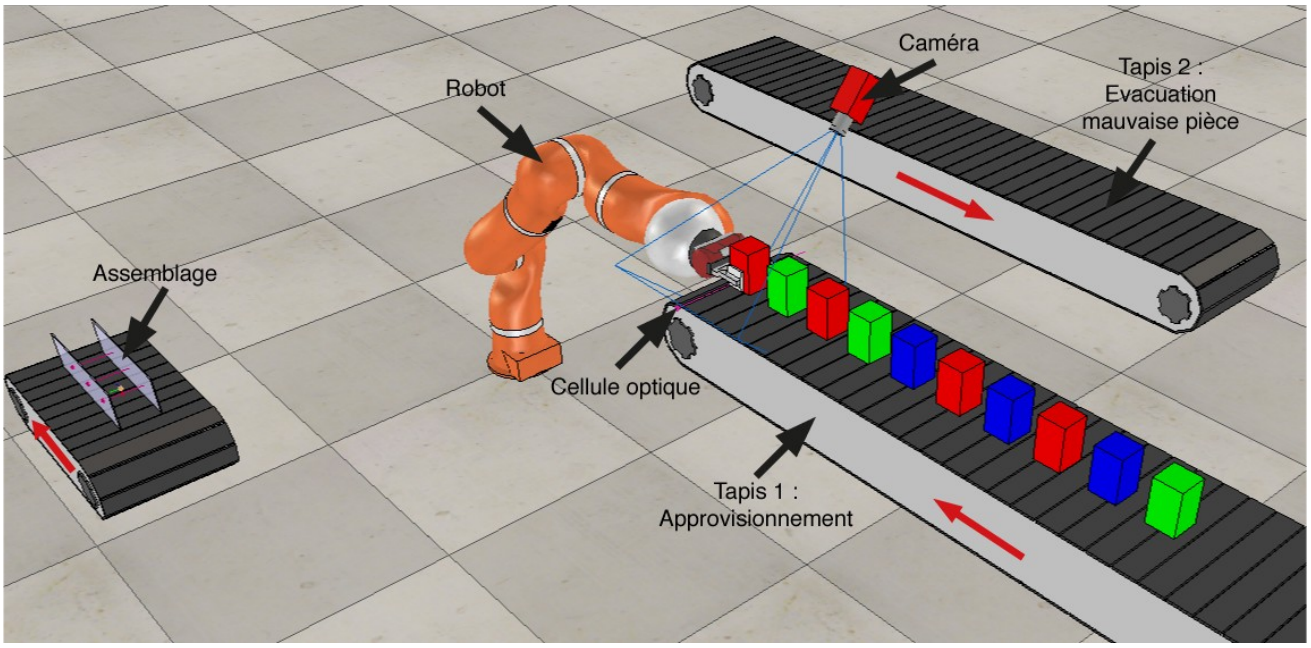
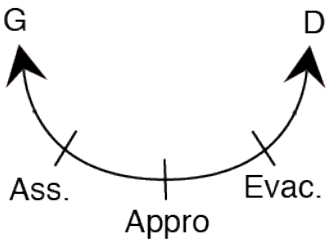
PILOTAGE D'UNE CELLULE D'ASSEMBLAGE ROBOTISEE

A. Système à commander

A un stade intermédiaire de la fabrication d'un produit manufacturé, des pièces produites dans une chaîne automatique doivent subir un assemblage. Le schéma de la figure ci-dessous représente la cellule robotisée au sein de laquelle est effectué cet assemblage.



Les éléments sont disposés de telle façon que le robot puisse accéder aux deux tapis et au poste d'assemblage. Les positions repérées du robot sont indiquées sur la figure ci-dessous.



B. Cahier des charges de la commande

Les actionneurs disponibles sont :

• sur les tapis 1 et 2 :	marche	**AV_T1**, **AV_T2**
• la caméra :	reconnaissance	**Reccam**
• sur le robot :	aller vers la droite	**D**
	aller vers la gauche	**G**
	prendre une pièce	**Prend**
	poser une pièce	**Pose**
	opérations d'assemblage	**OPi**
• au poste d'assemblage :	vérifier l'assemblage	**Verif**

Les capteurs utilisés, ou les signaux générés, sont :

• devant le tapis 1 :	cellule optique	**co**
• la caméra :	fin de la reconnaissance	**fin_reccam**

	type de pièce	**p1/p2/p3**
• sur le robot :	fin d'action prise ou pose	**fprise**, **fpose**
	robot au dessus d'un tapis	**pos_t1**, **pos_t2**
	robot en position assemblage	**pos_assem**
	fin opération d'assemblage	**fin_OPi**
• sur le tapis 2 :	arrêt du tapis	**arrêt_t2**
• au poste d'assemblage :	conformité	**assemblage_conforme**
	évacuation	**assemblage_évacué**

Cet assemblage de trois pièces P1 (rouge), P2 (verte) et P3 (bleue), repose sur trois opérations OP1, OP2 et OP3 effectuées par le robot R1. Ces opérations OPi correspondent à :

- OP1 : insérer la pièce P1 dans le premier emplacement,
- OP2 : insérer la pièce P2 dans le second emplacement,
- OP3 : insérer la pièce P3 dans le troisième emplacement et effectuer l'assemblage.

