Lycée CHEVROLLIER CRSA Divers sont les talents

MAI-PRE-0029-G
O Marchais
Le 14/08/2022

PROGRAMMATION D'UN ROBOT INDUSTRIEL (INITIATION)

Table des matières

1 Description des différents robots

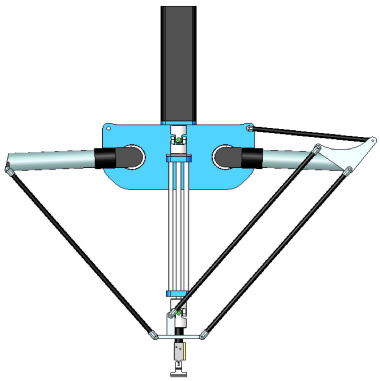
1.1 Définition

Un robot est un manipulateur, commandé en position, polyvalent et programmable.

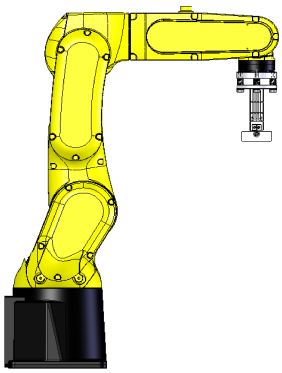
1.2 Constitution

Il est constitué d'un outil : torche de soudage, pince, ventouse... déplacé par une structure articulée : chaîne de corps rigides assemblés par des articulations.

Il existe des chaînes fermées (robots delta) ou ouvertes (RS40, LRMate).



CHAINE FERMEE



CHAINE OUVERTE

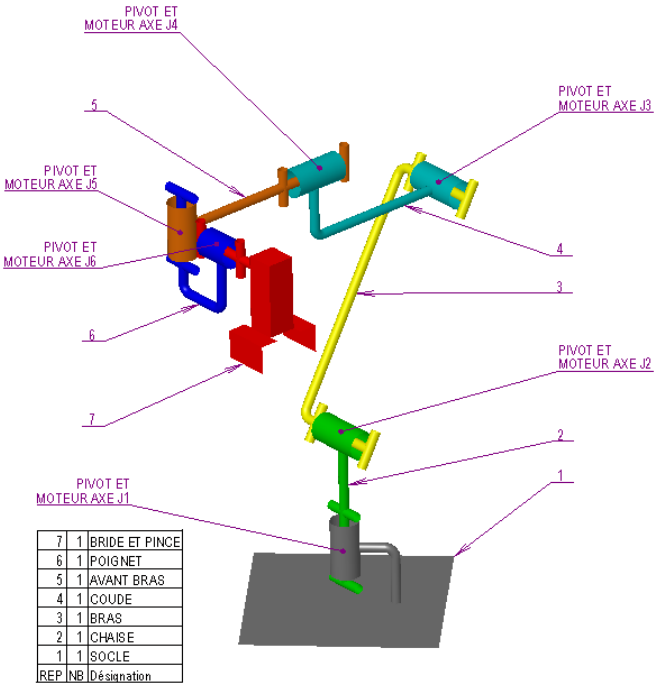
1.3 Architecture d'une chaîne ouverte

Elle est composée de **n+1** corps rigides (socle, bras, avant bras ...) liés par **n** articulations à 1 degré de liberté qui peuvent être des pivots (P) ou des glissières (G).

La position et l'orientation de l'outil est alors définie par **n** paramètres qui peuvent être des angles ou des distances.

On parle de robot à **n** axes dont la configuration est définie par **n** coordonnées articulaires (joint en anglais) : J1, J2, J3, ..., Jn

Ci-contre : exemple d'un robot 6 axes (anthropomorphe)



REP	NB	Désignation
7	1	BRIDE ET PINCE
6	1	POIGNET
5	1	AVANT BRAS
4	1	COUDE
3	1	BRAS
2	1	CHAISE
1	1	SOCLE

1.4 Position et orientation de l'outil

La situation de l'outil dans l'espace est définie par 6 coordonnées cartésiennes (frame en anglais) :

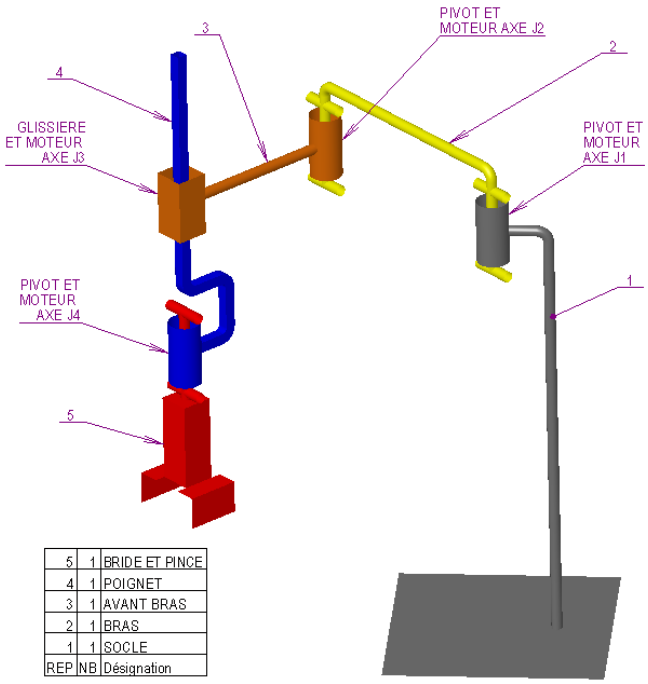
sa position X,Y,Z
et son orientation Rx,Ry,Rz

En fonction de la tâche à accomplir par le robot on peut se contenter de **n** < 6.

Pour la prise et la pose d'objets par exemple (Pick & Place en anglais) ou pour la palettisation on peut se contenter de **n** = 4.

Ci-contre : exemple d'un robot 4 axes (Scara)

Dans la chaîne ouverte, on distingue le porteur : les axes J1 à J3 qui définissent la position de l'outil et le poignet : les axes suivants qui définissent son orientation.



REP	NB	Désignation
5	1	BRIDE ET PINCE
4	1	POIGNET
3	1	AVANT BRAS
2	1	BRAS
1	1	SOCLE

1.5 Différents types de porteurs industriels

Les porteurs les plus courants sont (voir vidéos) :

Scara (P+P+G)
<= Exemple : RS40 (Staubli)

Anthropomorphes (P+P+P)
Exemples : LRMate (Fanuc) =>

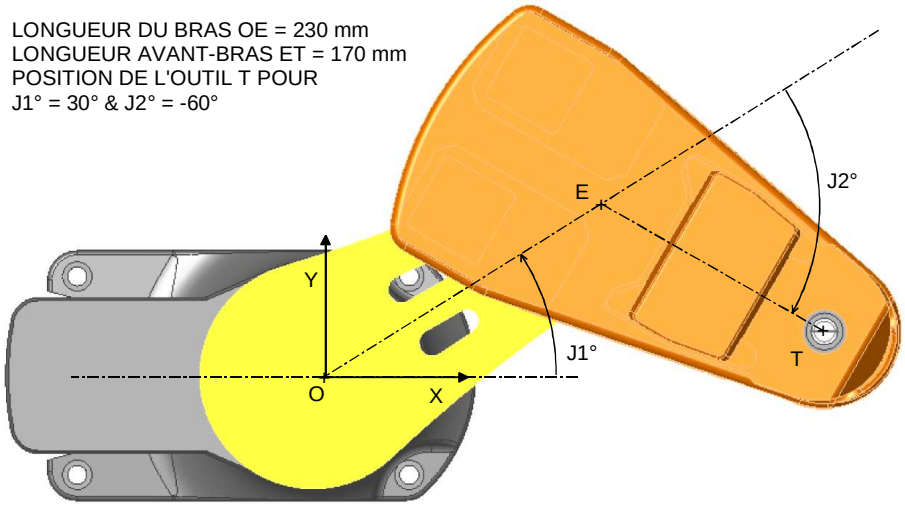
Cartésiens (G+G+G)
<= Exemple : S7 (Sepro)

Cylindriques (P+G+G)
Exemple : S3 (Sepro) =>

2 Positions et mouvements, exemple du RS40 (2D)

2.1 Objectif

Faire le lien entre les angles J1° & J2° des axes J1 et J2 du robot et la position (X,Y) de l'outil T (pour tool) dans le repère du robot (O,x,y).



2.2 Modèle géométrique direct

Les coordonnées de l'outil (X,Y) se déduisent des angles J1° et J2° par des relations trigonométriques simples :

X = 230 cos (J1°) + 170 cos (J1°+J2°) = 346,4 mm

Y = 230 sin (J1°) + 170 sin (J1°+J2°) = 30 mm

Cette transformation est simple à réaliser pour tous les porteurs. Elles est même immédiate pour les porteurs cartésiens.

Dans le cas du robot Scara, la coordonnées Z de l'outil est donnée directement par l'axe J3 (position de la glissière entre le poignet et l'avant bras).

Z = J3

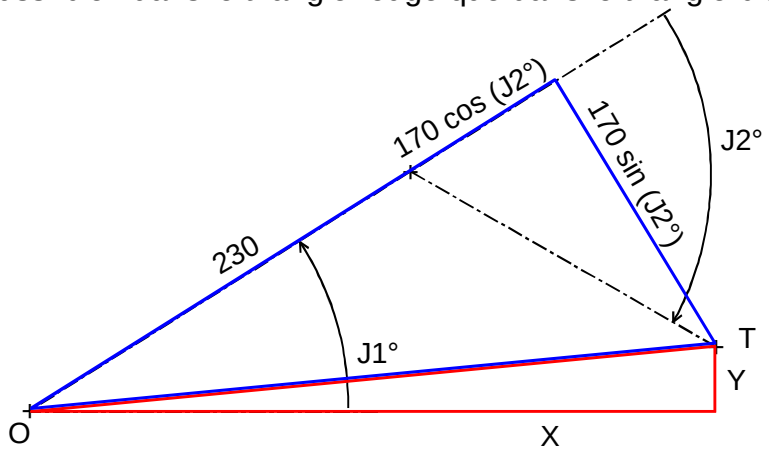
2.3 Modèle géométrique inverse

Trouver les angles J1° et J2° permettant d'obtenir les coordonnées (X,Y) de l'outil n'est pas aussi simple.

Pour trouver J1° et J2° qui positionnent l'outil du robot RS40 au coordonnées (350,30), une méthode consiste à résoudre d'abord l'équation :

$$X^2 + Y^2 = (230 + 170\cos(J2°))^2 + (170\sin(J2°))^2$$

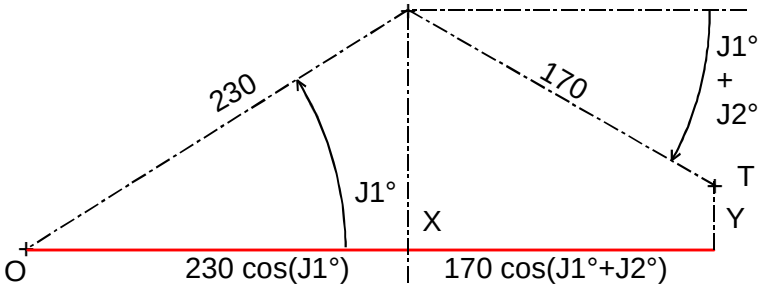
Cette équation traduit le fait que le carré de l'hypoténuse OT s'obtient par le théorème de Pythagore, aussi bien dans le triangle rouge que dans le triangle bleu.



Grâce au solveur de la calculette, on obtient deux solutions : J2° = 57,9° ou -57,9°

Connaissant J2°, il faut résoudre l'équation pour trouver J1° :

$$X = 230\cos(J1°) + 170\cos(J1°+J2°)$$



Grâce au solveur de la calculette, on obtient deux solutions :

J1° = -19,3° pour J2° = 57,9°

et J1° = 29,1° pour J2° = -57,9°

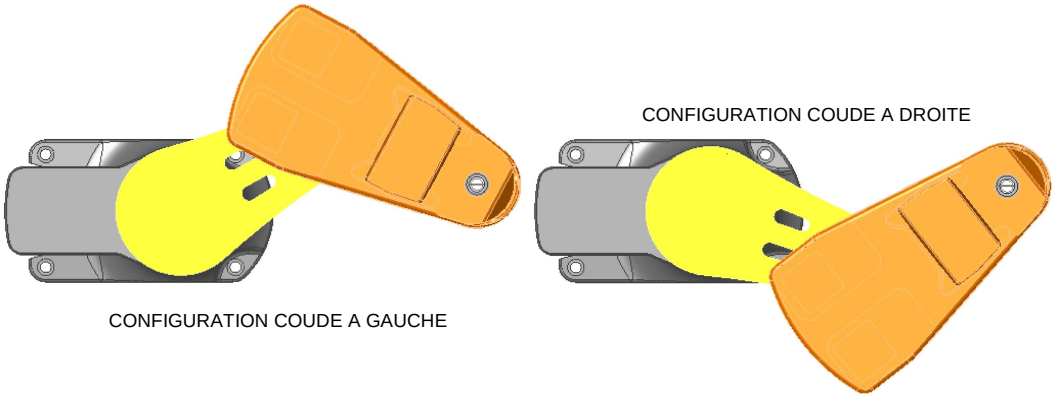
Ce calcul manuel, compliqué pour les deux angles J1° et J2°, devient impossible pour les 6 angles J1° à J6° d'un robot 6 axes. C'est le rôle du contrôleur du robot, de calculer en temps réel les angles à imposer à chaque articulation pour obtenir les coordonnées voulues de l'outil.

2.4 Deux définitions d'un point

Le calcul précédent montre que l'on peut utiliser deux définitions pour une position : la définition articulaire (joint = articulation en anglais) ou la définition cartésienne (frame = bâti en anglais), mais qu'elles ne sont pas équivalentes.

La définition articulaire d'un point, par les paramètres J1 à Jn est unique. Elle impose la posture du bras du robot.

