente un système complet de réalité augmentée dont chacun des points traités peut requérir une attention toute particulière. Les évolutions de ce système les plus immédiates concernent essentiellement le système de localisation par la vision dont le fonctionnement principal repose sur l'utilisation de marqueurs codés. Or ces derniers ne constituent pas une solution optimale pour les systèmes de réalité augmentée puisqu'ils imposent certaines contraintes aux objets sur lesquels ils sont fixés, comme par exemple les surfaces. D'autre part, dans des milieux poussiéreux, les marqueurs peuvent se retrouver occultés, ce qui va handicaper le bon fonctionnement du système de localisation par la vision. Le problème des systèmes d'extraction de primitives étant la première étape de sélection et de reconnaissance du modèle, nous pouvons imaginer une méthode hybride fonctionnant à l'aide d'un marqueur (non nécessairement collé sur l'objet) qui servirait à initialiser la pose de l'objet.

De plus, les algorithmes que nous avons développé pour les systèmes de réalité augmentée en vision directe doivent encore être implémenté sur un véritable système afin de confronter la simulation à la réalité. Ceci implique également l'utilisation d'algorithmes de calibration du système complet : d'une part la calibration du capteur hybride composé d'une caméra et de la centrale inertielle et d'autre part la calibration du système lunettes semi-transparentes et oeil de l'opérateur qui sont à elles seules des champs de recherche ouverts.

Au niveau du système de gestion des augmentations, en particulier dans le cadre de la préparation des modèles 3D pour la réalité augmentée, il convient d'automatiser cette étape en utilisant les modèles de CAO d'origine. De plus, le moteur multimédia présenté ne gère pas le problème des occlusions du virtuel par le réel et inversement.

Enfin, le système proposé est adapté à des environnements de travail de volumes restreints. Les applications de réalité augmentée en extérieur ("Outdoor Augmented Reality"), donc présentant un grand espace de travail, nécessiteront donc l'utilisation de capteurs supplémentaires.

# Bibliographie


[Ababsa et al., 2003] F. Ababsa, J.Y. Didier, M. Mallem, et D. Roussel (2003). Head motion prediction in augmented reality systems using monte carlo particle filters. Dans *Proceedings of the 13th International Conference on Artificial Reality and Telexixtance (ICAT 2003)*, pages 83–88. The Virtual Reality Society of Japan.

[Ababsa et Mallem, 2004] F. Ababsa et M. Mallem (2004). Robust camera pose estimation using 2d fiducials tracking for real-time augmented reality systems. Dans *Proceedings of ACM SIGGRAPH International Conference on Virtual-Reality Continuum and its Applications in Industry (VRCAI2004)*, Singapour.

[Abawi et al., 2004] D.F. Abawi, R. Dörner, M. Haller, et J. Zauner (2004). Efficient mixed reality application development. Dans *1st European Conference on Visual Media Production (CVMP)*, pages 289–294, Londres. IEEE.

[Abdel-Aziz et Karara, 1971] Y.I Abdel-Aziz et H.M. Karara (1971). Direct linear transformation into object space coordinates in close-range photogrammetry. Dans *Proceedings of the ASP/UI Symposium on Close-Range photogrammetry*, pages 1–18, Urbana. University of Illinois at Urbana-Champaign.

[Albrecht, 1989] R.E. Albrecht (1989). An adaptive digital filter to predict pilot head look direction for helmet-mounted displays. Master's thesis, University of Dayton, Ohio.

[Amire, url] Amire (url). Amire web site. http ://www.amire.net.

[Artoolkit, url] Artoolkit (url). Artoolkit web site. http ://www.hitl.washington.edu/artoolkit/.

[Arvika, url] Arvika (url). Arvika augmented reality for development, production and servicing. http ://www.arvika.de.

[Azuma, 1995] R. Azuma (1995). *Predictive Tracking for augmented reality*. thèse de doctorat, University of North Carolina.

[Azuma et al., 2001] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, et B. MacIntyre (2001). Recent advances in augmented reality. *IEEE Computer Graphics and Applications*, 21(1) :34–47.

[Bauer et al., 2001] M. Bauer, B. Bruegge, G. Klinker, A. MacWilliams, T. Reicher, S. Riss, C. Sandor, et M. Wagner (2001). Design of a component-based augmented reality framework. Dans *Proceedings of the International Symposium on Augmented Reality (ISAR)*.

[BBC, url] BBC (url). Bbc r&d projects. http ://www.bbc.co.uk/rd/projects/virtual/projects.shtml.

[Birell et Nelson, 1984] A.D. Birell et B.J. Nelson (1984). Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1) :39–59.

[Blum, 2001] S. Blum (2001). Towards a component-based system architecture for autonomous mobile robots. Dans *Proceedings of IASTED International Conference on Robotics and Applications (RA'01)*, pages 220–225, Tampa, Florida. IASTED.

[Boost, url] Boost (url). Boost web site. http ://www.boost.org/.

[Broll et al., 2000] W. Broll, E. Meier, et T. Shardt (2000). The virtual round table : a collaborative augmented multi-user environment. Dans *Proceedings of ACM Collaborative Virtual Environments*, pages 39–46, San Francisco. ACM.

[Brooks et al., 2005] A. Brooks, T. Kaupp, A. Makarenko, S. Williams, et A. Orebäck (2005). Towards component-based robotics. Dans *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'05)*, Edmonton, Canada.

[Butz et al., 1999] A. Butz, T. Höllerer, S. Feiner, B. MacIntyre, et C. Beshers (1999). Enveloping users and computers in a collaborative 3d augmented reality. Dans *IWAR '99 : Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality*, page 35, Washington, DC, USA. IEEE Computer Society.

[Chai et al., 1999] L. Chai, B. Hoff, T. Vincent, et K. Nguyen (1999). An adaptive estimator for registration in augmented reality. Dans *IWAR '99 : Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality*, page 23, Washington, DC, USA. IEEE Computer Society.

[Chai et al., 2002] L. Chai, W.A. Hoff, et T. Vincent (2002). Three-dimensional motion and structure estimation using inertial sensors and computer vision for augmented reality. *Presence : Teleoper. Virtual Environ.*, 11(5) :474–492.

[Cho et Neumann, 1998] Y. Cho et U. Neumann (1998). Multi-ring color fiducial systems for scalable fiducial tracking augmented reality. Dans *VRAIS '98 : Proceedings of the Virtual Reality Annual International Symposium*, page 212, Washington, DC, USA. IEEE Computer Society.

[Com, url] Com (url). Com : Component object model technologies. http ://www.microsoft.com/com/.

[Comport et al., 2003] A.I. Comport, E. Marchand, et F. Chaumette (2003). A real-time tracker for markerless augmented reality. Dans *Proceedings of the The 2nd IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 36–45. IEEE Computer Society.

[Cox et Song, 2001] P.T. Cox et B. Song (2001). A formal model for component-based software. Dans *HCC '01 : Proceedings of the IEEE 2001 Symposia on Human Centric Computing Languages and Environments (HCC'01)*, page 304, Washington, DC, USA. IEEE Computer Society.

