République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Abderrahmane MIRA-BEJAIA
Faculté de Technologie
Département d'Electrotechnique

En vue de l'obtention du diplôme Master recherche en Électrotechnique.
Option : Automatisme Industriel

MEMOIRE DE FIN DE CYCLE

Thème :

Simulation d'un Automate Programmable Industriel Sous MATLAB

Présenté par :

-Mr. BRAHMI Fateh
-Mr. CHIKHI Wahib

Encadré par :

- Mr. MELAHI Ahmed

Promotion :
2012/2013

DEDICACE

Je dédie ce modeste travail

A ceux qui ont fait de moi un homme

Ma MERE" et "Mon PERE" qui m'ont aidé et soutenu durent tout

Mon cursus d'étude

A mes chers frères

A toute ma famille

A tous mes amis et mes proches

A tous ceux qui m'ont aidé de loin ou de prés durant

Les moments difficiles

B.Fateh

Je dédie ce modeste travail :

A mes très chers parents

A mon cher frère

A toute ma famille

A mon binôme Fateh et sa famille

A tous mes amis

Sans oublier mes amis de la promotion sortante 2013

C.Wahib

Remerciements

Nous remercions Dieu pour le courage, la patience et la santé qui nous ont été utile au long parcours.

Nous tenons à dresser nos vifs remerciement à Mr MELAHI Ahmed qui a proposé et dirigé ce thème et pour son entière disponibilité, son aide et ses précieux conseils.

Notre sincère gratitude va aussi aux membres de jury qui ont bien accepté de juger ce présent travail.

Il nous est particulièrement agréable d'exprimer ici notre reconnaissance envers tous ceux qui ont rendu possible ce travail.

Que tous ceux qui ont contribué de prés ou de loin à notre formation trouvent ici nos sincères remerciements.

SOMMAIRE

TABLE DES MATIERES

TABLE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION GENERALE

INTRODUCTION GENERALE

L'Automate Programmable Industriel (API) ou (PLC) est aujourd'hui le constituant le plus répandu des automatismes **[1]**. On le trouve non seulement dans tous les secteurs de l'industrie, mais aussi dans les services (gestion de parkings, accès à des bâtiments, contrôle du chauffage, de l'éclairage, de la sécurité ou des alarmes). Il répond aux besoins d'adaptation et de flexibilité à de nombreuses activités économiques actuelles.

Il y a un grand nombre de logiciel de simulation d'API sur le marché. Tous les langages sont basés sur des modèles de représentation, certains sont des outils de développement industriel.

Dans notre travail, nous utiliserons un modèle sur l'environnement MATLAB/Simulink qui fait appel au langage (IL) pour programmer et simuler un Automate Programmable Industriel.

Pour la présentation de notre travail, nous avons organisé ce mémoire en trois chapitres.

Le premier chapitre est consacré à la présentation des automates programmables industriels, et aux différents langages de programmation des API.

Dans le deuxième chapitre, on présente un modèle pour la programmation et la simulation des API sous MATLAB, soutenu d'exemples d'applications.

Dans le troisième chapitre, on réalise une interface graphique pour nos applications à l'aide de Guide Matlab. En suite, on réalise nos systèmes avec V-Realm Builder pour les simuler en 3D.

CHAPITRE I

AUTOMATE PROGRAMMABLE INDUSTRIEL

CHAPITRE1

AUTOMATE PROGRAMMABLE INDUSTRIEL

I.1 Introduction

Les premières A.P.I ont été introduits en 1969 aux Etats-Unis pour satisfaire les besoins de l'industrie automobile. Le but recherché était de remplacer les armoires à relais utilisées pour l'automatisation des chaînes de fabrication par des équipements moins coûteux et surtout plus faciles à modifier. Depuis leur apparition, les automates programmables se sont répondus très rapidement dans l'industrie, au point de représente aujourd'hui plus de la moitie des équipements informatiques qui sont utilisés pour ce type d'application **[1]**.

Dans ce présent chapitre, nous allons donner une définition d'un Automate Programmable Industriel (A.P.I) d'une manière générale, sa structure (externe et interne), ses différentes fonctions et langage de programmation ainsi que ces critères de choix.

I.2 Définition

Un automate programmable industriel (A.P.I) est un appareil électronique programmable, adapté à l'environnement industriel, qui réalise des fonctions d'automatisme pour assurer la commande de pré-actionneurs et d'actionneurs à partir d'informations logique, analogique ou numérique.

I.3 Architecture générale

I.3.1 Structure extérieure

Les automates peuvent être de type compact ou modulaire.

I.3.1.1 Type compact

On distinguera les modules de programmation (LOGO de Siemens, ZELIO de Schneider, MILLENIUM de Crouzet ...) des micro-automates.

Il intègre le processeur, l'alimentation, les entrées et les sorties. Selon les modèles et les fabricants, il pourra réaliser certaines fonctions supplémentaires (comptage rapide, E/S analogiques ...) et recevoir des extensions en nombre limité.

Ces automates, de fonctionnement simple, sont généralement destinés à la commande de petits automatismes.

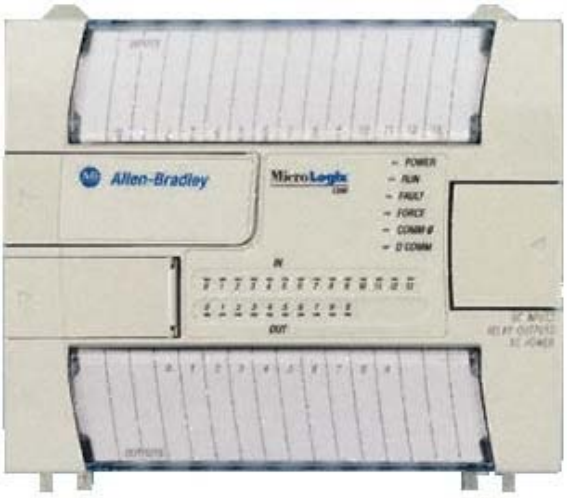
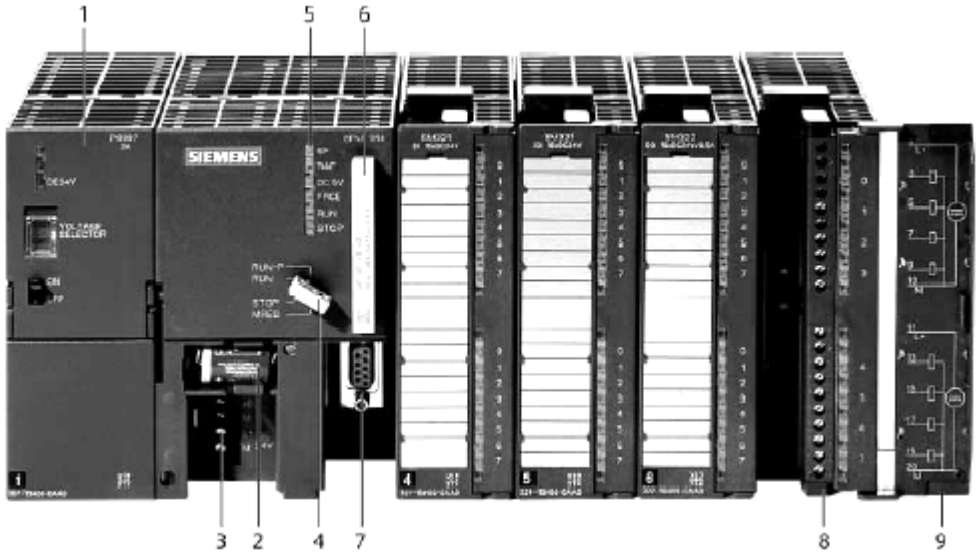


Figure I. 1 : Automate compact (Allen-bradley)

I.3.1.2 Type modulaire

Le processeur, l'alimentation et les interfaces d'entrées/sorties résident dans des unités séparées (modules) et sont fixées sur un ou plusieurs racks contenant le "fond de panier" (bus plus connecteurs). Ces automates sont intégrés dans les automatismes complexes où puissance, capacité de traitement et flexibilité sont nécessaires.



1- Module d'alimentation

2 - Pile de sauvegarde

3 - Connexion au 24V cc

4 - Commutateur de mode (à clé)

5 - LED de signalisation d'état et de défauts

6 - Carte mémoire

7- Interface multipoint (MPI)

8- Connecteur frontal

9- Volet en face avant

Figure I. 2 : Automate modulaire (Siemens).

I.3.2 Structure interne

L'architecture interne des automates programmables diffère d'un constructeur à un autre, et même pour le même constructeur, d'une gamme à une autre, mais en général ils sont constitués essentiellement de:

- une unité centrale
- un module d'entrées /sorties
- un module d'alimentation
- un module de communication
- des auxiliaires

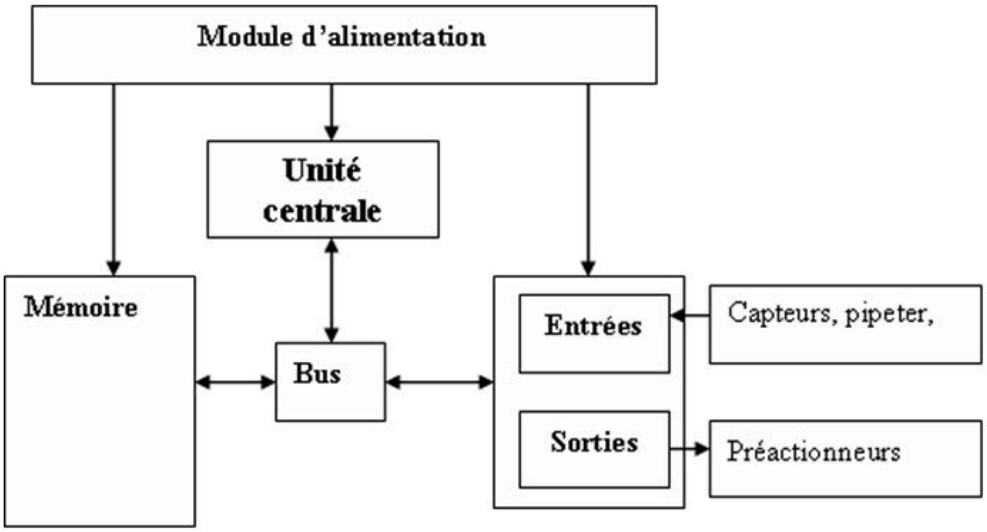


Figure I. 3 : Architecture d'un API

I.3.2.1 L'unité centrale

L'unité centrale (UC) est une carte électronique bâtie autour d'un ou plusieurs processeurs et qui comprend aussi des moyens de stockage servant à sauvegarder les programmes et les données. Elle est le cœur de l'automate et les performances de ce dernier lui sont directement dépendantes **[1]**.

I.3.2.1.1 Le processeur

Les premiers API étaient équipés de processeurs spécifiques à cycle de scrutation unique (on exécutait en permanence un programme gérant essentiellement des variables binaires). On est passé ensuite à des processeurs plus performants, issus du monde de l'informatique. Cette évolution a permis de baisser les coûts et d'accroître les possibilités opérationnelles.

I.3.2.1.2 La mémoire

Le stockage des données et des programmes s'effectue dans des mémoires. Elles sont l'élément fonctionnel qui peut recevoir, conserver et restituer les données. Elles sont découpées en zones où l'on trouve **[2]** :

- La zone mémoire programme (programme à exécuter)

- La zone mémoire des données (état des entrées et des sorties, valeurs des compteurs, temporisations)
- Une zone où sont stockés des résultats de calcul utilisés ultérieurement dans le programme.
- Une zone pour les variables internes.

I.3.2.2 Les entrées / sorties (E/S)

Les entrées et les sorties assurent le rôle d'interface de la partie commande, qui distingue une partie opérative (les sorties), la partie d'acquisitions (les entrées) récupère les informations sur l'état de ce processus et coordonne en conséquence les actions pour atteindre les objectifs prescrits (matérialisés par des consignes).

Les E/S doivent assurer le dialogue de l'Unité Centrale avec les processus externes, qui ne sont pas nécessairement des systèmes électriques ou qui n'ont pas des signaux électrique du même ordre de grandeurs que l'automate, donc il faut traduire les signaux industriels en informations API et réciproquement, avec une protection de l'unité centrale et un traitement adéquats **[3]**.

Les cartes de sortie peuvent être à relais ou à transistor. Celles à relais assurent une coupure entre l'alimentation et le pré actionneur mais sont relativement lentes. Celles à transistor commutent plus rapidement mais n'assurent pas de séparation électrique.

Le nombre total de modules est limité, pour des problèmes physiques (taille du châssis, l'alimentation…), pour des raisons de sécurité, La possibilité de configurer des voies d'accès en entrée ou en sortie est rarement utilisée.

En générale on peut distinguer 3 types d'E/S :

- Les cartes E/S TOR
- Les cartes E/S analogiques
- Les cartes spécialisées

I.3.2.3 Le module d'alimentation

Il est composé de blocs qui permettent de fournir à l'automate l'énergie nécessaire à son fonctionnement.

A partir d'une alimentation en 220 volts alternatif, ces blocs doivent délivrer des sources de tension dont l'automate a besoin : 24V, 12V ou 5V en continu avec de

bonnes performances, notamment face aux microcoupures du réseau électrique. Il est parfois nécessaire d'introduire un transformateur d'isolement et un onduleur, pour lutter contre les perturbations électriques, et éviter les risques de coupure.

Il ne faut pas oublier que les châssis d'extension et les entrées/sorties déportées doivent aussi disposer d'une alimentation.

Un voyant est positionné en générale sur la façade pour indiquer la mise sous tension de l'automate.

I.3.2.4 Le module de communication

Ce module comprend les consoles et les boîtiers de tests. Il existe deux types de consoles. L'une permet le paramétrage et les relevés d'informations (c'est-à-dire la modification des valeurs, et la visualisation), l'autre permet en plus la programmation, le réglage et l'exploitation.

Cette dernière dans la phase de programmation permet :

- l'écriture
- la modification
- l'effacement
- le transfert d'un programme dans la mémoire de l'automate ou dans une mémoire EPROM.

Dans la phase de réglage et d'exploitation elle permet :

- d'exécuter le programme pas à pas
- de le visualiser
- de forcer ou de modifier des données telles que les entrées, les sorties, les bits internes, les registres de temporisation, les compteurs, etc.
- la sortie sur une imprimante du programme si un port de sortie existe.

I.3.2.5 Les auxiliaires

Il s'agit principalement de :

1. Un support mécanique : Il peut s'agir d'un rack, l'automate se présentant alors sous forme d'un ensemble de cartes, d'une armoire, d'une grille et des fixations correspondantes.

2. Un ventilateur : il est indispensable dans les châssis comportant de nombreux modules ou dans le cas où la température ambiante est susceptible de devenir assez élevée (plus de 40 °C).
3. Un indicateurs d'état : il indique la présence de tension, l'exécution du programme (mode RUN), la charge de la batterie, le bon fonctionnement des coupleurs… **[1]**.

I.4 Les langages de programmations

Il existe cinq langages de programmation des automates programmables industriels qui sont répartis et définis en deux grandes familles qui sont les suivantes :

I.4.1 Les langages graphiques

I.4.1.1 Langage SFC (Sequential Function Chart) ou GRAFCET

Le GRAFCET est une méthode de représentation graphique qui décrit les comportements successifs de la partie commande d'un système automatisé, c'est un moyen de description du cahier des charges d'un automatisme, il facilite la communication entre les personnes concernées par l'automatisme, du concepteur a l'utilisateur sans oublier l'agent de maintenance, il impose une démarche formelle hiérarchisée qui permet par analyse préalable de détecter les incohérences et éviter les anomalies au cours du fonctionnement **[3]**.

L'acronyme GRAFCET signifie: Graphe Fonctionnel de Commande Etape Transition (SFC Sequential Function Chart).

a) Les éléments de base du GRAFCET

Le GRAFCET est défini par un ensemble d'éléments graphiques (étapes, transitions, liaisons orientés), traduisant le comportement de la partie commande vis-à-vis de ses entrées et de ses sorties **[3].**

b) Règles d'évolution du GRAFCET :

Règle1 :L'initialisation

Il existe toujours au moins une étape active lors du lancement de l'automatisme. Ces étapes activées lors du lancement sont nommées «Étapes Initiales », elles sont activées inconditionnellement **[3]**.

Règle2 : La validation

Une transition est validée ou non validée. Elle est validée lorsque toutes les étapes immédiatement précédentes sont actives.

Règle3 : Le franchissement

Une transition est franchie lorsqu'elle est validée et que la réceptivité associée à la transition est vraie. Le franchissement entraîne l'activation de toutes les étapes immédiatement suivantes et la désactivation de toutes les étapes précédentes.

Règle4 : Le franchissement de plusieurs transitions

Plusieurs transitions simultanément franchissables sont simultanément franchies.

Règle5 : Priorité de l'activation

Si au cours du fonctionnement, une même étape doit être activée et désactivée simultanément, elle reste activée.

I.4.1.2 Langage LD (Ladder Diagram)

Le langage LD (Ladder Diagram) est une représentation graphique qui traduit directement des équations booléennes combinant des contacts (en entrée) et des relais (en sortie) en un schéma électrique avec des symboles particuliers **[4].**

On peut résumer les différents types des composants graphiques et leurs fonctions dans le tableau (voir l'annexe) **[5].**

a) Les relais

Les relais sont des bons éléments graphiques qui se trouvent dans la zone des actions du programme, et ils sont représentés par une cellule graphique.

Les relais peuvent être utilisés dans un diagramme sont représentés dans un tableau (voir l'annexe) **[6] .**

b)Les éléments de liaisons

Les éléments de liaisons sont des éléments graphiques qui sont utilisés pour réaliser la connexion entre les différents éléments (contacts, relais,…) **[5].**

Noms	Symbole	Fonctions
Connexion horizontale	———	Relie en série les actions graphiques et les tests avec les bars de potentiel
Connexion verticale	|	Relie en parallèle les actions graphiques

Tableau I. 1: Les liaisons en langage LD

c) Barre d'alimentation

Un diagramme LD est limitée sur la gauche et la droite par des lignes verticales appelées respectivement barre d'alimentation à gauche et barre d'alimentation à droite.

Les symboles du diagramme LD sont reliés entre eux et aux barres d'alimentation par des arcs de liaisons verticales ou horizontales. Chaque segment de liaison peut prendre l'état booléen FALSE (0) ou TRUE (1) **[7]**.

