

Département Electronique, Energie Electrique,
Automatique

1ère année de Master ISTR

Ingénierie des Systèmes Temps Réel

COMMANDE DISCRÈTE DE BRAS ROBOT PAR MICROCONTRÔLEUR

Etudiants :

Ameur BOUCHNAK
Anass AMMAR
Farid KADI
Larbi OUCHALAL

Encadrant(s) :

Nicolas RIVIERE

Module Initiation à la Recherche et Projet
Année 2022/2023

Contacts : ameur.bouchnak@univ-tlse3.fr, anass.ammar@univ-tlse3.fr,
larbi.ouchalal@univ-tlse3.fr, farid.kadi@univ-tlse3.fr

Table des matières

1	Introduction	6
2	Bras robot et son environnement	7
2.1	Bras Dobot	7
2.1.1	Système de coordonnées du bras Dobot	8
2.1.2	Modes de mouvement	9
2.1.3	Caractéristiques techniques du bras Dobot	10
2.1.4	Logiciel DobotStudio	10
2.2	Description du convoyeur	11
2.3	Capteurs	11
2.3.1	Capteur de Proximité Photoélectrique Infrarouge	11
2.3.2	Capteur de couleur	12
3	Arduino et son environnement de développement	13
3.1	Présentation de la carte Arduino Mega 2560	13
3.1.1	Micro-contrôleur ATmega2560	14
3.1.2	Alimentation de la carte	14
3.1.3	Entrées et sorties numériques	14
3.1.4	Broches analogiques	14
3.2	Langage de programmation	15
3.3	Environnement de développement (IDE)	15
4	Bibliothèque de commande	16
4.1	Analyse de la problématique	16
4.2	Étude des bibliothèques fournies par le constructeur	16
4.3	Différentes positions de notre robot	21
4.4	Solution proposée et sa mise en œuvre :	21
4.4.1	Dobot_SetHOMECmd()	21
4.4.2	Dobot_Init()	22
4.4.3	Dobot_SetPTPCmd()	22
4.4.4	Dobot_GetIODI()	23
4.4.5	Dobot_SetIOMultiplexing()	23
4.4.6	Dobot_Pince()	24
4.4.7	Dobot_Tapis1()	24
4.4.8	Dobot_Tapis2()	25
4.4.9	Dobot_Piece()	25
4.4.10	Dobot_Getcolor()	26
4.4.11	Dobot_AllerTapis1()	27
4.4.12	Dobot_AllerTapis2()	27
4.4.13	Dobot_AllerPosHOME()	27
4.4.14	Dobot_AllerPos1()	28

4.4.15	Dobot_AllerPos2()	28
4.4.16	Dobot_AllerPos3()	29
4.5	Implémentation de commandes à partir de modèles (RdP, Grafcet)	29
5	Réalisation pratique	31
5.1	Protocole de communication série	31
5.2	Branchement entre l'arduino et Dobot	31
5.3	Scénario élaboré pour le test des bibliothèques	32
5.4	Programmation de l'Arduino MEGA	33
5.5	Ajustements proposés	34
6	Conclusion et perspectives	36

Table des figures

2.1	Description du bras Dobot	7
2.2	Branchement de la Pince (Gripper) et la pompe.	8
2.3	systèmes de coordonnées cartésiens	8
2.4	Système de coordonnées des articulations	8
2.5	Modes MOVJ et MOVL	9
2.6	Mode JUMP	9
2.7	Mode ARC	9
2.8	Caractéristique technique.	10
2.9	Moniteur série de l'Arduino.	10
2.10	Convoyeur	11
2.11	Branchement du Convoyeur 1 et 2	11
2.12	Capteur de Proximité Photoélectrique Infrarouge SENS-97 E18-D80NK. . .	11
2.13	Branchement du Capteur Infrarouge	12
2.14	Couleur détecter par le capteur	12
2.15	Branchement du capteur de couleur.	12
3.1	Carte Arduino Mega 2560.	13
3.2	Caractéristique de la carte Arduino Mega 2560.	14
3.3	Micro-contrôleur ATmega2560.	14
3.4	L'interface du logiciel Arduino.	15
3.5	Moniteur série de l'Arduino.	15
4.1	Les positions du bras.	21
4.2	Fonction Dobot_SetHOMECmd()	21
4.3	Fonction Dobot_Init()	22
4.4	Fonction Dobot_SetPTPCmd()	22
4.5	Fonction Dobot_GetIODI()	23
4.6	Fonction Dobot_SetIOMultiplexing()	23
4.7	Fonction Dobot_Pince()	24
4.8	Fonction Dobot_Tapis1()	24
4.9	Fonction Dobot_Tapis2()	25
4.10	Fonction Dobot_Piece()	25
4.11	Fonction Dobot_Getcolor()	26
4.12	Fonction Dobot_AllerTapis1()	27
4.13	Fonction Dobot_AllerTapis2()	27
4.14	Fonction Dobot_AllerPosHOME()	28
4.15	Fonction Dobot_AllerPos1()	28
4.16	Fonction Dobot_AllerPos2()	28
4.17	Fonction Dobot_AllerPos3()	29
5.1	Serial 2 [2]	31
5.2	Branchement entre l'arduino et Dobot	32

5.3	Le réseau de Pétri modélisant le scénario.	33
5.4	Image de la pièce conçu pour centraliser les pièces en 3D.	35

Remerciements

C'est avec joie que nous formulons notre remerciements qui témoignent par écrit, notre reconnaissance à tous ceux qui nous ont manifesté leur soutien et leur confiance tout au long de la période du projet.

Nous remercions chaleureusement, notre encadrant : Monsieur NICOLAS RIVIERE enseignant à la faculté de science et d'ingénierie de l'université Toulouse 3 Paul Sabatier pour sa disponibilité continue, pour ses conseils précieux et recommandations qui ont été très utiles durant le projet.

Nous exprimons nos profonds sentiments de respect et de gratitude à tout le cadre professorale du Master EEA parcours ingénierie des systèmes temps réel et à tous ceux qui nous ont apporté un soutien scientifique, technique et moral.

Finalement, nous terminons ces remerciements en saluant vivement les membres du jury pour l'honneur qu'ils nous ont fait en acceptant de juger ce travail, d'avoir accepté de lire ce manuscrit, le corriger et d'apporter les critiques nécessaires à l'amélioration de ce rapport.

Chapitre 1

Introduction

Ce projet est mené dans le cadre du module Initiation à la recherche et projet portant sur la commande discrète de bras robot par microcontrôleur. L'objectif principal derrière ce travail est de développer les atouts acquis lors de la formation suivie et notre esprit de synthèse. On souhaite par ce projet rendre la maquette du bras robotisé utilisable dans des travaux pratiques de systèmes à événements discrets.

Ce projet a pour objectif principal de créer une bibliothèque qui permettra de contrôler un bras robotique et son environnement en utilisant une carte Arduino Mega pour contrôler la maquette du bras robotisé Dobot-Magician qui était seulement contrôlable par son logiciel associé. Cette bibliothèque aura pour but de simplifier les commandes grâce à l'utilisation de modèles standardisés de programmation (RdP et Grafsets) en développant des fonctions booléennes permettant de commander de façon discrète notre maquette.

La problématique consiste donc à trouver une solution efficace pour la commande du bras Dobot et de son environnement en utilisant des modèles standardisés tout en étant capable de l'intégrer avec un microcontrôleur et de la tester dans des conditions réelles.

Dans le même ordre d'idées, ce travail est réparti en quatre chapitres principaux comme suit :

Le bras robot et son environnement : Il s'agit de présenter dans un premier temps le bras Dobot, son fonctionnement et système de coordonnées. Aussi, on a présenté une description du convoyeur et des capteurs lié à la maquette qui sont le capteur infrarouge et de couleur.

Arduino et son environnement de développement : Nous avons subdivisé ce chapitre en trois parties, la première partie est consacrée à la présentation de la carte Arduino Mega 2560. La deuxième partie montre le langage de programmation utilisé pour élaborer notre projet. Tandis que la troisième partie présente l'environnement de développement IDE comme l'interface du logiciel et le moniteur série utilisé dans l'Arduino.

Bibliothèque de commande : Dans ce chapitre nous détaillons l'étude que nous avons réalisée en commençant par une analyse de la problématique, après une étude approfondie des bibliothèques fournies par le constructeur. Ensuite, nous avons déterminé les positions intéressantes du robot. Après nous avons décrit les fonctions créées et faisant part de nos propres bibliothèques et finalement nous avons expliqué l'implémentation de la commande à partir des modèles standardisés comme le réseau de pétri et les Grafsets.

Réalisation pratique : Ce chapitre est dédié entièrement à la réalisation en pratique du projet en interagissant avec la maquette. Une description du protocole de la communication série et le branchement permettant d'assurer cette liaison entre le robot et l'Arduino fait partie de ce chapitre. Après, on a élaboré un scénario basé sur un modèle RdP pour la phase de test des bibliothèques. Ensuite, un code est écrit pour la mise en oeuvre de ce scénario en réalité. Finalement des suggestions d'ajustements sont proposées pour réduire ou éliminer les dysfonctionnements remarqués lors de la phase de test.

Chapitre 2

Bras robot et son environnement

Dobot Magician est un bras robotisé, à visée pédagogique, muni de différents outils et accompagné d'un environnement de programmation à l'image des robots industriels. Le bras du robot peut se contrôler via l'application DobotStudio ou via des micro-contrôleurs en utilisant des langages plus standards comme Python, C ou C++ [7].

2.1 Bras Dobot

Le bras robotique Dobot Magician est un système robotique programmable à 4 axes conçu pour les applications éducatives et de recherche. Le bras robotique est équipé de différents corps [7] :

- Un socle (base).
- Un bras arrière (rear arm).
- Un avant-bras (forearm).
- Un support permettant de fixer un outil (end-effector).

Le bras du robot comporte aussi 5 articulations, voir la figure suivante :

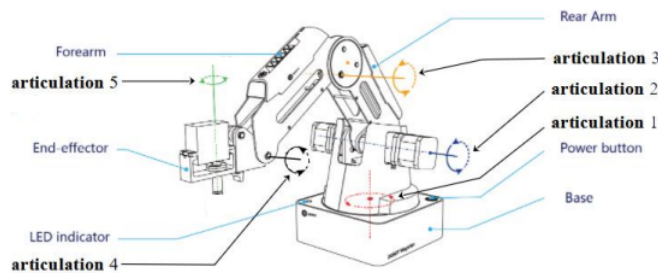


FIGURE 2.1 – Description du bras Dobot

- L'articulation 1 : Contrôle du moteur de la base.
- L'articulation 2 : Contrôle du moteur du bras arrière.
- L'articulation 3 : Contrôle du moteur d'avant-bras.
- L'articulation 4 : N'est pas contrôlable. Sa valeur est fixée mécaniquement en fonction des valeurs angulaires appliquées aux articulations 2 et 3, ceci afin que l'axe de l'articulation 5 soit toujours orthogonale au plan sur lequel repose la base du robot.
- L'articulation 5 : Contrôle du servo-moteur attaché à outil.

