Master 1 Électronique Électrotechnique Automatique, Ingénierie des Systèmes Temps-Réel

UNIVERSITÉ TOULOUSE III PAUL SABATIER — Université de Toulouse

STAGE
Intégration et commande sous ROS du robot Baxter au sein d'une cellule flexible d'assemblage

Auteur

Bruno Dato

Encadrants

C. Briand, M. Taïx

30 juin 2016

Remerciements

Je tiens à remercier mes encadrant de stage C. Briand et M. Taïx pour m'avoir permis de réaliser ce stage. Je remercie aussi toutes les personnes de l'AIP pour leur accueil au sein de la halle technologique durant toute la durée de mon stage.

Sommaire

Introduction

Chapitre 1

Présentation du stage

1.1 Projet global

Mon stage s'est déroulé au pôle AIP/PRIMECA, à Toulouse, sur le campus de l'université Paul Sabatier à la halle technologique. Ces locaux disposent de divers systèmes tels que des robots mobiles, des robots manipulateurs, une cellule flexible de production, un vidéo projecteur interactif, un réseau de caméras... Dans le cadre de mon stage, j'ai travaillé sur l'interaction entre un des robot manipulateurs : le robot Baxter, et la ligne transitique MONTRAC sur laquelle j'avais déjà effectué mon projet de TER.

Ce stage s'inscrit dans un projet de plus grande envergure visant à faire interagir de nombreux systèmes se trouvant à la halle technologique comme vous pouvez le voir sur la figure 1.1. En effet, depuis 2 ans, il y a eu différents projets et stages concernant le vidéo projecteur interactif, la ligne transitique et le réseau caméras.

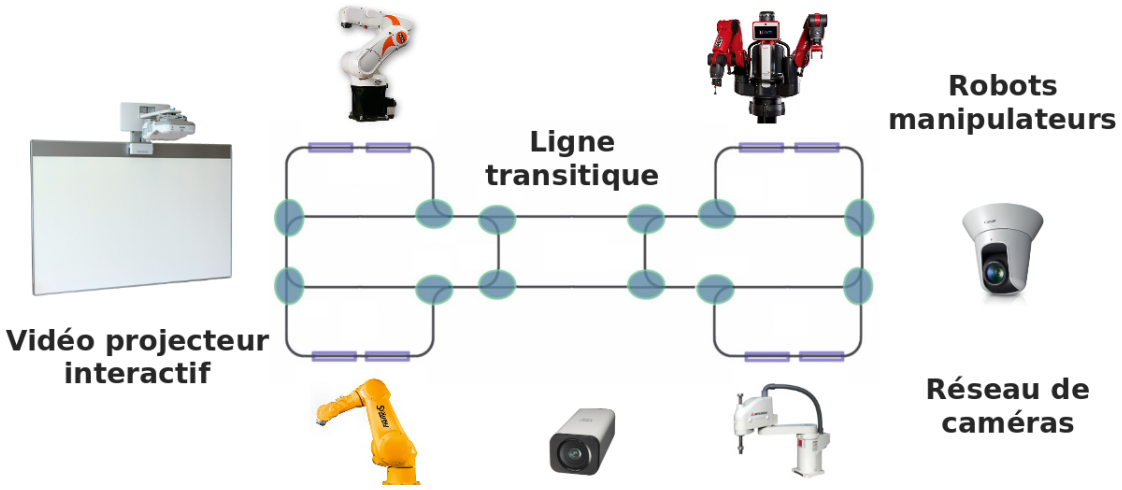

FIGURE 1.1 – Vue globale des systèmes à faire interagir dans un futur proche

1.2 Le robot Baxter

Le robot Baxter est un robot humanoïde collaboratif plus communément appelé cobot. Il a été conçu par Rodney Brooks et sa société Rethink Robotics. Un des principaux atouts de ce robot est que sa version dédiée à la recherche peut être commandé à l'aide du middleware ROS qui est très utilisé aujourd'hui en robotique.

FIGURE 1.2 – Le robot collaboratif Baxter

Baxter dispose de deux bras manipulateurs comme vous pouvez le voir sur la figure 1.2, chacun de ces deux bras peut être équipé de pinces ou de ventouses reliés à un système pneumatique. Il dispose de différents capteurs : un accéléromètre, un capteur infrarouge et une caméra sur chaque bras. Il possède aussi une troisième caméra et un sonar au niveau de sa tête, le sonar lui permet de détecter en 3D des objets ou des humains lorsqu'ils sont à une certaine distance. Il possède aussi différents boutons sur ces bras et ses épaules qui permettent d'interagir directement avec lui à l'aide de l'écran situé sur sa tête. Enfin il est possible de déplacer ses bras en les saisissant par leur manchette (chaque manchette est munie d'un capteur de pression).

Durant mon stage, j'ai surtout travaillé sur le mouvement des bras manipulateurs équipés de pinces afin de saisir et poser des objets bien qu'ils soient hypothétiques. Chaque bras possède sept degrés de liberté qui sont les angles de rotation *S1*, *E0*, *E1*, *W0*, *W1* et *W2* que vous pouvez voir sur la figure 1.3 ci dessous.

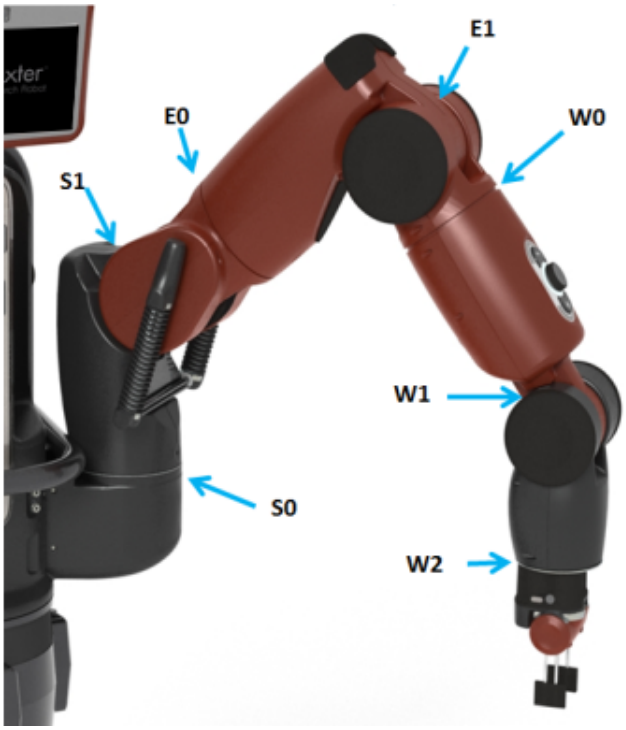

FIGURE 1.3 – Les différents angles des bras du robot Baxter

Pour le mouvement des bras, Baxter dispose de quatre modes différents :

- **Joint Position Control** : Il faut spécifier les valeurs de position que l'on souhaite atteindre pour chacun des sept angles. Le mouvement est ensuite contrôlé automatiquement par Baxter afin d'éviter des collisions entre les deux bras ou avec Baxter lui même mais aussi des positions impossibles. La vitesse est aussi contrôlée.
- **Raw Joint Position Control** : C'est le même fonctionnement que le mode Joint Position Control cependant, le mouvement est moins contrôlé cette fois : il n'y a plus d'évitement des collisions et les bras se déplacent à la vitesse maximale que Baxter peut fournir. Il faut donc faire très attention avec ce mode.
- **Joint Velocity Control** : Il faut spécifier les valeurs de vitesse que l'on souhaite atteindre pour chacun des sept angles. Il y a à nouveau un contrôle pour éviter les collisions cependant la vitesse maximale peut

être atteinte.