I.4.1.3 Langage FBD (function block daigram)

C'est un langage graphique qui permet la construction d'équations complexes à partir des opérateurs standards, ou de blocs fonctionnels **[7].**

Ce langage se compose de réseaux de fonctions préprogrammées ou non, représentées par des rectangles. Ces blocs fonctionnels sont connectés entre eux par des lignes, le flux des signaux se fait de la sortie (à droite) d'une fonction vers l'entrée à gauche de la fonction raccordée.

Exemples de blocs fonctionnels standards (fourni par le constructeur de logiciel):

- Bloc fonctionnel compteurs
- Bloc fonctionnel temporisateurs
- Bloc fonctionnel PID…

Les variables d'entrée et de sortie sont connectées aux boîtes fonctions par des arcs de liaison et les entrées sont à gauche des blocs ainsi que les sorties sont représentées à droite des blocs ces même sorties peuvent être connectées sur des entrées des autres boîtes.

La fonction complète représentée par un programme FBD est construite à l'aide de boîtes fonctions élémentaires. Chaque boîte fonction élémentaire, représentée par un rectangle, a un nombre prédéfini de points de connexion en entrée et en sortie. Elle réalise une fonction élémentaire entre ses entrées et ses sorties. Le nom de la fonction réalisée est inscrit dans le rectangle de la boîte **[7].**

I.4.2 Les langages textuels

I.4.2.1 Langage ST (Structured Text)

Le langage ST est un langage textuel de haut niveau dédié aux applications d'automatisation. Il est utilisé principalement pour décrire les procédures complexes et difficilement modélisables avec les langages graphiques et peut aussi travailler en tant que sous programme avec d'autre langage de programmation par exemple avec le langage LD.

C'est le langage par défaut pour la programmation des actions dans les étapes et des conditions associées aux transitions du langage GRAFCET.

Un programme ST est une suite d'instructions qui se termine par des point-virgule, les noms utilisés dans le code source (identificateurs de variables, constantes, mots clés du langage...) sont délimités par des séparateurs passifs (Espace , tabulation et de fin de ligne) ou des séparateurs actifs, qui ont un rôle d'opérateur, des commentaires peuvent être librement insérés dans la programmation, les séparateurs passifs peuvent être insérés entre les séparateurs actifs, les expressions constantes et les identificateurs **[7].**

Les expressions ST combinent des opérateurs et des opérandes variables ou constants, pour chaque expression simple, le typage des opérandes doit être cohérent.

Les parenthèses sont utilisées pour isoler des parties d'une expression et donner explicitement un ordre d'évaluation des opérations. Quand aucune parenthèse n'est insérée, l'ordre d'évaluation est donné par la priorité entre les opérateurs.

Les types d'énoncés standard du langage ST sont :

— assignation (variable := expression;).

— appel de fonction.

— appel de bloc fonctionnel.

— énoncés de sélection (IF, THEN, ELSE, CASE).

— énoncés d'itération (FOR, WHILE, REPEAT).

— énoncés de contrôle (RETURN, EXIT).

— énoncés spéciaux pour le lien avec le langage GRAFCET.

— opérateurs booléens (NOT, AND, OR et XOR)

I.4.2.2 Langage IL (Instruction List)

C'est un langage textuel de bas niveau, ce langage utilise un jeu d'instruction simple. Il est adapté aux applications de petite taille. Les instructions opèrent toujours sur un résultat courant (ou registre IL) , l'opérateur indique le type d'opération à effectuer entre le résultat courant et l'opérande, le résultat de l'opération est stocké à son tour dans le résultat courant.

Les programmes dans ce langage peuvent être traduit ou déduit des autres langages (Ex : LD)

Un programme IL est une liste d'instructions qui doivent commencer par une nouvelle ligne, et doivent contenir un opérateur, complété éventuellement par des modificateurs et si c'est nécessaire pour l'opération, un ou plusieurs opérandes, séparés par des virgules (',').

Une étiquette suivie de deux points (':') peut précéder l'instruction, et si un commentaire est attaché à l'instruction, il doit être le dernier élément de la ligne.

Des lignes vides peuvent être insérées entre des instructions, un commentaire peut être posé sur une ligne sans instruction **[7].**

Exemples

Étiquette	Opérateur	Opérande	Commentaire
Début:	**LD**	**IX1**	**(* bouton poussoir *)**
	ANDN	**MX5**	**(* commande valide *)**
	ST	**QX2**	**(* lance moteur *)**

Tableau I. 2 : La syntaxe du langage IL

a) Les étiquettes

Une étiquette suivie du caractère ' **:** ' peut précéder une instruction et elle peut aussi être sur une ligne sans instruction. Elles sont utilisées comme opérandes par certaines instructions telles que les sauts **[7].**

Leur nomenclature doit respecter les règles suivantes :

— la longueur du nom ne doit pas excéder 16 caractères.

— le premier caractère doit être une lettre.

— les caractères suivants doivent être des lettres, des chiffres ou '_'.

— une étiquette peut avoir le même nom qu'une variable.

b) Modificateurs d'instructions

Le caractère modificateur complète le nom de l'instruction, sans aucun caractère de séparation **[7].**

Voici la liste des modificateurs autorisés pour les instructions du langage IL :

- 'N' inversion booléenne de l'opérande : Le modificateur 'N'indique que l'opérande doit être inversé avant d'être utilisé par l'instruction

Par exemple : ANDN IX12 est équivalente à AND NOT(IX12)

- '(' opération différée : Parce que le langage IL ne traite qu'un seul registre (résultat courant), certaines opérations doivent être différées, pour changer l'ordre naturel d'exécution des instructions. Les parenthèses sont utilisées pour représenter les opérations différées. La modificatrice parenthèse ouvrante '(' indique que l'évaluation de l'instruction doit être différée jusqu'à la prochaine instruction parenthèse fermante ')'

- 'C'opération conditionnelle : Le modificateur 'C'indique que l'instruction ne doit être exécutée que si le résultat courant a la valeur booléenne TRUE (ou différent de 0 pour une valeur non booléenne). Le modificateur 'C' peut être combiné avec l'opérateur 'N' pour indiquer que l'instruction ne doit être exécutée que si le résultat courant vaut FALSE (ou 0 pour une instruction non booléenne).

c) Les opérateurs

Le tableau qui résume l'ensemble des opérateurs et leurs rôles dans le programme du langage IL (voir l'annexe).

I.5 Cycle d'exécution d'un programme

Lorsque l'A.P.I est en fonctionnement, c'est-à-dire, lorsqu'il exécute son programme de contrôle sur le système extérieur, une série d'opérations effectuée de façon séquentielle et répétitive.

I.5.1 Acquisition d'entrées

Au début de chaque cycle, l'automate programmable examine l'état de tous les signaux d'entrées et puis procède à leur écriture dans la mémoire image des entrées **[3]**.

I.5.2 Traitement des données

L'unité centrale lit successivement les instructions dans la mémoire interne. Ensuite, elle procède au traitement des instructions et évaluation des grandeurs de sorties. Une fois ces valeurs sont calculées, elles sont stockées en mémoire image des sorties **[3]**.

I.5.3 Affectation des sorties

Après l'exécution du programme, l'automate procède à la lecture des sorties dans la mémoire image de ces dernières, et puis les transferts vers les modules de sorties **[3]**.

I.6 Développement d'un projet sur un API

Avant d'entamer un projet sur un A.P.I. il faut être méthodique pour développer une application complexe en technologie. La démarche suivie pour réaliser un projet sur un A.P.I. s'apparente d'avantage à la méthodologie pratiquée sur ordinateur qu'aux procédures utilisées en logique câblée.

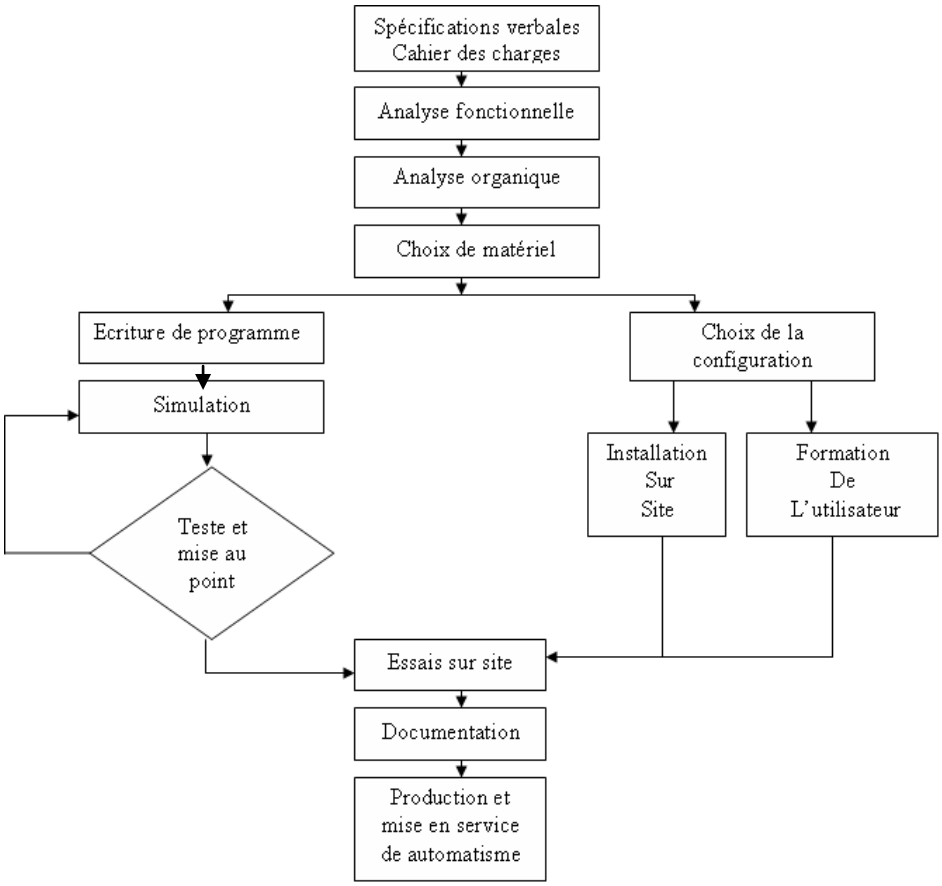


Figure I. 4 : Organigramme pour développé un projet sur un A.P.I.

I.7 Critères de choix d'un automate

Un automate utilisant des langages de programmation de type GRAFCET est préférable pour assurer les mises au point et dépannages dans les meilleures conditions.

La possession d'un logiciel de programmation est aussi une source d'économies (achat du logiciel et formation du personnel). Des outils permettant une simulation des programmes sont également souhaitables.

Il faut ensuite quantifier les besoins :

- Nombre d'entrées/sorties : le nombre de cartes peut avoir une incidence sur le nombre racks dès que le nombre d'entrées/sorties nécessaires devient élevé.
- Type de processeur : la taille mémoire, la vitesse de traitement et les fonctions spéciales offertes par le processeur permettront le choix dans la gamme souvent très étendue.
- Fonctions ou modules spéciaux : certaines cartes (commande d'axe, pesage…) permettront de "soulager" le processeur et devront offrir les caractéristiques souhaitées.

- Fonctions de communication : l'automate doit pouvoir communiquer avec les autres systèmes de commande (API, supervision…) et offrir des possibilités de communication avec standard normalisés (profibus…).

I.8 Conclusion

Dans ce chapitre nous avons présenté l'automate programmable industriel qui est destiné à être utilisé dans un environnement industriel. Nous avons aussi définit dans ce chapitre les différents langages de programmation qui peuvent être utilisés pour la programmation des automates programmables industriels et qui sera la suite de notre travail**.**

.

CHAPITRE II

MODELE API SOUS MATLAB

CHAPITRE 2

MODELE API SOUS MATLAB

II.1 Introduction

MATLAB est un système interactif et convivial de calcul numérique et de visualisation graphique destiné aux ingénieurs et scientifiques. Il possède un langage de programmation à la fois puissant et simple d'utilisation. Il permet d'exprimer les problèmes et solutions d'une façon aisée.

Dans ce chapitre nous présentons un modèle pour la programmation et la simulation des automates programmables industriels (API).

II.2 Présentation du modèle API sous MATLAB

Pour la simulation de l'automate programmable industriel (API) nous avons adopté le modèle schématisé dans la figure ci-après :

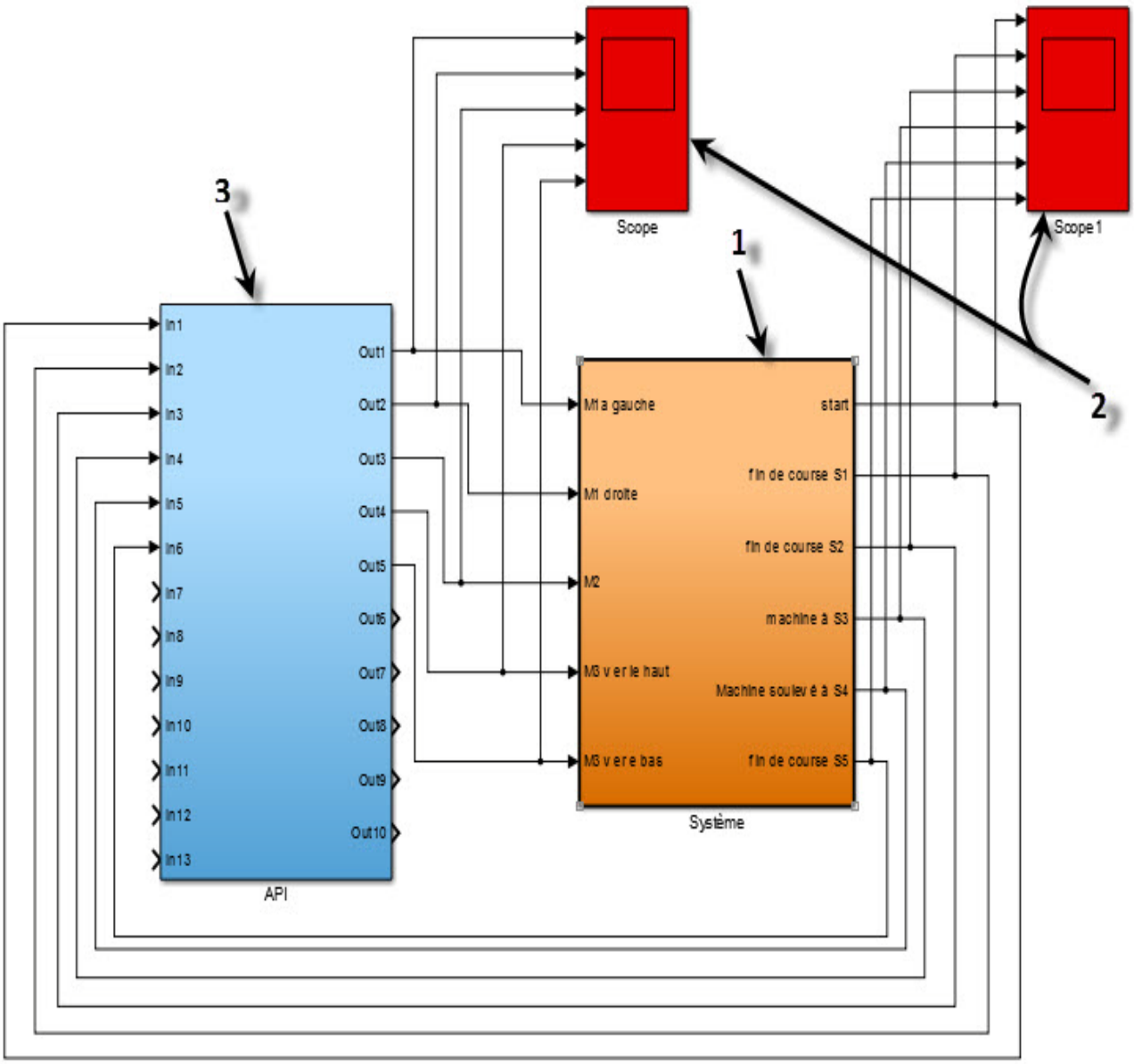


Figure II. 1 : Modèle de contrôle d'un système avec API

1- Bloc système : sert à la modélisation des capteurs à partir des signaux d'actionneurs.

2- Scope : permet la visualisation des signaux d'entrées /sorties

3- Bloc API : représente l'automate programmable industriel qui contient un multiplexeur, un démultiplexeur et une fonction MATLAB.

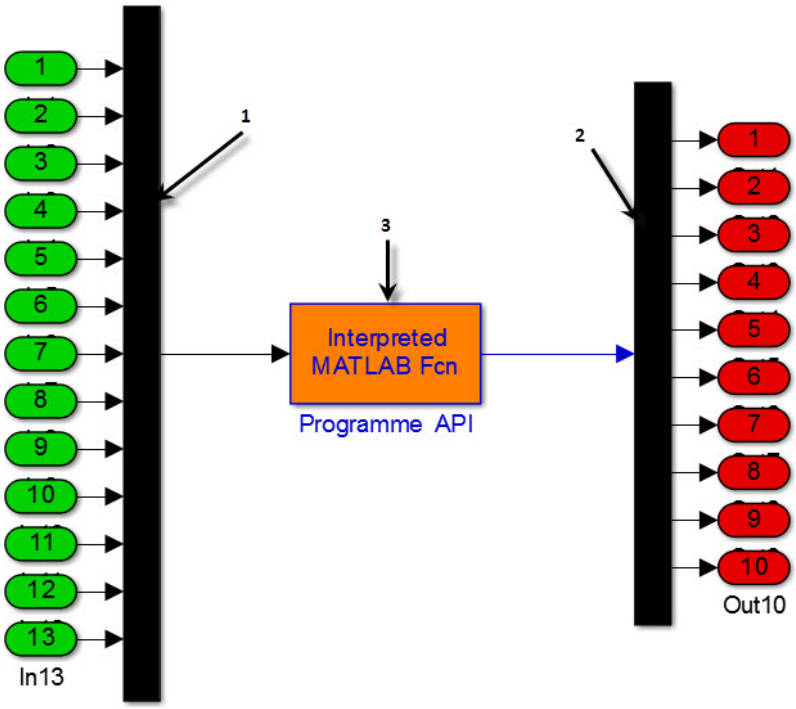


Figure II. 2 : Bloc API

1- Démultiplexeur : présente l'entrée de l'automate programmable industriel.

