Annexe 7

Quelques procédures

NOYAU D'UN CONTRÔLEUR FLOU (SOLUTION AVEC DES RÈGLES FIGÉES À DEUX PRÉMISSES EN PASCAL)

```
const      nr = 9; {nr = nb de règles}
type predicat = (N, Z, P, nb, nm, ns, ze, ps, pm, pb, ANY);
type trapeze = record  g, pg, d, pd : integer end;
           {bornes droite et gauches et pentes des trapèzes tous définis dans [-10,10]}
type regle = record pd: real ; h1, h2, co : predicat end;
           {chaque règle est de la forme «si dl est h1 et dr est h2 alors u est co»}
type distribution = array [-10 .. 10] of real;
           {représente les possibilités pour -1, -0,9, -0,8 .... 0,9, 1}
type baserel = array [N .. ANY] of trapeze;
type basereg = array [1..NR] of regle;
           {Ces 2 types imposés par pascal dans les en-têtes de procedures}
           var BP : baserel;  BR : basereg;
procedure initialisation (var BP : baserel; var BR : basereg, pe : integer);
           var i : integer;        {pe = pente des trapèzes }
           begin
           BP[ANY].g:= -100; BP[ANY].pg:=0; BP[ANY].d:=100; BP[ANY].pd:= 0;
           BP[N].g := -100 ; BP[N].pg := 0 ; BP[N].d := -100 ; BP[N].pd := 200;
           BP[P].g := 100 ; BP[P].pg := 200 ; BP[P].d := 100 ; BP[P].pd := 0;
           BP[Z].g := 0 ; BP[Z].pg := 100 ; BP[Z].d := 0 ; BP[Z].pd := 100;
           BP[NB].g := -100 ; BP[NB].pg := 0 ; BP[NB].d :=-100 ; BP[NB].pd := pe ;
           BP[NM].g := -67 ; BP[NM].pg := pe ; BP[NM].d := -67; BP[NM].pd := pe;
           BP[NS].g := -50 ; BP[NS].pg := pe ; BP[NS].d := -50 ; BP[NS].pd := pe ;
           BP[ZE].g := 0 ; BP[ZE].pg := pe ; BP[ZE].d := 0 ; BP[ZE].pd := pe ;
           BP[PS].g := 50 ; BP[PS].pg := pe ; BP[PS].d := 50 ; BP[PS].pd := pe ;
           BP[PM].g := 67 ; BP[PM].pg := pe ; BP[PM].d := 67 ; BP[PM].pd := pe ;
           BP[PB].g := 100 ; BP[PB].pg := pe ; BP[PB].d := 100 ; BP[PB].pd := 0 ;
{ ci-dessous les règles du pendule inversé pe = 33 } for i := 1 to nr do BR[i].pd := 1;
BR[1].h1 := n; BR[1].h2 := n; BR[1].co := nb; BR[2].h1 := n; BR[2].h2 := z; BR[2].co := nm;
BR[3].h1 := n; BR[3].h2 := p; BR[3].co := ze; BR[4].h1 := z; BR[4].h2 := n; BR[4].co := ns;
BR[5].h1 := z; BR[5].h2 := z; BR[5].co := ze; BR[6].h1 := z; BR[6].h2 := p; BR[6].co := ps;
```

```
BR[7].h1 := p; BR[7].h2 := n; BR[7].co := ze; BR[8].h1 := p; BR[8].h2 := z; BR[8].co := pm;
BR[9].h1 := p; BR[9].h2 := p; BR[9].co := pb} end;

function ver (x : integer; t : trapeze) : real; {donne son résultat t(x) entre 0 et 1}
begin if x <= t.d     then if t.g <= x       then ver := 1
                                             else if t.g -t.pg <= x  then ver := 1 - (t.g - x) / (t.pg)
                                                                     else ver := 0
                      else if x <= t.d + t.pd          then ver := 1 - (x - t.d) / (t.pd)
                                                       else ver := 0 end;
function min (a, b : real) : real; begin if a < b then min := a else min := b end;
function conj (n : integer; a, b : real) : real; {n est un numéro d'opérateur}
begin case n of       1 : if a + b < 1 then conj := 0 else conj := a + b - 1 ;
                      2, 3 : conj := a*b ; {0 Weber, 1 Luk, 2 et 3 Larsen, 4 Mamdani}
                      0, 4, 5 : conj := min(a, b) end end;
function disj (n : integer; a, b : real) : real; {une t-conorme, 2 proba}
begin case n of       1 : disj := min(1, a + b) ;
                      2 : disj := a + b - a*b;
                      0, 3, 4, 5 : if a < b  then disj := b else disj := a {max} end end;

function impl (n : integer; a, b : real) : real; {une implication 0 Weber}
begin case n of       1, 4 : impl := min(a, b); {Mamdani}
                      0, 2, 3, 5 : impl := a*b {Larsen 2 ou 3} end end;

procedure confrontation (n, x, y : integer; R : regle ; var U : distribution ); {construit une
distribution U en confrontant 2 valeurs x, y  à une règle R déterminant U, si aucune règle ne
s'applique U = 0 et le u sera nul}
          var i : integer; v : real;
          begin v := (R.pd) * conj (n, ver (x, BP[R.h1]), ver (y, BP[R.h2]));
          for i:= -10 to 10 do U[i] := impl (n, v, ver (i*10, BP[R.co])) end;

procedure fuzzy (n, x, y : integer; var U: distribution );
{réalise l'agrégation en examinant les NR règles du tableau BR, x et y  dans [-100,100]}
          var UR : distribution; i, r : integer ;
          begin for i := -10 to 10 do U[i] := 0 ;{U est construit suivant disj}
          for r := 1 to NR do begin confrontation (n, x, y, BR[r], UR) ;
                                    for i:= -10 to 10 do U[i] := disj (n, U[i], UR[i]) end end;

function moyenne (U : distribution) : integer {renvoyé entre -100 et 100} ;
{la distribution est formée par 21 valeurs d'appartenance dans [0, 1]}
          var i : integer; c, s : real;
          begin c := 0; s := 0;
          for i := -10 to 10 do begin s := s + i*U[i] ; c := c + U[i] end;
          if c = 0 then moyenne := 0 else moyenne := round ((10*s) / c) end;

function f (n, x, y : integer) : integer; {renvoyé entre -100 et 100}
{fonction générale calculant le résultat u dans [-100,100] en fonction de x, y}
var U : distribution; r : integer; s, c, v : real;
begin  if op = 5
          then begin s := 0; c := 0; for r := 1 to nr do
          begin v := (BR[r].pd)*conj(op, ver(x, BP[BR[r].h1]), ver(y, BP[BR[r].h2]));
          s := s + impl(op, v, (BP[BR[r].co].g + BP[BR[r].co].d)/2); c := c + v end;
          if c = 0 then f := 0 else f := round(s / c) end
          else begin fuzzy (n, x, y, U); f := moyenne (U) end end;
```

