Uracode, un logiciel de vision industrielle robuste et temps réel

par Philippe GUILLEMANT

Adapté à tous les fonds et objets naturels, Uracode s'articule autour d'une bibliothèque informatique qui remplit les fonctionnalités permettant de donner le sens de la vision à un robot dans des conditions dynamiques et d'ambiance perturbée.

Philippe GUILLEMANT est chercheur au CNRS. Il est l'auteur d'Uracode et a créé la société Uratek qui en réalise le transfert vers l'industrie.

1. Problématique résolue

Depuis les années 1980, le traitement d'images a commencé à être exploité dans l'industrie pour répondre à des tâches d'automatisation et de contrôle. Depuis lors, plusieurs générations d'algorithmes se sont succédé pour rendre les machines de vision de plus en plus robustes et fiables. Bien que les algorithmes mathématiques se soient considérablement améliorés [1] et que des approches sans cesse plus intelligentes aient été introduites, de la corrélation normalisée aux réseaux de neurones en passant par les caractérisations par invariants locaux, on ne savait toujours pas bien résoudre jusqu'à récemment le problème de la reconnaissance et de la localisation en temps réel d'objets en mouvement ou placés dans le champ de vision d'une caméra elle-même en mouvement, sans éviter le recours à des hypothèses simplificatrices contraignantes telles que :

— conditions d'éclairage uniforme ou maîtrisées ;

— positionnement de mires sur l'objet ;

— utilisation de raies laser ;

— recherche de transitions contrastées ;

— géométrie particulière connue a priori ;

— isolement de l'objet du fond (tapis non texturé, etc.) ;

— maintien d'une distance fixe à la caméra, etc.

Ces hypothèses simplificatrices ont été l'une des principales contraintes dans la conception des machines de vision industrielle, qui offrent des enceintes isolées des perturbations et assurent une prise d'image dans des conditions bien maîtrisées, par déclenchement mécaniquement asservi de la prise de vue. Ces enceintes hébergent également l'électromécanique pour l'automatisation.

Les causes des difficultés qui justifient ces précautions soignées au niveau de la prise de vue sont pourtant bien connues. Durant son mouvement, un objet apparaît sous différents aspects qui modifient invariablement la structure de ses éléments caractérisants, essentiellement des contours ou zones de contrastes. Indépendamment de tout mouvement, la variation des conditions d'éclairage, en ambiance extérieure par exemple, crée un effet perturbateur additionnel. Au total, les causes de déformation de ces éléments caractérisants, ou primitives visuelles, sont nombreuses :

— les déformations dues aux conditions ambiantes, telles que les variations d'éclairage, leurs hétérogénéités, les reflets, les ombres portées... ;

— les déformations géométriques dues au déplacement de l'objet : ses variations angulaires, de distance, d'inclinaison, de position... ;

— les déformations optiques dues à la discrétisation en pixels et aux non-linéarités du capteur, qui modifient sensiblement l'aspect de l'objet en fonction de son angle et de sa distance angulaire au centre de l'image... ;

— les occultations partielles de l'objet par d'autres objets ;

— les déformations intrinsèques de l'objet, qu'il s'agisse d'un même objet réellement déformable ou de plusieurs échantillons légèrement différents...

Précisons que le terme « déformation » est ici entendu au sens large et s'applique non à l'objet mais à l'ensemble des primitives visuelles qui caractérisent son image. La figure **1** illustre différents types de déformations courantes de contours fermés, primitives utilisées dans Uracode.

L'approche théoriquement la plus adéquate pour caractériser un objet visuel de façon indépendante des sources de déformation de ses primitives implique la connaissance a priori des paramètres et fonctions qui permettent de modéliser ces sources, principalement :

— la connaissance a priori de la géométrie 3D de l'objet et de tous ses déplacements (6D) ;

— la connaissance a priori de la répartition angulaire et spatiale des luminances ;

— la connaissance des paramètres des caméras (focale, taille des pixels...).