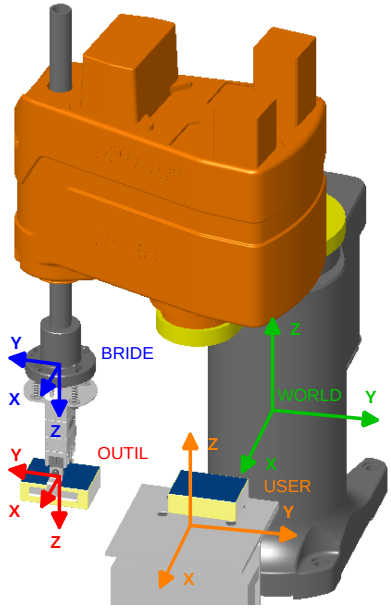
La définition cartésienne d'un point est multiple. Dans le cas précédent, deux couples d'angles J1° et J2° conduisent l'outil aux coordonnées (350,30). Il faut alors préciser si l'on veut atteindre le point avec une configuration « coude à gauche» ou avec une configuration « coude à droite».



Pour être sûr de la position du robot lors de l'initialisation d'une cellule robotisée, on doit toujours définir la position initiale du bras en coordonnées articulaires.

2.5 Modificateurs des coordonnées cartésiennes

Pour aider l'utilisateur à définir la position cartésienne de l'outil, deux modifications sont calculées automatiquement par le contrôleur du robot :

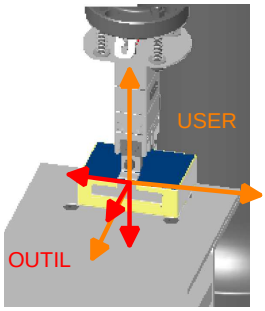


La prise ne compte du repère outil : les coordonnées imposées sont celles de l'extrémité de l'outil et plus celles de la bride (bout du bras).

La prise en compte du repère utilisateur : les coordonnées sont exprimées dans le repère utilisateur (user) plutôt que dans le repère robot (world).

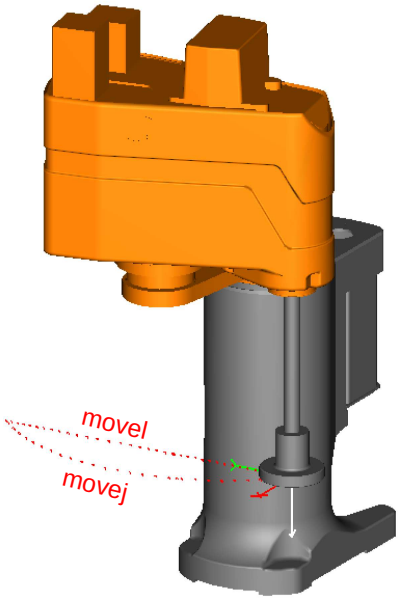
Ce repère utilisateur peut même être en mouvement dans le cas du tracking.

Dans la position « naturelle » du robot, le repère outil et le repère utilisateur sont tête-bêche. Le point origine du repère utilisateur a donc pour coordonnées 0,0,0 pour la position, mais 180°,0°,0° pour l'orientation.



2.6 Définition des mouvements

2.6.1 Deux types de mouvements



Les robots peuvent se déplacer d'un point à un autre avec ou sans contrôle de trajectoire.

Le premier mode (movel, movec) permet d'imposer une trajectoire linéaire ou circulaire mais peut déboucher sur des blocages (position inaccessible en milieu de trajectoire).

Le deuxième mode (movej) laisse le robot libre d'emprunter le chemin qu'il veut et de changer de configuration en cours de route. Il garantit l'accès à tout les points du domaine accessible.

En l'absence de contrainte (mouvement d'approche précis), il faut laisser le robot aussi libre que possible de ses mouvements. Il faut donc utiliser des mouvements sans contrôle de trajectoire et ne pas imposer la configuration du bras (coude libre).

2.6.2 Mouvement d'approche & lissage de trajectoire

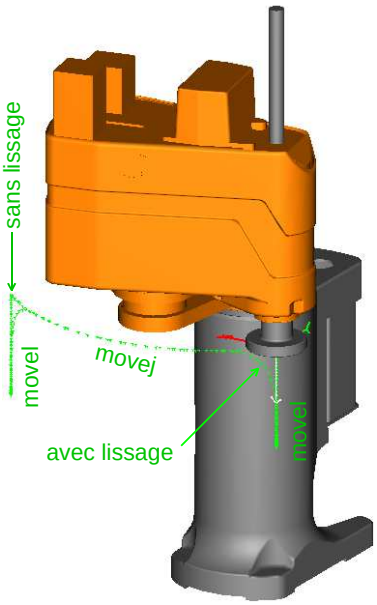
Le mouvement vers la cible demande la plupart du temps de définir des points intermédiaires (contournement d'obstacles, approche de la cible suivant une direction particulière).

Un mouvement de prise et dépose d'objet se construit généralement de la manière suivante :

Se déplacer au dessus du point de prise sans contrôle
Descendre en ligne droite vers l'objet
Saisir l'objet
Remonter en ligne droite pour dégager l'objet

Se déplacer au dessus du point de pose sans contrôle
Descendre en ligne droite vers l'emplacement
Lâcher l'objet
Remonter en ligne droite pour dégager l'outil

Le fractionnement de la trajectoire entraîne alors des arrêts à chaque changement de direction. Pour supprimer les à-coups et gagner en vitesse, une fonction de lissage des trajectoire est proposée par tous les langages de programmation.



3 Programmation du robot STAUBLI RS40 (langage VAL3)

3.1 Programme élémentaire

Voici l'application « MINIMUM » de prise et dépose d'objet composée des deux programmes : START qui se lance au démarrage de l'application et STOP qui est exécuté avant la fermeture de l'application. Les signes // désignent les commentaires.

Les mouvements effectués par le robot sont décrits dans la vidéo du même nom.

La partie INITIALISATION définit :
- l'outil, de longueur 60 mm, câblé sur la sortie de mnémonique dOutil et s'ouvrant et se fermant avec un délai de 0,1 s.
- la loi de vitesse des mouvements imposant une accélération et un freinage de 50 %, une vitesse de 75 % et un rayon de lissage de 100 mm
- les coordonnées des repères utilisateur de prise et de pose dans le repère world et les transformations permettant de passer de l'un à l'autre.
- Le point origine utilisable dans ces deux repères et qui laisse le coude libre (pour 180 voir § 2.5).

La partie DIALOGUE efface l'écran utilisateur avant d'y afficher le nom et la version de l'application.

La partie MOUVEMENTS :
- déclenche tout d'abord un mouvement de remontée à l'altitude de 199 mm à la verticale de la position courante (point défini en articulaire),
- puis calcule le point cible comme origine du repère de prise, et le point d'approche situé à la verticale du point cible à 139 mm d'altitude (points définis en cartésien),
- déplace enfin le bras vers le point d'approche sans contrôle de trajectoire puis vers le point cible en ligne droite. Ces déplacement se font à la vitesse et avec l'outil définis plus haut.

```
START :
begin
  // INITIALISATION
    // Définition de l'outil
      tOutil={{0,0,60,0,0,0}
      ,dOutil,0.1,0.1}
      open(tOutil)
    // Définition de la loi de vitesse
      mVitesse={50,75,50,99999,
      99999,joint,100,100}
    // Définition des repères utilisateur
      fPrise={{200,-250,-10,0,0,0}}
      trPrise=position(fPrise,world)
      fPose={{200,150,-10,0,0,0}}
      trPose=position(fPose,world)
    // Points origine des repères
      pOrigine={{0,0,0,180,0,0},{sfree}}
  // DIALOGUE
    // Affichage à l'écran
      userPage()
      cls()
      put("PROGRAMME MINI PRISE-POSE
      190218")
  // MOUVEMENTS
  // Dégagement de sécurité
    // Mesure du point courant
    jDegage=herej()
    // Modification de l'altitude
    jDegage.j3=199
    // Déplacement de dégagement
    movej(jDegage,tOutil,mVitesse)
  // Déplacement vers le point de prise
    // Calcul du point cible
   pCible=compose(pOrigine,world,trPrise)
    // Calcul du point d'approche
    pApproche=pCible
    pApproche.trsf.z=139
    // Déplacement au point d'approche
    movej(pApproche,tOutil,mVitesse)
    // Déplacement au point cible
    movel(pCible,tOutil,mVitesse)
```

La partie MOUVEMENTS continue :
- par la fermeture de l'outil (pince) et la remontée verticale au point d'approche.

La même succession de mouvements est répétée pour le déplacement vers le point de pose avec deux différences :
- le repère de pose remplace le repère de prise,
- l'ouverture de la pince remplace sa fermeture.

```
    // Fermeture pince
    close(tOutil)
    // Remontée
    movel(pApproche,tOutil,mVitesse)
  // Déplacement vers le point de pose
    // Calcul des coordonnées
    pCible=compose(pOrigine,world,trPose)
    // Calcul du point d'approche
    pApproche=pCible
    pApproche.trsf.z=139
    // Déplacement au point d'approche
    movej(pApproche,tOutil,mVitesse)
    // Déplacement vers le point cible
    movel(pCible,tOutil,mVitesse)
    // Ouverture pince
    open(tOutil)
    // Remontée
    movel(pApproche,tOutil,mVitesse)
    waitEndMove()
end
```

3.2 Avance du programme sur le mouvement

Vous aurez remarqué, que le programme précédent utilise le point pCible, une fois pour la prise et une autre fois pour la dépose. Cette réutilisation de la même variable pour définir deux points différents doit être effectuée avec précaution.

En effet, le programme va plus vite que le robot, et au moment de lancer le mouvement, la définition du point n'est peut-être plus celle que l'on croit. Exemple :

```
pCible=compose(pOrigine,world,trPrise)
movel(pCible,tOutil,mVitesse)
pCible=compose(pOrigine,world,trPose)
movel(pCible,tOutil,mVitesse)
pCible=compose(pOrigine,world,trPrise)
movel(pCible,tOutil,mVitesse)
```

Ce programme n'engendre qu'un seul mouvement vers le point de prise, car au moment où le robot démarre le mouvement de la ligne 4, pCible a déjà la valeur imposée à la ligne 5. Le robot étant déjà en ce point, il ne bouge pas.

```
pCible=compose(pOrigine,world,trPrise)
movel(pCible,tOutil,mVitesse)
waitEndMove()
pCible=compose(pOrigine,world,trPose)
movel(pCible,tOutil,mVitesse)
waitEndMove()
pCible=compose(pOrigine,world,trPrise)
movel(pCible,tOutil,mVitesse)
waitEndMove()
```

Pour passer par les 3 points il faut ajouter la commande waitEndMove() qui impose au programme d'attendre la fin des mouvements du robot pour continuer.
Les commandes open et close ont le même effet.

C'est pourquoi une trajectoire est construite de la manière suivante : calcul de tous les points nécessaires en utilisant des variables différentes, puis exécution des mouvements.

3.3 Structures de base et sous-programmes

Voici l'application « INTERMED » de prise et dépose d'objet composée des programmes START, STOP et d'un certain nombre de sous-programmes. Appelés par le programme START, ces sous-programmes permettent d'alléger son écriture.

Les mouvements effectués par le robot sont décrits dans la vidéo « INTERMED ».On remarquera que l'opérateur peut effectuer plusieurs cycles sans avoir à relancer le programme et que le robot se met tout seul sous tension.

La partie INITIALISATION du programme START est remplacée par un appel au programme initialisations qui reprend toutes les définitions précédentes et où la commande enablepower() met le bras sous tension (voir annexe).

Le mouvement de remontée du bras est reporté dans le sous programme remonte. Il est paramétrable grâce au paramètre d'altitude.

Les mouvements de prise et de pose sont définis dans les sous programmes mvt_prise et mvt_pose qui utilisent aussi le sous programme remonte (voir annexe).

La structure while – endWhile est une boucle qui s'exécute tant que la variable !bQuitter est vraie. On en sort quand bQuitter est vraie (! définit l'opposé).

La structure switch – endSwitch exécute une action en fonction de la valeur de nTouche (n°de la touche activée par l'opérateur).

La structure if – endIf exécute les commandes intercalées si bCycle est vraie. Chaque appui de l'opérateur déclenche un seul cycle car on prend soin de remettre bCycle fausse.

```
START :
begin
  // INITIALISATION
    call initialisations()
    // Dégagement de sécurité
    call remonte(199)
  // DIALOGUE
    // Affichage à l'écran
    userPage()
    cls()
    putln("PROGRAMME INTER PRISE-POSE
      200511")
    putln("Appuyer sur F1 pour nouveau
      cycle")
    put("Appuyer sur F8 pour quitter")
  // MOUVEMENTS
    // Boucle de fonctionnement
    while !bQuitter
      // Saisie des touches
      nTouche=getKey()
      switch nTouche
        case 271 // Touche F1
          bCycle=true
        break
        case 278 // Touche F8
          bQuitter=true
        break
      endSwitch
      // Cycle de prise et pose
      if bCycle
        // Mouvement de prise
        call mvt_prise()
        // Mouvement de pose
        call mvt_pose()
        bCycle=false
      endIf
    endWhile
end
```

Il existe d'autres structures, notamment la boucle for – endFor qui permet d'exécuter une boucle un nombre donné de fois.

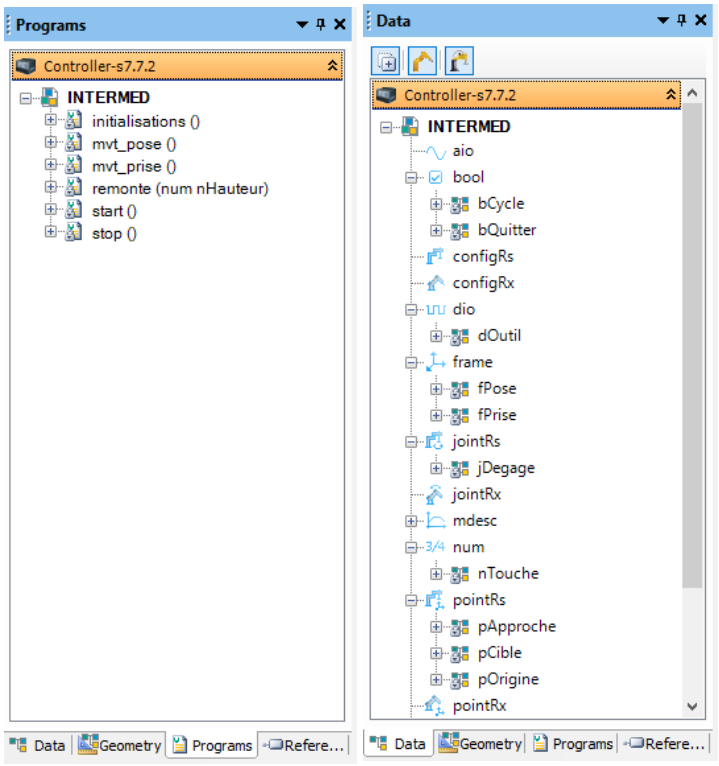
Ces structures sont souvent imbriquées les unes dans les autres.

