Institut National Polytechnique de Toulouse

Ecole Nationale Supérieure d'Electrotechnique, d'Electronique, d'Informatique et d'Hydraulique de Toulouse

Filière INFORMATIQUE

Rapport de fin d'étude

PROJET R.E.V.E.R.I.E:

Contribution à la segmentation et à la représentation symbolique des régions pour le système de vision bas niveau.

Vincent RODIN

Juin 1989

Je tiens à remercier tous les membres du laboratoire "Vision par Calculateur ANDRE BRUEL" et plus particulièrement Alain, César, Charlie, Georges, Jean-Claude et Zoubir pour leurs précieux conseils

Sommaire

I Introduction

Le projet R.E.V.E.R.I.E. (REasonnig Vision Engineering in Robotic Industrial Environment) mené en collaboration avec le laboratoire du professeur LUCIA VAINA du Boston University doit permettre de concevoir un robot capable de mener à bien des tâches industrielles.

Mon stage s'intégrait dans la partie vision du projet. Des primitives de bas niveau ont été réalisées pour extraire des attributs de l'image de la scène à analyser (régions et contours).

Ces attributs ne permettent généralement pas d'entreprendre une reconnaissance des objets de la scène.

Un système expert est alors utilisé pour compléter et améliorer les attributs de l'image en vue d'une reconnaissance des objets s'y trouvant.

Le stage a été effectué dans le laboratoire "Vision par Calculateur **ANDRE BRUEL**" de l'**ENSEEIHT** en vue de l'obtention du diplôme d'ingénieur option informatique.

MOTS CLES: Vision, Traitement d'images et Système Expert

II Le laboratoire "Vision par Calculateur ANDRE BRUEL" et le projet R.E.V.E.R.I.E.

II-1 Le laboratoire "Vision par Calculateur ANDRE BRUEL"

L'équipe vision par calculateur A.BRUEL appartient au laboratoire de **Langages et Systèmes Informatique** (L.S.I), composante de l'**Institut de Recherche en Informatique de Toulouse** (I.R.I.T.) associé au CNRS (UA 347).Elle développe ses activités de recherche dans les trois axes principaux suivants:

. Recherche au plan fondamental, portant principalement sur l'étude des mécanismes de la vision.

. Recherche au plan des capteurs, sur l'étude et la réalisation de capteurs intelligents intégrés, pour la robotique.

. Recherche finalisée avec de fortes collaborations industrielles, qui ont abouti à plusieurs transferts industriels et de nombreux brevets internationaux.

Parmi les résultats très marquants, au niveau innovations, nous pouvons citer:

- Robot pour aveugles (Projet Delta),
- Micro caméra MICAM pour la robotique (la plus petite du monde),
- Machine à numériser automatiquement les logos,
- Machine d'ébavurage automatique de cuirs,
- Capteur de blancheur intelligent pour le contrôle en continu du talc...

Certains de ces projets mettent en scène un système de raisonnement à partir de techniques d'intelligence artificielle. L'équipe, persuadée de l'intérêt de tels outils dans le domaine de la vision par calculateur appliquée au domaine industriel, s'oriente vers des techniques d'intelligence artificielle, dans la définition et la conception de machines de vision haut niveau.

La collaboration avec le laboratoire du professeur Lucia VAINA, initialisée en 1987, porte principalement sur cet aspect (projet R.E.V.E.R.I.E.) .

II-2 Le projet R.E.V.E.R.I.E.

Le travail de recherche que nous menons actuellement avec le laboratoire du professeur Lucia VAINA du Boston University, s'inscrit dans la cadre d'un projet de réalisation d'un robot capable de mener à bien des tâches industrielles. Les deux atouts majeurs dont disposera le robot sont la **Vision** et le **Raisonnement**.

La plupart des robots opérationnels à ce jour exécutent des tâches répétitives dont la connaissance a été soit entièrement fournie au préalable, soit acquise au cours d'une phase d'apprentissage.Peu de ces robots industriels sont capables de mettre en jeu et d'exploiter une forme de raisonnement, lorsqu'une situation nouvelle se présente; le plus souvent, ils se contentent de détecter l'anomalie et de le signaler à l'opérateur.

Le raisonnement peut alors intervenir dans plusieures phases du fonctionnement du robot:

- d'une part sur les plans d'actions,afin d'offrir une autonomie suffisante au robot, pour déterminer la conduite à tenir,

- d'autre part, sur la vision, afin d'interpréter les scènes industrielles 3D, à partir de connaissances approximatives des objets et des scènes , par raisonnement sur une ou plusieures images.

II-2.1 Raisonnement et plans d'actions

Pour accomplir une tâche, un robot doit connaître la définition de la tâche (état de départ, état d'arrivée ou but à atteindre), les objets qu'il peut manipuler et de quelle façon il peut atteindre le but fixé.Il est donc nécessaire de fournir une description pour chacune des trois entités: objet, tâche, but.

La tâche est décrite comme une succession d'actions à accomplir pour passer d'un état de départ à un état final définissant le but à atteindre.
L'ordre dans lequel les différentes actions sont exécutées revêt une importance particulière et nécessite la mise en oeuvre de techniques de planification.De nombreux travaux on été réalisés dans le domaine du "planning", certains ont été effectivement implantés, mais la presque totalité concerne un environnement constitué de blocs ("blocs world").Citons pour mémoire NOAH, STRIPS ...

Le raisonnement dans ces applications ne s'applique qu'à l'ordonnancement des actions et les contraintes de réalisation ne portent que sur la faisabilité des mouvements élémentaires à chaque étape.

La différence essentielle entre notre projet et ces réalisations réside dans les objets manipulés; nous avons choisi des outils réalisés par l'homme (marteaux, tournevis, tenailles ...), au lieu d'objets simples comme les cubes.Nous supposons aussi que le robot qui manipulera ces objets possèdera une "main agile",c'est à dire non pas seulement 2 ou 3 doigts formant une pince mais une main semblable à celle de l'homme avec 5 doigts.Cette main et les logiciels adaptés sont en cours de réalisation dans le laboratoire américain.

De plus, les outils choisis sont susceptibles d'intervenir dans la réalisation de plusieures actions et d'être manipulés de différentes manières.Chaque outil possède une fonction principale qui le désigne comme outil idéal pour une action donnée, mais ses caractéristiques géométriques et physiques permettent d'envisager d'autres utilisations: une paire de tenailles peut servir à enfoncer une punaise par exemple.

Enfin, la partie utile de l'objet et la manière de le tenir dépendent de l'action elle-même: on ne tient pas un tournevis de la même manière pour visser que pour enfoncer une punaise.

Nous pouvons résumer le problème que nous tentons de résoudre par les deux questions suivantes:

Etant donné un but à atteindre ou une tâche à exécuter, quelles sont les caractéristiques des actions qui vont permettre de choisir un objet et de l'utiliser comme outil pour exécuter une tâche?

Une fois l'objet choisi, comment doit-on le tenir et quel mouvement doit-on exécuter pour atteindre le but?

II-2.2 Raisonnement et vision

Nous nous proposons, non pas de réaliser des logiciels de traitements d'images ultra-performants et longs,analysant jusque dans ses moindres détails une image, mais au contraire, se contenter de la connaissance de quelques éléments caractéristiques pour tenter d'identifier les objets. En cas d'échec, l'analyse d'autres images, prises sous des angles de vue différents doit permettre d'aboutir à l'identification.

Le rôle du raisonnement est de mettre en correspondance la description bi-dimensionnelle des objets résultant de l'analyse, avec la représentation tri-dimensionnelle des objets susceptibles de se trouver dans la scène. Une mise en correspondance partielle avec plusieurs objets conduit à une reconnaissance multiple.

Le raisonnement déduit les angles de prise de vue qui permettront de lever les ambiguïtés, en fonction d'une partie, des objets susceptibles d'être présents et d'autre part, des objets entrant en concurrence ou déjà partiellement reconnus.

Le raisonnement est fondé à la fois sur les valeurs des attributs résultant de l'analyse de l'image, et sur la connaissance partielle de la scène, c'est à dire le milieu industriel dans lequel se situe l'application: type d'objet, type de scènes.

Cette étude doit permettre à un robot d'évoluer dans un environnement préalablement décrit, puis d'agir sur celui-ci, en fonction d'ordres transmis ou de buts à atteindre. Les applications potentielles de cette étude se situent dans le domaine industriel classique (robotisation) et dans les domaines à environnement hostiles (nucléaire, espace ...).

II-2.3 La mise en commun des compétences

Les compétences respectives des deux laboratoires se complètent dans le cadre de ce projet.