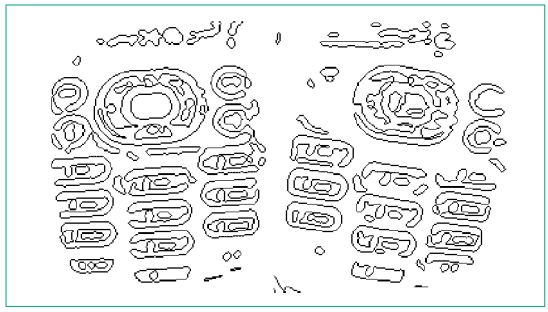

Figure 1 – Déformations des primitives visuelles (ici des contours fermés) d'un téléphone portable légèrement déplacé devant la caméra

Dans l'état actuel de la recherche en vision artificielle, les solutions intéressantes restent donc très académiques. Inversement, si l'on considère les travaux qui tentent l'évitement de toute connaissance a priori sur l'objet, la reconstruction automatique 3D de la géométrie de l'objet bute sur deux écueils : les contraintes temps réel et la complexité des objets [2].

Uratek
http://www.uratek.fr

IUSTI : Institut universitaire des systèmes thermiques et industriels (UMR CNRS 6595)

La capacité de satisfaire ces deux contraintes est la qualité première de la méthode du « **plongement fractal** » [3] qui est au cœur du logiciel Uracode. Cette méthode a été inventée en 1997 [4] par Philippe Guillemant, chercheur au CNRS, auteur d'Uracode et créateur de la société Uratek qui le transfère dans l'industrie sous licence CNRS-IUSTI. Le « plongement fractal » a tout d'abord été appliqué à la caractérisation d'objets réellement déformables tels que des sources de fumées, pour élaborer un procédé efficace de détection des feux de forêts [5].

Son principe est le suivant : chaque primitive visuelle est indexée par un nombre entier unique à 32 ou 64 bits qui représente l'ordre de rencontre d'un point sur une courbe fractale qui remplit l'espace multidimensionnel de représentation de la primitive. Pour caractériser une primitive, on utilise en effet habituellement un ensemble de coordonnées qui quantifient chacune une grandeur caractéristique de cette primitive. Dans le cas d'un contour, ces grandeurs peuvent être par exemple :

— le périmètre ;
— la surface ;
— l'excentricité ;
— le nombre d'angularités (coins) ;
— le diamètre moyen ;
— les distances extrêmes au centre d'inertie, etc.

On rassemble ainsi aisément pour chaque primitive un ensemble de 8 coordonnées au moins, avec pour chacune d'entre elles une quantification de 4 bits au moins, de façon à référencer un point dans un espace 8D avec au moins 32 bits de définition. La méthode du « plongement fractal » telle qu'elle est implémentée dans le logiciel Uracode consiste alors à :

— calculer l'index (ou rang) de ce point sur une courbe fractale de Hilbert (figure **2**) ;

— classer cet index selon une chaîne d'index qui relie directement chaque élément aux index suivant et précédent sur la chaîne.

Cette dernière opération est rendue très rapide en utilisant une troncation adéquate de l'index permettant un accès direct à la petite portion de chaîne dans laquelle le nouvel élément doit être inséré.

L'intérêt de l'utilisation d'une courbe fractale est de rendre la méthode particulièrement appropriée à la caractérisation d'objets déformables (au sens large) : les déformations ont en effet pour résultat de modifier sensiblement certaines coordonnées de primitive en provoquant un petit déplacement de son point représentatif dans l'espace de plongement. En conséquence, les différents points d'une primitive d'un même objet associés aux différentes images de l'objet engendrent dans cet espace un voisinage de points. En réalité, plusieurs voisinages de points sont créés à cause de différentes sources de discontinuité qu'il serait trop long de recenser ici ; néanmoins, chacun d'entre eux contient un nombre résiduel suffisant de points à distance réduite de leur centroïde. C'est là que les propriétés fractales de la courbe de Hilbert interviennent.