[Curtis et al., 1998] D. Curtis, D. Mizell, P. Gruenbaum, et A. Janin (1998). Several devils in the details : Making an ar app work in the airplane factory. Dans *Proceedings of IWAR'98*, pages 47–60, San Francisco. ACM.

[Dart, url] Dart (url). Dart web site. http ://www.gvu.gatech.edu/dart/.

[DeMenthon et Davis, 1992] D. DeMenthon et L.S. Davis (1992). Exact and approximate solutions of the perspective-three-point problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(11) :1100–1105.

[Dementhon et Davis, 1995] D.F. Dementhon et L.S. Davis (1995). Model-based object pose in 25 lines of code. *International Journal of Computer Vision*, 15(1-2) :123–141.

[Dhome et al., 1989] M. Dhome, M. Richetin, et J.-T. Lapreste (1989). Determination of the attitude of 3d objects from a single perspective view. *IEEE Trans. Pattern Anal. Mach. Intell.*, 11(12) :1265–1278.

[Didier, 2002] J.Y. Didier (2002). Recalage dynamique dans un système de réalité augmentée en vision indirecte. Master's thesis, Université d'Evry-Val d'Essone.

[Didier, 2003a] J.Y. Didier (2003a). Architecture informatique dédiée à l'estimation et à la prédiction du point de vue d'un opérateur dans un système de réalité augmentée multicapteurs en vision directe. Communication au Journées RA-Temps Réel du GdrISIS.

[Didier, 2003b] J.Y. Didier (2003b). Architecture logicielle modulaire adaptée au recalage dynamique dans un système de réalité augmentée en vision directe. Dans *17ème journée des Jeunes Chercheurs en Robotique*, pages 102–106.

[Didier et al., 2004] J.Y. Didier, D. Roussel, et M. Mallem (2004). A texture based time delay compensation method for augmented reality. Dans *3rd IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR 2004)*, pages 262–263, Arlington (USA).

[Didier et al., 2005a] J.Y. Didier, D. Roussel, et M. Mallem (2005a). A time delay compensation method improving registration for augmented reality. Dans *2005 IEEE International Conference on Robotics and Automation (ICRA 2005)*, pages 3396–3400, Barcelone (Espagne).

[Didier et al., 2005b] J.-Y. Didier, D. Roussel, M. Mallem, S. Otmane, S. Naudet, Q.-C. Pham, S. Bourgeois, C. Mégard, C. Leroux, et A. Hocquard (2005b). Amra : Augmented reality assistance in train maintenance tasks. Dans *Workshop on Industrial Augmented Reality (ISMAR'05)*, Vienne, Autriche.

[Doucet et al., 2001] A. Doucet, N. de Freitas, et N.J. Gordon (2001). *Sequential Monte Carlo Methods in Practice*. Springer-Verlag.

[Drummond et Cipolla, 2002] T. Drummond et R. Cipolla (2002). Real-time visual tracking of complex structures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7) :932–946.

[D'Souza et Wills, 1998] D.F. D'Souza et A.C. Wills (1998). *Objects, Components, and Frameworks with UML - The Catalysis Approach*. Addison-Wesley.

[Dutoit et al., 2001] A. Dutoit, O. Crighton, G. Klinker, R. Kobylinski, C. Vilsmeier, et B. Bruegge (2001). Architectural issues in mobile augmented reality systems : a prototyping case study. Dans *Conférence ASPEC 2001*, Macau.

[Dwarf, url] Dwarf (url). Dwarf web site. http ://wwwbruegge.in.tum.de/DWARF/WebHome.

[Dörner et al., 2002] R. Dörner, C. Geiger, M. Haller, et V. Paelke (2002). Authoring mixed reality. a component and framework-based approach. Dans *First International Workshop on Entertainment Computing (IWEC 2002)*, Makuhari, Chiba, Japon.

[Endres et al., 2005] C. Endres, A. Butz, et A. MacWilliams (2005). A survey of software infrastructures and frameworks for ubiquitous computing. *Mobile Information Systems Journal*, 1(1).

[Faugeras et Toscani, 1986] O. Faugeras et G. Toscani (1986). The calibration problem for stereo. Dans *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 15–20, Miami Beach, Floride. IEEE.

[Feiner et al., 1997] S. Feiner, B. MacIntyre, T. Hollerer, et A. Webster (1997). A touring machine : Prototyping 3d mobile augmented reality systems for exploring the urban environment. Dans *ISWC '97 : Proceedings of the 1st IEEE International Symposium on Wearable Computers*, page 74, Washington, DC, USA. IEEE Computer Society.

[Feiner et al., 1993] S. Feiner, B. MacIntyre, et D. Seligmann (1993). Knowledge-based augmented reality. *Communications of the ACM*, 36(7) :52–62.

[Feiner et al., url] S. Feiner, B. MacIntyre, et D. Seligmann (url). Karma. http ://www.cs.columbia.edu/graphics/projects/karma.

[Fiala, 2005] M. Fiala (2005). Artag, a fiducial marker system using digital techniques. Dans *CVPR '05 : Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2*, pages 590–596, Washington, DC, USA. IEEE Computer Society.

[Fischler et Bolles, 1981] M. Fischler et R. Bolles (1981). Random sample consensus : A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6) :381–395.

[Foxlin et Naimark, 2003] E. Foxlin et L. Naimark (2003). Vis-tracker : A wearable vision-inertial self-tracker. Dans *VR '03 : Proceedings of the IEEE Virtual Reality 2003*, page 199, Washington, DC, USA. IEEE Computer Society.

[Fuchs et Moreau, 2004] P. Fuchs et G. Moreau (2004). *Le traité de la réalité virtuelle*, volume 1 : Fondements et interfaces comportementales. Presses de l'Ecole des Mines de Paris, deuxième édition édition.

[Garlan et Shaw, 1993] D. Garlan et M. Shaw (1993). An introduction to software architecture. Dans V. Ambriola et G. Tortora, editors, *Advances in Software Engineering and Knowledge Engineering*, volume 1, pages 1–40, New Jersey. World Scientific Publishing Company.

[Gennery, 1982] D.B. Gennery (1982). Tracking known three-dimensionnal objects. Dans *Conf. American Association of Artificial Intelligence*, pages 13–17.

[Gennery, 1992] D.B. Gennery (1992). Visual tracking of known three-dimensionnal objects. *International Journal of Computer Vision*, 8 :243–270.

[Goldberg et Robson, 1989] Adele Goldberg et David Robson (1989). *Smalltalk-80 : The Language*. Addison-Wesley.

[Gordon, 1997] N.J. Gordon (1997). A hybrid bootstrap filter for target tracking in clutter. *IEEE Transactions on Aerospace and Electronics Systems*, 33(1) :353–358.

[Gordon et al., 1993] N.J. Gordon, D. Salmond, et A.F.M. Smith (1993). A novel approach to nonlinear/non gaussian bayesian state estimation. Dans *IEE Proceedings on Radar and Signal Processing*, volume 140, pages 107–113.

[Gosling, 1996] James Gosling (1996). *The Java language Specification*. Addison-Wesley.