Les compétences, très largement reconnues, et les recherches physiologiques et informatiques en intelligence artificielle, sur le raisonnement du laboratoire américain d'une part, et les recherches et le savoir-faire de notre laboratoire dans le domaine de la vision industrielle et robotique d'autre part, offrent toutes les chances de réussite au projet R.E.V.E.R.I.E.

III Le sujet du stage

III-1 Utilité du stage

Le stage s'intègre dans le cadre de la partie vision du projet R.E.V.E.R.I.E. . Le but de cette partie est de reconnaitre des objets dans une scène. Pour cela il faut avoir une idée précise des contours et des faces des objets.

Le problème principal est que les outils connus pour extraire des attributs d'une image (i.e: contours et régions) ne donnent pas d'assez bons résultats. Les contours obtenus ne sont généralement pas fermés, ou tout bonnement il en manque. Les régions obtenues grâce à des "segmenteurs" donnent, comme les contours, des renseignements incomplets (voire erronés).

Les contours et régions obtenus ne permettent généralement pas d'entreprendre une reconnaissance directe des objets (puisqu'il manque des détails).

Certaines méthodes, très calculatoires, permettent de modifier les attributs extraits de l'image. Il est alors possible de reconnaître les objets de la scène à analyser. Ce genre de méthodes pose un problème très important en robotique temps réel: Elles sont très longues.

Une méthode originale développée par AHMED M.NAZIF et MARTIN D.LEVINE met en oeuvre un système expert [NAZ 84].

Bref résumé de cette méthode:

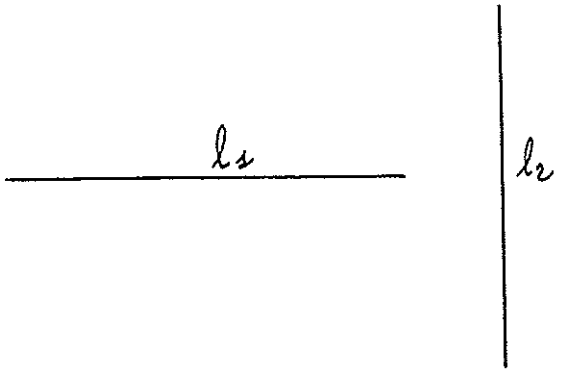
Afin d'analyser le contenu d'une scène, une segmentation d'image en régions et contours est réalisée grâce à un système expert.
Différents types de règles sont utilisés:

a) Des règles de connaissance sur la segmentation bas niveau

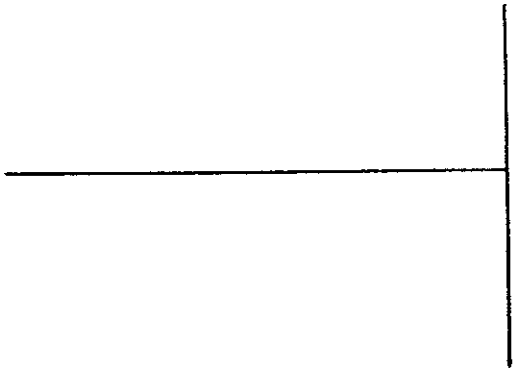
Ces règles permettent de segmenter l'image en régions uniformes et contours fermés. Elles représentent des propriétés des attributs de bas niveau entre eux.

L'exemple suivant est caractéristique:

Soit la configuration suivante représentant deux segments de droite:



Cette configuration doit être transformée en:



On peut ainsi exprimer une règle:

SI
Une ligne l1 n'est pas fermée

ET La distance de cette ligne l1 à une ligne l2 se trouvant devant l1 est faible

ALORS Prolonger la ligne l1 jusqu'à la ligne l2

b) Des règles de contrôle

Ces méta-règles permettent de déterminer la partie de l'image à analyser ("Focus of attention").
Ceci est utile pour savoir quelles données considérer lors de l'utilisation des règles exposées en a).
Par exemple: quelle est la ligne à compléter, quelle est la région que l'on doit diviser (spliter), etc ...

c) D'autres règles de contrôle

Ces règles sont d'un niveau encore supérieur aux règles exposées en b).
Ces méta-méta-règles permettent de déterminer si il faut activer les règles concernant les régions ou celles concernant les lignes.

Le temps de segmentation d'une image couleur d'une scène prise en extérieurs (i.e: scène complexe) est d'environ 20 minutes sur un Digital VAX 11/780.

Nous allons nous inspirer de cette méthode, en la simplifiant car la sulution à trois niveaux de règles parait trop complexe, pour réaliser une segmentation en régions et contours d'une image.

Si l'analyse d'une image est rapide on peut adopter la méthode suivante:
Si l'analyse n'aboutit pas a un résultat correct on reprend une autre vue (sous un angle différent bien sûr !) que l'on analyse pour résoudre les ambiguïtés de la première analyse.

III-2 Le travail à réaliser

A mon arrivée dans le laboratoire un certain nombre d'utilitaires fonctionnaient:

- initialisation des cartes Datacubes (cartes de traitement d'images)
- extraction et suivi de contours sur des images binaires
- approximation polygonale

Une partie était écrite en MODULA-2 et l'autre en assembleur 68000.

Le programme développé sur SM90 (avec un 68000) était transféré sur une machine cible (le prototype) à base de 68000 et des cartes de traitement d'images où il s'exécutait.

Un changement de système est intervenu:
Programmes développés sur SUN (avec un 68020) et transfert sur le nouveau prototype avec une carte 68020 et des cartes Datacube.

Ceci a impliqué un portage du programme existant:

Il y a eu un changement de compilateur MODULA-2, ce qui a nécessité une traduction des programmes MODULA-2 ainsi qu'une modification dans le passage des paramètres avec l'assembleur.

De plus les programmes étaient écrits sans aucun souci de standardisation et de maintenabilité. Un certain nombre de primitives ont dû ainsi être réécrites.

Une fois le portage réalisé, quelques segmenteurs en régions ont été étudiés:

- Le "blob coloring" (coloriage de tâches)
- La segmentation avec des arbres de côut minimum
- Une segmentation récursive.

A ce niveau là nous disposions d'un extracteur de contours et de divers segmenteurs en régions.

Avant toute chose il faut définir une représentation efficace des attributs extraits (i.e: contours et régions) en vue de leur exploitation avec le système expert. Par exemple il faut savoir faire une fusion ou une division sur des régions sans avoir à manipuler les 512 x 512 points d'une image.

Plusieurs représentations des régions sont envisagées.

Enfin l'écriture de règles doit permettre de segmenter l'image en régions et contours fermés.

IV Organisation matérielle du projet

IV-1 Le matériel proprement dit

- Un SUN 3/150 avec un 68020 et coprocesseur mathématique.

- Un prototype avec:
  . une carte 68020 et 1 MΦ de mémoire,
  . un coprocesseur mathématique,
  . des cartes de traitement d'images Datacube (Framestore et Digimax).

Les cartes Datacube sont des cartes de traitement d'images temps réel.

La carte Framestore est une carte de mémoire d'image.Elle possède trois plans mémoire (0, 1 et 2).

Les adresses d'implantation sont:

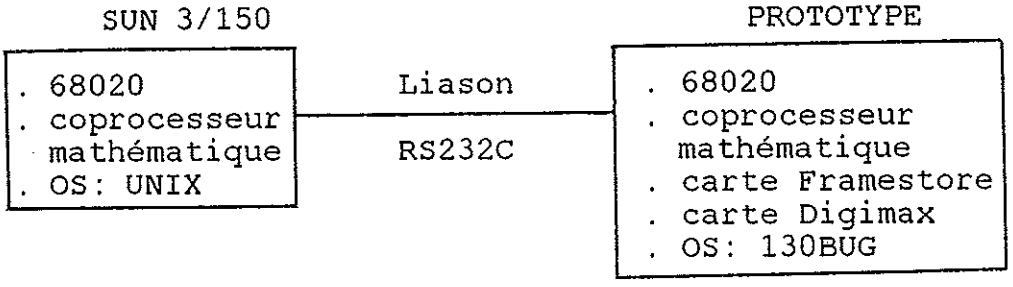
FRAMESTORE 0 en 580000H
FRAMESTORE 1 en 500000H
FRAMESTORE 2 en 540000H

La FRAMESTORE 0 est connectée en entrée sur une caméra et en sortie sur un moniteur couleur.
Les FRAMESTORE 1 et 2 servent à stocker des images.

La carte Digimax est une carte de numérisation et de visualisation.

