MIETLICKI Pascal
OTHMANI Jihed

TEMPS RÉEL
RAPPORT

Commande et simulation d'un bras manipulateur

INSA TOULOUSE

Temps réel - Rapport final

CONCEPTION ET SIMULATION D'UN BRAS MANIPULATEUR 2 AXES

1. INTRODUCTION

Afin de mettre en application les éléments de cours étudiés, il nous a été proposé de développer un logiciel temps réel permettant la gestion du mouvement d'un bras mécanique à deux axes.

L'élaboration de ce logiciel à pour but la familiarisation des étudiants avec les concepts et outils nécessaires au développement d'un système temps réel.

Nous pourrons ainsi réaliser l'étude du logiciel à partir de la décomposition fonctionnelle SART mise à notre disposition et de divers autres documents.

Le codage du logiciel se fera à l'aide du système temps réel VxWorks qui a comme support le langage C.

A travers la réalisation de ce projet, nous avons pu approfondir les concepts appris cette année et les appliquer au sein d'un travail concret.

2. CAHIER DES CHARGES

a. définition du projet

L'application doit permettre la gestion des mouvements d'un bras manipulateur 2 axes assurant le déplacement d'objets dans un plan. Ce bras est composé de 2 segments que nous appellerons « coude » et « épaule ». Il est équipé d'une pince permettant la saisie des objets.

La position angulaire de chaque segment est contrôlée par un moteur à courant continu équipé d'un capteur de position.

La commande de puissance du moteur est incluse dans le bras manipulateur.

b. réalisation

Un opérateur pilote le bras à partir d'un calculateur distant disposant d'une interface graphique. Cette dernière permet de saisir des commandes et de visualiser, en temps réel, l'état du système.

Cette interface permet à l'opérateur de saisir les ordres et de lui fournir un certain nombre de renseignements.

Les ordres sont :

-« REJOINDRE » : Permet de positionner le bras à une position angulaire pour chacun des deux axes avec une vitesse donnée.

Les deux articulations doivent atteindre leurs positions finales respectives simultanément.

-« FERMER » : Permet de fermer la pince afin de saisir l'objet.

-« OUVRIR » : Permet d'ouvrir la pince afin de libérer l'objet.

Les ordres ne pourront être exécutés que séquentiellement.

Il doit être possible d'effectuer un arrêt d'urgence. Cet arrêt d'urgence doit être possible à tout moment et doit provoquer l'arrêt immédiat du mouvement en cours. Les demandes opérateurs et celle du bras devront être conservées.