NOYAU D'UN CONTRÔLEUR FLOU MAMDANI (SOLUTION AVEC DES RÈGLES FIGÉES À DEUX PRÉMISSES EN LISP)

```
(de histog (TR) (cond; TR trapèze représenté en histogramme de 21 valeurs entre -1 et 1
      ((eq TR 'ANY) (compte 21 1))
      ((eq TR 'NUL) (compte 21 0))
      (t (let ; compteur, décrément, liste des valeurs, a, b, al, bt est le trapèze TR
          ((k 1) (d 0.1) (H nil) (a (car (eval TR))) (b (cadr (eval TR))) (al (caddr (eval TR)))
          (bt (cadddr (eval TR))))
          (while (< (+ b bt) k) (set 'H (cons 0 H)) (set 'k (- k d)))
          (while (< b k) (set 'H (cons (1+ (divide (- b k) bt)) H)) (set 'k (- k d)))
          (while (< a k) (set 'H (cons 1 H)) (set 'k (- k d)))
          (while (< (- a al) k) (set 'H (cons (1+ (divide (- k a) al)) H)) (set 'k (- k d)))
          (while (< -1 k) (set 'H (cons 0 H)) (set 'k (- k d)))
          (cons (if (eq TR 'NB) 1 0) H)) )))

(de compte (n v) (if (eq n 0) nil (cons v (compte (1- n) v))))
(setq PRED '(NB NS ZE PS PB) NB '(-1 -1 0 0.5) NS '(-0.5 -0.5 0.5 0.5)
      ZE '(0 0 0.5 0.5) PS '(0.5 0.5 0.5 0.5) PB '(1 1 0.5 0) npr 5 ; nb de prédicats
      HISTO (mapcar (lambda (x) (cons x (histog x))) PRED) am 1) ; définit les constantes

(de poss (x P) (cond ; donne la fonction d'appartenance de x au prédicat P
      ((eq P 'NUL) 0)
      ((eq P 'ANY) 1)
      ((and (<=  x -1) (eq P 'NB)) 1)
      ((and (<= 1 x) (eq P 'PB)) 1)
      (t (apply 'posbis (cons x (eval P)))) ))
(de posbis (x a b al bt) (cond ; un prédicat trapézoïdal est donné par (a b alpha beta)
      ((<= x (- a al)) 0)
      ((< x a) (divide (- (+ x al) a) al))
      ((<= x b) 1)
      ((< x (+ b bt)) (divide (- (+ b bt) x) bt))
      (t 0)))

(de mini (x y) (if (< x y) x y))
(de maxi (L M)  (cond ; réalise la liste composée des max dans les listes L et M
          ((null L) M)
          ((null M) L)
          ((< (car L) (car M)) (cons (car M) (maxi (cdr L) (cdr M))))
          (t (cons (car L) (maxi (cdr L) (cdr M))))))
(de troncature (L v) ; L est un histogramme, on renvoie le min avec v
          (if (null L) nil (cons (mini (car L) v) (troncature (cdr L) v))))

(de moyenne (L) (mbis L -1 0 0)) ; donne l'abscisse du barycentre
(de mbis (LR k S pr ) (if (null LR) (if (eq pr 0) 0 (divide S pr))
          (mbis (cdr LR) (+ k 0.1)(+ (* k (car LR)) S) (+ pr (car LR)))))

(de cantor (i j) (if (<= (+ i j) (1+ npr)) ; on parcourt le tableau en diagonale
      (+ (1- j) (div (* (- (+ i j) 2) (1- (+ i j))) 2))
      (- (* npr (1+ npr)) i (div (* (- (1+ (* 2 npr)) i j) (- (+ 2 (* 2 npr)) i j)) 2))))

(de fuzzy (C x y) ; x et y sont les entrées; C la base de règles
      (fuzzybis C x y PRED PRED PRED 1 1 nil nil))
```

```
(de fuzzybis (C x y PRED Px Py i j R v)
(cond ; donne le sous ensemble flou max, Px et Py sont les listes de prédicats
        ((null Px) (* am (moyenne R))); am désigne plus loin l'angle maximal
        ((or (null Py) (eq v 0)) (fuzzybis C x y PRED (cdr Px) PRED (1+ i) 1 R nil))
        ((null v)  (fuzzybis C x y PRED Px Py i j R (poss x (car Px))))
        (t (fuzzybis C x y PRED Px (cdr Py) i (1+ j)
            (maxi R (troncature (cassoc (nth (cantor i j) C) HISTO)
            (mini v (poss y (car Py))))) v) )))
```