2- Multiplexeur : il présente la sortie de l'automate programmable industriel.

3- Bloc Matlab Function : contient un fichier script qui fait appel à un programme généré à partir d'un Grafcet (équations logiques du Grafcet).

La structure du programme de l'automate est illustrée par la figure suivante :

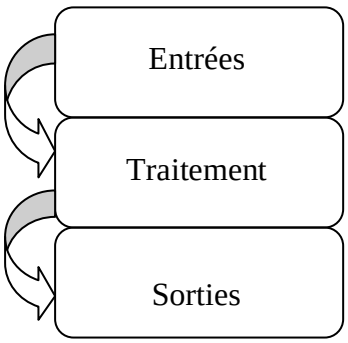


Figure II. 3 : Structure du programme API

- Entrées : cette partie du programme est toujours fixe quelque soit le système elle contient la fonction global pour que les variables (entrées, sorties, états logiques) soient reconnus dans l'environnement MATLAB ainsi que l'initialisation de ces derniers.

- Traitement : il contient le programme à traiter et varie d'un système à l'autre, on le construit à partir des équations logiques du système étudié.
- Sorties : cette partie du programme est toujours fixe quelque soit le système. Elle contient les équations d'affectation des sorties du système avec les sorties d'API.

Le bloc API, est la base de la méthodologie proposée. Il reproduit le cycle de l'opération de l'automate. Afin de construire ce bloc, les étapes suivantes doivent être suivies **[15]**:

1- Supposer que le procédé est déjà modélisé dans l'environnement MATLAB/Simulink ;
2- Considérer les spécifications fonctionnelles de l'API, comme étant des procédés contrôlables ;
3- Considérer que l'API est celui qui contrôle le processus ;
4- Elaborer le programme de contrôle d'API en respectant les spécifications fonctionnelles exemple Grafcet ;
5-Ecrire le programme de contrôle d'API en utilisant l'un des langages de programmation ;
6- Enregistrer le programme d'API écrit ;
7- Traduire le programme enregistré dans un programme en MATLAB qui sera un fichier m-file reconnu dans l'environnement MATLAB/Simulink ;
8- Tester le programme développé;
9- Elaborer les corrections nécessaires.

Pour établir le programme traduit (étape 7) il faudra prendre en compte les facteurs suivants :

- Type d'API ;
- Nombre d'entrées-sorties de l'automate ;
- Le fichier du programme d'API.

Type d'API

Pour élaborer les règles de conversion des langages API ver langage MATLAB il faut bien choisir le type d'API. Chaque constructeur d'API développe sa propre syntaxe de

programmation. Par cette méthode le module de traduction devrait connaitre le constructeur d'API afin d'appliquer les règles de traduction adéquates.

Le nombre d'entrées/sorties d'API

Le nombre d'entrées/sorties de l'API définit clairement les arguments du programme de contrôle d'API **[14].** La fonction de MATLAB/Simulink sera responsable d'exécuter le programme d'API dans l'environnement MATLAB/Simulink, et elle sera écrite sous la syntaxe suivante :

Function [Sortie]= Programme API (di1,…, din,… ai1,… aim)
……
(Programme de contrôle d'API en langage Matlab)
…….
Sortie = [do1,…, dop,…, ao1,… aoq]:

Le programme de l'API

Le programme de contrôle d'API peut être écrit dans cinq langages de programmation définis dans chapitre 1 section 4.

Dans cette section nous nous intéressons au langage IL (liste d'instruction) et nous définissent les équivalences entre ce langage et le langage MATLAB :

Instruction Booléenne	Instruction en IL	Instructions en MATLAB
ET	LD I 0.0 A I 0.1 = Q 0.0	do_1 = di_1&di_2
OU	LD I 0.2 O I 0.3 = Q 0.2	do_3 = di_3 | di_4
NON	LDN I 0.2. = Q 0.0	do_1 = ~ di_3
Combinaisons de divers instructions Booléenne	LD I 0.0 A I 0.1 LD I 0.2 OLD = Q 0.0	do_1 = (di_1 & di_2)| di_3

Tableau II.1 : Instructions booléennes

Instructions	Instructions en IL	Instructions en MATLAB
COMPTEUR	LD I 0.0// compte vers le haut LD I 0.1// compte vers le bas LD I 0.2// (Reset) CTUD C10, +4	if di_1 & ~ di_1_prev c_10_value = c_10_value + 1 end if di_2 & ~ di_2 c_10_value = c_10_value - 1 end if di_3 c_10_value = 0 c_10 = 0 End if c_10_value >= +4 c_10 = 1 End
TEMPORISATEUR	LD I 2.0 TON T33, 3	%Start Timer if di_20 & t33_start==0 t33_start=3 ; t33_end_time=clock_in+3 ; end %Timer ON if t33_start &clock _in>=t33_end_time t33_bin=1 ; end %Timer OFF if ~di_20 t33_bin=0 ; t33_start=0 ; end

Tableau II.2 : Instructions (Temporisateur- Compteur)

Instructions	Instructions en IL	Instructions en MATLAB
INCREMENTATION	LD I 0 .0 +IAIW 0, AQW0	if di_1 a0_1= ai_1 + ao_1 end
MULTIPLICATION	LD I 0 .0 MOVW +6 , VD4 MUL +9 , VD4	if di_1 v_4 = +6 v_4 = +9 * v_44 end

Tableau II.3 : Instructions (Incrémentation- Multiplication)

Ce modèle est une solution pour la programmation et la simulation des automates programmables industriels (API), qui nous permet de visualiser les signaux d'entrées/sorties de notre système ainsi que la simulation réelle en 3D.

Pour la suite de ce chapitre on va utiliser trois modèles d'applications. Le premier et le deuxième concernent les vérins et le troisième traite la tronçonneuse de découpage.

II.3 Exemples d'applications

Pour cette partie nous exploitons le modèle d'automate décrit précédemment pour automatisé trois systèmes :

II.3.1 Deux vérins doubles effets :

II.3.1.1 Application 1

Pour ce premier modèle d'application, nous voulons commander deux vérins doubles effets (V1) et (V2) sellons le cycle suivant :

Sur impulsion de mise en marche, (V1) sort, puis (V2) sort, en suite (V1) rentre, enfin (V2) rentre.

Le GRAFCET de ce système comprendra cinq étapes (une étape d'attente et quatre étapes d'actions). Les réceptivités des transitions entre étapes seront des indications sur l'état de notre système.

Pour passer au GRAFCET niveau 2, il faut préciser les choix technologiques : les détecteurs de positions des tiges des vérins sont repérés par des fins de course (a0, a1 pour V1 et b0, b1 pour V2).

Le GRAFCET de ce système est le suivant :

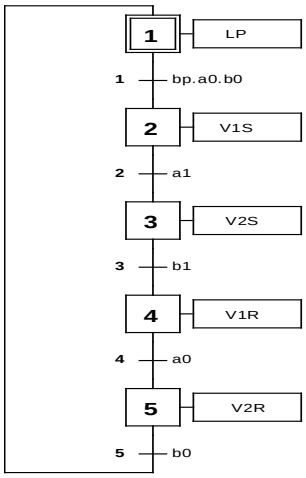


Figure II. 4 : GRAFCET du système

Les équations du GRAFCET

✓ Les entrées du système sont :

bp : bouton poussoir

a1 : capteur de fin de course (sortie de vérin1)

b1 : capteur de fin de course (sortie de vérin2)

a0 : capteur de fin de course (rentré du vérin1)

b0 : capteur de fin de course (rentré d vérin2)

✓ Les sorties du système :

V1S : vérin 1 sort

V1R : vérin 1 rentre

V2S : vérin 2 sort

V2R : vérin 2 rentre

LP : lampe

✓ Les équations des réceptivités :

r1=bp.a0.b0

r2=a1

r3=b1

r4=a0

r5=b0

✓ Les équations d'activation et de désactivation :

S1=Init+b0.X5

R1=X2

S2=X1.bp.a0.b0

R2=X3+Init

S3=X2.a1

R3=X4+Init

$$\begin{cases} S4=X3.b1 \\ R4=X5+Init \end{cases}$$

$$\begin{cases} S5=X4.a0 \\ R5=X1+Init \end{cases}$$

- ✓ Les équations des états :

  X1= (S1+X1).$\overline{R1}$

  X2= (S2+X2).$\overline{R2}$

  X3= (S3+X3).$\overline{R3}$

  X4= (S4+X4).$\overline{R4}$

  X5= (S5+X5).$\overline{R5}$

- ✓ Les équations des sorties :

  V1S=X2

  V2S=X3

  V1R=X4

  V2R=X5

  LP=X1

A partir du GRAFCET et les équations logiques, nous avons introduit le programme en langage MATLAB dans le modèle, comme le montre la figure suivante :

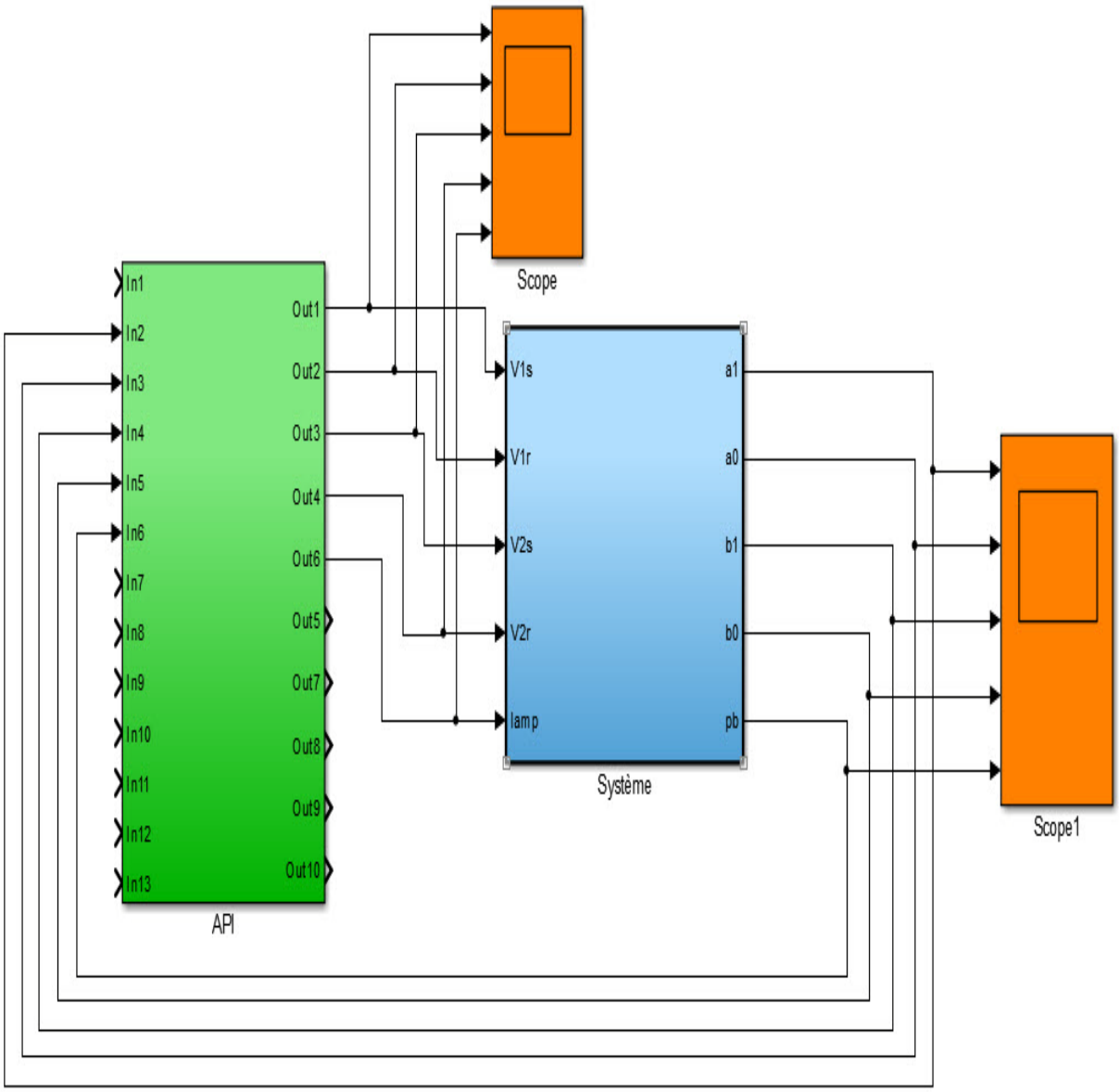


Figure II. 5 : Modèle d'application pour les deux vérins

```
%Traitement
m0=(m4 & di5)|(m0&~m1)|(~m0&~m1&~m2&~m3&~m4);
m1=(m0 & di6 & di3 & di5)|(m1&~m2);
m2=(m1 & di2) | (m2 & ~m3);
m3=(m2 & di4) | (m3 & ~m4);
m4=(m3 & di3) | (m4 & ~m0);
```

Après simulation, nous avons obtenu les résultats suivants :

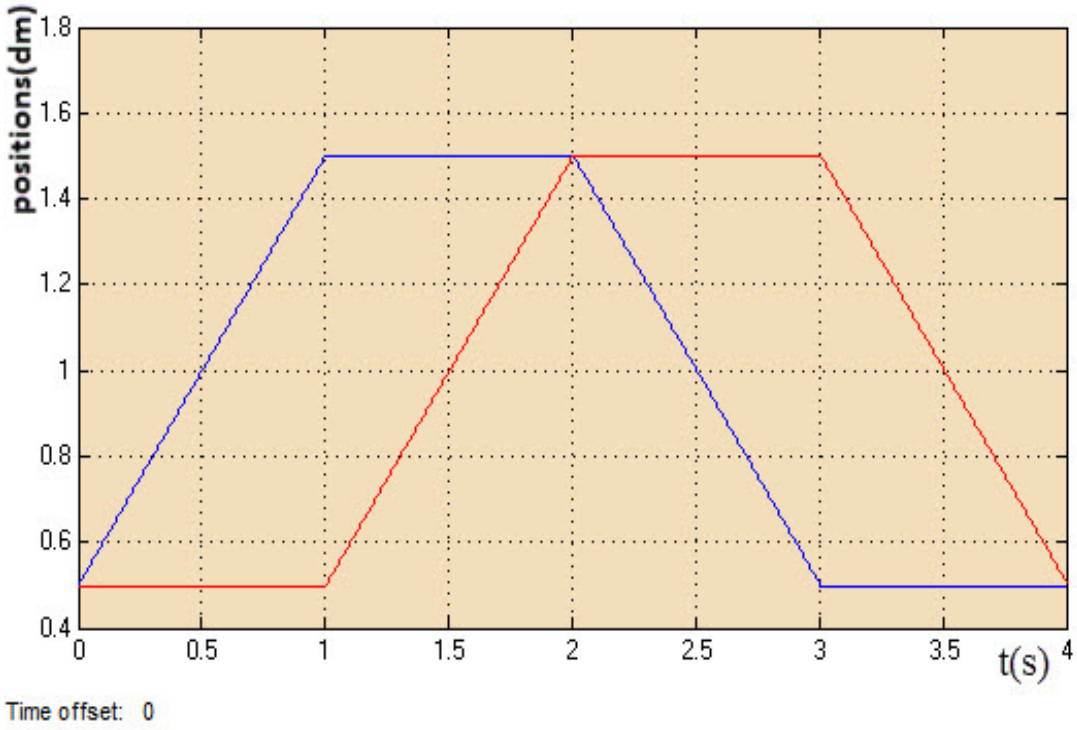


Figure II.6 : Les positions des deux vérins

Interprétation des résultats

En appuyant sur le bouton poussoir (bp), le vérin (V1) sort a son arrivé à la position (1.5) le capteur (a1) se déclenche et (V1) s'arrête, puis (V2) sort lui aussi a son arrivé à la position (1.5) le capteur (b1) se déclencher et (V2) s'arrête, en suite (V1) rentre jusqu'à l'arrivé à sa position initiale (position 0.5) le capteur (a0) se déclenche, enfin V2 rentre a sa position initiale où le capteur (b0) va se déclenché.

II.3.1.2 Application 2

Pour cet exemple d'application, nous reprenons l'exemple (application 1) et nous changerons quelques instructions dans le programme pour avoir le cycle suivant :

Sur impulsion de mise en marche, (V1) sort, puis (V1) rentre, en suite (V2) sort, enfin (V2) rentre.

