Laborator 6

Mișcare procedurală

As. ing. Alexandru Dumitrache
As. ing. Raluca Tudorie
www.scr.cimr.pub.ro

Introducere

Mișcările elementare ale robotului, folosite până acum, au fost:

- Mișcare în linie dreaptă în spațiul cartezian (`MOVES`);
- Mișcare interpolată liniar în spațiul articulațiilor (`MOVE`). Acest tip de mișcare este un arc de cerc în spațiul cartezian.

Până acum, programele de mișcare aveau forma:

```
APPROS a, 100
BREAK
MOVE a
BREAK
MOVES b
BREAK
```

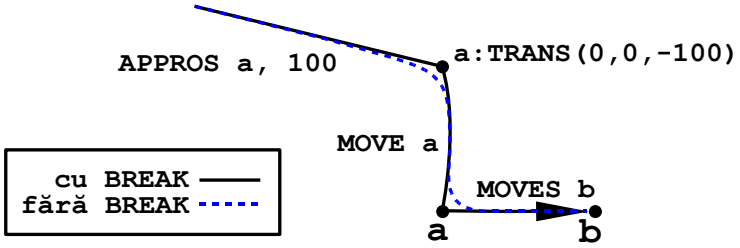

Figura 6.1: *Exemplu de mișcare executată cu și fără* `BREAK`

În acest caz, robotul se oprește în punctele (locațiile) programate înainte de a începe mișcarea spre următorul punct (Fig. 6.1 - linie continuă).

Dacă se scot instrucțiunile `BREAK` din acest program, robotul va netezi tranziția între două segmente de mișcare succesive (Fig. 6.1 - linie întreruptă).

În această situație, în care se emite o instrucțiune de mișcare în timp ce robotul execută o altă mișcare, rezultând o traiectorie netedă, programul V^+ execută o *mișcare procedurală* [1].

Punctele robot folosite în mișcarea procedurală pot fi învățate manual, sau pot fi calculate matematic.

Avem două cazuri:

- Distanța dintre două puncte robot consecutive este mare (zeci sau sute de mm). În acest caz, traiectoria netezită se poate depărta semnificativ de traiectoria neprocedurală (cu `BREAK` după fiecare mișcare).
- Distanța dintre două puncte robot succesive este mică (de ordinul milimetrilor sau mai mică). În acest caz, traiectoria poate fi interpolată folosind fie `MOVE`, fie `MOVES`, diferența fiind minimă (deoarece segmentele sunt foarte scurte).

[1] Acest comportament este valabil atunci când switch-ul `CP` (Contiuous Path) este activ. Dacă se dezactivează acest switch, traiectoria nu mai este netezită, ci se ating punctele prescrise chiar dacă programul nu conține explicit instrucțiunea `BREAK`. Setarea implicită a switch-ului `CP` este `ON`.

Controlul vitezei... reloaded

O instrucțiune de mișcare simplă (neproceduală) cuprinde următoarele etape:

- accelerare
- deplasare cu viteză constantă
- decelerare
- anularea erorilor de poziționare

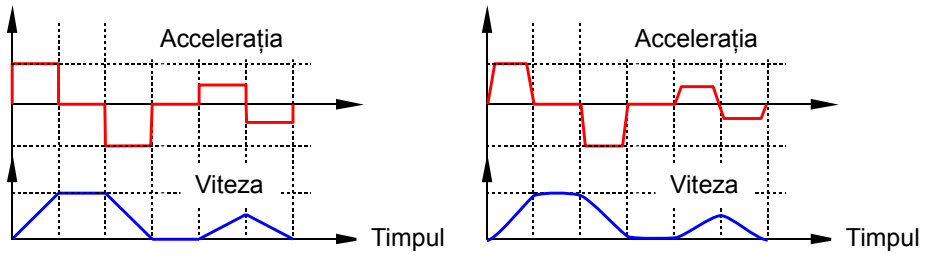

Figura 6.2: *Profil de accelerație dreptunghiular (a) și trapezoidal (b)*

Aceste etape formează *profilul de accelerație*. Cazul cel mai simplu este atunci când forma acestui profil este dreptunghiulară (Fig. 6.2 a). În Fig. 6.2 (b) accelerația crește liniar de la 0 la valoarea maximă, iar profilul de viteză devine rotunjit la capete, efectul fiind o mișcare mai lină.

Viteza program (setată cu `SPEED ... [ALWAYS]` în program) este viteza maximă, atinsă în etapa de mișcare cu viteză constantă. Pentru mișcările liniare în spațiul cartezian (`MOVES`, `APPROS` etc.), profilul de accelerație este calculat pentru punctul condus (efectorul terminal); pentru mișcările interpolate pe articulații (`MOVE`), profilul de accelerație este același pentru fiecare articulație a robotului, până la un factor de scalare.

Atunci când un segment de mișcare este rulat cu viteza program `100`, robotul va folosi rata de accelere / decelerare maximă posibilă. Dacă segmentul este prea scurt, se vor executa doar etapele de accelerare și decelerare (eventual tranziție), sărind peste etapa de mișcare cu viteză constantă.