NOYAU D'UN CONTRÔLEUR FLOU (SOLUTION AVEC DES RÈGLES SYMBOLIQUES DE N PRÉMISSES ET P CONCLUSIONS EN LISP)

Les règles sont du type ((c1 c2 ... cp) p1 p2 ..... pn) pour n entrées et p sorties (que des symboles pour les prémisses et des symboles dont on prendra les sommets, ou bien des nombres en conclusions)

```
(de repete (x L) (cond ; produit une liste d'autant de fois x qu'il y a d'éléments dans L
            ((listp L) (repete x (length L)))
            ((zerop L) nil)
            (t (cons x (repete x (1- L))))))
(de maxi (a b) (if (< a b) b a))
(de ramene (x a) (cond ; réalise une bijection de [-a, a] dans [-1, 1]
            ((< x (- a)) - 1)
            ((< a x) 1)
            (t  (divide x a))))
(de sommet (S) ; donne l'abscisse du sommet (qui va rester fixe) du prédicat (r = 1/2)
            (selectq  S (NB -1) (NS -0.5) (ZE 0) (ANY 0) (PS 0.5) (PB 1)))
(de opp (pred) ; donne l'opposé du prédicat pred
            (selectq  pred (NB 'PB) (NS 'PS) (ZE 'ZE) (ANY 'ANY) (PS 'NS) (PB 'NB)))
(de coef (E P)
; coef d'application de la règle de liste de prémisses P pour l'entrée E = (e1 e2 ....)
            (coefbis E  P nil))
(de coefbis (E P LC) ; LC liste de coefficients
            (if (or (null E) (null P)) (tnorm LC) (coefbis (cdr E) (cdr P)
            (cons (if (eq (car P) 'ANY) 1
                        (maxi 0 (- 1 (divide (abs (- (sommet (car P)) (car E))) r)))) LC))))
(de maxi (a b) (if (< a b) b a))      (de mini (a b) (if (< a b) a b))
(de tnorm (L) (if (null L) 1 (minibis (car L) (cdr L)))) ; t-norme de Zadeh
(de minibis (X L) (cond
            ((null L) X)
            ((> X (car L)) (minibis (car L) (cdr L)))
            (t (minibis X (cdr L)))))

(de fuzzy (E LR) ; donne une liste en sortie dans [-1, 1]p avec LR liste de règles
        (fuzzybis (coef E (cdar LR)) E (caar LR) (repete 0 (caar LR)) 0 (cdr LR)))
(de fuzzybis (co E LC S SC LR)
; c coeff E entrées LC liste des conclusions S sortie, SC somme des coef
(fuzzyter E (ajou co (if (numberp (car LC)) LC (mapcar 'sommet LC)) S) (+ co SC) LR))
(de fuzzyter (E S SC LR) ; E vecteur entrée, S vecteur sortie
        (if  (null LR) (if (eq SC 0) (repete 0 S) (mapcar (lambda (x) (divide x SC)) S))
        (fuzzybis (coef E (cdar LR)) E (caar LR)  S SC (cdr LR))))
(de ajou (co L M) ; produit la liste co*L + M terme à terme
        (if (or (null L) (null M)) nil (cons (+ (* co (car L)) (car M)) (ajou co (cdr L) (cdr M)))))
```