- **Joint Velocity Control** : Il faut spécifier les valeurs de moment (ou couple) que l'on souhaite atteindre pour chacun des sept angles. Pour ce mode, les contrôles du mouvement sont minimes, il faut donc l'utiliser avec beaucoup de précautions.

Pour les commandes que nous verrons par la suite, j'ai utilisé le premier mode : Joint Position Control, couplé avec un service ROS proposé par le robot. Le service permet, à partir d'une position cartésienne dans l'espace ainsi que d'une orientation souhaitée pour l'extrémité d'un des bras, de fournir les valeurs de position des sept angles qui permettent d'atteindre cette disposition.

1.3 Ligne transitique MONTRAC®

1.3.1 Présentation

La ligne transitique MONTRAC est composée de rails alimentés en énergie électrique sur lesquels circulent des navettes. Les navettes ne peuvent se déplacer que dans un sens sur ces rails.

Pour commander la ligne, on dispose de cinq automates programmables industriels que nous appellerons API. Il y en a 2 de la marque Siemens® et 3 de type Schneider®. Ces automates gèrent les différents actionneurs et capteurs situés sur les rails. Les navettes ne sont pas programmables, une fois allumées elles avancent jusqu'à ce qu'on les arrête. Elles possèdent un capteur de proximité situé à l'avant pour éviter les collisions entre les navettes et les stopper lorsqu'elles rencontrent un obstacle.

Cette ligne comporte 5 "secteurs" dont un central et 4 postes de travail. Chaque "secteur" est contrôlé par un des API qui agit sur la ligne via un système d'air comprimé. Les zones 1 à 3 correspondent aux automates Schneider® et les deux autres zones correspondent aux automates Siemens®. La ligne possède 12 aiguillages comme on peut le voir sur la figure 1.4.

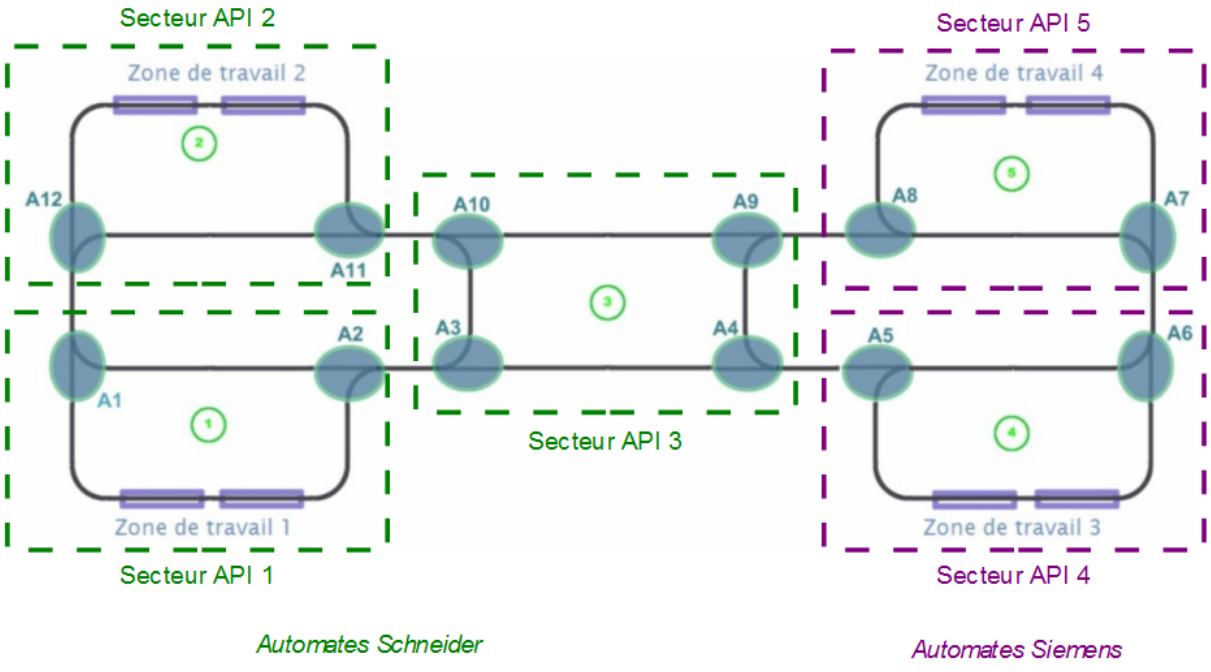

FIGURE 1.4 – Schéma simplifié de la ligne transitique

1.3.2 Capteurs et actionneurs

Les actionneurs permettent de commander les points d'arrêts des navettes, les aiguillages et des ergots pouvant bloquer les navettes à certains endroits lorsqu'elles sont arrêtées. Les capteurs nous permettent de connaître les positions des navettes, des aiguillages et des ergots. Les listes des actionneurs et capteurs sont données ci-dessous.

RxG	Positionne l'aiguillage x à gauche
RxD	Positionne l'aiguillage x à droite
Dx	Dévérrouille l'aiguillage X
Vx	vérrouille l'aiguillage X
STx	Quand STx vaut 0, les navettes s'arrêtent au niveau de l'actionneur
PIx	Blocage des navettes sur le poste de travail

Table 1.1 – Actionneurs

CPx	Capteur de Position. Vaut 1 quand une navette est sur le capteur
PSx	Capteur de Stop situé juste en face d'un actionneur STx pouvant arrêter la navette
CPIx	Vaut 1 quand l'ergot PI est sorti
DxD	Vaut 1 quand l'aiguillage x est à droite
DxG	Vaut 1 quand l'aiguillage x est à gauche

Table 1.2 – Capteurs

Durant ce stage, j'ai commandé de la ligne via les API des zones 1 et 2 (automates Schneider®). Ces automates disposent de 16 entrées et sorties chacun mais toutes ne sont pas utilisées. Les tables 1.3 et 1.4 décrivent respectivement le câblage des API 1 et 2.