3.4 Déclaration des variables

Toutes les variables doivent être déclarées au préalable. Elles peuvent être initialisées lors de cette déclaration, mais leur initialisation par le sous programme du même nom permet de les retrouver plus facilement et d'ajouter tous les commentaires nécessaires au programmeur.

On distingue les variables simples (booléens, réels, chaînes de caractères) et les variables complexes (points, lois de vitesse, repères).

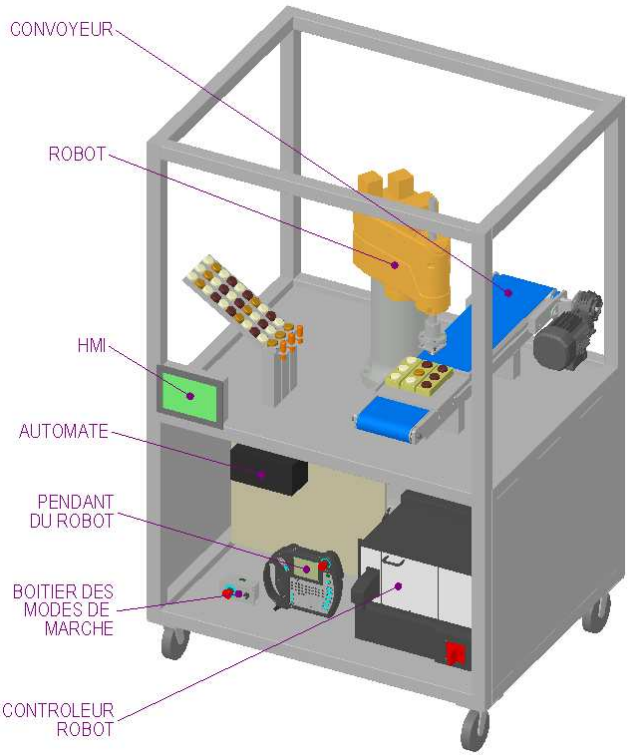
On peut même constituer des tableaux de variables : pPoint[0],pPoint[1],...,pPoint[10].



3.5 Décomposition en tâches robot

Le robot ne fonctionne pas seul. Il est intégré dans une machine automatisée plus vaste, au même titre qu'un convoyeur. On parle de cellule robotisée. Le séquencement des tâches à effectuer lui est imposé par l'automate programmable de celle-ci, qui seul a une vue d'ensemble du processus.

Cependant, c'est un actionneur intelligent qui doit disposer d'une certaine autonomie. L'automate va donc lui donner des ordres complexes, exemple : « va chercher l'objet, ramène-le et attend », « dépose l'objet retourne en position et attend ». On parle de tâche robot. C'est une suite d'actions que le robot peut gérer sans l'aide de l'automate.



Le choix du découpage des mouvements du robot en tâches, conditionne la rapidité et la répartition des capteurs.

L'outil et le capteur anti-collision sont généralement gérés directement par le robot, les autres capteurs et actionneurs étant gérés par l'automate.

3.6 Modes de fonctionnement

Le robot peut fonctionner selon trois modes de marche différents :

En mode manuel, l'opérateur peut faire bouger le bras en l'absence de tout programme, à l'aide du pendant du robot (console de dialogue). Ce mode est réservé aux réglages et aux dépannages.
En mode local, le robot suit son application sans communiquer avec l'extérieur, le pendant du robot sert au dialogue avec l'opérateur. Ce mode est réservé à la mise au point du programme.
En mode déporté, le robot est en communication constante avec l'automate. Ce dernier lui envoi l'ordre d'exécution d'une tâche et attend un compte rendu en retour. C'est le mode normal de fonctionnement en production.

La sélection du mode de marche se fait à l'aide du boîtier des modes de marche.

3.7 Programmes parallèles communicants

Pour permettre ces différents modes de fonctionnement, l'application robot est constituée de plusieurs programmes fonctionnant de manière simultanée (en parallèle) :

un programme qui gère les mouvements,
un programme qui gère la communication avec l'automate en mode déporté,
un programme qui gère le dialogue avec l'opérateur en mode local,
un programme qui gère la securite (collision par exemple).

Ces quatre programmes communiquent entre eux par des variables communes.

Voici l'application «PARALLELE» de prise et dépose d'objet composée du programme START, qui lance en parallèle 4 programmes qui eux-mêmes peuvent faire appel à des sous-programmes.

Les mouvements effectués par le robot sont décrits dans la vidéo «PARALLELE». On remarquera qu'en mode local (LOC) l'opérateur peut déclencher chaque tâche à l'aide du pendant. En mode déporté (API), c'est l'automate qui effectue ce dialogue.

Le programme START se contente d'appeler le programme d'initialisation puis de lancer les 4 programmes en parallèles ensuite, il s'arrête.
Ces 4 programmes tournent en continu car il possèdent tous une boucle
while !bQuitter – endWhile
La variable bQuitter peut être rendue vraie par le programme de dialogue pour mettre fin aux 4 programmes simultanément.
	START :
	begin
	  // INITIALISATION
	  call initialisations()
	  // LANCEMENT DES PROGRAMMES
	  taskCreate "Communication",100,communication()
	  taskCreate "Mouvement",100,mouvement()
	  taskCreate "Dialogue",100,dialogue()
	  taskCreate "Securité",100,securite()
	end

3.7.1 Programme mouvement

Le programme mouvement gère 4 tâches robot :
- tâche n°1 : mise en position initiale,
- tâche n°2 : mouvement de prise,
- tâche n°3 : mouvement de pose,
- tâche n°9 : mise sous tension du bras ou mise hors tension.

Pour lancer la tâche X, l'opérateur (en local) ou l'automate (en déporté) met à X la variable nTache.
Le robot envoi le compte rendu X1 pour annoncer que la tâche X est en cours.
Il envoi le compte rendu X9 lorsque la tâche est terminée.

Ceci n'est qu'une convention adoptée par l'équipe du lycée Chevrollier. D'autres protocoles de dialogue sont possibles.

La variable nRun repasse aussitôt à zéro, empêchant le lancement de plusieurs tâches X successives (voir le programme dialogue).

La commande delay(0) indique au contrôleur qu'il peut passer au programme suivant (dialogue). En effet, le contrôleur est mono-processeur et l'exécution des 4 programmes ne se fait pas réellement en parallèle. Leur simultanéité est apparente et résulte de l'extrême rapidité d'exécution de chaque boucle while !bQuitter – endWhile.

```
MOUVEMENT :
begin
  // GESTION DES MOUVEMENTS
  // Boucle de fonctionnement
  while !bQuitter
    if !isPowered()
      // Mise sous tension du bras
      if nTache==9 and nRun==1
        nCptrendu=91
        enablePower()
        nCptrendu=99
      endIf
    else
      // Mise en position initiale
      if nTache==1 and nRun==1
        nCptrendu=11
        call mvt_init()
        nCptrendu=19
      endIf
      // Prise
      if nTache==2 and nRun==1
        nCptrendu=21
        call mvt_prise()
        nCptrendu=29
      endIf
      // Pose
      if nTache==3 and nRun==1
        nCptrendu=31
        call mvt_pose()
        nCptrendu=39
      endIf
      // Mise hors tension du bras
      if nTache==9 and nRun==1
        nCptrendu=91
        disablePower()
        nCptrendu=99
      endIf
    endIf
    // Amélioration du séquencement
    delay(0)
  endWhile
end
```

3.7.2 Programme dialogue

Le programme dialogue gère l'interaction avec l'opérateur en mode local.

Ce mode n'est utile que lors de la phase de mise au point de la cellule robotisée. Il permet le développement simultané du robot et du reste de la machine. Il aide à concevoir et à mettre au point les différentes tâches robot.

Ce programme comporte une phase de mise en place de l'affichage : n°de tâche et de compte rendu. Cet affichage est rafraîchi à chaque cycle d'exécution de la boucle.

Il comporte ensuite une phase d'affichage des mnémoniques des différents boutons accessibles à l'opérateur.

Il comporte enfin une phase de lecture des touches activées par l'opérateur et qui mettent à jour la valeur des variables.

Parmi ces variables on notera bQuitter qui permet de mettre fin à l'application, nTache qui défini le n° de tâche à effectuer et bModbus qui permet de basculer du mode local au mode déporté et inversement.

En mode local, c'est le programme dialogue qui va imposer la valeur de nTache. En mode déporté, c'est le programme communication.

On remarquera les deux sous-programmes boutons qui sert à afficher un bouton et saisie qui permet la saisie d'un nombre par l'opérateur. Ces sous-programmes sont détaillés en annexe.

```
DIALOGUE :
begin
  // GESTION DU DIALOGUE OPERATEUR
  title(sTitre)
  userPage()
  // Boucle de fonctionnement
  while !bQuitter
    // Composition de l'écran
    cls()
    gotoxy(0,0)
    putln(sSepar)
    put("Compte rendu : ")
    putln(nCptrendu)
    putln(sSepar)
    putln("T1:MPI T2:Prise T3:Pose
      T9:PWR")
    putln(sSepar)
    put("Tache en cours : ")
    putln(nTache)
    putln(sSepar)
    // Affichage des boutons
    call boutons(1,"QUIT")
    if !bModbus
      call boutons(3,"#TA?")
      call boutons(8,"LOC")
    else
      call boutons(8,"API")
    endIf
    // Scrutation des touches
    nTouche=getKey()
    switch nTouche
      case 271
        bQuitter=true
      break
      case 273
        call saisie(17,5)
        nTache=nSaisie
        nRun=1
        delay(0)
        nRun=0
      break
      case 278
        bModbus=!bModbus
      break
    endSwitch
    // Amelioration du séquencement
    delay(0)
  endWhile
end
```

3.7.3 Programme communication

Le programme communication gère l'interaction avec l'automate programmable en mode déporté. Il remplace alors le programme dialogue.

Ce mode est celui qui est utilisé lorsque la machine terminée est en phase de production.

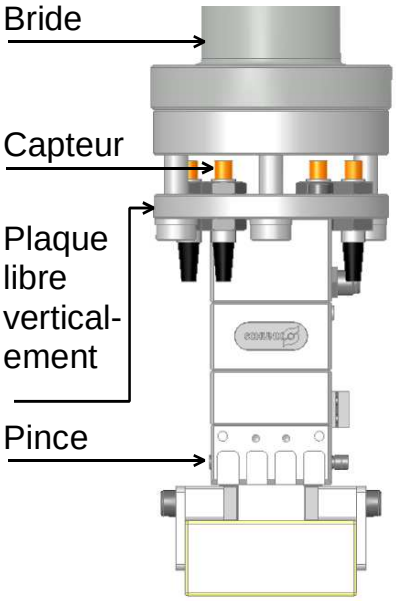
Ce programme effectue à chaque cycle d'exécution, une lecture des variables réseau (réseau modbus dans notre exemple) aTache et aRun qui sont aussitôt affectées à nTache et nRun et une écriture de la variable nCptrendu sur la variable réseau aCptrendu.

```
COMMUNICATION :
begin
  // GESTION DU DIALOGUE AVEC L'API
  //boucle de fonctionnement
  while !bQuitter
    if bModbus
      // Entrées : n° tâche
      nTache=aioGet(aTache)
      nRun=aioGet(aRun)
      // Sorties : Compte rendu
      aioSet(aCptrendu,nCptrendu)
    endIf
    // Amelioration du séquencement
    delay(0)
  endWhile
end
```

3.8 Capteur anti-collision

Un capteur (inductif, photo électrique, pneumatique) est intercalé entre la bride et la pince et déclenche l'arrêt du mouvement en cas de collision.

Le dégagement se fait de manière automatique et le cycle est interrompu. Pour continuer le cycle, l'opérateur doit acquitter le défaut, soit sur le pendant en mode local, soit sur le HMI de la machine en mode déporté. C'est le rôle du programme securite de définir la stratégie à adopter en cas de collision.



3.9 Programme sécurité

Le capteur de collision est câblé sur la sortie de mnémonique dColli.

Dès que la collision a lieu, le programme entre dans la boucle if dColli – endIf.

La variable bAutom a un rôle d'auto-maintien. En effet, le programme ne doit pas sortir de cette boucle dès que le capteur de collision est relâché mais seulement lorsque l'opérateur aura acquitté le défaut.

La stratégie est alors la suivante :
- arrêter le programme mouvement et annuler tous les mouvements en cours,
- arrêter le programme dialogue et avertir l'opérateur et l'automate du défaut,
- effectuer le mouvement de dégagement,
- attendre l'acquittement du défaut par l'opérateur (mode local) ou par l'automate
- en cas d'acquittement, relancer les deux programmes dialogue et mouvement.

```
SECURITE :
begin
  // TRAITEMENT DES ANOMALIES
  bAutom=false
  // Boucle de fonctionnement
  while !bQuitter
    // Conditions anormales
    if ((dColli==true) or bAutom)
      bAutom=true
      // Interruption dialogue & mvt
      taskKill("Mouvement")
      resetMotion()
      waitEndMove()
      taskKill("Dialogue")
      // Avertissement de l'opérateur
      if (dColli==true)
        nCptrendu=998
        gotoxy(15,1)
        put(nCptrendu)
        put(" : Collision !")
        call remonte(nHplafond)
        waitEndMove()
      else
        // Acquittement du défaut
        gotoxy(18,11)
        put("Acquittez ou Quittez !")
        call boutons(1,"QUIT")
        call boutons(8,"ACQU")
        nTouche=getKey()
        switch nTouche
          case 271
            bQuitter=true
          break
          case 278
            bAcquit=true
          break
        endSwitch
      endIf
      // Reprise après acquittement
      if bAcquit
        taskCreate
      "Dialogue",100,dialogue()
        taskCreate
      "Mouvement",100,mouvement()
        bAcquit=false
        bAutom=false
      endIf
    endIf
  endWhile
end
```

3.10 Dégagement du bras et anticipation

Dans tous les cas (collision ou redémarrage du programme après un déplacement manuel), le mouvement du bras doit se faire de façon réfléchie. On doit éviter toute collision, et cela à partir de n'importe quelle position. Dans le cas d'un robot Scara (RS40), la stratégie de dégagement est évidente : vers le haut. C'est la même chose pour un robot cylindrique ou cartésien.

