وزارة التعليم العالي والبحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
جامعة عبد الحميد بن باديس -مستغانم
Université Abdelhamid Ibn Badis Mostaganem
كلية العلوم والتكنولوجيا
Faculté des Sciences et de la Technologie
قسم الهندسة الكهربائية
Département de Génie Électrique

MEMOIRE

Pour obtenir le diplôme de

MASTER EN ELECTRONIQUE

Spécialité : Electronique des systèmes embarqués

Présenté par

Belkacimi Abdelkader

Hamani Abderrahmane

Contrôle à distance d'un bras robot à 4 DDL à l'aide d'un ESP32

Soutenu le 29/10/2023 devant le jury composé de :

Président :	M^{r} HADJ BEKLAOUZ Larbi	MCA	Université de Mostaganem
Examinateur :	M^{me} BENCHELLAL Amel	MCB	Université de Mostaganem
Rapporteur :	Mr DJELTI Benbella	MCB	Université de Mostaganem

Année Universitaire : 2022/2023

Remerciement

Louange à allah tout puissant qui nous a guidées pour l'accomplissement de ce travail.

Au terme de notre cycle d'études, il nous parait opportun de nous acquitter d'un devoir noble, celui de remercier tous ceux qui ont contribué par leur assistance tant morale que physique à notre cursus universitaire et à la réalisation de ce mémoire.

Nos remercîments s'adressent à notre encadreur Dr djelti benbella pour sa disponibilité et son aide jusqu'à la finalisation de ce travail. Qu'il daigne de recevoir nos sentiments de gratitude et de respect profond

Nous remercions les membres de jury d'avoir bien voulu étudier ce travail et participer la commission du jury. Qu'ils en soient louablement gratifiés.

Ainsi que tous nos remerciements à nos amis

Adnan Ben Zaza

Nous remercions tous les enseignants de la Faculté des Sciences et de la Technologie de l'université de Mostaganem pour la qualité de l'enseignement.

Nous remercions toutes les personnes qui nous ont aidées de près ou d loin pour la réalisation de notre travail

Résumé

Cette mémoire présente une méthode pour programmer un bras robot à 4 degrés de liberté (DDL) avec un ESP32. Le bras robot est composé de quatre servomoteurs, chacun responsable d'un degré de liberté. Le ESP32 est utilisé pour contrôler les servomoteurs et pour communiquer avec un ordinateur.

La méthode proposée comprend les étapes suivantes :

Configuration du matériel. Le bras robot est monté et les servomoteurs sont connectés à l'ESP32.

Programmation du ESP32. Une bibliothèque est écrite pour contrôler les servomoteurs.

Communication avec l'ordinateur. Une interface graphique est créée pour permettre à l'utilisateur de contrôler le bras robot depuis un ordinateur.

Les résultats de la méthode proposée sont présentés. Le bras robot est capable de se déplacer dans un espace 3D. L'interface graphique permet à l'utilisateur de contrôler le bras robot de manière intuitive.

La méthode proposée est une solution efficace pour programmer un bras robot à 4 DDL avec un ESP32. Elle est simple à mettre en œuvre et permet de contrôler le bras robot de manière intuitive.

Mots-clés

Bras robot, ESP32, servomoteurs, programmation, interface graphique

ملخص

تقدم هذه الذاكرة طريقة لبرمجة ذراع روبوت بأربع درجات حرية (DDL) باستخدام .ESP32 يتكون الذراع الروبوتي من أربع محركات خدمة، حيث يتحمل كل محرك مسؤولية درجة حرية واحدة. يُستخدم ESP32 للتحكم في محركات الخدمة والتواصل مع الكمبيوتر .الطريقة المقترحة تشمل الخطوات التالية:

1. تكوين الأجهزة. يتم تركيب الذراع الروبوتية وتوصيل محركات الخدمة بـ.ESP32
2. برمجة .ESP32 يتم كتابة مكتبة للتحكم في محركات الخدمة.
3. التواصل مع الكمبيوتر. يتم إنشاء واجهة رسومية للسماح للمستخدم بالتحكم في الذراع الروبوتي من الكمبيوتر .تُقدم الذاكرة نتائج الطريقة المقترحة. يمكن للذراع الروبوتي التنقل في مساحة ثلاثية الأبعاد، وتسمح الواجهة الرسومية للمستخدم بالتحكم في الذراع الروبوتي بشكل بديهي .الطريقة المقترحة هي حلاً فعاً لا لبرمجة ذراع روبوت بأربع درجات حرية باستخدام .ESP32 إنها سهلة التنفيذ وتسمح بالتحكم في الذراع الروبوتي بشكل بديهي.

كلمات مفتاحية

ذراع روبوت، ESP32، محركات الخدمة، برمجة، واجهة رسومية

Summary

This thesis presents a method for programming a 4-degree-of-freedom (DOF) robot arm using an ESP32. The robot arm consists of four servo motors, with each responsible for one degree of freedom. The ESP32 is used to control the servo motors and communicate with a computer.

The proposed method includes the following steps:

1. **Hardware Configuration**: The robot arm is assembled, and the servo motors are connected to the ESP32.
2. **ESP32 Programming**: A library is written to control the servo motors.
3. **Communication with the Computer**: A graphical interface is created to allow the user to control the robot arm from a computer.

The results of the proposed method are presented. The robot arm is capable of moving in a 3D space, and the graphical interface enables intuitive control by the user.

The proposed method is an efficient solution for programming a 4-DOF robot arm with an ESP32. It is easy to implement and allows for intuitive control of the robot arm.

Keywords

Robot arm, ESP32, servo motors, programming, graphical interface

Table des matières

Liste des figures

Liste des tableaux

Listes des acronymes et abréviations

RUR Rossums Universal Robots.

PPP : 3 articulations prismatiques.

DDL : dégrée de liberté.

RPP : articulation rotoïde avec 2 articulation prismatique.

RRP : 2 articulations rotoide avec articulation prismatique.

RRR : Trois articulations rotoïdes.

SCARA : Sélective Compliance Assembly Robot Arm.

PWM : Pulse width modulation.

Introduction générale

La robotique, depuis ses débuts, a été un domaine d'innovation incessante, transformant notre façon d'interagir avec le monde qui nous entoure. À mesure que les technologies numériques ont évolué, l'intégration de microcontrôleurs polyvalents tels que l'ESP32 a révolutionné la manière dont nous pouvons concevoir, programmer et interagir avec des robots. Cette convergence de la robotique et de l'informatique a ouvert la voie à de nouvelles possibilités dans des domaines aussi divers que l'automatisation industrielle, la santé, l'éducation et les loisirs.

Le présent projet s'inscrit dans cette dynamique d'innovation en explorant un domaine particulièrement fascinant de la robotique moderne : la commande à distance, contrôlé de manière intuitive à travers une interface HTML. Le parrainage robotique, aussi connu sous le nom de télé opération, est une méthode qui permet à un opérateur humain de prendre le contrôle d'un robot à distance, en lui transmettant des instructions en temps réel. Cette approche trouve des applications dans des domaines aussi variés que l'exploration spatiale, la recherche en environnements hostiles, ou encore la télémédecine.

L'ESP32, en tant que plateforme de développement polyvalente, offre une base solide pour la création d'une interface de contrôle interactive via une page HTML. Cette combinaison offre un potentiel exceptionnel pour développer des systèmes de télé-opérations accessibles et adaptables à une multitude de contextes.

Au fil des chapitres qui suivent, nous explorerons en détail la conception et la mise en œuvre de notre système de parrainage robotique basé sur l'ESP32 et l'interface HTML. Nous aborderons les défis techniques rencontrés, les choix de conception, ainsi que les performances du système. De plus, nous discuterons des implications et des applications potentielles de cette technologie dans des domaines spécifiques, en mettant en évidence ses avantages et ses limites.

Ce projet est le fruit d'efforts de recherche bibliographique, d'expérimentations pratiques et d'une vision d'avenir pour l'intégration harmonieuse de la robotique et de l'informatique. Nous espérons que ce travail contribuera à ouvrir de nouvelles perspectives dans le domaine de la télé-opérations et qu'il inspirera des projets futurs visant à exploiter pleinement le potentiel de l'ESP32 pour la commande des robots à distance.

Chapitre I
Généralités sur les robots

I.1 Introduction

La robotique est un vaste sujet qui peut inclure des dispositifs mécaniques parfois ressemblant à des humains, ou des entités logicielles. En général, ces /robots/ peuvent configurés pour exécuter une ou plusieurs tâches spécifiques.

Un robot est un automate doté de capteurs et d'effecteurs lui donnant une capacité d'adaptation et de déplacement proche de l'autonomie Un robot est un agent physique réalisant des tâches dans l'environnement dans lequel il ´évolue.

Par robot on entend parler un système mécanique équipé de capteurs, d'actionneurs et d'un système de commande permettant d'agir sur les actionneurs en fonction d'une part, de la tâche à accomplir, et d'autre part, des informations données par les capteurs.

Un robot est un système mécanique articulé et actionné, contrôlé par un ordinateur, Les robots manipulateurs sont l'un des systèmes mécatroniques les plus utilisés dans l'industrie, dont les applications comprennent l'assemblage d'éléments, ainsi que le soudage et la peinture des pièces. En raison de sa grande utilité dans l'industrie, il est très important d'étudier sa cinématique, sa dynamique ainsi que son contrôle automatique[1].

Une troisième orientation est apparue après la robotique industrielle et la robotique d'intervention en raison des progrès en miniaturisation, en microélectronique et en micromécanique, combinés avec les nouvelles capacités des systèmes de traitement de l'information et de communication. Ceci a créé les conditions technologiques favorables au développement de robots mobiles autonomes ou semi-autonomes grâce aux capacités d'apprentissage et d'intelligence d'entités artificielles, matérielles ainsi que leur introduction dans des environnements en forte interaction avec l'homme pour réaliser des applications de service professionnel (agriculture, médical, nettoyage...) et de service personnel (jeux, tâches domestiques...)[2].