[Grasset et Gascuel, 2001] R. Grasset et J.-D. Gascuel (2001). Environnement de réalité augmentée collaboratif : Manipulation d'objets réels et virtuels. Dans *AFIG '01 (Actes des 14èmes journéees de l'AFIG)*, pages 101–112.

[Harris et Stennet, 1990] C. Harris et C. Stennet (1990). Rapid, a video rate object tracker. Dans *British Machine Vision Conference*, pages 73–77.

[Hejlsberg et Wiltamuth, 2001] A. Hejlsberg et S. Wiltamuth (2001). *Microsoft C# Language Specifications*. Microsoft Press.

[Holloway, 1995] R. L. Holloway (1995). Registration errors in augmented reality systems. Master's thesis, UNC Chapel Hill, Department of Computer Science.

[Hung et al., 1985] Y. Hung, P. Yeh, et D. Harwood (1985). Passive ranging to known planar point sets. Dans *Proceedings of IEEE International Conference on Robotics and Auto mation*, volume 1, pages 80–85.

[Ietm, 2000] Ietm (2000). *IETM. Interactive Electronic Technical Manuals.* US Navy. available at "www.ietm.net".

[Inc, 1993] NeXT Software Inc (1993). *Object-Oriented Programming and the Objective C Language*. Addison-Wesley.

[Kato et Billinghurst, 1999] H. Kato et M. Billinghurst (1999). Marker tracking and hmd calibration for a video-based augmented reality conferencing system. Dans *IWAR '99 : Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality*, pages 85–92, Washington, DC, USA. IEEE Computer Society.

[Kato et al., 2000] H. Kato, M. Billinghurst, I. Poupyrev, K. Imamoto, et K. Tachibana (2000). Virtual object manipulation on a table-top ar environment. Dans *Proceedings of the International Symposium on Augmented Reality (ISAR 2000)*, pages 111–119, Munich, Germany.

[Kijima et Ojika, 2002] R. Kijima et T. Ojika (2002). Reflex hmd to compensate lag and correction of derivative deformation. Dans *International Conference on Virtual Reality 2002 (VR2002)*, pages 172–179. IEEE.

[Liang et al., 1991] J. Liang, C. Shaw, et M. Green (1991). On temporal-spatial realism in the virtual reality environment. Dans *Proceedings of the 4th Annual ACM Symposium on User Interface Software & Technology*, pages 19–25, Hilton Head.

[Loukil, 1993] A. Loukil (1993). Interface homme-machine de contrôle-commande en robotique téléopérée. Master's thesis, Université d'Evry Val d'Essonnes.

[Lowe, 1987] D. Lowe (1987). Three-dimensional object recognition from single two-dimensional images. *Artificial Intelligence*, 31(3) :355–395.

[Lowe, 1992] D. Lowe (1992). Robust model-based motion tracking through the integration of search and estimation. *Interantional Journal of Computer Vision*, 8(2) :113–122.

[Lu et al., 2000] C.-P. Lu, G.D. Hager, et E. Mjolsness (2000). Fast and globally convergent pose estimation from video images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(6) :610–622.

[MacIntyre et al., 2003] B. MacIntyre, M. Gandy, J. Bolter, S. Dow, et B. Hannigan (2003). Dart : The designer's augmented reality toolkit. Dans *The Symposium on User Interface Software and Technology (UIST 04)*, Vancouver, BC, Canada.

[MacIntyre et al., 2004] B. MacIntyre, M. Gandy, S. Dow, et J.D. Bolter (2004). Dart : a toolkit for rapid design exploration of augmented reality experiences. Dans *Proceedings of the 17th annual ACM symposium on User interface software and technology (UIST 04)*, pages 197–206. ACM Press.

[MacWilliams et al., 2004] A. MacWilliams, T. Reicher, G. Klinker, et B. Brüegge (2004). Design patterns for augmented reality systems. Dans *Proceedings of the International Workshop exploring the Design and Engineering of Mixed Reality Systems (MIXER), Funchal, Madeira, CEUR Workshop Proceedings*.

[Malbezin et al., 2002] P. Malbezin, W. Piekarski, et B.H. Thomas (2002). Measuring artoolkit accuracy in long distance tracking experiments. Dans *1st Int'l Augmented Reality Toolkit Workshop*, Darmstadt. IEEE.

[Marchand et al., 1999] E. Marchand, P. Bouthemy, F. Chaumette, et V. Moreau (1999). Robust real-time visual tracking using a 2d-3d model-based approach. Dans *International Conference on Computer Vision*, volume 1, pages 262–268, Corfu, Greece. IEEE Computer Society.

[Mark et al., 1997] W. Mark, L. McMillan, et G. Bishop (1997). Post-rendering 3d warping. Dans *Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 7–16, Providence, RI.

[Mehta et al., 2000] N.R. Mehta, N. Medvidovic, et S. Phadke (2000). Towards a toxonomy of software connectors. Dans *Proceedings of the 22nd International Conference on Software Engineering (ICSE2000)*, pages 178–187, Limerick, Irlande.

[Microsystems, 1997a] Sun Microsystems, editor (1997a). *Java Core Reflection - API and specification*. Javasoft.

[Microsystems, 1997b] Sun Microsystems (1997b). Javabeans for java studio : Architecture and api. White paper, Sun Microsystems.

[Milgram et al., 1994] P. Milgram, H. Takemura, A. Utsumi, et F. Kishino (1994). Augmented reality : A class of displays on the reality-virtuality continuum. *SPIE : Telemanipulator and Telepresence Technologies*, 2351 :282–292.

[MIME, url] MIME (url). Mime media types. http ://www.iana.org/assignments/media-types/.

[Moore, 1956] E. F. Moore (1956). Gedanken experiments on sequential machines. Dans *Automata Studies*, pages 129–153, Princeton.

[More, 1977] J.J. More (1977). The levenberg-marquardt algorithm, implementation and theory. Dans Watson G.A., editor, *Numerical Analysis*, Lecture Notes in Mathematics. Springer-Verlag.

[Naimark et Foxlin, 2002] L. Naimark et E. Foxlin (2002). Circular data matrix fiducial system and robust image processing for a wearable vision-inertial self-tracker. Dans *ISMAR '02 : Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR'02)*, pages 27–36, Washington, DC, USA. IEEE Computer Society.

[Neunmann et Majoros, 1998] U. Neunmann et A. Majoros (1998). Cognitive, performance and system issues for augmented reality applications in manufacturing and maintenance. Dans *Proceedings of IEEE Virtual Reality Annual International Symposium (VRAIS'98)*, pages 4–11, Atlanta. IEEE.

[Ohshira et al., 1998] T. Ohshira, K. Satoh, H. Yamamoto, et H. Tamura (1998). $Ar^2$hockey : A case study of collaborative augmented reality. Dans *Proceedings of IEEE Virtual Reality Annual International Symposium (VRAIS'98)*, pages 268–275, Atlanta.

[Olwal et al., 2005] A. Olwal, C. Lindfors, J. Gustafsson, T. Kjellberg, et L. Mattsson (2005). Astor : An autostereoscopic optical see-through augmented reality system. Dans *Fourth IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 24–27, Vienne (Autriche).