C'est pourquoi les sous programmes mvt_prise et mvt_pose ont été modifiés.

Ils commencent systématiquement par une phase de remontée (I). Cette phase n'est utile qu'en cas de redémarrage dans une position quelconque après un déplacement manuel.

Une deuxième modification consiste à anticiper la phase de pose après la prise (ou inversement). C'est la phase IV. Cela permet de réduire le temps de cycle et d'attendre l'ordre automate au plus près de l'action.

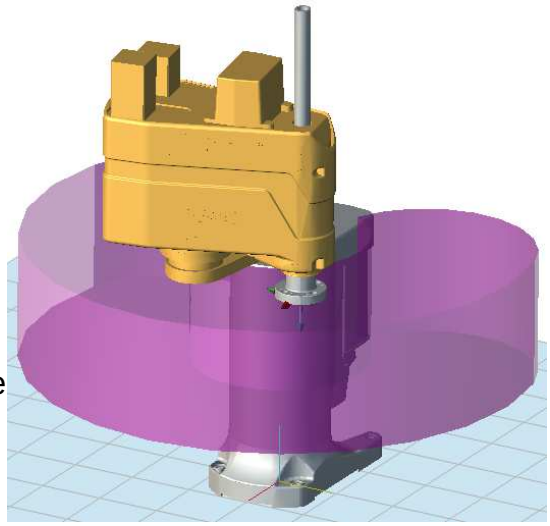
La phase d'approche normale (phase II) ne sera utile que lors du premier cycle. Dans les cycles suivants cette approche se fera par anticipation après la pose.

Ce mode de programmation repose sur le fait que le bras n'effectue aucun mouvement s'il est déjà en position. Le temps de calcul des points effectué inutilement est négligeable.

```
MVT_PRISE :
begin
  // MVT DE PRISE ET AVANCE VERS POSE
I-Dégagement de sécurité---------------
  call remonte(nHplafond)
II-Approche prise----------------------
  // Calcul du point de prise
 pCible=compose(pOrigine,world,trPrise)
  // Calcul du point au dessus
  pApproche=pCible
  pApproche.trsf.z=nHdegage
  // Déplacement au dessus
  movej(pApproche,tOutil,mVitesse)
III-Prise------------------------------
  // Déplacement vers prise
  movel(pCible,tOutil,mVitesse)
  // Prise de l'objet
  close(tOutil)
  // Dégagement vers le haut
  call remonte(nHplafond)
IV-Anticipation pose-------------------
  // Calcul du point de pose
pApproche=compose(pOrigine,world,trPose)
  // Modification de sa hauteur
  pApproche.trsf.z=nHdegage
  // Déplacement au dessus
  movej(pApproche,tOutil,mVitesse)
  // Attente de la fin du mouvement
  waitEndMove()
end
```

3.11 Butées logiciel, aspect

Le débattement des axes J1 à J4 peut-être limité par modification de la configuration du robot. On appelle ces limites des butées logiciel. Cela permet de restreindre la zone accessible par le robot. Cette zone est appelée « aspect » (Workspace en anglais).



D'autres méthodes plus élaborées permettent de définir des zones d'exclusion qui peuvent tenir compte de l'outil mais aussi de toutes les articulations du bras (poignet, coude).

On réduit généralement la zone accessible à la taille de l'enceinte. Cela évite tout risque d'endommagement des vitres pendant la phase de mise au point des programmes et même pendant les déplacements du bras en mode manuel.

3.12 Arrêt d'urgence

L'arrêt d'urgence de la cellule est géré par le module de sécurité de l'automate. Il entraîne l'arrêt d'urgence du robot et sa mise hors tension. Aucun mouvement n'est plus possible, même en mode manuel. Le redémarrage passe alors par la phase d'initialisation de la machine complète.

L'arrêt d'urgence du robot, souvent déclenché par l'ouverture des portes, peut être géré par le sous programme securite. Il entraîne un dialogue avec l'automate. Il bloque immédiatement les mouvements programmés du bras et fige l'état de la pince. Il permet néanmoins un pilotage manuel du bras à partir du pendant du robot. Après acquittement de la part de l'opérateur, le cycle de production peut repartir.

4 Programmation du robot FANUC LRMate 200 (langage TP)

Les robots FANUC peuvent se programmer dans deux langages différents : le langage Karel, proche du VAL3 exposé au § 3 et le langage TP. Ce dernier a pour avantage d'être plus simple et de permettre la programmation « on line » directement sur le robot. C'est celui-ci qui est décrit dans les pages suivantes.

4.1 Déclaration des variables

En langage TP, le nom des variables n'est pas libre. Le programmeur dispose principalement de 6 tableaux pour définir ses variables :

- le tableau (on parle de registre) R[1] à R[200] pour les variables numériques,
- le registre PR (point register) PR[1] à PR[100] pour les points et les repères,
- les registres DO (digital output) et DI (digital input) pour les entrée-sorties logiques,
- les registres RO et RI pour les entrées sorties logiques câblées sur le bras (pince).

Le programme INTER décrit un mouvement de prise et dépose équivalent à l'application INTERMED décrite précédemment. Il est illustré par la vidéo du même nom.

Dans ce programme les registres sont les suivants :

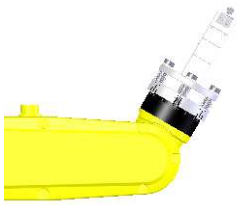
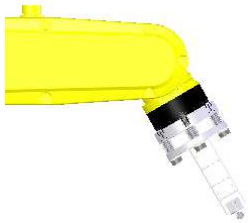
Registre numérique R R[1] valeur de la vitesse rapide (0 à 100%) R[2] valeur de la vitesse lente 250 mm/s R[3] valeur de l'arrondi de lissage (0 à 100)	NUMREG [1] = 50 'VITESSE' [2] = 250 'VITESSE APPROCHE' [3] = 50 'ARRONDI'
Registre des points et repères PR PR[1] est le repère outil : longueur 135,65 mm et rotation de 90° par rapport à la bride. PR[2] est le repère utilisateur de prise exprimé dans le repère du robot (world). PR[3] est le repère utilisateur de pose. PR[4] est le point origine utilisable dans chacun des deux repères utilisateur. Il impose la configuration du bras (voir plus loin). W=180° est indispensable (voir § 2.5). PR[5] est le décalage (100 mm vers le haut) que l'on va imposer entre le point d'approche et le point cible (voir plus loin). PR[6] définira la position initiale du bras en articulaire.	POSREG [1] = 'PINCE FLACON' X: 0.0 Y: 0.0 Z: 135.650 W: 0.0 P: 0.0 R: 90.0 [2] = 'BUTEE' X: 550.0 Y: -280.0 Z: -122.0 W: 0.0 P: 0.0 R: 0.0 [3] = 'TAPIS' X: 550.0 Y: 330.0 Z: -122.0 W: 0.0 P: 0.0 R: 0.0 [4] = 'ORIGINE' Config: N U T X: 0.0 Y: 0.0 Z: 0.0 W: 180.0 P: 0.0 R: 180.0 [5] = 'DEGAGEMENT' X: 0.0 Y: 0.0 Z: 100.0 W: 0.0 P: 0.0 R: 0.0 [6] = 'POINT INIT' J1 = 0.0 deg J2 = 30.0 deg J3 = 0.0 deg J4 = 0.0 deg J5 = -90.0 deg J6 = -90.0 deg

4.2 Configurations du bras

Dans le registre PR, le champ « Config » est destiné à indiquer la configuration du bras. Il n'est actif que si le registre désigne un point. Dans le cas d'un repère utilisateur, d'un repère outil ou d'un décalage, il ne sert à rien.

Dans le cas d'un robot Scara, il n'y avait que deux configurations : coude à gauche et coude à droite. Dans le cas d'un robot anthropomorphe, on ajoute le configuration du poignet et de l'épaule.

Cela donne 6 possibilités :

Poignet	plié vers le haut F pour « flip »		plié vers le bas N pour « no flip »	
Epaule	plaçant le coude au dessus de la ligne épaule – poignet U pour « up »		plaçant le coude en dessous de la ligne épaule – poignet D pour « down »	
Coude	plaçant le bras à l'avant de l'axe vertical J1 T pour «toward»		plaçant le bras à l'arrière de l'axe vertical J1 B pour « backward »	

La configuration du point origine dans le programme INTER est donc N,U,T.

Attention : le langage TP ne permet pas d'aller à un point défini en coordonnées cartésiennes si cela nécessite un changement de configuration. Il faut alors intercaler un point défini en articulaire qui impose la même configuration que le point final.

La vidéo « Configurations » montre un mouvement entièrement réalisé avec des points définis en coordonnées articulaires.

4.3 Programmation intermédiaire

Voici le programme INTER de prise et dépose d'objet et les sous-programmes INTER_PRISE et INTER_POSE. !DESIGNE DES COMMENTAIRES.

Les mouvements effectués par le robot sont décrits dans la vidéo « INTER ».On remarquera que l'opérateur peut effectuer plusieurs cycles sans avoir à relancer le programme.

Le programme commence par une initialisation du repère outil (UTOOL) et des deux repères utilisateur (UFRAME) qui seront utiles par la suite.

On sélectionne ensuite l'outil qui sera actif pour tous les déplacements : UTOOL[1].

On se déplace au point PR[6] sans contrôle de trajectoire (J) à la vitesse de 10 % et sans lissage (CNT0).

La marche du tapis et la détection du pot est gérée par le contrôleur du robot à l'aide de la sortie DO[101] et de l'entrée DI[101].

La boucle assurant la marche continue est du type :

```
Répéter
   Répéter
      lire DO[1] et DO[2]            (WAIT)
   Jusqu'à DO[1]=ON ou DO[2]=ON
   Si DO[2]=OFF faire
      ...
   Fin si
Jusqu'à DO[2]=ON
```

DO[1] et DO[2] sont reliés aux boutons poussoir virtuels C/C et QUIT de la page html qui permet le dialogue avec l'opérateur (voir plus loin).

Les structures de programmation à base de labels (LBL) et d'instructions JUMP (JMP) sont plus souples mais plus difficiles à mettre au point que les structures while – endWhile du langage VAL3.

```
!DEFINITION DES REPERES
  UTOOL[1]=PR[1:PINCE FLACON]
  UFRAME[1]=PR[2:BUTEE]
  UFRAME[2]=PR[3:TAPIS]
!INITIALISATION DU BRAS
  UTOOL_NUM=1
  J PR[6:POINT INIT] 10% CNT0
!AVANCE TAPIS
  DO[101:AVANCE TAPIS]=ON
!DEBUT BOUCLE
  LBL[1]
  WAIT DO[1]=ON OR DO[2]=ON
  IF DO[2]=ON,JMP LBL[2]
    !PRISE POT
    WAIT DI[101:DETECTION POT]=ON
    CALL INTER_PRISE
    !ARRET TAPIS
    DO[101:AVANCE TAPIS]=OFF
    !DEPOSE POT
    CALL INTER_POSE
    !AVANCE TAPIS
    DO[101:AVANCE TAPIS]=ON
!FIN BOUCLE
  JMP LBL[1]
  LBL[2]
!RETOUR POINT INIT
  J PR[6:POINT INIT] 10% CNT0
```

Le sous-programme INTER_PRISE reprend la structure du sous programme de prise du RS40.

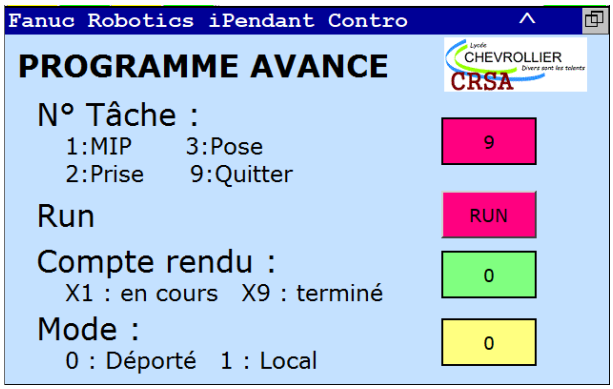
On sélectionne d'abord le repère utilisateur qui sera actif pour tous les déplacements : UFRAME[1]. On se déplace ensuite au point origine (du repère UFRAME[1]) sans contrôle de trajectoire (J), à la vitesse définie dans R[1], avec l'arrondi de lissage défini dans R[3] et le décalage (offset) défini dans PR[5]. C'est le point d'approche. On se déplace ensuite au même point mais en ligne droite (L), à la vitesse définie dans R[2], sans lissage (CNT0) et sans décalage. C'est le point cible. L'appel à INTER_SIM_PRISE ne sert qu'à la simulation. La fermeture de la pince est obtenue par la mise à off de la sortie RO[7]. On revient au point d'approche en ligne droite par l'ultime mouvement du sous programme.	!PRISE POT UFRAME_NUM=1 !MVT APLOMB CIBLE J PR[4:ORIGINE] R[1:VITESSE]% CNT R[3:ARRONDI] Offset,PR[5:DEGAGEMENT] !MVT CONTACT CIBLE L PR[4:ORIGINE] R[2:VITESSE APPROCHE]mm/sec CNT0 !ACTION PREHENSEUR *CALL INTER_SIM_PRISE* RO[7:OPEN/CLOSE PINCE]=OFF WAIT .20(sec) !MVT APLOMB CIBLE L PR[4:ORIGINE] R[2:VITESSE APPROCHE]mm/sec CNT R[3:ARRONDI] Offset,PR[5:DEGAGEMENT]

Le sous-programme INTER_POSE ne diffère que par les deux lignes :

UFRAME_NUM=2 qui rend actif le repère utilisateur de pose et
RO[7:OPEN/CLOSE PINCE]=ON qui ouvre la pince.