Le GRAFCET du système est le suivant :

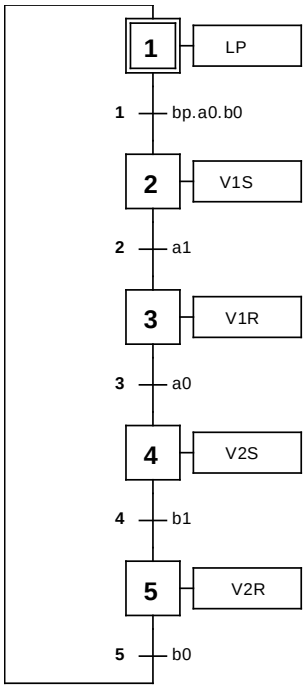


Figure II.7 : GRAFCET des vérins

Les équations du GRAFCET

✓ Les équations des réceptivités :

r1=bp.a0.b0

r2=a1

r3=a0

r4=b1

r5=b0

✓ Les équations d'activation et de désactivation :

S1=Init+b0.X5

R1=X2

S2=X1.bp.a0.b0

R2=X3+Init

S3=X2.a1

R3=X4+Init

S4=X3.a0

R4=X5+Init

S5=X4.b1

R5=X1+Init

- ✓ Les équations des états des étapes :

  X1= (S1+X1).$\overline{R1}$

  X2= (S2+X2).$\overline{R2}$

  X3= (S3+X3).$\overline{R3}$

  X4= (S4+X4).$\overline{R4}$

  X5= (S5+X5).$\overline{R5}$

- ✓ Les équations des sorties :

  V1S=X2

  V1R=X3

  V2S=X4

  V2R=X5

A partir des équations logiques du GRAFCET nous avons introduit le programme en langage MATLAB dans le modèle comme le montre la figure suivante :

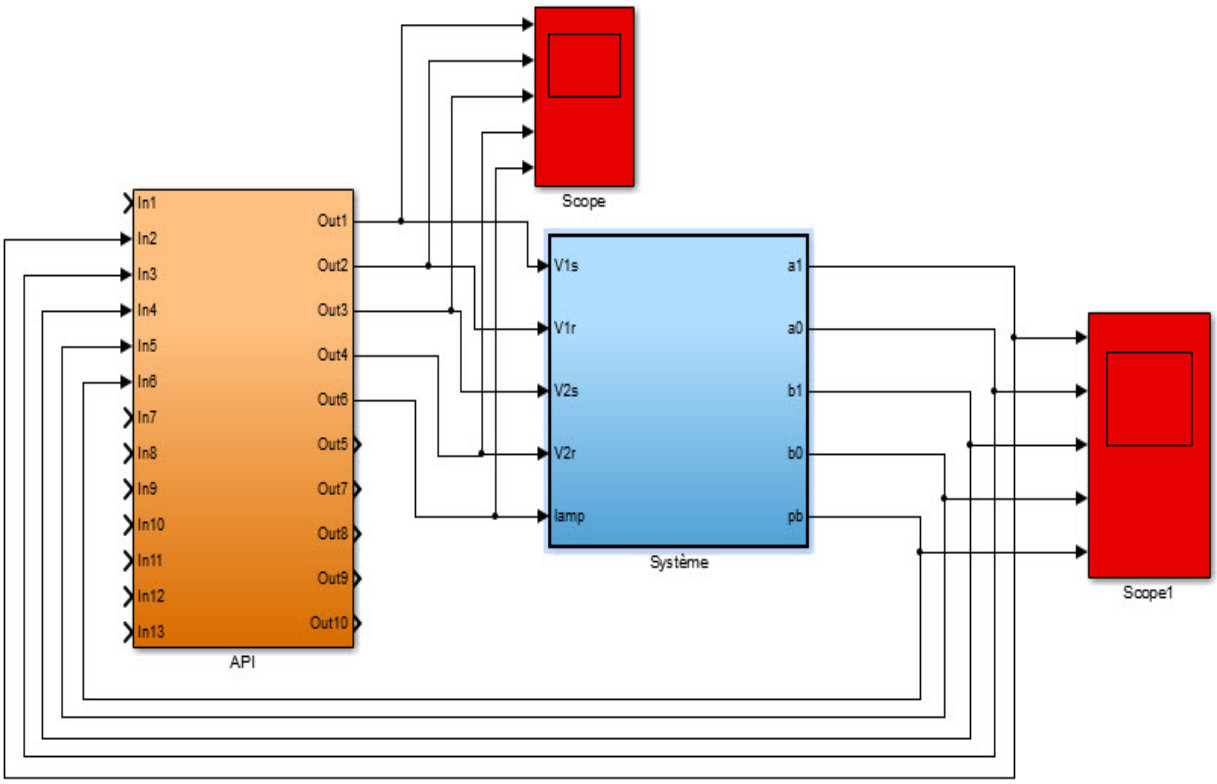


Figure II.8 : Modèle d'application

```
%Traitement
m0=(m4 & di5)|(m0&~m1)|(~m0&~m1&~m2&~m3&~m4);
m1=(m0 & di6 & di3 & di5)|(m1&~m2);
m2=(m1 & di2) | (m2 & ~m3);
m3=(m2 & di3) | (m3 & ~m4);
m4=(m3 & di4) | (m4 & ~m0);
```

Après simulation, nous avons obtenu les résultats suivants :

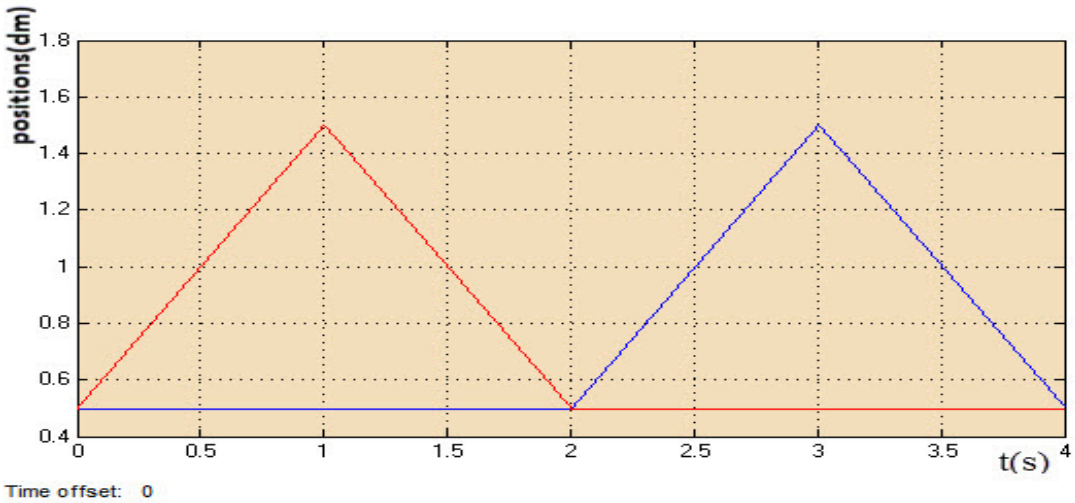


Figure II. 9 : Les positions des deux vérins

Interprétation des résultats

En appuyant sur le bouton poussoir (bp), le vérin (V1) sort jusqu'à l'arrivé au capteur (a1), puis il rentre jusqu'à l'arrivé au capteur (a0) où il s'arrêt, en suite (V2) sort jusqu'à l'arrivé au capteur (b1), enfin il rentre jusqu'à l'arrivé au capteur (b0) où il s'arrêt.

II.3.2 Tronçonneuse de bois

Pour ce troisième modèle d'application nous voulons automatiser une tronçonneuse de découpage suivant ce cycle :

Après avoir appuyé sur le bouton de démarrage la machine de découpage se déplace vers la droite. À l'arrivée au capteur S1, le moteur2 fait tourner le disque. Lorsque la machine arrive au capteur S2, elle se met à monter jusqu'à atteinte la position haute .En ce moment la machine s'arrête et se déplace ver la gauche jusqu'au retour à la position initiale ou la lame devrait s'abaisser.

Pour cela nous avons utilisé des capteurs de fin de course notés (S1, S2, S3, S4, S5) pour détecter les positions de la machine comme le montre la figure suivante :

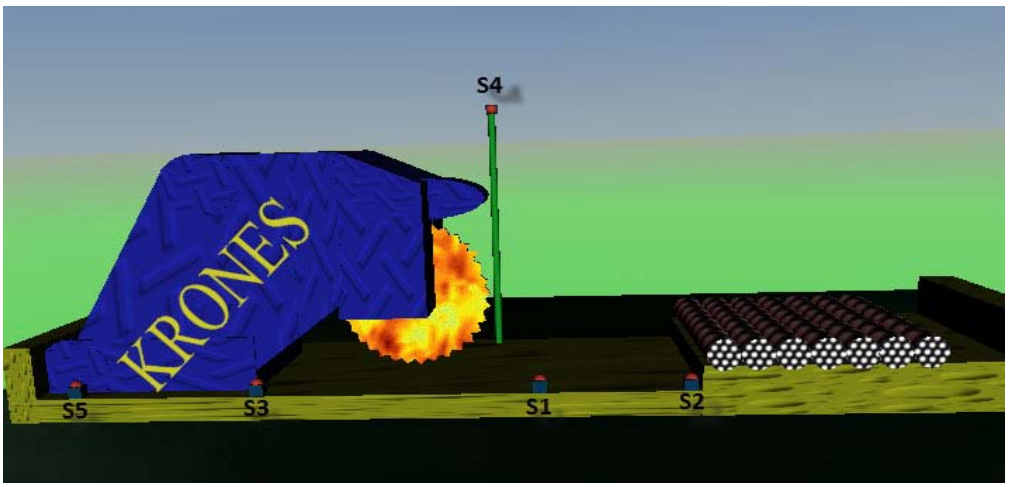


Figure II. 10 : Schéma de la tronçonneuse

Le GRAFCET de ce système est le suivant :

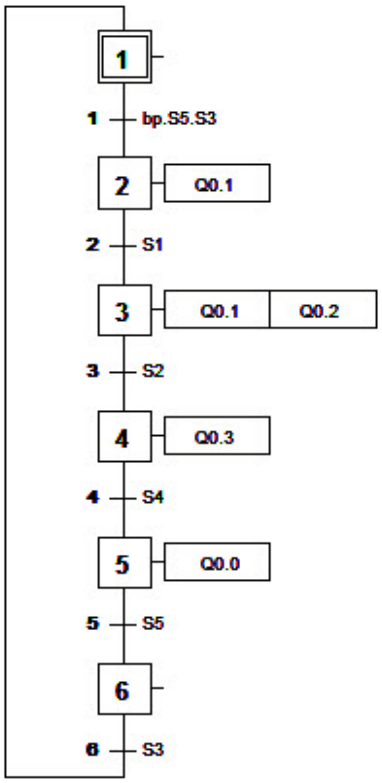


Figure II. 11 : GRAFCET de la tronçonneuse

- **Les équations du GRAFCET**

✓ Les entrées du système sont :

bp(I0.0) : bouton poussoir

S1(I0.1) : capteur de fin de course (machine commence a coupé)

S2(I0.2) : capteur de fin de course (machine à droite)

S3(I0.3) : capteur de fin de course (machine en bas)

S4(I0.4) : capteur de fin de course (machine soulevé)

S5(I0.5) : capteur de fin de course (machine à l'état de dépare)

✓ Les sorties du système sont :

O0.0 : machine à gauche (M1)

O0.1 : machine à droite (M2)

O0.2 : moteur2 tourne

O0.3 : machine vers le haut (M3)

O0.4 : machine vers le bas (M3)

- ✓ Les équations des réceptivités :

r1=bp.S5.S3

r2=S1

r3=S2

r4=S4

r5=S5

r6=S3

- ✓ Les équations d'activation et de désactivation :

$$\begin{cases} S1=Init+X6.S3 \\ R1=X2 \end{cases}$$

$$\begin{cases} S2=X1.bp.S5.S3 \\ R2=Init+X3 \end{cases}$$

$$\begin{cases} S3=X2.S1 \\ R3=Init+X4 \end{cases}$$

$$\begin{cases} S4=X3.S2 \\ R4=Init+X5 \end{cases}$$

$$\begin{cases} S5=X4.S4 \\ R5=Init+X6 \end{cases}$$

$$\begin{cases} S6=X5.S5 \\ R6=Init+X1 \end{cases}$$

- ✓ Les équations d'états :

$X1= (S1+X1).\overline{R1}$

$X2= (S2+X2).\overline{R2}$

$X3= (S3+X3).\overline{R3}$

$X4= (S4+X4).\overline{R4}$

X5= (S5+X5).$\overline{R5}$

X6= (S6+X6).$\overline{R6}$

- ✓ Les équations des sorties:

O0.0=X5

O0.1=X2+X3

O0.2=X3

O0.3=X4

O0.4=X6

A partir de ce GRAFCET, transcrit en équations logiques nous avons écrit un programme en langage MATLAB auquel on fait appel du bloc MATLAB Function comme le montre la figure suivante :

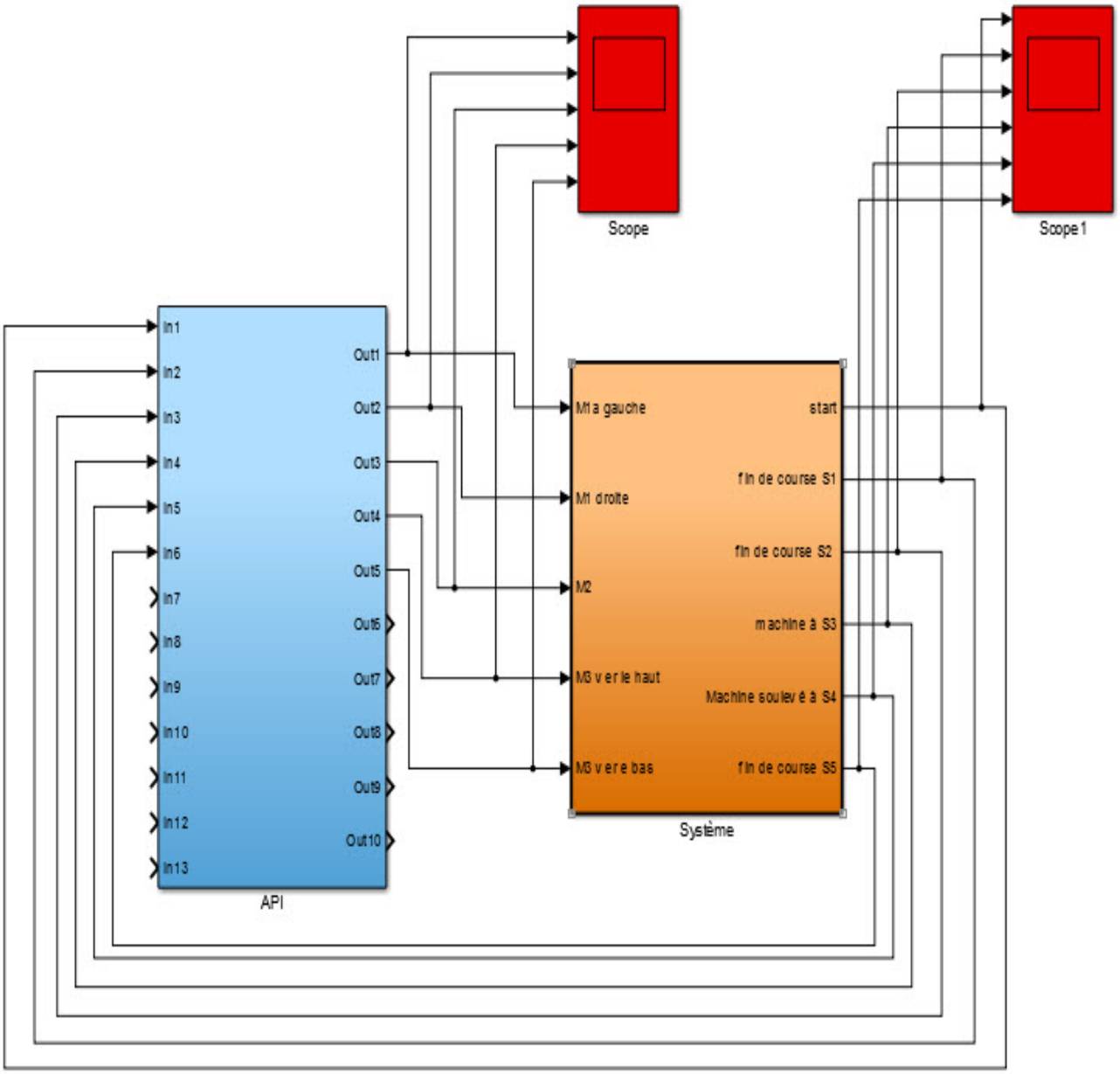


Figure II.12: Modèle de la tronçonneuse sous MATLAB/Simulink

```
initialisation ;
%traitement
m0=(m5 & di4)|(m0&~m1)|(~m0&~m1&~m2&~m3&~m4&~m5);
m1=(m0 & di1 & di4 & di6)|(m1&~m2);
m2=(m1 & di2) | (m2 & ~m3);
m3=(m2 & di3) | (m3 & ~m4);
m4=(m3 & di5) | (m4 & ~m5);
m5=(m4 & di6) | (m5 & ~m0);
affectation ;
```

Après simulation, nous obtenons les résultats suivants :

✓ Les signaux d'entrées du système sont :

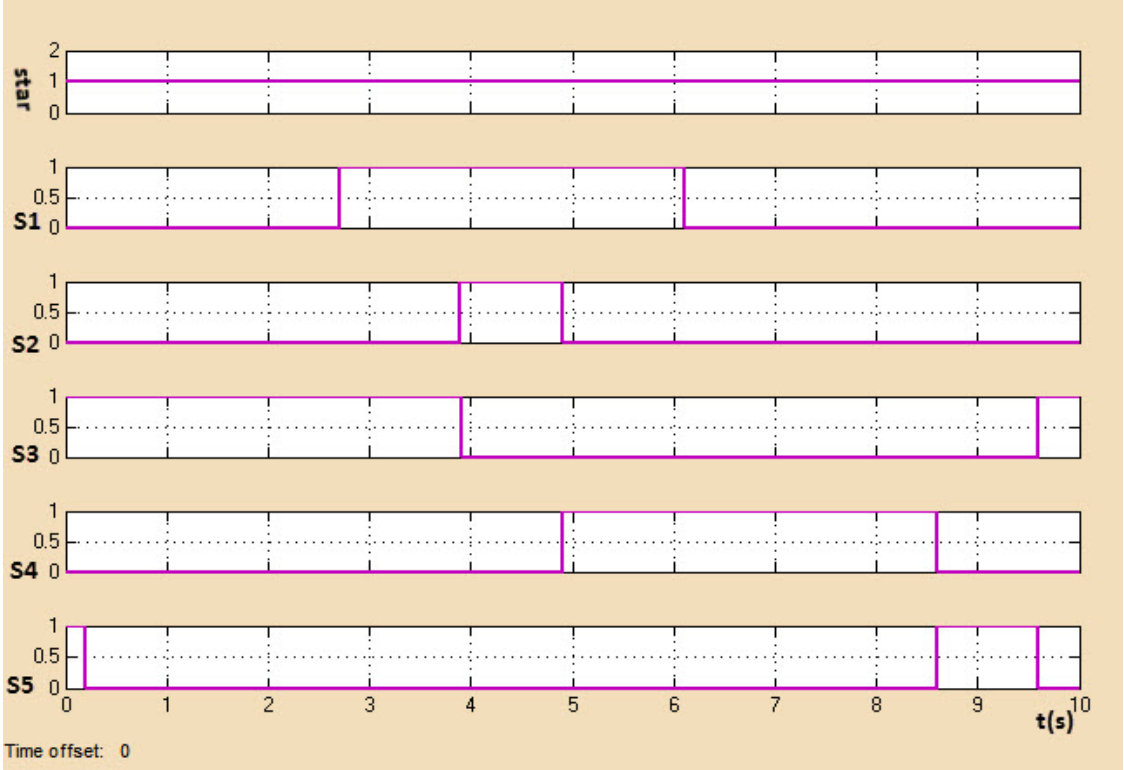


Figure II.13 : Signaux d'entrées de l'API

✓ Les signaux de sortie du système sont :