[Owen et al., 2002] C.B. Owen, X. Fan, et P. Middlin (2002). What is the best fiducial ? Dans *Augmented Reality Toolkit, The First IEEE International Workshop*. IEEE.

[Piekarski et Thomas, 2003] W. Piekarski et B.H. Thomas (2003). An object-oriented software architecture for 3d mixed reality applications. Dans *The Second IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'03)*, Tokyo, Japan.

[Popescu et al., 2000] V. Popescu, J. Eyles, A. Lastra, J. Steinhurst, N. England, et L. Nyland (2000). The warpengine : An architecture for the post-polygonal age. Dans *Proceedings of SIGGRAPH 2000*, pages 433–442, New Orleans, La.

[Qt, url] Qt (url). Qt web site. http ://www.trolltech.com/products/qt/index.html.

[Regan et Pose, 1994] M. Regan et R. Pose (1994). Priority rendering with a virtual reality address recalculation pipeline. Dans *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 155–162.

[Reiners et al., 1998] D. Reiners, D. Stricker, G. Klinker, et S. Muller (1998). Augmented reality for contruction task : doorlock assembly. Dans *Proceedings 1st International Workshop on Augmented Reality (IWAR'98)*, pages 31–46, San Francisco.

[Reitmayr et Schmalstieg, 2001] G. Reitmayr et D. Schmalstieg (2001). An open software architecture for virtual reality interaction. Dans *Proceedings of the ACM symposium on Virtual reality software and t echnology*, pages 47–54. ACM Press.

[Rekimoto, 1998] J. Rekimoto (1998). Matrix : A realtime object identification and registration method for augmented reality. Dans *APCHI '98 : Proceedings of the Third Asian Pacific Computer and Human Interaction*, pages 63–68, Washington, DC, USA. IEEE Computer Society.

[Rekimoto et Ayatsuka, 2000] J. Rekimoto et Y. Ayatsuka (2000). Cybercode : designing augmented reality environments with visual tags. Dans *DARE '00 : Proceedings of DARE 2000 on Designing augmented reality environments*, pages 1–10. ACM Press.

[Render to Texture, url] Render to Texture (url). Fast texture transfers. http ://developer.nvidia.com/object/fast_texture_transfers.html.

[RFC, url] RFC (url). Internet rfc/std/fyi/bcp archives search. http ://www.faqs.org/rfcs/rfcsearch.html.

[RNTL, url] RNTL (url). Réseau national des technologies logicielles. http ://www.telecom.gouv.fr/rntl/.

[Schmalstieg et al., 2002] D. Schmalstieg, A. Fuhrmann, G. Hesina, Z. Szalavari, L. M. Encarnaçäo, M. Gervautz, et W. Purgathofer (2002). The studierstube augmented reality project. *Presence : Teleoperators and Virtual Environments*, 11(1) :33–54.

[Schwald et de Laval, 2003] B. Schwald et B. de Laval (2003). An augmented reality system for training and assistence to maintenance in the industrial context. Dans *Proc. 11th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2003 (WSCG)*.

[Schwald et al., 2001] B. Schwald, J. Figue, E. Chauvineau, F. Vu-Hong, A. Robert, M. Arbolino, M. Schnaider, B. De Laval, F. Dumas De Rauly, F. Anez, O. Baldo, et J.M. Santos (2001). Starmate : Using augmented reality for computer guided maintenance of complex mechanical elements. Dans *Conference e2001*, Venise.

[Sigc++, url] Sigc++ (url). Sigc++ web site. http ://libsigc.sourceforge.net/.

[Sigslot, url] Sigslot (url). Sigslot web site. http ://sigslot.sourceforge.net/.

[So et Griffin, 1992] R.H.Y So et M.J. Griffin (1992). Compensating lags in head coupled displays using head position prediction and image deflection. *Aircraft*, 29(6) :1064–1068.

[Sutherland, 1968] I. Sutherland (1968). A head-mounted three dimensional display. Dans *Proceedings of the Fall Joint Computer Conference*, pages 757–764, Washington DC. Thompson Books.

[Szyperski, 2002] C. Szyperski (2002). *Component Software - Beyond Object-Oriented Programming*. Addison-Wesley, Harlow, England, second édition.

[Szyperski, 2003] C. Szyperski (2003). Component technology - what, where, and how ? Dans *Proceedings of the 25th International Conference on Software Engineering (ICSE2003)*, pages 684–693, Portland, Oregon, USA.

[Tinmith, url] Tinmith (url). Tinmith web site. http ://www.tinmith.net/.

[Tsai, 1987] R. Y. Tsai (1987). A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Journal of Robotics and Automation*, RA-3(4) :323–344.

[Vacchetti et al., 2004] L. Vacchetti, V. Lepetit, et P. Fua (2004). Stable real-time 3d tracking using online and offline information. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE Intelligence*, 26(10) :1385– 1391.

[Ventura, 1988] C. Ventura (1988). Why switch from paper to electronic manuals. Dans *DOCPROCS '88 : Proceedings of the ACM Conference on Document Processing Systems*, pages 111–116.

[Vigueras et al., 2003] F. Vigueras, M.O. Berger, et G. Simon (2003). Iterative multi-planar camera calibration : Improving stability using model selection. Dans *Vision, Video and Graphics (VVG)'03*, Bath, UK. Eurographics Association.

[Vigueras-Gomez et al., 2005] J.-F. Vigueras-Gomez, G. Simon, et M.-O. Berger (2005). Calibration errors in augmented reality : A practical study. Dans *Fourth IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 154–163, Vienne (Autriche).

[Vrml, 1997] Vrml (1997). *The Virtual Reality Modeling Language Specification, ISO/IEC DIS 14772-1*. Web3D Consortium. available at "www.web3d.org/x3d/specifications/vrml/vrml97/index.htm".

[Webster et al., 1996] A. Webster, S. Feiner, B. MacIntyre, W. Massie, et T. Kruegger (1996). Augmented reality in architectural construction, inspection, and renovation. Dans *Proceedings of the Third ASCE Congress for Computing in Civil Engineering*, pages 913–919, Anaheim.

[Welch et Foxlin, 2002] G. Welch et E. Foxlin (2002). Motion tracking : No silver bullet, but a respectable arsenal. *IEEE Computer Graphics and Applications*, 22(6) :24–38.

[Welsh et Bishop, 1995] G. Welsh et G. Bishop (1995). An introduction to the kalman filter. Technical Report TR95-041, University of North Carolina at Chapel Hill.

[Wuest et al., 2005] H. Wuest, F. Vial, et D. Stricker (2005). Adaptive line tracking with multiple hypotheses for augmented reality. Dans *Fourth IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 62–69, Vienne (Autriche).

[X3D, url] X3D (url). *X3D and Related Specifications*. Web3D Consortium. available at "www.web3d.org/x3d/specifications/".

[Xpcom, url] Xpcom (url). Xpcom. http ://www.mozilla.org/projects/xpcom/.

[Yergeau et al., 2004] F. Yergeau, T. Bray, J. Paoli, C. M. Sperberg-McQueen, et E. Maler (2004). Extensible markup language (xml) 1.0 (third edition). http ://w3c.org/TR/2004/REC-xml-20040204.