L'outil utilisé dans notre projet est la pince (Gripper), le servo-moteur associé à la pince pour activer l'articulation 5, il est connecté au connecteur GP3 de l'avant-bras, nous avons utilisé aussi une pompe à air pour contrôler l'ouverture et la fermeture de la pince, la

pompe est connecté à l'interface SW1 et GP1. La figure suivante montre les connexions nécessaires pour ces composants :



FIGURE 2.2 – Branchement de la Pince (Gripper) et la pompe.

2.1.1 Système de coordonnées du bras Dobot

Le Dobot Magician est équipé de deux types de systèmes de coordonnées, à savoir les systèmes de coordonnées des articulations et cartésiens, illustrés respectivement dans les deux figures suivantes [7] :

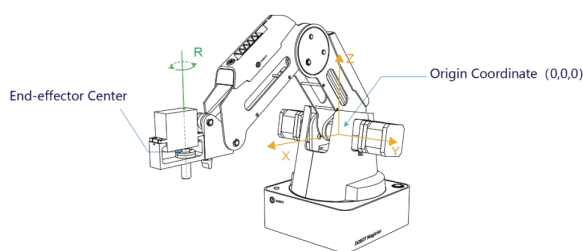


FIGURE 2.3 – systèmes de coordonnées cartésiens

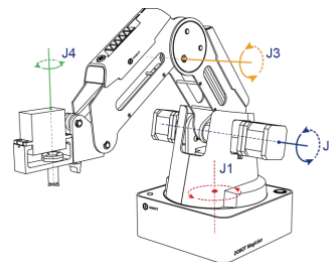


FIGURE 2.4 – Système de coordonnées des articulations

- **Système de coordonnées des articulations** : Les coordonnées sont déterminées par les axes de mouvement.
 - Si l'outil n'est pas installé, Dobot Magician contient trois axes : J1, J2 et J3, qui sont tous les axes rotatifs. La direction positive de ces axes est dans le sens anti-horaire.
 - Si l'outil avec servo est installé, tel que le kit de pince, Dobot Magician contient quatre articulations : J1, J2, J3 et J4, qui sont toutes les articulations rotatives. La direction positive de ces axes est dans le sens anti-horaire.
- **Système de coordonnées cartésiens** : Les coordonnées sont déterminées par la base.
 - L'origine est le centre des trois moteurs (bras arrière, avant-bras, base).
 - La direction de l'axe X est perpendiculaire à la base vers l'avant.
 - La direction de l'axe Y est perpendiculaire à la base vers la gauche.
 - La direction de l'axe Z est verticale vers le haut, qui est basée sur la règle de la main droite.
 - L'axe R est l'attitude du centre de servo par rapport à l'origine du bras robotique, dont la direction positive est dans le sens anti-horaire. L'axe R n'existe que lorsqu'un outil avec servo est installé.

2.1.2 Modes de mouvement

Le bras robotisé Dobot Magician offre trois modes de mouvement [7] :

- **Le mode de mouvement "Point To Point" (PTP)** : Ce mode comprend plusieurs types de mouvement, notamment MOVJ, MOVL et JUMP.
- **MOVJ (Joint movement)** : Ce type permet de déplacer le robot en définissant les angles de chaque articulation, le mouvement est effectué en faisant bouger tous les articulations en même temps pour atteindre la position désirée.
- **MOVL (Linear movement)** : Le bras du robot se déplace linéairement entre deux positions spécifiées dans l'espace de travail, le mouvement est effectué en créant un trajet linéaire entre les positions de départ et d'arrivée.

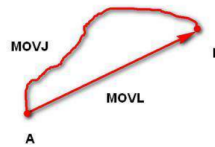


FIGURE 2.5 – Modes MOVJ et MOVL

- **JUMP** : Le robot saute directement à une position spécifiée dans l'espace de travail. Il ignore tous les points intermédiaires entre les positions de départ et d'arrivée et se déplace directement à l'emplacement cible.



FIGURE 2.6 – Mode JUMP

- **Le mode de mouvement en "Continuous Path" (CP)** : Est un autre mode de mouvement disponible pour le bras robotisé Dobot Magician, il permet au robot de se déplacer en suivant une trajectoire continue plutôt qu'en déplacement point à point, cela signifie que le robot peut suivre une courbe ou une ligne continue sans arrêt.

Ce mode est particulièrement utile pour les tâches qui nécessitent une trajectoire fluide, telles que le dessin ou la gravure.

- **Mouvement en mode "ARC"** : Le bras robotique Dobot Magician peut décrire des trajectoires courbes à partir d'une position de départ jusqu'à une position d'arrivée, en passant par une position intermédiaire.

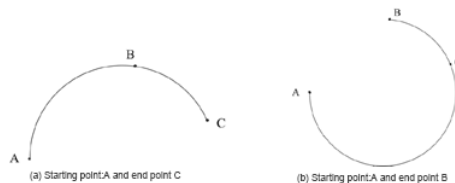


FIGURE 2.7 – Mode ARC

Afin de réaliser notre objectif qui est de développer des bibliothèques de fonctions booléennes pour contrôler de manière discrète notre bras robotisé Dobot, nous allons utiliser le mode de mouvement "Point To Point" (PTP). Ce mode de mouvement permet d'effectuer

des tâches qui nécessitent des mouvements précis et répétitifs, comme le tri et le placement de pièces.

2.1.3 Caractéristiques techniques du bras Dobot

La figure suivante résume les caractéristique de notre robot dobot magicien [7] :

Charge maximale	500 g
Atteignabilité maximale	320 mm
Latitude de l'articulation $q_1(-\dot{j}_1)$	$[-90^\circ; +90^\circ]$
Latitude de l'articulation $q_2(-\dot{j}_2)$	$[0^\circ; +85^\circ]$
Latitude de l'articulation $q_3(-\dot{j}_3)$	$[-10^\circ; +90^\circ]$
Latitude de l'articulation $q_5(-\dot{j}_4)$	$[-90^\circ; +90^\circ]$
Vitesse maximale (avec une charge de 250g)	Vitesse de rotation des articulations $\dot{j}_1, \dot{j}_2, \dot{j}_3 : 320^\circ / s$, Vitesse de rotation de l'articulation $\dot{j}_4 : 480^\circ / s$
Répétabilité	0.2 mm
Software	DobotStudio

FIGURE 2.8 – Caractéristique technique.

2.1.4 Logiciel DobotStudio

DoboStudio est un logiciel fournit par le constructeur du robot Dobot pour la réalisation de certaines tâches et la manipulation du robot telles que l'enseignement et la lecture de mouvements, l'écriture et le dessin, la programmation graphique avec Blockly et le contrôle par script[7]. Tout cela est présenté dans la figure.

Les modules de fonction sur la page DobotStudio sont :

- Enseignement et lecture : permet d'enregistrer et de reproduire les mouvements du Dobot Magician.
- Écriture et dessin : permet au Dobot Magician d'écrire et de dessiner en suivant une trajectoire précise.
- Programmation graphique avec Blockly : permet de programmer le Dobot Magician à l'aide de blocs graphiques.
- Contrôle de script : permet de programmer le Dobot Magician en utilisant un script personnalisé.
- Leap Motion : Prend en charge les mouvements de la main en tant qu'entrée pour contrôler le bras robotique via un contrôleur Leap Motion.
- Souris : Contrôle du bras robotique à l'aide d'une souris.
- Gravure laser : Gravure d'une image bitmap sur un objet à l'aide d'un laser.
- Imprimante 3D : Capable d'imprimer en 3D.
- Ajouter plus : Ajouter des fonctions personnalisées pour manipuler le bras robotique.



FIGURE 2.9 – Moniteur série de l'Arduino.

2.2 Description du convoyeur

Le convoyeur est un système automatisé de transport de matériaux largement utilisé dans diverses applications industrielles afin d'accroître l'efficacité de la production. Il se compose d'une bande de transport en PVC (tapis roulant) qui permet de déplacer des matériaux ou des pièces d'un emplacement à un autre à l'aide d'un moteur électrique en toute simplicité et efficacité [3] [9].

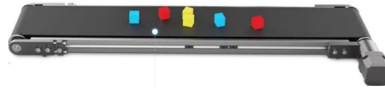


FIGURE 2.10 – Convoyeur

Dans le cadre de notre projet, nous avons utilisé deux tapis qui sont connectés au robot. Le premier tapis est relié au stepper 1 et le deuxième tapis est relié au stepper 2. Cette configuration nous donne une représentation visuelle de la liaison entre le robot et les tapis.

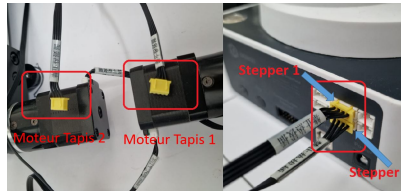


FIGURE 2.11 – Branchement du Convoyeur 1 et 2

2.3 Capteurs

2.3.1 Capteur de Proximité Photoélectrique Infrarouge

Le capteur photoélectrique IR est un dispositif électronique qui peut être utilisé dans de nombreuses applications pour détecter la présence ou l'absence d'objets. Ce type de capteur est souvent utilisé dans l'industrie, dans des systèmes de contrôle de processus, des machines de production automatisées, des portes automatiques, des systèmes de sécurité, et bien plus encore[1].

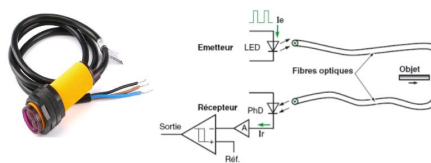


FIGURE 2.12 – Capteur de Proximité Photoélectrique Infrarouge SENS-97 E18-D80NK.

Le capteur utilise la lumière infrarouge pour détecter la présence d'objets. Il émet un faisceau lumineux sur l'objet à détecter, puis détecte la réflexion de la lumière infrarouge avec un détecteur photoélectrique. Le capteur dispose de trois fils, un fil d'alimentation (+V) pour connecter la tension d'alimentation (entre 6V et 36V), un fil de sortie (OUT) qui fournit un signal de sortie lorsque l'objet est détecté, ce fil est relié au collecteur du transistor NPN et un fil de masse (GND) pour connecter la masse du circuit [11].

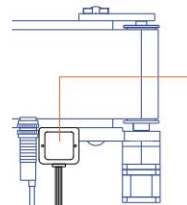
Pour notre projet, nous avons intégré un capteur infrarouge qui est connecté à GP4, comme illustré dans la figure ci-dessous.



FIGURE 2.13 – Branchement du Capteur Infrarouge

2.3.2 Capteur de couleur

Le bras robotisé Dobot Magiciancouleurs est équipé d'un capteur de couleur spécialement conçu pour le convoyeur. Ce capteur mesure la réflexion lumineuse émanant des objets colorés. Chaque couleur d'objet se traduit par une quantité de lumière réfléchie différente. Les résultats obtenus par le capteur sont donner par la table de correspondance des couleurs, la figure suivante nous donne une idée sur les couleurs détecter[8].



Color Information	I01	I02
Red	0	0
Green	0	1
Blue	1	0
None	1	1

FIGURE 2.14 – Couleur détecter par le capteur

En étant positionné sur le convoyeur, le capteur permet au bras robotique Dobot Magician de placer les pièces colorées au-dessus de lui. Selon la couleur détectée, le bras robotique est en mesure d'effectuer diverses actions telles que le tri, le traitement ou la manipulation des objets.