4.4 Dialogue avec l'opérateur

4.4.1 Création de la page de dialogue



Elle est créée en html, avec le logiciel « Microsoft Office Sharepoint Designer » à partir de l'exemple FANUC « ipctrls.stm ».

Ce fichier comporte tous les objets nécessaires (interrupteurs, zone d'édition) pour agir sur les registres numériques R et les sorties logiques DO.

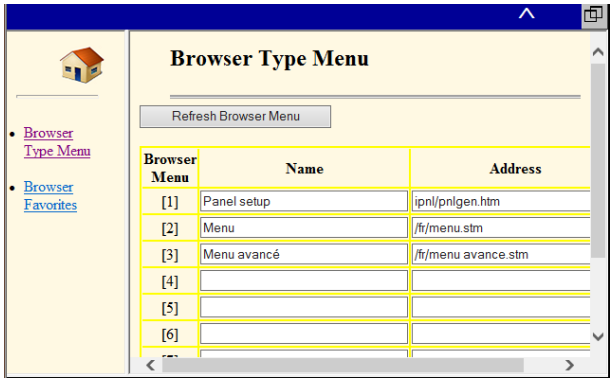
La page créée (menu avance.stm) doit être chargée en mémoire à partir par exemple, du port USB UD1 (Menu FILE/File). Toutes les images (.GIF) doivent être chargées de même.

4.4.2 Accès à la page de dialogue

Pour être disponible, la page doit être référencée dans la liste des menus utilisateur.

On accède à cette liste par le menu BROWSER/Browser/HOME/Brower Type Menu. Il faut alors ajouter un nouveau menu : « Menu avancé » et faire le lien avec le fichier /fr/menu avance.stm.

Une fois la page référencée elle apparaît dans le menu BROWSER.



Elle peut être stockée dans les favoris en appuyant longuement sur la touche Menu.

4.5 Lancement des tâches

Une méthode permettant de lancer les tâches robot, soit du pendant (mode local), soit de l'automate (mode déporté) consiste à utiliser un programme principal qui distribue les tâches en fonction d'une variable numérique.

4.5.1 programme principal

Le programme AVANCE décrit ci-dessous, montre cette solution dans la vidéo du même nom. L'équipe Chevrollier n'ayant pas testé cette solution en projet industriel, il ne s'agit que d'une piste possible.

Appel au sous programme d'initialisation.
Entrée dans la boucle « répéter » :
si le mode est déporté,
- le n°de tâche est lu sur un groupement de 8 entrées logiques* formant une variable numérique (* câblées ou réseau).
- le compte-rendu est écrit sur un groupement de 8 sorties logiques* formant une variable numérique,
- le déclenchement (RUN) est lu sur une entrée logique.
Si le mode est local,
- le déclenchement (RUN) est lu sur une sortie logique de la page menu utilisateur.
 	
Les registres de tâche et de compte rendu étant en lien permanent avec les champs numériques de la page utilisateur, ils n'apparaissent pas dans le programme.
	!INITIALISATION
	CALL AVANCE_INIT
	!DEBUT BOUCLE
	LBL[1]
	!SELECTION MODE
	IF (R[16:MODE]=0) THEN
	!MODE DEPORTE
	R[14:N°TACHE]=GI[1:ENTREES TACHES]
	
	GO[1:SORTIES CPT RENDU]
	=R[15:COMPTE RENDU]
	
	R[17:RUN]=DI[17]
	ELSE
	!MODE LOCAL
	R[17:RUN]=DO[3:BOUTON RUN]
	ENDIF

Si le déclenchement est détecté (RUN) et selon le n°de tâche, on effectue :
- l'envoi du compte rendu de début de tâche,
- l'appel du sous programme de tâche,
- l'envoi du compte rendu de fin de tâche.
	!SELECTION TACHE
	!MIP
	IF (R[14:N°TACHE]=1 AND R[17:RUN]=1) THEN
	R[15:COMPTE RENDU]=11
	CALL AVANCE_MIP
	R[15:COMPTE RENDU]=19
	ENDIF
	!PRISE
	...
	!POSE
	...

On sort de la boucle « répéter » si n°tâche=9	!FIN BOUCLE IF R[14:N°TACHE]<>9,JMP LBL[1]

Remarque : des entrées logiques spécifiques (UIP) permettent à un automate de démarrer à distance les programmes robot (lire à ce sujet, l'excellent ouvrage « Fanuc academy, La robotique industrielle, guide de l'utilisateur» chez hachette technique). Mais ces entrées ne sont pas utilisables par la page html de menu qui ne peut forcer que des sorties logiques ou éditer des registres.

4.5.2 Sous-programmes

Le sous programme d'initialisation reprend la partie initialisation du programme INTER.

Définition des repères outil et utilisateurs.	!DEFINITION DES REPERES UTOOL[1]=PR[11:PINCE FLACON] UFRAME[1]=PR[12:BUTEE] UFRAME[2]=PR[13:TAPIS]
Choix de l'outil actif.	!CHOIX DE L'OUTIL UTOOL_NUM=1
Remise à zéro des variables : numéro de tâche et compte rendu du robot. Choix du mode déporté par défaut.	!RAZ TACHE ET CPT RENDU R[14:N°TACHE]=0 R[15:COMPTE RENDU]=0 R[16:MODE]=0
Ouverture préventive de la pince.	!OUVERTURE PINCE RO[7:OPEN/CLOSE PINCE]=ON

Le sous programme de prise est enrichi par l'anticipation du mouvement de pose afin de réduire le temps de cycle.

Choix du repère utilisateur actif (prise).	!PRISE FLACON UFRAME_NUM=1
Ouverture préventive de la pince.	RO[7:OPEN/CLOSE PINCE]=ON
Déplacement rapide au point d'approche de prise (origine avec décalage).	!MVT APLOMB CIBLE J PR[14:ORIGINE] R[11:VITESSE]% CNT R[13:ARRONDI] Offset,PR[15:DEGAGEMENT]
Déplacement linéaire au point cible (origine).	!MVT CONTACT CIBLE L PR[14:ORIGINE] R[12:VITESSE APPROCHE]mm/sec CNT0
Fermeture de la pince.	!ACTION PREHENSEUR RO[7:OPEN/CLOSE PINCE]=OFF WAIT .20(sec)
Dégagement vertical de l'objet (origine avec décalage).	!MVT APLOMB CIBLE L PR[14:ORIGINE] R[12:VITESSE APPROCHE]mm/sec CNT R[13:ARRONDI] Offset,PR[5:DEGAGEMENT]
Changement du repère utilisateur actif (pose)	!ANTICIPATION POSE UFRAME_NUM=2
Déplacement rapide au point d'approche de pose (origine avec décalage).	J PR[14:ORIGINE] R[11:VITESSE]% CNT R[13:ARRONDI] Offset,PR[15:DEGAGEMENT]

4.6 Coexistence des programmes dans le contrôleur

Faire coexister plusieurs programmes en même temps sur le contrôleur (INTER et AVANCE par exemple) nécessite de prendre des précautions. En effet, les registres R, PR, DI, DO... sont communs. Il faut donc attribuer des tranches à chaque programme : dans notre cas, R[1] à R[10] sont réservés au programme INTER alors que R[11] à R[20] sont réservés au programme AVANCE.

Tous les programmes apparaissent dans la liste de sélection. Il convient de les nommer de manière qu'ils soient groupés : AVANCE, AVANCE_INIT, AVANCE_MIP,…

4.7 Autres fonctions

La mise sous tension du bras et la détection de collision sont possibles grâce à des entrées et des sorties logiques spécifiques. Câblées sur l'automate, elles lui permettent le piloter le robot.

Une zone de sécurité complexe, tenant compte du volume du bras et de son outil peut être définie.

Nous n'avons pas exploré ces fonctions.

5 Programmation du robot ABB IRB 120 (langage Rapid)

Les robots ABB se programment en langage Rapid, proche du VAL3 exposé au § 3. L'application RAPID décrite ci-dessous, réalise le même processus de prise et pose d'objet que l'application PARALLELE ou que le programme AVANCE présentés précédemment. Elle est illustrée dans la vidéo du même nom.

5.1 Déclaration des variables

La déclaration des variables et leur initialisation est réalisée dans le module INIT qui fait partie du programme MOUVEMENT de l'application RAPID (voir l'ensemble du programme en annexe).

On distingue 4 types de données :

- les variables VAR susceptibles d'être modifiées au cours de l'exécution,
- les constantes CONST qui ne le seront pas,
- les variables persistantes PERS et TASK PERS dont les valeurs seront conservées, même après l'arrêt du programme et dont les valeurs initiales tiendront compte de l'exécution précédente.

On retrouve les données numériques num, les lois de vitesse speeddata et les arrondis de lissage zonedata.

On retrouve également les points définis en articulaires jointtarget avec leurs 6 coordonnées.

Les points définis en cartésien robtarget possèdent 3 coordonnées de position X,Y et Z, mais leur orientation est définie par un quaternion (voir plus loin).

Leur configuration est définie par 3 chiffres qui définissent l'intervalle angulaire (quadrant) dans lequel se situe les axes 1, 4 et 6 (le dernier chiffre n'est pas utilisé).

```
MODULE INIT
!*****************************
! INITIALISATION
!*****************************
  ! Variables inter tâches
    PERS num nTache:=4;
    PERS num nMode:=4;
    PERS num nCptrendu:=39;
  ! Variables intra tâches
    VAR num nStache:=0;
  ! Vitesses
    CONST speeddata vHigh:=v200;
    CONST speeddata vLow:=v100;
  ! Conditions de passage
    CONST zonedata zArret:=z0;
    CONST zonedata zFin:=z20;
    CONST zonedata zLarge:=z50;
  ! Dégagements
    CONST num nDegagpot:=100;
  ! Points
    CONST jointtarget pInit:=[[30,-10,10,90,90,0],
    [0,0,0,0,0,0]];
    CONST robtarget pPrisepot:=[[0,0,0],[0,0,-1,0],
    [-1,0,-1,0],[0,0,0,0,0,0]];
    CONST robtarget pPosepot:=[[0,0,20],[0,0,-1,0],
    [-1,0,-1,0],[0,0,0,0,0,0]];
```

Le repère outil tooldata comporte également 3 coordonnées de position et un quaternion définissant son orientation. Idem pour les repères utilisateur wobjdata de prise et de pose. On verra au §5.6 l'usage des variables déclarées dans la rubrique !Interruptions	! Repères outils TASK PERS tooldata tPincepot:=[TRUE, [[0,-74.8,63.8],[0.7071,0.7071,0,0]], [1,[0,0,1],[1,0,0,0],0,0,0]]; ! Repères utilisateur TASK PERS wobjdata wPrisepot:=[FALSE,TRUE, "",[[350,-55,325],[1,0,0,0]],[[0,0,0],[1,0,0,0]]]; TASK PERS wobjdata wPosepot:=[FALSE,TRUE, "",[[350,320,325],[1,0,0,0]],[[0,0,0],[1,0,0,0]]]; ! Interruptions VAR Intnum sColl; VAR robtarget pArret:=[[0,0,0],[0,0,0,0],[0,0,0,0], [0,0,0,0,0,0]]; TASK PERS tooldata tArret:=[TRUE, [[0,-74.8,63.8],[0.7071,0.7071,0,0]],[1,[0,0,1], [1,0,0,0],0,0,0]]; TASK PERS wobjdata wArret:=[FALSE,TRUE, "",[[350,-55,325],[1,0,0,0]],[[0,0,0],[1,0,0,0]]]; ENDMODULE

Les modules DIALOGUE et COMMUNICATION des programmes des autres tâches (voir plus loin) doivent aussi déclarer les variables qu'ils utilisent. Certaines variables : nTache, nMode et nCptrendu sont communes à toutes les tâches. Elles doivent néanmoins être déclarées 3 fois.

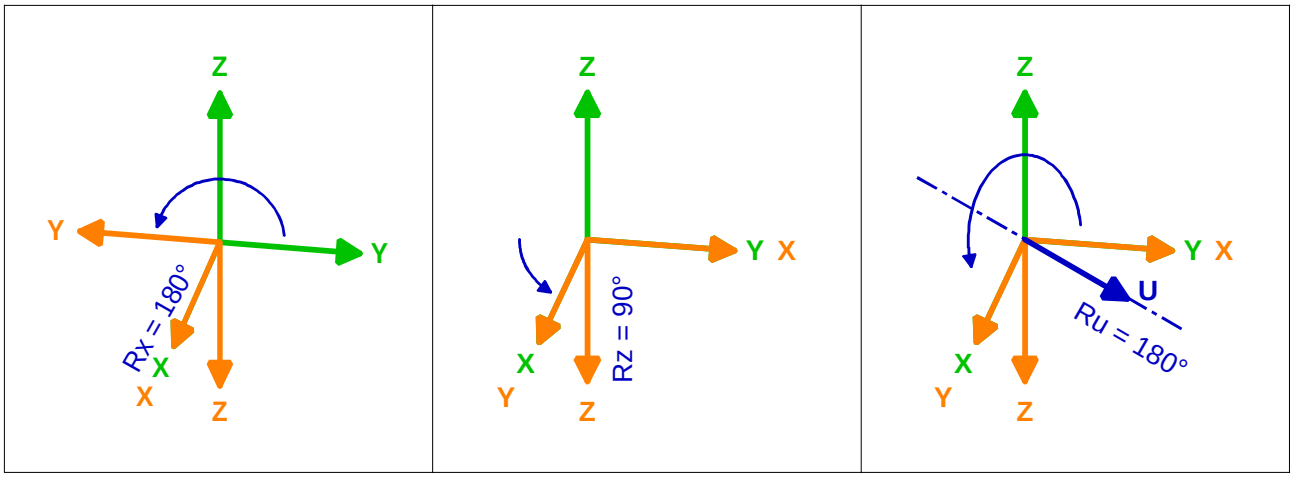
5.2 Quaternions

Le langage Rapid décrit l'orientation des repères ou des points par un élément mathématique appelé quaternion. Les autres langages utilisent trois angles appelés angles d'Euler.