Dacă viteza program este specificată în procente, viteza maximă obținută este raportată la viteza atinsă la 100%. Reducerea vitezei se face prin micșorarea ratelor de accelerare / decelerare (vezi `SCALE.ACCEL`) și prin apariția (sau extinderea) etapei de mișcare cu viteză constantă.

Forma profilului de accelerație nu este influențată de viteza monitor.

O mișcare rulată cu viteza monitor 100 și viteza program 20 va fi diferită de o mișcare rulată cu viteza monitor 20 și viteza program 100 între aceleași puncte. Diferența este subtilă, în sensul că viteza maximă atinsă va fi aceeași (20%), însă forma profilului de accelerație va fi diferită.

Profilul de accelerație pentru mișcarea procedurală cuprinde:

- accelerare, pentru primul segment de mișcare;
- decelerare și anulare a erorilor, pentru ultimul segment;
- etape de tranziție de la un segment la altul;
- mișcare cu viteză constantă.

Lista completă a etapelor se găsește în manualul V^+ Language Reference Guide, la instrucțiunea `STATE` apelată cu `select=11`.

Dacă segmentele care compun o mișcare procedurală sunt lungi, *viteza maximă* a efectorului terminal poate fi exprimată, pe lângă procente (implicit) și în milimetri pe secundă: `SPEED nnn MMPS`.

În cazul mișcării procedurale formată din segmente scurte, robotul va petrece o mare parte din timp în etapele de tranziție între segmente (care sunt executate cu o viteză mai mică). Rezultă că viteza medie de lucru va fi mai mică decât cea setată prin `SPEED`.

În acest caz, viteza poate fi controlată precis cu instrucțiunea `DURATION`, care stabilește *durata minimă* permisă pentru un segment de mișcare [2].

```
DURATION d [ALWAYS]              ; d exprimat în secunde
```

Pentru ca `DURATION` să aibă efect, vom seta `SPEED 100 ALWAYS`. Atunci când în program setăm viteza folosind și `SPEED` și `DURATION`, V^+ va alege varianta cea mai lentă (pentru fiecare segment de mișcare în parte).

V^+ are nevoie de un ciclu sistem (16 ms) pentru a planifica un segment de mișcare; de aceea, argumentul lui `DURATION` nu poate fi mai mic de `0.016`.

Dacă argumentul lui `DURATION` este mai mic de `0.016` secunde [3] , V^+ îl va seta automat la `0.016`, încetinind mișcarea robotului [4].

Vitezele setate cu `SPEED` și `DURATION` sunt scalate cu viteza monitor. De exemplu, dacă viteza monitor este 50 și viteza program este `SPEED 30 MMPS`, efectorul terminal se va deplasa cu maxim 15 mm/s.

[2] `DURATION` funcționează corect doar pentru segmentele din interiorul unei mișcări procedurale. Primul și ultimul segment se vor executa mai incet. Pentru detalii consultați V^+ Language Reference Guide.

[3] În documentația V^+ se precizează că segmentele de mișcare de tip `MOVE` sunt planificate în 16 ms, iar segmentele de tip `MOVES` necesită 32 ms. Experimental s-a observat că pe roboții din laborator, `MOVES` necesită doar 16 ms, la fel ca si `MOVE`.

[4] În versiunile anterioare de V^+ , setarea unei valori prea mici pentru `DURATION` putea să ducă la o mișcare sacadată.

Optimizarea operației *Pick and Place*

If it ain't broke, it doesn't have enough features yet.

La întâlnirea unui `BREAK` după o mișcare executată cu viteză mare (în special după mișcările prin aer), robotul frânează până la oprire, folosind rata de accelerație maximă posibilă. Dacă nu ar frâna până la oprire, accelerația necesară pentru schimbarea direcției ar fi infinită [5].

În cazul unei traiectorii netezite, robotul nu mai este nevoit să frâneze până la oprire la sfârșitul unui segment de mișcare. Efectele pozitive sunt:

- Motoarele robotului sunt solicitate mai puțin. Cuplul în motoare, direct proporțional cu accelerația, este mai mic.
- Vibrațiile sunt mult mai mici. Bucla de reglare răspunde mult mai ușor la o schimbare lentă a referinței, decât la o schimbare bruscă.
- Timpul necesar pentru execuția mișcării scade semnificativ.

Prețul plătit pentru aceste avantaje:

- Efortul de calcul este mai mare (pentru controller-ul robot)
- Traiectoria netedă se poate abate semnificativ de la traiectoria formată din segmente distincte.

Efortul de calcul crescut este neglijabil în cazul operației de Pick and Place, deoarece punctele robot succesive sunt la distanță suficient de mare, încât să permită controller-ului să planifice tranziția dintre două segmente în timp util (fără a opri robotul pentru a face calcule).

Fenomenul de abatere a traiectoriei de la punctele învățate ne deranjează în special atunci când dorim ca robotul să execute o mișcare liniară pe o distanță minimă impusă, de exemplu, la introducerea (extragerea) piesei în (din) axele metalice.

Formula exactă a traiectoriei rotunjite nu este documentată în V^+, însă se poate observa în Fig. 6.3 că prima jumătate din primul segment și ultima jumătate din ultimul segment sunt executate fără rotunjire, exact ca în cazul mișcării neprocedurale.