IV-2 Communication entre les deux machines

Les deux machines sont reliées par une liaison serie RS232C.
On obtient le schéma suivant:



IV-3 Philosophie de développement

IV-3.1 Les programmes destinés au prototype

Ils sont développés en assembleur et MODULA-2 sur le SUN. Le langage MODULA-2 est utilisé car il peut facilement être utilisé sur une machine nue (le prototype).

Après compilation, assemblage et éditions de liens, le code généré est envoyé du SUN vers le prototype via la liaison RS232, où il s'exécutera.

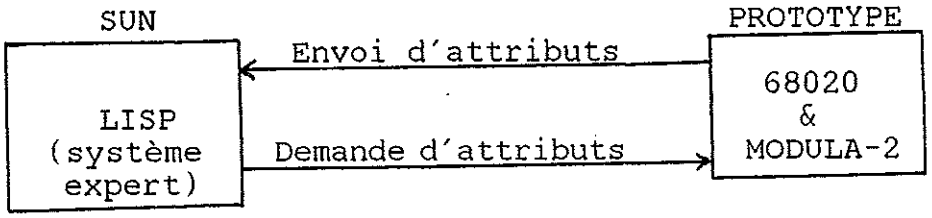
Les programmes s'exécutant sur le prototype sont des programmes de traitement d'images utilisant les cartes Framestore et Digimax.

IV-3.2 Les programmes destinés au SUN

Ces programmes sont écrits en LISP. Ils traitent les attributs extraits (régions et contours) par le prototype. Le système expert écrit en LISP doit permettre d'obtenir des arêtes et régions "propres".

Le système expert doit lors de son raisonnement, demander des attributs au prototype (ex: quelle est la moyenne d'une région ?).
Le prototype calcule les attributs demandés et les envoie au système expert tournant sur le SUN.

On peut résumer le dialogue entres les deux machines ainsi:



VOIR EN ANNEXE 1 l'utilisation du matériel et du logiciel associé.

V Le portage du programme existant

A mon arrivée dans le laboratoire un changement de machine est intervenu, ce qui a impliqué un transport du programme existant. (voir III **Le sujet du stage**)

Le transport d'un logiciel d'une machine à l'autre est réellement d'un ennui profond. C'est très long et sans aucun intérêt. Rien d'étonnant si le coût de maintenance d'un programme représente environ 70% du coût total d'un logiciel.

V-1 Les changements intervenus au niveau du compilateur MODULA-2

La précédente version du compilateur MODULA-2 s'éxécutait en 5 passes. La nouvelle version est une version beaucoup plus rapide en 1 seule passe.

Les principales modifications sont:

a) Au niveau instructions:

- Les instructions NEW et DISPOSE n'exitent plus. Il faut utiliser ALLOCATE et DEALLOCATE du module Storage.

- Les références en avant ne sont plus autorisées. Si on doit en faire une il faut utiliser la clause FORWARD.

- La clause EXPORT d'un module de DEFINITION est supprimée. Tous les objets décrits sont implicitement exportés.

- Des fonctions de conversion de type sont disponibles ainsi que la fonction VAL qui convertit 2 types compatibles.

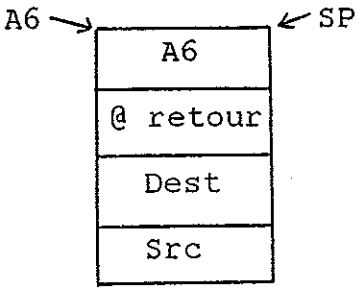
- L'instruction INLINE remplace l'instruction CODE qui permettait de mettre du code 68020 dans les instruction MODULA-2.

b) Au niveau du passage des paramètres du MODULA-2 vers l'assembleur:

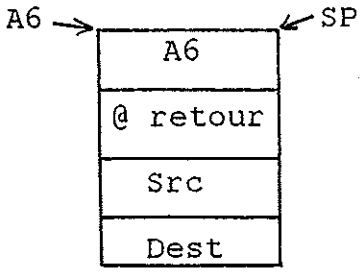
Soit la procédure Proc définie ainsi:

```
PROCEDURE Proc (Src,Dest : ADDRESS);
```

Avant lors de l'appel de la procédure Proc on a la configuration de pile suivante:



On a ensuite:



c) Au niveau des étiquettes des sous programmes écrits en assembleur:

Précédemment la procédure Proc écrite en assembleur devait avoir comme étiquette Proc. On devait déclarer: .glob Proc .

Avec le nouveau compilateur l'etiquette _Proc doit être écrite dans la partie assembleur du module en mettant un .glob _Proc .

V-2 Standardisation et structuration de l'existant

Le moins que l'on puisse dire est que les programmes qui existaient à mon arrivée n'étaient pas très standards.Cela a impliqué une réécriture d'un certain nombre de modules.

V-2.1 Les entrées/sorties MODULA-2

Le programme MODULA-2 fonctionnant en "stand-alone" sur le prototype, il n'est pas possible d'utiliser le module InOut du MODULA-2 standard. Il existait plusieurs modules d'E/S qui n'étaient absolument pas standards.

a) Un module InOut a été écrit en utilisant des E/S caractère par caratère.

Principe:

. Le module Dsse définit l'entrée et la sortie d'un caractère.
(voir /usr/M2/M2sa/objects/Dsse)

. Le module Conversion convertit une suite de caractères en une valeur d'un certain type ou bien le contraire.
(voir /usr/M2/M2sa/objects/Conversion)

. Le module InOut permet de lire ou d'écrire une chaine de caractères via les modules Dsse et Conversion.
(voir /usr/M2/M2sa/objects/InOut)

Exemple: pour lire un réel:

. lire une chaine de caractères en utilisant les entrées/sorties caractère par caractère du module Dsse.

. puis conversion de la chaine de caractères en la valeur réelle correspondante.

Avantage de cette approche:

. Très standard.

. Si l'on change de système cible, il suffit de changer les E/S de bas niveau de Dsse.

b) Un module Ecran permet de gérer une console alphanumérique VT4200

Possibilités offertes:

. Effacement d'une partie de l'écran (ou de tout l'écran)

. Position du curseur

. Changement de mode vidéo

Evidemment c'est moins complet que le logiciel Curses mais cela suffit.
(voir /usr/M2/M2sa/vision/Ecran)

c) Un module Menu permet de créer des menus et de les utiliser par simple déplacement d'un curseur avec les flèches.
(voir /usr/M2/M2sa/vision/Menu)

V-2.2 La gestion mémoire sur le prototype

Comme pour les entrées/sorties, la gestion mémoire de MODULA-2 ne peut pas être utilisée car le programme doit fonctionner sur une machine nue (i.e: sans système d'exploitation).

Il est donc nécessaire d'écrire un module de gestion mémoire StAlloc. Le module Storage n'est présent que pour faire le passage du MODULA-2 vers le module StAlloc.

La gestion de la mémoire se fait par blocs chainés entre eux. Le changement de système n'implique qu'un changement de la base des adresses (Début de la zone mémoire disponible).
(voir /usr/M2/M2sa/objects/StAlloc et /usr/M2/M2sa/objects Storage)

L'allocation de place se fait par ALLOCATE, et la libération de place se fait par DEALLOCATE.
(i.e: plus de NEW et DISPOSE !)

VI Modules de traitement d'images d'intérêt général

Ces modules fonctionnent sur le prototype et utilise les cartes de traitement d'image.

VI-1 Le module TYPES

Les principaux types définis dans le projet sont dans le module TYPES.

On y trouve la définition d'un point image, d'un vecteur de points et surtout la définition d'une image:

```
MaxCol = 511; et MaxLin = 511;
Ligne = ARRAY [0..Maxcol] OF CHAR;
Image = ARRAY [0..MaxLin] OF Ligne;
```

Les images sont sur 256 niveaux de gris, donc un caractère suffit pour représenter un point (0-255).

(voir le fichier /usr/M2/M2sa/vision/TYPES.DEF).

VI-2 Le module Saisie

Ce module contient les procédures manipulant directement les cartes Datacube:

- InitCarte : réalise l'initialisation des cartes
- Saisie : saisie une image dans la Framestore 0
- Continue : fait passer en mode vidéo continu
- Visu : permet de sélectionner le plan mémoire en visualisation sur le moniteur. Par défaut c'est la Framestore 0.
- InitSeuil, FinSeuil, IncSeuil, DecSeuil et LevelSeuil : permettent de faire du seuillage dynamique (binarisation intéractive) .
- MoveImage : copie une image dans une autre

La procédure InitCarte doit être appelée avant toute chose.

(voir le fichier /usr/M2/M2sa/vision/Saisie)

VI-3 Un module OpLogic

Réalise des opérations logiques sur des images. Ceci permet de faire des manipulations intéressantes sur 2 plans mémoire.

