A.P.I. Kuka

Manuel de l'utilisateur

Version 0.1.0

Paul Chavent

Kubrick

12 février 2004

Table des matières

1 Utilisation sur le robot (serveur)

Cette première partie vous explique comment utiliser le serveur. Le serveur est un programme d'écoute qui répondra aux requettes des clients.
Ces actions s'opèrent sur le PC du robot Kuka.

1.1 Installation

Première étape : installer le programme. Notez que le moyen le plus simple est d'utiliser les binaires fournis sous forme "d'auto-installeur".

Télécharger :

- Les binaires disponibles à :
  http ://nestor.imerir.com/kuka/kukaApi/bin/kuka_api_serveur-0.1.0 -i386-1.exe (auto-installeur)
  ou
  http ://nestor.imerir.com/kuka/kukaApi/bin/kuka_api_serveur-0.1.0 -i386-1.tgz
- Les sources disponibles à :
  http ://nestor.imerir.com/kuka/kukaApi/src/kuka_api_serveur-0.1.0 .tar.gz

Compiler :

Attention : cette étape n'est pas indispensable. Il est recommandé pour la plus part d'entre vous de passer directement à l'installation des binaires. Si vous avez choisis d'utiliser directement les binaires passer à la section **Installer**, sinon :

1. Téléchargez http ://nestor.imerir.com/kuka/kukaApi/src/oncrpc_1.12.zip, décompressez l'archive, et exécutez le fichier `Setup.exe`. Vous pouvez choisir d'installer dans le répertoire proposé par défaut (`Program Files`), et d'utiliser le type d'installation `Developper`.
2. Décompressez l'archive kuka_api_serveur-0.1.0 .tar.gz dans un répertoire de votre choix (avec winzip par exemple).
3. Compiler en lançant le script adapté à votre environnement de développement (make_vc.bat pour Ms Visual, ou make_bc.bat pour Borland).
   Attention, il vous faudra certainement modifier les fichiers `Makefile` suivant où vous avez installés les fichiers du packetage oncrpc.

Installer :

Si vous avez téléchargé l'auto-installeur :

1. double cliquez sur kuka_api_serveur-0.1.0 -1.exe et suivez les instructions.

ou si vous avez téléchargé l'archive :

1. décompressez l'archive kuka_api_serveur-0.1.0 .tar.gz dans un répertoire de votre choix (avec winzip par exemple).
2. copier `pm_ascii.exe` (le portmaper pour rpc) `kukaApi.exe` (le serveur de l'API) sur votre disque dur (celui du PC du robot),
3. copier le fichier `oncrpc.dll` dans votre `PATH` (c :/windows/system par exemple),
4. copier le programme de commande[1] (kuka_api.src et kuka_api.dat) dans le répertoire des programmes du robot (Par exemple `c :\KRC\ROBOTER\KRC\R1\Program`).
   Note : le programme de commande fourni dans l'archive est minimaliste et vous pourrez le modifier ou en faire d'autre plus adaptés à vos besoins. Toutefois, dans la suite des explications, nos exemples s'appuieront sur lui,

1.2 Configuration

Pour rendre utilisable le serveur il suffit de déclarer les variables que nous aurons besoin de partager sur le réseau de manière globale. Voici la démarche à suivre :

1. Si le KukaBof est lancé, vérifiez d'`abandoner` l'exécution du programme Submit. Ainsi vous pourrez éditer le fichier `$config.dat`.
2. Sauvegarder `C :\KRC\ROBOTER\KRC\R1\SYSTEM\$config.dat` dans `C :\KRC\ROBOTER\KRC\R1\SYSTEM\$config.dat.bak`.
3. Ouvrir `C :\KRC\ROBOTER\KRC\R1\PROGRAM\kuka_api\kuka_api.dat`.
4. Ouvrir `C :\KRC\ROBOTER\KRC\R1\SYSTEM\$config.dat`.
5. Copier le FOLD “Section a copier dans $config.dat” du fichier `kuka_api.dat` dans le fold “USER GLOBALS” du fichier `$config.dat` en dessous de “Userdefined Variables”, et décomenter cette section.
6. Sauver `$config.dat`.
7. Relancer le programme Submit (`sub.sps`).
8. Redémarrer la machine.