EVITEMENTS D'OBSTACLES EN LISP

Dans ce programme, (exemple du parcours d'obstacles du chapitre 5) un contrôleur flou est déterminé par la structure (r h R1 R2 R3 ....) où r est la valeur de la demi-base des prédicats (r = 1/2 normalement) et h ≥ 1 leur hauteur, les règles R = (pr deg (c1 c2 ... cp) p1 p2 ..... pn) pour n entrées et p sorties (uniquement des symboles NB NS ZE PS PB ANY pour les prémisses et des symboles dont on prendra les sommets, ou bien des nombres en conclusions. Exemple de contrôleur ayant deux règles indiquant qu'un obstacle proche nécessiteun virage et un ralentissement brusques, et une règleaccélératrice si la distance au plus proche obstacle est grande:
(set 'C '((24 0.7 1.15 (0 0 (pb nb) ze ns) (0 0 (nb nb) ze ps) (0 0 (ze ps) pb any))
On verra un exemple voisin en fin de paragraphe.
La fonction "coef" est modifiée par rapport au paragraphe précédent :

```
(de coef (E prm r h) ; coef d'application de la liste de prémisses prm pour l'entrée E = (e1 e2)
        (coefbis E  prm nil r h))
(de coefbis (E prm LC r h) ;LC liste de coefficients, r demi-base h hauteur
        (if (or (null E) (null prm)) (tnorm LC)
        (coefbis (cdr E) (cdr prm) (cons (if (eq (car prm) 'ANY) 1
        (mini 1 (maxi 0
                (* h (- 1 (divide (abs (- (sommet (car prm)) (car E))) r)))))) LC) r h)))
```

La fonction "fuzzy" donne, suivant la méthode de Sugeno, la liste des sorties dans $[-1, 1]^p$ avec LR liste de règles de même priorité la précédant c.a.d (LR u1 u2 .... up) où LR a été modifiée, renvoie nil si LR ne s'applique pas.

```
(de fuzzy (E LR r h)
        (fuzzybis nil (car LR) (coef E (cdddar LR) r h) E (caddar LR)
                (repete 0 NC) 0 (cdr LR) r h)) ; LV règles vues, Rc règle courante
(de fuzzybis (LV Rc co E LC S SC LR r h)
; co coeff E les entrées LC liste des conclusions S sortie, SC somme des coef
        (fuzzyter (cons (mcons (car Rc) (+ co (cadr Rc)) (cddr Rc)) LV)
        ; la règle Rc est modifiée et ajoutée à LV
        E (ajou co (if (numberp (car LC)) LC (mapcar 'sommet LC)) S) (+ co SC) LR r h))
(de fuzzyter (LV E S SC LR r h) ; E vecteur entrée, S vecteur sortie
        (if  (null LR) (cons LV (if (eq SC 0) nil ; (repete 0 NC)
        (mapcar (lambda (x) (divide x SC)) S)))
        (fuzzybis LV (car LR) (coef E (cdddar LR) r h) E (caddar LR) S SC (cdr LR) r h)))

(de decision (E C) ; renvoie pour une liste d'entrées E et un contrôleur C = (r h R1 R2 R3 ...)
; le vecteur sortie précédée de C où ont été modifiés les degrés d'application des règles
      (remettre (car C) (cadr C) (decibis E (car C) (cadr C) nil (list (caddr C)) 0 (cdddr C))))

(de decibis (E r h LV Lpr pr LR)
; LV règles vues non appliquées, Lpr règles de priorité pr, LR reste à voir
        (cond     ((null LR) (deciter E r h LV (fuzzy E Lpr r h) pr nil))
                  ((equal (caar LR) pr) (decibis E r h LV (cons (car LR) Lpr) pr (cdr LR)))
                  (t (deciter E r h LV (fuzzy E Lpr r h) pr LR))))
(de deciter (E r h LV F pr LR) ; F est de la forme (Règles de forces modifiées s1 ... sp)
        (if (null (cdr F)) (if (null LR) (cons (append LV (car F)) (repete 0 (cddr (cdaar F))))
                (decibis E r h (append LV (car F)) nil (caar LR) LR))
                (cons (append (append LV (car F)) LR) (cdr F)))) ;  -> (Règles s1 ... sp)
(de remettre (a b L) ; renvoie ((a b X) m n k...) si L = (X m n k ...)
        (cons (mcons a b (car L)) (cdr L)))
```