[You et Neumann, 2001] S. You et U. Neumann (2001). Fusion of vision and gyro tracking for robust augmented reality registration. Dans *In IEEE Conference on Virtual Reality*, pages 71–78, Yokohama.

[Zhang et al., 2002] X. Zhang, S. Fronz, et N. Navab (2002). Visual marker detection and decoding in ar systems : A comparative study. Dans *ISMAR '02 : Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR'02)*, page 97, Washington, DC, USA. IEEE Computer Society.

[Zhang, 1999] Z. Zhang (1999). Flexible camera calibration by viewing a plane from unknown orientations. Dans *International Conference on Computer Vision*, volume 1, page 666, Corfu, Greece.

# Annexe A

# DTD des divers modèles XML employés

## A.1 Notations employées dans les DTD

Nous pouvons décrire l'ensemble des balises d'un document XML ainsi que leur agencement les unes par rapport aux autres grâce à sa DTD (Document Type Definition) La DTD contient pour ce faire un ensemble de balises figées qui sont les suivantes :

- La balise **!ELEMENT** est utilisée pour déclarer des éléments (ou balises).
- La balise **!ATTLIST** est utilisée pour lister les différents attributs associés à l'élément qu'il a en paramètre.

Chacune de ces balises possède des paramètres spécifiques.

### A.1.1 La balise `!ELEMENT`

Sa syntaxe est la suivante : `<!ELEMENT nomBalise contenu>`
Le contenu peut être vide (`EMPTY`), rempli avec du contenu quelconque (`#PCDATA`) ou encore un assemblage de plusieurs autres éléments. Cet assemblage possède certaines similarités avec les expressions régulières. Ainsi, les symboles ***** (nombre indéterminé), **?** (un au plus), **+** (au moins 1), et **|** (le symbole ou) peuvent être utilisés. Par exemple :

`<!ELEMENT sheets (sheet*)>`
Ici, l'élément sheets est composé de plusieurs éléments sheet.
`<!ELEMENT sheet ( objects? , wires? )>`
Ici, l'élément sheet est composé d'au plus un éléments `wires` et d'au plus un élément `sheets`.

### A.1.2 La balise `!ATTLIST`

Pour chaque élément, il est possible de dresser la liste de ses attributs. La syntaxe est la suivante :

```
<!ATTLIST nomBalise
          nomAttribut1 type1 valeur1
          ...
  nomAttributN typeN valeurN>
```

Les types des attributs peuvent être les suivants :

- `ID`, l'attribut est un identifiant. Il doit être unique,
- `IDREF`, l'attribut fait référence à un identifiant,
- `CDATA`, l'attribut est de type quelconque.

La valeur des attributs, si elle est précisée indique sa valeur par défaut, ou si `#REQUIRED` est précisé, que cet attribut doit obligatoirement être renseigné.

## A.2 DTD d'une application

Nous allons présenter ici la DTD générale d'une application telle qu'elle est décrite dans notre système de programmation par composant (voir section 2.5.2 page 60).

```
<?xml version="1.0"?>
<!DOCTYPE application [
<!ELEMENT define (#PCDATA)>
<!ATTLIST define
          id     ID     #REQUIRED
          type   CDATA  #REQUIRED
          value  CDATA  #REQUIRED>
<!ELEMENT library (#PCDATA)>
<!ATTLIST library name CDATA #REQUIRED>
<!ELEMENT object (#PCDATA)>
<!ATTLIST object
          classname CDATA
          file CDATA
          persistent CDATA "true"
          id ID #REQUIRED>
<!ELEMENT init (#PCDATA)>
<!ATTLIST init
          object IDREF #REQUIRED
          slot CDATA #REQUIRED
          type (int|short|long|float|double|string|pointer|define|void|object) "
              int"
          value CDATA "NULL">
<!ELEMENT wire EMPTY>
<!ATTLIST wire
          objsource IDREF #REQUIRED
          signal    CDATA #REQUIRED
          objdest   IDREF #REQUIRED
          slot      CDATA #REQUIRED>
<!ATTLIST tokensender
          object    IDREF #REQUIRED>
<!ELEMENT tokensender (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
<!ELEMENT preconnection (init*)>
<!ELEMENT connection (wire*)>
<!ELEMENT postconnection (init*)>
<!ELEMENT sheet (tokensender?,preconnection?,connection?,postconnection?,comment
    ?)>
<!ATTLIST sheet
          id ID #REQUIRED>
<!ELEMENT transition (#PCDATA)>
<!ATTLIST transition
          stepA IDREF #REQUIRED
          stepB IDREF #REQUIRED
          token CDATA #REQUIRED>
<!ATTLIST statemachine
          terminal CDATA "end">
<!ELEMENT statemachine (transition*)>
<!ELEMENT sheets (sheet*)>
<!ELEMENT objects (object*)>
<!ELEMENT libraries (library*)>
<!ELEMENT defines (define*)>
<!ELEMENT application (defines, libraries, objects, sheets, statemachine) >
]>
```

## A.3 DTD d'un macro-bloc

Les applications de notre système de programmation par composants peuvent également contenir des macro-blocs. C'est à dire des pré-assemblages de composants (voir section 2.5.2 page 60). Voici la DTD associée aux macro-blocs :

```
<?xml version="1.0"?>
<!DOCTYPE block [
<!ELEMENT define (#PCDATA)>
<!ATTLIST define
          id     ID     #REQUIRED
          type   CDATA  #REQUIRED
          value  CDATA  #REQUIRED>
<!ELEMENT library (#PCDATA)>
<!ATTLIST library name CDATA #REQUIRED>
<!ELEMENT object (#PCDATA)>
<!ATTLIST object
          classname CDATA
          file CDATA
          id ID #REQUIRED>
<!ELEMENT init (#PCDATA)>
<!ATTLIST init
          object IDREF #REQUIRED
          slot CDATA #REQUIRED
          type (int|short|long|float|double|string|pointer|void|object) "int"
          value CDATA "NULL">
<!ELEMENT wire EMPTY>
<!ATTLIST wire
          objsource IDREF #REQUIRED
          signal    CDATA #REQUIRED
          objdest   IDREF #REQUIRED
          slot      CDATA #REQUIRED>
<!ELEMENT comment (#PCDATA)>
<!ELEMENT preconnection (init*)>
<!ELEMENT connection (wire*)>
<!ELEMENT postconnection (init*)>
<!ELEMENT signals (method*)>
<!ELEMENT slots (method*)>
<!ELEMENT method (#PCDATA)>
<!ATTLIST method
          name   CDATA #REQUIRED
          object IDREF #REQUIRED
          method CDATA #REQUIRED>
<!ELEMENT sheet (preconnection?,connection?,postconnection?,comment?)>
<!ELEMENT objects (object*)>
<!ELEMENT libraries (library*)>
<!ELEMENT defines (define*)>
<!ELEMENT block (defines, libraries, objects, sheet, signals, slots) >
]>
```