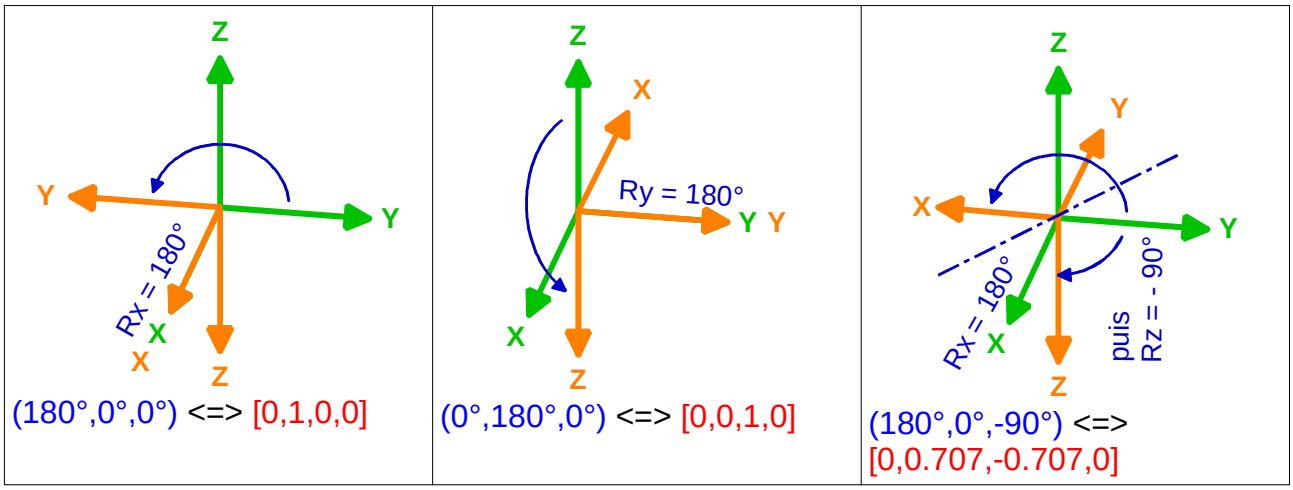
Prenons l'exemple de l'orientation « naturelle » de l'origine d'un repère utilisateur. On a vu au §2.5 que son orientation était Rx=180°, Ry=0° et Rz=0°. Ses coordonnées en Val3 ou en langage TP seront donc (0,0,0) pour la position et (180°,0°,0°) pour l'orientation. Si on veut tourner l'objet de 90° selon Z, l'orientation du point devient (180°,0°,90°).

Cette succession de deux rotations est équivalente à une rotation unique de 180° selon un axe orienté suivant le vecteur unitaire u de coordonnées (0.707,0.707,0) (voir ci-dessous).

Le quaternion décrivant cette rotation est alors composé de 4 nombres : le cosinus de l'angle de rotation divisé par et les 3 coordonnées du vecteur u. On obtient donc [cos(90°),0.707,0.707,0] soit [0,0.707,0.707,0].



Autres exemples simples :



Le contrôleur des robots ABB permet de mesurer la position du bras dans les deux systèmes, mais la déclaration dans les programmes doit utiliser les quaternions.

5.3 Programmation des mouvements

5.3.1 Module principal

La routine Main() du module PRINCIPAL du programme MOUVEMENT gère 3 tâches robot :
- tâche n°1 : mise en position initiale,
- tâche n°2 : mouvement de prise,
- tâche n°3 : mouvement de pose.

La tâche n°9 du RS40 (mise sous tension ou hors tension du bras) est remplacée par une entrée logique câblée sur l'automate.

On adopte la même logique de compte rendus que dans le cas du RS40 :
tâche X en cours => Compte rendu = X1,
tâche X finie => Compte rendu = X9.

La variable nTache repasse à zéro, empêchant le lancement de plusieurs tâches X successives (alternative à nRun).

La commande WaitTime(0) indique au contrôleur qu'il peut passer au programme suivant (équivalent de Delay(0) de Val3).

Chaque programme comporte une boucle WHILE nMode<>4 DO – ENDWHILE.

nMode=4 entraîne l'arrêt de tous les programmes.

La réinitialisation des variables, nMode, nTache et nCptrendu est indispensable car les valeurs déclarées dans le module INIT changent en fonction de l'exécution précédente (variables persistantes).

```
MODULE PRINCIPAL
!*****************************
! PROGRAMME PRINCIPAL  22 05 20
!*****************************
   PROC Main()
      !Initialisation du menu
      nMode:=1;
      nTache:=0;
      nCptrendu:=101;
      Worldzones ;
      ! Boucle des tâches
      WHILE nMode<>4 DO
         ! Boucle des mouvements
         TEST nTache
         CASE 1:
          nCptrendu:=11;
          Minit;
          nTache:=0;
          nCptrendu:=19;
         CASE 2:
          nCptrendu:=21;
          Mprise;
          nTache:=0;
          nCptrendu:=29;
         CASE 3:
          nCptrendu:=31;
          Mpose;
          nTache:=0;
          nCptrendu:=39;
         ENDTEST
         WaitTime 0;
      ENDWHILE
      WaitTime 0.5;
   ENDPROC
ENDMODULE
```

5.3.2 Module de prise

Dans la routine Mprise() appelée par la routine Main() du programme principal, on retrouve la succession des 4 mouvements : approche, prise, dégagement et anticipation.

Le premier mouvement est décrit de la façon suivante :
Mouvement sans contrôle de trajectoire MoveJ vers un point décalé par rapport au point de prise Offs(pPrisepot,0,0,nDegagpot) à la vitesse vHigh avec le lissage zLarge en prenant en compte l'outil tPincepot et le repère utilisateur wPrisepot.

Contrairement au langage TP, l'outil et le repère utilisateur doivent être spécifiés dans chaque commande de mouvement.

La fermeture de la pince est activée par la mise à 1 de la sortie DO2.
L'attente de la fin du mouvement de prise et un délai de 0,2 s est imposé avant sa fermeture.

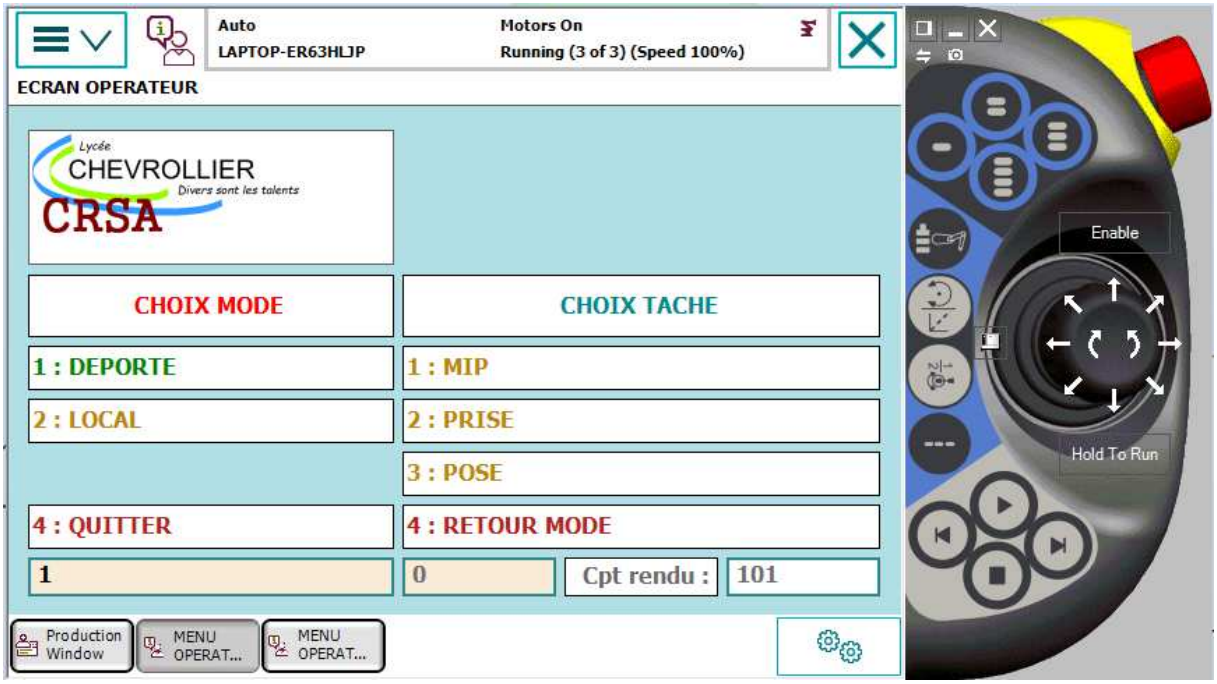
```
MODULE PRISE
!*******************************
! MOUVEMENT DE PRISE 22 05 20
!*******************************
  PROC Mprise()
    MoveJ
          Offs(pPrisepot,0,0,nDegagpot),
          vHigh,zLarge,
          tPincepot
          \Wobj:=wPrisepot;
    MoveL pPrisepot,
          vLow,zArret,
          tPincepot
          \WObj:=wPrisepot;
    WaitTime \InPos,0.2;
          SetDO DO2,1; ! Fermeture pince
    WaitTime 0.2;
    MoveL
          Offs(pPrisepot,0,0,nDegagpot),
          vLow,zFin,
          tPincepot
          \WObj:=wPrisepot;
    MoveJ
          Offs(pPosepot,0,0,nDegagpot),
          vHigh,zLarge,
          tPincepot\
          WObj:=wPosepot;
    ENDPROC
ENDMODULE
```

Voir en annexe les sous programmes de pose et de mise en position initiale.

5.4 Dialogue opérateur

5.4.1 Ecran de dialogue

Le logiciel ScreenMaker permet de réaliser des pupitres virtuels en associant des objets graphiques à des variables logiques ou numériques utilisées dans les programmes. Dans notre exemple, 2 zones de saisie permettent de choisir nMode et nTache et une zone d'affichage permet et de visualiser nCptrendu. ScreenMaker génère deux fichiers qui, placés dans le contrôleur du robot, rendent l'écran utilisateur immédiatement disponible.



5.4.2 Programme de dialogue

Bien que le pendant soit muni de son propre processeur et que le menu de dialogue puisse fonctionner en parallèle du robot, il est préférable de réaliser un programme de dialogue afin de gérer la logique des menus. Dans notre exemple, c'est ce programme qui permet de basculer du menu « Choix Mode » au menu « Choix tâche » et inversement.

La structure du programme de gestion des menus opérateur est la suivante :

Routine principale lancée au démarrage du programme :

```
Main()
Tant que nMode ≠ 4 faire :
  activer le menu « Choix mode »
  désactiver le menu « Choix tâche »
  Si  nMode = 2 faire :
     appeler Saisietache
  Fin si
Fin tant que
```

Routine auxiliaire (sous-programme) appelée par Main() :

```
Tant que nTache ≠ 4 faire :
  désactiver le menu « Choix mode »
  activer le menu « Choix tâche »
Fin tant que
```

La saisie proprement dite et la gestion des différentes couleurs d'affichage sont gérées par les objets graphiques de la page ScreenMaker qui réagissent aux valeurs des variables logiques bMode et bTache et des variables numériques nMode et nTache.

```
MODULE DIALOGUE
!*****************************
! DIALOGUE OPERATEUR 22 05 20
!*****************************
 ! Définition des variables
   PERS num nTache:=0;
   PERS num nMode:=4;
   PERS num nCptrendu:=101;
   PERS bool bMode:=TRUE;
   PERS bool bTache:=FALSE;
   ! MENU MODE DE FONCTIONNEMENT
   PROC Main()
      WHILE nMode<>4 DO
         bMode:=TRUE;
         bTache:=FALSE;
         IF nMode=2 THEN
               Saisietache;
         ENDIF
         WaitTime 0.1;
      ENDWHILE
   ENDPROC
   ! MENU DE CHOIX DES TACHES
   PROC Saisietache()
      nTache:=0;
      WHILE nTache<>4 DO
         bMode:=FALSE;
         bTache:=TRUE;
         WaitTime 0.1;
      ENDWHILE
      nMode:=0;
   ENDPROC
ENDMODULE
```

5.5 Communication avec l'automate

Le programme COMMUNICATION a pour rôle de synchroniser constamment les variables nTache, nMode et nCptrendu avec les entrées et les sortie analogiques de l'automate. Ici, ces entrées sorties empruntent le câble Ethernet.

La structure de ce programme est la suivante :

Routine principale lancée au démarrage du programme :

```
Main()
Tant que nMode ≠ 4 faire :
  Si  nMode = 1 faire :
     nTache =  EthAI1
     nMode =  EthAI0
     EthAO0 = nCptrendu
  Fin si
Fin tant que
```

Si nMode = 2 (mode local), les entrées – sorties de l'automate ne sont pas lues.

```
MODULE COMMUNICATION
!*****************************
! COMMUNICATION AUTOMATE 22 05 20
!*****************************
 ! Définition des variables
   PERS num nTache:=0;
   PERS num nMode:=1;
   PERS num nCptrendu:=19;
   ! CYCLE DE FONCTIONNEMENT
   PROC Main()
      WHILE nMode<>4 DO
         IF nMode=1 THEN
            nTache:=EthAI1;
            nMode:=EthAI0;
            SetAO EthAO0,nCptrendu;
         ENDIF
         WaitTime 0;
      ENDWHILE
      WaitTime 0.5;
   ENDPROC
ENDMODULE
```

5.6 Programmation multitâches

On s'appuie sur le fonctionnement simultané de trois programmes pour assurer la communication avec l'automate ou avec l'opérateur indépendamment des mouvements du robot, comme dans le cas du RS40. Si un seul programme gérait l'ensemble, cette communication serait interrompue lorsque le programme attend la fin d'un mouvement du robot.

En programmation Rapid, il faut pour cela configurer le contrôleur de manière qu'il gère 3 tâches. Un programme est chargé dans chacune de ces tâches (MOUVEMENT, DIALOGUE, COMMUNICATION). Un seul de ces programme peut contenir des mouvements.