Fonctions graphiques pour le dessin du champ d'obstacles :

```
(de tan (x) (divide (sin x) (cos x)))
(de degre (a) (truncate (* 180 (// a pi))))
(de arc (a b c d) (cond ; donne l'arc en radians du vecteur (a, b) (c, d) avec l'horizontale
        ((eq b d) (if (< a c) 0 pi))
        ((eq a c) (if (< b d) (// pi 2) (- (// pi 2))))
        ((< c a) (if (< b d) (+ pi (arc c d a b)) (- (arc c d a b) pi)))
        (t (atan (divide (- d b) (- c a))))))

(de rpi (a) (cond ; représentation dans ]-pi, pi]
        ((< pi a) (rpi (- a (* 2 pi))))
        ((<= a (- pi)) (rpi (+ a (* 2 pi))))
        (t a)))
(de point (u v) (move-to u v) (line-to u v) )

(de croix (u v) (move-to (1- u) v) (line-to (1+ u) v) (move-to u (1- v))(line-to u (1+ v)))
(de sqr (x) (* x x))

(de obstacles (n) ; produit une liste de n obstacles aléatoires symbolisés par des points
        (let ((E nil) (r 0) (a 0))
        (repeat n (setq r (distrib ry) a (* pi (divide (random -98 98)99))
        E (cons (list (truncate (+ u0 (* r (cos a)))) (truncate (+ v0 (* r (sin a))))) E))) E))

(de dessin (E) (if (null E) nil (apply 'point (car E)) (dessin (cdr E))))

(de cercle (x y r) (let ((ag 0)) (move-to (+ x r) y) (repeat 32
      (line-to (truncate (+ x (* r (cos (set 'ag (+ ag 0.2)))))) (truncate (+ y (* r (sin ag))))))))

(de radar (u v dir L d a)
; donne le couple (d a) de l'obstacle de L le plus proche de (u v) dans la dir
        (if (null L) (list (ramene d dm) (ramene a am))
        (radar1 u v dir (abs (- u (caar L))) (abs (- v (cadar L))) L d a)))
(de radar1 (u v dir x y L d a)
        (if (or (< dm x) (< dm y)) (radar u v dir (cdr L) d a)
        (radar2 u v dir x y (- (arc u v (caar L) (cadar L)) dir) (cdr L) d a)))
(de radar2 (u v dir x y ag L d a)
        (if (or (< am ag) (< ag (- am))) (radar u v dir L d a)
        (radar3 u v dir (sqrt (+ (* x x) (* y y))) ag L d a)))
(de radar3 (u v dir di ag L d a) (if (< di d) (radar u v dir L di ag) (radar u v dir L d a)))

(de bord (u v a)
; donne la distance relative à dm, de (u, v) au bord du disque dans la direction a
        (ramene (let ((co (cos a)) (ta (tan a)))
        (if (eq co 0) (- (sqrt (- (sqr ry) (sqr (- u u0)))) (* (sgn a) (- v v0)))
        (let ((del (sqrt (- (* (sqr ry) (1+ (sqr ta))) (sqr (+ (- (* ta (- u u0)) v) v0)))))
                (x  (- u0 u (* ta (- v v0)))))
                (set 'x (- x del)) (if (< (* x co) 0) (set 'x (+ x (* 2 del)))) (* x co)))) dm))

(de cap (u v dir) (capbis (radar u v dir OB dm 0) (bord u v (+ dir am)) (bord u v (- dir am))))
(de capbis (rad d1 d2) ; renvoie le couple relatif (d a) du plus proche obstacle (bord compris)
        (if (< d1 (car rad))            (if (< d2 d1) (list d2 -1) (list d1 1))
                                        (if (< d2 (car rad)) (list d2 -1) rad)))
```

La fonction principale "parcours" permet d'effectuer une expérience en prenant des mesures.

```
(de parcours (C) ; C= (r h (pr f (da dx) g-d f ) (pr f (da' dx') g'-d' f' ) .....)
        (let ((u u0) (v v0) (dir0 2)(dir 2) (pas0 ds) (Z nil) (k 0) (tr 0) (cp 0) (val 0) (da 0)
        (pas ds) (drap t) )
        (print '" Règles : ") (mapcar 'print (cddr C))
        (pen-size 2 2) (dessin OB) (cercle u0 v0 ry) (pen-size 1 1) (move-to u v)  (croix u v)
        (while  (and (< k km) drap) ; drap signifie "plus petite distance à un obstacle > ds"
        (setq val (+ val pas) cp (cap u v dir) Z (decision cp  C) C (car Z) da (* (cadr Z) am)
                dir (+ dir da) pas (mini (maxi ds (+ pas (* (caddr Z) acc))) pm))
        (if (and (< (* (car cp) dm (cos (- (cadr cp) da))) pas)
                (< (* (car cp) dm (sin (abs (- (cadr cp) da)))) ds)) (set 'drap nil))
        (line-to    (set' u (+ u (truncate (* pas (cos dir)))))
                (set 'v (+ v (truncate (* pas (sin dir)))))))            (croix u v)
        (if (< (sqr ry) (+ (sqr (- u u0)) (sqr (- v v0)))) (set 'drap nil))
        (setq tr (+ tr (abs (- 0.5 (divide(- pas pas0)(* 2 acc)) (divide (abs (- dir dir0)) am))))
                pas0 pas dir0 dir k (1+ k)))
        (cons (print '" Reste distance " ; val est la somme des pas donc la distance parcourue
        (set 'val (truncate (* 100. (- 1 (divide val (* km pm)))))) ; val % parcours restant
                '" Tenue de route " (set 'tr (truncate (// (* 100. tr) k)))
                '" (" k '" pas) Valeur " (truncate (agreg (nbsymbol C) tr val))) )))
```

Agreg est une fonction d'agrégation entre les trois critères à optimiser.