On peut voir sur la figure 1.5 une vision globale de la ligne transitique avec tous les capteurs et actionneurs, les zones contrôlées par les API ainsi que les orientations de chaque rail.

OUT	A1		V1		A2		V2		S1		S2		S3		S4		S5		UP1		UP2	
	R1D	R1G	V1	D1	R2D	R2G	V2	D2	ST1		ST2		ST3		ST5		ST4		PI1		PI2	
IN	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15						
	PS1	PS2	PS3	PS5	PS4	D1D	D1G	CP1	CPI1	CPI2	D2D	D2G	CP2	PS6								

Table 1.3 – Configuration des entrées/sorties de l'automate Schneider 1

OUT	A1		V1		A2		V2		S1		S2		S3		S4		S5		UP1		UP2	
	R11D	R11G	V11	D11	R12D	R12G	V12	D12	ST20		ST21		ST22		ST24		ST23		PI7		PI8	
IN	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15						
	PS20	PS21	PS22	PS24	PS23	D11D	D11G	CP9	CPI7	CPI8	D12G	D12G	CP10	PS1								

Table 1.4 – Configuration des entrées/sorties de l'automate Schneider 2

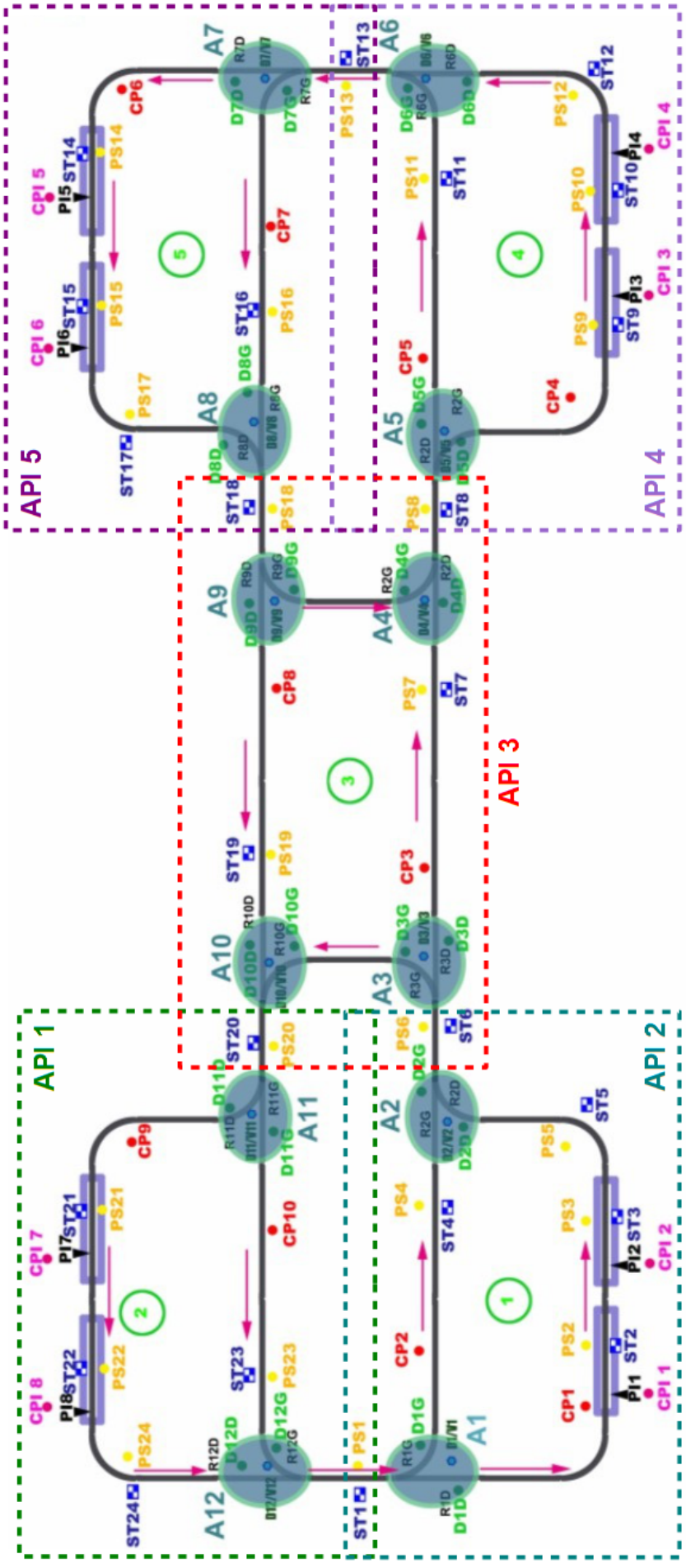

FIGURE 1.5 – Schéma détaillé de la ligne transitique

1.4 Simulation de la ligne transitique

Une simulation de la ligne transitique a été conçue par des étudiant de l'ENSEEIHT à l'aide du logiciel V-REP [1]. Cette simulation se comporte comme la ligne MONTRAC®, bien qu'elle ne possède pas tous les capteurs et actionneurs (les différences sont détaillés table 1.5). Elle permet de valider un grand nombre de commandes avant des les tester directement sur la ligne réelle.