I.2 Historique

L'histoire de la robotique est considérée comme l'un des événements les plus importants dont le monde et la technologie ont été témoins, de sorte qu'elle contribue grandement à la vie humaine, qui a connu un grand développement à travers différentes époques, et ce sont les dates et les découvertes les plus importantes pour le monde de robotique :

➢ Avant 1950

- 1921: **Karel Capek**, RUR (Rossum's Universal Robots).
- 1940 : **Isaac Asimov** ´écrit un ensemble de nouvelles sur les robots.
- Enoncé des « Trois lois de la robotique ».

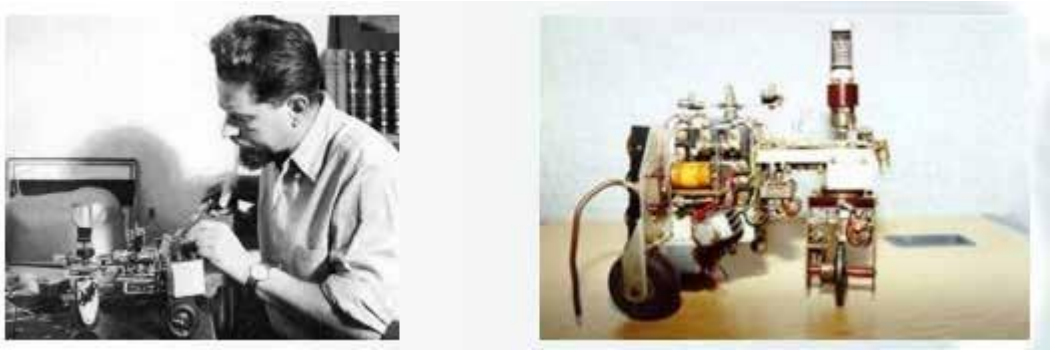


Figure I. 1 Robotique avant 1950

➢ 1950-1960

• 1948 : Grey Walter invente le premier robot mobile autonome, une tortue se dirigeant vers les sources de lumière qu'elle perçoit. Cependant, ce robot n'est pas programmable.

• La mise en place de robots n'a été possible que dans les années 50 avec la création des transistors et circuits intégrés.

➢ 1950-1970 : Robotique industrielle

• 1961 : Premier robot industriel mis en place dans une usine de General Motors : UNIMATE (tubes cathodiques de t´el 'vision).

• 1972 : Nissan ouvre la première chaine de production complétement robotisée.

• 1978 : PUMA (Programmable Universel Machine for Assembla) d´enveloppé par General Motors (toujours utilisé)

Figure I. 2 Robotique industrielle

➢ 1960-1980 : Premiers robots mobiles

• 1960-64 : Ouverture des laboratoires d'Intelligence Artificielle au M.I.T., Stanford Research Institute (SRI), Stanford University, University of Edinburgh.

• Fin des années 60 : Mise en place de “Shakey” premier robot mobile intégrant perception, planification et exécution.

• 1970 : Standford Cart.

• 1977 : premier robot mobile français HILARE au LAAS (CNRS Toulouse) .

Figure I. 3 Premiers robot mobiles

➢ Depuis 2000 : démocratisations des robots

• 2000 : Lancement d’Asimo

• Diversification des compétitions de robotique.

• Utilisation de drones en situation réelle (Irak...).

• 2006 : le projet Aibo n’est plus assez rentable, fin de la production.

• 2009 : robot Nao utilisé à la RobocupSoccer .

Figure I. 4 Démocratisation des robots depuis 2000

I.3 Définitions

Plusieurs définitions ont été apportées au mot robot depuis son apparition et parmi elles, nous avons retenu :

- "Dans les œuvres de science-fiction : machine à l'aspect humain capable, d'exécuter des opérations et de parler"[3]
- "Appareil automatique capable de manipuler des objets ou d'exécuter des opérations selon un programme fixe, modifiable ou adaptable ".
- "Un système capable d'accomplir des tâches, en tout ou en partie, habituellement dévouées aux humains." [4]

I.3.1 Définition de la robotique

La robotique est la science qui étudie les systèmes électromécaniques actionnés et contrôlés par le biais d'un ensemble de logiciels leur conférant une intelligence dite artificielle ; aussi l'ensemble des techniques permettant la conception et la réalisation de machines automatiques ou de robots[5].

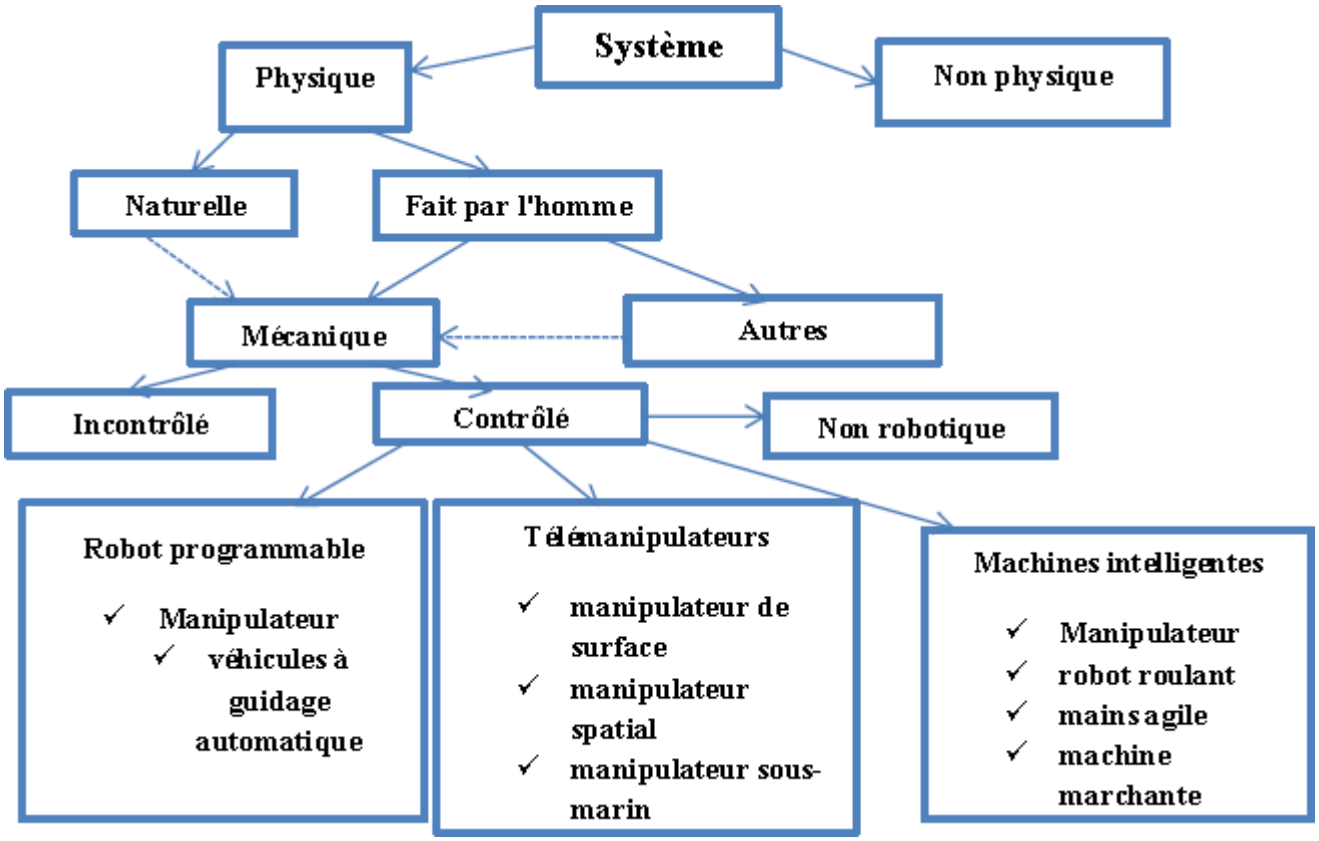


Figure I. 5 Généalogie des systèmes mécanique robotique

I.4 Composants des robots

Un robot, en tant que système, il est composé d'éléments, qui sont intégrés sous forme d'un ensemble. La plus par des robots contient les éléments suivants :

I.4.1 Bras manipulateur

C'est un système de positionnement robotisé où les forces agissant au niveau désarticulations sont produites par des actionneurs. Ces derniers peuvent être de type électrique hydraulique, ou pneumatique.

I.4.2 Mécanismes

Système de corps articules disposant d'une haute mobilité. C'est le corps principal du robot qui comprend les segments, les jonctions, les articulations et d'autres éléments de structure du robot, le mécanisme cinétique de tout robot font partie des éléments de base qui incarnent la forme et la structure du robot, qui combinent toutes ses articulations cinétiques externes et aident le robot à accomplir ses tâches naturellement.

I.4.3 Effecteur

Cette partie représentée sur la figure suivante, est reliée à la dernière jonction (main), d'un manipulateur qui gère généralement les objets, établit des connexions à d'autres machines ou effectue les tâches requises.

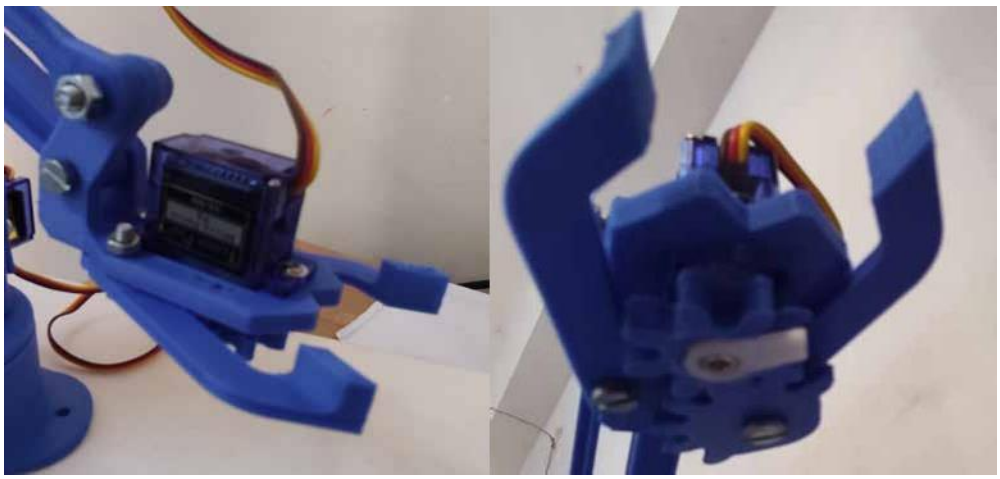


figure I. 6 Effecteur d'un manipulateur.