[5] Schimbarea bruscă a direcției poate fi privită ca o mișcare circulară pe un arc a cărui rază tinde spre 0 $\Rightarrow$ dacă viteza tangențială nu ar fi 0, accelerația ar tinde la infinit.

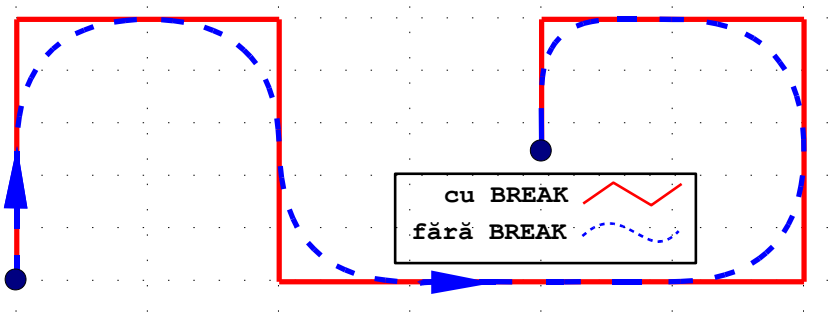

Figura 6.3: *Traiectorie de probă executată pe Cobra s600 cu viteza program 100%*

Pentru mișcarea de tip Pick and Place, avem două soluții posibile:

- setăm o distanță de apropiere/depărtare (`z.pick` și `z.place`) de minim două ori mai mare decât înălțimea axurilor;
- așteptăm ca robotul să parcurgă distanța minimă impusă (înălțimea axurilor), și abia apoi emitem următoarea instrucțiune de mișcare:

```
...
CLOSEI
DEPARTS 100
WAIT DISTANCE(HERE, DEST) < 30
MOVE #safe
```

Posibilități de optimizare suplimentare:

- Se pot folosi valori diferite pentru parametrul `HAND.TIME` la deschidere și la închidere. La deschiderea gripper-ului, piesa este eliberată de îndată ce degetele au început să se depărteze. La închidere în schimb, piesa este fixată numai după ce degetele au ajuns la capătul cursei, și-au stabilizat mișcarea și vibrațiile și au aplicat o presiune (implicit o forță de frecare) suficient de mare pentru a ridica piesa.

- Se poate regla rata de accelerare și decelerare folosind `ACCEL`:

  - `ACCEL factor.accelerare, factor.decelerare`
  - `ACCEL 100, 100` ; valoarea implicită
  - `ACCEL 100, 50` ; decelerarea de două ori mai lentă

  Atunci când robotul vibrează, este posibilă micșorarea timpului de execuție prin reducerea (!) vitezei / accelerației, deoarece se diminuează suprareglajul, iar eroarea de poziționare se va încadra mai repede în intervalul admis.

- Se poate seta toleranța (eroarea admisă) în cazul buclei de reglare:
  - `FINE tol [ALWAYS]` ; toleranța este în procente
  - `COARSE tol [ALWAYS]` ; față de valoarea implicită

  Setarea implicită este `FINE 100 ALWAYS`.

- Se poate dezactiva etapa de anulare a erorilor de poziționare la întâlnirea unui `BREAK` (folosiți cu atenție!):
  - `NONULL [ALWAYS]`

  Setarea implicită este `NULL ALWAYS`.

```
.PROGRAM ppo(pick, place)

    ; Laborator 6 - Pick and Place optimizat

    GLOBAL #safe
    AUTO z.pick, z.place
    z.pick = 100
    z.place = 100

    PARAMETER HAND.TIME = 0.2

    APPRO pick, z.pick
    SPEED 50
    MOVES pick
    CLOSEI
    DEPARTS z.pick

    PARAMETER HAND.TIME = 0.02

    APPRO place, z.place
    SPEED 50
    MOVES place
    OPENI
    DEPARTS z.place
.END
```

Traiectorii descrise matematic

Anything practical you learn will be obsolete before you use it, except the complex math, which you will never use.

Punctele robot pot fi calculate matematic pentru a obține traiectorii definite de utilizator. Pentru o aproximare cât mai bună, prezintă interes cazul în care două puncte robot succesive sunt foarte apropiate (distanța este de ordinul milimetrilor sau mai mică).

Punctele pot fi stocate într-un vector:

```
FOR i = 0 TO LAST(puncte[])
    MOVE puncte[i]
END
```

Sau pot fi calculate pe loc:

```
FOR t = 0 TO 360 STEP 2
    MOVE SHIFT(centru BY raza * COS(t), raza * SIN(t), 0)
END
```

În acest caz, viteza va fi specificată cu `DURATION`. Dacă toate segmentele au lungimi egale `ds` [mm] și se dorește o viteză `v` [mm/s], se va folosi:

```
SPEED 100 ALWAYS
DURATION ds / v ALWAYS
```

Dacă segmentele au lungimi diferite `ds[i]`, durata va fi specificată pentru fiecare instrucțiune de mișcare în parte, astfel încât să se obțină viteza constantă `v` [mm/s]:

```
SPEED 100 ALWAYS
FOR i = 1 TO LAST(puncte[])
    ds[i] = DISTANCE(puncte[i-1], puncte[i])
    DURATION ds[i] / v
    MOVE puncte[i]
END
```

Dacă argumentul lui `DURATION` va rezulta mai mic de `0.016`, viteza robotului va fi mai lentă decât cea specificată. Pentru a obține viteza dorită, vom mări pasul de discretizare, însă ne vom mulțumi cu o mișcare mai puțin precisă.