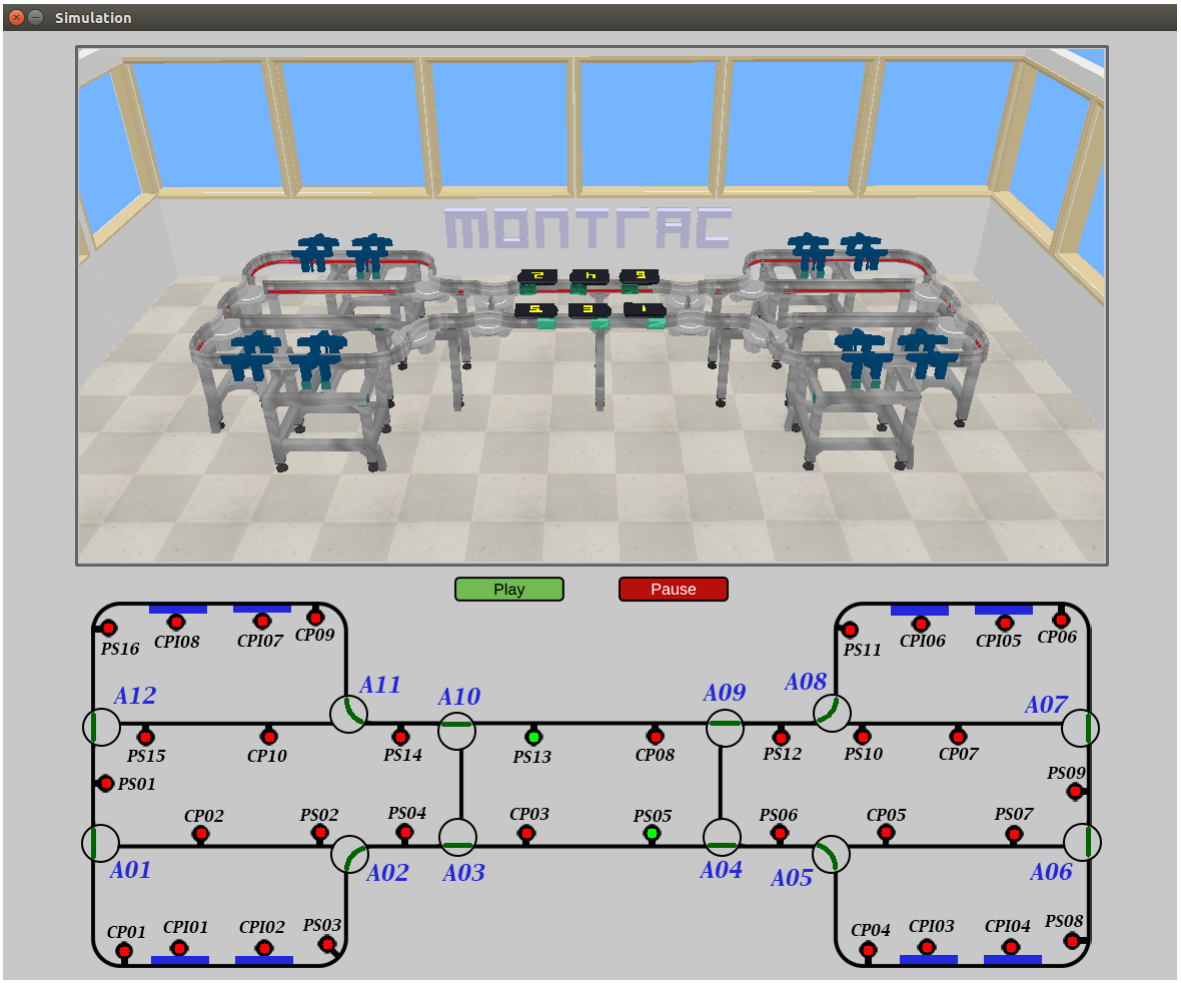

FIGURE 1.6 – Simulation de la ligne transitique

Pour la conception de la simulation, les étudiants on regroupé les capteurs en catégories : CPI, CP, PS, DG et DD qui correspondent à ceux cités précédemment (table 1.2). Il y a cependant quelques différences pour les capteur PS et CPI comme on peut le voir sur la table 1.5.

Ligne transitique	PS1	PS2	PS3	PS4	PS5	PS6	PS7	PS8	PS9	PS10	PS11	PS12	PS13
Simulation	PS1	CPI1	CPI2	PS2	PS3	PS4	PS5	PS6	CPI3	CPI4	PS7	PS8	PS9

Ligne transitique	PS14	PS15	PS16	PS17	PS18	PS19	PS20	PS21	PS22	PS23	PS24
Simulation	CPI5	CPI6	PS10	PS11	PS12	PS13	PS14	CPI7	CPI8	PS15	PS16

TABLE 1.5 – Capteurs de stop de la ligne transitique et de la simulation

1. V-REP est une plateforme de simulation pour tout type de robots **[7]**.

Sur la simulation, les capteur CPI ne sont pas les capteurs de position des ergots mais des capteurs de stop aussi. Pour les actionneurs, il y a 5 registres : RD, RG, LOCK, STOP et GO ; RD et RG fonctionnement de la même manière que ceux de la table 1.1 mais LOCK, STOP et GO ont un fonctionnement différent :

- LOCKx = $\overline{\text{Vx}}$.Dx
- STOPx = $\overline{\text{STx}}$
- GOx = STx

1.5 Problématique et solution mise en place

Comme je l'ai dit précédemment, le but de mon stage était de faire interagir le ligne transitique (ou sa simulation) avec le robot Baxter. Pour cela j'ai utilisé le middleware ROS (Robot Operating System), c'est un ensemble de librairies et d'outils qui permettent de mettre en place toutes sortes d'applications robotiques.

Je me suis servi de ce que mon groupe de TER et moi même avions effectué pour établir la communication entre les automates de la ligne transitique et ROS **[5]**. J'ai alors ajouté la communication de la ligne transitique (ou la simulation) avec le robot Baxter ainsi que sa commande à lui même à l'aide de ROS.

Les commandes de la ligne transitique ont été modélisées par des réseaux de Petri (RdP) afin d'utiliser plusieurs navettes en parallèle. Les commandes du robot Baxter ont été modélisé par des machinés à états finis (MEF), une pour chaque bras du robot.

La commande ainsi réalisé possède l'architecture figure 1.7 que vous pouvez observer ci dessous.