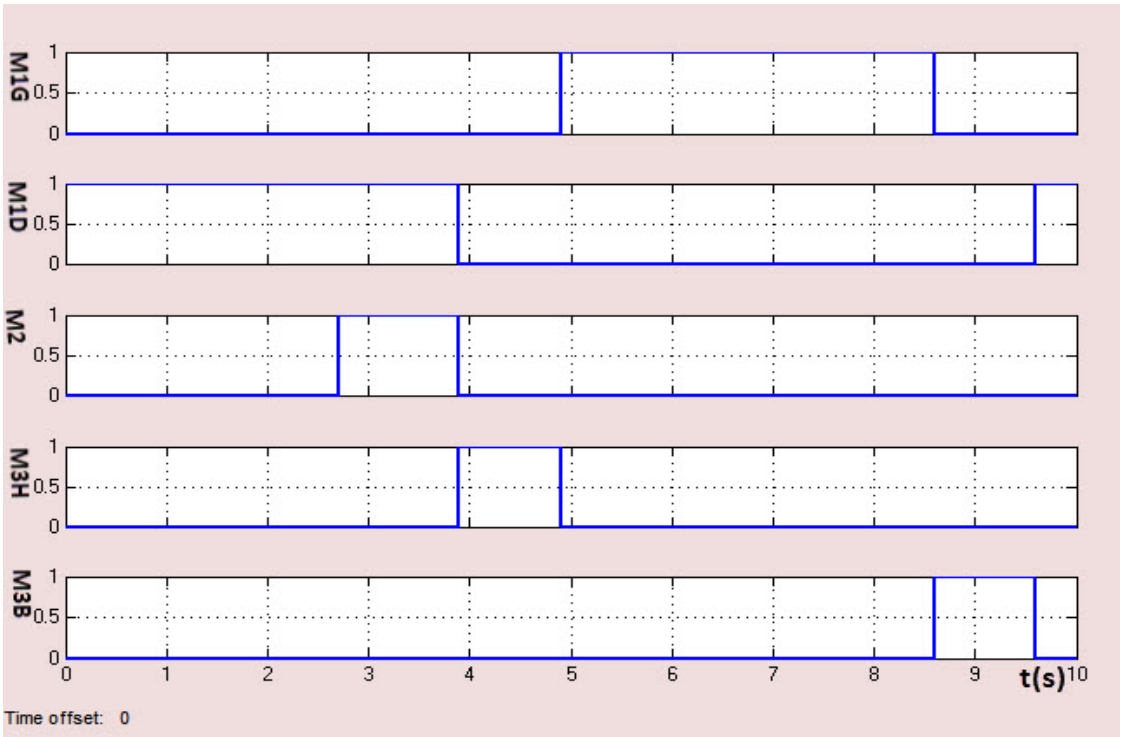


Figure II.14 : Signaux de sorties de l'API

Interprétation des résultats

Au départ la machine est à l'état initial, donc les capteurs de fin de course S5 et S3 sont déclenchés tendis que les autres sont éteints (S1, S2, S4). Quant la machine démarre, S5 s'éteins tendis que S3 reste actif. A son arrivée au capteur S1, ce dernier s'active et provoque le démarrage du moteur 2 et la machine commence à découper et ne s'arrête qu'une fois arrivée au capteur S2.Ce dernier donne l'ordre au moteur 3 de la faire monter, S3 est alors éteint. Puis elle arrive au capteur S4 et donne l'ordre au moteur 1 de faire tourner à gauche et le capteur S5 s'allume donc la machine est au point de départ et elle donne l'ordre au moteur 3 de la faire descendre.

II.4 Méthode de traduction automatique

Pour cette section, nous avons opté pour le langage IL (List Instruction) pour écrire nos programmes,

```
(*m0=(m5 & di4)|(m0&~m1)|(~m0&~m1&~m2&~m3&~m4&~m5)*);
LD      m0.5
AND     I0.3
ST      m6.0
LDN     m0.1
AND     m0.0
ST      m7.0
LDN     m0.0
ANDN    m0.1
ANDN    m0.2
ANDN    m0.3
ANDN    m0.4
ANDN    m0.5
OR      m7.0
OR      m6.0
ST      m0.0
(*m1=(m0 & di1 & di4 & di6) | (m1 & ~m2)*)
LD      m0.0
AND     I0.0
AND     I0.3
AND     I0.5
ST      m1.0
LDN     m0.2
AND     m0.1
OR      m1.0
ST      m0.1
(*m2=(m1 & di2) | (m2 & ~m3)*);
LD      m0.1
AND     I0.1
ST      m2.0
LDN     m0.3
AND     m0.2
OR      m2.0
ST      m0.2
(*m3=(m2 & di3) | (m3 & ~m4)*);
LD      m0.2
AND     I0.2
```

```
ST        m3.0
LDN       m0.4
AND       m0.3
OR        m3.0
ST        m0.3
(*m4=(m3 & di5) | (m4 & ~m5)*);
LD        m0.3
AND       I0.4
ST        m4.0
LDN       m0.5
AND       m0.4
OR        m4.0
ST        m0.4
(*m5=(m4&di6)|(m5&~m0)*)
LD        m0.4
AND       I0.5
ST        m5.0
LDN       m0.0
AND       m0.5
OR        m5.0
ST        m0.5
(*sortie*)
LD        m0.4
ST        O0.0
LD        m0.1
OR        m0.2
ST        O0.1
LD        m0.2
ST        O0.2
LD         m0.3
ST        O0.3
LD        m0.5
ST        O0.4
```

Lors de la simulation de ce programme, MATLAB n'a pas reconnu ce langage.

Pour régler ce problème, nous avons pensé à trouver pour chaque instruction du programme en IL son équivalente en programme MATLAB. Le programme résultant est le suivant :

```
res=m(1,6);
res=res&I(1,4);
m(7,1)=res;
res=~m(1,2);
res=res&m(1,1);
m(8,1)=res;
res=~m(1,1);
res=res&~m(1,2);
res=res&~m(1,3);
res=res&~m(1,4);
res=res&~m(1,5);
res=res&~m(1,6);
res=res|m(8,1);
res=res|m(7,1);
m(1,1)=res;
;
res=m(1,1);
res=res&I(1,1);
```

```
res=res&I(1,4);
res=res&I(1,6);
m(2,1)=res;
res=~m(1,3);
res=res&m(1,2);
res=res|m(2,1);
m(1,2)=res;
;
res=m(1,2);
res=res&I(1,2);
m(3,1)=res;
res=~m(1,4);
res=res&m(1,3);
res=res|m(3,1);
m(1,3)=res;
;
res=m(1,3);
res=res&I(1,3);
m(4,1)=res;
res=~m(1,5);
res=res&m(1,4);
res=res|m(4,1);
m(1,4)=res;
;
res=m(1,4);
res=res&I(1,5);
m(5,1)=res;
res=~m(1,6);
res=res&m(1,5);
res=res|m(5,1);
m(1,5)=res;
;
res=m(1,5);
res=res&I(1,6);
m(6,1)=res;
res=~m(1,1);
res=res&m(1,6);
res=res|m(6,1);
m(1,6)=res;
;
res=m(1,5);
O(1,1)=res;
res=m(1,2);
res=res|m(1,3);
O(1,2)=res;
res=m(1,3);
O(1,3)=res;
res=m(1,4);
O(1,4)=res;
res=m(1,6);
O(1,5)=res;
```

Après avoir obtenu le programme équivalent en MATLAB. Nous avons utilisé le modèle de simulation application3 (tronçonneuse), en changeant son programme par le programme équivalent (langage IL en langage MATLAB). Et nous avons obtenu les mêmes résultats.

D'après ces résultats, on constate que cette méthode d'équivalence est une solution pour que le langage IL soit reconnu dans le langage MATLAB.

Pour cela nous avons élaboré un algorithme qui lit et traduit le programme en langage IL vers un programme en langage MATLAB.

```
f1=fopen(filename,'r');
f2 = fopen('filename2.m','w');
 while ~feof(f1)
 % Lecture d'une ligne du fichier
 s=fgetl(f1);
 [a,b]=strtok(s);
 s1=strcat(b(2),'(',num2str(str2num(b(3))+1),',',num2str(str2num(b(5))+1),')');
 switch a
 case 'LD'
 c=strcat('res=',s1);
 case 'LDN'
 c=strcat('res=~',s1);
 case 'AND'
 c=strcat('res=res&',s1);
 case 'ANDN'
 c=strcat('res=res&~',s1);
 case 'OR'
 c=strcat('res=res|',s1);
 case 'ORN'
 c=strcat('res=res|~',s1);
 case 'ST'
 c=strcat(s1,'=res');
 case 'STN'
 c=strcat(s1,'=~res');
 case 'R'
 c=strcat(s1,'=0');
 case 'S'
 c=strcat(s1,'=1');
 case 'ADD'
 c=strcat('res=res+',s1);
 case 'SUB'
 c=strcat('res=res-',s1);
 case 'DIV'
 c=strcat('res=res/',s1);
 case 'MUL'
 c=strcat('res=res*',s1);
 otherwise
   warning('la commande n''est pas reconnue!');
 c=strcat('');
 end;
 c = fprintf(f2,[c, '; \n']);
 end
 fclose('all')
```

Nous avons défini quelque équivalence entre ces deux langages comme illustré par le tableau suivant :

Programme en IL	Programme en MATLAB
LD m0.5	res=m (1,6)
LDN I0.2	res=res~ I(1,3)
AND I0.4	res =res& I(1,5)
ANDN I0.4	res=res&~ I(1,5)
OR m0.3	res =res|m(1,4)
ORN m0.2	res=res|~ m(1,3)
ST m0.0	m(1,1)=res
STN m1.0	m(2,1)=~res
R O0.0	O(1,1)=0
S I0.1	I(1,2)=1
ADD m0.0	res=res+m(1,1)
SUB M0.4	res=res-M(1,5)
DIV m0.2	res=res/m(0,2)
MUL m0.3	Res=res*m(1,3)

Tableau II.4 : Les équivalences des instructions du langage IL en langage MATLAB

Cet algorithme de traduction lit le fichier en programme (IL) et le traduit ligne par ligne en langage MATLAB. Sachant que MATLAB connait les variables sous forme de vecteurs et matrices et ses derniers commencent par 1 non par 0, donc lorsque le programme lit par exemple m0.0, il le traduit en m(1,1) et lorsque le programme lit une instruction qui n'existe pas dans le langage IL, il affiche un message d'erreur dans le programme traduit. À chaque fin d'instruction l'algorithme met « ; » et il saute la ligne. Le résultat sera enregistré dans un fichier .m.

II.5 Conclusion

Dans la première partie de ce chapitre, nous avons donné un modèle pour la simulation et la programmation des automates programmables industriels qui est une solution réelle pour la simulation des systèmes avec API.

Dans la deuxième partie, nous avons créé un programme qui traduit automatiquement un programme écrit sous forme Liste d'Instruction en langage MATLAB qui sera enregistré dans un fichier M-File.

Dans le chapitre suivant nous utilisons le GUI MATLAB pour la réalisation de l'interface graphique et le V-Realm Builder pour la simulation de nos exemples en 3D.

CHAPITRE III

INTERFACE GRAPHIQUE ET SIMULATION EN 3D

CHAPITRE 3
INTERFACE GRAPHIQUE ET SIMULATION EN 3D

III.1 Introduction

MATLAB est avant tout un logiciel de calcul matriciel. Il possède néanmoins une panoplie complète d'objets graphiques qui permettent d'une part d'afficher les résultats de calculs sous des formes variées (point, courbe, surface, graphique) et d'autre part, de créer des interfaces graphique (GUI) permettant à l'utilisateur d'interagir avec le programme **[10].**

Dans ce présent chapitre, on va présenter d'une manière générale l'outil de programmation GUI MATLAB, de programmer notre interface et nous terminons par une réalisation virtuelle du système sous le logiciel V-Realm Builder et une simulation en 3D.

III.2 Interface graphique

III.2.1 Présentation du GUI Matlab

Les IHM (Interface Homme-Machine), appelées GUI (Graphical User Interfaces) dans MATLAB permettent à l'utilisateur d'interagir avec un programme informatique grâce à des objets graphiques (boutons, menus, cases à cocher,…).

Du fait du nombre important d'objets et surtout du nombre encore plus élevé des paramètres associés, leur programmation "à la main" déroute généralement le débutant. Depuis l'apparition de la version 5.0 (1997), MATLAB possède un outil IDE dédié à la création des interfaces graphiques. Cet outil, appelé GUIDE (Graphical User Interface Development Environment), permet de concevoir intuitivement ces interfaces graphiques, (Voir l'annexe) **[10]**.

III.2.2 Le cahier des charges

Nous voulons réaliser une interface graphique permettant de traduire automatiquement un programme écrit en langage IL vers un programme en langage MATLAB et qui répond aux exigences suivantes :

- L'interface doit contenir au moins trois zones : deux pour la traduction et une pour la simulation ;
- Avoir la possibilité d'ouvrir un fichier en programme IL et le traduire en programme MATLAB puis le simuler ;
- Saisir un programme en langage IL et le traduire en un programme MATLAB, puis le simuler ;
- Affichage des résultats de simulation ;
- Enregistrement de tous programmes (IL ou MATLAB) sous différents formats;
- Avoir un bouton pour quitter;
- L'interface doit être facile à utiliser et moins encombrante.

III.2.3 Présentation de l'interface réalisée

L'interface de notre application est décomposée en trois zones principales qui contiennent des boutons et des zones de textes, ainsi qu'une figure pour la visualisation des signaux (voir la figure suivante).

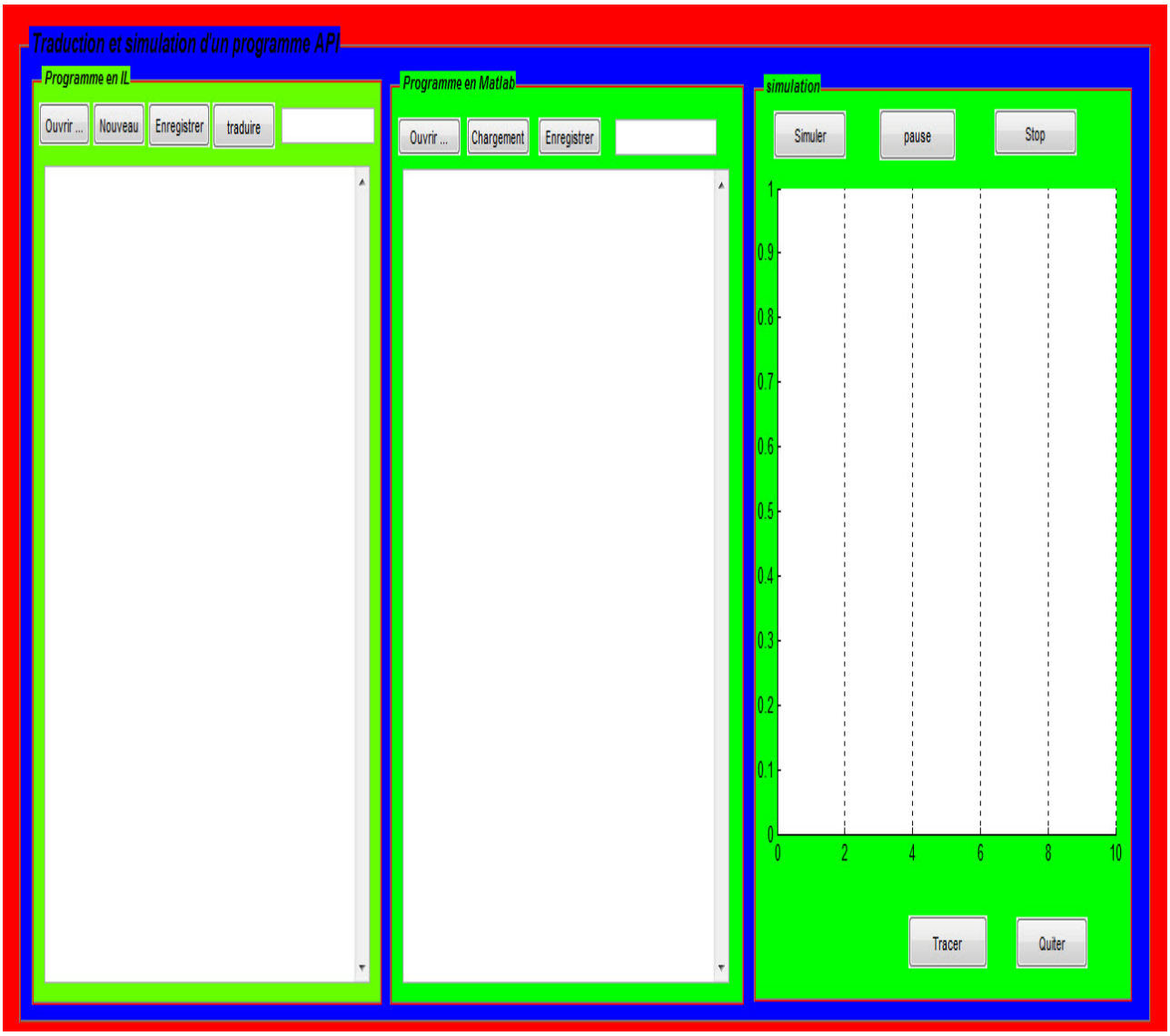


Figure III. 1 : Présentation de l'interface réalisée.

La première zone est réservée au programme IL et contient deux zones de texte, celle de bas pour saisir ou modifier un programme en IL et l'autre pour spécifier le nom du fichier, ainsi que des boutons pour Ouvrir, Nouveau, Enregistrer, Traduire.

La deuxième zone est réservée à un programme en MATLAB. Elle contient aussi deux zone de texte dont première permet de saisir ou modifier un programme en MATLAB la console spécifier le nom du fichier à ouvrir ou à sauvegarder, ainsi que des boutons pour Ouvrir, Chargement, Enregistrer.