Opérations disponibles: ET, OU, XOR et NON.
Ces opérations sont réalisées bit à bit.

En appliquant ces opérateurs sur des images binaires:

Le ET garde les parties communes des 2 images.
Le OU superpose les 2 images.
Le NON inverse une image.

Ces opérations peuvent bien sûr être utilisées sur des images non binaires.

VI-4 Un module Masques

Ce module contient quelques masques de détection de contours. En fait un masque met en évidence des changement brusques de contraste ce qui correspond généralement à un contour ou une arête d'objet.

L'application d'un masque est en fait un calcul de produit de convolution: $g(x,y) = f(x,y) * h(x,y)$.
Cela revient à calculer l'image gradient d'une image initiale.

Où :
g correspond à l'image résultat
f correspond à l'image initiale
et h est un masque de convolution (2m+1) x (2m+1).

Sous forme discrétisée on a:

$$g(x,y) = \sum_{i=-m}^{i=+m} \sum_{j=-m}^{j=+m} f(x+i,y+j).\ h(i,j)$$

Plusieurs masques sont disponibles:

a) Sobel diagonal:

$$h1 = \begin{vmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{vmatrix} \qquad h2 = \begin{vmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{vmatrix}$$

b) Sobel horizontal-vertical:

$$h1 = \begin{vmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{vmatrix} \qquad h2 = \begin{vmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & 1 \end{vmatrix}$$

c) Prewitt diagonal:

$$h1 = \begin{vmatrix} 1 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -1 \end{vmatrix} \qquad h2 = \begin{vmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{vmatrix}$$

d) Prewitt horizontal-vertical:

$$h1 = \begin{vmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{vmatrix} \qquad h2 = \begin{vmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & 1 \end{vmatrix}$$

Des masques de taille plus importante existent: 5 x 5 ou 21 x 21. Ces masques sont très longs à exécuter car il n'y a pas d'optimisation possible au niveau des registres lorsque l'on programme en assembleur (il y a trop de coefficients)

Résultats obtenus:

Des ruptures de contraste sont obtenues, mais il y a beaucoup de bruit résiduel. On n'obtient pas des contours nets et précis des objets ==> nécessité d'avoir un suivi de contour pour faire le ménage.

Le calcul de l'image moyenne d'une image est également disponible.

(voir /usr/M2/M2sa/vision/Masques.DEF et
/usr/M2/M2sa/vision/masques.s)

VI-5 Un module histogramme

Réalisation de différents calculs d'histogrammes :

- sur l'image entière
- sur une fenêtre rectangulaire de l'image
- sur une zone quelconque définie par l'utilisateur
- sur une droite de coupe
- histogramme en relief de l'image

(voir /usr/M2/M2sa/vision/Histo)

VI-6 Un module Mires

Le module /usr/M2/M2sa/vision/Mires n'a pas d'intérêt en soit mais permet de dessiner des mires (en dégradés de niveaux de gris) sur le moniteur.

Permet de faire des tests de programmes de traitement d'image sur des images connues.

VI-7 Un module Calculs

Réalise les opérations suivantes:

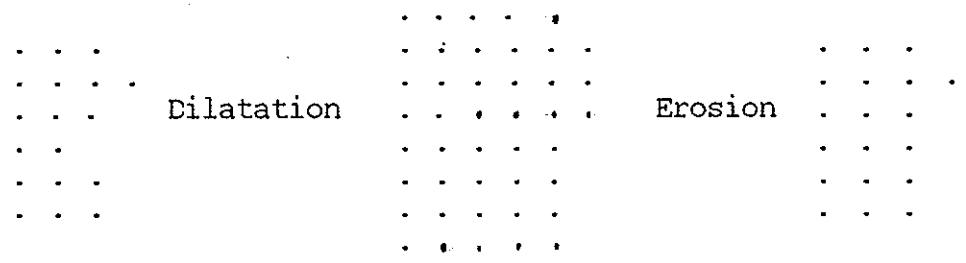
- **ÉROSION**
- **DILATATION**

Ces opérations sont classiques en traitement d'images.

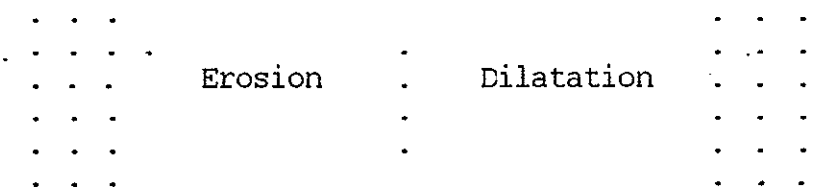
L'érosion (la dilatation) réalise le ET (OU) logique sur les 8 voisins d'un point.

Les opérations combinées dilatation-retraction permettent d'atténuer les irrégularités locales.

exemple:



puis:



VII La segmentation d'image en régions

VII-1 La segmentation

Le but de la segmentation est d'affecter les points de l'image à un nombre réduit de "classes" appelées régions. Dans le cas d'une image en niveaux de gris, la segmentation regroupe des points en fonction de leur homogénéité de niveaux de gris.

Les algorithmes de détermination des régions peuvent être classés en trois catégories:

1- Techniques locales:

Les pixels sont regroupés en régions sur la base de leurs propriétés ou des propriétés de leurs voisins.

2- Techniques globales:

Les pixels sont regroupés en régions sur la base des propriétés d'un grand nombre de pixels distribués à travers de l'image.

3- Techniques de division et fusion (SPLIT AND MERGE)

Les techniques précédentes sont en rapport avec des pixels individuels, ou des ensembles de pixels. Des techniques d'espace d'états fusionnent ou séparent des régions en utilisant des structures de graphe pour représenter ces régions et leurs frontières. On peut faire des fusions et séparations en utilisant des critères locaux ou globaux.

Avant toute chose donnons quelques définitions:

A- x_i est connecté à x_j si ils sont dans la même région.
x_i et x_j étant des points image.

B- L'image entière $I = \bigcup_{k=1}^{m} R_k$:
L'union des régions forme l'image entière.

C- $R_i \cap R_j = \phi \qquad i \neq j$:
Toutes les régions sont disjointes.

D- Soit H un critère d'homogénéité.

$H(R_K)$ = Vrai quelque soit k.
$H(R_i \cup R_j)$ = Faux pour $i \neq j$ et
R_i et R_j voisines.

Nous avons étudié trois algorithmes de segmentation en régions:

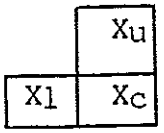
- Le "blob coloring" (Le coloriage de tâches),
- La segmentation par arbres de coût minimum,
- Un algorithme récursif de segmentation.

VII-2 Le blob coloring et ses variantes

VII-2.1 **une version du coloriage de tâches** est donné dans l'ouvrage de Ballard [BAL 82].

L'algorithme 1:

soit: k:=1 /* la couleur initiale */
et soit l'élément structurant suivant:



En parcourant l'image de gauche à droite et du haut vers le bas avec l'élément structurant:

```
SI f(Xc) = 0

   ALORS continuer

   SINON DEBUT

         SI f(Xu) = 1 ET f(Xl) = 0
                ALORS color(Xc):=color(Xu)

         SI f(Xl) = 1 ET f(Xu) = 0
                ALORS color(Xc):=color(Xl)

         SI f(Xl) = 1 ET f(Xu)= 1
                ALORS color(Xc):=color(Xl)

         /* on a maintenant une nouvelle région */
         SI f(Xl) = 0 ET f(Xu) = 0
                ALORS color(Xc):=k;
                      k:=k+1;
         FIN
```

Où:
f(X) correspond à la valeur de l'image d'entrée au point X et color(X) correspond à la valeur de l'image de sortie au point X.

Le problème de cet algorithme est qu'il ne s'applique qu'à des images binaires. Voyons comment le modifier pour qu'il puisse s'appliquer à des images en niveaux de gris.

VII-2.2 1ére variante du blob

Idées principales:

. on élargit l'élément structurant:

X_2	X_3	X_4
X_1	X_C	

. X_C reçoit la valeur d'un X_i si $|X_C-X_i| < Seuil$.

Problème:
Il peut y avoir plusieurs X_i candidats.