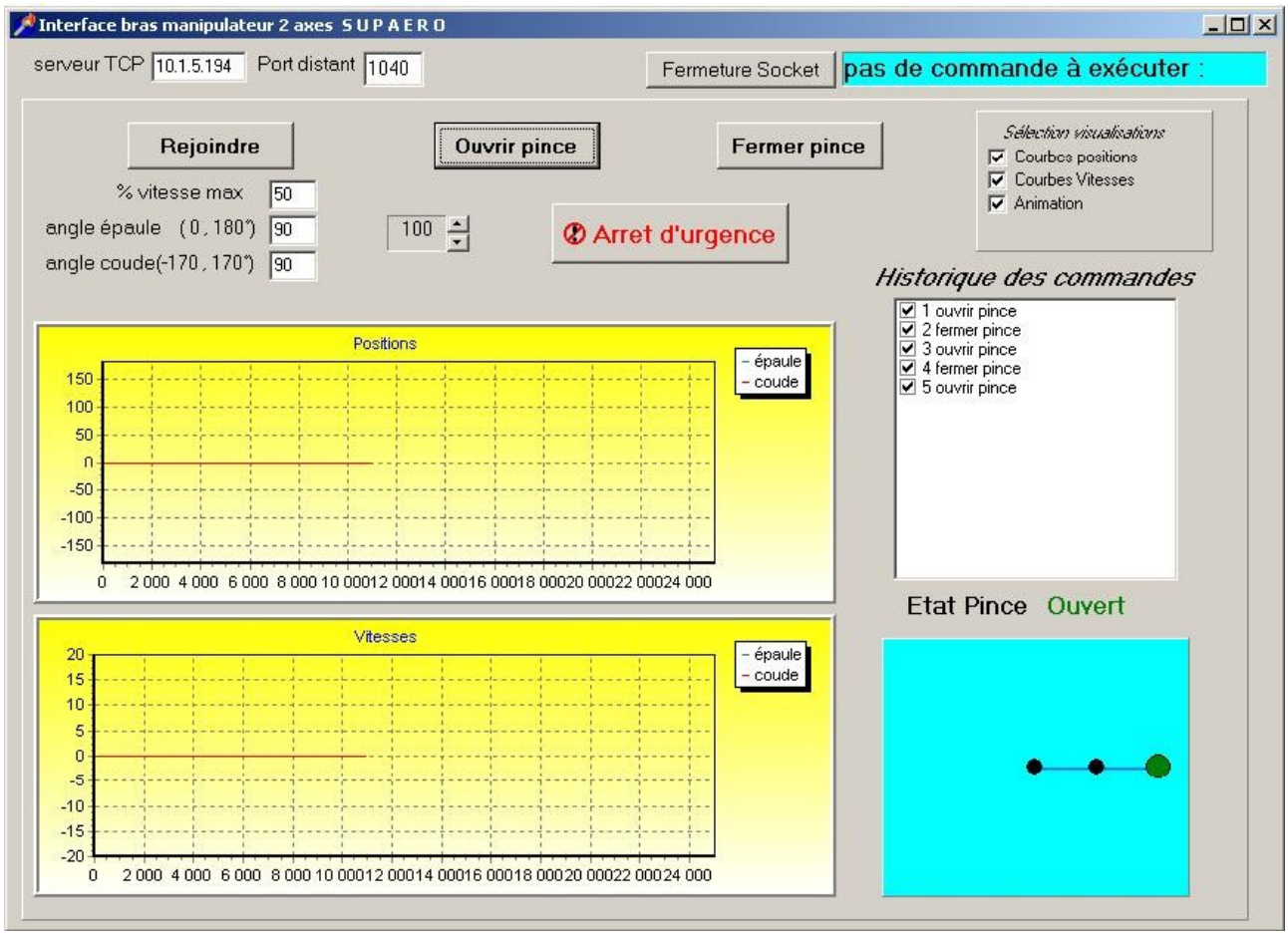


Image 1: Interface graphique pour la gestion du bras

3. CONCEPTION

a. méthodologie utilisée

Pour l'étude du découpage en tâches, nous avons utilisé le schéma de décomposition fonctionnel SART.

Celui-ci nous permet de définir les différentes tâches selon les critères de :

-cohésion fonctionnelle

-cohésion au niveau des ressources et données communes

-cohésion des contraintes temporelles.