La troisième zone, est réservée à la simulation du programme dans l'API. Elle est composée des boutons «Simuler, Pause, Stop », ainsi qu'un bouton pour Tracer les signaux issus de l'API, et un bouton permet de fermer la fenêtre de l'application.

a) Zone programme en IL

La première zone est située à gauche de l'interface elle est réservée au programme en langage (IL) et contient quatre boutons (pushbutton), deux edit text telle que illustrée dans la figure suivante:

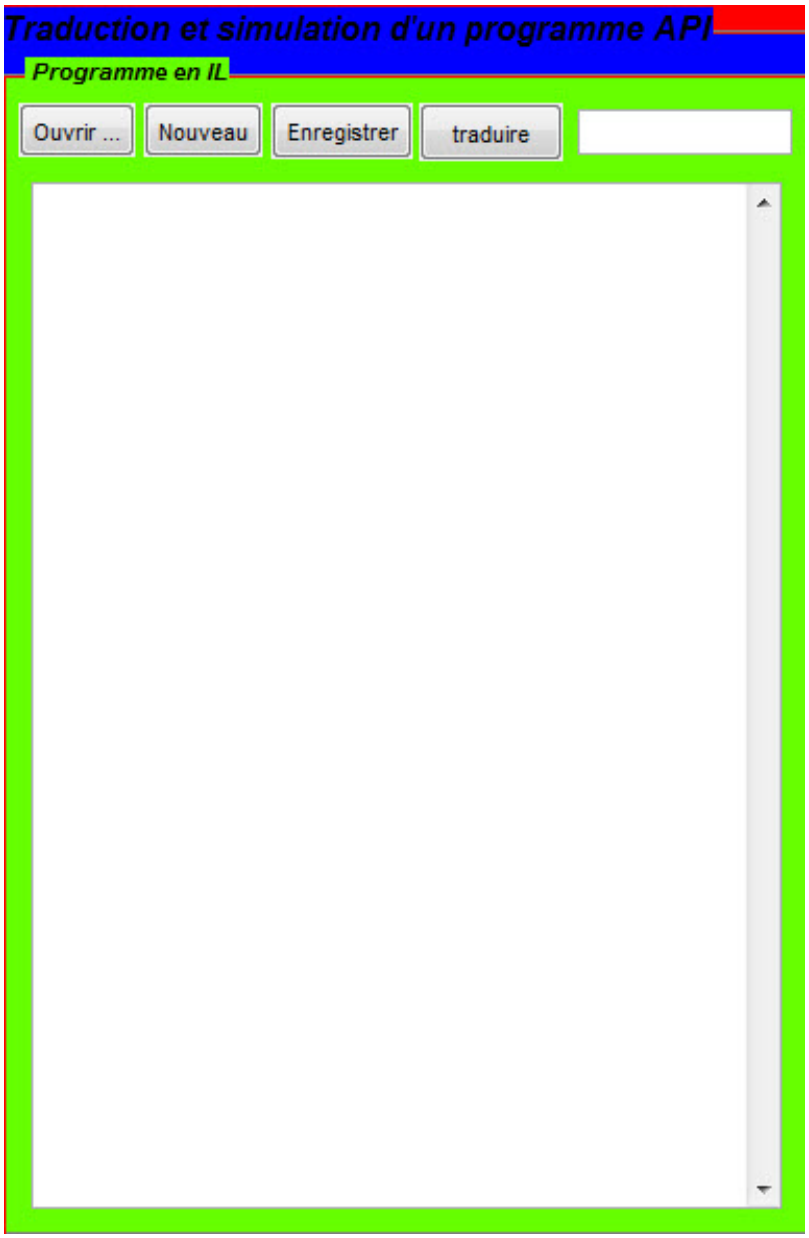


Figure III.2 : Zone programme en IL

1) Bouton « Ouvrir »: permet d'ouvrir le fichier qui contient le programme.

2) Bouton « Nouveau » : sert à la création d'une nouvelle page.

3) Bouton « Enregistrer » : enregistre le programme saisi.

4) Bouton « Traduire » : réalise la traduction d'un programme écrit en langage IL (zone1) en un programme MATLAB (zone2).

5) Zone de texte: c'est l'espace de travail pour saisir ou modifier un programme en IL.

Le travail sur cette zone est très facile. On écrit le nom du fichier en IL à ouvrir puis on appuie sur le bouton ouvrir, le programme en IL s'affiche sur la zone de texte où en peut faire des modifications dans le programme. Puis, on choisit le chemin de sauvegarde et on appuis sur enregistrer. Le bouton traduire réalise la traduction et affiche le résultat de traduction dans la zone2.

b) Zone programme en MATLAB

La deuxième zone est située au milieu de l'interface. Elle contient aussi trois boutons et deux zones de texte, (voir la figure suivante).

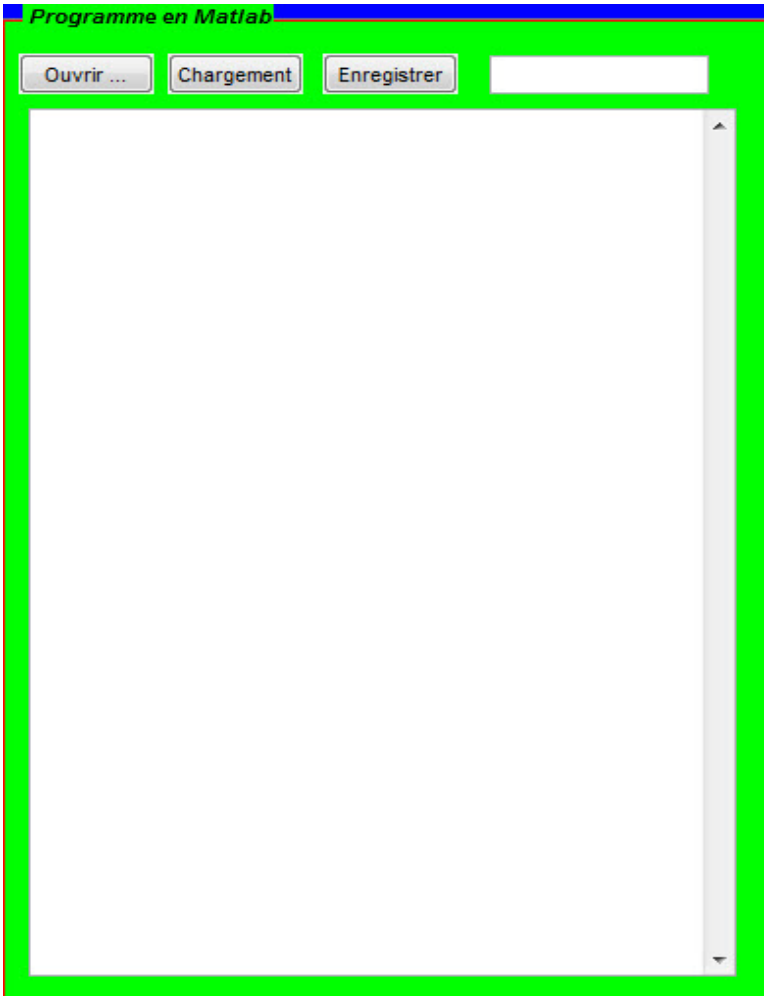


Figure III. 3 : Zone programme en MATLAB

1) Bouton « Ouvrir » : sert à ouvrir le fichier qui contient le programme en MATLAB.

2) Bouton « Chargement » : pour chargé le programme dans l'automate.

3) Bouton « Enregistrer » : sert à enregistrer le programme saisi ou traduit.

4) zone texte: c'est l'espace de travail pour saisir ou modifier un programme en Matlab ou afficher le programme résultant de la traduction.

Cette zone est réservée au programme MATLAB. On choisit le nom du fichier à ouvrir et en appuyant sur le bouton « Ouvrir » le programme s'affiche dans la zone texte où on peut faire des modifications. Puis, on choisit le chemin de sauvegarde et on appuis sur « Enregistrer ». Enfin, en appuient sur le bouton « chargement » le programme sera chargé dans l'automate.

c) Zone simulation

La troisième zone est située à droite de l'interface qui contient cinq boutons (voir la figure suivante).

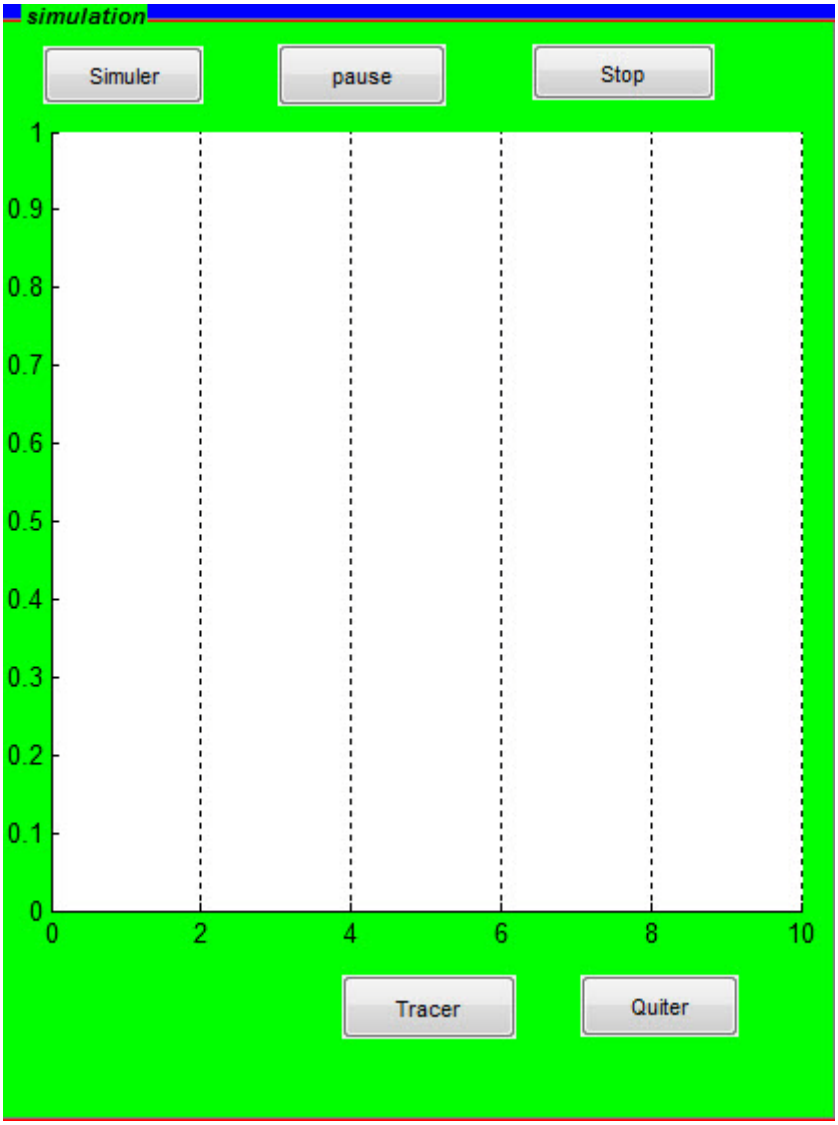


Figure III. 4 : Présentation de la 3éme zone

1) Bouton « Simuler » : permet la simulation du programme.
2) Bouton « Pause » : fait une pause à la simulation.
3) Bouton « Stop » : sert à l'arrêt de la simulation.
4) Bouton « Tracer » : pour afficher les résultats de la simulation.
5) Bouton « Quitter » : nous permet de fermer l'interface.

Le programme chargé précédemment sera simulé dans cette fenêtre. En cliquant sur le bouton « Simuler », le programme commence la simulation, puis sur les boutons « Stop ou Pause » pour arrêter l'exécution du programme. Le bouton « Tracer » sert à visualiser les signaux lors de la simulation en coure de l'exécution.

III.2.4 Exemple d'application

Pour illustrer l'efficacité de cette interface, on reprend l'exemple d'application de la tronçonneuse (Chapitre II).

- On saisi le nom du fichier qui contient le programme en IL, puis on clique sur le bouton « Ouvrir », on aura le programme qui s'affiche dans la zone1.
- On appuie sur le bouton « traduire » le programme traduit s'affiche dans la zone2.
- On choisi le chemin pour enregistrer le fichier, puis on appuie sur le bouton « Enregistrer », le programme sera enregistré avec l'extension .m reconnue par MATLAB.
- On appuie sur le bouton « Chargement » permet de charger le programme dans l'automate.
- On appuie sur le bouton « Simuler », puis sur le bouton « Tracer » et on aura les résultats de la simulation (voir la figure suivante).

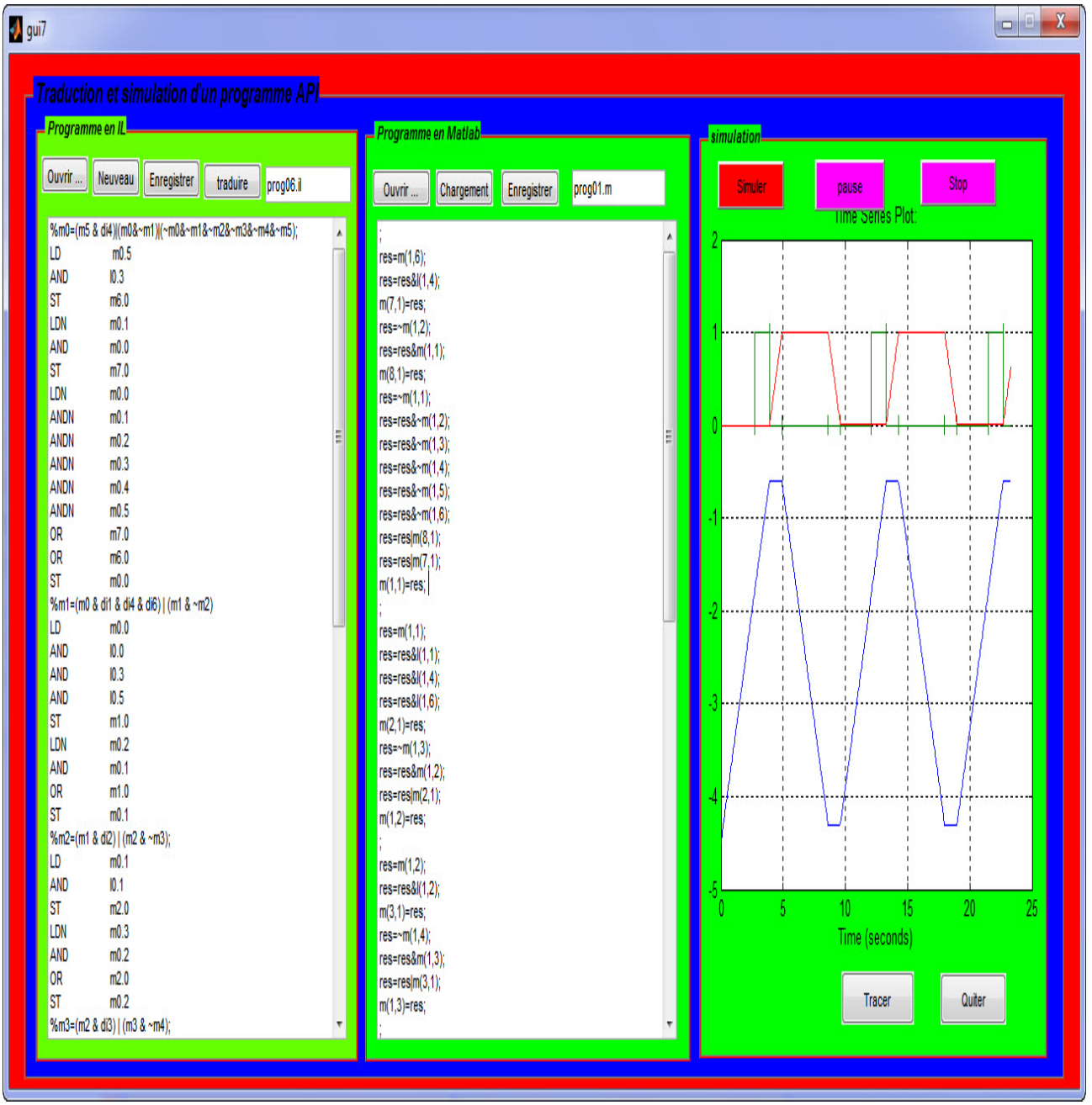


Figure III. 5 : Simulation de la tronçonneuse avec GUIDE MATLAB

III.3 Simulation en 3D avec V-Realm Builder

III.3.1 Présentation du logiciel V-Realm Builder

V-Realm Builder est un logiciel pour la création d'objets 3D et des mondes virtuels, qui peuvent êtres vues avec le navigateur V-Realm ou tout autre navigateur conforme à VRML 2.0. **[12].**

III.3.2 Le langage VRML

Le langage de modélisation de réalité virtuelle [VRML] est une norme ISO qui est ouverte, basé sur du texte, et utilise un format axé sur WWW. On utilise VRML pour définir un monde virtuel qu'on peut afficher avec une visionneuse VRML et de se connecter à un modèle Simulink **[13]**.

Il y a plus d'un moyen de créer un monde virtuel décrit avec le code VRML. Par exemple, on peut utiliser un éditeur de texte pour écrire directement un code VRML, ou bien utiliser un éditeur VRML pour créer un monde virtuel sans avoir à connaître quoi que ce soit sur le langage VRML.

Cependant, on doit comprendre la structure d'un arbre VRML pour connecter un monde virtuel à des blocs Simulink et des signaux.

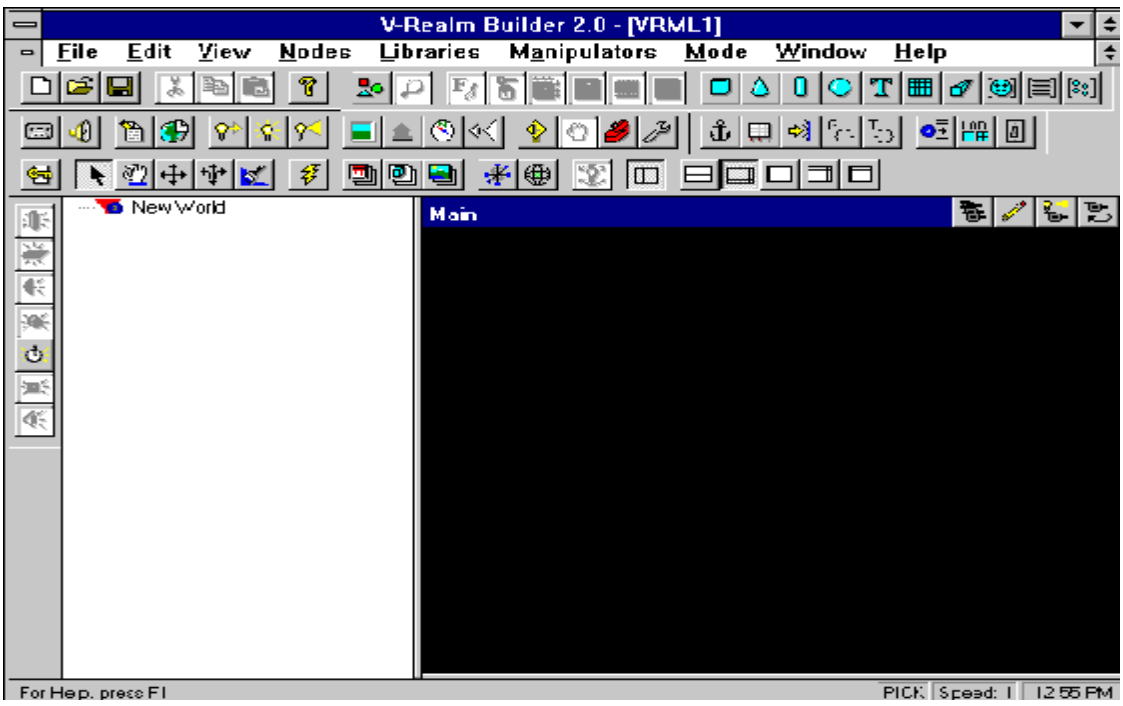


Figure III. 6 : Fenêtre principale du V-Realm Builder

III.3.3 Exemple d'application

Pour illustrer la réalisation en 3D, En reprend l'exemple d'application3 de la tronçonneuse qu'on a étudié précédemment.