Solution:
Prendre X_i tel que l'on ait $|X_C-X_i|$ minimum.
(i.e: pour faire le moins d'erreur possible)

L'Algorithme 2:

```
k:=1      /* la couleur initiale */
```

En parcourant l'image de gauche à droite et du haut vers le bas avec le nouvel élément structurant:

```
E:={ Xi / |f(Xc)-f(Xi)| < Seuil }
SI E = ϕ
     ALORS  /* on doit créer une nouvelle région */
            color(Xc):=k;
            k:=k+1

     SINON  Xmin:= un Xi ∈ E tel que |Xi-Xc| minimum
            color(Xc):=color(Xmin)
```

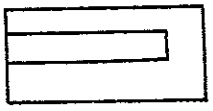
Où:
f(X) correspond à la valeur de l'image d'entrée au point X et color(X) correspond à la valeur de l'image de sortie au point X.

On peut prendre Seuil = 20.

Cet algorithme présente quelques défauts:

. Si sur 3 lignes écrans nous avons les régions suivantes:

1ére ligne
2ème ligne
3ème ligne



plus de 2 régions seront trouvées puisque la ligne du bas se trouve en dehors de l'élément structurant.

. On ne tient pas compte de la notion de gradient (voir opérateurs Sobel et Prewitt dans le module **Masques** dans VI-4)

VII-2.3 2ème variante du blob

Idées principales:

. Mémorisation des moyennes des régions déja obtenues. Si l'on doit créer une nouvelle région on regarde si il n'existe pas déjà une région avec un niveaux de gris proche.

. Utilisation de l'image gradient: On ne crée une nouvelle région que si le gradient est supérieur à un seuil.

L'Algorithme 3:
Avec toujours le même élément structurant.

k:=1; /* couleur initiale */

En parcourant l'image de gauche à droite et du haut vers le bas avec l'élément structurant:

E:={X_i / $|f_1(X_c)-f_1(X_i)| < Seuil1$ et $f_2(X_c) > Seuil2$}
SI E = ϕ

ALORS . Trouver dans MR une région R telle que:
|Moyenne de R - $f_1(X_c)$| < Seuil1
. SI R existe

ALORS color(X_c):=color(un point de R);
Modifier MR[R]

SINON /* on crée une nouvelle région */
color(X_c):=k;
MR[k] <= X_c;
k:=k+1;

SINON X_{min}:= un $X_i \in E$ tel que $|X_i-X_c|$ mininum
color(X_c):=color(X_{min})
modifier MR[color(X_c)]

Où:
$f_1(X)$ correspond à la valeur de l'image d'entrée au point X , $f_2(X)$ correspond à l'image gradient au point X et color(X) correspond à la valeur de l'image de sortie au point X.
MR est un tableau [1..MaxRégions] DE Moyenne.

On peut prendre Seuil1 = 20 et Seuil2 = 10.

RESULTATS OBTENUS:

. Le traitement est rapide
. Le résultat ne permet pas de définir les surfaces des objets précisemment. Néanmoins cela suffit pour le traitement que l'on veut faire (utilisation du système expert).

VII-3 La segmentation par arbres de coût minimum

Ceci n'est qu'un bref exposé d'une méthode originale de segmentation.
Pour plus de détail voir l'article [MIN 83].

Un exemple plutôt qu'un long discour explique très bien l'algorithme.

Soit une image 4 x 4:

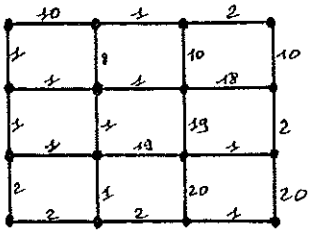
1. 11. 12. 10.

2. 3. 2. 20.

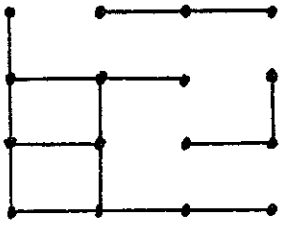
3. 2. 21. 22.

1. 3. 1. 2.

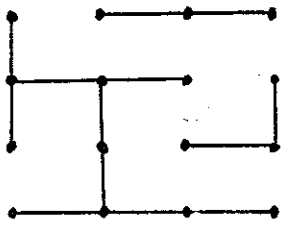
On peut construire un graphe avec des arêtes, entre 2 points, valuées par la différence de niveaux de gris entre ces 2 points:



En supprimant toutes les arêtes qui ont un poids supérieur à un certain seuil (par exemple 3) on obtient:



En enlevant les cycles on a:



Ce qui correspond aux régions de l'image.

Formalisons maintenant un peu tout cela:

Soit V={vi} l'ensemble des noeuds correspondant aux pixels de l'image.

Soit E={ei} l'ensemble des arêtes qui lient 2 points entre eux. (i.e: un point est relié à ses 4-voisins)

Pour une image n x n: card(V)=n2 et card(E)=2n.(n-1)

Algorithme:

Etape 1:
G=(V,E)

Etape 2:
G'=(V,E') est le graphe G auquel on a enlevé les arêtes de poids supérieur à un certain seuil.

Etape 3:
Chercher G0=(V,E0) la forêt recouvrante de coût minimun.
Chaque arbre de G0 correspond à une région.

Quelques améliorations sont possibles:

. Si 2 régions ont une moyenne proche on peut les regrouper.
. Si une région de petite taille existe, on peut la regrouper avec une région voisine qui a une moyenne de niveau de gris proche de la moyenne de la petite région.

Cette méthode n'a pas été programmée car elle doit prendre beaucoup de temps:
- construction du graphe
- élimination des cycles pour trouver la forêt recouvrante de coût minimum.

VII-4 La segmentation récursive

L'idée principale de cet algorithme est que les régions sont déterminées l'une après l'autre. Lorsque l'on détermine une région, on la détermine complètement.

Pour déterminer une région on cherche un point qui n'appartient encore à aucune région, puis on construit **complètement** la région à laquelle il appartient.

Algorithme 1:

```
k := 1   /* la couleur initiale */

tant que toute l'image n'est pas étudiée:

        . trouver un point P non marqué
        . marquer P
        . moyenne := f(P)
        . nbpoints := 1
        . RegionGrowing (P)
        . k := k+1

fintantque
```

Avec la procédure récursive **RegionGrowing** (P:Point)

Cette procédure détermine complètement la région à laquelle appartient P.

Le point P est le point central d'une fenêtre 3 x 3:

P1	P2	P3
P8	P	P4
P7	P6	P5

```
PROCEDURE RegionGrowing ( P:Point )

    Pour chaque Pi:

     SI Pi non marqué et |moyenne-f(Pi)| < Seuil1
         ALORS . marquer Pi
               . modifier moyenne et nbpoints
               . color(Pi):=k
               . RegionGrowing (Pi)
     FINSI
    FINPCUR
```

Où:

f(X) correspond à la valeur de l'image d'entrée au point X et color(X) correspond à la valeur de l'image de sortie au point X.

On peut prendre Seuil1 = 20.

Le problème principal de cet algorithme est qu'il est très sujet au bruit. Dès qu'une petite région est présente, l'algorithme la détecte (même si la région n'est qu'un point). On trouve par conséquent un nombre trop élevé de régions.

La solution à ce problème est présenté dans l'algorithme 2:

L'algorithme 2:

L'idée est la suivante: Avant de vouloir créer une nouvelle région avec comme point de départ un point P, on regarde si il n'existe pas déjà une région avec une moyenne proche de la valeur de point P.

```
k := 1;

tant que toute l'image n'est pas étudiée:

        . trouver un point P non marqué
        . chercher dans MR si il existe un région R tq
          |moyenne de R - f(P)| < Seuil2
        . SI R existe
              ALORS color(P):=color(un point de R)
                    modifier MR[R]

              SINON marquer P
                    nbpoints := 1
                    moyenne := f(P)
                    RegionGrowing (P)
                    MR[k] <= P
                    k := k+1
          FINSI
fintantque
```

Avec:
RegionGrowing inchangé,
f(X) correspond à la valeur de l'image d'entrée au point X et color(X) correspond à la valeur de l'image de sortie au point X,
MR est un tableau de 1..MaxRégions DE Moyenne

On peut prendre Seuil1 = 20 et Seuil2 = 4.

(voir /usr/reverie/rodin/rev/proj/SegRec)

VII-5 Conclusion

La méthode du blob est rapide et donne des résultats suffisants pour l'utilisation que l'on veut en faire.

La méthode de la segmentation récursive est un peu longue est très sujette aux bruits dûs au capteur.

Dans un premier temps les deux méthodes ont été programmées en MODULA-2. Comme au vue des résultats le blob va être utilisé, il a été réécrit en assembleur 68020. On obtient ainsi un temps de segmentation en région d'une image 512 x 512 de 6 secondes.

VIII Représentations symboliques des régions

Nous avons décidé de représenter une région sous la forme d'une liste de segments de droites. Cette liste de segments correspond au contour de la région considérée.