c. diagramme globale

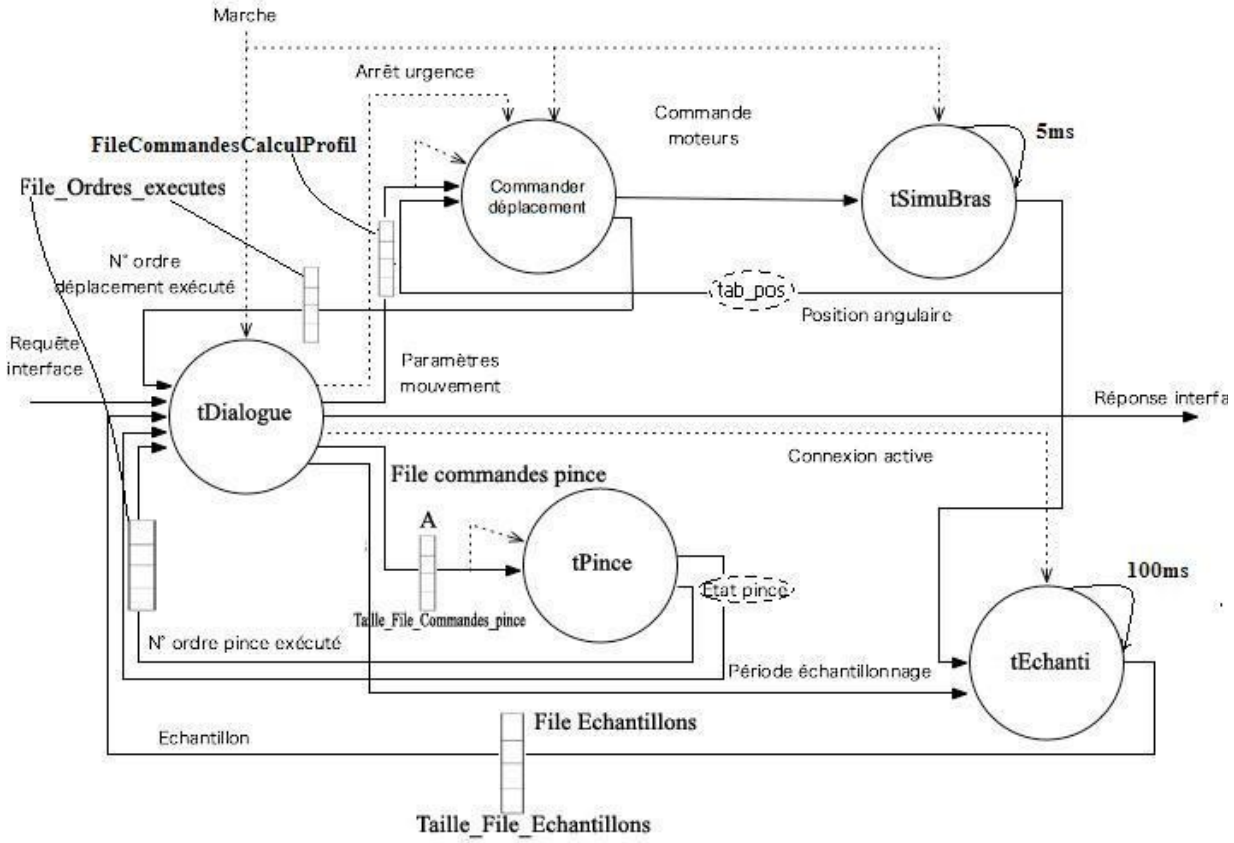
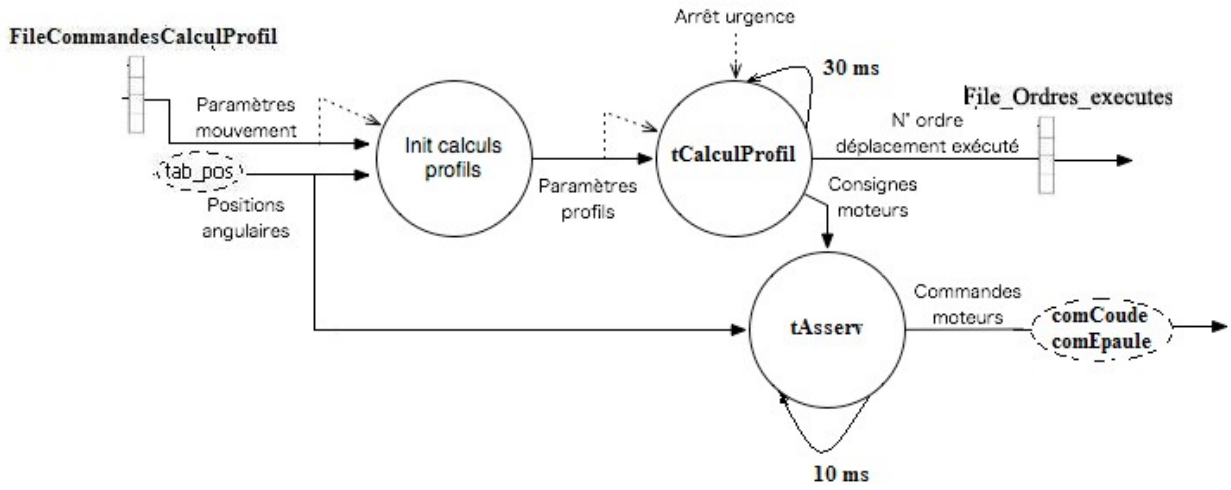


Image 2: Diagramme de conception globale

d. diagramme de décomposition du déplacement



e. découpage en tâches

Le code commenté de l'ensemble des tâches est joint. Toutefois, afin d'en faciliter la lecture, le code principal est donné ci-dessous pour les tâches principales.

- Tâche dialogue (tDialogue) :

Cette tâche gère l'interaction avec l'utilisateur par l'intermédiaire de l'interface graphique. C'est le « point d'entrée » de notre programme afin de traiter les actions demandés par l'utilisateur.

- Tâche pince (tPince) :

Cette tâche regroupe les fonctions de gestion pour la saisie de la pince.

```
void tPince( )
{
	int error;
	int status;
	struct commandePince cmd;
	while(arretUrgence!=TRUE)
	{
		/* Lecture du message */
		error = msgQReceive(idFileCommandesPince,(char*)&cmd,sizeof(struct commandePince),WAIT_FOREVER);
		if (error != sizeof(struct commandePince)){
			perror("Probleme de reception des messages dans la msgQ de la pince");
		}
		else{
			/* On verifie que l'on n'execute pas 2 fois la même commande d'affile */
			if(etat_pince == cmd.commande){
				printf("Pas d'action sur la pince\n");
			}
			else
			{
				/* execute l'action associe */
				etat_pince = cmd.commande;
				/*Pause pour simuler l'action*/
				taskDelay(sysClkRateGet());
			}
			if (status != OK)
				perror("Probleme semTake dans tPince\n");
			/* Ajout de la commande dans la liste des commandes effectuées (pour l'aquitement)  */
			error = msgQSend(idFileCommandesExecutees, (char*)&cmd.num, sizeof(int), WAIT_FOREVER,
MSG_PRI_NORMAL);
			if (error != OK)
				perror("Probleme d\'envoi des messages dans la msgQ de la pince");
			if (status != OK)
				perror("Probleme semGive dans tPince\n");
		}
	}
}
```

Le fonctionnement est relativement simple et explicite, on effectue l'action associé dans la file idFileCommandesPince puis on l'acquitte par l'intermédiaire de idFileCommandesExecutees.