Dans le cadre de notre projet, nous avons utilisé ce capteur pour faire le tris des objet par couleur, ce capteur est lier a GP5, cette figure nous montre le câblage.



FIGURE 2.15 – Branchement du capteur de couleur.

Chapitre 3

Arduino et son environnement de développement

3.1 Présentation de la carte Arduino Mega 2560

Le système Arduino donne la possibilité d'allier les performances de la programmation à celles de l'électronique. Plus précisément, pour programmer des systèmes électroniques. Le gros avantage de l'électronique programmée c'est qu'elle simplifie grandement les schémas électroniques et par conséquent, le coût de la réalisation, mais aussi la charge de travail à la conception d'une carte électronique.

Arduino est un circuit imprimé en matériel libre sur lequel se trouve un micro-contrôleur qui peut être programmé pour analyser et produire des signaux électriques, de manière à effectuer des tâches très diverses comme la domotique (le contrôle des appareils domestiques - éclairage, chauffage...), le pilotage d'un robot, etc.

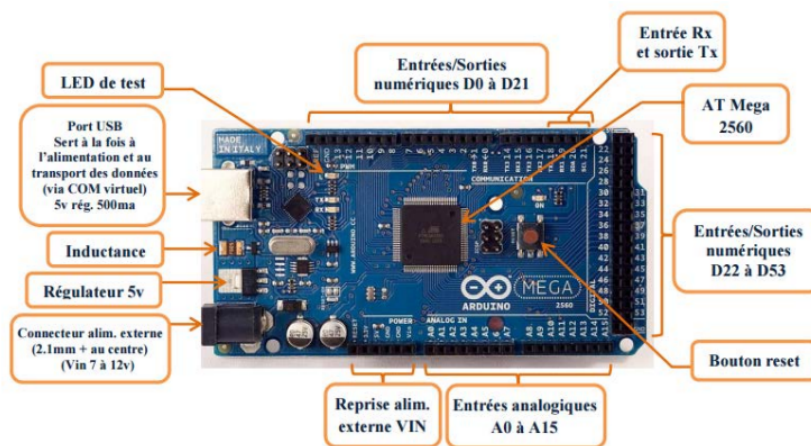


FIGURE 3.1 – Carte Arduino Mega 2560.

La carte Arduino Mega 2560 est basée sur un ATmega2560 cadencé à 16 MHz. Elle dispose de 54 E/S dont 14 PWM, 16 analogiques et 4 UARTs. Des connecteurs situés sur les bords extérieurs du circuit imprimé permettent d'enficher une série de modules complémentaires. Cette carte peut se programmer avec le logiciel Arduino disponible gratuitement. La figure suivante présente les caractéristiques de notre carte Arduino Mega 2560 :

Microcontrôleur	ATMEGA2560
Tension de fonctionnement	5V
Tension d'alimentation	7 à 12V
Broches E/S numérique	54 (dont 14 disposent de sortie PWM)
Broches d'entrées analogiques	16
Vitesse d'horloge	16 MHz
Mémoire programme Flash	25 6KB dont 8 KB utilisés en bootloader
Mémoire SRAM	8 KB
Mémoire EEPROM	4 KB

FIGURE 3.2 – Caractéristique de la carte Arduino Mega 2560.

3.1.1 Micro-contrôleur ATmega2560

Le micro-contrôleur ATmega2560 est une puce électronique qui intègre plusieurs composants complexes dans un espace restreint. Il sert de processeur pour la carte et gère les fonctions de calcul, d'exécution des instructions du programme et de gestion des ports d'entrée/sortie. En d'autres termes, il est responsable de la gestion des données et de la communication de la carte électronique. Son format compact en fait un choix populaire pour de nombreuses applications.

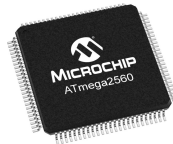


FIGURE 3.3 – Micro-contrôleur ATmega2560.

3.1.2 Alimentation de la carte

La carte Arduino Mega 2560 peut être alimentée soit via une connexion USB qui fournit une tension de 5V jusqu'à 500mA, soit par une alimentation externe. La carte sélectionne automatiquement la source d'alimentation.

La carte est conçue pour fonctionner avec une alimentation externe allant de 6 à 20 volts. Cependant, si la tension d'alimentation est inférieure à 7 volts, la broche 5V peut fournir moins de 5 volts, ce qui peut entraîner une instabilité de la carte. D'un autre côté, si la tension d'alimentation est supérieure à 12 volts, le régulateur de tension de la carte risque de surchauffer.

3.1.3 Entrées et sorties numériques

La carte Mega dispose de 54 broches numériques, chacune pouvant être utilisée comme entrée ou sortie numérique. La tension de fonctionnement des broches est de 5V et chacune d'entre elles peut fournir ou recevoir jusqu'à 40mA d'intensité. Par défaut, une résistance interne de rappel pull-up est désactivée, mais peut être activée en utilisant l'instruction `digitalWrite(broche, HIGH)` lorsque la broche est configurée en entrée.

3.1.4 Broches analogiques

La carte Mega2560 est équipée de 16 entrées analogiques qui peuvent fournir une mesure avec une résolution de 10 bits, soit 1024 niveaux allant de 0 à 1023, grâce à la fonction

analogRead() du langage Arduino. Par défaut, ces broches sont configurées pour mesurer des tensions entre 0V (valeur 0) et 5V (valeur 1023) avec la possibilité de les utilisées comme broches numériques.

3.2 Langage de programmation

La programmation s'effectue dans un langage propre à Arduino dont la structure s'apparente aux langages C/C++. Lorsqu'on évoque une fonction Arduino, on fait appel à une ou plusieurs bibliothèques rédigées en C ou C++ qui seront incluses à la compilation.

3.3 Environnement de développement (IDE)

Arduino Software (IDE) est une plate-forme open-source qui permet de facilement écrire du code pour les cartes Arduino et de les transférer. Cette plate-forme fonctionne sur les systèmes d'exploitation Windows, Mac OS X et Linux. Il est compatible avec toutes les cartes Arduino disponibles sur le marché.

Après avoir tapé ou modifié le programme, il peut être transféré et stocké dans la carte Arduino via une liaison USB. Le câble USB fournit à la fois l'énergie nécessaire à la carte et transfère le programme, le tout étant géré par l'ide Arduino. La figure suivante présente l'interface du logiciel Arduino :

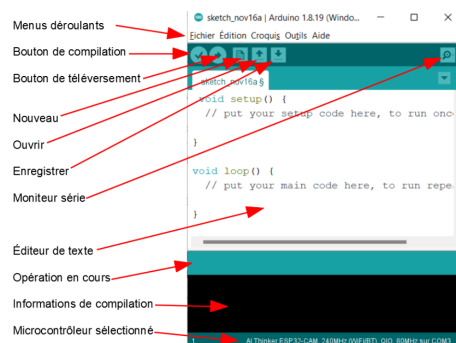


FIGURE 3.4 – L'interface du logiciel Arduino.

La fonction "setup" contient toutes les instructions nécessaires à l'initialisation du programme qui seront exécutées une seule fois lors de la mise sous tension de la carte. Dans la fonction principale les instructions sont répétées indéfiniment tant que l'Arduino fonctionne.

Aussi, il existe le moniteur série Arduino qui sert à visualiser les données envoyées ou reçues par la carte, ainsi qu'à déboguer et communiquer en temps réel entre la carte et l'ordinateur hôte.



FIGURE 3.5 – Moniteur série de l'Arduino.

Chapitre 4

Bibliothèque de commande

4.1 Analyse de la problématique

On souhaite développer une bibliothèque booléenne donc toutes nos fonctions doivent avoir un seul paramètre qui prend soit 0 ou 1.

On possède deux tapis un robot un capteur de couleur, un capteur de position (IR) fonctionnel et un autre qui ne marche pas.

Notre bibliothèque doit permettre de réaliser des scénarios de RDP. Pour que sa soit possible on doit développer des fonctions qui permettent de contrôler chaque élément de la maquette de façon booléenne, exemple : *DobotTapis1(par)*, une fonction qui permet de faire marcher et arrêter le tapis suivant la valeur de son paramètre d'entrée(0 : arrêt , 1 :marche).

Ce type de fonction permettra de les utiliser dans un RDP soit comme prédicat de nos transitions si nos fonctions ont un retour ou comme des actions si nos fonctions active ou désactive un actionneur comme la fonction expliquer précédemment.

Comme nos fonctions sont booléenne,pour bouger notre robot il est nécessaire d'avoir des positions pré-définie qui doivent avoir un sens dans un RDP, exemple : Position d'arrêt d'une pièce après sa détection par le capteur de position.

Nos coordonnées de positions seront toutes par rapport a notre repère de base,il faut avoir un moyen de les définir,Pour cela on utilisera le logiciel DobotStudio qui avec sont mode "Teaching Playback" permet de donner la position de l'organe terminal du bras a tout instant.

4.2 Étude des bibliothèques fournies par le constructeur

Le constructeur fournit des bibliothèques [10] permettant de contrôler et de gérer la communication entre l'Arduino et le bras Dobot Magician,la section suivante représente la description de ces bibliothèques :

FlexiTimer2 : La bibliothèque "FlexiTimer2" fournit des fonctions pour créer des temporisateurs flexibles pour Arduino. Ces temporisateurs peuvent être utilisés pour synchroniser les mouvements du bras robotique avec d'autres actions, telles que la lecture de capteurs ou la commande d'autres périphériques.

Message : Cette bibliothèque permet l'envoi et la réception de messages entre le bras robotique et l'Arduino. Les messages sont envoyés sous forme de paquets de données structurés.

Packet : Le fichier "Packet" fournit des fonctions pour créer des paquets de données à envoyer au bras robotique Dobot Magician. Ces paquets de données contiennent des

informations telles que l'identifiant de commande, les paramètres de commande, etc.

ProtocolDef : La bibliothèque "ProtocolDef" fournit des définitions pour le protocole de communication utilisé entre le bras robotique Dobot Magician et Arduino. Cette bibliothèque définit les formats de messages et de paquets de données utilisés pour communiquer avec le bras.

Protocol : La bibliothèque "Protocol" fournit des fonctions pour gérer la communication entre le bras robotique Dobot Magician et Arduino, en utilisant le protocole défini par "ProtocolDef". Cette bibliothèque gère les connexions série avec le bras et s'assure que les messages et les paquets de données sont correctement formatés et envoyés.

ProtocolID : La bibliothèque "ProtocolID" fournit des identificateurs pour les différentes commandes et messages utilisés dans le protocole de communication. Ces identificateurs permettent à Arduino de reconnaître les commandes envoyées par le bras et de les traiter correctement.

RingBuffer : La bibliothèque "RingBuffer" fournit une implémentation de buffer circulaire pour Arduino. Ce buffer circulaire peut être utilisé pour stocker les données à envoyer au bras robotique Dobot Magician. Cette bibliothèque assure que les données sont stockées de manière efficace et que le bras reçoit toutes les données envoyées.