I.4.4 Actionneurs

Les actionneurs sont les « muscles » de manipulateurs. Le contrôleur envoie des signaux aux actionneurs, qui, à leur tour, déplacent les articulations du robot et des jonctions. Les actionneurs les plus utilisés en Robotique sont les servomoteurs, les

moteurs pas à pas, les actionneurs pneumatiques et les vérins hydrauliques. La figure illustre les différents types d'actionneurs qui sont sous le contrôle du contrôleur[6].

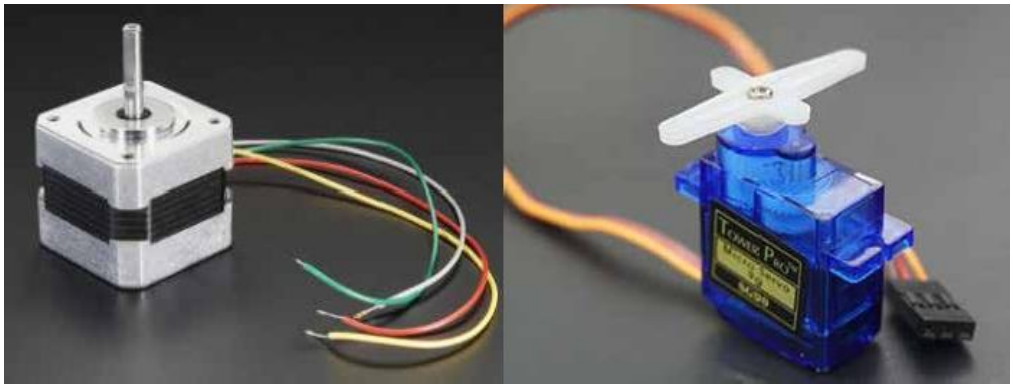


Figure I. 7 Différents actionneurs d'un robot

I.4.5 Système de contrôle

C'est un système qui permet au robot d'analyser les données provenant des capteurs et d'envoyer les ordres relatifs aux actionneurs.

I.5 Catégories des robots

Il existe deux grandes familles de robots à savoir les robots manipulateurs et les robots mobiles :

I.5.1 Les robots manipulateurs

Un robot manipulateur également appelé bras robotique, est un type de robot industriel conçu pour effectuer des tâches de manipulation et de déplacement d'objets dans un environnement de production ou de fabrication.

Les robots manipulateurs sont souvent utilisés dans les usines de production pour manipuler des pièces, assembler des produits, souder des composants, emballer des produits, etc. Ils sont programmables et peuvent être configurés pour effectuer une variété de taches en fonction des besoins de production.

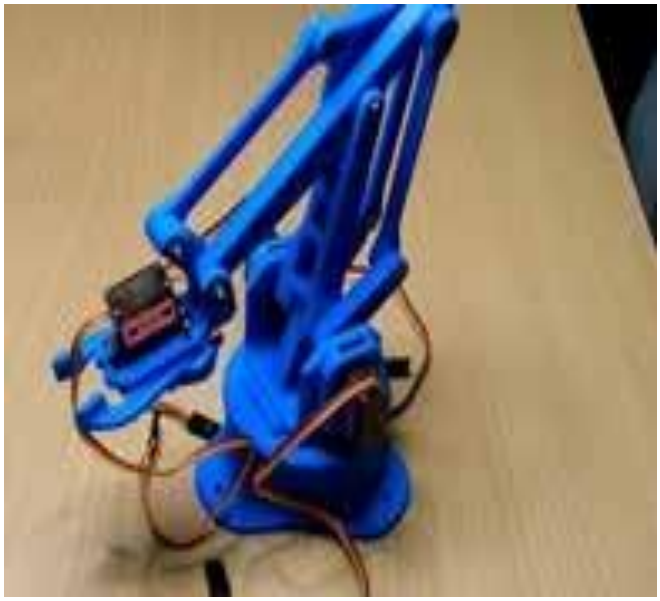


Figure I. 8 Bras robot manipulateur à 4 dégrée liberté

I.5.2 Les robots mobiles

Les robots mobiles sont des machines autonomes capables d'effectuer des tâches sans intervention humaine directe. Ils sont équipés de capteurs et d'actuateurs qui leur permettent de se déplacer dans leur environnement, de percevoir les objets et les obstacles, et d'interagir avec leur environnement en effectuant des actions programmées.

Figure I. 9 Le robot mobile avec bras robotique.

I.6 Constitution du bras manipulateur

Un bras manipulateur est composé de quatre parties principales :

- **Structure mécanique.**
- **Les actionneurs.**
- **La partie commande.**

I.6.1 Structure mécanique :

La constitution du bras manipulateur dépend du type de robot manipulateur. Cependant, la plupart des bras manipulateurs sont constitués de plusieurs éléments de base qui leur permettent de se déplacer et d'effectuer des taches, Une structure mécanique qui sera le squelette du robot, peut être divisée en trois parties distinctes :

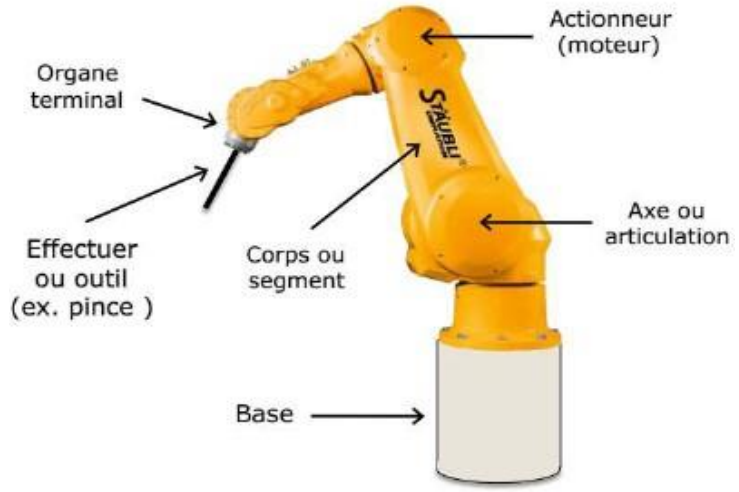


Figure I. 10 La structure de bras manipulateur

- ✓ **La base**

Pour qu'un manipulateur exécute des tâches dans un lieu donné, il faut d'abord l'amener sur place. Pour les robots à poste fixe, dépourvue de véhicule, on parle alors de base du manipulateur qui est fixée sur le lieu de travail. Ceci est le cas de la quasi-totalité des robots industriels[7].

- ✓ **Élément porteur**

Il est composé d'un ensemble de corps souples ou rigides liés par des articulations, servants à déplacer l'organe terminal d'une configuration à une autre. Le porteur représente l'essentiel du système mécanique articulé (segment, articulation, actionneur, l'organe terminal), il a pour rôle d'amener l'organe terminal dans une situation donnée imposée par la tâche. Il est constitué de : segment, articulation, actionneur, l'organe terminal[8].

- ✓ **Organe terminal**

On le qualifie par tout dispositif destiné à manipuler des objets ou à les transformer (dispositifs de serrage, dispositifs magnétiques, torche de soudage, pistolet de peinture…etc.). En d'autres termes, il s'agit d'une interface permettant au robot

d'interagir avec son environnement. L'organe terminal peut être équipé de plusieurs dispositifs ayant des fonctionnalités différentes[9].

I.6.2 Les actionneurs

Les actionneurs d'un robot manipulateur servent à animer le système mécanique articulé, ils sont généralement fixés au niveau des articulations. Les actionneurs sont souvent des moteurs électriques à aimant permanent, à courant continu…etc. Combiné avec des transmissions. Pour les petits robots, la plupart du temps en utilise des servomoteurs ou des moteurs pas à pas. En ce qui concerne les robots manipulant de grandes charges, en remplace les moteurs par des actionneurs hydrauliques ou pneumatique selon le besoin[10].

I.6.3 La partie commande

La partie commande c'est l'unité opérationnelle qui reçoit les instructions décrivant la tâche à accomplir, les mesures relatives à l'état interne de la structure mécanique qui constitue le bras manipulateur et les observations concernant son environnement. Elle élabore en conséquence les commandes de ses différentes articulations en vue de l'exécution de ses tâches. Les systèmes actuels fonctionnent en interaction permanente selon le cycle information-décision-action.

I.7 Classification des robots manipulateurs

Les robots manipulateurs peuvent être classés en fonction de différents critères tels que leur configuration, leur capacité de charge, leur commande etc. voici quelques-unes des classifications les plus courantes :

I.7.1 Configuration et structure des robots

I.7.1.1 Structure cartésienne (PPP)

Les axes des robots cartésiens se déplacent dans des directions linéaires le long des axes X, Y et Z, ce qui leur permet de se déplacer dans un espace rectangulaire. Cette structure est relativement peu utilisée, sauf dans quelques applications particulières : robots pratiques, robots de magasinage, par exemple.

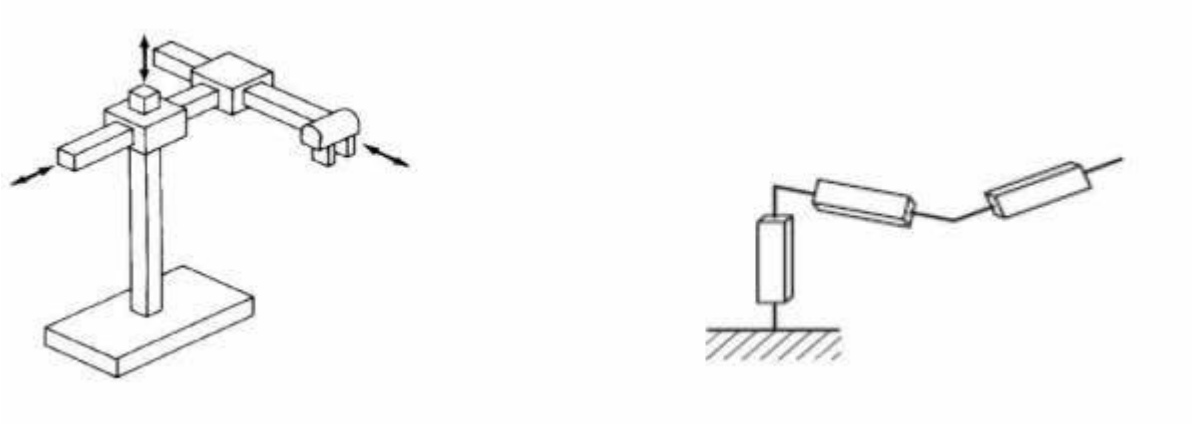


Figure I. 11 Structure d'un robot cartésien

I.7.1.2 Structure cylindriques (RPP) ou (PRP)

Les robots cylindriques ont un axes rotatif vertical et un bras horizontal qui peut se déplacer le long de cet axe. Elle présente l'inconvénient d'offrir un volume de travail faible devant un encombrement total important. Elle n'est pratiquement plus utilisée.