- Tâche calcul (tCalculProfil):

Cette tâche permet l'initialisation ainsi que le calcul des positions intermédiaires.

```
void tCalculProfil( )
{
	int error;
	int status;
	int i=0;
	short finMouvement=0;
	struct commandeCalculProfil cmd;
	while(arretUrgence!=TRUE)
	{
		error = msgQReceive(idFileCommandesCalculProfil, (char*)&cmd, sizeof(struct
commandeCalculProfil),WAIT_FOREVER);
		if (error != sizeof(struct commandeCalculProfil))
		{
			perror("Probleme de reception des messages dans la msgQ de CalculProfil");
		}
		else {
			/* execute l'action associe */
			printf("Action sur CalculProfil(%c)\n",cmd.commande);
			semTake(idMutexPos, WAIT_FOREVER);
			initCalculsProfils(angleEpauleCourant, angleCoudeCourant, cmd.vitesse, cmd.epaule,
cmd.coude);
			semGive(idMutexPos);
			printf("\n Calcul profil do while");
			if (finMouvement!=1)
			{
				semTake(idMutexPos, WAIT_FOREVER);
				calculPositionIntermediaire (&angleInterEpaule, &angleInterCoude, &finMouvement);
				semGive(idMutexPos);
			}
		}

	}
	printf("\nArret Urgence!!!!!!");
	exit(1);
}
```

Cette tâche récupére les commandes à effectuer par l'intermédiaire de la file « idFileCommandesCalculProfil ». Elle initialise le calcul grâce à la commande « initCalculsProfils » puis calcule les positions intermédiaires jusqu'à ce que le mouvement soit terminé (en vérifiant la valeur de finMouvement).

- Tâche de simulation du bras (tSimuBras) :

Cette tâche permet d'effectuer la simulation du bras au niveau de l'interface graphique.

```
int tSimuBras() {

float inter1, inter2;
initSimulationEpaule(0);
initSimulationEpaule(0);

while (arretUrgence!=TRUE) {
		semTake(idSemSimu, WAIT_FOREVER);
		/* calculer la position courante du bras par rapport au vitesse du moteur  toutes les 5 ms */
		semTake(idMutexPos, WAIT_FOREVER);

		if (isNaN(comEpaule))
		comEpaule = asservissementMoteurEpaule(angleEpauleCourant, ((consEpaule*PI)/180.0));

		inter1 = simulationEpaule(comEpaule);

		if (!isNaN(inter1))
		angleEpauleCourant = inter1;

		if (isNaN(comCoude))
		comCoude = asservissementMoteurCoude(angleCoudeCourant, ((consCoude*PI)/180.0));

		inter2 = simulationCoude(comCoude);

		if (!isNaN(inter2))
		angleCoudeCourant = inter2;
		}

		semGive(idMutexPos);
	}
}
```

On initialise tout d'abord la simulation par l'intermédiaire des fonctions appropriées puis, si les valeurs sont correctes, on effectue la simulation.

Il est à noter que nous avons utilisé des variables intermédiaires (inter1 et inter2) afin de vérifier les valeurs des angles et de la commande car, au cours de la conception, nous nous sommes rendus compte que certaines valeurs étaient, parfois, ponctuellement incorrectes (valeur NaN : Not a Number). C'est pourquoi, afin d'être certain que la simulation reçoit les bonnes valeurs, nous faisons une vérification des valeurs. Ceci est fait dans un souçis de « débuguage » car nous n'avons pas trouver la raison principale de ces valeurs incorrectes qui survenaient de manière aléatoires.

- Tâche asservissement (tAsserv) :

Cette tâche est utilisée pour effectuer l'asservissement du moteur.

```
int tAsserv() {

int error;

.       while (arretUrgence!=TRUE) {
.           semTake(idSemAsserv, WAIT_FOREVER);
.       .       /* calculer la vitesse du moteur toutes les 30 ms */
.       .       semTake(idMutexPos, WAIT_FOREVER);
.       .       comEpaule = asservissementMoteurEpaule(angleEpauleCourant, ((consEpaule*PI)/180.0));
.       .       comCoude = asservissementMoteurCoude(angleCoudeCourant, ((consCoude*PI)/180.0));
.       .       semGive(idMutexPos);

.       }

}
```

Elle est assez triviale et permet simplement d'effectuer périodiquement l'asservissement du moteur.