Robotul care desenează

Nume de cod: Robotul Picasso

Avem la dispoziție:

- un pix modificat astfel încât să poată fi fixat în gripper-ul robotului;
- rutina `ia.pixul()`, care prinde pixul în gripper, de pe un suport;
- rutina `pune.pixul()`, care pune pixul înapoi pe suport și se întoarce în punctul `#safe`;
- o foaie de hârtie de tip *Post-it*, 70×70 mm, pe care va desena robotul;
- rutina `vezi.foaia()`, care folosește camera mobilă de pe braț pentru a detecta poziția și orientarea foii de hârtie;
- un punct robot `p0` învățat cu vârful pixului în colțul stânga jos al foii de hârtie, în sistemul de referință `loc.foaie` calculat de subprogramul `vezi.foaia`.

```
.PROGRAM ia.pixul()
    PARAMETER HAND.TIME = 0.3
    TYPE "Iau pixul..."
    SPEED 30 ALWAYS
    OPEN
    APPRO loc.pix, 100
    MOVES loc.pix
    CLOSEI
    DEPARTS 100
    TYPE "Am luat pixul."
.END

.PROGRAM pune.pixul()
    TYPE "Pun pixul inapoi..."
    PARAMETER HAND.TIME = 0.3
    SPEED 30 ALWAYS
    APPRO loc.pix, 100
    MOVES loc.pix
    OPENI
    DEPARTS 100
    TYPE "Am pus pixul, ma retrag in #safe."
    MOVE #safe
.END
```

```
.PROGRAM vezi.foaia()
    ; Localizeaza foaia de hartie folosind AdeptSight
    ; Returneaza un sistem de coordonate local pe foaia
    ; de hartie, in variabila globala loc.foaie
    ;
    ; Daca foaia de hartie nu este gasita,
    ; variabila loc.foaie este stearsa din memorie (nedefinita).

    GLOBAL loc.foaie

    AUTO $ip, elem[6], loc.vor, rz

    TYPE "Caut foaia..."
    $ip = "172.16.200.20"
    PARAMETER VTIMEOUT = 5
    SPEED 30
    MOVE #loc.img
    BREAK

    VRUN $ip, 1
    IF VRESULT($ip, 1, 2, 1, 1310, 1, 1) == 1 THEN
        TYPE "Am gasit foaia."
        SET loc.foaie = VLOCATION($ip, 1, 2, 1, 1371, 1, 1)

        ; artificiu (ar trebui sa mearga fara el)
        SET loc.vor = VLOCATION($ip, -1, 2, 1, 10050, 1, 1)
        DECOMPOSE elem[] = loc.vor
        rz = elem[5]
        SET loc.foaie = loc.foaie:RZ(rz)
    ELSE
        TYPE "Nu am gasit foaia."
        MCS "deletel loc.foaie"
    END
.END
```

Programul utilizează instrucțiuni de vedere artificială, care vor fi prezentate mai târziu.

Pentru a desena un cerc, vom rula următorul program:

```
.PROGRAM cerc()
    GLOBAL loc.foaie
    CALL vezi.foaia()
    IF DEFINED(loc.foaie) THEN
        CALL ia.pixul()
        CALL deseneaza.cerc()
        CALL pune.pixul()
    END
.END

.PROGRAM deseneaza.cerc()
    GLOBAL loc.foaie, p0
    AUTO centru, start, raza, t, pas.mm, pas.deg
    raza = 30
    pas.mm = 2
    pas.deg = (pas.mm / raza) * 180 / PI

    SET centru = SHIFT(p0 BY 35, 35, 0)
    SET start = SHIFT(centru BY raza, 0, 0)

    ; robotul va folosi setarea SPEED pentru segmentele lungi
    ; si DURATION pt. cele scurte (va alege varianta cea mai lenta)

    SPEED 100 ALWAYS
    DURATION pas.mm / 50 ALWAYS

    APPRO loc.foaie:start, 10
    MOVES loc.foaie:start
    BREAK

    FOR t = 0 TO 360 STEP pas.deg
        MOVES loc.foaie:SHIFT(centru BY raza*COS(t), raza*SIN(t), 0)
    END

    MOVES loc.foaie:start
    BREAK
    DEPARTS 10
.END
```

Programul care desenează pe baza unei imagini bitmap încărcată din calculator funcționează astfel:

- Utilizatorul încarcă o imagine alb-negru (BMP, JPG, TIF etc).
- Se apelează programul Potrace (http://potrace.sourceforge.net), care extrage conturul imaginii și îl salvează în formatul SVG.
- Aplicația de pe PC citește fișierul SVG, care conține contururile imaginii în format vectorial. Contururile sunt re-eșantionate folosind segmente de dreaptă de lungimi egale.
- Fiecare contur de pe imagine este reprezentat ca o polilinie. Aceste polilinii sunt reordonate, folosind un algoritm euristic, pentru a reduce mișcarea robotului prin aer.
- Desenul este redimensionat în funcție de mărimea și marginile paginii selectate de utilizator.
- Rezultatul procesării imaginii este un fișier de tip text, care conține o secvență de polilinii, reprezentate prin puncte 2D (coordonate `x` și `y`).
- Structura fișierului text este:

```
n           ; numarul de polilinii din fisier
m1          ; numarul de puncte pt. prima polilinie
x y         ; primul punct
x y         ; al doilea punct
...
m2          ; nr. de puncte pt. a doua polilinie
x y         ; primul punct
...
```