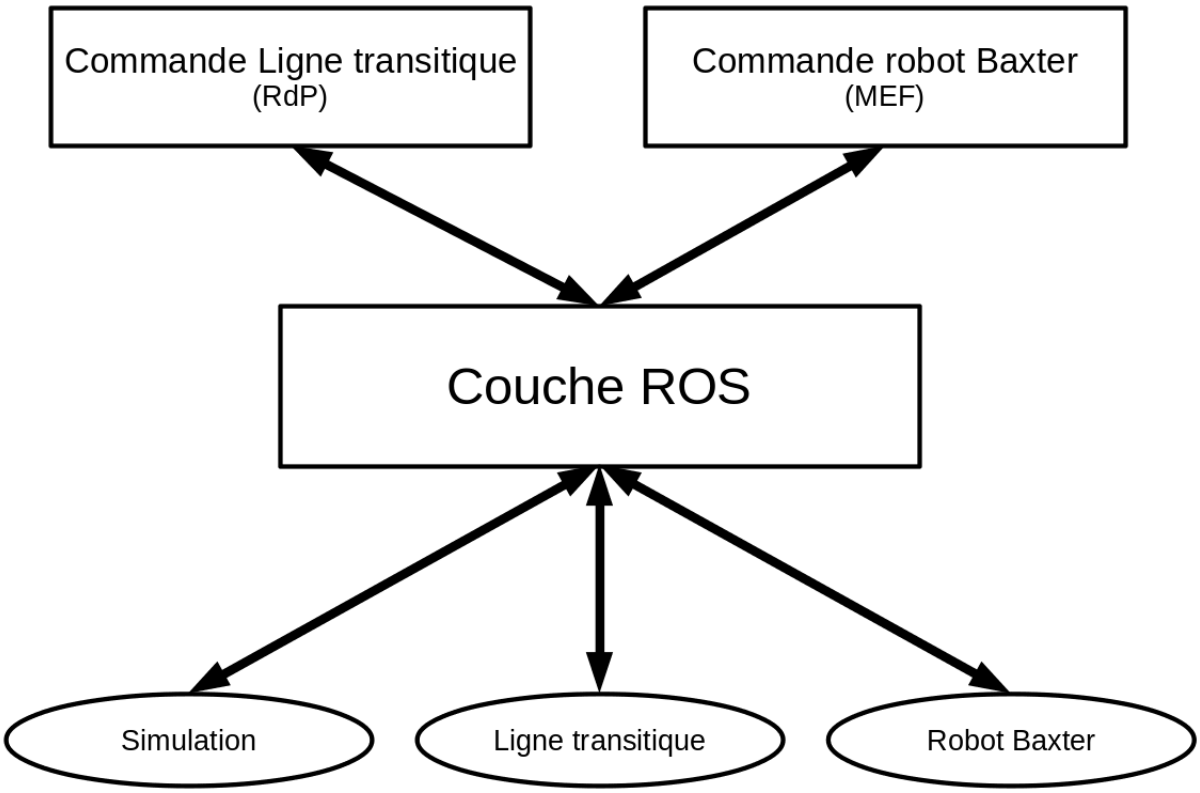

FIGURE 1.7 – Architecture de la solution mise en place

1.6 Présentation du middleware ROS

ROS (Robot Operating System) est une plateforme de développement logiciel qui facilite le développement d'applications robotiques car il permet de créer des pont de communication entre différentes entités de façon très simple. ROS peut être implémenté dans 2 langages : le Python et le C++, nous avons choisi le C++.

ROS permet l'échange de messages entre nœuds qui sont stockés dans des répertoires nommés packages. Les nœuds sont des exécutables qui utilisent ROS afin de communiquer avec d'autres nœuds. Pour que les nœuds puissent communiquer entre eux, il faut lancer la plateforme en activant un maître. Les nœuds peuvent échanger alors les informations entre eux soit de manière asynchrone via un topic (sujet) ou de manière synchrone via un service. Chaque nœud ROS évolue en parallèle par rapport aux autres.

Le principe des topics est assez simple, il est possible de publier sur un topic ou bien d'y être abonné. Lorsqu'un nœud publie sur un topic, il envoie un message à des instants choisis par le programmeur (à une certaine fréquence ou sous certaines conditions). Lorsqu'un nœud est abonné à un topic, il reçoit tous les messages publiés sur ce topic et pour chaque message reçu, une fonction appelée "Callback" est lancée afin de traiter le message comme on le souhaite. Il est possible de publier sur plusieurs topics et d'être abonné à plusieurs topics à la fois. Pour publier et lire des messages sur un topic, il faut aussi définir les types de messages que l'on souhaite utiliser.

Lorsque qu'un nœud utilise un service, il est le client de ce service, il doit alors définir une requête à destination de ce service qui est elle aussi sous forme de message et il recevra ensuite la réponse à sa requête une fois le service appelé.

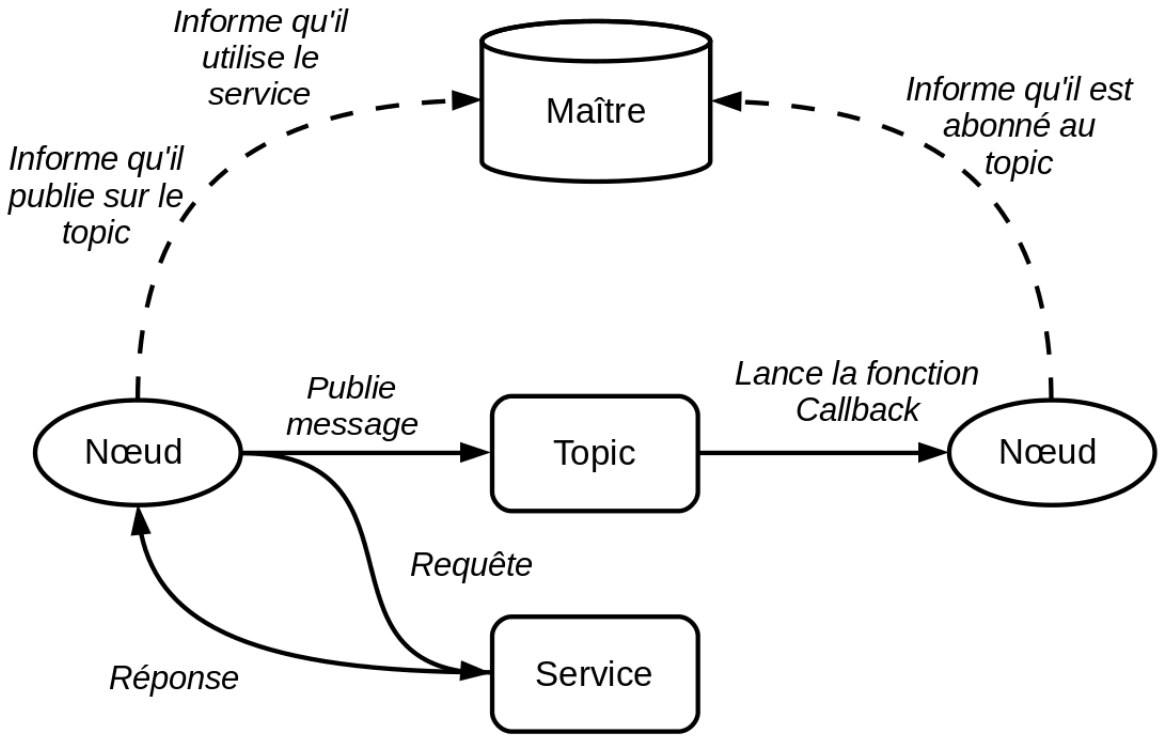

FIGURE 1.8 – Fonctionnement d'un topic sur ROS

Chapitre 2

ROS

2.1 Les topics et services de Baxter

Afin de communiquer avec Baxter, le robot possède plusieurs topics ROS sur lesquels il publie régulièrement pour envoyer les valeurs de ses capteurs et d'autres auxquels il est abonné pour pouvoir le commander. Je présenterais ici les topics et un service que j'ai utilisés pour la commande de Baxter ainsi que d'autres topics qui peuvent s'avérer utiles.