- Tâche échantillonage (tEchantillonage) :

Cette tâche est utilisée pour échantilloner les capteurs.

```
int tEchantillonage()
{

Echantillon echantillon;

.
.       while(arretUrgence!=TRUE) {
.           semTake(idSemEchanti, WAIT_FOREVER);
.           /* calculer la position courante du bras par rapport au vitesse du moteur  toutes les 5 ms */
.       .
.       .       semTake(idMutexPos, WAIT_FOREVER);
.       .       /* envoyer la position au dialogue */
.       .       echantillon.epaule = ((angleEpauleCourant*180.0)/PI);
.       .       echantillon.coude = ((angleCoudeCourant*180.0)/PI);
.       .       semGive(idMutexPos);
.       .
.       .       if(msgQSend(idFileEchantillon, (char *) &echantillon, sizeof(Echantillon), WAIT_FOREVER,
        MSG_PRI_NORMAL)==ERROR)
.       .       .       printf("tEchantillonage : echec emission position bras\n");
.       .
.       .
.       .
.       }
}
```

Elle calcule, successivement, les angles respectifs que l'on convertit en degré puis on les passe à tDialogue dans un tableau afin d'afficher le graphique correspondant aux angles successifs.

Il est à noter que, pour protéger l'affectation des valeurs, nous avons utilisé un mutex (idMutexPos), celui-ci est utilisé dès qu'une affectation des angles est faite.

f. partage des ressources

Avec le découpage des tâches et l'analyse de l'outil SART, nous avons observé la présence de ressources critiques à partager par les différentes tâches.

Pour accéder aux ressources, nous avons mis en place des outils tels que sémaphores binaires ou d'exclusion mutuelle, variables globales, files de messages...

Par exemple :

-La tâche asservissement doit utiliser une ressource pour l'acquisition de la position courante du bras.

-La tâche calcul doit acquérir la position courante pour la mise à jour de la position du bras en fonction de l'ordre de déplacement donné par l'opérateur.

-La tâche pince doit accéder à une variable globale protégé par un mutex pour la fermeture ou l'ouverture de celle-ci

g. priorité des tâches

Lors de la création et du paramétrage des tâches, les priorités ont été définies comme suit :

Nom de la tâche	Priorités
tDialogue	110
tPince	107
tCalculProfil	103
tEchantillon	104
tAsserv	102
tSimuBras	106

Le tableau des priorités a été établi de manière arbitraire mais en prenant en compte la nature prioritaire de l'asservissement du bras.

h. spitTimer

```
int spitTimer()
{
.       static int simu = 0;
.       static int profil = 0;
.       static int asserv = 0;
.       static int echanti = 0;
.       simu++;
    if (simu>=5) {
.           .       simu = 0;
.           .       semGive(idSemSimu);
.       }
.       profil++;
    if (profil >=30) {
.           .       profil = 0;
.           .
.           .
.           .       semGive(idSemCalcul);
.       }
.       asserv++;
    if (asserv>=10) {
.           .       asserv = 0;
.           .       semGive(idSemAsserv);
.       }
.
.       echanti++;
    if (echanti>=periodeEchanti) {
.           .       echanti = 0;
.           .       semGive(idSemEchanti);
.       }
    return 0;

}
```

Celle-ci va permettre de lancer périodiquement les tâches afin de respecter les délais demandés.

4. TRAVAIL RÉALISÉ

a. démarche de conception

Nous avons tout d'abord effectué la conception qui nous semblait représenter le mieux le système de gestion du bras (conception fournie au début de ce rapport). Au fur et à mesure de la programmation, nous avons, cependant, changé quelques éléments.

b. autocritique de la démarche

Toutefois, si nous avons dû modifié certaines parties de la conception, cela prouve que nous aurions dû y passer plus de temps et décortiquer, au maximum, cette démarche d'analyse avant de passer à la partie programmation. Ce projet nous aura appris qu'il est extrêmement important de bien spécifier le problème avant de passer à sa réalisation concrète.