[1] Voir la description du programme de commande page 16.

Note : Lors du redémarage si le BOF vous signale une erreur dans l'en-tête du fichier uninstall.dat, ce n'est pas grave.

1.3 Utilisation

Le serveur fonctionne avec trois programmmes :

1. le programme qui permet d'écouter les requêtes (le portmapper rpc),
2. le programme qui implémente les requêtes,
3. le programme de commande qui actionne le robot.

Voici la marche à suivre pour lancer ces programmes.

Lancement :

1. Dans Windows® :
   (a) Menu démarrer programmes kuka_api kuka_serveur.exe

   ou :

   (a) Lancer port_map.exe (portmapper pour les rpc).
   (b) Lancer kuka_api_serveur.exe
2. Dans le BOF :
   (a) Vérifiez au préalable d'être en mode automatique ($MODE_OP=#AUT) , d'avoir fermé le portillon du park de kuka et d'avoir acquitté tous les défauts.

   (b) Sélectionner le programme de commande (kuka_api.src pour nos exemples).
   (c) Enclencher la puissance.

   (d) Amener le robot au COI (position home) avec la touche start+ du KCP.

   (e) Appuyez à nouveau sur la touche start+ afin de lancer la boucle d'attente.

Le serveur est désormais prêt à recevoir des ordres.

Arrêt :

1. Dans le BOF :
   (a) Arrêter le programme de commande à l'aide de la touche stop.

   (b) Mettre les moteurs hors puissance.

2. Dans Windows® :
   (a) Terminer kuka_api_serveur.exe puis pm_ascii.exe.

2 Utilisation sur le pc distant (client)

Cette partie décrit les fonctions que l'API met à votre disposition. A la fin de cette partie vous aurez construit, à travers des exemples simples, un programme minimaliste pour faire fonctionner le robot à distance.
Si vous recherchez des exemples d'utilisation avancé, passez à la partie suivante.

2.1 Installation

Les étapes décritent ci-dessous sont effectuées sur un PC distant. A ce niveau, vous avez le choix d'utiliser Windows® ou Linux. Voici la démarche à suivre suivant votre système.

2.1.1 Windows®

A partir des sources :

Attention, sous Windows®, la compilation de la doc n'a pas été testé et ne le sera jamais. Vous pourrez donc récupérer toutes la documentation en ligne (http ://www.imerir.com/kuka) ou dans les archives des binaires.

Les étapes suivantes décrivent uniquement la compilation de la librairie.

1. Téléchargez http ://nestor.imerir.com/kuka/kukaApi/src/oncrpc_1.12.zip, décompressez l'archive, et exécutez le fichier `Setup.exe`. Vous pouvez choisir d'installer dans le répertoire proposé par défaut (`Program Files`), et d'utiliser le type d'installation `Developper`.
2. Télécharger les sources disponibles sur
   http ://nestor.imerir.com/kuka/kukaApi/src/kuka_api_client-0.1.0 .tar.gz
3. Décompresser l'archive kuka_api_client-0.1.0 .tar.gz dans un répertoire de votre choix (avec winzip par exemple).
4. Dans ce répertoire, aller dans le dossier des sources :
   ```
   #> cd src\win
   ```
5. Compiler avec le script adapté à votre environnement de développement. Par exemple avec les outils Microsoft® taper :
   ```
   #\src\win> nmake -f Makefile.msvc install
   ```

Pour les exemples, nous vous laissons libre de les compiler avec le moyen que vous voudrez.