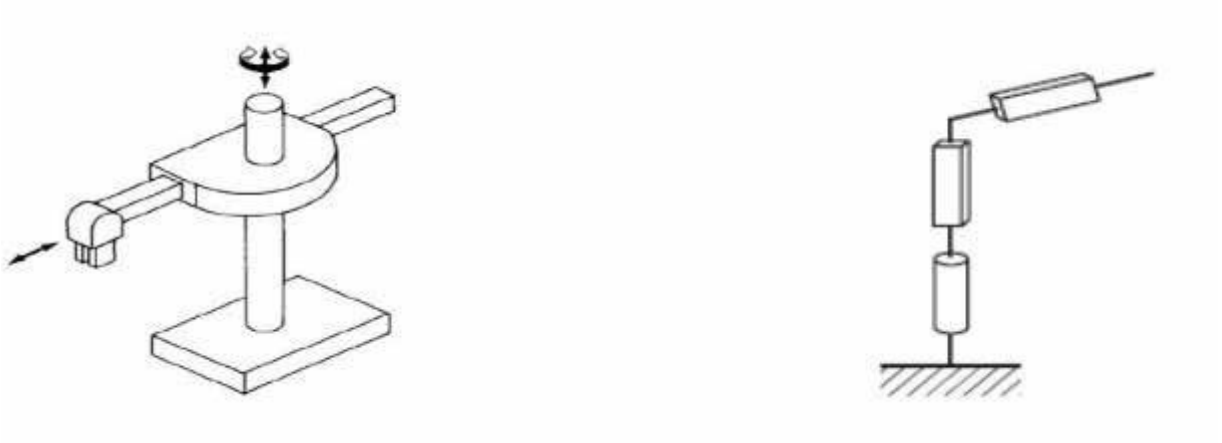


Figure I. 12 Structure d'un robot cylindrique

I.7.1.3 Structure sphériques

Les robots sphériques, également appelés robots polaires, ont un bras rotatif qui peut se déplacer dans toutes les directions à partir d'un point fixe.

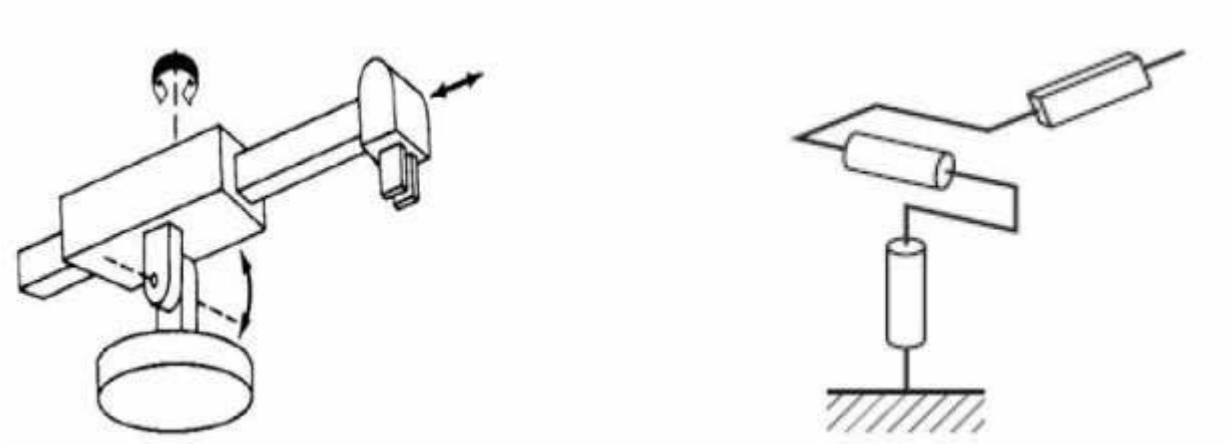


Figure I. 13 Structure d'un robot sphérique

I.7.1.4 Structure dite SCARA

Les robots SCARA ont un bras en forme de L qui peut se déplacer dans un plan horizontal, avec une rotation sur un axe vertical. Ce succès commercial est lié au fait que le ratio entre le volume de travail et l'encombrement est très favorable et aussi que la structure SCARA est très adaptée à ce type de taches.

Figure I. 14 Structure d'un robot SCARA

I.7.1.5 Structure 3R (anthropomorphiques)

Les robots anthropomorphiques ont une structure similaire à celle d'un bras humain, avec une épaule, un coude et un poignet, pouvant se programmer facilement pour différent types de taches et disposant d'un volume de travail conséquent.

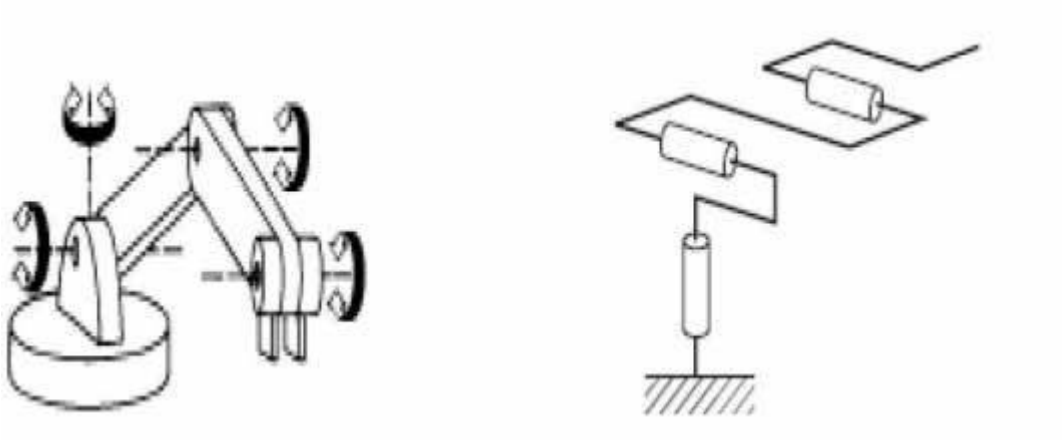


Figure I. 15 Structure d'un robot 3R

I.7.2 Capacité de charge des robots

Les robots légers ont une capacité de charge limitée, généralement inférieure à 10 Kg, les robots moyens ont une capacité de charge comprise entre 10et100Kg et les robots lourds ont une capacité de charge supérieure à100Kg.

II.7.3 Les Méthodes de commande des robots manipulateurs

I.7.3.1 Les robots manipulateurs à commande manuelle

Ils sont commandés à distance et « en temps réel » par un humain. Cette télécommande se fait plus ou moins longue distance par signaux mécaniques, hydraulique, ou le plus souvent électriques, par exemple la figure suivante :

Figure I. 16 manipulateur à commande manuelle

I.7.3.2 Les robots manipulateurs automatiques

Ils sont commandés à l'aide de logiques à relais ou pneumatiques (séquences fixes), ou par Automates programmables et cartes à microprocesseurs (séquences variables), généralement modulaires, Ces appareils sont conçus pour une application déterminée.

Figure I. 17 manipulateur automatique

I.7.3.3 Les robots manipulateurs programmables

Ils sont pilotés par des ordinateurs ou des armoires de commande numérique leurs mouvements continus dans l'espace sont alors programmés par apprentissage ou en langage symbolique par l'intermédiaire d'un clavier. Ils assurent des manipulations complexes, des opérations de soudage, usinage, découpe, peinture et pulvérisation, etc.

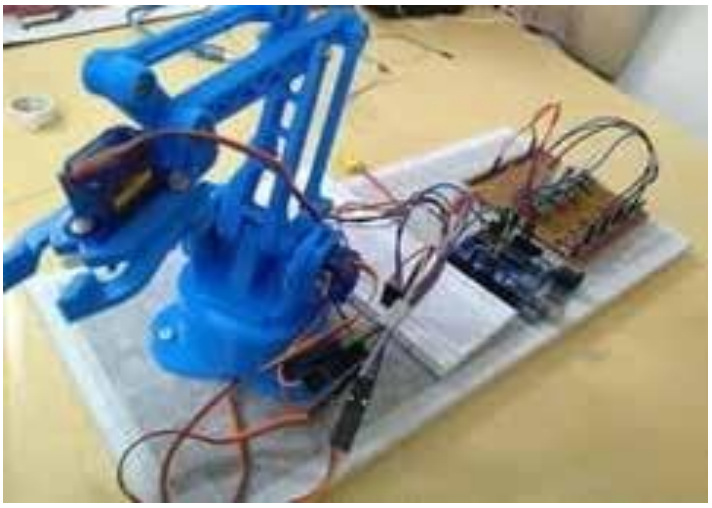


Figure I. 18 manipulateur programmable

I.7.3.4 Les Robots intelligents

On trouve actuellement des robots qui sont capables d'acquérir et d'utiliser certaines informations sur leur environnement (systèmes de vision, détecteurs de proximité, capteurs d'efforts...) comme le montre la figure suivante, les robots de troisième génération sont capables de comprendre un langage oral proche du langage naturel et de se débrouiller de façon autonome dans un environnement complexe grâce à l'utilisation de l'intelligence artificielle. [9]

Figure I. 19 Robot intelligent

I.8 Morphologie

La structure mécanique d'un robot manipulateur est composée de plusieurs corps connectés les uns aux autres par des liaisons appelées articulations ou joints, à un seul d.d.l de translation ou de rotation. Cette structure mécanique peut constituer une chaîne cinématique continue ouverte simple, une chaîne arborescente ou une chaîne complexe[11].