Pour réaliser ce système, nous avons adopté les procédures suivantes :

✓ Réalisation plan, capteur, bois (voir la figure suivante)

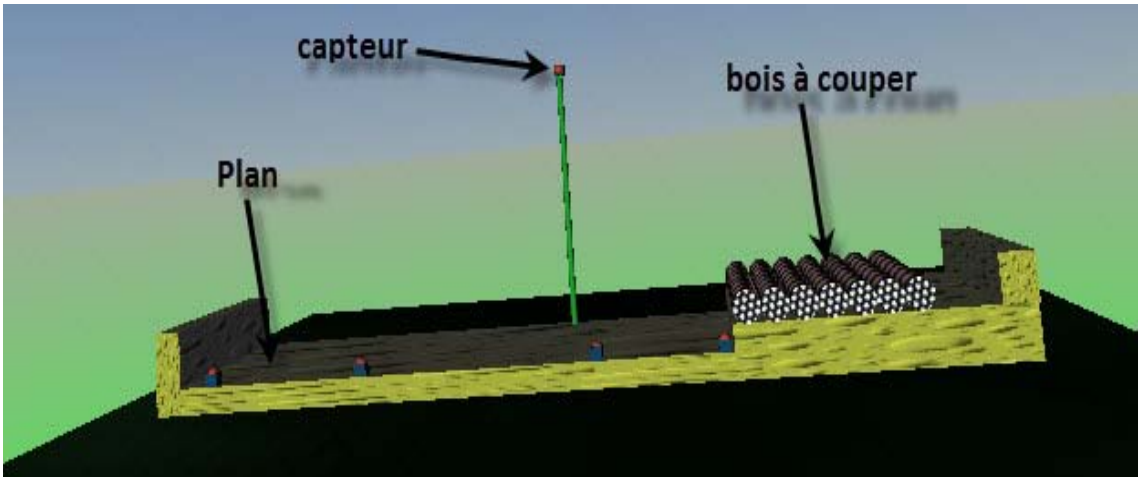


Figure III.7 : Réalisation des éléments (capteur, plan, bois)

✓ Réalisation de la tronçonneuse, équipée d'un disque tournant est illustrée dans la figure suivante :

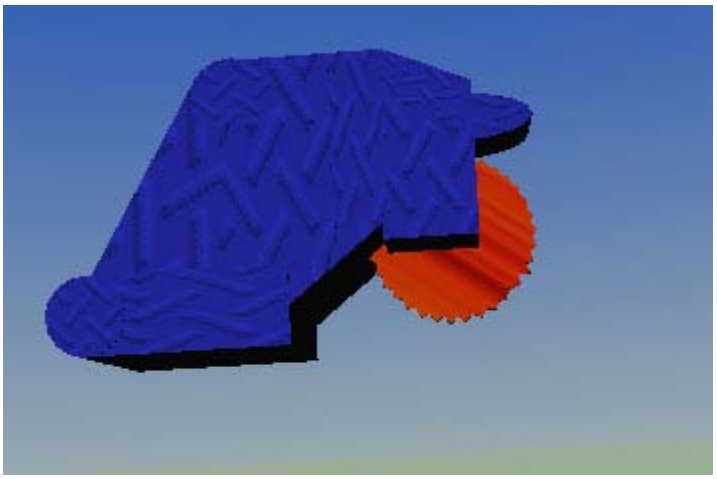


Figure III. 8 : Réalisation de la tronçonneuse

La figure suivante montre l'implantation du système réalisé par V-Rleam Builder pour avoir une simulation en 3D.

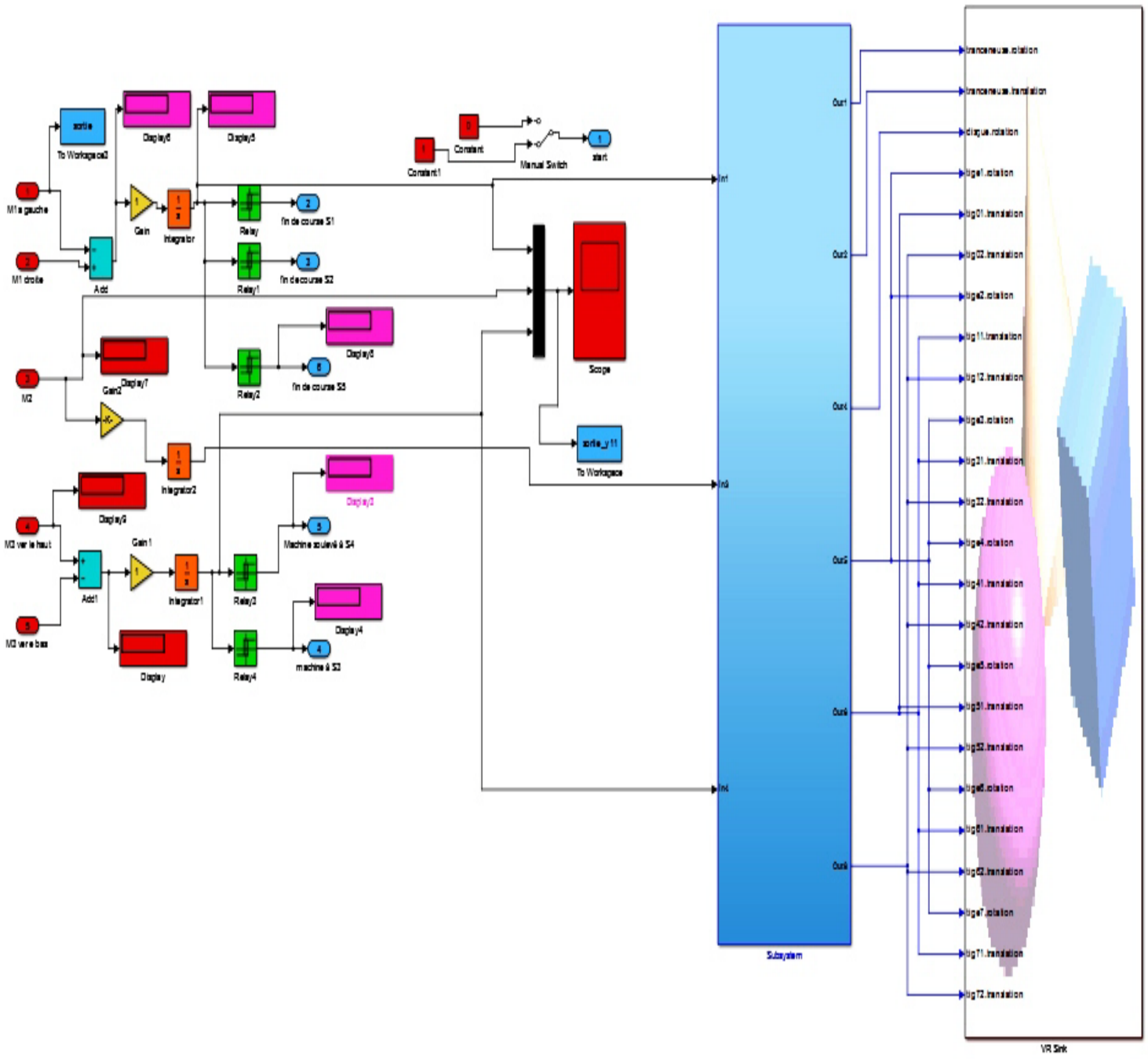


Figure III.9 : Implémentation du système réaliser dans le modèle

Après simulation de ce modèle on aura les résultats suivants, voir la figure suivante :

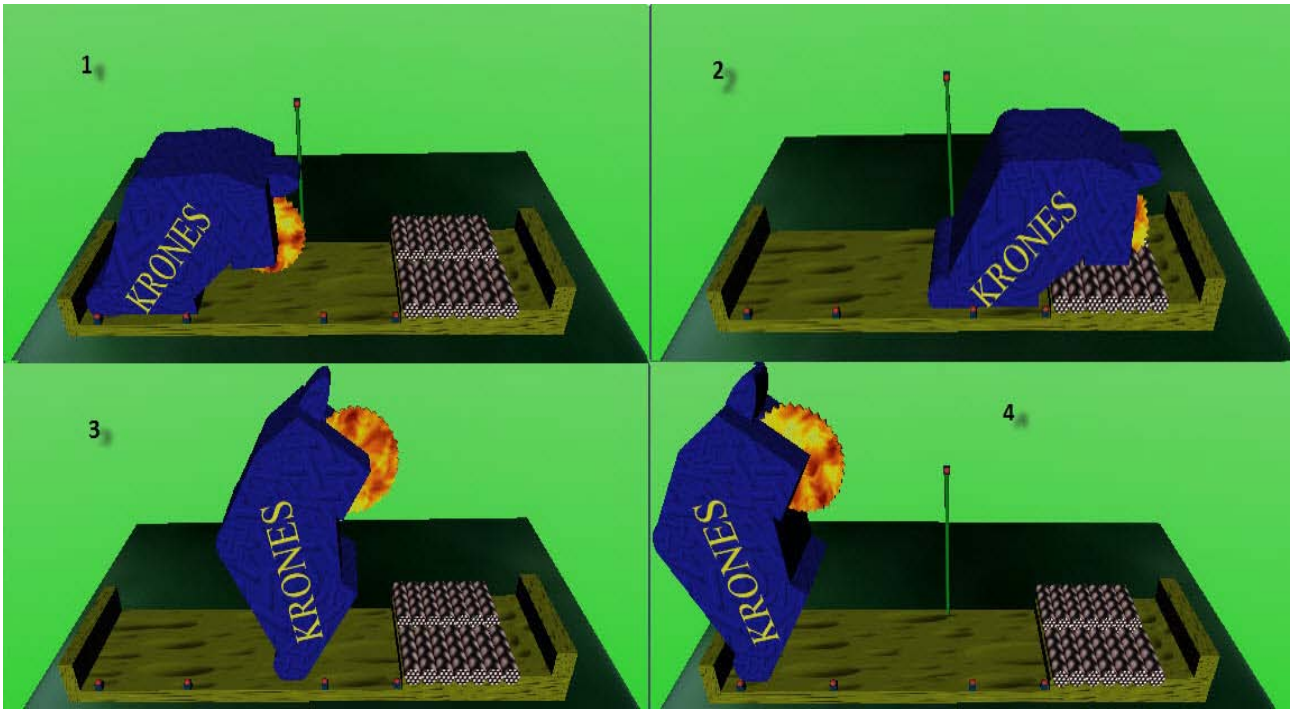


Figure III.10 : Résultats de simulation en 3D (quelques vues)

D'après la figure précédente on remarque que le système réalisé avec V-Rleam Builder suit les signaux de sortie du modèle.

III.4 Conclusion

Dans la première partie de ce chapitre, nous avons présenté GUIDE MATLAB d'une manière générale pour avoir une idée de son fonctionnement. Ensuite et a base de ces notions, on a proposé une solution à notre cahier des charges en réalisant une interface graphique qui répond aux exigences demandées.

Dans la deuxième partie de ce chapitre, nous avons réalisé une tronçonneuse virtuelle à l'aide du logiciel V-Realm Builder, puis nous avons fait la simulation en 3D tout en utilisant l'API sous MATLAB, le GUIDE Matlab et la visualisation en 3D.

Cette démarche est très utile pour concevoir, tester, contrôler des systèmes automatisés avant de les réaliser physiquement.

CONCLUSION GENERALE

CONCLUSION GENERALE

L'automate programmable est un composant électronique, facile à programmer, à connecter, adapté aux conditions industrielles. L'extension considérable de ses possibilités, et celle corrélative de son marché, le prouvent. Toutefois, il faut faire une bonne analyse du problème à résoudre.

Grâce à ce travail, nous avons un aperçu plus large sur les possibilités offertes par la programmation et l'informatique d'une manière générale. Par ailleurs, elle a aussi fait ressortir nos aptitudes autodidactes d'autant plus que nous étions dans l'obligation pour atteindre nos objectifs de faire appel à toutes nos connaissances informatiques. Le but de ce travail a consisté principalement à trouver un environnement pour la programmation et la simulation des API qui sera proche de l'industrie.

Au cours de ce travail, nous avons présenté et rappelé les déférentes notions nécessaires concernant les automates programmables industriels et leurs architectures, puis nous avons défini les cinq langages de programmation des API, ce qui nous a permis de comprendre d'avantage son fonctionnement.

Deuxièmement on a présenté un modèle pour la programmation et la simulation des automates programmables industriels (API) sous MATLAB, puis nous avons adapté un programme en MATLAB pour la traduction automatique du langage (IL) vers le langage MATLAB.

Enfin, nous avons utilisé l'outil de programmation des interfaces homme machine « GUIDE MATLAB », et on a conçu une interface pour la saisie, la traduction et la simulation de notre système. Puis à l'aide du logiciel V-Rleam Builder, nous avons réalisé des tests de simulation en 3D. Les résultats de simulations valident le travail effectué.

Nous avons acquis une bonne expérience sur le plan théorique et approfondi nos connaissances sur les automates programmables industriels. Nous avons atteint le but de notre projet qui été concentré sur l'utilisation du logiciel MATLAB, et nous espérons l'amélioration du modèle en incluant d'autres fonctionnalités: actionneurs, capteurs,… Et qu'il deviendra un aide pour les prochains projets d'implantation dans l'industrie et les services.

BIBLIOGRAPHIE

REFERENCES BIBLIOGRAPHIQUES

[1] GILLES Michel, « Architecture et application des automates programmables », DUNOD, Paris, 1988.

[2] Morriss S. B., « Automated Manufacturing Systems », McGraw Hill, 1994.

[3] ANDRE Simon, « Automates programmables industriels », EYROLLES, Paris, 1991.

[4] KIM HyungSeok, « A translation method for ladder diagram with application to a manufacturing process», Proceedings of the IEEE International Conference on Robotics and Automation, USA, 1999.

[5] FANUC G, **«** Automation, Automates programmable industriels Logiciel de Programmation pour API », GFK-0467j-FR , France,1997.

[6] M. Chmiel, E. Hrynkiewicz, M. Muszynski**,** « The way of ladder diagram analysis for small compact programmable controller», KORUS, Russie, 2002.

[7] JARGOT Patricia**, «** Langages de programmation pour API. Norme IEC 1131-3 », Technique d'Ingénieur, traité Informatique industrielle,France,1999.

[8] HUGH Jack, **«** Automating Manufacturing Systems with PLC», ebooks, New York,2005.

[9]HUG Hoang, **«**Introduction à Simulink »,www.math works.com/product/Simulink/, le: 03-04-2013.

[10] JEROME Briot, **«**A guide to MATLAB for Beginners Epecienced Users »,Handback, Cambridge,.2000

[11] BOUZEMBOUA & BOUKHENAK ; « Etude et réalisation d'un système de positionnement à 2D/3D à l'aide d'un logiciel de réalité virtuelle », Mémoire de fin d'étude master en automatisme industriel, Bejaia, 2011-2012

[12] Realm^{TM} Builder, **«**Guide d'utilisation et de référence », Ligos Corporation, France,1997.

[13] Help Matlab, **«**VRLM Eiting tools Virtual words virtual reality toolbox».

[14]MIKELL.K, **«**Automation, Production Systems and Computer Integrated Manufacturing», Prentice Hall, 1987.

[15] LEITE Emilson Pereira, **«**Matlab-Modelling Programming and Simulations», SCIYO, India, 2010.

ANNEXES

ANNEXE

1) Annexe A

a) Les contacts

Nom	Symbole	Fonctions
Contact Normalement Ouvert (contact direct)		Passer le contact quand le bit de contrôle est a l'état 1, l'état de la liaison à droite est le résultat du ET logique entre l'état de la liaison à gauche et la valeur de la variable associée au contact
Contact Normalement Fermer (contact inversé)		Passer au contact quand le bit de contrôle est à l'état 0, l'état de la liaison à droite est le résultat du ET logique entre l'état de la liaison à gauche et l'Inverse logique de la variable associée au contact
Contact à détection de front positif (front montant)		Détection de changement du bit de contrôle de 0 à 1, l'état de la liaison à droite prend 1 quand la liaison à gauche est à 1 et que de la variable associée passe de 0 à 1. Il prend 0 dans tous les autres cas.
Contact à détection de front négatif (front descendant)		Détection de changement du bit de contrôle de 1 à 0, l'état de la liaison à droite prend 1 quand la liaison à gauche est à 1 et que de la variable associée passe de 1 à 0. Il prend 0 dans tous les autres cas.