Données globales :
(setq PRED '(ANY NB NS ZE PS PB) NH 2 NC 2 pi 3.14159 ry 130 u0 300 v 0 150
OB (obstacles 200) km 30 ds 3 pm 30 acc 4 dm 35 am (// (* pi 45) 180))
Exemple :

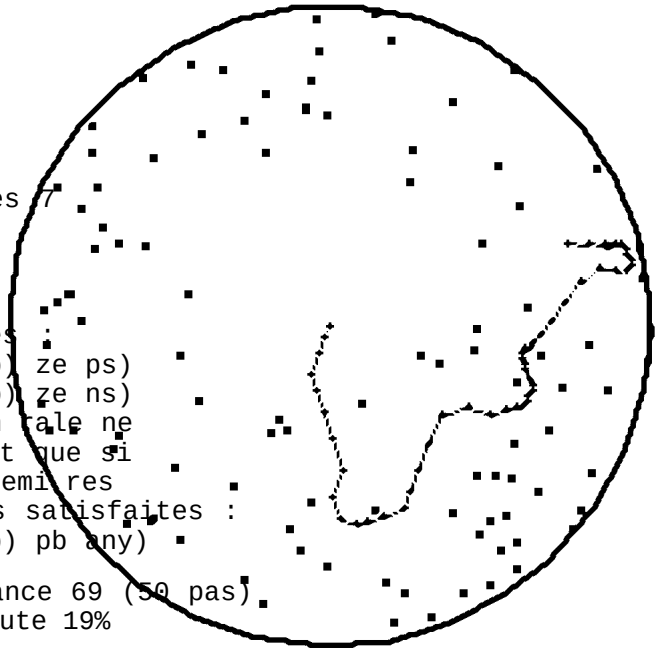

Figure A7.1 Parcours d'un robot à 3 règles modifiant sa direction et sa vitesse.

Tout un module de fonctions lisp complète le programme afin de simuler une évolution pour trouver de "bons contrôleurs" C, ceux-ci sont en fait codés pour moitié, le programme se chargeant de les compléter par symétrie et de vérifier la cohérence de leurs paramètres.

NOYAU D'UN CONTRÔLEUR FLOU

(SOLUTION EN LISP AVEC DES RÈGLES NUMÉRIQUES DE R^2 DANS R^2 DU TYPE (A B R C D) SIGNIFIANT «SI (X, Y) VÉRIFIE LE PRÉDICAT CÔNE DE CENTRE (A, B) ET RAYON R, ALORS (U, V) EST (C, D)».

```
(de fuzzy (LR x y) ; renvoie le couple (u v) de [-1 1] en fonction de x y de [-1, 1]
      (if (null LR) '(0 0) (fuzzybis (car LR) x y 0 0 0 (cdr LR))))
(de fuzzybis (R x y u v Sc RR) ; règle R, u, v numérateurs, Sc somme des coefficients
      (fuzzyter (maxi (- 1 (divide (maxi (abs (- x (car R))) (abs (- y (cadr R)))) (caddr R)))0)
          (cadddr R) (car (last R)) x y u v Sc RR))
(de fuzzyter (cf c d x y u v Sc RR)
; cf dans [-1, 1] est le coef d'application de la dernière règle étudiée
      (if          (null RR) (if (eq (+ cf Sc) 0) '(0 0)
          (list (divide (+ u (* cf c)) (+ Sc cf)) (divide (+ v (* cf d)) (+ Sc cf))))
          (fuzzybis (car RR) x y (+ u (* cf c)) (+ v (* cf d)) (+ Sc cf) (cdr RR))))
```

Exemple : (print (fuzzy '((0 0 2 2 4) (2 0 2 6 8)) 1 0)) = (4 6)