c. captures d'écran

Fonctionnement de la pince

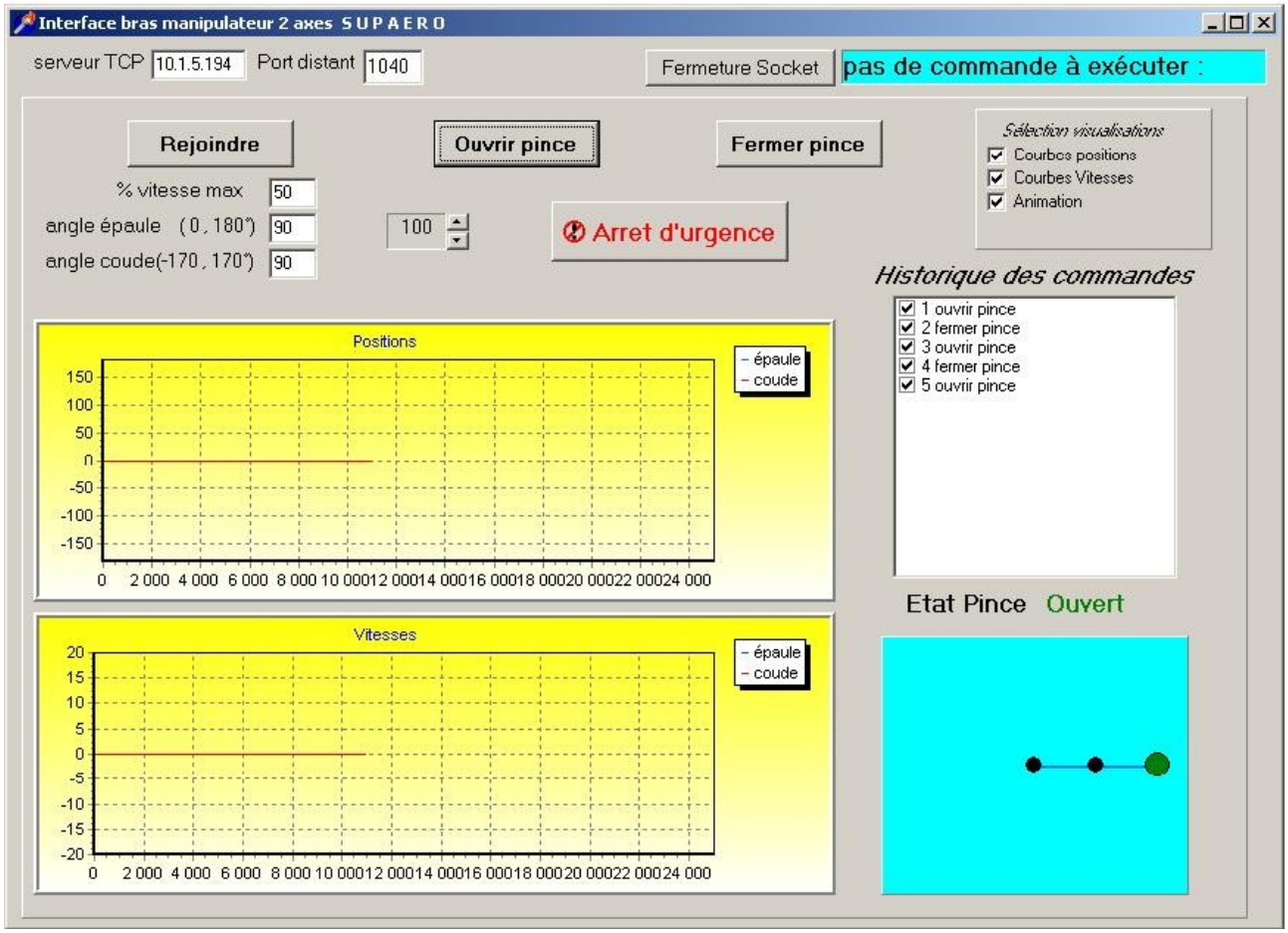


Image 3: Fonctionnement de la pince (ouverture et fermeture)

Fonctionnement de la commande « rejoindre »