Une région peut avoir d'autres attributs comme:

- la surface,
- la moyenne,
- la variance,
- l'histogramme des niveaux de gris,
- etc ...

Voyons comment représenter une région en LISP.
quelles sont les structures de données à mettre en oeuvre?

VIII-1 Structures de données LISP

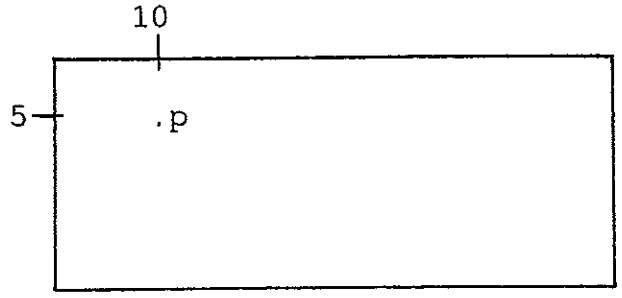
VIII-1.1 Les structures de données

a) structure de données associée aux points image

Un point est représenté par une S-expression:

(ligne . colonne)

exemple: p = (5 . 10)

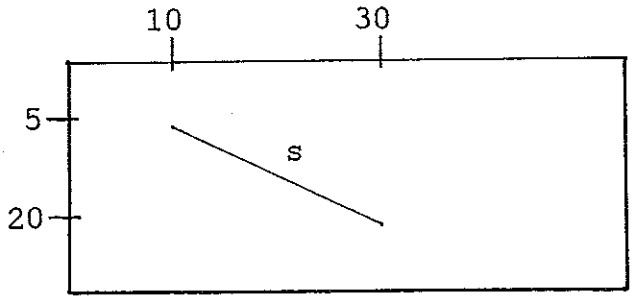


b) Structure de données associée aux segments

Un segment est représenté par une liste de 2 points:

((x1 . y1) (x2 . y2))

exemple: s = ((5 . 10) (20 . 30))



On définit une **écriture unique** d'un segment:

s = ((x1 . y1) (x2 . y2)) est un segment si:

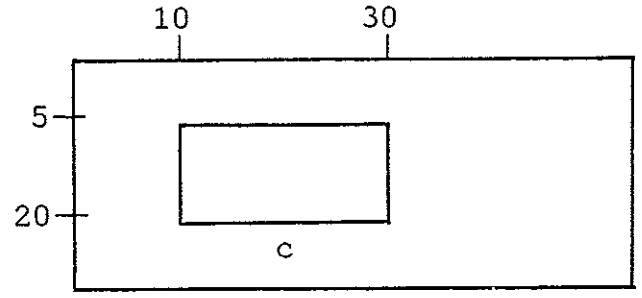
x1 < x2
ou (x1 = x2 et y1 < y2)

c) Structure de données associée aux contours

Un contour est représenté par une liste de segments

```
(
  ((x1 . y1) (x2 . y2))
  ((~~ . ~~) (~~ . ~~))
  ((~~ . ~~) (~~ . ~~))
          .
          .
          .
  ((~~ . ~~) (~~ . ~~))
)
```

exemple: c=((5 . 10) (5 . 30)
(5 . 30) (20 . 30)
(20 . 10) (20 . 30)
(5 . 10) (20 . 10))



d) Structure de données associée aux régions

A chaque région r on associe un atome gi

. La valeur de l'atome contient le numéro de la région.
. La P-liste de l'atome contient tous les attributs de la région qu'il représente.

Les champs de la P-liste sont:

- moyenne : donne la moyenne de la région r.

- surface : donne la surface de la région.r.

- contour : contient le contour de la région r. (description donnée en c))

- adjacence : donne une liste des régions adjacentes à la région r.

- SegmentCommun : donne pour chaque région r' de la liste d'adjacence de la région r, les segments communs à r et r'.

- SegmentRegion : donne pour chaque segment s du contour de r, la région r' qui est adjacente à r vis à vis de s.

REMARQUES :

. Les champs moyenne, surface, contour doivent être calculés par le prototype.

. Les champs adjacence, SegmentCommun, SegmentRegion sont calculés par le programme LISP.

. La liste des champs n'est pas complète! On peut très facilement en rajouter (ex : histogramme, moyenne gradient, etc ...)

e) Représentation d'une image

Une image est une liste de régions.

lr = (r1 r2 ... rn)

f) Exemple: **voir annexe 2**

VIII-1.2 Les algorithmes généraux

a) Ajouter_région r

Ajoute une région r dans une image en calculant les adjacences avec les régions déjà présentes.

Algorithme:

POUR chaque segment s du contour de la région r:

Existe-t-il une région r' qui a s (ou une partie de s) dans son contour ?

SI oui **ALORS** traiter_adjacence(r s r')

/*
on a trouvé une adjacence entre r et r' vis à vis du segment s. (ou une partie de s: dans ce cas appeler traiter_adjacence avec la partie de s au lieu de s)
*/

FINSI
FINPOUR

La fonction **traite_adjacence** (r s r'):

Cette fonction met à jour tous les champs des régions concernant l'adjacence entre r et r' vis à vis de s (i.e: adjacence, SegmentCommun, SegmentRegion) .

b) Enlever_région r

Enlève une région r de la structure représentant l'image.

Algorithme:

.Supprimer tous les liens (i.e:adjacence, SegmentCommun, SegmentRegion) qu'une région r' peut avoir avec r.

. Enlever r de la liste des régions existantes (lr)

VIII-2 Première représentation des régions

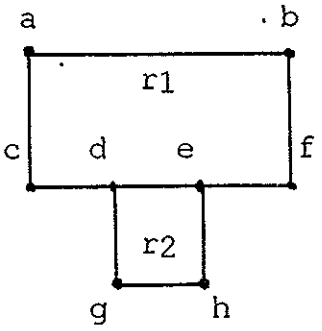
Voir le fichier /usr/reverie/rodin/lisp/OpRegion1.ll

VIII-2.1 La représentation

Une région est définie par son contour qui est lui même défini par des segments.
Ces segments peuvent être quelconques.

Donnons un exemple:

Soient les 2 régions r1 et r2



La région r1 est définie par: (a b),
(b f),
(f c) et
(c a)

et la région r2 est définie par: (d e),
(e h),
(h g) et
(g d)

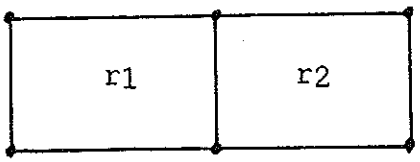
Cette représentation implique un calcul pour déterminer l'adjacence entre r1 et r2 puisque le segment (d e) n'appartient pas explicitement à la région r1

VIII-2.2 Les principaux algorithmes

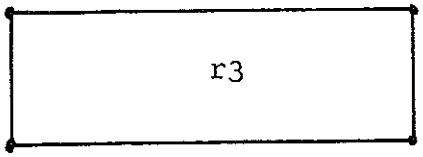
a) La fusion de deux régions r1 et r2 : la région r3

Le contour de la région résultat r3 correspond à l'union des deux contours des régions r1 et r2 moins les parties communes aux deux contours.

exemple 1:

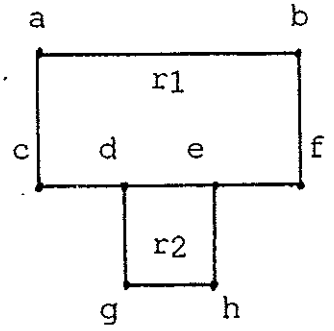


fusion(r1 r2) = r3



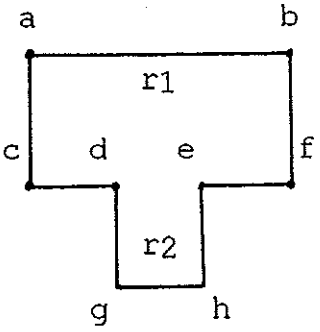
Ce cas se traite sans problème car c'est une simple différence symétrique sur les contours.

exemple 2: Un cas plus ennuyeux est le suivant:



Dans ce cas il faut calculer que le segment (d e) est un segment commun à r1 (qui contient le segment (c f)) et r (qui contient le segment (d e)).