La fonction de sécurité, qui comporte un arrêt puis un recul du bras, ne peut donc pas être assurée par un quatrième programme fonctionnant en parallèle des trois premiers.

5.7 Interruptions

La fonction de sécurité, ne pouvant pas être assurée par un quatrième programme, est gérée par une routine d'interruption. Lorsque le capteur détecte une collision, le programme mouvement est interrompu ainsi que le sous programme en cours (prise, pose,...) et c'est la routine Collision qui s'exécute à la place. A la fin de celle ci, le programme reprend son cours normal sauf si la routine d'interruption le fait redémarrer au début. C'est ce qui se passe dans notre exemple.

Au début de la routine Main() du module PRINCIPAL du programme MOUVEMENT, la variable sColl est :
- remise à zéro (IDelete),
- liée à la routine nommée Collision,
- liée à l'entrée logique DI0 (front montant)

Une routine de type TRAP est ajoutée en fin du module PRINCIPAL
La routine Collision consiste à :
- arrêter le mouvement
- stocker la trajectoire en cours
- l'effacer
- ré-autoriser les mouvements
- stocker dans les variables tArret, wArret et pArret, l'outil, le repère utilisateur et le point courant,
- déplacer le bras au point courant, mais avec un décalage vertical de 100 mm à petite vitesse et sans lissage, en tenant compte de l'outil et du repère utilisateur courant.

La routine demande enfin le redémarrage du programme MOUVEMENT (ExitCycle).

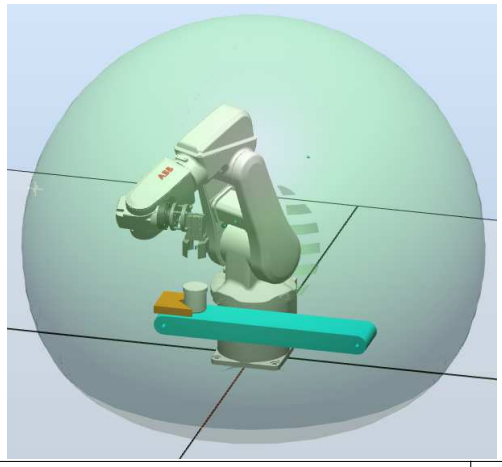
```
! Démarrage de la surveillance
    IDelete sColl;
    CONNECT sColl WITH Collision;
    ISignalDI DI0, high, sColl;
-------------------------------------------------------------------
! INTERRUPTION COLLISION
  TRAP Collision
    StopMove \Quick;
    StorePath;
    ClearPath;
    StartMove;
    tArret:=Ctool();
    wArret:=CWobj();
    pArret:=CRobT(\Tool:=tArret, \Wobj:=wArret);
    MoveL
      Offs(pArret,0,0,100),
      vLow,zFin,
      tArret\WObj:=wArret;
    ExitCycle;
  ENDTRAP
```

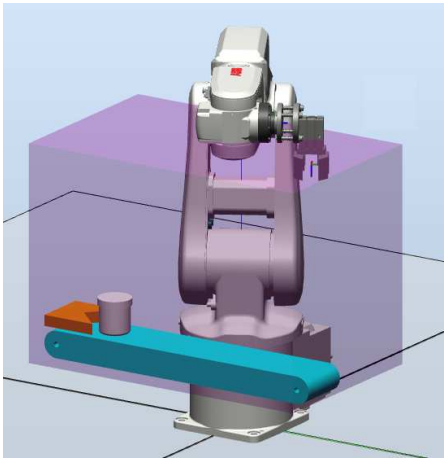
Les deux autres programmes DIALOGUE et COMMUNICATION ont aussi une routine Collision qui réagit à la même variable sColl et qui se contente de redémarrer le programme après un délai de 3 secondes (voir intégralité des programmes en annexe).

D'autres stratégies plus évoluées sont naturellement possibles pour réagir à une collision.

5.8 Zones de travail

Le langage Rapid permet de définir deux types de zones de sécurité : une zone définie en limitant le débattement des axes J1 à J6 et une zone définie comme un volume dans lequel doit rester l'extrémité de l'outil. Ces zones sont modifiables lors de l'exécution du programme et peuvent rester actives en mode manuel.





Les deux zones définies dans notre exemple sont :

- La zone Limitaxes définie par les butées Aspecttravail, définies elles-mêmes par les deux points de coordonnées articulaires low_pos et high_pos.

Ce qui donne :

-30 < J1< 35 70<J4<110
-15 < J2 < 30 60<J5<120
-5 < J3 < 15 -45<J6<5

- La zone Zoneactive définie par la forme Cubetravail, parallélépipède formé par les limites

100<X<450 ;
-150<Y<400 ;
200<Z<600.

Lors de l'exécution, un mouvement qui conduirait le bras hors de ces zones n'est pas exécuté et conduit à l'arrêt des programmes. L'erreur n'est pas aiguillable vers une routine d'interruption.

```
MODULE WORLDZONE
!*******************************
! DEFINITION DES ZONES DE TRAVAIL 22 05 20
!*******************************
   ! Worldzones
   VAR shapedata Cubetravail;
   VAR shapedata Aspecttravail;
   VAR wztemporary Zoneactive;
   VAR wztemporary Limitaxes;
   CONST jointtarget low_pos:=
      [[-30, -15, -5, 70, 60, -45],[0, 0, 0, 0, 0, 0]];
   CONST jointtarget high_pos:=
      [[35, 30, 15, 110,120, 5],[0, 0, 0, 0, 0, 0]];
   ! MISE EN PLACE
   PROC Worldzones()
      WZFree Zoneactive;
      WZFree Limitaxes;
      WZBoxDef \outside, Cubetravail,
      [100,-150,200],[450,400,600];
      WZLimJointDef \Outside, Aspecttravail,
                              low_pos, high_pos;
      WZLimSup \Temp, Zoneactive, Cubetravail;
      WZLimSup \Temp, Limitaxes, Aspecttravail;
   ENDPROC
ENDMODULE
```

6 Programmation du robot UR5 (Universal Robot)

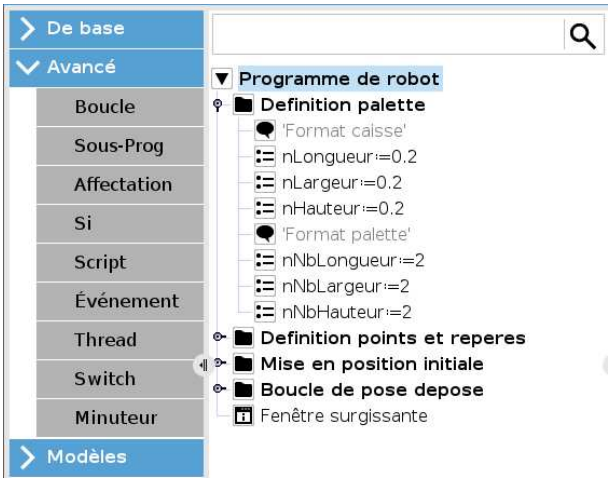
Les robots Universal Robot sont des robots collaboratifs. Ils sont conçus pour être programmés facilement par les opérateurs et pour travailler auprès d'eux sans risques.

A ce titre, le programme présenté ci-dessous est simple et ne traite ni de la sécurité qui dépend du réglage de la sensibilité du robot, ni du dialogue qui est limité au choix et au lancement du programme, ni de la communication avec un automate souvent absent.

La programmation se fait à partir du pendant du robot dans un langage graphique (PolyScope).

L'application décrite ci-dessous, réalise un processus de prise et de pose d'objets selon un schéma de palletisation. Elle est illustrée dans la vidéo du même nom.

6.1 Initialisation des variables

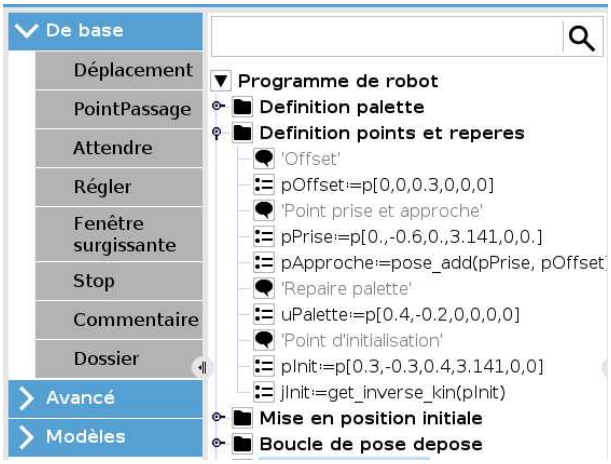


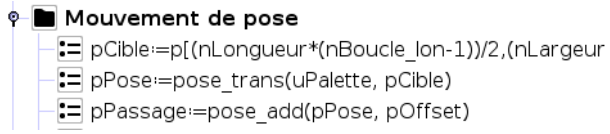
Les variables n'ont pas besoin d'être déclarées au préalable. Elles sont crées lors de leur initialisation.

Les différentes variables (exemple : nombre et dimensions des objets à palettiser) sont définies par la commande "Affectation".

Cette phase d'initialisation des variables est rangée dans un dossier du programme pour plus de clarté.

6.2 Initialisation des points et repères





Un point (variable de type "pose") est défini en cartésien par ses 6 coordonnées exprimées en mètre et en radian, exemple pPrise (on reconnaît le Rx = 180°).

On peut utiliser pour cela des variables numériques, exemple pCible.

Un décalage (offset) peut être ajouté (fonction "pose_add") pour créer un point d'approche, exemple pApproche.

Un repère utilisateur sera défini comme un point, exemple uPalette.

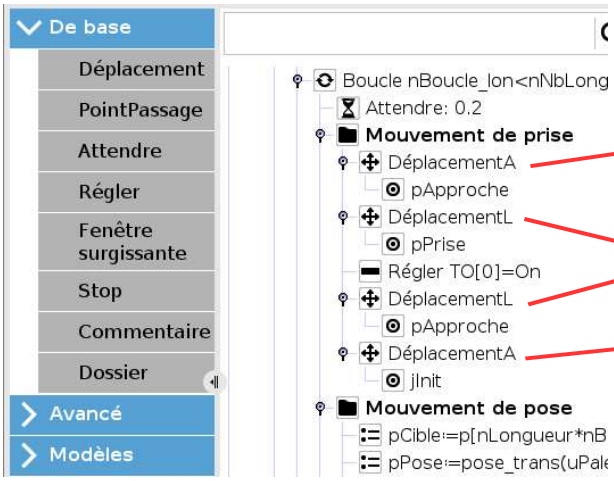
L'expression d'un point (pCible) dans ce repère utilisera la fonction "pose_tran", exemple pPose.

Un point défini en coordonnées articulaires, le sera par la liste de ses 6 coordonnées, ou à l'aide de la fonction get_inverse_kin, exemple jInit.

Les robots UR utilisent le vecteur rotation et non les angles d'Euler, pour définir l'orientation d'un point. Les 2 rotations π selon x puis π/2 selon z seront remplacées par une rotation unique de π selon le vecteur unitaire (0,707;0,707;0) (voir §5.2). L'orientation sera alors définie par le vecteur rotation π x (0,707;0,707;0) = (2,221;2,221;0).

6.3 Mouvements

Comme pour les autres langages de programmation on peut définir 3 mouvements :



- mouvement vers un point défini en coordonnées cartésiennes sans contrôle le trajectoire
- mouvement linéaire vers un point défini en coordonnées cartésiennes
- mouvement vers un point défini en coordonnées articulaires sans contrôle le trajectoire

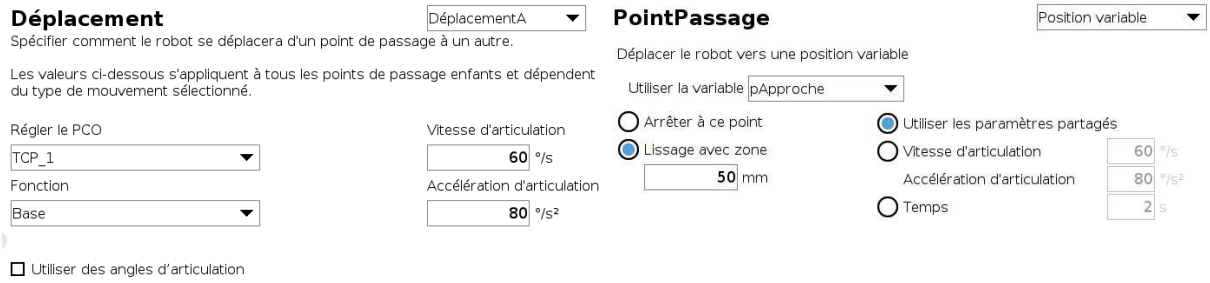
Le type de coordonnées (cartésiennes ou articulaires), la loi de vitesse et le repère outil utilisé (PCO) sont définis dans l'instruction de déplacement.

La condition d'arrêt (lissage de trajectoire) est définie dans le point.

6.3.1 Mvt vers un pt défini en coor. cartésiennes ss contrôle le trajectoire

Exemple d'un déplacement de type A qui s'effectue avec l'outil TCP_1, dont la vitesse et l'accélération sont définies en °/s et °/s².

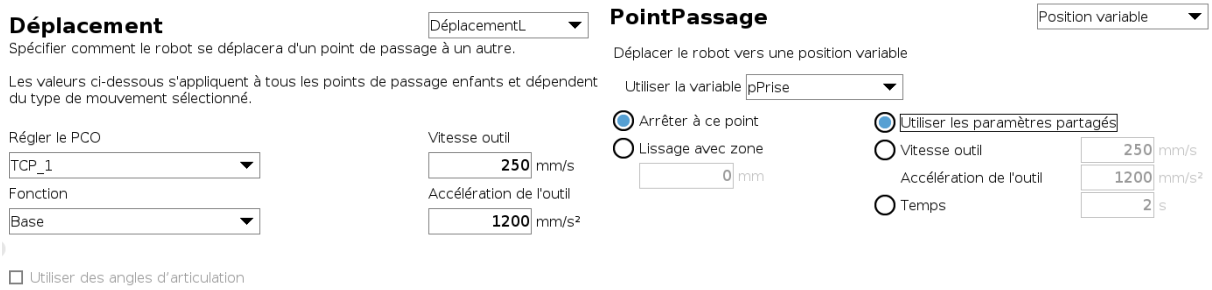
Le point est défini par la variable pApproche et le lissage de trajectoire est de 50 mm.