Symbol : La bibliothèque "Symbol" fournit des symboles utilisés dans le protocole de communication, tels que des codes pour différentes actions du bras robotique. Ces symboles permettent à Arduino de comprendre les commandes envoyées par le bras et de les traiter correctement.

Type : La bibliothèque "Type" fournit des définitions de types de données utilisés dans les autres bibliothèques.

Command : La bibliothèque "Command" fournit des fonctions pour envoyer des commandes spécifiques au bras robotique Dobot Magician via la liaison série. Les fonctions de cette bibliothèque permettent de déplacer le bras robotique vers une position donnée, de configurer la pince et d'autres fonctions spécifiques du bras robotique. (Elle est utilisée pour contrôler les mouvements du robot, régler les paramètres et effectuer d'autres actions).

Mais ces fonctions ne sont pas booléennes donc on doit les modifier en créant une bibliothèque contenant des fonctions booléennes, la section suivante représente la description des fonctions qu'on a utilisé pour atteindre notre objectif :

SetQueuedCmdStartExec() : Est une fonction qui envoie une commande au robot Dobot Magician pour qu'il commence à exécuter les commandes mises en file d'attente. Voici ce que fait la fonction :

- Elle appelle "INIT_MESSAGE()" qui initialise la variable "gMessage" de type structure "Message" définie dans le fichier "Message.h" avec les valeurs par défaut. Cette structure contient les informations nécessaires pour envoyer une commande au Dobot Magician.
- Elle affecte la valeur "ProtocolQueuedCmdStartExec" à l'attribut "id" de "gMessage". Cette valeur est une constante définie dans "Protocol.h" qui représente le code de la commande de démarrage de l'exécution des commandes en file d'attente.
- Elle affecte la valeur "false" aux attributs "rw" et "isQueued" de "gMessage". Ces valeurs indiquent que la commande est en lecture seule et qu'elle n'est pas mise en file d'attente.
- Elle affecte la valeur 0 à l'attribut paramsLen de "gMessage". Cette valeur indique que la commande n'a pas de paramètres supplémentaires.
- Elle appelle la fonction "WaitCmdEcho()" qui attend que le robot Dobot Magician renvoie un message d'accusé de réception.
- Elle retourne la valeur "true" pour indiquer que la fonction a réussi.

WaitQueuedCmdFinished() : Est une fonction qui attend que le robot Dobot Magician termine d'exécuter toutes les commandes mises en file d'attente. Voici ce que fait la fonction :

- Elle entre dans une boucle infinie qui s'exécute tant que le robot n'a pas fini d'exécuter les commandes en file d'attente.
- Elle appelle la fonction "delay(50)" qui met en pause le programme pendant 50 millisecondes.
- Elle appelle la fonction "GetQueuedCmdCurrentIndex()" qui renvoie l'index de la commande que le robot est en train d'exécuter. Cette fonction met à jour la variable globale "gQueuedCmdCurrentIndex" avec cette valeur.
- Elle compare la valeur de "gQueuedCmdCurrentIndex" avec la valeur de "gQueuedCmdWriteIndex", qui est une variable globale qui représente l'index de la dernière commande ajoutée à la file d'attente. Si ces deux valeurs sont égales ou si "gQueuedCmdCurrentIndex" est supérieure à "gQueuedCmdWriteIndex", cela signifie que le robot a terminé d'exécuter toutes les commandes en file d'attente et qu'il peut sortir de la boucle.
- Elle ne retourne aucune valeur.

GetQueuedCmdCurrentIndex() : Est une fonction qui renvoie l'index de la commande que le robot Dobot Magician est en train d'exécuter. Voici ce que fait la fonction :

- Elle appelle "INIT_MESSAGE()" qui initialise la variable "gMessage" de type structure "Message" définie dans le fichier "Message.h" avec les valeurs par défaut.
- Elle affecte la valeur "ProtocolQueuedCmdCurrentIndex" à l'attribut "id" de "gMessage". Cette valeur est une constante qui représente le code de la commande de demande de l'index courant.
- Elle affecte la valeur "ProtocolQueuedCmdCurrentIndex" à l'attribut "id" de "gMessage". Cette valeur est une constante qui représente le code de la commande de demande de l'index courant.
- Elle affecte la valeur "false" aux attributs "rw" et "isQueued" de "gMessage". Ces valeurs indiquent que la commande est en lecture seule et qu'elle n'est pas mise en file d'attente.
- Elle affecte la valeur 0 à l'attribut "paramsLen" de "gMessage". Cette valeur indique que la commande n'a pas de paramètres supplémentaires.
- Elle appelle la fonction "WaitCmdEcho()" qui attend que le robot Dobot Magician renvoie un message d'accusé de réception contenant l'index courant.
- Elle copie la valeur renvoyée par le robot dans la variable globale "gQueuedCmdCurrentIndex", qui est un entier de 64 bits qui représente l'index courant.
- Elle retourne la valeur "true" pour indiquer que la fonction a réussi.

SetHomeCmd() : Cette fonction est utilisée pour envoyer une commande au Dobot Magician pour qu'il retourne à sa position d'origine (position "Home" ou position de référence). Voici les étapes de la fonction :

- Elle initialise un message avec "INIT_MESSAGE()".
- Elle affecte à l'attribut "id" du message la valeur "ProtocolHOMECmd", qui est une constante définie dans le fichier "ProtocolID.h".
- Elle affecte à l'attribut "rw" du message la valeur "true", ce qui signifie que le message est en mode écriture et non lecture.
- Elle affecte à l'attribut "isQueued" du message la valeur "true", ce qui signifie que le message est mis en file d'attente et non exécuté immédiatement.
- Elle affecte à l'attribut "paramsLen" du message la valeur 0, ce qui signifie que le message n'a pas de paramètres supplémentaires.

- Elle appelle la fonction "WaitCmdEcho()", qui attend que le Dobot Magician renvoie une confirmation de réception du message.
- Elle copie la valeur renvoyée par le Dobot Magician, qui est un entier de 64 bits représentant l'index de la commande dans la file d'attente, dans la variable globale "gQueuedCmdWriteIndex", en utilisant la fonction "memcpy()" et le pointeur "gParamsPointer".
- Elle renvoie la valeur "true", ce qui signifie que la fonction a réussi.

SetPTPCmd(PTPCmd *ptpCmd) : Elle permet d'envoyer une commande de positionnement au bras Dobot Magician pour qu'elle se déplace d'un point A à un point B dans l'espace de travail. Elle prend en paramètre un pointeur sur une structure "PTPCmd" qui contient les coordonnées de la position cible (X, Y et Z) et le mode de déplacement (JUMP, MOVJ ou MOVL). Voici ce que fait la fonction :

- Elle appelle "INIT_MESSAGE()" qui initialise (met à zéro) tous les champs du message (structure "gMessage").
- Elle remplit la structure "gMessage" avec les informations de la commande "PTP" : l'identifiant du protocole "ProtocolPTPCmd", le mode de lecture/écriture ("rw" = "true"), le mode de file d'attente ("isQueued" = "true") et la longueur des paramètres "sizeof(PTPCmd)".
- Elle copie les données du pointeur "ptpCmd" vers le champ "params" de la structure "gMessage" avec la fonction "memcpy()".
- Elle appelle la fonction "WaitCmdEcho()" qui attend que le bras robotique confirme la réception de la commande.
- Elle copie la valeur du pointeur "gParamsPointer" vers la variable globale "gQueuedCmdWriteIndex" avec la fonction "memcpy()". Cette valeur correspond à l'index de la commande dans la file d'attente du bras robotique.
- Elle renvoie "true" pour indiquer que la fonction a réussi.

SetIOMultiplexing(IOConfig *ioConfig) : Elle permet de configurer le mode de fonctionnement des ports d'entrée/sortie du Dobot Magician. Elle prend en paramètre un pointeur sur une structure "IOConfig" qui contient les informations de configuration. Voici ce que fait la fonction :

- Elle appelle "INIT_MESSAGE()" qui initialise (met à zéro) tous les champs du message (structure "gMessage").
- Elle remplit la structure "gMessage" avec les informations de la configuration d'entrée/sortie : l'identifiant du protocole "ProtocolIOMultiplexing", le mode de lecture/écriture "true", le mode de file d'attente "false" et la longueur des paramètres "sizeof(IOConfig)".
- Elle copie les données du pointeur "ioConfig" vers le champ "params" de la structure "gMessage" avec la fonction "memcpy()".
- Elle appelle la fonction "WaitCmdEcho()" qui attend que le bras robotique confirme la réception de la commande.
- Elle renvoie "true" pour indiquer que la fonction a réussi.

GetIODI(EIODI *eIODI) : Elle permet de récupérer l'état des entrées numériques d'entrée/sortie (IO) du Dobot Magician. Elle prend en paramètre un pointeur sur une structure "EIODI" qui contient l'état d'une entrée/sortie numérique du bras robotique Dobot. Voici ce que fait la fonction :

- Elle appelle "INIT_MESSAGE()" qui initialise (met à zéro) tous les champs du message (structure "gMessage").
- Elle remplit la structure "gMessage" avec les informations de la lecture d'entrée/sortie

numérique : l'identifiant du protocole "ProtocolIODI", le mode de lecture/écriture "false" (lecture seule), le mode de file d'attente "false" (non-queue) et la longueur des paramètres "sizeof(EIODI)".

- Elle copie les données du pointeur "eIODI" vers le champ "params" de la structure "gMessage" avec la fonction "memcpy()".
- Elle appelle la fonction "WaitCmdEcho()" qui attend que le bras robotique renvoie l'état de l'entrée/sortie numérique.
- Elle copie les données du pointeur "gParamsPointer" vers le pointeur "eIODI" avec la fonction "memcpy()". Ces données correspondent à l'état de l'entrée/sortie numérique (0 ou 1).
- Elle renvoie true pour indiquer que la fonction a réussi.

SeEndEffectorGripper(EndEffectorGripper *endEffectorGripper) : Elle est utilisée pour configurer le gripper (pince) de l'extrémité de l'effecteur du robot. Elle prend en paramètre un pointeur sur une structure "EndEffectorGripper" qui contient les informations de contrôle du gripper (pince). Voici ce que fait la fonction :

- Elle appelle "INIT_MESSAGE()" qui initialise (met à zéro) tous les champs du message (structure "gMessage").
- Elle remplit la structure "gMessage" avec les informations de contrôle du gripper : l'identifiant du protocole "ProtocolEndEffectorGripper", le mode de lecture/écriture "true" (écriture seule), le mode de file d'attente "false" (non-queue) et la longueur des paramètres "sizeof(EndEffectorGripper)".
- Elle copie les données du pointeur "endEffectorGripper" vers le champ "params" de la structure "gMessage". Ces données contiennent deux champs : "isEnabled", qui indique si le gripper est activé ou non, et "isGriped", qui indique si le gripper est fermé ou ouvert (ces deux paramètres booléens sont copiés dans les deux premiers éléments du tableau "params").
- Elle appelle la fonction "WaitCmdEcho()" qui attend que le bras robotique confirme la réception de la commande.
- Elle renvoie "true" pour indiquer que la fonction a réussi.