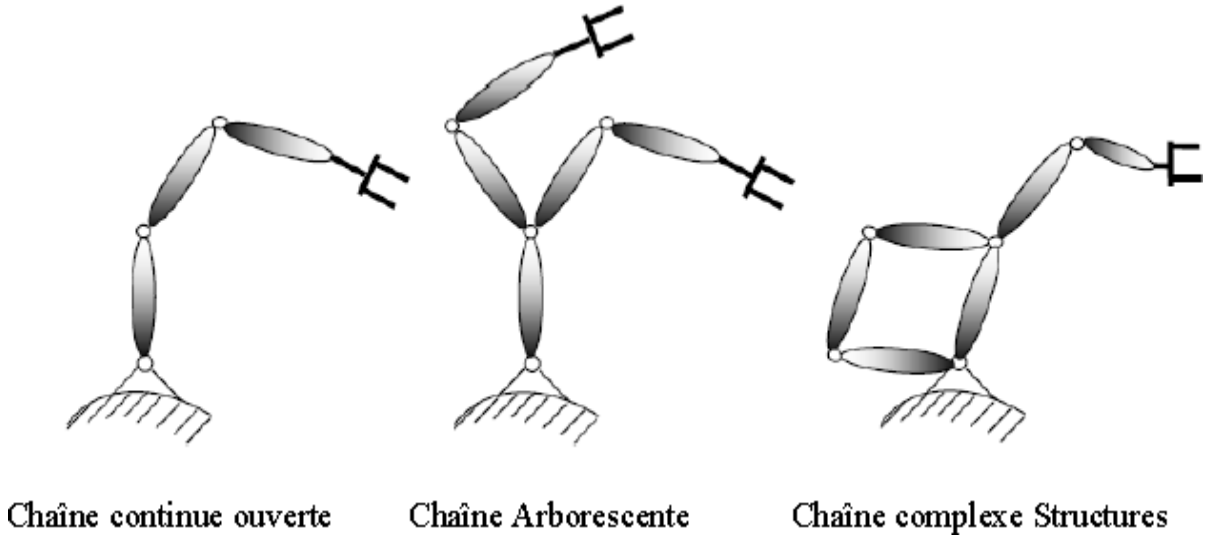


Figure I. 20 La structure utilisée dans la conception des robots

I.9 Articulation

Une articulation lie deux corps successifs en limitant le nombre de degré de liberté de l'un par rapport à l'autre. Soit m le nombre de degré de liberté résultant, encore appelé mobilité de l'articulation. La mobilité d'une articulation est telle que : $0 \ m \ 6$.

Lorsque m = 1 ; ce qui est fréquemment le cas en robotique, l'articulation est dite simple : soit rotoïdes, soit prismatique[12].

I.9.1 Articulation rotoïde

Il s'agit d'une articulation de type pivot, notée R, réduisant le mouvement entre deux corps à une rotation autour d'un axe qui leur est commun. La situation relative entre les deux corps est donnée par l'angle autour de cet axe.

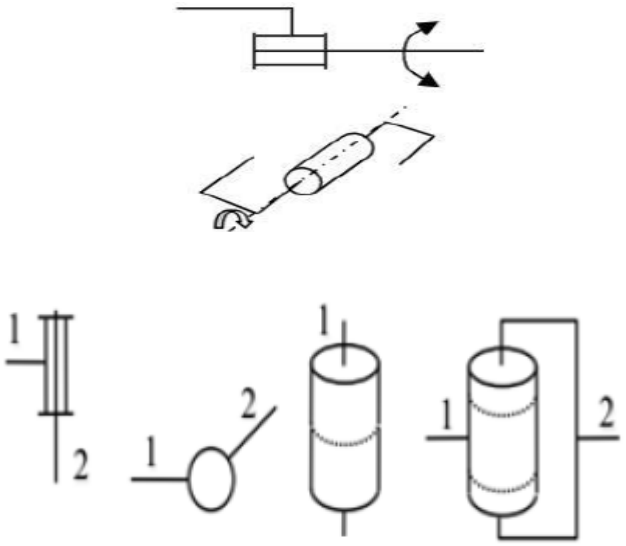


Figure I. 21 Articulation rotoïde.

I.9.2 Articulation prismatique

Il s'agit d'une articulation de type glissière, notée P, réduisant le mouvement entre deux corps à une translation le long d'un axe commun. La situation relative entre les deux corps est mesurée par la distance le long de cet axe.

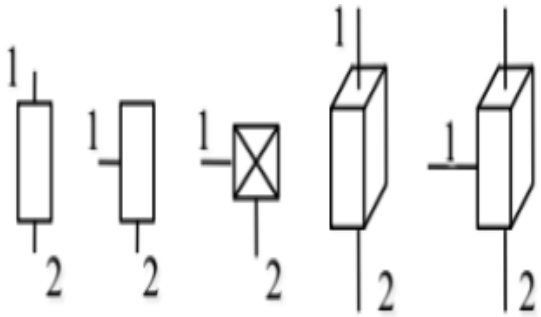


Figure I. 22 Articulation prismatique

I.10 Domaines d'application des robots manipulateurs

La robotique permet de réaliser des tâches répétitives, pénible, dangereuse et dans des milieux hostiles, avec précision et fiabilité, dans tout environnement, à des cadences élevées et continues, tout en palliant au manque de main-d'œuvre, favorisant la haute productivité, et donnant ainsi à la robotique un fort intérêt humain, technique et économique. Ce qui lui a permis de se diversifier dans de nombreux domaines d'utilisation qui sont les suivants[13].

Domaines industriels

- O La transformation de pièces : moulage, usinage, perçage, rectification, enlèvement de matière.
- O La préparation de surface : peinture, enduction, projection thermique/plasma, Grenaillage.
- O L'assemblage : rivetage, vissage, soudage, insertion, collage.
- O Les manutentions : manipulation de pièces, chargement/déchargement de machine, positionnement.
- O La distribution.
- O La palettisation et le conditionnement.
- O L'inspection et le test.
- O Le nettoyage.

Domaine médical

- O Opérations.
- O Diagnostique.
- O Radiothérapie.
- O Prothèses.

Domaine militaire

- O Désamorçage.
- O Déminage.

I.11 Avantages & Inconvénients des robots

I.11.1 Avantages

- ❖ Augmentation des taux de production.
- ❖ Amélioration de la qualité du produit.
- ❖ Réduction du gaspillage de matériel.
- ❖ Réduction des accidents.
- ❖ Faible taux d'emploi (réduite de la main-d'œuvre).
- ❖ Plus grande flexibilité et programmabilité (d'évolutivité).

I. 11.2 Inconvénients

- Remplacement du travail humain.
- Les robots représentent un coût initial coûteux.
- Ne peut faire que ce qui lui est demandé ; ni plus ni moins.
- Plus grand chômage.

I.12 Conclusion

Dans ce chapitre, nous avons donné un aperçu général sur la robotique : l'historique des robots, leurs structures, leurs utilisations, leurs différents types, leurs classifications ainsi que leurs domaines d'application, leurs avantages et leurs inconvenants, ce qui va nous servir pour la construction de notre bras.

Chapitre II

Partie matérielle et Logicielle du projet

II.1 Introduction

Depuis longtemps, la création des circuits électroniques à partir des composants comme (résistance, capacité, inductance, transistor…) a été difficile, et sa modification est vraiment complexée parce que ces circuits sont fabriqués pour faire un travail spécifique et pour faire un petit changement il faut passer par plusieurs opérations comme (des calculs, refaire les schémas, soudage, …).

Mais après le développement de la technologie électronique, la création des systèmes à base d'un circuit électronique est devenue très simple et facile grâce aux cartes de développements et des modules programmables, qui permettent de réaliser plusieurs fonctions dans le même système[14].

Dans ce chapitre, nous allons parler spécialement de la carte ESP32 WROOM mentionnant ses caractéristiques et nous donnerons le schéma de principe de cette carte avec les composants que nous utiliserons et l'installation de la librairie ESP32 dans ce système.

II.2 Hardware :

II.2.1 Définition de carte esp 32-wroom

La carte ESP-WROOM-32 est l'une des meilleures cartes de prototypage de la famille NODEMCU8266 à cause de sa petite taille, son faible coût et sa faible consommation d'énergie, elle est dotée des modules de communication sans fil WI-FI et Bluetooth. Elle permet d'implémenter une grande variété d'applications comme les réseaux de capteurs à faible consommation et l'encodage vocal[15]

II.2.2 Les Caractéristiques techniques [16]

Composant	Caractéristique
Microcontrôleur	ESP32
Tension de fonctionnement	3-5V
Pins I/O digitales	18
Pinces d'E/S numériques PWM	16
Pins output Analogique	2
Courant DC pour la broche 3.3V	50 mA
SRAM	512KB
Mémoire Flash	16 Mb
Fréquence de l'horloge	Jusqu'à 240 MHz
Wi-Fi	802.11 b//n

Tableau I 1 Les Caractéristiques techniques de la carte ESP-WROOM-32.

II.2.3 Brochage de la carte ESP32

La carte ESP32 contient 30 broches externes, la figure 2 montre le brochage de la carte,

Figure II. 1 Brochage de la carte d'ESP32

II.2.4 Les actionneurs (servomoteurs)

Un servomoteur est tout simplement un moteur à courant continu, doté d'un capteur de position (typiquement un potentiomètre) et d'une électronique d'asservissement de position. Ils sont pilotés par un fil de commande et alimentés par deux autres fils, le premier est relié à l'alimentation positive **+5 ou +6 V** selon le servo, le deuxième est relié à la masse **(GND).**

Le signal de commande quant à lui est de type modulation de largeur d'impulsion (**PWM).** En modifiant le rapport cyclique de ce signal, on indique au moteur quelle est la position

Figure II. 2 servomoteur

désirée dans une plage de positions possibles, généralement **[0, 180°].** On a utilisé :

Habituellement, ces 3 fils sont rassemblés dans une prise au format.