Programul V^+ care citește traiectoriile din fișierul text și desenează imaginea încărcată de utilizator este:

```
.PROGRAM picasso()
    GLOBAL loc.foaie
    CALL vezi.foaia()
    IF DEFINED(loc.foaie) THEN
        CALL ia.pixul()
        CALL deseneaza()
        CALL pune.pixul()
    END
.END
```

```
.PROGRAM deseneaza()
    AUTO lun, p, n, m, zup, x, y, i, j
    GLOBAL loc.foaie, p0
    zup = 5 ; inaltimea la care pixul se deplaseaza in aer

    TYPE "Desenez..."
    TIMER 1 = 0
    SPEED 100 ALWAYS

    ATTACH (lun, 4) "NFS"
    FOPENR (lun) "desen.txt"

    APPROS loc.foaie:p0, 50
    BREAK

    READ (lun) n
    FOR i = 1 TO n
        READ (lun) m
        IF m > 1 THEN
            READ (lun) x, y
            SET p = loc.foaie:SHIFT(p0 BY x,y,0)

            SPEED 70
            APPROS p, zup
            MOVES p
            BREAK
            ; Nu am DURATION => se deseneaza cu viteza max. posibila.
            ; Dist. intre puncte = ds  =>  viteza = (ds / 0.016) mm/s
            FOR j = 2 TO m
               READ (lun) x, y
               SET p = loc.foaie:SHIFT(p0 BY x,y,0)
               MOVES p
            END
            BREAK
            DEPARTS zup
        END
    END

    FCLOSE (lun)
    DETACH (lun)
    TYPE "Desenarea a durat ", INT(TIMER(0)), " secunde."
.END
```

Prelucrare pe contur

Nume de cod: Frunza

Se dă o piesă din plexiglas în formă de frunză, realizată pe mașina CNC pornind de la un model CAD (Fig. 6.4).

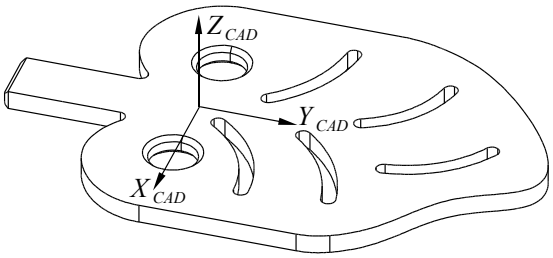

Figura 6.4: *Modelul CAD pentru piesa în formă de frunză*

Se dorește realizarea unei prelucrări (de exemplu șlefuire) pe conturul piesei, folosind o unealtă circulară fixată în spațiul de lucru. Robotul va deplasa piesa astfel încât unealta să parcurgă conturul acesteia (Fig. 6.5).

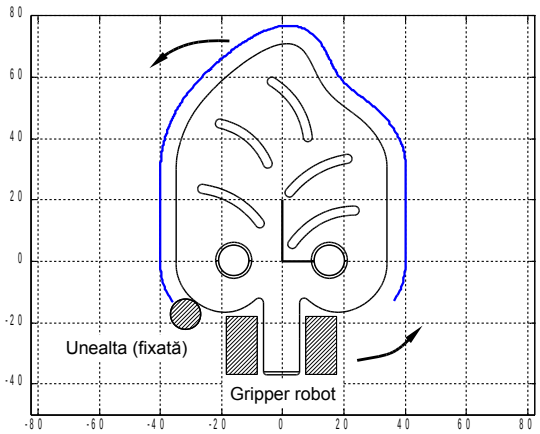

Figura 6.5: *Traiectoria uneltei pe conturul piesei*

În laborator, prelucrarea va fi simulată folosind un ax cu diametrul de 9.5 mm, care înlocuiește unealta. Robotul va deplasa piesa în jurul axului, astfel încât între piesă și ax va fi menținută o distanță de 0.25 mm.

Traiectoria se citește dintr-un fișier text ASCII, care are pe fiecare linie o pereche de coordonate `x y` separate prin spațiu. Punctele citite din fișier sunt memorate în vectorul de transformări `lista.puncte[]`.

Datele din fișier conțin o traiectorie aflată la distanță de 5 mm față de conturul fizic al piesei. Traiectoria a fost obținută prin comanda *Offset* în programul CAD, și este aproximată prin segmente de dreaptă cu lungimea de 1 milimetru fiecare (eșantionată uniform).

Se cunoaște distanța între cele două găuri pe piesă, `d = 31.75 mm`.

Vom alege sistemul de coordonate Tool pe piesă într-o poziție convenabilă, dacă se poate, identic cu sistemul de coordonate în care a fost proiectată piesa, $(XYZ)_{CAD}$. Principalul inconvenient este dat de axa Z_{tool}, care trebuie aleasă în jos (altfel nu ar mai funcționa instrucțiunile `APPRO/S` și `DEPART/S`).