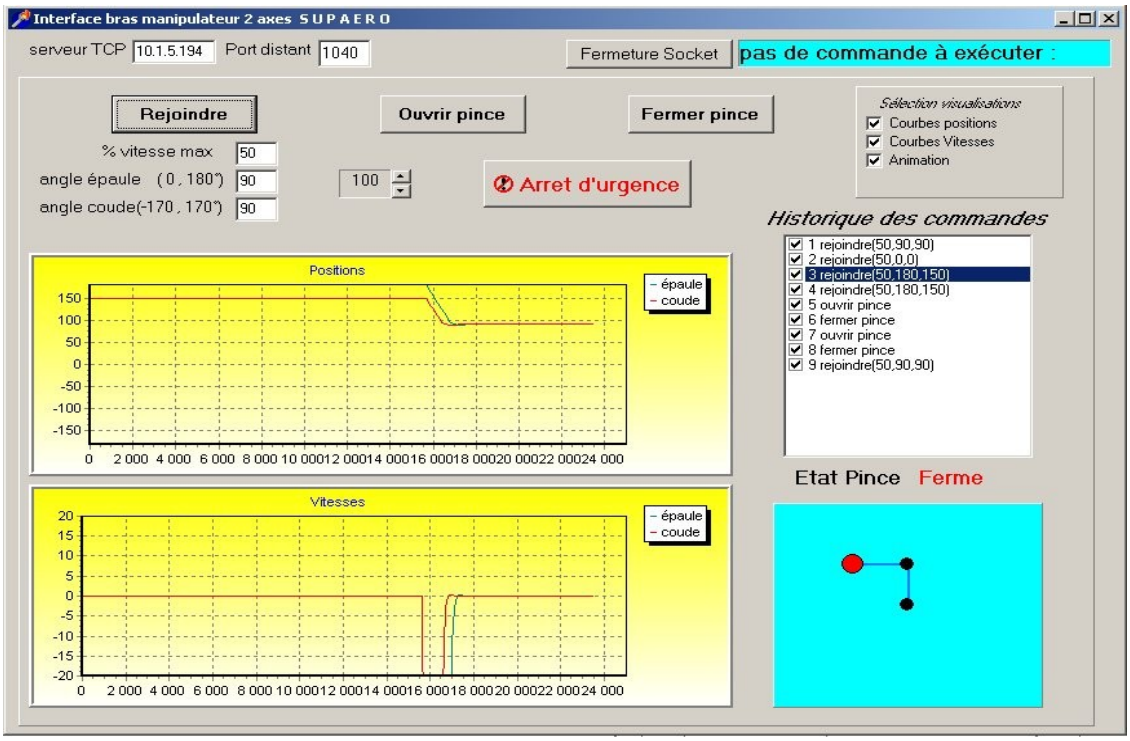


Image 3: Fonctionnement de la commande rejoindre

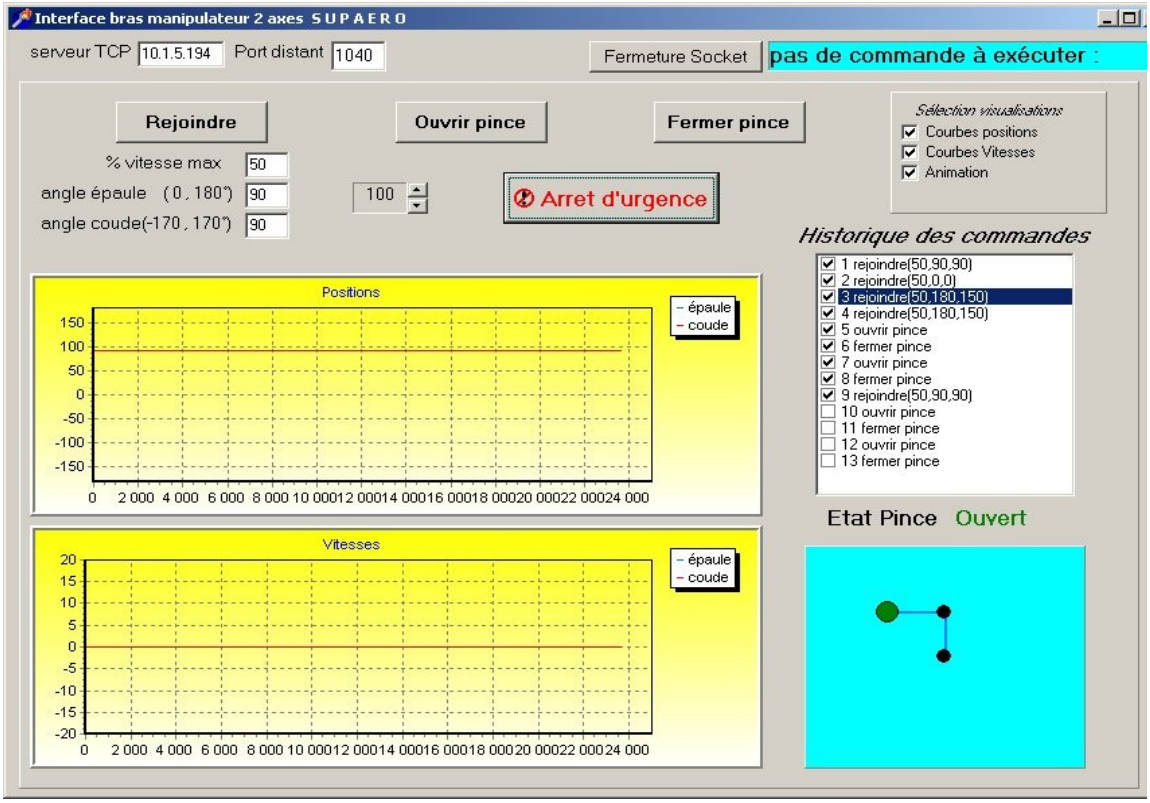


Image 4: Fonctionnement de la commandes rejoindre avec enchainement d'actions

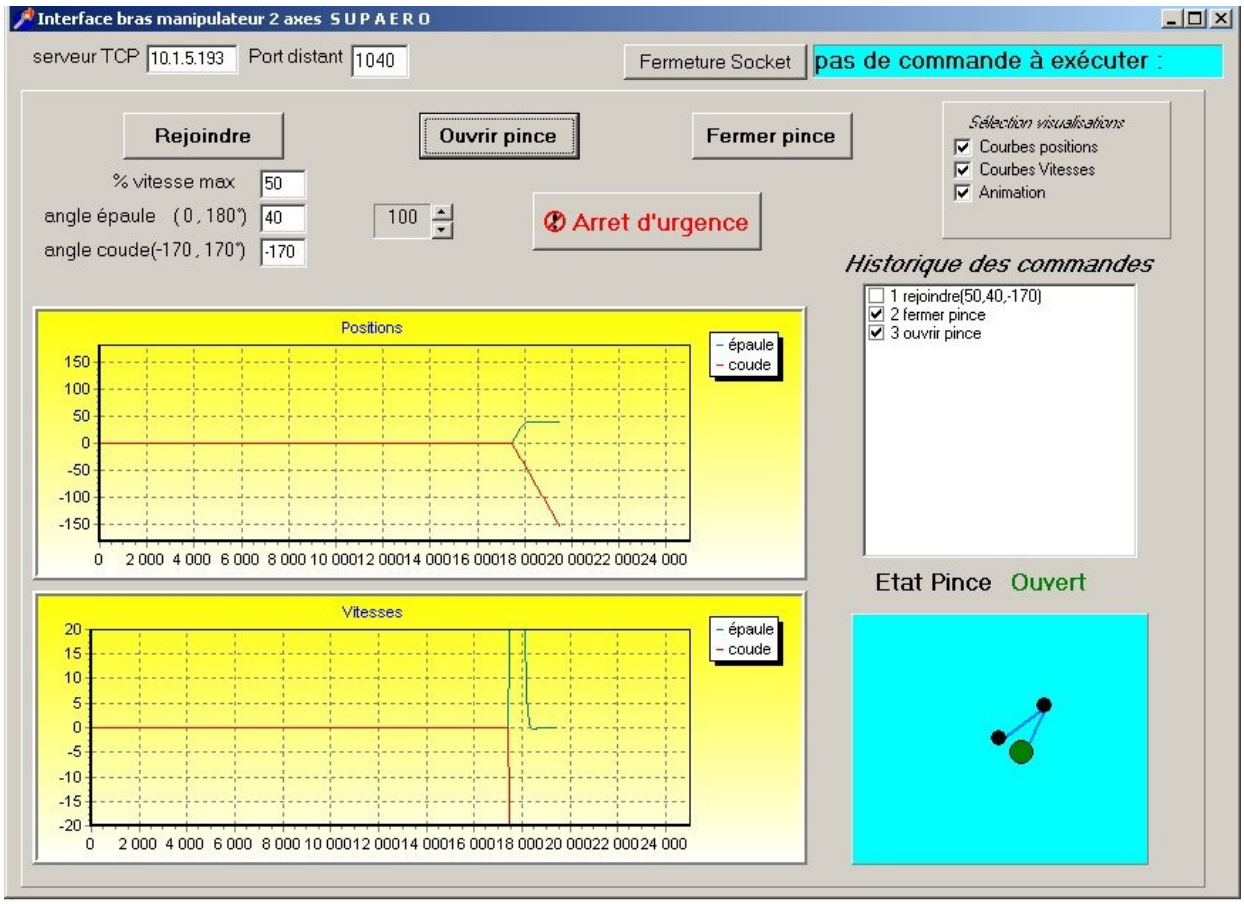


Image 5: Commande rejoindre avec d'autres paramètres

5. CONCLUSION

Bilan

Nous avons pu atteindre une grande partie des objectifs demandés dans le temps imparti.

Cependant, il reste quelques petits problèmes au niveau du développement. En effet, il y a un temps de latence assez important entre le moment où l'utilisateur effectue l'action « rejoindre » et la concrétisation de cette action. Ce problème est sans doute dû à la prise du mutex idMutexPos utilisés par beaucoup de tâches.

Sur le plan pédagogique, ce projet nous a permis d'approfondir les concepts de temps réel et de les appliquer au sein d'un travail concret.