Attention les actions Prend et Pose ne concernent donc pas le pose d'assemblage.

Les pièces P1, P2, P3 arrivent sur le tapis T1 dans un **ordre aléatoire**. Dès qu'une pièce est détectée par la cellule optique (signal *co*), le tapis T1 doit alors être arrêté et l'opération de reconnaissance par la caméra (*Reccam*) doit être lancée.

Suite à l'opération de reconnaissance, si la pièce reçue est celle attendue alors l'opération correspondante doit être lancée. Dès que la pièce a été prise par le robot, le tapis T1 peut être relancé. Si la pièce n'est pas du type attendu, le robot doit évacuer cette mauvaise pièce sur le tapis de retour au stock T2. Le tapis T2 est toujours en mouvement, sauf pour la pause d'une pièce pour laquelle il doit être arrêté au préalable. Comme le tapis T2 met un certain temps pour s'arrêter, il faut attendre le signal *arrêt_t2* pour être sûr de l'arrêt.

Lorsque l'assemblage est terminé, il doit être vérifié (Verif). S'il est déclaré conforme (assemblage_conforme) il est automatiquement évacué et le poste d'assemblage devient à nouveau libre (assemblage_évacué). Aucun assemblage ne peut reprendre sur le pose d'assemblage tant que ce dernier n'a pas été évacué.

État initial : le robot est supposé au dessus du poste d'assemblage (libre), aucune pièce n'est arrivée sur le tapis 1 ; les tapis 1 et 2 sont en marche.

C. Démarche de l'étude

La spécification de la commande se fait par réseaux de Petri. La synthèse du contrôleur doit être effectuée selon une approche multitâche, le corps de chaque tâche s'appuyant sur une *mise en œuvre guidée par le marquage* (ce qui implique une modélisation par machines à états finis - MEF). Le simulateur du système piloté est également une tâche de votre application (cf annexe de mise en œuvre).

D. Complément pour la réalisation logicielle

Ne disposant pas du système physique réel, un **simulateur** vous est fourni. Ce simulateur réagi aux commandes reçues et génère les signaux adéquats. Il permet aussi de suivre visuellement l'évolution du processus.

E. Travail requis

Dans un premier temps, on considérera un modèle simplifié de l'installation afin de se familiariser avec les aspects de modélisation par réseaux de Petri et leur traduction en un programme multi-tâches.

Pour ceux ayant réussis cette première étape, ils pourront compléter leur modélisation pour correspondre à l'installation complète.

Modélisation simplifiée

- Les opérations d'assemblage OPi sont regroupées en une seule opération d'assemblage OP.
- Les ordres Prend, Pose et OP sont immédiats et ne nécessitent donc pas d'attendre les signaux de fin d'opération.
- La caméra fournit directement l'information « assembler » ou « évacuer » en fonction de la pièce reçue.

Modélisation complète

Dans le cas de la modélisation complète, le système se comporte exactement comme décrit dans le cahier des charges.

1) Modélisation du problème

La première partie du travail consiste à modéliser le système sous forme de réseaux de Petri. Il est impératif que le système complet soit découpé en plusieurs sous réseaux. Ceci permet de tester le fonctionnement de chaque partie de façon indépendante et simplifie la traduction en tâche. Tous les sous réseaux seront connectés ensemble à la fin pour vérifier le comportement global.

Le logiciel TINA, téléchargeable depuis http://projects.laas.fr/tina/, permet d'établir et de vérifier le bon fonctionnement de vos RdP. Il est gratuit est disponible pour Windows, Mac OS X et Linux.

2) Implémentation multi-tâches

Une fois la modélisation du système effectuée, il faut la traduire en un programme C/C++ multi-tâches. Un projet de base est disponible ici https://github.com/BenjaminNavarro/assembly_control. CMake (version ≥ 2.6) est nécessaire pour compiler ce projet, il est téléchargeable depuis https://cmake.org/. L'aspect multi-tâches sera implémenté via les mécanismes proposés par C++11. Un compilateur tel que GCC (version ≥ 4.7) est donc nécessaire.
Le simulateur utilisé, téléchargeable depuis http://www.coppeliarobotics.com/downloads.html (version PRO-EDU), permet de simuler des environnements dynamiques et est axé pour la robotique. Il est lui aussi gratuit et disponible sur les systèmes d'exploitation principaux. La scène à charger dans le simulateur est incluse dans le projet (sous dossier « vrep »).