SetEMotor(EMotor *eMotor) : Elle permet de contrôler un moteur externe connecté au bras Dobot. Elle prend en paramètre un pointeur sur une structure "EMotor" qui contient les informations de contrôle. Voici ce que fait la fonction :

- Elle appelle "INIT_MESSAGE()" qui initialise (met à zéro) tous les champs du message (structure "gMessage").
- Elle remplit la structure "gMessage" avec les informations de contrôle du moteur externe : l'identifiant du protocole "ProtocolEMotor", le mode de lecture/écriture "true" (écriture seule), le mode de file d'attente "false" (non-queue) et la longueur des paramètres "sizeof(EMotor)".
- Elle copie les données du pointeur "eMotor" vers le champ "params" de la structure "gMessage" avec la fonction "memcpy()". Ces données contiennent trois champs : "index", qui indique le numéro du port du moteur externe (0 ou 1), "isEnabled", qui indique si le moteur externe est activé ou non, et "speed", qui indique la vitesse du moteur externe en tours par minute.
- Elle appelle la fonction "WaitCmdEcho()" qui attend que le bras robotique confirme la réception de la commande.
- Elle renvoie "true" pour indiquer que la fonction a réussi.

4.3 Différentes positions de notre robot

Pour pouvoir créer les fonctions de notre bibliothèque et garantir sa cohérence avec la maquette, il faut avoir les coordonnées de déplacement du robot entre les différentes positions permettant la réalisation du scénario souhaité vu que notre étude est basé sur une maquette fixe. Donc, nous avons déterminé six positions, qui sont illustrées dans la figure ci-dessous.

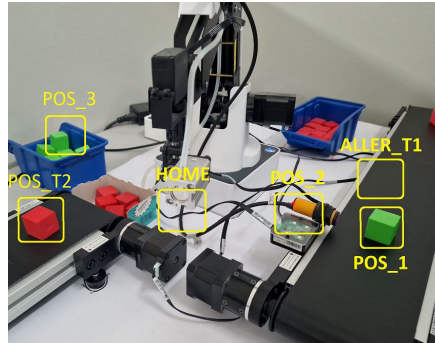


FIGURE 4.1 – Les positions du bras.

Description des positions :

- Home : position initiale du bras (état de repos).
- ALLER-T1 : position située au-dessus du tapis 1.
- POS-1 : position en face du capteur IR (infra rouge).
- POS-2 : position au-dessus du capteur couleur.
- POS-3 : position dans la zone des pièces vertes.
- POS-T2 : position sur le tapis 2 (zone d'éjection des pièces rouges).

4.4 Solution proposée et sa mise en œuvre :

On detailera dans cette partie notre bibliothèque élémentaire développée. Notre bibliothèque contient des fonctions booléennes qui permettent de contrôler notre maquette.

4.4.1 Dobot_SetHOMECmd()

Cette fonction ramène le robot à sa position initiale.

Prototype	void Dobot_SetHOMECmd(void)
Description	ramène le robot à une position initiale.
Parameter	void
Return	Void

FIGURE 4.2 – Fonction Dobot_SetHOMECmd()

Description de la fonction

```
FUNCTION Dobot_SetHOMECmd(void) : Void
DEBUT
SetHomeCmd();
WaitQueuedCmdFinished();
FIN
```

4.4.2 Dobot_Init()

Cette fonction Permet de configurer le fonctionnement des ports de notre robot.

Prototype	void Dobot_Init()
Description	Initialise la communication entre l'Arduino et le robot
Parameter	Void
Return	Void

FIGURE 4.3 – Fonction Dobot_Init()

Description de la fonction

FONCTION Dobot_Init() : Void

DEBUT

```
SERIALNUM.begin(115200);
```

```
delay(1000);
```

```
ProtocolInit();
```

```
Serial.println("Dobot Init");
```

```
Pour i de 0 a 20 faire :
```

```
i++
```

```
Dobot_SetIOMultiplexing(i,IOFunctionDI);
```

```
fin Pour
```

```
SetQueuedCmdStartExec();
```

FIN

Notre fonction initialise la communication entre Serial2 (SERIALNUM=Serial2) de l'Arduino et le robot, elle initialise aussi toutes les IO du robot comme des entrées numériques.

4.4.3 Dobot_SetPTPCmd()

Cette fonction Permet de bouger le robot d'un point vers un autre.

Prototype	void Dobot_SetPTPCmd(uint8_t Model,float x,float y,float z,float r)
Description	déplace le bras d'un point A a un point B donnée.
Parameter	Model : Mode de mouvement du robot, x,y,z,r : coordonnée du point B
Return	Void

FIGURE 4.4 – Fonction Dobot_SetPTPCmd()

Description de la fonction

FONCTION Dobot_SetPTPCmd(uint8_t Model,float x,float y,float z,float r) : Void

Var ptpCmd : PTPCmd;

DEBUT

```
ptpCmd.ptpMode = Model; // Mode de mouvement
```

```
ptpCmd.x = x; // coordonnée x de la position cible
```

```
ptpCmd.y = y; // coordonnée Y de la position cible
```

```
ptpCmd.z = z; // coordonnée Z de la position cible
```

```
ptpCmd.rHead = r; // orientation de l'organe terminale
```

```
SetPTPCmd(&pipâmes;
```

```
WaitQueuedCmdFinished();
```

FIN

Notre fonction remplit une structure `ptpCmd` qui est elle mise comme argument (Passage par adresse) dans la fonction `SetPTPCmd()` définie dans la bibliothèque commande cette dernière est expliquée précédemment nous permet de communiquer nos paramètres avec notre maquette a travers le protocole de communication série.

4.4.4 Dobot_GetIODI()

Cette fonction Permet de configurer le fonctionnement des ports de notre robot.

Prototype	<code>uint8_t Dobot_GetIODI(uint8_t address)</code>
Description	Donne le Résultat de l'entrée DI (Digital input)
Parameter	address : (1 a 20)
Return	0 ou 1

FIGURE 4.5 – Fonction `Dobot_GetIODI()`

Description de la fonction

```

FONCTION Dobot_GetIODI(address : uint8_t) : Void
VAR eIODI : EIODI;
DEBUT
eIODI.address = address;
GetIODI(&eIODI);
RETOURNER(eIODI.value);
FIN

```

Notre fonction remplit une structure `eIODI` qui est elle mise comme argument (Passage par adresse) dans la fonction `GetIODI()` définie dans la bibliothèque commande cette dernière est expliquée précédemment nous permet de communiquer nos paramètres avec notre maquette a travers le protocole de communication série, la fonction retourne la valeur de l'entrée numérique.

4.4.5 Dobot_SetIOMultiplexing()

Cette fonction Permet de configurer le fonctionnement des ports de notre robot.

Prototype	<code>void Dobot_SetIOMultiplexing(uint8_t address,uint8_t function)</code>
Description	Permet de choisir le fonctionnement des ports de notre robot
Parameter	address : (1 a 20) address du port a configurer, function :PWM,DI,AI,ADC
Return	Void

FIGURE 4.6 – Fonction `Dobot_SetIOMultiplexing()`

Description de la fonction

```

FONCTION Dobot_SetIOMultiplexing(address : uint8_t,function : uint8_t) : Void
VAR iOConfig : IOConfig;
DEBUT
iOConfig.address = address; //Adresse de l'IO (1 a 20)
iOConfig.function = (IOFunction)function; //Fonctionnement de l'IO (PWM,DI,AI..)
SetIOMultiplexing(&iOConfig);
FIN

```


Notre fonction remplit une structure `iOconfig` qui est elle mise comme argument (Passage par adresse) dans la fonction `SetIOMultiplexing()` définie dans la bibliothèque commande cette dernière est expliquée précédemment nous permet de communiquer nos paramètres avec notre maquette a travers le protocole de communication série.

4.4.6 Dobot_Pince()

Cette fonction booléenne nous permet de contrôler notre organe terminal (Pince).

Prototype	<code>void Dobot_Pince(bool isGriped)</code>
Description	Permet l'ouverture ou fermeture de la pince
Parameter	0 : Ouvrir Pince, 1 : Fermer Pince
Return	Void

FIGURE 4.7 – Fonction `Dobot_Pince()`

Description de la fonction

```

FONCTION Dobot_Pince(isGriped : Bool) : Void
VAR Pince : EndEffectorGripper;
DEBUT
Pince.isEnabled = 1; // Active la Pince
Pince.isGriped = isGriped; // 0 : Ouvrir Pince, 1 : Fermer Pince
SeEndEffectorGripper(&Pince);
RETOURNER(Void)
FIN

```

Notre fonction remplit une structure `Pince` qui est elle mise comme argument (Passage par adresse) dans la fonction `SeEndEffectorGripper()` définie dans la bibliothèque commande cette dernière est expliquée précédemment nous permet de communiquer nos paramètres avec notre maquette a travers le protocole de communication série.

4.4.7 Dobot_Tapis1()

Cette fonction booléenne nous permet de contrôler le moteur lié au tapis1 soit en le mettant en marche ou en arrêt.

Prototype	<code>void Dobot_Tapis1(bool enable)</code>
Description	Permet de mettre le tapis1 en Marche ou en arrêt
Parameter	0 : Arrêt Tapis1, 1 : Marche Tapis1
Return	Void

FIGURE 4.8 – Fonction `Dobot_Tapis1()`

Description de la fonction

```

FONCTION Dobot_Tapis1(enable : Bool) : Void
VAR eMotor : EMotor;
DEBUT
eMotor.address = 0; // adresse de Stepper1 ou le moteur du tapis1 est branché
eMotor.enable = enable; // 1 : Activer, 0 : Désactiver
eMotor.speed = 4000; // Vitesse de notre moteur

```

```
SetEMotor(&eMotor);
RETOURNER(Void)
FIN
```

Notre fonction remplit une structure eMotor qui est elle mise comme argument (Passage par adresse) dans la fonction SetEMotor() définie dans la bibliothèque commande cette dernière est expliquée précédemment nous permet de communiquer nos paramètres avec notre maquette a travers le protocole de communication série.

4.4.8 Dobot_Tapis2()

Cette fonction booléenne nous permet de contrôler le moteur lié au tapis2 soit en le mettant en marche ou en arrêt.

Prototype	void Dobot_Tapis2(bool enable)
Description	Permet de mettre le tapis2 en Marche ou en arrêt
Parameter	0 : Arrêt Tapis2, 1 : Marche Tapis2
Return	Void

FIGURE 4.9 – Fonction Dobot_Tapis2()

Description de la fonction

```
FONCTION Dobot_Tapis2(enable : Bool) : Void
VAR eMotor : EMotor;
DEBUT
eMotor.address = 1; // adresse de Stepper2 ou le moteur du tapis2 est branché
eMotor.enable = enable; // 1 : Activer, 0 : Désactiver
eMotor.speed = 4000; // Vitesse de notre moteur
SetEMotor(&eMotor);
RETOURNER(Void)
FIN
```

Notre fonction remplit une structure eMotor qui est elle mise comme argument (Passage par adresse) dans la fonction SetEMotor() définie dans la bibliothèque commande cette dernière est expliquée précédemment nous permet de communiquer nos paramètres avec notre maquette a travers le protocole de communication série.