Contrairement à ce qui se passerait si l'on utilisait un index cartésien à 32 ou 64 bits semblable à une adresse mémoire obtenue à partir de l'empilement de chaque coordonnée à 4 bits, l'index de Hilbert varie très peu lorsque le point reste proche de son centroïde de voisinage, quelle que soit la taille de ce voisinage. Cela est dû d'une part aux propriétés d'autosimilarité de cette courbe fractale, d'autre part à l'absence de discontinuités le long de cette courbe, qui traverse l'espace en respectant une distance unitaire entre chaque point.

Outre les objets déformables, la méthode du « plongement fractal » est particulièrement adaptée aux objets complexes [6] parce que leur caractérisation se fait selon un processus heuristique qui avantage naturellement les primitives pertinentes par rapport aux primitives non pertinentes : les primitives pertinentes engendrent des voisinages de points denses et peuplés, ce qui augmente la probabilité pour qu'une primitive pertinente serve de référence à l'identification et à la localisation d'une nouvelle primitive. La méthode est ainsi autoadaptative.

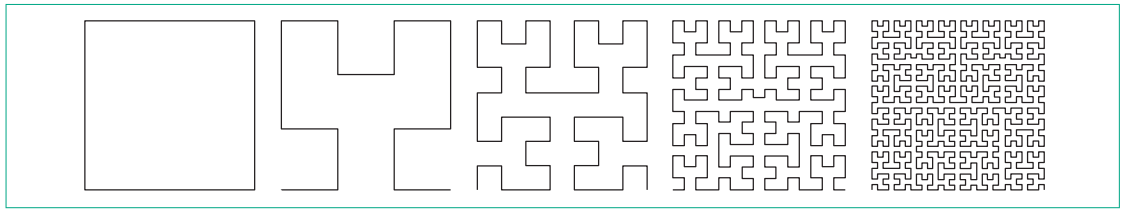

Figure 2 – Processus de construction de la courbe de Peano-Hilbert en dimension 2

Enfin, il faut insister particulièrement sur le caractère ultrarapide de l'algorithme du plongement fractal, qui provient du fait que pour identifier et localiser une primitive nouvelle (issue de la dernière image acquise), il n'est pas nécessaire de comparer son point représentatif à tous les autres points de l'espace, dont le cardinal atteint couramment des dizaines ou même des centaines de milliers de points : il suffit d'affecter à ce nouveau point l'identificateur et les coordonnées du point le plus proche par comparaison à seulement *deux points* : le précédent et le suivant sur la chaîne.

C'est ce qui confère à la méthode toute sa puissance et une capacité d'identification surprenante [7] car souvent supérieure à celle de l'œil humain, notamment pour certains types d'objets. Par exemple, la figure **5** illustre le suivi d'un élément de moquette après rotation : l'œil nu est incapable de réussir la même prouesse sans une observation très attentive.

2. Mise en œuvre du logiciel

Le logiciel Uracode est composé d'un kit de développement pour différents compilateurs (Borland C++, Visual C++...), contenant différents projets offrant chacun un prototype de développement pour différentes applications. Au cœur de chaque programme, on retrouve l'instruction suivante, qui constitue l'unique appel à la bibliothèque Uracode (dll sous Windows), rendant sa programmation extrêmement simple :

```
UCI = Uracode (UC_command, UC_in, UC_out, UC_mouse) ;
```

Cette instruction s'insère à seulement deux endroits dans le programme source du prototype :

— dans la procédure d'initialisation, en début de programme ;

— dans la boucle d'acquisition temps réel (*callback* appelé par l'acquisition d'image).

Les quatre paramètres de la fonction Uracode() définissent respectivement les commandes, entrées, sorties et états de la souris (cette dernière à destination du laboratoire d'essais UCI).

Les spécifications du logiciel Uracode sont résumées dans le tableau **1**.

2.1 Commandes

Chaque commande est reliée à une fonctionnalité bien précise ; parmi celles qui sont proposées, deux sont des actions instantanées et quatre correspondent à des changements d'état (figures **3** et **4**). Elles sont générées par le paramètre UC_command, entier pouvant prendre les six valeurs suivantes.