Le résultat de la fusion est le suivant:



L'algorithme de la fusion ("merge") peut se formaliser ainsi:

1) enlever r_1 et r_2 de l'image

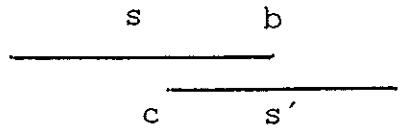
2) C:= Différence Symétrique (contour r_1, contour r_2)
Avec Différence Symetrique(A,B)=(A U B) \ (A ∩ B)

3) $C := (C \setminus \{ s \in C / \exists s' \in C \text{ et } s \cap s' \neq \phi \})$

U

$\{ s_1 \in C / \exists (s_2, s, s') \in C3$
avec $s \cap s' \neq \phi$ et
$s_1 \cup (s \cap s') \cup s_2 = s \cup s'$ et
$s_1 \cap (s \cap s') = \phi$ et
$s_1 \cap s_2 = \phi$ et
$(s \cap s') \cap s_2 = \phi$
$\}$

Ce qui veut dire:
. enlever de C les segments s et s' qui ont une partie commune:



. ajouter dans C les parties extérieures à l'intersection de s et s'.
dans exemple: ajoute (a c) et (b d) dans C

4) Ajouter dans la structure la région résultat dont on vient de calculer le contour C.

b) La division ("split") d'une région en plusieurs

On suppose connues les descriptions (contours) des régions résultats.

Le division se fait de façon triviale:

Algorithme:

. enlever la région à diviser de la structure des régions.

. ajouter chaque région résultat dans l'image.

VIII-2.3 Avantages et inconvénients de cette représentation

Avantages:

. Un segment peut être représenté sous n'importe quelle forme.

. Division très simple.

Inconvénients:

. Temps de calcul qui peut être important lorsque l'on ajoute une région dans la structure (calcul des inclusions inter-segments).

. fusion compliquée.

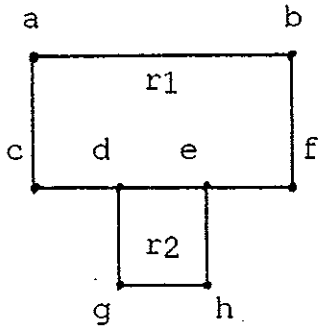
VIII-3 Deuxième représentation des régions

VIII-3.1 La représentation

Une région est définie par son contour qui est lui même défini par une liste de segments.
Ces segments peuvent être non quelconques.

Donnons un exemple:

Soient les 2 régions r1 et r2



La région r1 est définie par: (a b),
(b f),
(f e),
(e d),
(d c) et
(c a)

et la région r2 est définie par: (d e),
(e h),
(h g) et
(g d)

Contrairement à la première représentation, il n'est pas nécessaire de faire des calculs pour déterminer l'adjacence entre r1 et r2. Une seule comparaison de segment suffit.

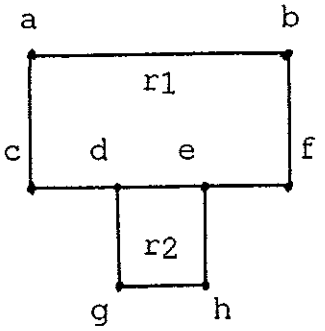
VIII-3.2 Les principaux algorithmes

a) La fusion de deux régions r1 et r2

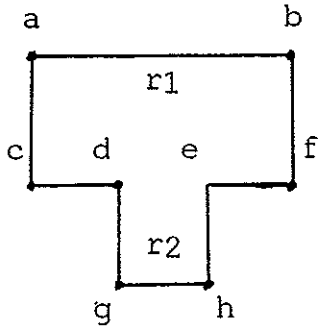
C'est très simple:

Le calcul du contour de la région résultat est seulement une différence symétrique sur les contour de r1 et r2.

exemple:



==>



(on enlève le segment (d e))

Algorithme: merge(r1 , r2)

1) enlever r1 et r2 de la structure (image)

2) C:= Différence_symétrique(contour r1 , contour r2)

3) Ajouter la région dont on vient de calculer le contour

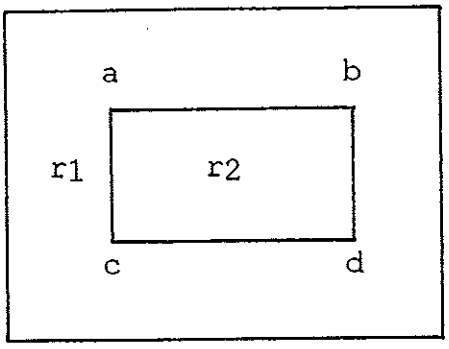
b) La division d'une région en plusieures

On suppose connues les descriptions (contours) des régions résultats.

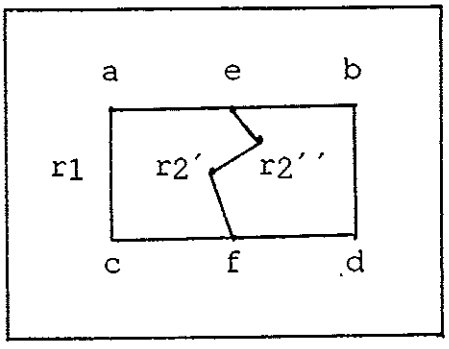
La division est un peu plus compliqué que dans le cas de la première représentation.

Voilà pourquoi:

Soient les régions r1 et r2



Si on divise r2 en



Les segments (a b) et (c d) sont découpés en 2. Il faut donc remplacer dans la description du contour de r1:

le segment (a b) par (a e) et (e b)
et le segment (c d) par (c f) et (f d)

Algorithme: du split(r liste_region)

Divise de la région r. Les régions résultantes sont dans la liste liste_region

1) enlever la région r de la structure (image)

2) Pour chaque segment s de la région r

   SI s appartient à une des régions de la liste des régions

   ALORS /* le segment s n'a pas été découpé */
   Pas de problème: RIEN À FAIRE

   SINON /* le segment s a été découpé */
   . trouver l'ensemble E de tous les segments qui composent s.

   . remplacer s par E dans la région adjacente à r vis à vis du segment s.

3) Ajouter chaque région résultante dans la structure (image)

VIII-3.3 Avantages et inconvénients de cette représentation

Avantages:

. Lorsque l'on ajoute une région dans la structure c'est très rapide.
(Le calcul d'adjacence se fait par simple comparaison de segments).

. Fusion très rapide.

Inconvénients:

. La division est compliqué (recherche des segments découpés)

. Un segment ne peut être représenté sous n'importe quelle forme.

IX Conclusion

Participant à la partie vision du projet R.E.V.E.R.I.E., j'ai dans un premier temps développé des algorithmes de segmentation d'image en régions. Puis je me suis attaché à définir une représentation symbolique des régions.

Ce travail a été nécessaire afin de pouvoir traiter les attributs de l'image (contours et régions) par un système expert. Le traitement doit permettre d'obtenir des attributs exploitables par un programme de reconnaissance d'objets (ou de parties d'objets).

Il reste encore du travail:

- définir une représentation des contours,
- réaliser un systéme expert,
- trouver un ensemble de règles de bas niveau permettant de segmenter l'image, en contours et régions, efficacement.

Je suis assez optimiste sur les chances de réussite de cette méthode. Le problème qui risque de se poser est le temps d'exécution de la segmentation en contours et régions (notamment à cause des échanges entre le SUN et le prototype).

Ce stage doit se poursuivre en vue de l'obtention du DEA informatique.

Annexe 1 Utilisation du logiciel et du matériel

I) Sur le SUN avec UNIX (marque déposée !!!)

. Utilisation du lisp:

```
$ lelisp
```

. Pour utiliser MODULA-2:

La commande m2v permet de faire à peu près ce que l'on veut.

```
appel: $ m2v Fichier.MOD
       MODULA-2 -sa (Fichier.MOD): v/c/e/l/s/z/p/m/q ? ?
       Vi edit                 = v
       Compile                 = c
       Errors                  = e
       Link                    = l
       Send To VME             = s
      (* envoi le fichier Fichier.ex vers le prototype *)
       DEF <-> MOD             = z
       Print                   = p
       changement de module = m
       Quit                    = q
```

. Pour utiliser l'assembleur:

Ecrire un fichier en assembleur appelé par MODULA-2.
Soit fic.s le fichier en assembleur

```
Pour assembler: $ as fic.s -o Fic.o
                => création de Fic.o
```

Ecrire en MODULA-2 le fichier de DEFINITION correspondant à fic.s .
Soit Fic.DEF le fichier de DEFINITION.

```
Pour compiler :$ m2c -sa FIC.DEF -ia
     => création des fichier Fic.s, Fic.mo, Fic.SBM,
     Fic.OBM, Fic.dat, Fic.lst .
```

II) Sur le prototype avec 130DEBUG

A l'allumage du VME, faire la commande 'BO'(i.e: bootstrap).
Elle charge deux blocs résidents (fichier ipl) à l'adresse 5D00 et charge le mini-shell.
Celui-ci lit une commande et l'exécute directement si c'est une commande interne (on peut en obtenir la liste en tapant '?' puis RETURN), ou recherche le fichier correspondant sur disque et l'exécute s'il possède l'attribut correspondant.