## A.4 DTD d'un projet

Les applications peuvent être directement éditées à l'aide d'une interface graphique (voir section 2.5.4 page 69). Cette dernière entrepose également des paramètres concernant le placement graphique des composants qui sont stockés dans un fichier XML. Sa DTD est donc également fournie ici.

```
<?xml version="1.0"?>
<!DOCTYPE application [
<!ELEMENT       application (#PCDATA)>
<!ATTLIST       application fileApplication  CDATA  #REQUIRED>
<!ELEMENT       object EMPTY>
<!ATTLIST       object
        x       CDATA   #REQUIRED
        y       CDATA   #REQUIRED
        id      ID              #REQUIRED>
<!ELEMENT   grip EMPTY>
<!ATTLIST       grip
        x       CDATA   #REQUIRED
        y       CDATA   #REQUIRED>
<!ELEMENT       wire( grip? )>
<!ATTLIST       wire
        objsource       IDREF   #REQUIRED
        objdest         IDREF   #REQUIRED
        signal  CDATA   #REQUIRED
        slot            CDATA   #REQUIRED>
<!ELEMENT       objects (object*)>
<!ELEMENT       wires (wire*)>
<!ELEMENT       sheet ( objects? , wires? )>
<!ATTLIST       sheet
        x       CDATA   #REQUIRED
        y       CDATA   #REQUIRED
        id      ID              #REQUIRED >
<!ELEMENT       sheets (sheet*)>
<!ELEMENT       graphique ( application , sheets )>
]>
```

## A.5 DTD d'une procédure de maintenance

Les procédures de maintenance extraites des manuels techniques et numérisées sous un format XML (voir section 4.2.2 page 126) disposent également d'une DTD décrivant l'agencement des balises employées.

```
<?xml version="1.0"?>
<!DOCTYPE procedure [
<!ELEMENT media (#PCDATA)>
<!ATTLIST media mid CDATA #REQUIRED>
<!ATTLIST media mfile CDATA #REQUIRED>
<!ATTLIST media mtype (animation3D|photo|photoAugmented|comment|registrated) #
    REQUIRED>

<!ELEMENT detail (#PCDATA)>
<!ATTLIST detail link CDATA #REQUIRED>

<!ELEMENT note (#PCDATA)>

<!ELEMENT procedure (proc_title,pre_requirement,available_media, phases)>
<!ELEMENT proc_title (#PCDATA)>
```

```
<!ELEMENT pre_requirement (req_conditions,tools,spares,consumables)>
<!ELEMENT req_conditions (safety*,caution*,note*)>

<!ELEMENT safety (#PCDATA|detail)*>
<!ATTLIST safety did CDATA #IMPLIED>

<!ELEMENT caution (#PCDATA)>
<!ELEMENT tools (tool*)>
<!ELEMENT tool (#PCDATA)>
<!ELEMENT spares (req_spare*)>
<!ELEMENT req_spare (reference,name,quantity)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT reference (#PCDATA)>
<!ELEMENT quantity (#PCDATA)>
<!ELEMENT consumables (consumable)*>
<!ELEMENT consumable (reference,name,quantity)>
<!ELEMENT available_media (media)*>
<!ELEMENT phases (phase+)>

<!ELEMENT phase (title?,(content|note|idmedia|phase)*)>
<!ATTLIST phase pid ID #REQUIRED>
<!ATTLIST phase fid CDATA #IMPLIED>

<!ELEMENT idmedia (#PCDATA)>
<!ATTLIST idmedia mid CDATA #REQUIRED>

<!ELEMENT title (#PCDATA)>
<!ELEMENT content (#PCDATA|spare|detail)*>

<!ELEMENT spare (#PCDATA)>
<!ATTLIST spare sid CDATA #REQUIRED      >
]>
```

# Annexe B

# Exemples d'applications et de macro-blocs décrits à l'aide de notre jeu de balises XML

Cette annexe présente quelques exemples de fichiers XML employés pour décrire une application et un macro-bloc. Il s'agit de l'application directe des jeux de balises XML décrites dans la section 2.5.2 page 60.

## B.1 Un premier exemple simple

Cet exemple emploie les deux composants dont le listing est référencé à la page 53. L'application est composée de trois états (donc feuilles) différents. Dans le premier, 10 itérations sont réalisées. Dans le deuxième, 5 itérations seulement sont réalisées. Enfin, le dernier permet de sortir de l'application.

```
<?xml version="1.0"?>
<!DOCTYPE application SYSTEM "../xml/application.dtd">
<application>
  <libraries>
    <library name="libs/boucle"/>
  </libraries>
  <objects>
    <object classname="Boucle" id="B"/>
    <object classname="DisplayInt" id="D"/>
    <object classname="Boucle" id="B1"/>
    <object classname="DisplayInt" id="D1"/>
  </objects>
  <sheets>
    <sheet id="S1">
      <tokensender object="B"/>
      <connection>
        <wire objsource="B" signal="newIteration(int)" objdest="D" slot="display
            (int)"/>
      </connection>
      <postconnection>
        <init object="B" slot="setIterations(int)" type="int" value="10"/>
      </postconnection>
    </sheet>
    <sheet id="S2">
      <tokensender object="B1"/>
      <connection>
        <wire objsource="B1" signal="newIteration(int)" objdest="D1" slot="
            display(int)"/>
      </connection>
```

```
      <postconnection>
        <init object="B1" slot="setIterations(int)" type="int" value="5"/>
      </postconnection>
    </sheet>

    <sheet id="E"/>
  </sheets>
  <statemachine terminal="E">
    <transition stepA="S1" stepB="S2" token="end"/>
    <transition stepA="S2" stepB="E" token="end"/>
  </statemachine>
</application>
```

Le lancement de cette application à l'aide de notre moteur d'exécution donne le résultat suivant où l'on va effectivement retrouver 10 itérations puis 5 autres :

```
Waiting 1s before launching runtime ...
Launching Runtime, go go go !
============================================================
Emitting iteration 0
Recieved integer 0
Emitting iteration 1
Recieved integer 1
Emitting iteration 2
Recieved integer 2
Emitting iteration 3
Recieved integer 3
Emitting iteration 4
Recieved integer 4
Emitting iteration 5
Recieved integer 5
Emitting iteration 6
Recieved integer 6
Emitting iteration 7
Recieved integer 7
Emitting iteration 8
Recieved integer 8
Emitting iteration 9
Recieved integer 9
Emitting iteration 0
Recieved integer 0
Emitting iteration 1
Recieved integer 1
Emitting iteration 2
Recieved integer 2
Emitting iteration 3
Recieved integer 3
Emitting iteration 4
Recieved integer 4
```