6.3.2 Mvt linéaire vers un pt défini en coordonnées cartésiennes

Exemple d'un déplacement de type L qui s'effectue avec l'outil TCP_1, dont la vitesse et l'accélération sont définies en mm/s et mm/s².

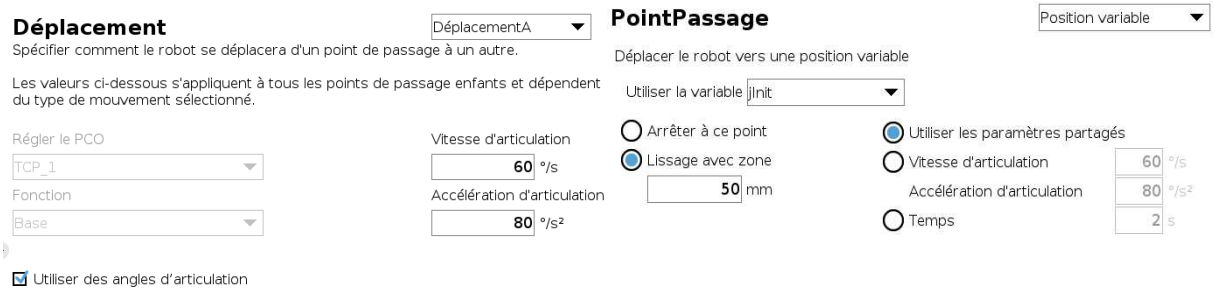
Le point est défini par la variable pPrise avec arrêt au point.



6.3.3 Mvt vers un pt défini en coor. articulaires ss contrôle le trajectoire

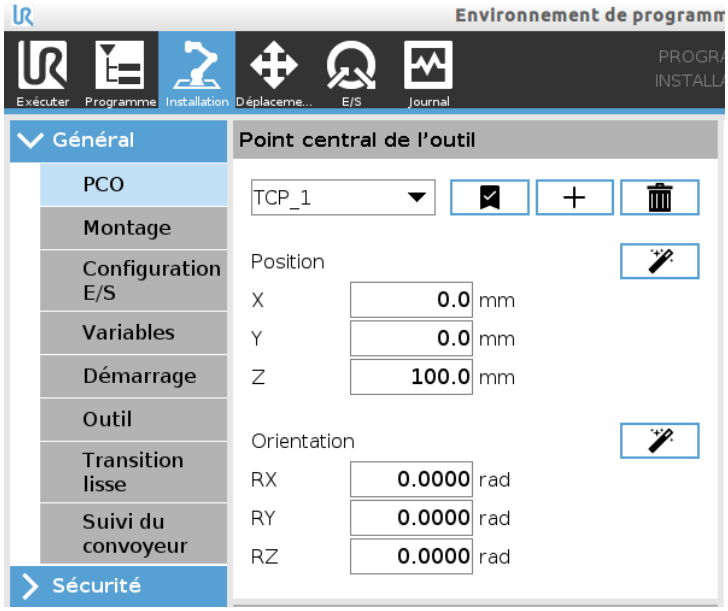
Exemple d'un déplacement de type A dont l'outil n'est pas spécifié car le point est défini en articulaire, dont la vitesse et l'accélération sont définies en °/s et °/s².

Le point est défini par la variable Jinit avec un lissage de trajectoire de 50 mm.



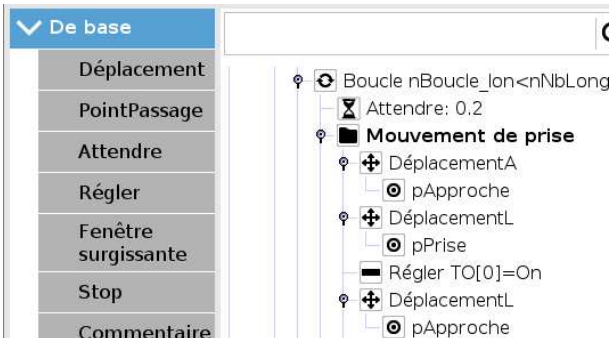
6.4 Définition de l'outil

C'est dans l'onglet "Installation" que l'on peut définir les coordonnées du repère outil qui sera utilisé pour tous les points définis en coordonnées cartésiennes (ici TCP_1).



6.5 Gestion de l'outil et de la charge utile

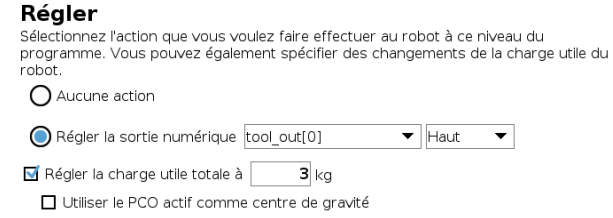
L'ouverture et la fermeture de l'outil se fait à l'aide de la commande "Régler". On spécifie alors la sortie du contrôleur utilisée pour piloter l'outil et le niveau de cette sortie (haut ou bas).



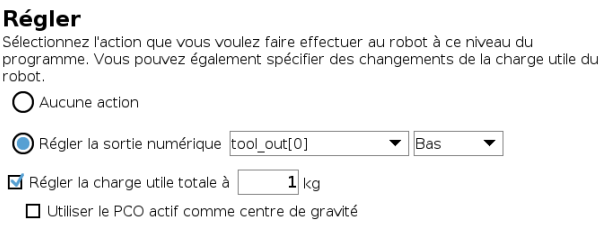
C'est à ce moment que l'on peut modifier la charge déclarée au bout du bras.

La capacité du robot à distinguer les efforts normaux (déplacement de la charge) des efforts anormaux (collision) dépend en grande partie de la charge qu'on lui déclare. Il est donc important de déclarer chaque modification de la charge.

Prise :



Pose :



7 Comparaison des différents langages

7.1 Clarté des programmes

7.1.1 Commentaires

Les quatre langages de programmation permettent de rédiger d'abondants commentaires. Ces commentaires sont indispensables à la clarté des programmes. C'est primordial pour un travail d'équipe, lors de la conception et la mise au point de la cellule robotisée, mais aussi lors de sa maintenance et de son éventuel retrofit.

7.1.2 Structures de programmation

La compréhension du fonctionnement d'un programme dépend, de la clarté des structures de bases, structures qui sont communes à tous les langages informatiques.

Structures	VAL3	RAPID	TP	Polyscope

Si (...) alors
...
sinon
...
Fin si
	`if(...)`
	`...`
	`else`
	`...`
	`endIf`
		IF (...) THEN
		...
		ELSE
		...
		ENDIF
			IF (...) THEN
			...
			ELSE
			...
			ENDIF
				Si
				...
				SinonSI
				...
				Sinon
				...

Pour X de 1 à N
faire
...
Fin pour
	`for X=1 to N`
	`...`
	`endFor`
		FOR X FROM 1 TO N
		...
		ENDFOR
			FOR R[...]=1 TO R[...]
			...
			ENDFOR
				Boucle X fois

Selon X =
1, faire
...
2, faire
...
Fin selon
	`switch X`
	`case 1`
	`...`
	`break`
	`case 2`
	`...`
	`break`
	`endSwitch`
		TEST X
		CASE 1 :
		...
		CASE 2 :
		...
		ENDTEST
			SELECT R[...] =1,
			CALL ...
			=2,
			CALL ...	

Tant que (...) faire
...
Fin tant que
	`while(...)`
	`...`
	`endwhile`
		WHILE (...) DO
		...
		ENDWHILE
			LBL[1] ;
			IF $\overline{(...)}$, JMP LBL[2] ;
			...
			JMP LBL[1] ;
			LBL[2] ;
				Boucle quand l'expression (...) est vraie

Faire
...
Jusqu'à (...)
	`do`
	`...`
	`until(...)`
		retour
		...
		IF $\overline{(...)}$ GOTO retour
			LBL[1] ;
			...
			IF $\overline{(...)}$, JMP LBL[1] ;	

7.2 Structure des fichiers, parallélisme, dialogue avec l'opérateur

Les 4 marques de robot ont des stratégies totalement différentes pour ranger les variables et les programmes d'une application. VAL3 permet de faire coexister sur un même robot plusieurs applications différentes en même temps. C'est un critère important pour un robot destiné à l'enseignement, pas pour un robot dédié à un processus de fabrication unique.

VAL3	TP	RAPID
Les programmes et les variables sont « encapsulés » dans une application. Le parallélisme des programmes et le dialogue avec l'opérateur sont construits librement par des instructions du langage VAL3. La sécurité (collision, zones de travail) est gérée par les programmes et n'entraîne pas l'arrêt de ceux-ci.	Les variables sont initialisées dans les registres qui sont partagés entre toutes les applications. Les programmes sont regroupés par leur seul nom. Le dialogue avec l'opérateur est géré par une page html servant de pupitre virtuel. Le parallélisme n'a pas été étudié.	Le parallélisme est fixé à priori par la configuration du contrôleur. Chaque tâche a son programme composé de modules composés de routines. Les variables sont déclarées dans les modules de chaque programme. Le dialogue avec l'opérateur est géré par un pupitre virtuel créé par ScreenMaker. La sécurité est gérée par des routines d'interruption et de zones de travail.
application PARALLELE - variables - programmes : -- start() -- mouvement() -- mvt_prise() -- dialogue() -- communication() -- securite() -- ... -- stop() application INTERMED - variables - programmes : -- start() -- ...	registre R[ ] registre PR[ ] AVANCE AVANCE_INIT AVANCE_PRISE ... INTER INTER_INIT INTER_PRISE	Tache n°1 programme MOUVEMENT - module INIT -- variables - module PRINCIPAL -- routine Main() -- trap routine Collision() - module WORLDZONES -- routine Worldzones() - module PRISE -- routine Mprise() Tâche n°2 programme DIALOGUE - module DIALOGUE -- variables -- routine Main() -- routine Saisietache() Tâche n°3 programme COMMUNICATION - module COMMUNICATION -- variables -- routine Main()

POLYSCOPE
Les variables sont locales au programme en cours d'exécution ou communes à tous les programmes (variables d'installation). La sécurité est intrinsèque, la sensibilité étant réglée par les paramètres du contrôleur et toute collision déclenche l'arrêt du programme. Aucun dialogue n'est prévu avec l'opérateur, hormis le démarrage, l'arrêt du programme, et des fenêtre pop-up d'informations.

7.3 Gestion des erreurs

La qualité d'un programme robot se mesurera en production, à sa robustesse face aux erreurs. En cas de défaut (collision, mauvaise saisie opérateur, ouverture porte), il ne faut pas « planter » le programme robot et nécessiter le redémarrage complet de la machine.

Les quatre langages de programmation permettent d'atteindre cet objectif.

VAL3 laisse toute latitude au programmeur pour adopter la stratégie voulue aussi bien dans la réaction du bras que dans le dialogue avec l'automate (programme securite).

Rapid permet la gestion des erreurs à l'aide des routines d'interruption. Cependant, les erreurs dues aux zones de travail ne peuvent pas être récupérées. Elles apparaissent principalement dans la phase de mise au point ou en mode manuel et ce n'est pas gênant.

En langage TP, nous n'avons pas encore testé de stratégie.

7.4 Mise au point des trajectoires, apprentissage

Lors de la mise au point de la cellule, le programmeur du robot doit ajuster les trajectoires en fonction de la partie mécanique de la cellule (point de prise, de pose, de contournement). Cela demande la mesure précise de la position et de l'orientation des repères utilisateurs, des dimensions et de l'orientation de l'outil.

Les quatre langages de programmation offrent la possibilité de mesurer ces repères par apprentissage. Certains processus de fabrication (soudage ou collage sur de grosses pièces) demandent que cette opération soit faite à chaque changement de pièce.

Les repères obtenus lors de l'apprentissage peuvent être utilisés directement par le programme. Mais en ce qui nous concerne, pour des applications de "pick and place", nous préférons reporter leurs coordonnées dans l'initialisation des variables de manière à regrouper tous les paramètres du programme au même endroit.

7.5 Habitudes et standard de l'entreprise

Chaque langage offre de nombreuses possibilités pour répondre à tous les besoins. Seule son étude approfondie permet d'en connaître toutes les ficelles. Le meilleur langage reste celui qui est maîtrisé par l'équipe qui conçoit et réalise la cellule robotisée.

Le client, utilisateur de la machine, n'a pas intérêt à multiplier le nombre de robots différents dans son usine. C'est souvent lui qui imposera le choix de la marque et donc du langage.

8 Annexes

8.1 Vidéos de présentation des porteurs

VIDEOS	
LRMATE	S3
RS40	S7

8.2 Programmes VAL 3 pour RS40 et vidéos

LISTINGS	VIDEOS
PROGRAMME MINIMUM PROGRAMME INTERMED PROGRAMME PARALLELE	RS40 1 MINIMUM RS40 2 INTERMED RS40 3 AVANCE RS40 4 COLLISION

8.3 Programmes TP pour LRMate et vidéos

LISTINGS	VIDEOS
PROGRAMME INTER PROGRAMME AVANCE	LRMATE 1 INTER LRMATE 2 AVANCE LRMATE 3 CONFIGURATIONS

8.4 Programmes Rapid pour IRB 120 et vidéos

LISTINGS	VIDEOS
PROGRAMME RAPID	IRB120 1 RAPID IRB120 2 COLLISION

8.5 Programmes PolyScope pour UR5 et vidéos

LISTINGS	VIDEOS
PROGRAMME PALETTISATION	UR5 1 PALETTISATION

9 Bibliographie

FANUC ACADEMIE « LA ROBOTIQUE INDUSTRIELLE » HACHETTE TECHNIQUE

MANUELS ABB :

RAPID Reference Manual : RAPID Overview On-line

RAPID Reference Manual : System Data Types and Routines On-line

MANUELS STAUBLI :

MANUEL DE REFERENCE VAL3

EXAMPLE APPLICATION