A partir des binaires :

1. Télécharger les binaires disponibles sur :
   http ://nestor.imerir.com/kuka/kukaApi/bin/kuka_api_client-0.1.0 -i386-1-w32.tgz
2. Installer : copier le fichier kuka_api.h dans le dossier où vous rangez vos en-têtes, et copier le fichier kuka.lib dans le dossier où vous rangez vos librairies.
3. Tester : lancer un test en exécutant :
   ```
   # test_api kuka
   ```

2.1.2 Linux

A partir des sources :

1. Télécharger les sources disponibles sur
   http ://nestor.imerir.com/kuka/kukaApi/src/kuka_api_client-0.1.0 .tar.gz
2. Décompressez l'archive, par exemple :
   ```
   :$ tar -zxvf kuka_api_client-0.1.0 .tar.gz /usr/local/src
   ```
3. Compiler :
   ```
   :$ cd /usr/local/src/kuka_api_client
   :$ make
   ```
4. Installer :
   ```
   :$ make install
   ```

A partir des binaires :

1. Télécharger les binaires disponibles sur :
   http ://nestor.imerir.com/kuka/kukaApi/bin/kuka_api_client-0.1.0 -i389-1.tgz
2. Installer :
   - Si vous utilisez une slackware :
     ```
     :$ installpkg kuka_api_client-0.1.0 -i386-1.tgz
     ```
   - Sinon, décompréssez l'archive, par exemple :
     ```
     :$ tar -zxvf kuka_api_client-0.1.0 -i386-1.tgz
     ```

2.2 Utilisation

Nous allons décrire ici les fonctionnalités de l'api [2].

Bien sûr, pour pouvoir utiliser l'A.P.I. vous devez avant toutes choses inclure dans votre source l'entête.

```
#include ''kuka_api.h'';
```

Pour plus de détails voir http ://www.imerir.com/kuka/kukaApi/doc/kuka_api_man_ref.

2.2.1 Initialisation

Pour pouvoir faire appel aux fonctions de l'api vous devez initialiser la connection au robot. De même, avant de quitter le programme vous devez stopper la connection. Vous disposez pour cela de deux fonctions :

```
int kuka_initialize(char *serveur);
int kuka_uninitialize(void);
```

ATTENTION : l'API est prévue pour etre utilisée que depuis un seul client. Ceci se traduit par :

- Lors du premier appel à `kuka_initialize` le client a ouvert une session sur kuka.
- Lors des appels suivant à `kuka_initialize` les autres clients utiliseront la session ouverte par le premier.
- Lors du premier appel à `kuka_uninitialize` le client ferme la session. Plus aucun client ne peut communiquer avec kuka.
- Les appels suivant à `kuka_uninitialize` seront sans effet.

Ceci a été conçu de cette facon dans un souci de simplification. Les implémentrations futures pourraient gérer plusieurs connections. Pour plus de détails voir http ://www.imerir.com/kuka/kukaApi/doc/kuka_api_man_ref.

2.2.2 Définition des variables

Le programme de commande[3], qui tourne sur Kuka, contrôle un certain nombre de variables destinées à commander le robot.

[2] Le manuel de référence de l'api décrit en détail les fonctions (paramètres, retours), les structures, les types, etc.

[3] Voir la description du programme de commande page 16.

Par exemple, nous pourrions[4] avoir à notre disposition les variables suivantes :

Nom	Type	Descritpion
CIBLE	POS	Contrôle la position de la pince (en mm)
CIBLE_FLAG	BOOL	
PINCE	BOOL	Contrôle l'état (ouvert=true/fermé=false) de la pince
PINCE_FLAG	BOOL	
BASE	FRAME	Contrôle la base de travail (en mm)
BASE_FLAG	BOOL	
VITESSE	INT	Contrôle la vitesse (en pourcentage)
VITESSE_FLAG	BOOL	

Pour utiliser ces variables sur le poste client il suffit de les déclarer étant de type `kukaVar_t`.

```
kukaVar_t cible={"CIBLE",{KUKA_POS,{0}}};
kukaVar_t cibleFlag={"CIBLE_FLAG",{KUKA_BOOL,{1}}};
```

Note : les unités dans lesquelles nous manipulerons ces variables est la même que celles du KRL.