## B.2 Exemple de macro-bloc

Ce macro-bloc décrit l'ensemble des composants nécessités pour récupérer une image d'une caméra branchée sur une carte d'acquisition de type Meteor II. Il s'agit en réalité des composants d'acquisition présentés dans l'application vue à la section 3.4.2.6 page 102.

```
<?xml version="1.0"?>
<!DOCTYPE application SYSTEM "macroblock.dtd">
<block>
<defines>
<define id="NB_CHANNEL" type="int" value="3"/>
</defines>
  <libraries>
    <library name="../libs/input"/>
    <library name="../libs/images"/>
    <library name="../libs/ocv"/>
  </libraries>
  <objects>
    <object classname="CameraMeteor" id="met"/>
    <object classname="MeanRawDeinterlace" id="dei" />
    <object classname="ChannelExtractor" id="ext" />
    <object classname="RawCVImageConvertor" id="rcv"/>
  </objects>
  <sheet>
    <preconnection>
      <init object="met" slot="setDevice(QString)" type="string" value="/dev/
          video0" />
      <init object="met" slot="setBuffers(int)" type="int" value="2" />
      <init object="ext" slot="extractRed(bool)" type="bool" value="true" />
      <init object="ext" slot="extractBlue(bool)" type="bool" value="false" />
      <init object="ext" slot="extractGreen(bool)" type="bool" value="false" />
      <init object="ext" slot="extractAlpha(bool)" type="bool" value="false" />
      <init object="rcv" slot="setChannels(int)" type="define" value="NB_CHANNEL
          "/>
      <init object="dei" slot="setChannels(int)" type="define" value="NB_CHANNEL
          "/>
    </preconnection>
    <connection>
      <wire objsource="met" signal="sendImage(int,int,char*)" objdest="dei" slot
          ="deinterlaceRaw(int,int,char*)" />
      <wire objsource="dei" signal="rawProcessed(int,int,char*)" objdest="rcv"
          slot="convertRaw(int,int,char*)"/>
      <wire objsource="rcv" signal="rawProcessed(IplImage*)" objdest="ext" slot=
          "extractChannels(IplImage*)"/>
    </connection>
    <postconnection></postconnection>
  </sheet>
  <signals>
    <method name="sendImage(IplImage*)" object="ext" method="sendRedChannel(
        IplImage*)" />
  <method name="sendWidth(int)" object="met" method="sendWidth(int)" />
  <method name="sendheight(int)" object="met" method="sendHeigh(int)" />
  </signals>
  <slots>
  <method name="initDevice()" object="met" method="initDevice()" />
  <method name="setThreaded(bool)" object="met" method="setThreaded(bool)" />
  <method name="start()" object="met" method="start()" />
  <method name="stop()" object="met" method="stop()" />
  </slots>
</block>
```

# Annexe C

# Limitations du moteur multimédia par rapport aux standards supportés

Les principales limitations du moteur multimédia viennent des limitations des diverses librairies qui ont été exploitées pour l'élaboration de ce dernier. Nous recensons essentiellement trois types de limitations :

- les limitations sur l'interprétation du format VRML,
- les limitations sur l'interprétation des scripts Javascript,
- les limitations sur l'implémentation du standard VRML en Javascript.

## C.1 Limitations VRML

La librairie employée pour interpréter le VRML est OpenInventor 4.0 commercialisé par TGS. Cette librairie, très complète par ailleurs, présente quelques défauts et quelques erreurs d'interprétation des fichiers VRML. Certaines ont pu être compensées lors des développements, d'autres n'ont pu l'être pour des raisons techniques liées à la manière dont fonctionne la librairie OpenInventor.

Le caractère technique de ces limitations impose de connaître la spécification du format VRML97 [X3D, url]. Nous allons maintenant détailler ces limitations.

### C.1.1 Problème d'import des noeuds de type Switch de la spécification VRML

Ce problème n'apparaît vraisemblablement que sous la version unix de OpenInventor. L'initialisation du graphe de scène n'est pas forcément compatible avec les paramètres donnés dans le fichier VRML. Le rendu de la première branche du switch n'est pas forcément effectué.

### C.1.2 Implémentation du mot clé *IS* de la spécification VRML :

Lors de la définition des protos VRML, certains champs des noeuds déclarés dans les protos sont identifiés à des champs globaux du proto. Ceci se fait au moyen du mot clé *IS* comme dans l'exemple ci-dessous :

```
PROTO blue [ exposedField SFColor diffColor .0 .0 1 ]
{
    Material {
        diffuseColor IS diffColor
    }
}
```

Ce mot clé impose que la modification d'un des champs soit reportée sur l'autre, et réciproquement. Or le système de fonctionnement de OpenInventor ne permet pas ce genre de choses. On ne peut répercuter les changements réciproquement. Il faut donc choisir une direction privilégiée. Par défaut OpenInventor considère l'expression *"A **IS** B"* comme étant : *"toute modification de B doit être répercutée sur A"*. Non seulement ce système est unidirectionnel, contrairement aux spécifications VRML, mais de plus la direction est mauvaise dans certains cas précis surtout lorsque les champs sont de type *eventOut* [1]. L'utilisation de la libraire OpenInventor ne nous permet pas de maintenir la notion de bidirectionnalité imposée par le mot clé $IS$. Nous sommes donc obligés de retourner le sens par défaut dans le cas où l'un des éléments est un champ de type *eventOut*.

En résumé, le mot clé $IS$ est traité de manière particulière. L'interprétation de celui-ci qui est différente de celle de la spécification VRML est la suivante :

| *"A **IS** B"* se lit : | |
|---|---|
| Si $B$ est un champ de type $eventOut$ | → *"toute modification de A est répercutée sur B"* |
| Sinon | → *"toute modification de B est répercutée sur A"* |

### C.1.3 Restrictions des possibilités de routage dans un fichier VRML

Le standard VRML permet de créer ce que l'on appelle des routes. Une route se présente sous cette forme :

***ROUTE** noeud1.champA **TO** noeud2.champB*

Ceci signifie : *"toute modification du champA du noeud1 est répercutée sur le champB du noeud2"*. Ici *"noeud1.champA"* représente la source et *"noeud2.champB"* représente la destination de la route. En VRML, plusieurs routes peuvent être connectées sur le même champs. Dans ce cas, nous distinguons deux configurations :

– les *"fan-in"* : plusieurs routes sont à destination du même champ,
– les *"fan-out"* : plusieurs routes ont le même champ à la source.

OpenInventor ne pose pas de problème dans le cas de figure où nous sommes en présence d'un fan-out dans un fichier VRML. Malheureusement, il s'avère incapable de gérer correctement un fan-in. Il ne conserve qu'une seule route : la première rencontrée. Dès lors, le seul moyen de palier à ce problème est d'employer des scripts javascripts pour y parvenir

### C.1.4 Restrictions sur l'implémentation des protos externes

Bien que la possibilité d'utiliser des protos VRML externes soit implémentée dans notre application, ces fonctionnalités n'ont pas été testées. Il est donc possible d'avoir des dysfonctionnements du widget de mixage lors de l'utilisation de ces fonctionnalités avancées.

## C.2 Limitations JavaScript

L'interpréteur Javascript employé est NJS-2.0.5 qui est une librairie distribuée sous licence LGPL. Cette librairie est issue du moteur d'interprétation Javascript d'une ancienne version de Netscape qui est depuis tombée dans le domaine public. La version Javascript interprétée est donc relativement ancienne.

Lorsque des objets Javascript sont créés, il est normalement possible d'accéder à chacun des champs de cet objet de manière implicite par des indirections sous forme de tableau, d'après les spécifications. Ainsi, si un objet *objetLambda* a un premier membre *membreA*, puis d'autres membres

[1] Un proto VRML a des champs dits *eventIn* qui sont en quelque sorte des paramètres pour les traitements effectués dans le proto. Pareillement, certains de ces champs sont dits *eventOut*. Ce sont alors les résultats des traitements effectués dans le proto.