4.4.9 Dobot_Piece()

Cette fonction booléenne nous permet de détecter la présence d'une pièce sur le Tapis, elle retourne 0 ou 1.

Prototype	uint8_t Dobot_Piece()
Description	Permet la détection d'une pièce sur le Tapis
Parameter	Void
Return	0 : Pièce détectée , 1 : Pièce non détectée

FIGURE 4.10 – Fonction Dobot_Piece()

Description de la fonction

```
FONCTION Dobot_Piece(Void) : uint8_t
```

```

VAR state : uint8_t ;
DEBUT
Dobot_SetIOMultiplexing(7,IOFunctionDI) ; //Set IO7(ADC-GP4) en entrée
state=Dobot_GetIODI(7) ; //Résultat de l'entrée IO7
RETOURNER(state) ;
FIN

```

Notre fonction Définie le port IO7 de GP4 ou le capteur de Position (IR) est branché comme un port d'entrée numérique,et récupère sa valeur, si la fonction retourne 0 une pièce est détectée si le capteur retourne 1 aucune pièce est détectée.

4.4.10 Dobot_Getcolor()

Cette fonction nous permet de détecter la couleur d'une pièce.

Prototype	uint8_t Dobot_Getcolor()
Description	Permet la detection de la couleur d'une pièce
Parameter	Void
Return	00 :Red, 01 : Green, 10 : Blue 11 :None

FIGURE 4.11 – Fonction Dobot_Getcolor()

Description de la fonction

```

FONCTION Dobot_Getcolor(Void) : uint8_t
VAR res1,res2 : uint8_t ;
DEBUT
Dobot_SetIOMultiplexing(4,IOFunctionDI) ; //Set IO4(PWM-GP5) en entrée
Dobot_SetIOMultiplexing(5,IOFunctionDI) ; //Set IO5(ADC-GP5) en entrée
res1=Dobot_GetIODI(4) ; // donne la Valeur de l'entrée IO4
res2=Dobot_GetIODI(5) ; // donne la Valeur de l'entrée IO5
si (res1== 0 ET res2== 0) alors
RETOURNER(0) // Red
Fin si
si (res1== 0 ET res2== 1) alors
RETOURNER(1) //Green
Fin si
si (res1== 1 ET res2== 0) alors
RETOURNER(2) //Blue
Fin si
Sinon
RETOURNER(3)
Fin sinon
FIN

```

Notre fonction Définie les ports IO4 et IO5 de GP5 ou le capteur de couleur est branché comme des ports d'entrée,et récupère leurs valeurs, selon la valeur récupérer on a la couleur correspondante.

4.4.11 Dobot_AllerTapis1()

Cette fonction booléenne nous permet de déplacer l'organe terminal vers une position prédéfinie sur le tapis1.

Prototype	void Dobot_AllerTapis1(bool state)
Description	Permet de déplacer le robot vers la position de la pièce sur le tapis1
Parameter	1 : déplacement vers la position, 0 : Non
Return	Void

FIGURE 4.12 – Fonction Dobot_AllerTapis1()

Description de la fonction

FONCTION Dobot_AllerTapis1(state : bool) : Void

DEBUT

si (state == 1) alors

Dobot_SetPTPCmd(JUMP_XYZ,X1,Y1,Z1,R1);

fin si

FIN

Notre fonction appelle une autre fonction qui permet au bras de se déplacer vers la position "ALLER-T1" prédéfinie.

4.4.12 Dobot_AllerTapis2()

Cette fonction booléenne nous permet de déplacer l'organe terminal vers une position prédéfinie sur le tapis2.

Prototype	void Dobot_AllerTapis2(bool state)
Description	Permet de déplacer le robot vers une position prédéfinie sur le tapis2
Parameter	1 : déplacement vers la position, 0 : Non
Return	Void

FIGURE 4.13 – Fonction Dobot_AllerTapis2()

Description de la fonction

FONCTION Dobot_AllerTapis2(state : bool) : Void

DEBUT

si (state == 1) alors

Dobot_SetPTPCmd(JUMP_XYZ,X5,Y5,Z5,R5);

fin si

FIN

Notre fonction appelle une autre fonction qui permet au bras de se déplacer vers la position "POS-T2" prédéfinie.

4.4.13 Dobot_AllerPosHOME()

Cette fonction booléenne nous permet de déplacer l'organe terminal vers une position de repos prédéfinie.

Prototype	void Dobot_AllerPosHome(bool state)
Description	Permet de déplacer le robot vers une position Home prédéfinie
Parameter	1 : déplacement vers la position, 0 : Non
Return	Void

FIGURE 4.14 – Fonction Dobot_AllerPosHOME()

Description de la fonction

```

FONCTION Dobot_AllerTapis2(state : bool) : Void
DEBUT
si (state == 1) alors
Dobot_SetPTPCmd(JUMP_XYZ,X6,Y6,Z6,R6) ;
fin si
FIN

```

Notre fonction appelle une autre fonction qui permet au bras de se déplacer vers la position "Home" prédéfinie.

4.4.14 Dobot_AllerPos1()

Cette fonction booléenne nous permet de déplacer l'organe terminal vers une position prédéfinie de la pièce.

Prototype	void Dobot_AllerPos1(bool state)
Description	Permet de déplacer le robot vers la position prédéfinie de prise d'une pièce
Parameter	1 : déplacement vers la position, 0 : Non
Return	Void

FIGURE 4.15 – Fonction Dobot_AllerPos1()

Description de la fonction

```

FONCTION Dobot_AllerPos1(state : bool) : Void
DEBUT
si (state == 1) alors
Dobot_SetPTPCmd(JUMP_XYZ,X2,Y2,Z2,R2) ;
fin si
FIN

```

Notre fonction appelle une autre fonction qui permet au bras de se déplacer vers la position "POS_1" prédéfinie.

4.4.15 Dobot_AllerPos2()

Cette fonction booléenne nous permet de déplacer l'organe terminal vers une position prédéfinie de prise de la pièce.

Prototype	void Dobot_AllerPos2(bool state)
Description	Permet de déplacer l'organe terminale du robot vers le capteur de couleur
Parameter	1 : déplacement vers la position, 0 : Non
Return	Void

FIGURE 4.16 – Fonction Dobot_AllerPos2()

Description de la fonction

```
FONCTION Dobot_AllerPos2(state : bool) : Void
DEBUT
si (state == 1) alors
Dobot_SetPTPCmd(JUMP_XYZ,X3,Y3,Z3,R3) ;
fin si
FIN
```

Notre fonction appelle une autre fonction qui permet au bras de se déplacer vers la position "POS_2" prédéfinie.

4.4.16 Dobot_AllerPos3()

cette fonction booléenne nous permet de déplacer l'organe terminal vers une position prédéfinie du lot des pièces vertes.

Prototype	void Dobot_AllerPos2(bool state)
Description	Permet de déplacer l'organ terminale du robot vers la position du lot des pièces vertes
Parameter	1 : déplacement vers la position, 0 : Non
Return	Void

FIGURE 4.17 – Fonction Dobot_AllerPos3()

Description de la fonction

```
FONCTION Dobot_AllerPos3(state : bool) : Void
DEBUT
si (state == 1) alors
Dobot_SetPTPCmd(JUMP_XYZ,X4,Y4,Z4,R4) ;
fin si
FIN
```

Notre fonction appelle une autre fonction qui permet au bras de se déplacer vers la position "POS_3" prédéfinie.

4.5 Implémentation de commandes à partir de modèles (RdP, Grafcet)

Une des fonctionnalités clés de notre projet est l'implémentation de commandes à partir de modèles tels que les Réseaux de Pétri et les Grafcet. Cette fonctionnalité permet de rendre le système plus flexible et plus efficace en automatisant les mouvements du bras robot et du tapis roulant en fonction des étapes existantes.

Nous avons choisi d'utiliser les Réseaux de Pétri et les Grafcet pour représenter graphiquement les étapes de production et les transitions entre ces étapes. Nous avons créé des modèles de Grafcet pour représenter des séquences de mouvements à effectuer. À chaque étape, le micro-contrôleur envoie les commandes nécessaires pour déplacer le bras robot et activer et désactiver le tapis roulant et les capteurs en fonction du modèle.

L'implémentation de ces modèles nécessitait la création de fonctions spécifiques dans la bibliothèque de commandes pour interpréter les graphes de ces modèles. Les commandes nécessaires à chaque étape ont été programmées en conséquence.

Cette fonctionnalité a permis de rendre le système plus arrangeant. Les changements dans les processus peuvent être plus facilement pris en compte en ajustant simplement le

modèle. De plus, l'automatisation a permis de réduire les erreurs humaines et d'optimiser les temps.

Dans l'ensemble, l'implémentation de commandes à partir de modèles tels que les Réseaux de Pétri et les Grafctet a apporté une grande valeur ajoutée à notre projet. Elle a permis d'automatiser les mouvements du bras robot et du tapis roulant en fonction des étapes envisagé pour la maquette (par exemple système de production, de tri ou d'assemblage), rendant le système plus flexible et plus efficace pour simuler des applications industrielles.

Un scénario est présenté dans la partie suivante basé sur la modélisation par réseau de Pétri qui est utiliser pour tester la fiabilité et la efficacité de notre bibliothèque.

Réalisation pratique

L'élaboration de cette partie est fait dans la salle de TP I3. En premier lieu, on a testé le fonctionnement des différents éléments de la maquette (les capteurs et le convoyeur). Cette phase est importante avant de procéder à des tests plus avancés. Après, on s'est assuré de la communication série entre le robot et l'Arduino en réalisant des tâches simples. Cette étape permet de détecter et de corriger rapidement les éventuels problèmes avant de procéder aux tests plus avancés du projet. Une fois validé, on a passé à la phase des tests de nos bibliothèques, en traduisant un scénario basé sur une modélisation par réseau de Pétri (RdP) en code contenant les fonctions booléennes qui font partie de nos bibliothèques.

5.1 Protocole de communication série

Le protocole UART (Universal Asynchronous Receiver/Transmitter) est une technique de communication de données série qui permet à deux dispositifs électroniques de transmettre et de recevoir des données via une seule ligne de communication, appelée ligne série. Dans Notre projet nous utilisons le port serial2 de l'arduino.

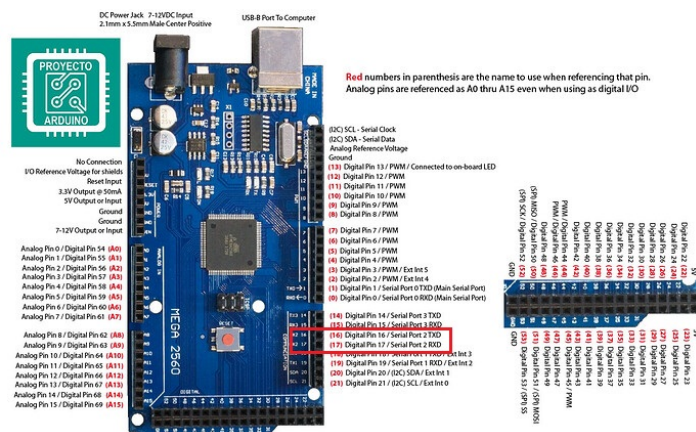


FIGURE 5.1 – Serial 2 [2]

5.2 Branchement entre l'arduino et Dobot

L'arduino communique avec le robot avec le protocole de communication série expliquer précédemment. Pour établir cette communication les Pins 16 et 17 (Tx/Rx) de l'arduino

sont branchées aux pins Rx/Tx de l'interface de communication du robot située a sa base, on branche aussi le GND de l'Arduino au GND de l'interface de communication.