■ UC_INIT initialise le composant, affecte des valeurs par défaut aux paramètres d'entrée et définit le format de l'image (résolution, mode couleurs ou niveaux de gris).

■ UC_ANALYSE place Uracode dans un mode de réglage facultatif des paramètres d'entrée où l'on peut définir les zones de l'image à apprendre. Ces zones peuvent être programmées par l'utilisateur, extraites automatiquement par Uracode ou encore définies à l'aide de la souris dont les coordonnées sont alors entrées au moyen du paramètre UC_mouse.

■ UC_LEARN place Uracode dans un état d'apprentissage de la zone de l'image à apprendre. Pendant l'apprentissage, l'objet appris ou la caméra peuvent se déplacer car l'objet est reconnu et localisé dès la deuxième image. Il est donc possible, et même recommandé, de provoquer des variations d'aspect de l'objet pendant l'apprentissage en modifiant l'inclinaison, la position, l'éclairage de l'objet, etc., ce qui rend la détection plus robuste. La durée d'apprentissage peut varier d'une seconde à une minute selon la diversité des variations d'aspect induites durant cette phase.

Tableau 1 – Caractéristiques techniques d'Uracode

Temps de cycle d'apprentissage	15 ms (base : Pentium 4, 1,5 GHz)
Temps de cycle de détection	25 ms (base : Pentium 4, 1,5 GHz)
Temps de cycle d'identification et de recalage	35 ms (base : Pentium 4, 1,5 GHz)
Précision du recalage angulaire	De 0,04° à 0,2° suivant l'objet
Précision du recalage spatial	De 1/16 de pixel à 1/4 de pixel suivant l'objet
Tolérance aux occultations	50 % à 90 % de l'objet (1)
Orientation des objets	Indifférente : de 0 à 360°
Inclinaison des objets	Jusqu'à 40° en détection, 20° en localisation
Forme des objets	Aucune limite de complexité (1)
Variations de distance	100 % en détection, 30 % en localisation
Variations d'éclairage	Limite de visibilité à l'œil nu
Taille des objets	De 4 % à 400 % de l'image
Systèmes	Windows 98/NT/2000/XP
Compilateurs	Visual C++, Borland C++... (dll)

(1) La complexité et la taille de l'objet accroissent les performances.