Pour plus de détails voir http ://www.imerir.com/kuka/kukaApi/doc/kuka_api_man_ref.

2.2.3 Lecture/Écriture des variables

Les variables peuvent être lues et écrites. A cette fin nous disposons de deux fonctions :

```
int kuka_setVar(kukaVar_t *variable);
int kuka_getVar(kukaVar_t *variable);
```

Pour afficher le contenu d'une variable sur la sortie standard (`stdout`) nous avons :

```
void kuka_displayVar(kukaVar_t *variable);
```

Pour plus de détails voir http ://www.imerir.com/kuka/kukaApi/doc/kuka_api_man_ref.

[4]Ce sont effectivement les variables qui sont à notre disposition si c'est le programme kuka_api.src qui est chargé dans le robot

2.2.4 Gestion des erreurs

Enfin, comme une commande peut parfois échouer, vous disposez d'une variable qui contient la dernière erreur. Vous pouvez obtenir un pointeur sur cette variable en invoquant la fonction :

```
void kuka_getError(kukaVar_t *kukaError);
```

Pour plus de détails voir http ://www.imerir.com/kuka/kukaApi/doc/kuka_api_man_ref.

2.2.5 Exemple

Pour atteindre une nouvelle position, vous pouvez suivre l'exemple suivant :

```
#include ''kuka_api.h''
/* Declarations */
kukaVar_t *kukaError;
kukaVar_t cible={"CIBLE",{KUKA_POS,{0}}};
kukaVar_t cibleFlag={"CIBLE_FLAG",{KUKA_BOOL,{1}}};
/* Initialisation du pointeur sur les erreurs */
kuka_getError(&kukaError);
/* Initialisation des valeurs des variables */
cible.valeur.kukaVal_u.kukaPos.x=(kukaReal_t)(10); // milimetre
cible.valeur.kukaVal_u.kukaPos.y=(kukaReal_t)(60); // mm
cible.valeur.kukaVal_u.kukaPos.z=(kukaReal_t)(110); // mm
/* Initialisation de l'api */
if( kuka_initialize("kuka") ){
    printf("Erreur a l initialisation.\n");
    kuka_displayVar( kukaError );
    exit(1);
}
/* Mise-a-jour de la valeur sur kuka */
if( kuka_setVar(&cible) || kuka_setVar(&cibleFlag) ){
  printf("Erreur, la cible ne sera pas atteinte.\n");
  kuka_displayVar( kukaError );
  /* Traitement des erreurs */
  [...]
}
/* Desinitialisation de l'api */
if( kuka_uninitialize() ){
  printf("Erreur a la desinitialisation.\n");
  kuka_displayVar( kukaError );
}
/* Fin du programme */
[...]
```

2.2.6 Astuces

Aussi, pour simplifier la programmation vous pouvez déclarer des macros :

– Pour permettre la mise à jour de la variable et de son flag rapidement :

```
#define SET(x) kuka_setVar(x) || kuka_setVar(x##Flag)
```

– Pour accéder plus facilement aux membres des variables :

```
#define CIBLE cible.valeur.kukaVar_u.kukaPos
#define OUVRE(x) x.valeur.kukaVar_u.kukaBool=(kukaBool_t)1
#define FERME(x) x.valeur.kukaVar_u.kukaBool=(kukaBool_t)0
```

Nous pouvons alors utiliser ces macros comme suit :

```
CIBLE.x=32;
OUVRE(pince);
SET(pince);
SET(cible);
```

Attention : remarquez que dans ce cas les macros prennent en “paramètre” le nom de la variable alors que les fonctions de l’API prennent en paramètre un pointeur.