MODUS-PONENS GÉNÉRALISÉ

L'intérêt d'un langage fonctionnel comme Lisp est de pouvoir ici définir la fonctionnelle appelée «mpg», qui, ayant pour données une t-norme, une relation d'implication et trois fonctions d'appartenance μA, μB, et μA', va renvoyer la fonction μB'.

```
(de weber (a b) (cond ((eq a 1) b) ((eq b 1) a) (t 0)))          ; t-normes
(de lukas (a b) (maxi 0 (1- (+ a b))))
(de einstein (a b) (divide (* a b) (- (+ (* a b) 2) a b)))
(de proba (a b) (* a b))
(de hamacher (a b) (if (eq (zadeh a b) 0) 0 (divide (* a b) (- (+ a b) (* a b)))))
(de zadeh (a b) (if (< a b) a b)) ; également nommée maxi  dans les paragraphes précédents.

(de reichenbach (p q) (1+ (- (* p q) p)))                        ; implications
(de willmott (p q) (maxi (- 1 p) (zadeh p q)))
(de mamdani (p q) (zadeh p q))
(de rescher (p q) (if (< p q) 1 0))
(de kleene (p q) (maxi (- 1 p) q))
(de godel (p q) (if (< q p) q 1))
(de goguen (p q) (if (eq p 0) 1 (zadeh 1 (divide q p))))
(de implukas (p q) (- 1 (maxi 0 (- p q))))
(de larsen (p q) (* p q))
(df mpg (norme implic muA muB muAp)                              ; modus-ponens
      (list 'lambda '(y)
          (list 'mpgbis norme implic muA (list 'funcall muB 'y) muAp -1 0)))
(de mpgbis (norme implic muA my muAp x sup)
      (if (> x 1) sup)
      ; c'est la valeur de muB' renvoyée, les ensembles considérés étant dans [-1, 1]
      (mpgbis norme implic muA my muAp (+ 0.1 x) ; 0.1 est l'incrément utilisé
      (maxi sup (funcall norme (funcall muAp x) (funcall implic (funcall muA x) my)))) ))
```

Exemple, on définit A, B, A' et des fonctions de mise à l'échelle de l'écran

```
(de muA (x) (cond ((< x -1) 0) ((< x -0.5) (* 2 (1+ x))) ((< x -0.2) 1) ((< x 0) (* x -5)) (t 0)))
(de muAp (x) (cond ((< x -0.8) 0) ((< x -0.6) (+ 4 (* 5 x))) ((< x -0.4) (- -2 (* 5 x))) (t 0)))
(de muB (x) (cond ((< x 0.1) 0) ((< x 0.6) (- (* 2 x) 0.2)) ((< x 0.8) 1)
                  ((< x 1) (+ (* -5 x) 5)) (t 0)))
(de phi (x) (truncate (+ 50 (* 200 (1+ x)))))
(de phir (u) (divide (- u 250) 200)) ; est la réciproque de phi
(de psi (y) (truncate (+ 50 (* 100 (- 1 y)))))

(de graphe (f) (move-to 0 150); trace la fonction f pour 0 < y < 1 et -1 <  x < 1
      (let ((x 0)) (repeat 500 (line-to x (psi (funcall f (phir x)))) (set 'x (1+ x))) ))
(de essai (norme implic)
      (pen-size 2 2) (move-to 500 150) (line-to 0 150) (graphe 'muA) (graphe 'muB)
      (pen-size 1 1) (graphe 'muAp) (graphe (mpg norme implic 'muA 'muB 'muAp))
      (print "'t-norme : " norme "'      implication : " implic))
```

Nous présentons ci-dessous un échantillon (pour la même prémisse «X est A", la même conclusion «Y est B" et le même fait observé A' comme ci-dessus) de la conclusion approchée (en traits fins) pour différentes t-normes et différentes implications.

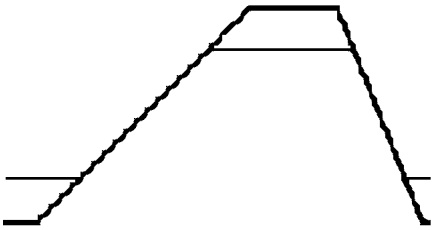

Figure A.7.2 Toute norme sauf Weber, implication de Willmott

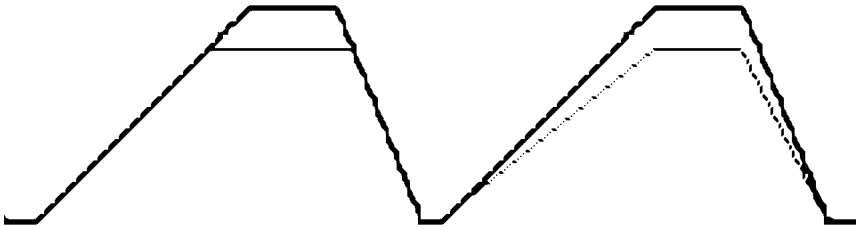

Figure A.7.3 Toutes normes sauf Weber, imp Mamdani et Proba, Hamacher et imp Larsen