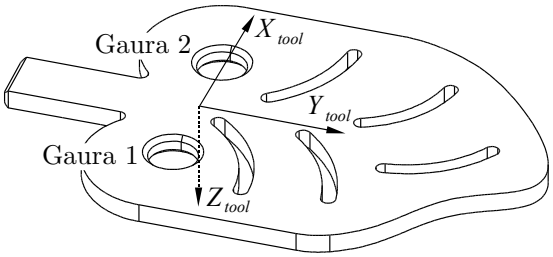

Figura 6.6: *Sistemul de referință Tool ales pentru piesa de tip frunză*

Avem două variante: fie întoarcem piesa, fie inversăm și sensul axei X_{tool}. Alegem varianta a doua și definim `tool.frunza` ca în Fig. 6.6.

Vom învăța punctele `a`, `b` și `c` pe paletă (Fig. 6.7). Punctul `c` ajută la determinarea unghiului `ang` dintre paletă și X_{world}. Dacă paleta este aliniată suficient de bine cu axele robotului, transformarea Tool poate fi învățată și fără acest punct (considerând `ang = 0` sau multiplu de 90 °).

Vom defini de asemenea două transformări Tool pentru cele două găuri: `tool.gaura.1` și `tool.gaura.2`. După ce cunoaștem `tool.frunza` și distanța între găuri `d = 31.75`, celelalte două transformări se calculează prin compunere la dreapta:

```
SET tool.gaura.1 = tool.frunza:TRANS(-31.75 / 2, 0, 0)
SET tool.gaura.2 = tool.frunza:TRANS( 31.75 / 2, 0, 0)
```

Folosind transformarea Tool pentru una din găuri, vom învăța poziția uneltei simulate (axul fixat):

```
.TOOL tool.gaura.1
.HERE loc.ax
```

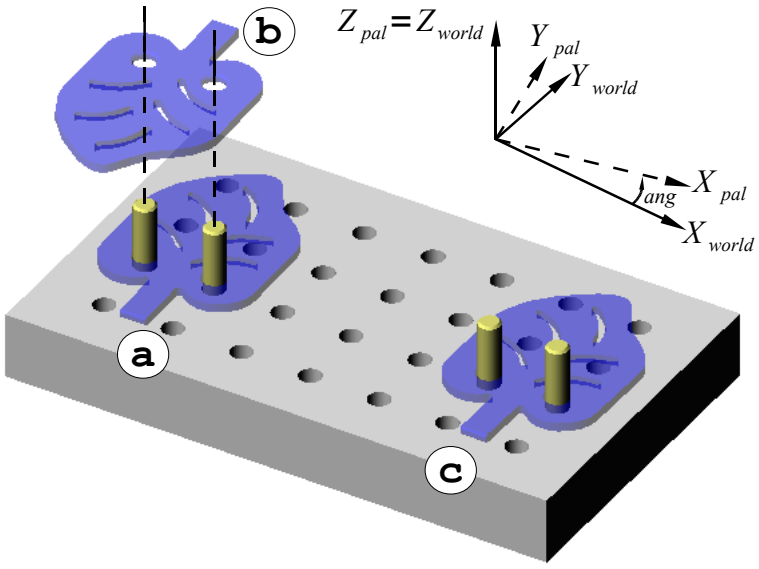

Figura 6.7: *Punctele învățate pentru a calcula transformarea* ***tool.frunza***

Programul care determină cele 3 transformări Tool este:

```
.PROGRAM calc.tools()
    ; intrare: a, b, c - invatate cu HERE, cu TOOL-ul curent
    ; iesire: transf. tool.frunza, tool.gaura1, tool.gaura2
    GLOBAL a, b, c, tool.frunza, tool.gaura.1, tool.gaura.2
    AUTO ang, ref.loc
    ang = ATAN2(DY(c) - DY(a), DX(c) - DX(a))
    SET ref.loc = TRANS((DX(a) + DX(b))/2,
                        (DY(a) + DY(b))/2, DZ(a), 0, 180, -ang)

    SET tool.frunza = TOOL:INVERSE(a):ref.loc
    SET tool.gaura.1 = tool.frunza:TRANS(-31.75 / 2, 0, 0)
    SET tool.gaura.2 = tool.frunza:TRANS( 31.75 / 2, 0, 0)
.END
```

O posibilitate de a implementa urmărirea conturului ar fi următoarea: mutăm punctul condus al robotului, pe rând, în fiecare punct de pe traiectorie, și facem acest punct să coincidă cu `loc.ax`:

```
FOR i = 1 TO n
   x = -DX(lista.puncte[i])
   y =  DY(lista.puncte[i])
   TOOL tool.frunza:TRANS(x, y, 0)
   MOVE loc.ax
END
```

Deoarece V^+ introduce `BREAK` la schimbarea transformării Tool, rezultatul programului anterior va fi o mișcare sacadată. Pentru a obține o mișcare lină, vom folosi formula echivalentă din laboratorul 4, și vom specifica unde trebuie să se afle punctul din centrul piesei, astfel încât axul fix (unealta) să fie în poziția dorită (punctul curent de pe traiectorie).

```
TOOL tool.frunza
FOR i = 1 TO n
   x = -DX(lista.puncte[i])
   y =  DY(lista.puncte[i])
   MOVE loc.ax:INVERSE(TRANS(x, y, 0))
END
```