Figure II. 3 connecteur de servomoteur

Un fil rouge est relié à l'alimentation positive (+5 ou +6 V selon le servo), le fil noir est relié à la masse (GND) et le fil jaune est utilisé pour la commande. Il y aurait beaucoup à dire sur le fonctionnement d'un servomoteur, ses composants, son moteur et le petit potentiomètre qui permet de connaître sa position mais nous nous limiterons à son utilisation avec ESP32.

II.2.5 Les avantages et les inconvénients des servomoteurs

Les avantages sont les suivants

Le fil signal à faible courant peut être raccordé directement à une sortie du microcontrôleur. Pas besoin de circuit d'interface.

On peut commander la marche, l'arrêt, le sens de rotation et la vitesse du servomoteur à l'aide d'un seul fil. Économie d'E/S.

Le servomoteur offre un couple important sous un volume réduit.

Les inconvénients :

Il faut remplacer le servomoteur pour une rotation complète.

Le prix est légèrement plus élevé qu'un bloc motoréducteur à moteurs CC.

II.2.6 Les fils de connexion

La figure suivante montre les fils de connexion utilisés avec la carte esp 32.

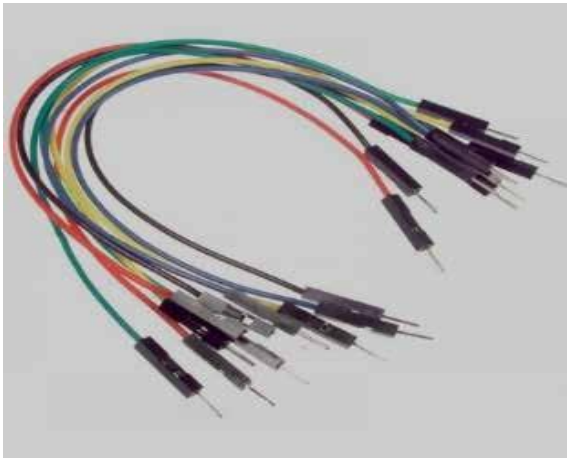


Figure II. 4 Les fils de connexion

II.3 Software :

II.3.1 Logiciel (Arduino IDE)

L'environnement de programmation Arduino (IDE en anglais) est une application écrite en Java. L'IDE permet d'écrire, de modifier un programme et de le convertir en une série d'instructions compréhensibles pour la carte. Le logiciel va nous permettre de programmer les cartes Arduino et les ESP32, il nous offre une multitude de fonctionnalités.

La structure des programmes Arduino est un peu particulière, en apparence, des structures habituelles du langage C. La syntaxe est la même qu'en langage C.

Figure II. 5 Logo du logiciel IDE Arduino

La syntaxe du langage arduino est composée de

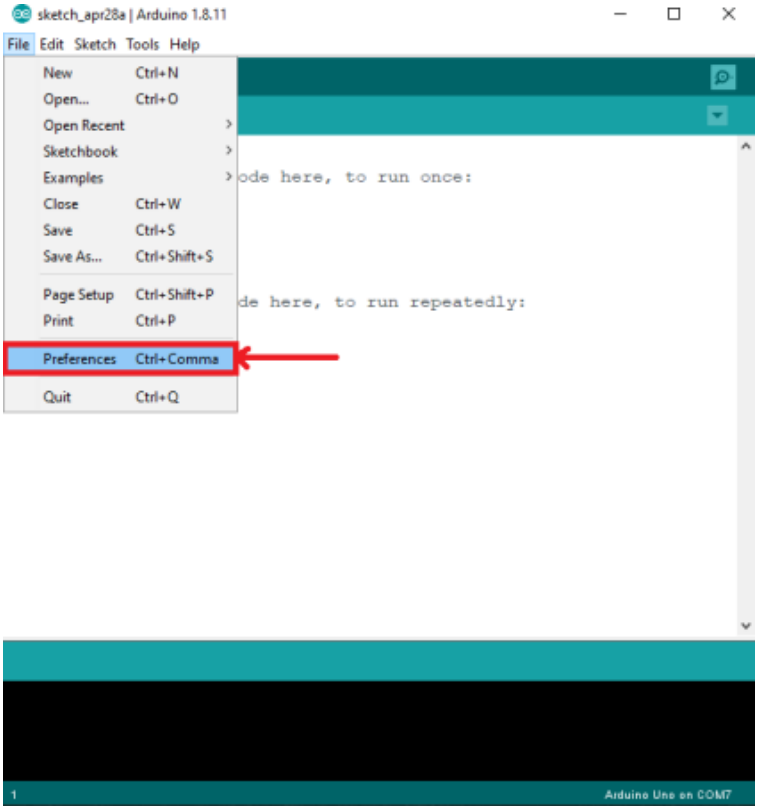
- **Ponctuation :** Le code est structuré par une ponctuation stricte
- **Les variables :** est un espace réservé dans la mémoire de l'ordinateur. C'est comme un compartiment dont la taille n'est adéquate que pour un seul type d'information.
- **Les fonctions :** est un bloc d'instructions que l'on peut appeler à tout endroit du programme.
- **Les structures de contrôle :** Les structures de contrôle sont des blocs d'instructions qui s'exécutent en fonction du respect d'un certain nombre de conditions. Il existe quatre types de structure : **(if, else) , while , for, (switch/case).**

Figure II. 6 Interface du logiciel de la carte Arduino

II.3.2 Arduino IDE et l'ESP32

Dans notre projet, on a utilisé l'**IDE** d'Arduino pour programmer **l'ESP32**, en procédant selon les étapes suivantes :

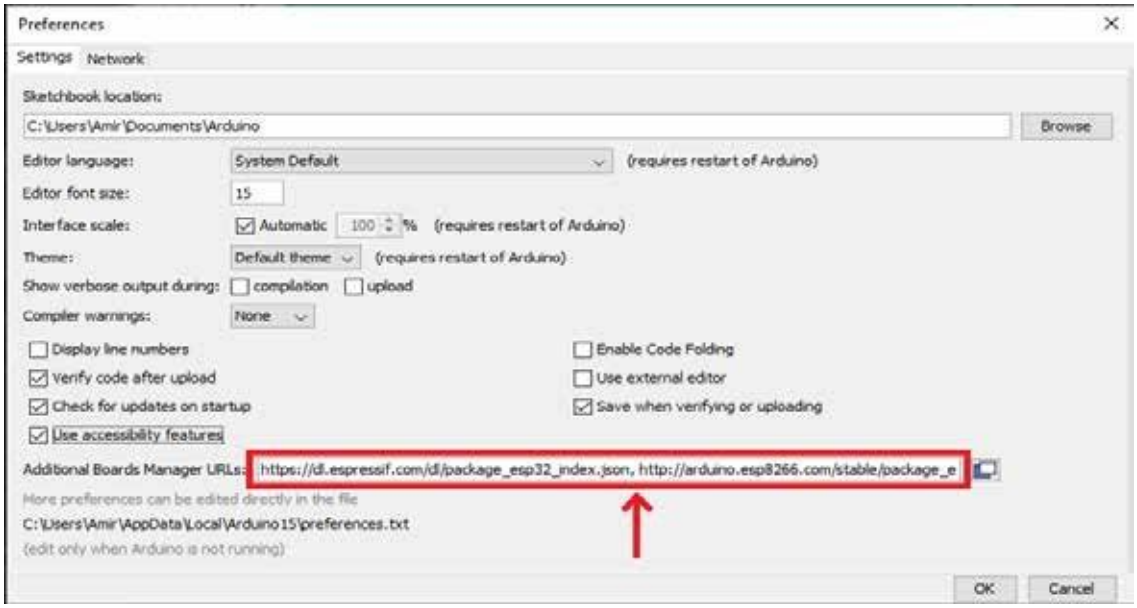
- En ouvre le programme Arduino IDE est dans la barre des menus on choisit **"File"** après « **Préférences ».**



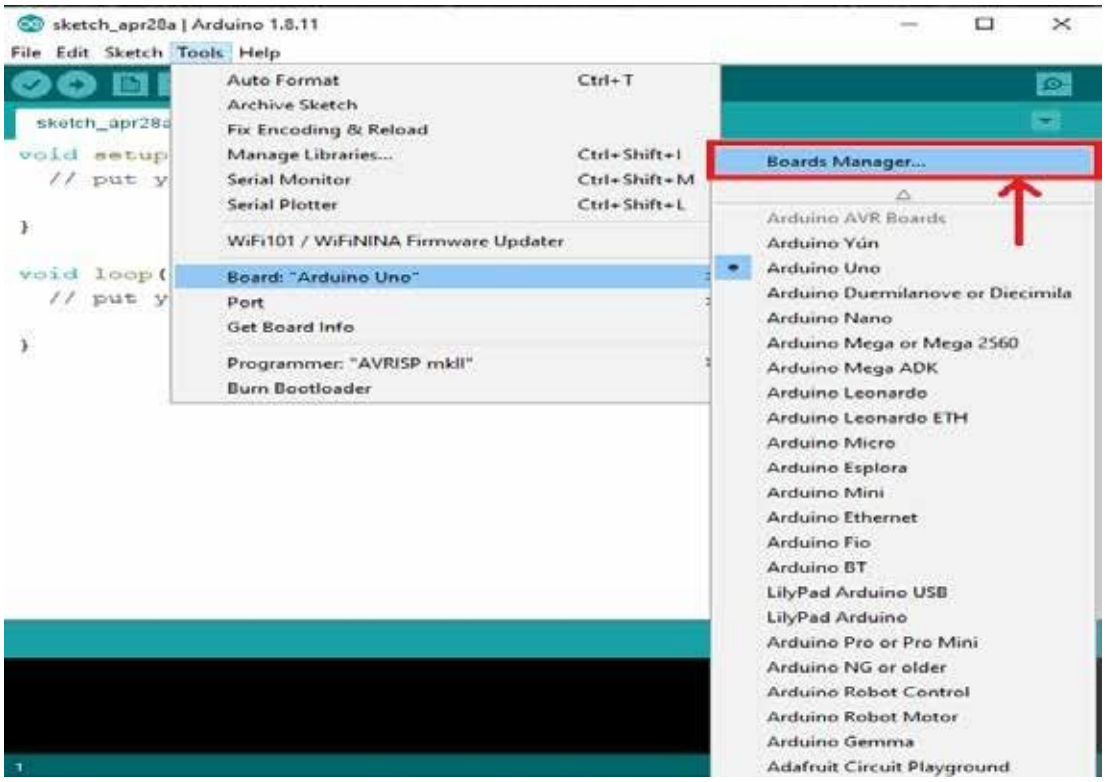
- Après l'ouverture de la fenêtre de "Preferences", et dans l'espace de "Additional Boards Manager URLs" on copie ce lien et on click "OK" :