Dans la suite nous choisirons d’appeler la structure de la variable par `nomDeLaVariable`, et d’appeler la valeur de la variable par `NOMDELAVARIABLE`.

Bien sûr, ces conseil sont donnés à titre d’exemple et il n’est pas impératif de les suivre pour faire fonctionner l’A.P.I.

2.2.7 Avertissement

ATTENTION : si certaines règles ne sont pas suivies, la trajectoire programmée peut ne pas être respectée.

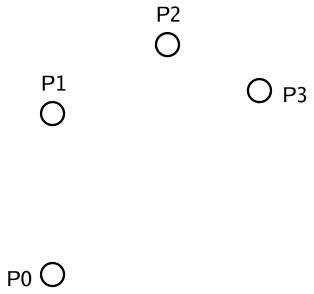
Soit le programme

```
1  kuka_getVar(&position);
2  CIBLE.y=POSITION.y+7;
3  kuka_setVar(&cible);
4  kuka_setVar(&cibleFlag);
5  kuka_getVar(&position);
6  CIBLE.x=POSITION.x+5;
7  CIBLE.y=POSITION.y+3;
8  kuka_setVar(&cible);
9  kuka_setVar(&cibleFlag);
10 kuka_getVar(&position);
11 CIBLE.x=POSITION.x+4;
12 CIBLE.y=POSITION.y-3;
13 kuka_setVar(&cible);
14 kuka_setVar(&cibleFlag);
```

Les déplacements devraient être :

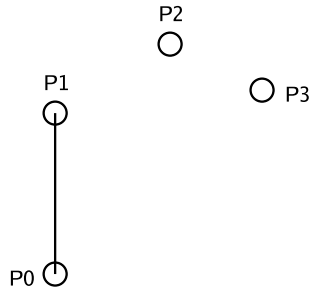
- 7mm en y,
- 5mm en x et 3mm en y,
- 4mm en x et -3mm en y.

La trajectoire devrait être :

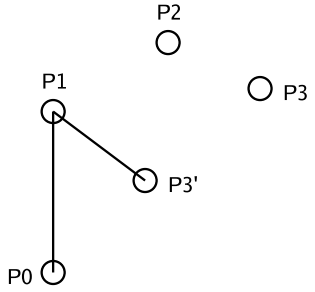


Commentons le déroulement des opérations :

1. A l'instruction **1**, le robot est en P0.
2. A l'instruction **4**, le robot part vers P1.
3. Au môment où le robot arrive en P1, il libère le flag de la cible (**CIBLE_FLAG=FALSE**) et se met en attente d'ordre nouveaux. Alors,plusieurs cas se présentent à nous.
   - le programme est en **14**, donc terminé. Le robot n'aura pas exécuté les dernières instructions et nous obtenons la trajectoire suivante :



– le programme est en **10** et continue de s'exécuter. Il récupère les coordonnées du point qu'il croit être P2, mais comme le robot est en P1, le déplacement relatif fausse la trajectoire et nous emmène en P3' comme l'illustre la figure ci-dessous :



L'objectif n'est pas atteint.

Nous proposons alors deux nouveaux programmes :

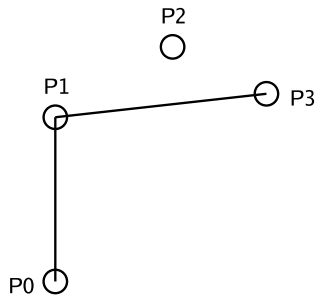
Déplacements relatifs sans mise à jour de la position actuelle :

```
1  kuka_getVar(&position);
2  CIBLE.y+=7;
3  kuka_setVar(&cible);
4  kuka_setVar(&cibleFlag);
5  CIBLE.x+=5;
6  CIBLE.y+=3;
7  kuka_setVar(&cible);
8  kuka_setVar(&cibleFlag);
9  CIBLE.x+=4;
10 CIBLE.y-=3;
11 kuka_setVar(&cible);
12 kuka_setVar(&cibleFlag);
```