Tableau A.1 : Les contacts en langage LD

b) Les relais

Noms	Symbole	Fonctions
Relais direct	—( )—	Il permet le forçage d'une sortie booléenne avec l'état de la liaison à gauche (zone de test), donc le relais représente l'affectation de la valeur renvoyée par la fonction.
Relais inversé	—(/)—	Il permet le forçage d'une sortie booléenne avec l'inverse logique de l'état de la liaison à gauche
Relais à action SET	—(S)—	Le relais SET permet le forçage d'une sortie booléenne 1 quand la liaison à gauche prend l'état 1.
Relais à action RESET	—(R)—	Le relais RESET permet le forçage d'une sortie booléenne à 0 quand la zone test prend l'état 1.
Relais de transition conditionnelle	—(#)—	C'est un relais associé au Grafcet, il est utilisé Quand la programmation des transitions conditionnelles associe avec des transitions cause des changements à des étapes ultérieures.
Jump (appelle de sous-programmes)	-»%Li -»%Sri	C'est des relais qui sont utilisés pour faire des connexions à des étiquettes, soit en amont ou en aval.
Retour à partir D'un sous-programme	< RET >	Il place à la fin de sous programme pour retourner au programme principal.
Arrêt du Programme	< END >	Définie la fin du programme

Tableau A. 2 : Les bobines en langage LD

b) Opérateur du langage IL

Opérateur	Modificateurs	Opérandes	Descriptions
LD	N	variable, Constante	charge l'opérande
ST	N	variable BOOL	stocke le résultat courant
S , R	/	variable BOOL	forçage à TRUE, forçage à FALSE
AND, &	N , (	variable BOOL	ET logique
OR	N , (	Variable BOOL	OU logique
XOR	N , (	variable BOOL	OU exclusif
ADD	(	variable, constante	Addition
SUB	(	variable, constante	Soustraction
MUL	(	variable, constante	Multiplication
DIV	(	variable, constante	Division
GT	(	variable, constante	test >
GE	(	variable, constante	test >=
EQ	(	variable, constante	test =
NE	(	variable, constante	test < >
LE	(	variable, constante	test <=
LT	(	variable, constante	test <
JMP	C, N	Etiquette	saut à une étiquette
CAL	C, N	instance de bloc fonctionnel	appel d'un bloc fonctionnel
RET	C, N	/	retour de fonction
)	/	/	exécute l'instruction différée

Tableau A. 3 : Opérateurs du langage IL

2) Annexe B

L'interface du Guide MATLAB

Guide MATLAB est doté d'une interface facile à utiliser composé d'un menu, d'une barre à outils, d'une palette d'objets et d'un espace dédié au placement des objets dont on est besoin **[11]**.

La figure suivante présente l'interface du GUIDE Matlab et ses différents éléments de base.

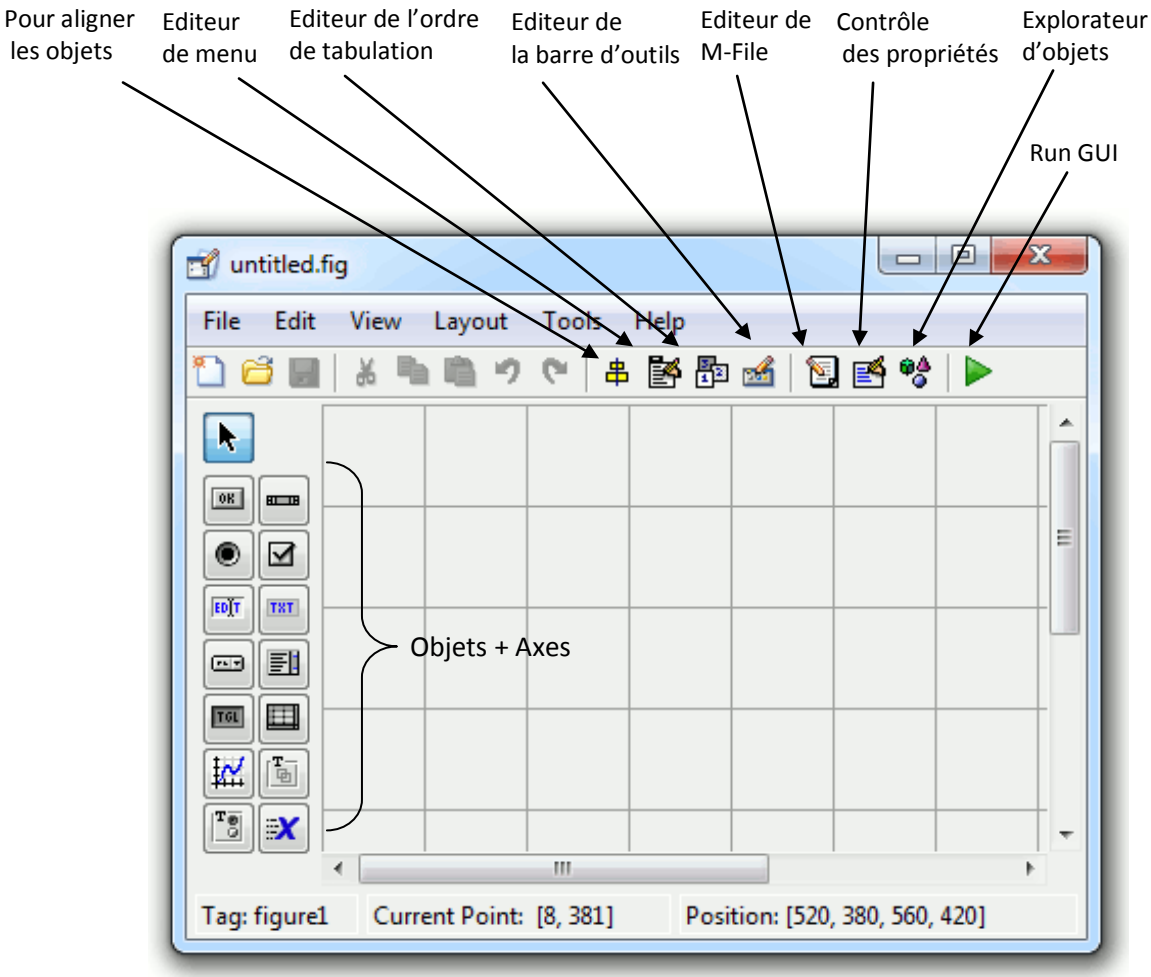


Figure B.1: Fenêtre principale du Guide

Les objets graphiques

Sous MATLAB, les objets graphiques sont disposés selon une hiérarchie pyramidale parent-enfant, voir figure suivante :

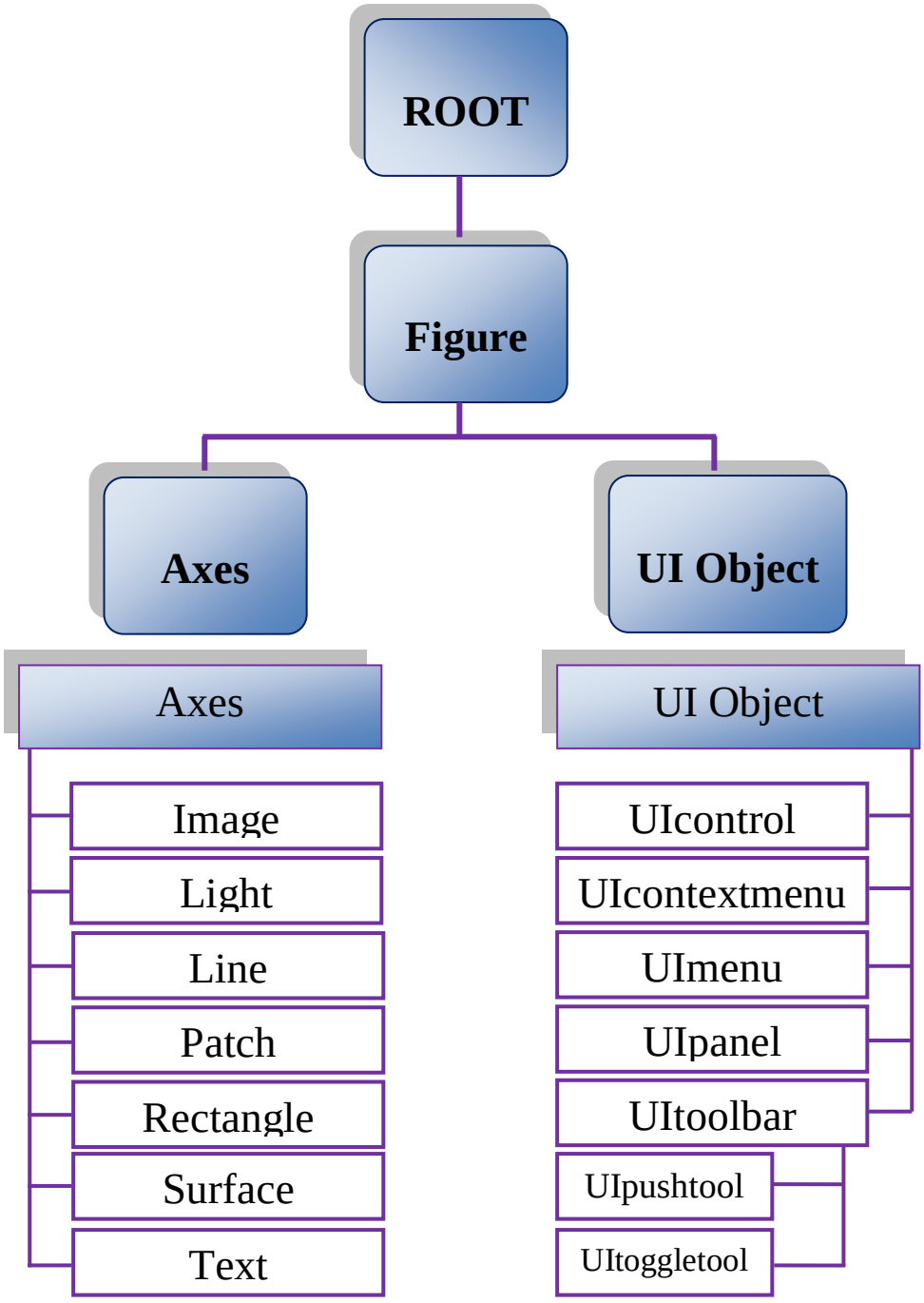


Figure B.2 : La hiérarchie parent-enfants des objets

Au sommet de la hiérarchie se trouve l'objet Root. Cet objet est invisible (on peut le représenter comme étant l'écran de l'ordinateur). Ensuite, on trouve les objets de type Figure. Ce sont les conteneurs visibles où sont disposés tous les autres objets enfants. Plusieurs objets Figure peuvent être ouverts simultanément et peuvent éventuellement communiquer entre eux **[10].**

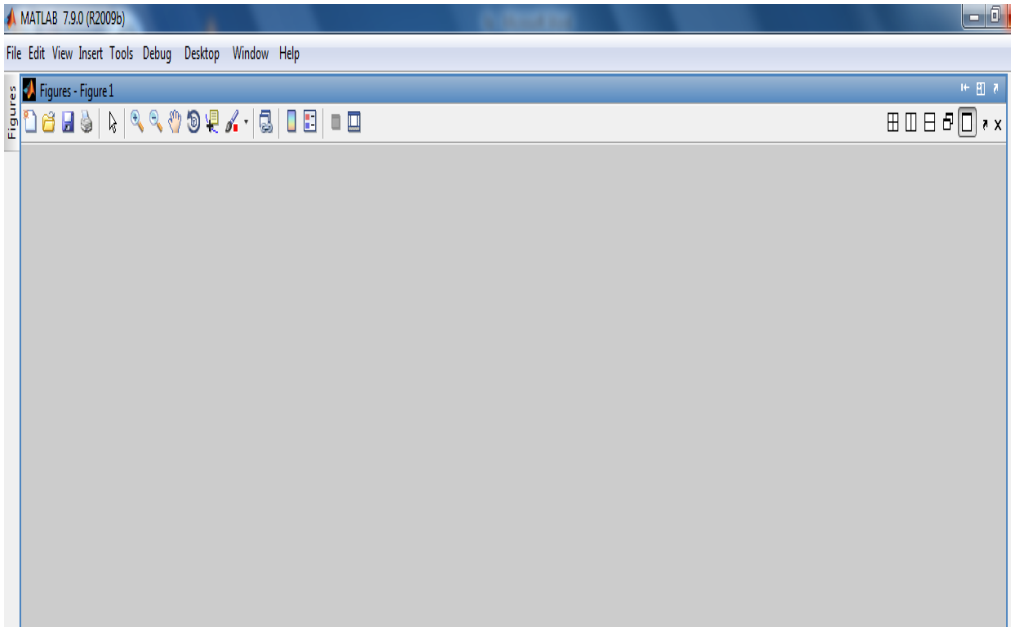


Figure B.3 : Présentation d'une figure

Viennent ensuite les objets de type Axes qui sont les zones de traçage des graphiques (2D ou 3D). Ces objets ont pour enfants, tous les objets représentants des résultats mathématiques (courbes, surfaces, images, .. etc.). Un objet Figure peut contenir plusieurs objets Axes simultanément.

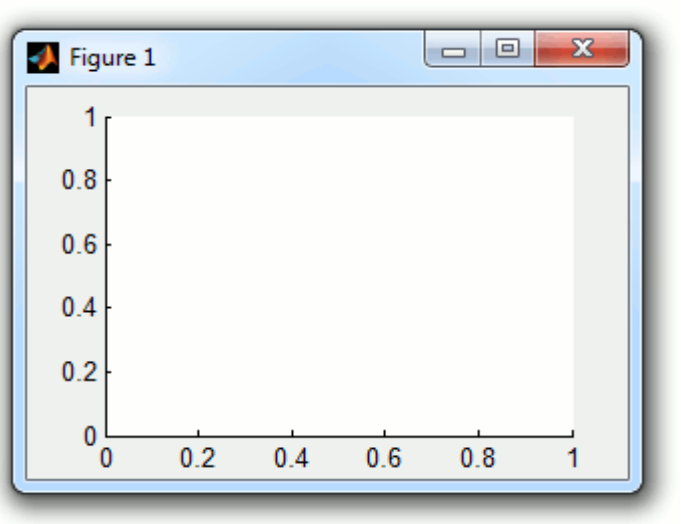


Figure B.4 : Présentation d'un Axe

On trouve également au même niveau les objets UI (User Interface), tels que des boutons, des menus, des cases à cocher, ..etc. Ces objets permettent à l'utilisateur d'interagir dynamiquement à la souris avec le GUI.

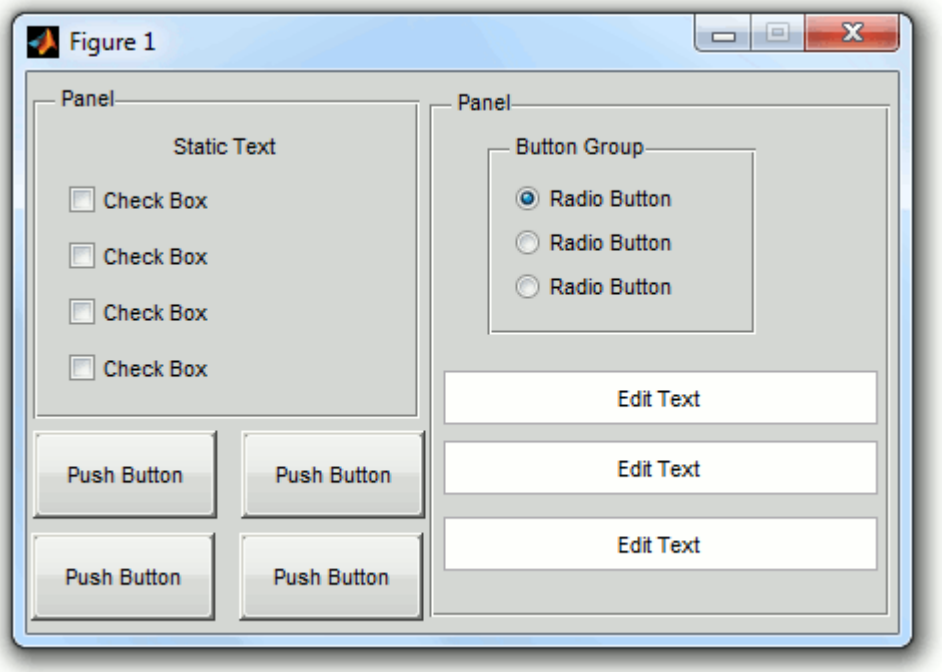


Figure B. 5 : Présentation des boutons

Les identifiants des objets graphiques

A la création d'un objet, MATLAB lui attribue automatiquement un identifiant (handle), sous la forme d'un nombre réel unique qui peut être stocké dans une variable.

Ceci permet de retrouver à tout moment un objet graphique au cours du fonctionnement d'une interface. Cet identifiant existe tant que l'objet existe. Dès que l'objet est détruit, cet identifiant disparait **[11]**.

On gère les identifiants, soit avec la fonction GUIHANDLES, soit avec les fonctions FINDOBJ/FINDALL.

Quelques identifiants particuliers peuvent être gérés avec les fonctions suivantes :

- GCA qui récupère l'identifiant de l'objet Axes courant
- GCBF qui récupère l'identifiant de l'objet Figure où se trouve l'objet graphique dont l'action est en cours
- GCBO qui récupère l'identifiant de l'objet graphique dont l'action est en cours
- GCF qui récupère l'identifiant de l'objet Figure courant
- GCO qui récupère l'identifiant de l'objet graphique courant

Les propriétés des objets graphiques

Chaque objet graphique possède des propriétés (position, couleur, action, etc.) qui sont définies à sa création et qui peuvent être modifiées dynamiquement au cours du fonctionnement du GUI. Ces propriétés peuvent être récupérées et modifiées en utilisant l'identifiant de l'objet et les fonctions GET et SET.

- SET : attribuer des valeurs à un paramètre spécifique d'un objet
- GET : obtenir des valeurs d'un objet

On utilise les syntaxes suivantes

- GET (handles. Nom de l'objet.'le paramètre')
- SET (handles. Nomde l'objet.'le paramètre', 'valeur')

Résumé

"Simulation d'un Automate Programmable Industriel Sous Matlab"

Au cours de ce travail, nous avons présenté et rappelé les déférentes notions nécessaires concernant les automates programmables industriels et leurs architectures, puis nous avons définit les cinq langages de programmation et les différents entrées/sorties, ce qui nous a permis de comprendre d'avantage son fonctionnement. En suite on a choisis un modèle pour la programmation et la simulation des automates programmables industriels (API), puis nous avons créé un algorithme qui traduit automatiquement le programme en langage (IL) vers un programme en langage Matlab. Et enfin on a conçu une interface graphique à l'aide du GUIDE Matlab pour la traduction et la simulation de notre système, puis à l'aide du logiciel de conception V-Rleam Builder on a réalisé des tests de simulation en 3D.

Mots-clés :

Automate programmable industriel(API), capteurs, Matlab, GUIDE MATLAB, V-Realm Builder, langages graphiques, langages textuels.