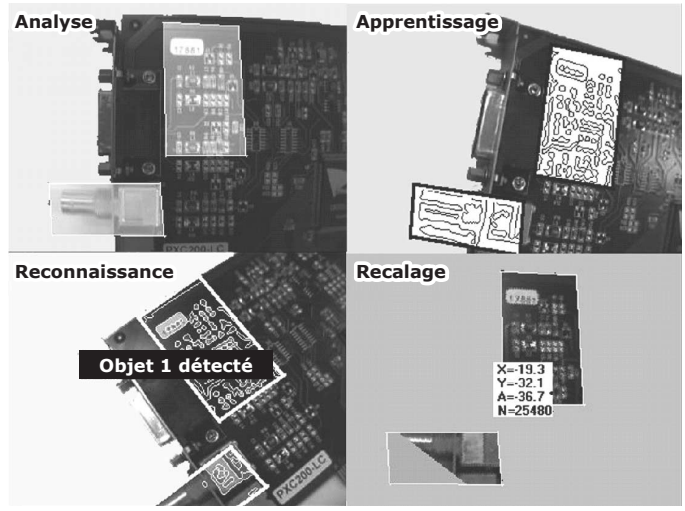

Figure 3 – Les quatre principales fonctionnalités / états d'Uracode

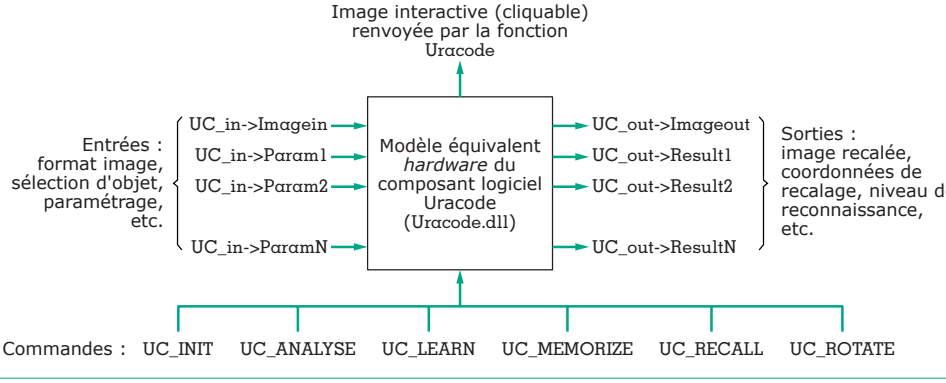

Figure 4 – Schéma équivalent *hardware* d'Uracode

■ UC_MEMORIZE mémorise l'objet appris dans un fichier dont la taille varie de 10 à 100 ko en fonction du paramètre d'entrée UC_in->Nbneuro. Ce paramètre est égal au nombre maximum de neurones à utiliser pour identifier l'objet. Chaque neurone est en fait constitué d'un maillon de la chaîne d'identificateurs fractals. Ces maillons sont appelés neurones car ils concentrent toutes les primitives d'un même voisinage, de façon tout à fait similaire à un neurone de position.

■ UC_RECALL place Uracode dans un état de veille pendant lequel la base d'objets mémorisés est parcourue à chaque pas de temps, jusqu'à ce qu'un objet présent dans le champ de la caméra soit reconnu. Une fois reconnu, l'objet est suivi en temps réel dans tous ses déplacements et les sorties d'Uracode fournissent les coordonnées de l'objet.

■ UC_ROTATE est une option des modes UC_RECALL et UC_LEARN qui fournit en sortie l'image de l'objet recalé en translation rotation.

2.2 Entrées

La structure d'entrée UC_in d'Uracode contient une vingtaine de paramètres ayant des valeurs par défaut optimales pour une grande variété d'objets. Leur affectation est donc facultative. Néanmoins, les paramètres suivants sont très utiles.

■ UC_in->Imagein, UC_in->Width, UC_in->Height et UC_in->Imagebits définissent l'image à traiter et son format (largeur et hauteur en pixels, couleurs ou nuances de gris).

■ UC_in->Method et UC_in->Threshold sont des paramètres de bas niveau qui influent essentiellement sur la quantité des primitives.

■ UC_in->Maxneuro définit la taille de la mémoire réservée à l'objet, qui doit être d'autant plus élevée que les variations d'aspect, provoquées durant l'apprentissage, sont importantes.