Déplacements absoluts :

```
1 kuka_setVar(&P1);
2 kuka_setVar(&cibleFlag);
3 kuka_setVar(&P2);
4 kuka_setVar(&cibleFlag);
5 kuka_setVar(&P3);
6 kuka_setVar(&cibleFlag);
```

Résultat :

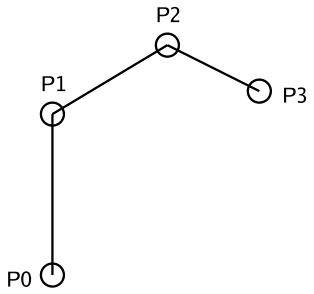


Nous atteignons bien P3, mais sans passer par P2 : ce n'est toujours pas la bonne solution.

La solution que nous vous suggérons pour éviter ces problèmes est la suivante :

```
1  kuka_getVar(&position);
2  CIBLE.y+=7;
3  kuka_setVar(&cible);
4  kuka_setVar(&cibleFlag);
5  while(CIBLEFLAG){kuka_getVar(&cibleFlag);} CIBLEFLAG=1;
6  CIBLE.x+=5;
7  CIBLE.y+=3;
8  kuka_setVar(&cible);
9  kuka_setVar(&cibleFlag);
10 while(CIBLEFLAG){kuka_getVar(&cibleFlag);} CIBLEFLAG=1;
11 CIBLE.x+=4;
12 CIBLE.y-=3;
13 kuka_setVar(&cible);
14 kuka_setVar(&cibleFlag);
15 while(CIBLEFLAG){kuka_getVar(&cibleFlag);} CIBLEFLAG=1;
```

Les instructions **5**, **10** et **15**, attendent que le flag soit remis à zéro par le programme de commande (ce qui signifie que l'action déclenchée précédemment est finie). Nous obtenons alors cette trajectoire.



3 Exemple : le jeu de dame

Cet exemple illustre une des utilisations possible de l'API. Vous serez guidé à travers la réalisation du plateau de jeu et de son interface.

3.1 Introduction

Un jeu de plateau, comme les dames, consiste à déplacer des pions sur une grille. Pour des applications d'I.A., par exemple, il serait intéressant de pouvoir communiquer au robot des coordonnées de la forme :

`C5` (repère indexé d'un damier),

plutôt que de la forme :

`{X 468.15, Y 206.98, Z 900, A 0.02, B 89.1, C 0.002}` (repère world du robot).

La conversion sera réalisée par une interface. L'implémentation de cette interface interprète ces coordonnées, les transforme en coordonnées cartésiennes et les transmet à Kuka via l'A.P.I. .

3.2 Le programme de commande : description

Voici quelques caractéristiques de notre programme de commande (kuka_api.src) :
- il est écrit en krl (KukaRobotLanguage),
- il est chargé dans le robot,
- il attend et reçoit un signal puis exécute un ordre associé à ce signal.

Le programme de commande livré avec l'api (kuka_api.src) boucle indéfiniment en attendant un signal que nous nommerons **flag**. Chaque **flag** est associé à une action qui est exécutée si il est activé (valeur `TRUE`).

Pour l'instant le modèle implémente les actions suivantes :
- déplacement point à point (PTP) contrôlé par les variables CIBLE et CIBLE_FLAG,
- déplacement circulaire et linéaire, lissés ou non, par les variables POINTS, APPUIS, ANGLES, TYPES, LISSAGES, DEPLACE et DEPLACE_FLAG,
- ouverture/fermeture de la pince contrôlé par les variables PINCE et PINCE_FLAG,
- changement de base contrôlé par les variables BASE et BASE_FLAG,
- changement de vitesse contrôlé par les variables VITESSE et VITESSE_FLAG.