2.1.1 Les topics des états

/robot/state

Ce topic permet de savoir si le robot est activé avant de le commander, il utilise des messages de type "baxter_core_msgs/AssemblyState.msg" :

bool enabled
bool stopped
bool error
uint8 estop_button
uint8 estop_source

/robot/joint_states

Ce topic permet de connaitre l'état de chacun des angles du robot (nom, position, vitesse et effort), il utilise des messages de type "sensor_msgs/JointState.msg" :

std_msgs/Header header
string[] name
float64[] position
float64[] velocity
float64[] effort

/robot/limb/left/endpoint_state et /robot/limb/right/endpoint_state

Ces topics permettent de connaître la position des extrémités des bras en cordonnées cartésiennes (pose.position), ainsi que leur orientation en quaternions (pose.quaternions) et les efforts qui sont exercés à ces extrémités (twist et wrench). Les messages sont de type "baxter_core_msgs/EndpointState.msg" :

geometry_msgs/Pose pose
geometry_msgs/Twist twist
geometry_msgs/Wrench wrench

avec "geometry_msgs/Pose.msg" :

geometry_msgs/Point position
geometry_msgs/Quaternion orientation

/robot/end_effector/left_gripper/state et /robot/end_effector/right_gripper/state

Ces topics permettent de connaître l'état de la pince du bras gauche ou droit, ils utilisent les messages de type "baxter_core_msgs/EndEffectorState" :

time timestamp
uint32 id
uint8 enabled
uint8 calibrated
uint8 ready
uint8 moving
uint8 gripping
uint8 missed
uint8 error
uint8 reverse
float32 position
float32 force
string state
string command
string command_sender
uint32 command_sequence

2.1.2 Les topics de commande

/robot/set_super_enable

Ce topic permet d'activer le robot afin de le commander, il utilise des message de type "std_msgs/Bool.msg" :

bool data

/robot/limb/left/joint_command et /robot/limb/right/joint_command

Ces topics permettent de commander les bras du robot suivant les modes décrit dans 1.2, ils utilisent les messages de type "baxter_core_msgs/JointCommand.msg" :

int32 mode
float64[] command
string[] names

/robot/end_effector/left_gripper/command et /robot/end_effector/right_gripper/command

Ces topics permettent de commander les pinces gauche ou droite, ils utilisent les messages de type "baxter_core_msgs/EndEffectorCommand" :

uint32 id
string command
string args
string sender
uint32 sequence

2.1.3 Le service Inverse Kinematics

2.2 Les topics de communication entre Baxter et la ligne transitique

Afin d'établir la communication entre le robot Baxter et la ligne transitiquen j'ai défini des topics supplémentaires qui utilisent tous des messages de type "std_msgs/Bool.msg" :

- **/pont_BaxterLigneTransitique/<left/right>_arm/attente_prise** : Ce topic permet au robot Baxter d'indiquer à la ligne transitique qu'il est en attente d'une prise pour le bras gauche ou droit.
- **/pont_BaxterLigneTransitique/<left/right>_arm/prise_demandee** : Ce topic permet à la ligne transitique de demander un prise au robot avec le bras gauche ou droit.
- **/pont_BaxterLigneTransitique/<left/right>_arm/prise_effectuee** : Ce topic permet au robot de notifier la ligne qu'une prise avec le bras gauche ou droit a été effectuée.