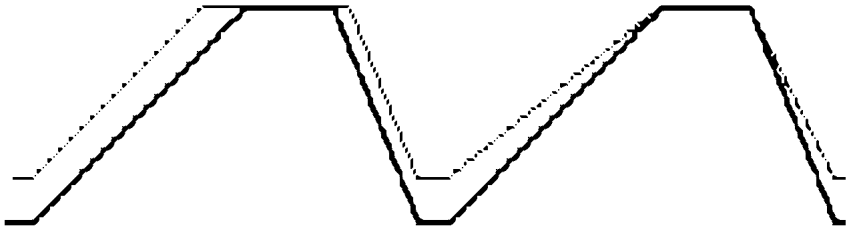

Figure A.7.4 Norme proba, implications Lukasiewicz et Reichenbach

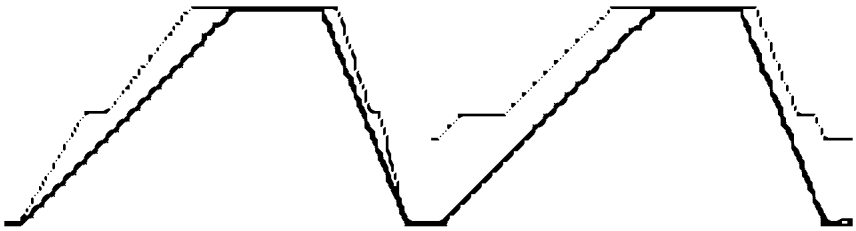

Figure A.7.5 Norme de Zadeh, implications Goguen et de Lukasiewicz

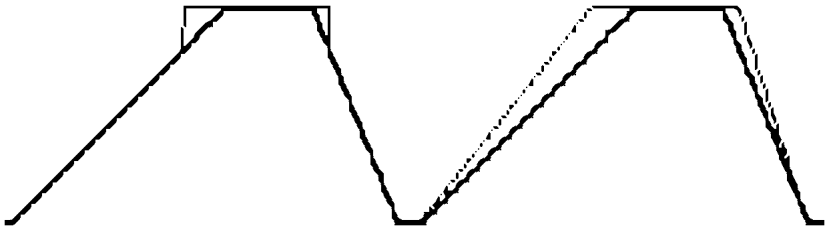

Figure A.7.6 Toute norme sauf Weber implication Gödel

Toute norme sauf Weber et Zadeh implication de Goguen

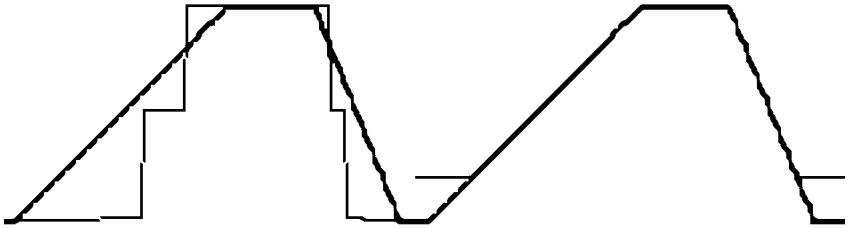

Figure A.7.7 Toute norme, implication de Rescher, Toute norme sauf Weber, imp. Kleene Norme Weber et implications de Gödel, Goguen et Lukasiewicz

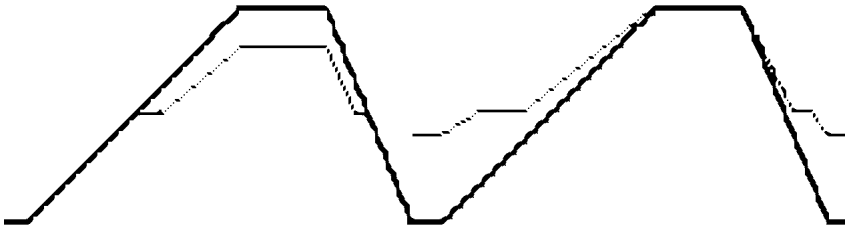

Figure A.7.8 Norme de Zadeh, implications de Larsen et de Reichenbach

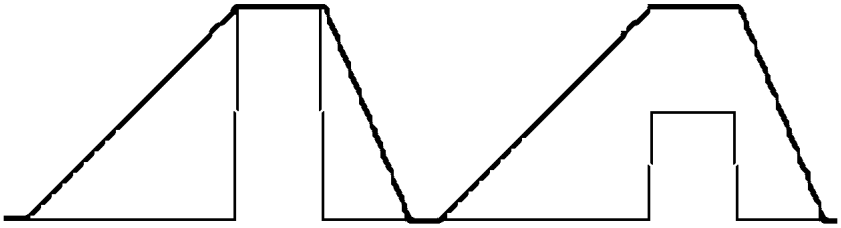

Figure A.7.9 Norme Weber, implication Kleene et à droite de Mamdani, Larsen, Willmott ou Reichenbach

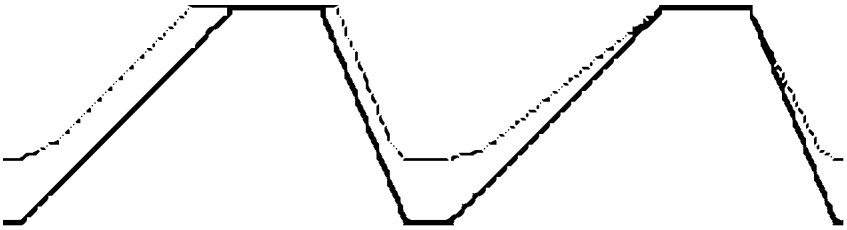

Figure A.7.10 Norme de Hamacher, implications de Lukasiewicz et de Reichenbach

Remarque, les implications de Lukasiewicz, Reichenbach et Willmott ont tendance à donner une incertitude en dehors du support de B, elles peuvent être considérées comme trop fortes.