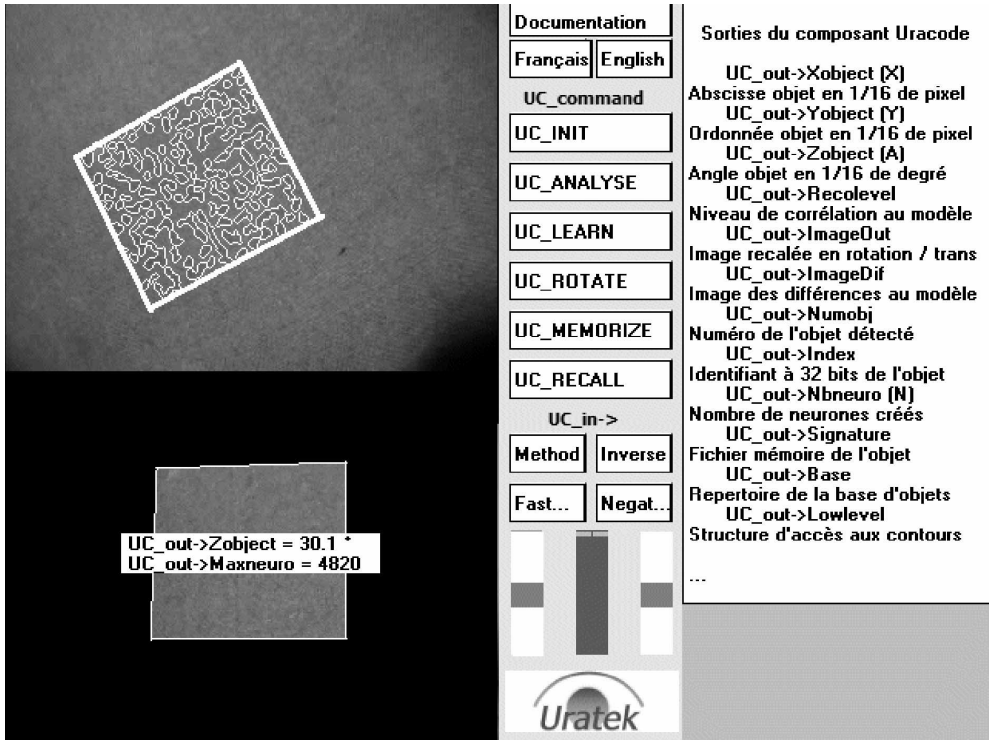

Figure 5 – Interface UCI du composant Uracode illustrant la localisation et le *tracking* d'un élément de moquette (indétectable à l'œil nu) pendant la rotation de la caméra

■ UC_in->Tolerance définit le seuil de tolérance aux variations d'aspect.

■ UC_in->Outline et UC_in->Mask agissent sur la définition des contours de l'objet à apprendre, qui peut être automatique ou programmée par l'utilisateur.

■ UC_in->Signature, UC_in->Label, UC_in->Numobj, UC_in->Firstobj... servent au paramétrage de la base d'objets : noms de fichiers et d'objets, nombre d'objets...

2.3 Sorties

Les sorties d'Uracode, au nombre d'une dizaine, sont obtenues par la lecture des résultats UC_out->ResultN dont la structure est fournie en paramètre d'entrée.

■ UC_out->Xobject, UC_out->Yobject et UC_out->Zobject contiennent respectivement le déplacement horizontal et vertical de l'objet, en 1/16 de pixel, et son angle de rotation en 1/16 de degré.

■ UC_out->Recolevel donne le niveau de correspondance de l'objet aux modèles appris.

■ UC_out->ImageOut fournit l'image recalée en translation rotation.

■ UC_out->Nobjdet fournit le numéro de l'objet détecté (– 1 en l'absence d'objet).

2.4 Laboratoire d'essais

Le composant logiciel Uracode contient un laboratoire d'essais interne, nommé UCI, activé par simple affichage de l'image cliquable renvoyée par la fonction Uracode(). En plus des sorties UC_out->ParamN, Uracode renvoie en effet une image illustrant les processus en cours (figure **5**). Cette image devient interactive lorsque l'utilisateur entre les coordonnées de la souris dans le dernier paramètre d'Uracode. Elle remplit alors la fonction d'un « laboratoire d'essais » pour le programme qui l'affiche dans son interface graphique IHM, sans aucune autre programmation à effectuer. Un clic sur cette image peut provoquer le déclenchement d'une commande, le réglage de certains paramètres et la visualisation de commentaires ou de la documentation relative à la commande en cours.

UCI : Uratek Component Interface
IHM : interface homme-machine

3. Applications

Les fonctionnalités d'Uracode proches de la vision humaine et son adaptation à toutes sortes d'objets, textures et scènes notamment extérieures, en font un composant logiciel généraliste dont les applications sont très nombreuses. On peut en différencier deux types :

— les **applications immédiates** (§ 3.1), qui ne nécessitent pas d'analyse d'image complémentaire et peuvent être testées directement à partir du laboratoire d'essais UCI ;

— les **applications dérivées** (§ 3.2), qui nécessitent une utilisation plus experte et/ou un complément d'analyse d'image. Rentrent par exemple dans cette catégorie les applications pour lesquelles la forme enveloppante de l'objet à apprendre (pourtour de l'objet) varie dans le temps et pour lesquelles l'extracteur dynamique interne à Uracode est insuffisant.