2.3 Le noeud Commande_Baxter

2.3.1 La classe Baxter

Attributs

Méthodes

2.3.2 Les classes Baxter_left_arm et Baxter_right_arm

Attributs

Méthodes

2.4 Le noeud Commande

2.4.1 Les classes Capteurs et Actionneurs

Rappel

2.4.2 La classe Communication_Baxter

Attributs

Méthodes

Chapitre 3

Synthèse de commande

3.0.3 Commande du robot seul

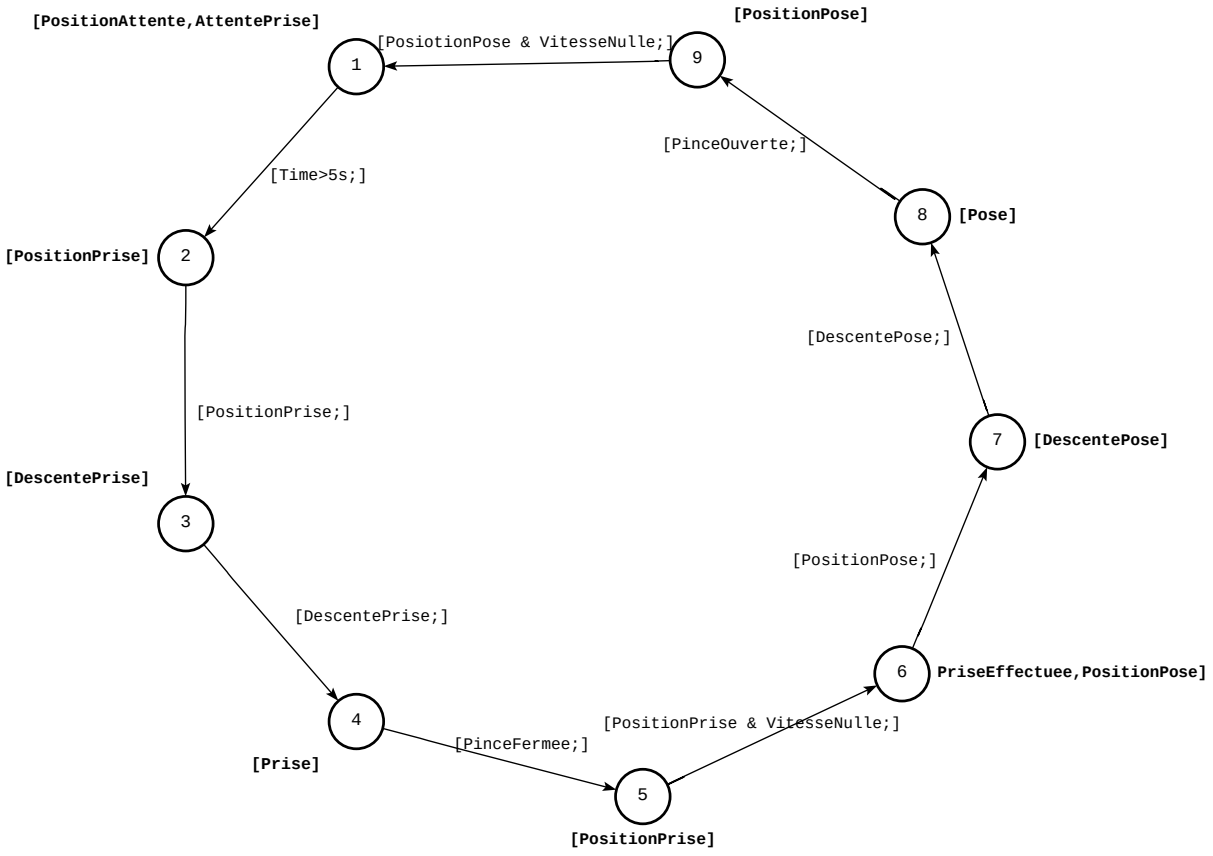

FIGURE 3.1 – Machine à états finis de la commande d'un bras du robot Baxter

3.0.4 Commande de la ligne transitique MONTRAC en intération avec le robot Baxter

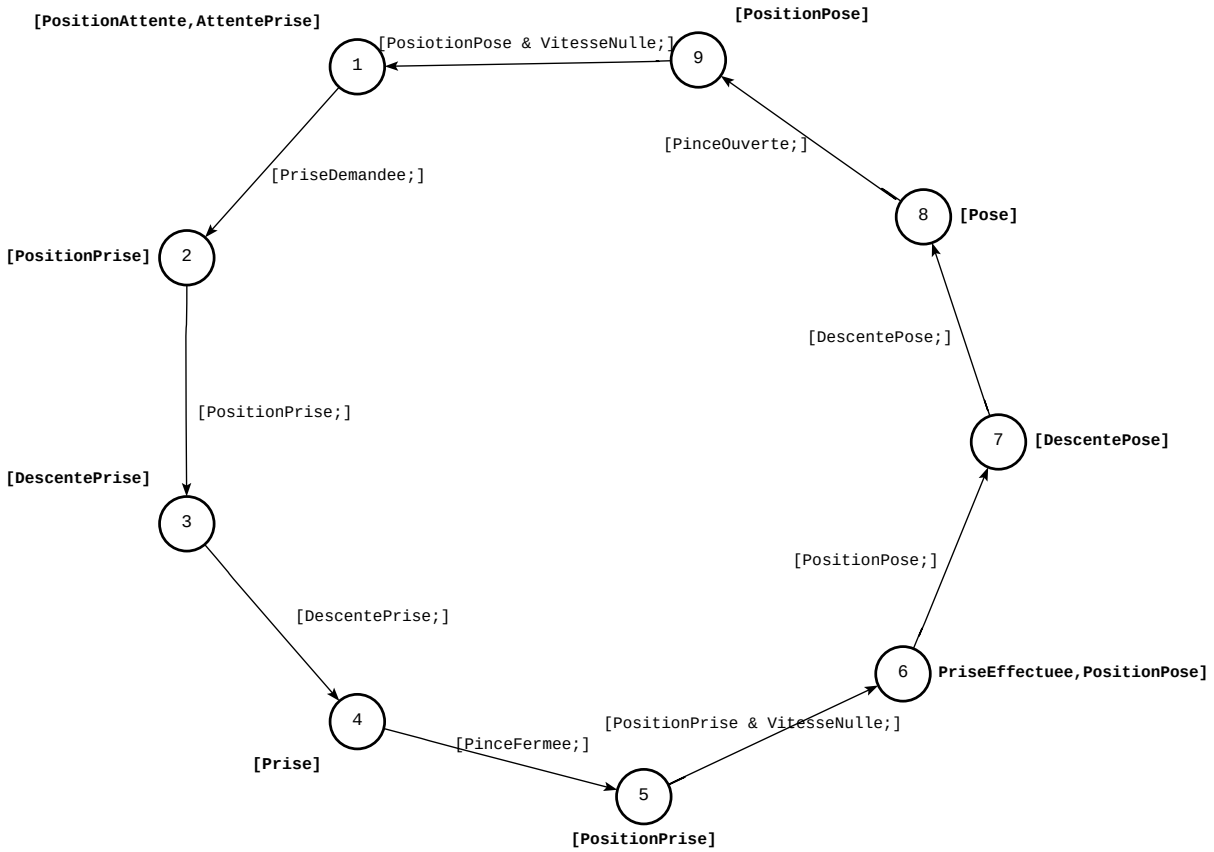

FIGURE 3.2 – Machine à états finis de la commande de chaqu'un des deux bras en interaction avec la ligne transitique

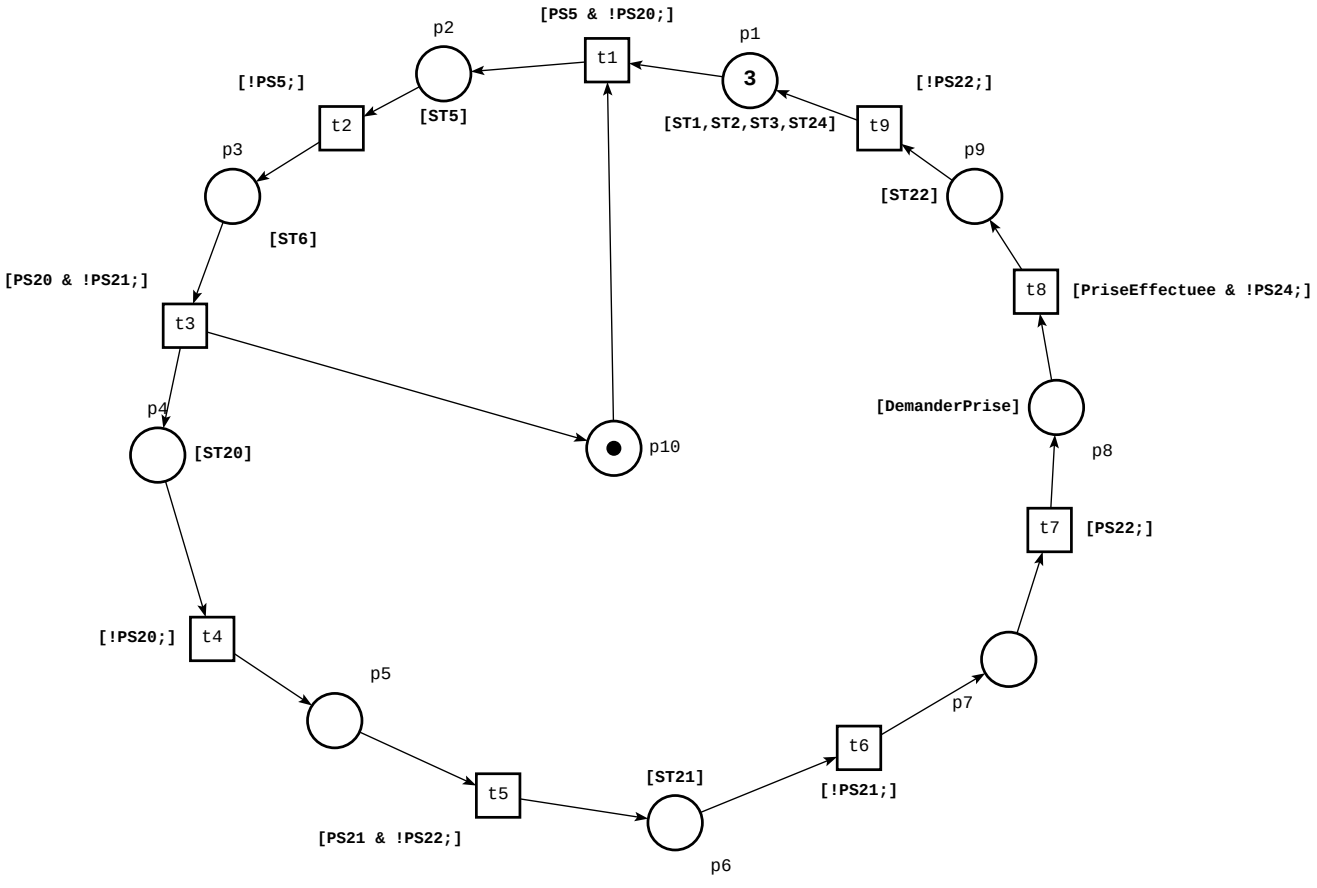

FIGURE 3.3 – Réseau de Petri de la commande de la ligne transitique en interaction avec un bras du robot Baxter