Următorul program poate fi folosit pentru a testa corectitudinea transformărilor Tool asociate găurilor. Programul efectuează o rotație de 360° în jurul punctului `loc.ax`, în sensul dat de `sgn`. Axa de rotație este Z_{tool} (axa Z a punctului `loc.ax`). Transformarea Tool dorită se setează manual, înainte de a executa programul.

```
.PROGRAM rot(loc.ax, sgn)
    SPEED 100 ALWAYS
    APPRO loc.ax, 50
    SPEED 20
    MOVES loc.ax
    BREAK

    MOVES loc.ax:RZ( 90*sgn)
    MOVES loc.ax:RZ(180*sgn)
    MOVES loc.ax:RZ(270*sgn)
    MOVES loc.ax:RZ(360*sgn) ; sau doar MOVES loc.ax
    BREAK

    SPEED 20
    DEPARTS 50
    BREAK
.END
```

Mișcarea frunzei va începe cu unealta (axul fixat) în punctul `(-40, -20)` și se va termina în punctul `(40, -20)`, pe graficul din Fig. 6.5.

Programul care translatează piesa, menținând unealta fixată tangentă, este listat în continuare. Transformarea `tool.frunza` este selectată automat.

```
.PROGRAM follow.contour(loc.ax)
    GLOBAL lista.puncte[]
    AUTO i, x, y, loc.start, loc.stop

    IF NOT DEFINED(lista.puncte[]) THEN
        TYPE "Traiectoria nu este incarcata, rulati citeste.puncte()"
        RETURN
    END

    TOOL tool.frunza

    ; punctul de start = (-40, -20) pe grafic
    ; punctul de stop  = ( 40, -20) pe grafic
    ; schimb semnul lui x
    SET loc.start = loc.ax:INVERSE(TRANS( 40,-20))
    SET loc.stop  = loc.ax:INVERSE(TRANS(-40,-20))

    SPEED 100 ALWAYS
    APPRO loc.start, 100
    SPEED 20
    MOVES loc.start

    ; Doresc 50 mm/s, iar punctele sunt esantionate la 1mm
    DURATION 1 / 50 ALWAYS

    FOR i = 0 TO LAST(lista.puncte[])
        x = -DX(lista.puncte[i])
        y =  DY(lista.puncte[i])
        MOVE loc.ax:INVERSE(TRANS(x, y))
    END

    MOVE loc.stop

    SPEED 20
    DEPARTS 50
    BREAK
.END
```

Programul care citește traiectoria din fișierul de intrare ASCII și construiește vectorul `lista.puncte[]` este:

```
.PROGRAM citeste.puncte()
    AUTO lun, i, x, y
    GLOBAL lista.puncte[]

    ATTACH (lun, 4) "NFS"
    FOPEN (lun) "frunza.txt"
    IF IOSTAT(lun) < 0 THEN
        TYPE $ERROR(IOSTAT(lun))
        RETURN
    END

    MCS "deletel lista.puncte[]"
    i = 0

    ; nu cunosc numarul de linii => citesc pana la EOF

    WHILE (TRUE) DO
        READ (lun) x, y
        IF IOSTAT(lun) < 0 THEN
            IF IOSTAT(lun) == -504 THEN
                TYPE "End of file, OK."
                EXIT ; din WHILE
             ELSE
                TYPE $ERROR(IOSTAT(lun))
                RETURN
             END
        END
        SET lista.puncte[i] = TRANS(x,y)
        i = i+1
    END
    FCLOSE (lun)
.END
```

Aplicații laborator

Optimizare Pick and Place

Învățați două puncte robot, `a` și `b`, în aer, astfel încât robotul se poate mișca liber între ele. Testați mișcarea robotului între aceste două puncte setând viteza 10 monitor / 100 în program, repectiv 100 monitor / 10 în program.

Învățați punctele `pick`, `place` și `#safe` și testați mișcarea robotului cu rutina `pick.place` din laboratorul 2.

Modificați programul `pick.place` astfel încât vitezele acestuia să corespundă cu vitezele din programul `ppo` din laboratorul curent. Se vor păstra instrucțiunile `BREAK`. Rulați programul "în aer" (fără piesă în gripper), cu viteza monitor setată mai întâi la 30 (pentru siguranță), apoi la 100%. Cronometrați programul folosind un timer.

Pentru a copia un program în vederea editării acestuia, folosiți comanda:

```
.copy prog.dest = prog.sursa
```

Observați vibrațiile robotului în momentul în care acesta se oprește la întâlnirea unui `BREAK` după o mișcare executată cu viteză maximă.

Îndepărtați instrucțiunile `BREAK`, comentați `SPEED` și rulați programul din nou. Comparați timpul de execuție cu cel obținut anterior. Estimați solicitarea din articulațiile robotului, ascultând zgomotul produs de motoare. Față de cazul precedent, este mai bine, mai rău sau nu observați nicio modificare?