Parmi les applications immédiates d'Uracode, on peut citer quelques produits réalisés ou projets en cours à partir de ce logiciel :

— **application médicale** (produit 6D-VOM) : la société CCA Biodigital (groupe Amplifon) utilise Uracode pour calculer les mouvements des yeux et de la tête (six composantes) à partir d'un capteur solidaire de la tête équipant les patients atteints de troubles ORL, l'intérêt médical d'un tel capteur résidant dans l'exploration des vertiges et des troubles de l'équilibre ;

— **application en réalité augmentée** (produit D'Fusion) : la société Total Immersion exploite Uracode pour insérer des images virtuelles dans des reportages filmés en direct lorsque la caméra est mobile, les calculs de mouvements de caméra (*yaw*, *pitch*, *roll*) nécessaires à cette opération étant effectués par Uracode et permettant d'éviter les traitements en postproduction ;

— **application en vidéosurveillance** (projet Fabrice) : la société Thalès met en œuvre Uracode pour détecter des visages en mode non coopératif, les différents aspects sous lesquels un visage particulier peut être mémorisé étant automatiquement couverts par la fonction UC_LEARN d'Uracode activée pendant le mouvement de l'individu.

3.1 Exemples d'applications immédiates

■ **Robotique industrielle** : tri et manipulation de pièces manufacturées, palettisation, dépalettisation... Ces applications utilisent essentiellement les coordonnées de la pièce identifiée (x, y, θ) pour la localiser sur une chaîne, un support, un conteneur..., afin d'orienter un processus de tri ou de manipulation (bras robotisé). Par rapport aux techniques existantes, Uracode peut s'affirmer dans les contextes imposés (naturels ou environnements à risques : intervention dans les centrales nucléaires) où la maîtrise des conditions ambiantes est difficile.

■ **Contrôle de qualité** : l'indicateur de niveau UC_out->Recolevel est utilisé comme seuil pour définir le degré de concordance au modèle au-dessous duquel l'objet doit être rejeté. La fonctionnalité de recalage peut servir en complément lorsque l'on recherche un diagnostic de présence ou d'absence d'un détail décisif sur l'objet.

■ **Identification d'objets** (logos, visages, paysages, photographies...) : la particularité de la fonction identifiante d'Uracode est l'acceptation de la mobilité de la caméra et/ou de l'objet et la fonction *tracking* qui lui est associée. L'objet est alors reconnu et suivi en temps réel durant ses déplacements relatifs. La fonctionnalité de recalage permet de restituer une imagette toujours correctement centrée autour de l'objet.

■ **Exploitation des mouvements de caméra** : les domaines concernés sont variés : réalité augmentée avec l'insertion d'objets virtuels dans des images réelles, reconstruction panoramique de scène balayée par une caméra, etc.

3.2 Exemples d'applications dérivées

■ **Vidéosurveillance** : le sujet de la vidéosurveillance suppose une extraction dynamique adaptée à l'événement à détecter. Selon la nature de l'événement, il pourra s'agir d'une fenêtre centrée sur un point significatif ou d'une enveloppe extraite via un prétraitement approprié visant à séparer l'objet du fond. Uracode propose une extraction de tous les éléments mobiles, il reste à choisir la zone pertinente.

■ **Analyse du trafic routier** : il s'agit d'un cas particulier de vidéosurveillance où Uracode peut être utilisé pour « accrocher » les véhicules et assurer un suivi robuste aux conditions ambiantes (effets d'ombre, variations de luminosité, transition jour-nuit...).