Enfin, quelques conseils si vous voulez changer/créer un programme de commande :

– La norme de dénomination des flags est la suivante :
`nomdevariable_FLAG`, où `nomdevariable` correspond à la variable associée à l'action.
– L'implémentation du programme de commande doit désactiver le flag (valeur `FALSE`) d'une action qui se termine.
– Le nom des variables ne devraient pas dépasser 12 caractères.
– Les implémentations d'attente de signaux sont variées : boucle (`LOOP` ou `WHILE`), interruptions (`INTERRUPT`), attente (`WAIT`), et peut-être d'autres.

3.3 Préparer le plateau de jeu

Vous devriez connaître les contraintes imposées par le robot afin de construire le plateaux de jeux en conséquence. Nous pouvons notamment citer qu'il faut prendre en compte :

Les positions singulières En effet certains points ne sont pas atteignables avec certaines configurations du robot[5].

L'encombrement maximum de la pince Celui-ci fait **80x80mm** pour le robot d'IMERIR, ce qui implique qu'il faut faire des cases d'une largeur maximum de 80mm.

La capacité de préemption de la pince Celle-ci fait **40mm** pour le robot d'IMERIR, ce qui implique qu'il faut des pièces d'une épaisseur maximum de 40mm.

La capacité de charge du robot Le robot d'IMERIR peut porter au maximum **6Kg** à l'extrémité de son bras..

Les obstacles qui peuvent nuire à l'évolution du robot. Imaginons que le robot ait une pièce encombrante, il faudrait vérifier si la pièce ne risque pas d'accrocher **la barrière de sécurité** du parc de kuka.

L'encombrement des pions (longeurxlargeurxhauteur). De leur hauteur, par exemple, dépendra l'amplitude du mouvement minimum pour lever la pièce et la déplacer au dessus des autres sans les renverser.

La configuration initiale des pions.

Les actions et les types de mouvements . Nous pouvons imaginer qu'il soit nécessaire de faire parcourir à la pince une trajectoire **circulaire**. Il faudra que le programme de commande implémente ces déplacements. Note : à partir de la version 0.0.5, le programme de commande par défaut permet de gérer des trajectoires complexes avec plusieurs points, du lissage, et des mouvements linéaires, circulaires ou point à point.

[5] Voir la documentation de kuka programmation expert kuka prog_r23_5_5.pdf

3.4 Implémenter une interface

Notre interface doit nous permettre de déplacer la pince sur un damier, de poser et de prendre des pions. Nous allons en premier lieu nous intéresser aux déplacements.

3.4.1 Les déplacements

Nous sommes à la frontière entre un monde qui ne comprend que des coordonnées sous forme de cases, et un autre qui ne les comprend que sous forme de "sextuples" (position et orientation).

ALGO

Notre but est donc de réaliser une simple conversion et nous avons plusieurs moyens d'y parvenir.

1. Nous pouvons créer une table qui à chaque case du plateau de jeux fait correspondre des coordonnées cartésiennes. Par exemple :

Case	X=	Y=	Z=
1	825.94	655.5	900.42
2	825.94	655.5	900.42

   Pour récupérer ces coordonnées, il suffit par exemple de :

   (a) amener le bras du robot au dessus d'une pièce (en mode manuel),

   (b) noter la position grâce au menu visualiser→position actuelle→cartésienne du BOF,

   (c) reporter dans la table les coordonnées correspondantes au point.

   Cette solution à l'avantage de pouvoir s'adapter à un plateau dont les espacements entre les cases seraient irréguliers, comme un jeu de l'oie par exemple.

2. Si le plateau est régulier nous pouvons utiliser un système de conversion qui calcule les coordonnées "dynamiquement". Il suffit de connaître la largeur et la longueur des cases. Ainsi la coordonnée de la case C4 est obtenue en calculant :

```
X <- 3 x largeur_de_case
Y <- 4 x hauteur_de_case
```

Nous obtenons donc des coordonnées **relatives à l'origine du damier**. Mais avant cela il aura fallu changer de repère (cf. 3.4.2 : les changements de base), car par défaut, nous travaillons dans le repère world du robot.