Înainte de execuția instrucțiunilor `OPENI` și `CLOSEI`, robotul va face o pauză. Dacă se rulează cu viteză mai mică (de exemplu 50% monitor), pauza nu mai este vizibilă. Explicați fenomenul, apoi încercați să îl eliminați, fie limitând rata de decelerare, fie micșorând viteza pe segmentul de mișcare imediat anterior instrucțiunilor `OPENI` și `CLOSEI`.

Puteți obține un timp de execuție mai bun micșorând viteza/accelerația în punctele critice?

Testați efectul instrucțiunilor `COARSE`, `FINE`, `NULL` și `NONULL`.

După ce programul rulează corect cu viteză ridicată, fără vibrații excesive, puteți testa din nou, de data aceasta cu o piesă în gripper. Atenție la distanța minimă de apropiere și depărtare!

Propuneți și alte metode pentru optimizarea rutinei *Pick and Place*.

Puteți folosi script-ul *MotioLyze* pentru a vedea grafic mișcarea robotului:

- Încărcați programul `motiolyze.v2` pe controller-ul robot;
- Deschideți `MATLAB` și selectați folder-ul care conține rutinele MotioLyze;
- Editați programul `motiolyze.test` astfel încât să apeleze rutina testată (în cazul nostru, Pick and Place);
- Rulați programul `motiolyze` pe controller-ul robot, pe task-ul 1:

  ```
  .exec 1 motiolyze
  ```

- Rulați următoarele comenzi Matlab:

  ```
  >> [T,H,D,S] = getmotion;
  >> plotmotion_cobras600(T,H,D,S);
  ```

- Dacă apar probleme, opriți forțat (cu `.abort`) task-urile 0, 1 și 2.

Traiectorii descrise matematic

Rulați programul `cerc` cu viteza monitor 50. Specificați o viteză în milimetri pe secundă (de exemplu 50 mm/s) în cadrul instrucțiunii `DURATION`, în subrutina `deseneaza.cerc`. Cronometrați mișcarea folosind un timer și verificați dacă robotul a respectat viteza impusă.

Rulați același program cu viteza 100 monitor și cronometrați.

Încercați să renunțați la `DURATION` și să folosiți `SPEED 50 MMPS ALWAYS`. Cronometrați.

Micșorați pasul de discretizare. Dacă se micșorează sub o anumită limită, viteza de desenare va scădea. Găsiți valoarea minimă a pasului pentru care viteza de desenare este cea dorită. Ce valoare are argumentul lui `DURATION` pentru valoarea găsită?

Folosiți `MOVES` în loc de `MOVE` în bucla interioară și repetați experimentul.

Rulați programul cu valori foarte mari ale pasului de discretizare, de ex. 90 °.

Exercițiu

Modificați programul `cerc` astfel încât să deseneze o altă figură descrisă parametric, de exemplu una din următoarele:

- curbe Rose - http://en.wikipedia.org/wiki/Rose_curve
- figuri Lissajous - http://en.wikipedia.org/wiki/Lissajous_curve

Frunza

Dacă se dorește rularea aplicației pe plan înclinat, se va folosi un robot articulat vertical; altfel, este suficient un robot SCARA.

Încărcați modulul `frunza` de pe disc. Rulați programul `citeste.puncte()` pentru a construi vectorul `lista.puncte[]`.

Puteți vedea traiectoria din fișierului `frunza.txt` (Fig. 6.5) în Matlab:

```
>> D = dlmread('frunza.txt'); x = D(:,1); y = D(:,2);
>> plot(x, y); axis equal; grid on;
```

Învățați transformarea Tool pentru piesa în formă de frunză. Pentru aceasta utilizați programul `calc.tool`. Activați transformarea Tool pentru una din găuri (de exemplu `tool.gaura.1`), verificați corectitudinea ei rotind piesa pe modul *RZ* din MCP și învățați `loc.ax` folosind gaura aleasă.

Verificați transformarea Tool pentru cealaltă gaură (`tool.gaura.2`) rulând programul `rot`. Alegeți sensul de rotație în funcție de semnul articulației terminale a robotului în punctul `loc.ax` (MCP: Disp → Joint Values). Dacă semnul este pozitiv, rotiți în sens negativ și invers. Ultima articulație se poate roti între $-360\,^\circ$ și $+360\,^\circ$.

Activați transformarea `tool.frunza`, observați originea și direcțiile axelor folosind MCP-ul pe modul *Tool*, apoi rulați programul `follow.contour`.

Învățați `loc.ax` folosind un ax înclinat aflat pe suportul cu baza magnetică, apoi rulați programul din nou.

Exercițiu

Modificați `follow.contour()` astfel încât, pe lângă translație, piesa să se și rotească (în jurul axei Z_{tool}). Unealta simulată va rămâne tangentă la piesă.

Exercițiu

Rezolvați problema *Frunza* fără a folosi transformarea Tool. Axele piesei vor rămâne aliniate cu axele World pe tot parcursul mișcării.

- Stabiliți câte puncte robot vor fi învățate și care sunt acestea;
- Modificați programul `follow.contour()`.

Indicație: puteți înlocui compunerea transformărilor cu instrucțiunea `SHIFT` ⇒ nu mai este necesară schimbarea semnului pe axa X.

Whew! ☺