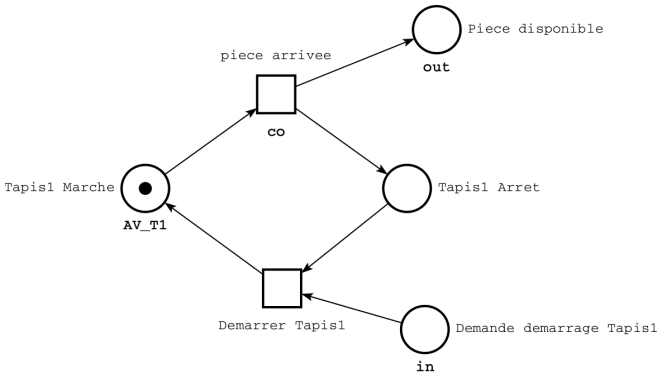
Note : Même s'il est possible de trouver des versions de GCC compatibles avec Windows, il est fortement recommandé de travailler avec un système UNIX tel que Linux ou Mac OS X. Aucun support ne sera apporté aux problèmes relatifs à l'installation et à l'utilisation de GCC sur Windows.

F. Exemple

Afin de bien comprendre la démarche à suivre, nous allons prendre la gestion du tapis 1 comme exemple.

1) *Modélisation*

Le tapis possède deux états : arrêté ou en marche. Le passage de l'état « en marche » à l'état « arrêté » se fait lorsqu'un objet franchit le capteur optique et que le signal « co » arrive. Le redémarrage du tapis est déclenché par une autre partie du processus lorsque la pièce a été prise par le robot. Le réseau de Petri équivalent est le suivant :



Dans TINA, il est possible de spécifier un nom et un label pour chaque place ou transition. Lors de la fusion de plusieurs RdP, les places et transitions qui ont le même nom seront fusionnées ensemble. Le label permet d'ajouter le nom de l'action ou du signal associé à une place ou une transition en commentaire. Pour les places d'entrée/sortie, le label permet de préciser le type de la place (in/out).

Pour le nommage, deux solutions sont possibles. La première est de donner un nom unique à toutes les places et transitions et d'effectuer la fusion manuellement (plus long mais plus sûr) ou bien de nommer les places d'entrée/sortie correspondantes de la même façon pour que la fusion se fasse automatiquement (plus rapide mais peut générer des problèmes difficiles à résoudre en cas de conflits de noms non prévus). Dans tous les cas, la convention de nommage doit être respectée et fera partie des points évalués.

2) *Traduction*

Une fois le réseau de Petri établi, on peut le traduire en C++ afin de le tester avec le simulateur. Pour cela, trois éléments seront toujours nécessaires :

- une énumération des états possibles :

```
enum EtatsTapis1 {
    Tapis1EnMarche,
    Tapis1Arrete
};
```

- la déclaration des signaux d'entrée/sortie utilisés (la classe Signal sera fournie) :

```
Signal demande_demarrage_tapis1;
```

- une fonction contenant le code correspondant à la machine à état du RdP :

```
void tache_tapis1() {
	EtatsTapis1 etat = Tapis1EnMarche;
	while(1) {
		switch(etat) {
			case Tapis1EnMarche:
				sim.set_AV_T1(true);			// AV_T1 = 1
				sim.wait_co();				// wait signal, blocking call
				etat = Tapis1Arrete;
			break;

			case Tapis1Arrete:
				sim.set_AV_T1(false);			// AV_T1 = 0
				demande_demarrage_tapis1.wait();	// wait signal, blocking call
				etat = Tapis1EnMarche;
			break;
		}
	}
}
```

Dans cette fonction, tous les états possèdent un appel bloquant (sim.wait_xxx()). Ça ne sera pas toujours le cas car certains signaux du simulateur sont uniquement accessible en lecture (sim.read_xxx()). Dans ce cas là, il faut venir lire régulièrement le signal et changer d'état ou pas en fonction de sa valeur. Un exemple d'une tâche fonctionnant sur ce principe est donnée ci-dessous :

```
enum EtatsAutreTache {
	etat1,
	etat2
};

void autre_tache() {
	typedef chrono::duration<int, chrono::milliseconds::period> cycle; //define the type 'cycle'

	EtatsAutreTache etat = etat1;

	while(1) {
		auto start_time = chrono::steady_clock::now();		// start_time = current time
		auto end_time = start_time + cycle(10);		// end_time = current_time + 10ms

		switch(etat) {
			case etat1:
				if(sim.read_fin_reccam())		// read signal, non-blocking call
					etat = etat2;
			break;

			case etat2:
				//unlock tasks waiting on this signal
				demande_demarrage_tapis1.notify();
			break;
		}
		// Stop thread execution until 'end_time' (next cycle)
		this_thread::sleep_until(end_time);
	}
}
```