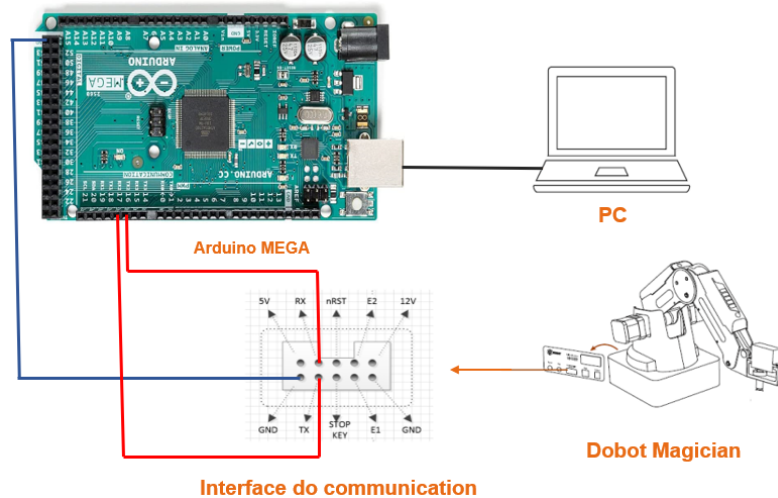


FIGURE 5.2 – Branchement entre l'arduino et Dobot

5.3 Scénario élaboré pour le test des bibliothèques

La partie suivante étant une partie de test, il s'avère intéressant de décrire un scénario pouvant être réaliser en séance de TP. Ce scénario doit inclure le maximum des fonctions de nos bibliothèques pour assurer leur validité dans un code de réalisation complexe.

Nous avons opter pour une modélisation réseau de Pétri vu qu'elle permet de représenter les différents étapes d'un processus en incluant tout les composants de la maquette (Capteurs, convoyeur et robot), de plus ce projet vise les travaux pratiques du module système à événement discret faisant partie du programme du M1 ISTR de l'année prochaine.

Le scénario choisi est le suivant :

- le système est en repos(bras position "HOME") initialement et les pièces sont présente sur la tapis 1.
- Le tapis 1 se met en marche déplaçant les pièces à trier.
- Le capteur infrarouge détecte la pièce, le tapis 1 s'arrête et le bras se déplace a la première position ALLER-T1 (au-dessus du tapis 1).
- La pince s'ouvre et le bras se déplace en position 1.
- La pince se ferme et le bras déplace la pièce à la position 2 (Capteur de couleur).

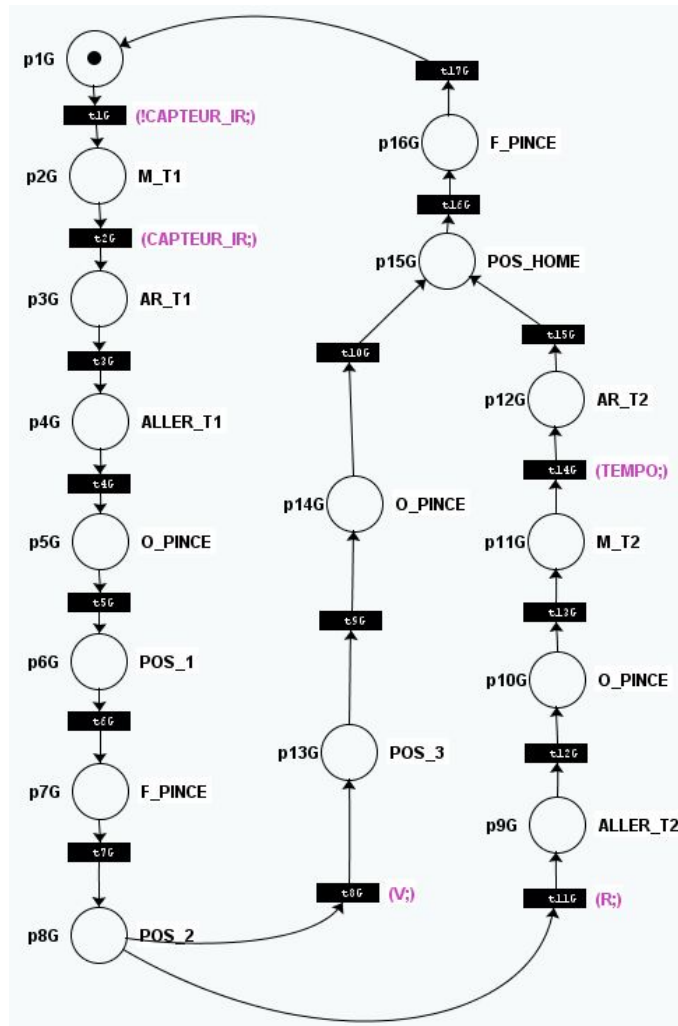
Après deux scénarios sont possibles, le premier est le suivant :

- la pièce prise est verte (retourner par le capteur de couleur).
- le bras se déplace en position 3 (au-dessus de la zone des pièces vertes).
- La pince s'ouvre.
- Le bras se déplace à sa position initiale.
- La pince se ferme.

Le deuxième scénario possible est le suivant :

- La pièce prise est item rouge (retourner par le capteur de couleur).
- Le bras se déplace vers la position-Tapis-2 (au dessus du deuxième tapis 2) et la pince s'ouvre.
- Le tapis 2 se met en marche et déplace la pièce rouge vers la zone de ces derniers.
- Le bras se déplace à sa position initiale.

La figure suivante représente le réseau de Pétri établi pour le test et décrivant le scénario :



- La boucle "while" permet de continuer à tester la présence de pièces sur le tapis roulant jusqu'à ce qu'une pièce soit détectée.
- Nous avons effectué un test de la présence de pièces sur le premier tapis roulant à l'aide du capteur infrarouge, en appelant la fonction "*Dobot_Piece()*".
- Après avoir détecté la pièce sur le premier tapis roulant, le moteur de celui-ci est stoppé en appelant la fonction "*Dobot_Tapis1(0)*", et le bras robotique retourne à sa position initiale au-dessus de ce même tapis en utilisant la fonction "*Dobot_AllerTapis1(1)*".
- Le bras robotique se dirige vers le premier tapis roulant en descendant avec la fonction "*Dobot_AllerPos1(1)*", et une fois arrivé, la pince s'active et attrape la pièce en utilisant la fonction "*Dobot_Pince(1)*".
- Le bras robotique se déplace ensuite vers le capteur de couleur pour déterminer la couleur de la pièce. La couleur est vérifiée avec la fonction "*Dobot_Getcolor()*", et le bras se déplace vers la position appropriée pour déposer la pièce en fonction de sa couleur.
- Si la pièce est verte, le bras se déplace vers la position du lot de pièces vertes avec la fonction "*Dobot_AllerPos3(1)*" et la pince s'ouvre pour déposer la pièce verte avec la fonction "*Dobot_Pince(0)*".
- Si la pièce est rouge, le bras se déplace vers le deuxième tapis roulant avec la fonction "*Dobot_AllerTapis2(1)*", puis la pince s'ouvre pour déposer la pièce rouge sur le tapis roulant avec la fonction "*Dobot_Pince(0)*". Le moteur du deuxième tapis roulant est activé avec la fonction "*Dobot_Tapis2(1)*" pour faire avancer la pièce rouge, puis désactivée avec la fonction "*Dobot_Tapis2(0)*" après un temps de déplacement défini.
- Le bras robotique revient à sa position initiale avec "*Dobot_AllerPosHOME(1)*", ensuite une fermeture de la pince avec *Dobot_pince(1)* et attente de la détection d'une nouvelle pièce à trier.

5.5 Ajustements proposés

L'exécution de notre scénario a révélé un défaut surprenant lié aux performances de notre robot. Nous avons constaté un dysfonctionnement et un blocage du processus et du robot lorsque les pièces détectées par le capteur du tapis 1 ne sont pas bien centrées sur le convoyeur.

Partant de l'hypothèse que tous les éléments de la maquette restent dans la même position, nous avons opté pour des déplacements du robot basés sur des coordonnées définies précédemment à l'aide du logiciel DobotStudio et sur une commande en boucle ouverte. Cette hypothèse est fondée sur la nature des fonctions que nos bibliothèques doivent contenir, où l'impossibilité d'insérer des coordonnées comme entrées est remarquable. En effet, ces fonctions doivent être booléennes en entrée et sortie.

Les dimensions du matériel utilisé imposent également ce problème (une pince qui s'ouvre de 2,7 cm et une pièce de 2,5 cm) car le robot se dirige directement vers la position déjà définie, censée être souvent au centre du tapis 1 et sur le capteur infrarouge. La moindre déviation de cette position ne permettra pas au robot de récupérer la pièce.

Pour résoudre ce problème, nous avons proposé une solution plutôt matérielle que logicielle, en ajustant le convoyeur en ajoutant deux pièces à l'extrémité des longueurs du convoyeur, qui seront imprimées à l'aide de l'imprimante 3D, afin de centrer les pièces avant d'atteindre le capteur infrarouge.

Cette solution est la plus appropriée à notre situation en termes de coût et de qualité, étant donné qu'il est difficile de résoudre le problème en modifiant les fonctions.

La figure suivante présente la pièce en 3D conçue à l'aide du logiciel SolidWorks [12] :

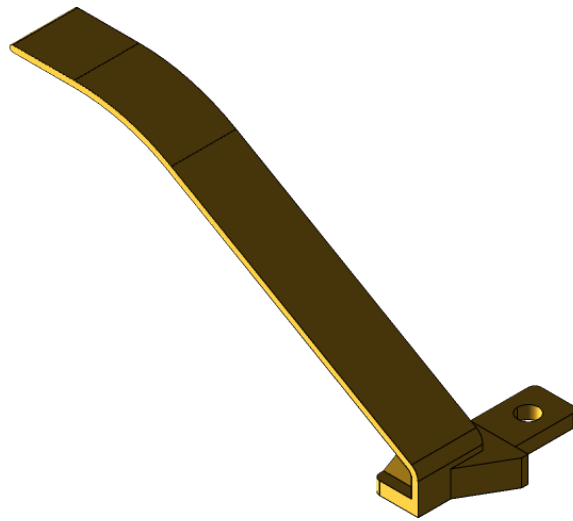


FIGURE 5.4 – Image de la pièce conçu pour centraliser les pièces en 3D.

Chapitre 6

Conclusion et perspectives

En conclusion, Le but de ce projet est de créer une bibliothèque pour contrôler un bras robotique et son environnement via une carte Arduino Mega, en utilisant des modèles standardisés de programmation pour simplifier les commandes. Cette bibliothèque permettra de contrôler de façon discrète une maquette du bras robotisé Dobot-Magician qui était jusqu'alors uniquement contrôlable via son logiciel associé.