https://dl.espressif.com/dl/package_esp32_index.json,

http://arduino.esp8266.com/stable/package_esp8266com_index.json

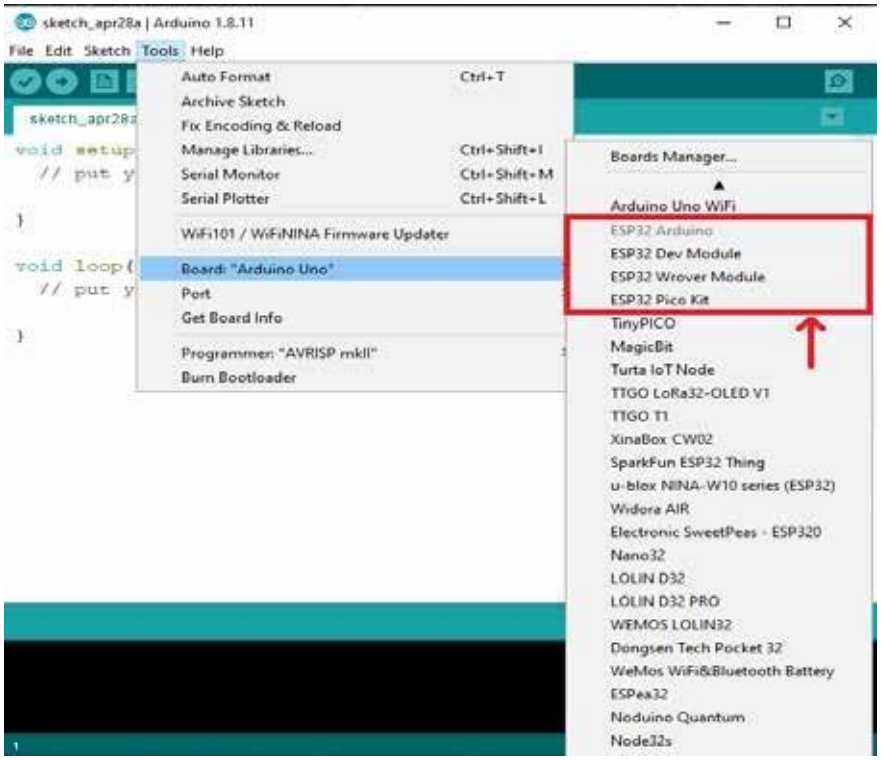


• Dans une deuxième étape, on ouvre la liste de "Tools" et on choisit "Board "après "Boards Manager…" :



• Dans la fenêtre de "Boards Manager" on recherche la carte "ESP32", un résultat est affiché de "Espressif Systems" qui contient le software des cartes ESP32, on clique sur "Install" et on attend quelques minutes.

• À la fin d'installation, on retourne à la liste de "Tools" après "Board" et maintenant on peut programmer n'importe quel carte ESP32.



II.3.3 Étude des fonctions de la librairie ESP32

II.3.3.1 Fonctions de la bibliothèque Wi-Fi ESP32 (Arduino IDE)

Les bibliothèques Wi-Fi prennent en charge la configuration et la surveillance des fonctionnalités Wi-Fi ESP32. Cela inclut une configuration pour :

✓ Mode station (également appelé mode **STA** ou mode client Wi-Fi). ESP32 se connecte à un point d'accès.

✓ Mode AP (également appelé mode Soft-AP ou mode point d'accès). Les stations se connectent à l'ESP32.

✓ Mode de cohabitation Station/AP (ESP32 est à la fois un point d'accès et une station connectée à un autre point d'accès).

✓ Différents modes de sécurité pour ce qui précède (WPA, WPA2, WEP, etc.)[17]

II.3.3.2 La bibliothèque Wi-Fi

La première chose à faire pour utiliser les fonctions Wi-Fi ESP32 est d'inclure la bibliothèque Wifi. h dans le code spécial, comme suit : #include <WiFi.h>

II.3.3.2 Point d'accès

Lorsque vous définissez votre carte ESP32 comme point d'accès, vous pouvez être connecté à l'aide de n'importe quel appareil doté de capacités Wi-Fi sans vous connecter à votre routeur. Ce point d'accès permet de créer son propre réseau Wi-Fi et les périphériques Wi-Fi (stations) à proximité peuvent s'y connecter, comme votre smartphone ou votre ordinateur. Ainsi, vous n'avez pas besoin d'être connecté à un routeur pour le contrôler.

Cela peut également être utile si vous souhaitez que plusieurs périphériques ESP32 communiquent entre eux sans avoir besoin d'un routeur.

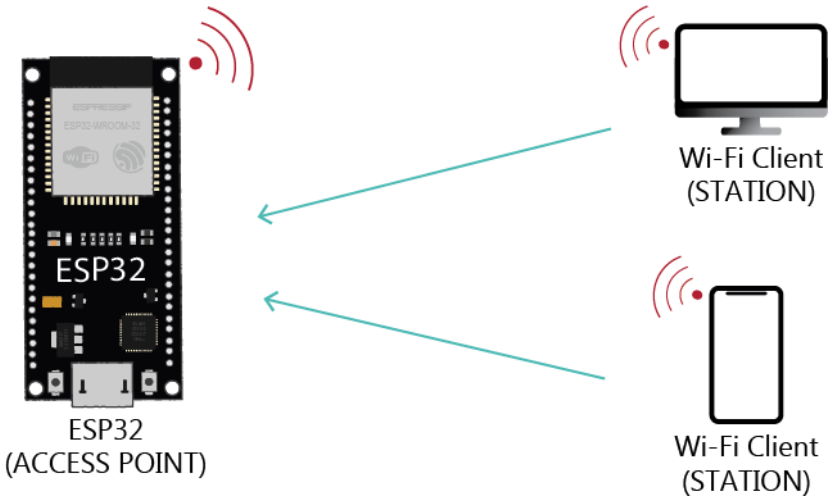


Figure II. 7 connectes-vous au Wi-Fi

II.3.4 Les modules wifi avec l'esp32-wroom

C'est un module WiFi et Bluetooth intégré basé sur le microcontrôleur ESP32. Il peut être utilisé pour créer à la fois des clients et des serveurs dans des applications réseau.

Voici comment nous avons pu configurer un client et un serveur avec l'ESP32-WROOM en utilisant le framework Arduino **:**

II.3.4.1 Client ESP32-WROOM

1. Initialisation des paramètres WiFi en se connectant à un réseau WiFi existant à l'aide de la fonction `WiFi.begin (ssid, password)`. Sans oublier d'inclure la bibliothèque WiFi.
2. Connection au serveur distant en créant une instance de la classe « WiFiClient » et en utilisant la fonction connect (serverIP, serverPort), où « serverIP » et « serverPort » sont respectivement l'adresse IP et le port du serveur distant.

3. Envois des données au serveur distant à l'aide de la fonction « write(data) » de l'objet « WiFiClient ».

Voici un exemple de code pour un client ESP32-WROOM :

```
#include <WiFi.h>

const char* ssid = "Votre_SSID";
const char* password = "Votre_Mot_de_passe";
const char* serverIP = "Adresse_IP_du_serveur";
const int serverPort = 1234;

void setup() {
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println("Connexion au WiFi...");
    }
    Serial.println("Connecté au WiFi");
}

void loop() {
    WiFiClient client;
    if (client.connect(serverIP, serverPort)) {
        Serial.println("Connecté au serveur");
        client.write("Données à envoyer");
        // Faites d'autres opérations avec le serveur si nécessaire
        client.stop(); // Fermez la connexion
    } else {
        Serial.println("Échec de la connexion au serveur");
        }
    delay(5000); // Attendez un certain temps avant d'envoyer de nouvelles données
}
```

II.3.4.2 Serveur ESP32-WROOM

1. Initialisation des paramètres Wi-Fi en créant un point d'accès WiFi à l'aide de la fonction « WiFi.softAP (ssid, password) ». Assurez-vous d'inclure la bibliothèque WiFi.

2. Créez un objet « Wifi Server » en spécifiant le port sur lequel le serveur écoutera les connexions entrantes.

3. Attendre qu'un client se connecte en utilisant la fonction « available () » de l'objet « Wifi Server ». Cette fonction renvoie un objet « Wifi Client » qui représente la connexion avec le client.

4. Lecture des données envoyées par le client à l'aide de la fonction « Read () » de l'objet « Wifi Client ».

5. Effectuer d'autres opérations avec les données reçues du client si nécessaire.

6. Fermeture de la connexion avec le client en utilisant la fonction « stop () » de l'objet « Wifi Client ».

II.3.4.3 Le code programme pour un serveur ESP32-WROOM

voici un exemple

```
#include <WiFi.h>

const char* ssid = "Nom_du_reseau";
const char* password = "Mot_de_passe_du_reseau";
const int serverPort = 1234;
WiFiServer server(server Port);

void setup() {
    WiFi.softAP(ssid, password);
    IPAddress localIP = WiFi.softAP();
    Serial. Print("Adresse IP du serveur : ");
    Serial.println(localIP);
    server.begin();
}

void loop() {
    WiFiClient client = server.available();
    if (client) {
        Serial.println("Client connecté");
        while (client.connected()) {
            if (client.available()) {
                String data = client.readStringUntil('\n');
                Serial.print("Données reçues : ");
                Serial.println(data);
```

```
                // Effectuez d'autres opérations avec les données reçues
            }
        }
        client.stop(); // Fermez la connexion avec le client
        Serial.println("Client déconnecté");
    }
}
```

II.4 Conclusion

Dans ce chapitre, nous avons présenté une étude détaillée sur les composants principaux qu'on a utilisé dans notre projet concernant la programmation d'un bras robots à 4 dégrées liberté avec ESP32WROOM. Nous avons présenté les caractéristiques de L'ESP32, les définitions et le brochage. Nous avons parlé aussi des éléments utilisés comme les servos et leurs fil de connexion. Nous avons expliqué comment utiliser le logiciel Arduino IDE et La syntaxe du langage Arduino. Nous avons montré comment utiliser la carte ESP32 avec Arduino IDE, sa bibliothèque, ses modules et sa fonction Wi-Fi.