Cette solution à l'avantage de pouvoir être très rapidement adaptable à une nouveau plateau de jeu.

CODE

Ceci se traduit finalement par :

```
/* En-tetes */
#include ''kukaApi.h''
[...]
/* Constantes */
const float largeur_de_case=80;
const float hauteur_de_case=80;
[...]
/* Declarations */
kukaVar_t cible={"CIBLE",{KUKA_POS,{0}}};
#define CIBLE cible.valeur.kukaVal_u.kukaPos
kukaVar_t cibleFlag={"CIBLE_FLAG",{KUKA_BOOL,{1}}};
[...]
/*! Algo 2 : la fonction retourne 0 si elle n a pas generee d erreur */
int deplaceSur(int alpha, int num){
  /* Nous ajustons la cible a atteindre */
  CIBLE.x=alpha*largeur_de_case;
  CIBLE.y=num*hauteur_de_case;
  /* Nous demandons a kuka d executer la commande */
  return kuka_setVar(&cible) || kuka_setVar(&cible_flag);
}
[...]
```

3.4.2 Changement de base

Les changements de base permettent une manipulation de coordonnées plus adaptée au contexte. Nous avons vu par exemple que c'est pratique d'avoir l'origine du repère de travail à l'origine du plateau de jeu. Pour ce faire il faut récupérer les paramètres qui caractérisent une base. Ces paramètres sont de type FRAME, c'est à dire une structure contenant les coordonnées du centre de la base, et celles de l'orientation de la base. Vous devez vous référer au manuel d'utilisateur du robot pour définir une nouvelle base.
Vous devrez alors initialiser une variable avec les valeurs cette nouvelle base.

```
/* En-tetes */
#include ''kukaApi.h''
[...]
/* Constantes */
/* Declarations/Initialisation base */
kukaVar_t base={"BASE",{KUKA_FRAME,{0}}};
#define BASE cible.valeur.kukaVal_u.kukaFrame
kukaVar_t baseFlag={"BASE_FLAG",{KUKA_BOOL,{1}}};
BASE.x= 812.5909;
BASE.y= 435.7992;
BASE.z= 900.0000;
BASE.a=-88.30630;
BASE.b= 0.001254;
BASE.c=-0.002854;
[...]
```

3.4.3 Poser/Prendre

Cette action comprend la descente et la remontée de la pince, ainsi que sa fermeture/ouverture.

Monté Descente Ici il faut prendre en compte la hauteur des pièces afin de ne pas les renverser lors des déplacements.

Préemption Voila un listing de code modèle pour utiliser la pince.

```
/* En-tetes */
#include ''kukaApi.h''
[...]
/* Declarations */
kukaVar_t pince={"PINCE",{KUKA_BOOL,{0}}};
kukaVar_t pinceFlag={"PINCE_FLAG",{KUKA_BOOL,{1}}};
/* Macro pour ouvrir ... */
#define O(x) x.valeur.kukaVar_u.kukaBool=(kukaBool_t)1
/* ... et pour fermer. */
#define F(x) x.valeur.kukaVar_u.kukaBool=(kukaBool_t)0
[...]
```

4 Résolution de problèmes

4.1 Messages d'erreur

1. Erreur 2047

   ```
   nom    = ""
   type   = KUKA_ERROR
   valeur = { KUKA_KUKA_E , 2047, "kuka: you should call showVar with
              "&2047" as var name and KUKA_STRING as type to get a
              description."}
   ```

   Cette erreur arrive lorsque vous essayez d'obtenir la valeur d'une variable qui est inconnue du cross. Vous devriez alors vérifier d'avoir déclaré dans le fichier `$config.dat` toutes les variables auxquelles vous avez besoin d'accéder à distance. Voir section 1.2 page 4.