En effet, grâce à ce projet, nous avons pu développer une bibliothèque efficace qui permet le pilotage avec succès du bras en se basant sur des commandes simples. Nous avons également réussi à intégrer cette bibliothèque avec un microcontrôleur Arduino Mega et à la tester dans des environnements réels. Les résultats obtenus ont clairement démontré que cette bibliothèque permet un contrôle plus ou moins précis et fiable du robot avec quelques problèmes résolus avec des ajustements matériels.

Néanmoins, même si la bibliothèque développée a montré des performances satisfaisantes, il reste des possibilités d'amélioration pour ce projet. Diverses perspectives pourraient être envisagées, telles que l'ajout de fonctionnalités supplémentaires pour une meilleure gestion des erreurs et des exceptions, l'intégration avec des outils de simulation pour une validation plus rapide et efficace des modèles, et l'extension de la bibliothèque pour permettre la commande de robots plus complexes.

En somme, ce projet a abouti à une bibliothèque de commande efficace pour un bras robot et son environnement, avec des possibilités d'amélioration et d'extension pour des applications ultérieures.

Bibliographie

- [1] ABRA. Sens-97 e18-d80nk capteur de proximité photoélectrique infrarouge. <https://abra-electronics.com/sensors/sensors-proximity-en/>.
- [2] Arduino. Arduino mega pinout. <https://www.arduino.cc/>.
- [3] Dobot. Dobot conveyor belt kit. www.dobot.nu/en/product/dobot-conveyor-belt-kit/.
- [4] Mohamed Fezari and Ali Al Dahoud. Integrated development environment “ide” for arduino. *WSN applications*, pages 1–12, 2018.
- [5] Nitrathor. Ide arduino. <https://www.nitrathor.fr/fiches/ide-arduino>.
- [6] M Pollak, J Tkac, and M Kocisko. Robot programming via arduino microcontroller and accessories in the educational process. In *ICERI2021 Proceedings*, pages 3911–3918. IATED, 2021.
- [7] ShenZhen Yuejiang. *Dobot Magician User Guide V 1.7.0*. Shenzhen Yuejiang Technology Co., Ltd, 3F, Building NO.3, Tongfuyu Industrial Town, Nanshan District, Shenzhen, China, 2019-01-09.
- [8] ShenZhen Yuejiang. *conveyor belt domo instructions*. Shenzhen Yuejiang Technology Co., Ltd, 3F, Building NO.3, Tongfuyu Industrial Town, Nanshan District, Shenzhen, China, 2019-03-26.
- [9] ShenZhen Yuejiang. *Conveyor Belt User Guide V 1*. Shenzhen Yuejiang Technology Co., Ltd, 3F, Building NO.3, Tongfuyu Industrial Town, Nanshan District, Shenzhen, China, 2019-03-26.
- [10] ShenZhen Yuejiang. *Dobot Magician Demo Description V 1.1*. Shenzhen Yuejiang Technology Co., Ltd, 3F, Building NO.3, Tongfuyu Industrial Town, Nanshan District, Shenzhen, China, 2020-04-01.
- [11] silanus. Les cellules photo électriques. https://silanus.fr/bts/activites/Maugenet/co/Les_cellules_photo-electriques.html.
- [12] Robin Z. Dobot conveyor belt parts guide. <https://grabcad.com/library/dobot-conveyor-belt-parts-guide-1>.
- [13] Iabbaden Zinedine and Lahlou Farid. *Réalisation d’un module de distribution d’énergie à base d’une carte Arduino méga 2560*. PhD thesis, Université Mouloud Mammeri, 2017.

Annexes

Code Arduino du scénario élaboré pour le test des bibliothèques (Vidéo du scénario élaboré : https://youtu.be/6_xvY-7Ddyo) :

```
#include <Dobot.h>
#include <FlexiTimer2.h>
#include <Magician.h>
#include <Message.h>
#include <Packet.h>
#include <Protocol.h>
#include <ProtocolDef.h>
#include <ProtocolID.h>
#include <RingBuffer.h>
#include <command.h>
#include <symbol.h>
#include <type.h>
#include <SoftwareSerial.h>
#include <Arduino.h>

void setup() {
    // Initialisation de la communication série avec le robot Dobot
    Dobot_Init();
    Dobot_SetHOMECmd();
}

void loop() {
    uint8_t resss;
    resss=Dobot_Piece(); //premier test de la présence d'une pièce sur le tapis 1 (capteur IR)

    while (resss==1) //boucle de test de la présence d'une pièce
    {
        resss=Dobot_Piece();
        Dobot_Tapis1(1);
    }
    Dobot_Tapis1(0); //Moteur du Tapis 1 mis a zéro
    Dobot_AllerTapis1(1); // Première position au dessus du tapis 1
    Dobot_Pince(0); //Ouvrir Pince
    delay(300);
    Dobot_AllerPos1(1); // le bras descend pour prendre la pièce.
    delay(300);
    Dobot_Pince(1); //Fermer Pince
```

```

delay(300);
Dobot_AllerPos2(1); // le bras se déplace vers le capteur couleur
delay(300);

int8_t color;
color= Dobot_Getcolor(); // Test de la couleur de la pièce
delay(300);

if (color == 1) // Si la pièce est verte
{
    Dobot_AllerPos3(1); // le bras se déplace vers la position du lot de pièces vertes
    delay(300);
    Dobot_Pince(0); //ouverture pince pour déposer la pièce verte
}
else // si la pièce est rouge
{
    Dobot_AllerTapis2(1); // le bras se déplace vers le tapis 2
    delay(300);
    Dobot_Pince(0); // ouverture de la pince pour déposer la pièce rouge sur le tapis 2
    delay(300);
    Dobot_Tapis2(1); //Actovation Moteur Tapis 2
    delay(3500);
    Dobot_Tapis2(0); // Désactivation Moteur Tapis 2
}
delay(300);
Dobot_AllerPosHOME(1); // le bras revient a sa position de repos.
Dobot_Pince(1); // fermeture de la Pince.
delay(300);
}

```

Code C++ de la bibliothèque développée :

```

#include "command.h"
#include "type.h"
#include "stdio.h"
#include "Dobot.h"
#include "FlexiTimer2.h"
#include "Protocol.h"
#include "HardwareSerial.h"
#include "arduino.h"

void Dobot_Init()
{
    SERIALNUM.begin(115200); //SERIALNUM==Serial2(pin 16(tx) et pin 17(rx))
    delay(1000);

    ProtocolInit();

    Serial.println("Dobot Init");
}

```



```

        for(int i = 0; i < 21; i++){
            Dobot_SetIOMultiplexing(i,IOFunctionDI);
        }
        SetQueuedCmdStartExec();
    }

// Fonction qui ramène le robot a une position initiale
void Dobot_SetHOMECmd(void)
{
    SetHomeCmd();
    WaitQueuedCmdFinished();
}

// Fonction qui determine le mode de mouvement ainsi que le point cible qu'on veut atteindre
void Dobot_SetPTPCmd(uint8_t Model,float x,float y,float z,float r)
{
    static PTPCmd ptpCmd;

    ptpCmd.ptpMode = Model; // Mode de mouvement
    ptpCmd.x = x; // coordonnée x de la position cible
    ptpCmd.y = y; // coordonnée Y de la position cible
    ptpCmd.z = z; // coordonnée Z de la position cible
    ptpCmd.rHead = r; // orientation de l'organe terminale

    SetPTPCmd(&ptpCmd);
    WaitQueuedCmdFinished();
}

void Dobot_SetIOMultiplexing(uint8_t address,uint8_t function) //Set IO en entrée ou sortie
{
    static IOConfig iOConfig;

    iOConfig.address = address; //Adresse de l'IO (1 a 20)
    iOConfig.function = (IOFunction)function; //Fonctionnement de l'IO (PWM,DI,AI..)

    SetIOMultiplexing(&iOConfig);
}

uint8_t Dobot_GetIODI(uint8_t address) //Donne le Résultat de l'entrée DI (Digital input)
{
    static EIODI eIODI;

    eIODI.address = address;

    GetIODI(&eIODI);

    return eIODI.value;
}

```

```

// Fonction pour ouvrir ou fermer la pince
void Dobot_Pince(bool isGriped)
{
    static EndEffectorGripper Pince;

    Pince.isEnabled = 1;
    Pince.isGriped = isGriped; // 1:fermer Pince 0:Ouvrir Pince

    SeEndEffectorGrippe(&Pince);
}

/*****/

void Dobot_AllerTapis1(bool state)
{
    if (state == 1)
    {
        Dobot_SetPTPCmd(JUMP_XYZ,X1,Y1,Z1,R1);
    }
}

void Dobot_AllerTapis2(bool state)
{
    if (state == 1)
    {
        Dobot_SetPTPCmd(JUMP_XYZ,X5,Y5,Z5,R5);
    }
}

void Dobot_AllerPos1(bool state)
{
    if (state == 1)
    {
        Dobot_SetPTPCmd(JUMP_XYZ,X2,Y2,Z2,R2);
    }
}

void Dobot_AllerPos2(bool state)
{
    if (state == 1)
    {
        Dobot_SetPTPCmd(JUMP_XYZ,X3,Y3,Z3,R3);
    }
}

void Dobot_AllerPos3(bool state)
{
    if (state == 1)
    {
        Dobot_SetPTPCmd(JUMP_XYZ,X4,Y4,Z4,R4);
    }
}

```

```

    }
}

void Dobot_AllerPosHOME(bool state)
{
    if (state == 1)
    {
        Dobot_SetPTPCmd(JUMP_XYZ,X6,Y6,Z6,R6);
    }
}

/*****

// Fonction pour controler (Activer/Désactiver) le Tapis_1
void Dobot_Tapis1(bool enable)
{
    static EMotor eMotor;

    eMotor.address = 0; // branché sur Stepper1
    eMotor.enable = enable;
    eMotor.speed = 4000;

    SetEMotor(&eMotor);
}

// Fonction pour controler (Activer/Désactiver) le Tapis_2

void Dobot_Tapis2(bool enable)
{
    static EMotor eMotor;

    eMotor.address = 1; // Branché sur Stepper2
    eMotor.enable = enable;
    eMotor.speed = 4000;

    SetEMotor(&eMotor);
}

*****/

uint8_t Dobot_Piece() //Fonction de detection d'une piece sur le Tapis
{
    uint8_t state;
    Dobot_SetIOMultiplexing(7,IOfunctionDI); //Set IO7(ADC-GP4) en entrée
    state=Dobot_GetIODI(7); //Résultat de l'entrée IO7
    return state;
}

*****/

uint8_t Dobot_Getcolor()
{
    uint8_t res1,res2;
    Dobot_SetIOMultiplexing(4,IOfunctionDI); //Set IO4(ADC-GP5) en entrée

```

```

Dobot_SetIOMultiplexing(5,IOfuctionDI); //Set IO5(ADC-GP5) en entrée
res1=Dobot_GetIODI(4);
res2=Dobot_GetIODI(5);
if(res1== 0 && res2== 0)
{
    return 0;
}
if(res1== 0 && res2== 1)
{
    return 1;
}
if(res1== 1 && res2== 0)
{
    return 2;
}
else
{
    return 3;
}
}

```