(*membreB* jusqu'à *membreX*), on peut désigner *objetLambda.membreA* (qui est la méthode explicite) par *objetLambda[0]*, *objetLambda.membreB* par *objetLambda[1]* et ainsi de suite.

L'interpréteur Javascript employé n'est pas conforme à cette spécification. Il faut donc toujours adopter la manière explicite de référencement des membres d'un objet Javascript.

## C.3 Statut de l'implémentation Javascript pour les types VRML

Pour les besoins de notre démonstrateur, il n'a pas été nécessaire de faire une implémentation totale de cette spécification. Nous allons donc exposer dans le détail ce qui a été implémenté et ce qui reste à faire pour devenir conforme à la spécification [Vrml, 1997]. Pour faciliter la lecture, lorsque l'objet ou la méthode est :

- implémentée, elle est précédée d'un carré vert ■ ,
- partiellement implémentée, elle est précédée d'un triangle orange ▲ ,
- non implémentée,elle est précédée d'une étoile rouge ★ .

### C.3.1 Types simples implémentés

Ce sont les types de base. Ils ne comportent pas de méthodes spécifiques. La liste de ces types est la suivante :

- ■ `SFBool`
- ■ `SFInt32`
- ■ `SFFloat`
- ■ `SFTime`

#### C.3.1.1 Fonctions générales liées à l'interpréteur

Les scripts Javascripts inclus dans VRML doivent s'exécuter dans un contexte appelé *Browser*. Ce contexte possède ses méthodes propres qui peuvent être appelées. Ces dernières n'ont pas été implémentées par nos soins puisqu'elles ne nous étaient d'aucune utilité immédiate. La liste de ces méthodes est :

- ★ `String getName()`
- ★ `String getVersion()`
- ★ `numeric getCurrentSpeed()`
- ★ `numeric getCurrentFrameRate()`
- ★ `String getWorldURL()`
- ★ `void replaceWorld(MFNodes nodes)`
- ★ `void createVrmlFromURL(MFString url, Node node, String event)`
- ★ `void addRoute(SFNode fromNode,String fromEventOut, SFNode toNode, String toEventIn)`
- ★ `void deleteRoute(SFNode fromNode,String fromEventOut, SFNode toNode, String toEventIn)`
- ★ `void loadURL(MFString url,MFString parameter)`
- ★ `void setDescription(String description)`

#### C.3.1.2 Types complexes

La spécification VRML comprend des types qui correspondent à des champs uniques et d'autres qui correspondent à des champs multiples. Ces derniers sont en réalité des tableaux de champs uniques. Une partie des types complexes se référant à des champs uniques ont été implémentés. Voici le statut de l'implémentation réalisée pour chacun d'eux :

- ★ SFColor
- ★ SFImage
- ★ SFNode
- ▲ SFRotation
- ★ SFVec2f
- ■ SFVec3f
- ★ VrmlMatrix

Puisque seuls les types *SFRotation* et *SFVec3f* ont été partiellement implémentés, voici le statut détaillé de leur implémentation, par membre et par méthode, en commençant tout d'abord par le type *SFRotation* :

- ■ `new SFRotation(numeric x,numeric y, numeric z, numeric angle)`
- ★ `new SFRotation(SFVec3f axis, numeric angle)`
- ★ `new SFRotation(SFVec3f fromVector, SFVec3f toVector)`
- ■ `numeric x`
- ■ `numeric y`
- ■ `numeric z`
- ■ `numeric angle`
- ■ `SFVec3f getAxis()`
- ■ `SFRotation inverse()`
- ★ `SFRotation multiply(SFRotation rot)`
- ★ `SFVec3f multVec(SFVec3f vec)`
- ■ `void setAxis(SFVec3f vec)`
- ★ `SFRotation slerp(SFRotation dest, numeric t)`
- ■ `toString()`

Pour ce qui est du statut détaillé du type *SFVec3f*, le voici :

- ■ `new SFVec3f(numeric x, numeric y, numeric z)`
- ■ `numeric x`
- ■ `numeric y`
- ■ `numeric z`
- ■ `SFVec3f add(SFVec3f vec)`
- ■ `SFVec3f cross(SFVec3f vec)`
- ■ `SFVec3f divide(numeric n)`
- ■ `numeric dot(SFVec3f vec)`
- ■ `numeric length()`
- ■ `SFVec3f multiply(numeric n)`
- ■ `SFVec3f negate()`
- ■ `SFVec3f normalize()`
- ■ `SFVec3f substract(SFVec3f vec)`
- ■ `String toString()`

#### C.3.1.3 Types multiples

Les types multiples n'ont pas bénéficié d'une implémentation. Pour information, voici la liste des types multiples contenus dans la spécification :

- ★ MFColor
- ★ MFFloat
- ★ MFint32
- ★ MFNode

- ★ `MFRotation`
- ★ `MFString`
- ★ `MFTime`
- ★ `MFVec2f`
- ★ `MFVec3f`

# Contributions à la dextérité d'un système de réalité augmentée appliqué à la maintenance industrielle.

**Résumé :**

La dextérité d'un système de Réalité Augmentée (RA) mobile appliqué à la maintenance industrielle repose sur sa flexibilité, sa précision et sa robustesse intrinsèque face aux verrous scientifique et technologique. Nous introduisons d'abord une architecture orientée composants flexible, innovante et satisfaisant les contraintes de temps réel des systèmes de RA. Sur cette architecture est bâti un système de localisation par la vision utilisant des cibles codées. De nouveaux algorithmes d'estimation de pose sont proposés. Notre architecture comporte un Système de Gestion des Augmentations (SGA) utilisant des procédures de maintenance numériques. Chacune est liée à des documents multimédia l'illustrant, dont des modèles 3D animés. Le projet RNTL-Assistance à la Maintenance en RA intègre ce SGA. Enfin, le système est élargi à la RA en vision directe pour laquelle nous combinons une technique de prédiction du point de vue basée sur le filtre particulaire et une méthode de post-rendering.

**Mots clés :**

Réalité augmentée, maintenance industrielle, architecture orientée composants, suivi basé marqueurs, augmentations visuelles contextuelles, prédiction du point de vue.

# Contributions to the dexterity of an augmented reality system applied to industrial maintenance.

**Abstract :**

The dexterity of a mobile Augmented Reality (AR) system applied to industrial maintenance is relying on flexibility, accuracy and robustness to overcome scientific and technological locks. First, we introduce a novel component based software architecture satisfying the real-time requirements of AR systems. On top of this framework is built a localization system using computer vision and relying on coded fiducials. New pose estimation algorithms are performed. Our architecture is embedding an Augmentation Manager (AM) using numerical maintenance procedure. Each one of them is linked to multimedia documents, including specifically animated 3D-models. This AM was integrated in the AMRA (French acronym of Maintenance Assistance using AR) RNTL project. Last, this system is extended to optical see-through AR in which we combine a viewpoint prediction technique based on particle filtering and a post-rendering method.

**Keywords :**

Augmented reality, industrial maintenance, component-based software architecture, maker based tracking, visual contextual augmentations, viewpoint prediction.