4. Conclusion

La diversité des applications, les fonctionnalités temps réel, la simplicité, la souplesse et la facilité de détection et de localisation, dès le début même de l'apprentissage d'un objet, font d'Uracode un logiciel qui surprend réellement : en premier lieu par sa capacité à concurrencer l'œil humain, en second lieu parce que nous sommes habitués depuis des décennies à entendre parler de technologies « dites » intelligentes telles que les réseaux de neurones, mais que nous nous sommes avec le temps accoutumés à les considérer comme immatures, car rien n'est jamais venu modifier notre vie quotidienne en ce domaine depuis qu'elles sont annoncées.

C'est malgré tout oublier qu'un certain temps est nécessaire à l'industrialisation de toute invention, a fortiori lorsqu'elle vient modifier nos idées reçues, et surtout lorsqu'elle démarre en France et non outre-Atlantique. S'agissant de se substituer à l'œil humain pour certaines tâches, tout un chacun est en mesure de douter des performances d'algorithmes de vision, s'agissant d'analyser à notre place des informations visuelles pertinentes qu'il convient d'exploiter pour détecter un objet. En dehors des films de science-fiction, nous ne sommes pas du tout prêts à accepter aisément les potentialités concurrentes d'une « brique de code ».

Pourtant, certains indices ne trompent pas :

— Uracode se bonifie lorsque la complexité de l'objet visuel augmente, ce qui témoigne de facultés d'adaptation qui n'ont rien à voir avec le cadre classique de la vision où reste indispensable l'injection a priori d'hypothèses sur l'objet (couleur, forme, etc.) ;

— Uracode est tout à fait le contraire d'une « usine à gaz » avec quantité de réglages pour parvenir à un résultat : sa programmation est extrêmement simple, une seule ligne de programme suffisant à tester un nouveau prototype d'application ;

— Uracode est exploité de façon optimale lorsque ses calculs sont effectués dans un intervalle temporel inférieur à 20 ms, ce qui réserve à l'application la moitié du temps de cycle dans des conditions d'analyse en temps réel (25 images/s).

Ces caractéristiques paradoxales témoignent bien de la nouveauté. Nous sommes en face d'une « brique de code » qui défie les performances de la vision humaine, à la fois en matière de facilité de reconnaissance et d'aptitude à reconnaître des objets ou événements très complexes, et tout ceci dans un temps de cycle analogue à celui du cerveau visuel (40 Hz, soit 25 ms). Cela mérite le détour, de préférence avant que nos cousins d'outre-Atlantique ne s'en rendent compte.

Bibliographie

Références

[1] FAUGERAS (O.), LUONG (Q.T.) et PAPADOPOULO (T.). – *The Geometry of Multiple Images*. MIT Press (2001).

[2] ESPIAU (F.-X.) et RIVES (P.). – *Extracting robust features and 3D reconstruction in underwater images*. Int. Conf. OCEANS MTS/IEEE 2001, Hawaii, États-Unis (2001).

[3] GUILLEMANT (P.) et VICENTE (J.). – *Analyse automatique de scènes dynamiques complexes par la méthode du plongement fractal*. Traitement du signal, *17*, n° 5/6, pp. 463-477 (2000).

[4] GUILLEMANT (P.) et REY (M.). – *Procédé d'analyse d'un signal - Fractomètre Imageur*. Brevet d'invention CNRS n° 9710455 (12 août 1997).

[5] GUILLEMANT (P.) et VICENTE (J.). – *Real time identification of smoke images by clustering motions on a fractal curve with a temporal embedding method*. Optical engineering, *40*, n° 4, pp. 554-563 (2001).

[6] VICENTE (J.) et GUILLEMANT (P.). – *Complex natural scene analysis with the fractal embedding method.* Symposium On Advanced Concepts For Intelligent Vision Systems, ACIVS'00, Baden-Baden, pp. 114-118 (2000).

[7] *Le logiciel reconnaît et localise en temps réel toutes les formes*. Revue Mesures (juin 2003).

Dans les Techniques de l'Ingénieur

PAINDAVOINE (M.). – *Traitement des images en temps réel. Applications industrielles*. [R 6 720], traité Mesures mécaniques et dimensionnelles (2000).