Commande de la ligne transitique en interaction avec les deux bras manipulateurs

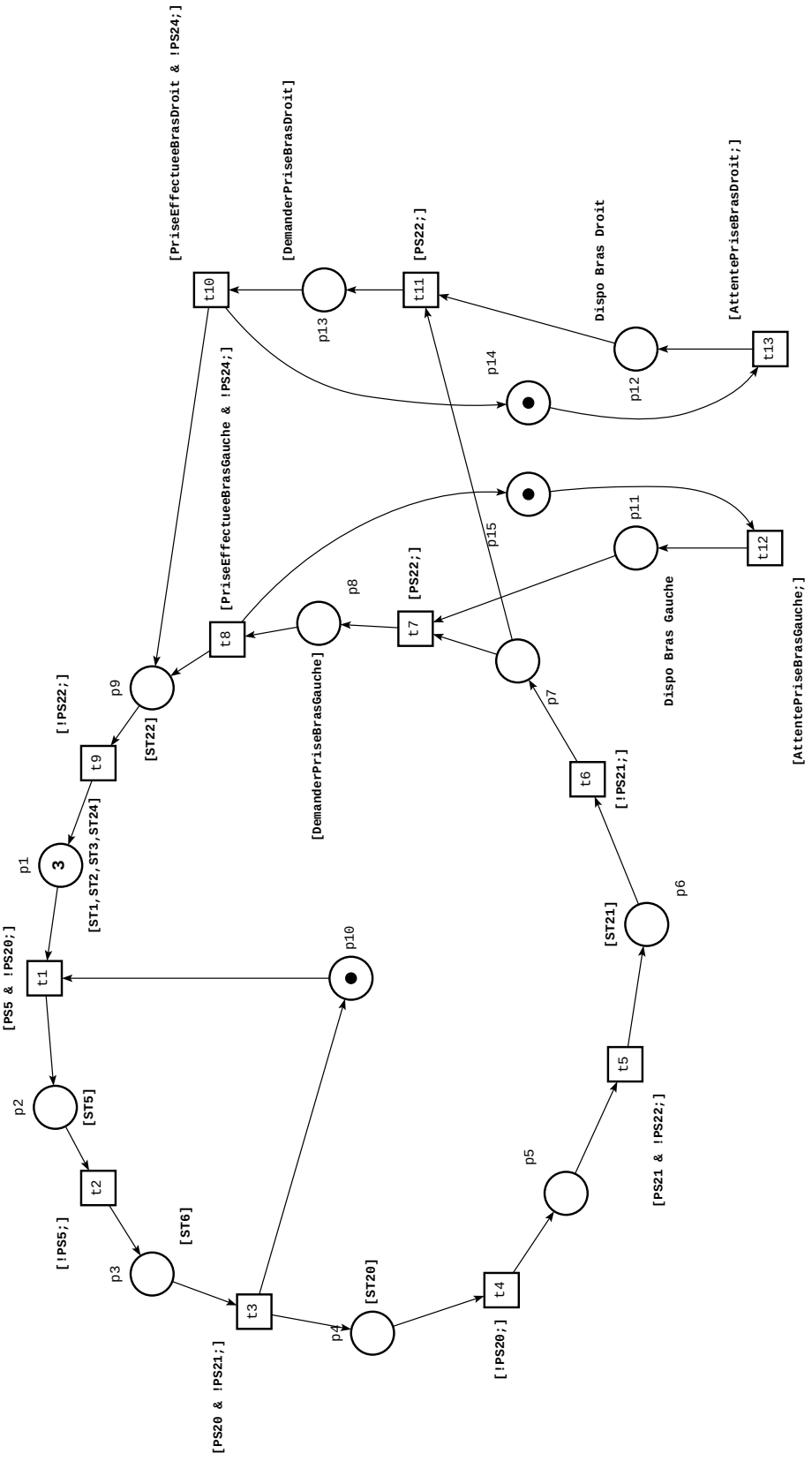

FIGURE 3.4 – Réseau de Petri de la commande de la ligne transitique en interaction avec les deux bras du robot Baxter

Conclusion

Bibliographie

[1] ABIVEN Cédric, CARDONE Grégoire, GAO Shengheng. *Commande d'une ligne transitique MONTRAC*. Rapport de TER, Master 1 Électronique Électrotechnique Automatique, Ingénierie des Sytèmes Temps-Réel. Toulouse : Université Paul Sabatier, 2015.

[2] ANTONIUTTI Emilie, BERTIN Thibault, DEMMER Simon, LE BIHAN Clément. *Commande et Simulation d'un réseau de transport d'un système de production*. Rapport de Projet Long, GEA CDISC. Toulouse : ENSEEIHT, 2016.
Disponible sur <`https://github.com/ClementLeBihan/CelluleFlexible/blob/master/Livrables/Rapport_Projet_Long`> (Consulté le 26.05.2016)

[3] ANTONIUTTI Emilie, BERTIN Thibault, DEMMER Simon, LE BIHAN Clément. *Simulation de la ligne transitique MONTRAC*. Code source (GIT). Toulouse : ENSEEIHT, 2016.
Disponible sur <`https://github.com/ClementLeBihan/CelluleFlexible`> (Consulté le 26.05.2016)

[4] GORRY POLLET Alexandra. *Commande d'une cellule flexible de production robotisée*. Rapport de stage, IUT Génie Électrique et Informatique Industrielle. Toulouse : Université Paul Sabatier, 2015.

[5] DATO Bruno, ELGOURAIN Abdellah, SHULGA Evgeny. *Commande d'une ligne transitique MONTRAC*. Rapport de TER, Master 1 Électronique Électrotechnique Automatique, Ingénierie des Sytèmes Temps-Réel. Toulouse : Université Paul Sabatier, 2016.

[6] AIP-PRIMECA. *Pôle AIP-PRIMECA Toulouse*.
Disponible sur <`http://aip-primeca.ups-tlse.fr/`> (Consulté le 26.05.2016)

[7] Robot Operating System. *ROS*.
Disponible sur <`http://www.ros.org/`> (Consulté le 26.05.2016)

[8] V-REP. *v-rep virtual robot experimentation platform*.
Disponible sur <`http://www.coppeliarobotics.com/`> (Consulté le 26.05.2016)