On peut toujours revenir à 130DEBUG en faisant un RESET sur la machine.

Notes:

a) Pour charger un fichier exécutable à partir du SUN tapez:

```
$ LOAD fic_a_charger <RETURN>
```

Il faut avoir fait un send to VME sur le SUN (avec la commande m2v)

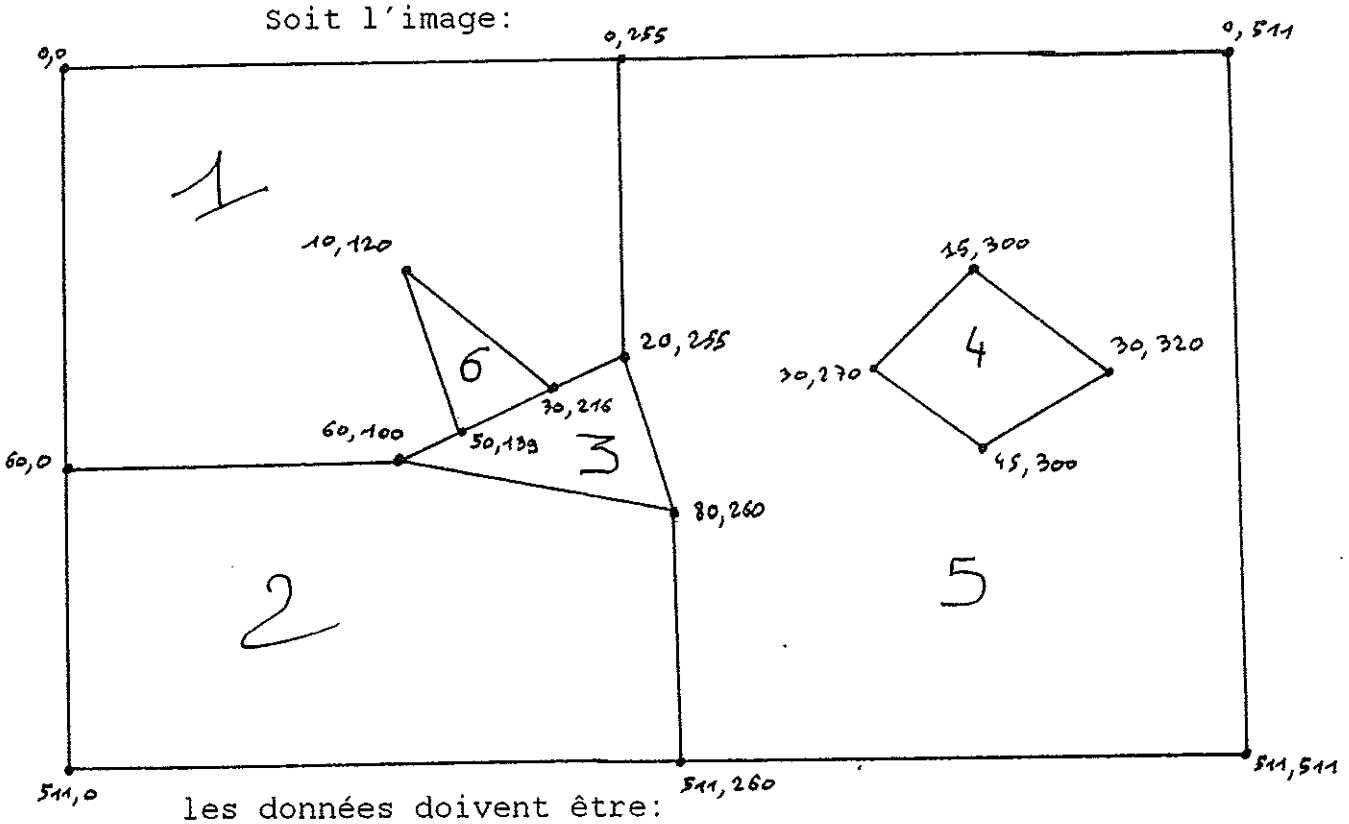
b) Pour charger en mémoire et exécuter un fichier tapez:

```
$ fic_a_executer <RETURN>
```

c) Le SGF sur le VME a été écrit par Mr G. BIGUET.

Annexe 2 données associées à une image

Soit l'image:



les données doivent être:

1) Liste d'adjacence des régions: adjacence

région 1: (2 3 5 6)
région 2: (1 3 5)
région 3: (1 2 5 6)
région 4: (5)
région 5: (1 2 3 4)
région 6: (1 3)

2) Liste des segments communs inter-régions: SegmentCommun

région 1: avec région 2: ((60 . 0) (60 . 100))
région 3: ((50 . 139) (60 . 100))
((20 . 255) (30 . 216))
région 5: ((0 . 255) (20 . 255))
région 6: ((10 . 120) (50 . 139))
((10 . 120) (30 . 216))

région 2: avec région 1: ((60 . 0) (60 . 100))
région 3: ((60 . 100) (80 . 260))
région 5: ((80 . 260)(511 . 260))

région 3: avec région 1: ((50 . 139) (60 . 100))
((20 . 255) (60 . 216))
région 2: ((60 . 100) (80 . 260))
région 5: ((20 . 255) (80 . 260))
région 6: ((30 . 216) (50 . 139))

```
région 4: avec région 5: ((15  . 300) (30  . 270))
                         ((30  . 270) (45  . 300))
                         ((30  . 320) (45  . 300))
                         ((15  . 300) (30  . 320))
région 5: avec région 1: ((0   . 255) (20  . 255))
                région 2: ((80  . 260)(511 . 260))
                région 3: ((20  . 255) (80  . 260))
                région 4: ((15  . 300) (30  . 270))
                         ((30  . 270) (45  . 300))
                         ((30  . 320) (45  . 300))
                         ((15  . 300) (30  . 320))

région 6: avec région 1: ((10  . 120) (50  . 139))
                         ((10  . 120) (30  . 216))
                région 3: ((30  . 216) (50  . 139))
```

3) Liste SegmentRegion: SegmentRegion

```
région 1:
      vis à vis de ((60  .   0) (60  . 100)):région 2
                   ((50  . 139) (60  . 100)):région 3
                   ((20  . 255) (30  . 216)):région 3
                   ((0   . 255) (20  . 255)):région 5
                   ((10  . 120) (30  . 216)):région 6
                   ((10  . 120) (50  . 139)):région 6

région 2:
      vis à vis de ((60  .   0) (60  . 100)):région 1
                   ((60  . 100) (80  . 260)):région 3
                   ((80  . 260)(511  . 260)):région 5

région 3:
      vis à vis de ((50  . 139) (60  . 100)):région 1
                   ((20  . 255) (30  . 216)):région 1
                   ((60  . 100) (80  . 260)):région 2
                   ((20  . 255) (80  . 260)):région 5
                   ((30  . 216) (50  . 139)):région 6
région 4:
      vis à vis de ((15  . 300) (30  . 270)):région 5
                   ((30  . 270) (45  . 300)):région 5
                   ((30  . 320) (45  . 300)):région 5
                   ((15  . 300) (30  .  320)):région 5
région 5:
      vis à vis de ((0   . 255) (20  . 255)):région 1
                   ((80  . 260)(511  . 260)):région 2
                   ((20  . 255) (80  .  260)):région 3
                   ((15  . 300) (30  . 270)):région 4
                   ((30  . 270) (45  . 300)):région 4
                   ((30  . 320) (45  . 300)):région 4
                   ((15  . 300) (30  . 320)):région 4

région 6:
      vis à vis de ((10  . 120) (50  . 139)):région 1
                   ((10  . 120) (30  . 216)):région 1
                   ((30  . 216) (50  . 139)):région 3
```

Bibliographie:

[NAZ 84] :

"LOW LEVEL IMAGE SEGMENTATION: AN EXPERT SYSTEM."
IEEE Transactions on pattern analysis and machine intelligence,VOL. PAMI-6 No.5,Sept 84.

[BAL 82] :

"Computer Vision"
Dana H.Ballard
Chrsitopher M.Brown
PRENTICE-HALL 1982.

[MIN 82] :

"Segmentation of images using minimum spanning trees"
Minsoo suk and Tai Hoon Cho
PROCEEDING of SPIE
VOLUME 397
Applications of Digital Image Processing
April 19-22, 1983
Geneva Switzeland.

[CHA 86] :

"LE_LISP de L'INRIA version 15.2"
Manuel de référence
Jérome Chailloux.