Chapitre III

Réalisation du bras, développent software et résultats.

III.1 Introduction

Les chapitres précédents ont porté sur l'étude des principaux constituants d'un robot qui représentent les piliers de la réalisation de notre bras. Le monde assiste à une évolution de la technologie, où il était difficile d'imprimer des éléments de bras de robot et de réaliser des projets, mais maintenant, grâce à une imprimante 3D, il est facile de concevoir n'importe quelle forme le plus rapidement possible et avec une haute qualité pour bien accomplir vos recherches scientifiques. Dans ce chapitre, on va présenter la partie Réalisation du bras, développent software et résultat de notre bras manipulateur ainsi que la structure et les segments du bras. Dans la partie commande on va expliquer comment commander le bras manipulateur et ses mouvements.

III.2 Impression des composants du bras robot

L'impression des composants de bras de robot à l'aide d'une imprimante 3D est une pratique courante et largement adoptée dans le domaine de la robotique. Les imprimantes 3D permettent de produire rapidement des pièces sur mesure à partir de modèles numériques, offrant ainsi une grande flexibilité et une économie de coûts par rapport aux méthodes de fabrication traditionnelles.

Figure III. 2 L'imprimante 3D

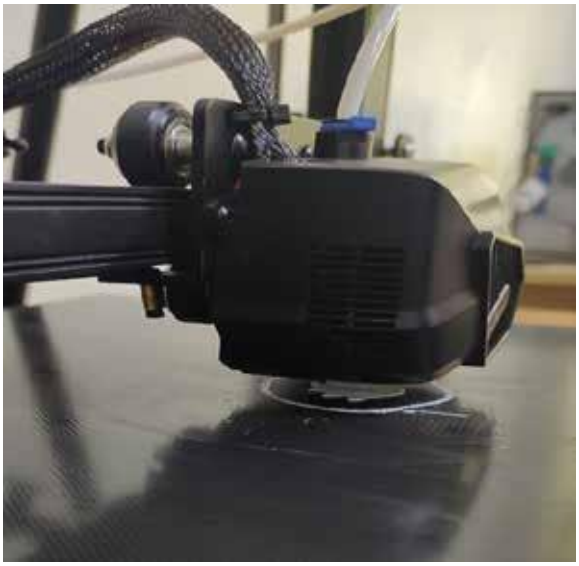


Figure III. 1 La tête de l'imprimante

Pour utiliser une imprimante 3D, voici les étapes générales à suivre :

- ✓ **Conception :** Utilisez un logiciel de modélisation 3D pour concevoir ou télécharger le modèle du composant de bras de robot que vous souhaitez imprimer. Assurez-vous que le modèle est adapté à l'impression 3D et qu'il est compatible avec votre imprimante.

- ✓ **Préparation du fichier :** Exportez le modèle en un format de fichier pris en charge par votre imprimante 3D, tel que STL ou OBJ. Vous pouvez également utiliser un logiciel de découpage (slicing) pour préparer le modèle, ce qui implique de définir les

paramètres d'impression, tels que la densité de remplissage, la résolution et les supports.

- ✓ **Configuration de l'imprimante :** Assurez-vous que votre imprimante 3D est correctement configurée et calibrée. Vérifiez les niveaux de lit, chargez le matériau d'impression (par exemple, le filament en plastique) et préchauffez l'extrudeuse à la température appropriée en fonction du matériau utilisé.

- ✓ **Impression :** Utilisez un logiciel de contrôle d'impression 3D pour envoyer le fichier préparé à l'imprimante. Assurez-vous que l'imprimante est correctement positionnée et lancez le processus d'impression.
  Surveillez attentivement le processus pour vous assurer que tout se déroule correctement et sans problèmes.

- ✓ **Post-traitement :** Une fois l'impression terminée, retirez délicatement le composant de bras de robot de la plateforme d'impression.
  Selon le matériau utilisé, vous pourriez avoir besoin d'effectuer des opérations supplémentaires, telles que le retrait des supports d'impression ou le ponçage pour obtenir une surface lisse.

III 3 : Résultats d'impression des composantes de notre bras robot

À l'aide d'une imprimante tridimensionnelle, nous avons imprimé les éléments du bras robot dans l'atelier de la faculté :

- ❖ **La base**

Figure III. 3 L'élément de la base

❖ **L’épaule**

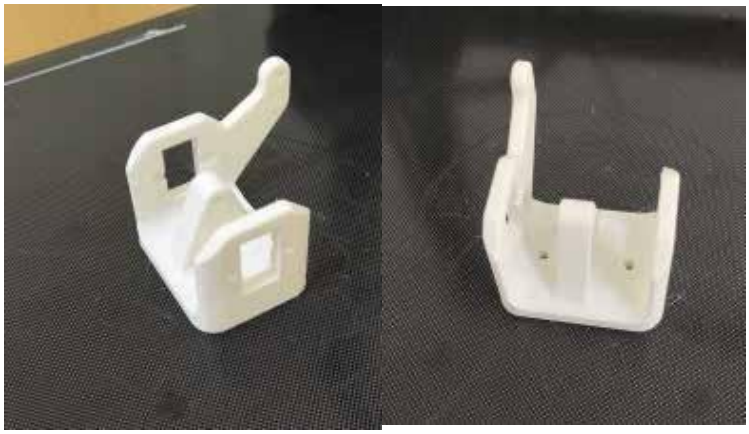


Figure III. 4 Les segmentes d'épaule

❖ **Le bras**

Figure III. 5 Les segmentes de coude

❖ **L’effecteur (outil final)**

Figure III. 6 les éléments de la pince

III.4 Développement logiciel

III.4.1 Etape de test des servos

Pour programmer le bras robot, nous avons d’abord fait plusieurs essais visant à contrôler un servo en utilisant ESP32 pour différents angles [0° , 180°] de rotation avec l’utilisation d’une bibliothèque <ESP32servo.h> comme indiqué dans la figure ci-dessous :

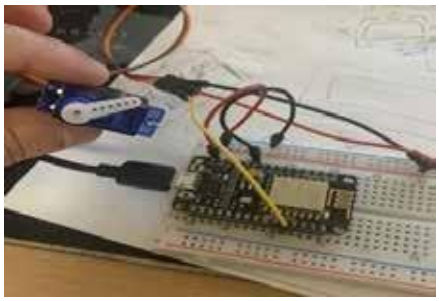


Figure III. 8 connexion électrique du servo à l'ESP32

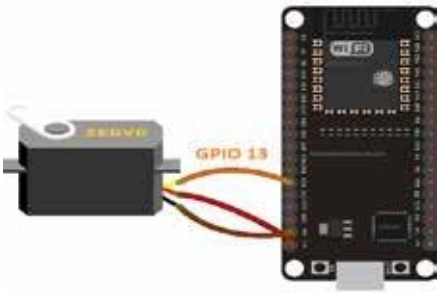


Figure III. 7 schéma électrique du servo

III.4.2 Liaison Client-Serveur

Les étapes à suivre pour accéder au serveur à partir d'un client

- ✓ Nous téléversons le programme de l'Arduino sur la carte esp32.
- ✓ Nous nous connectons au smartphone en utilisant le réseau Wi-Fi de la carte esp32 et l'interface du téléphone avec adresse IP et mot de passe.
- ✓ Enfin, nous pouvons contrôler les servos du bras robots avec précision et douceur pour les déplacer sous différents angles et effectuer diverses tâches.

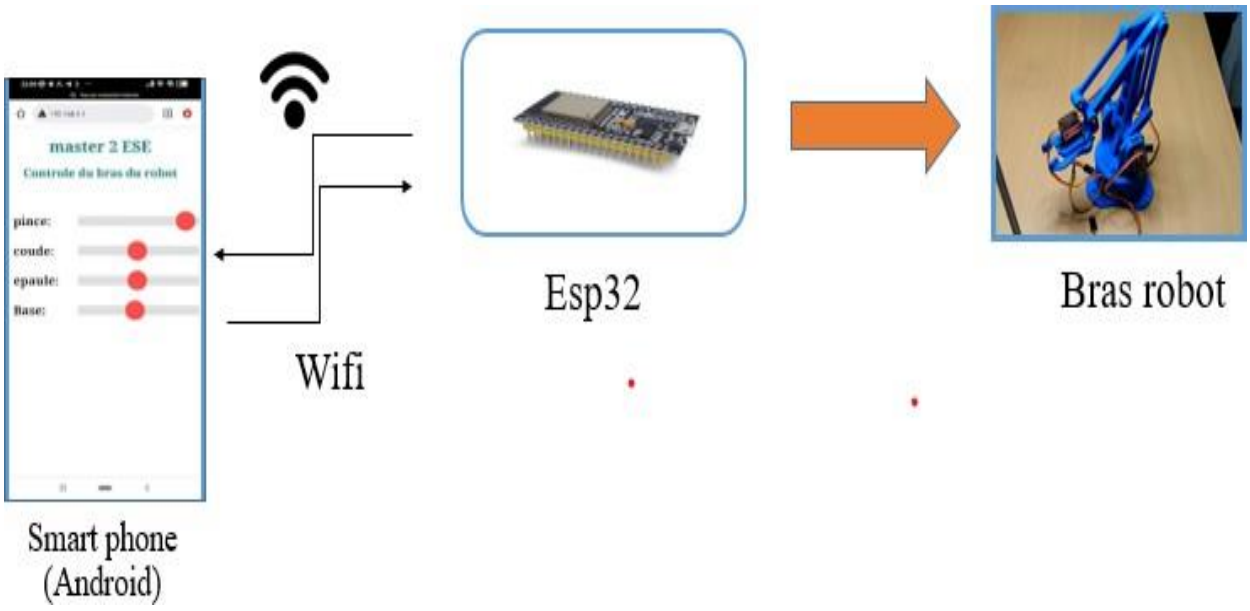
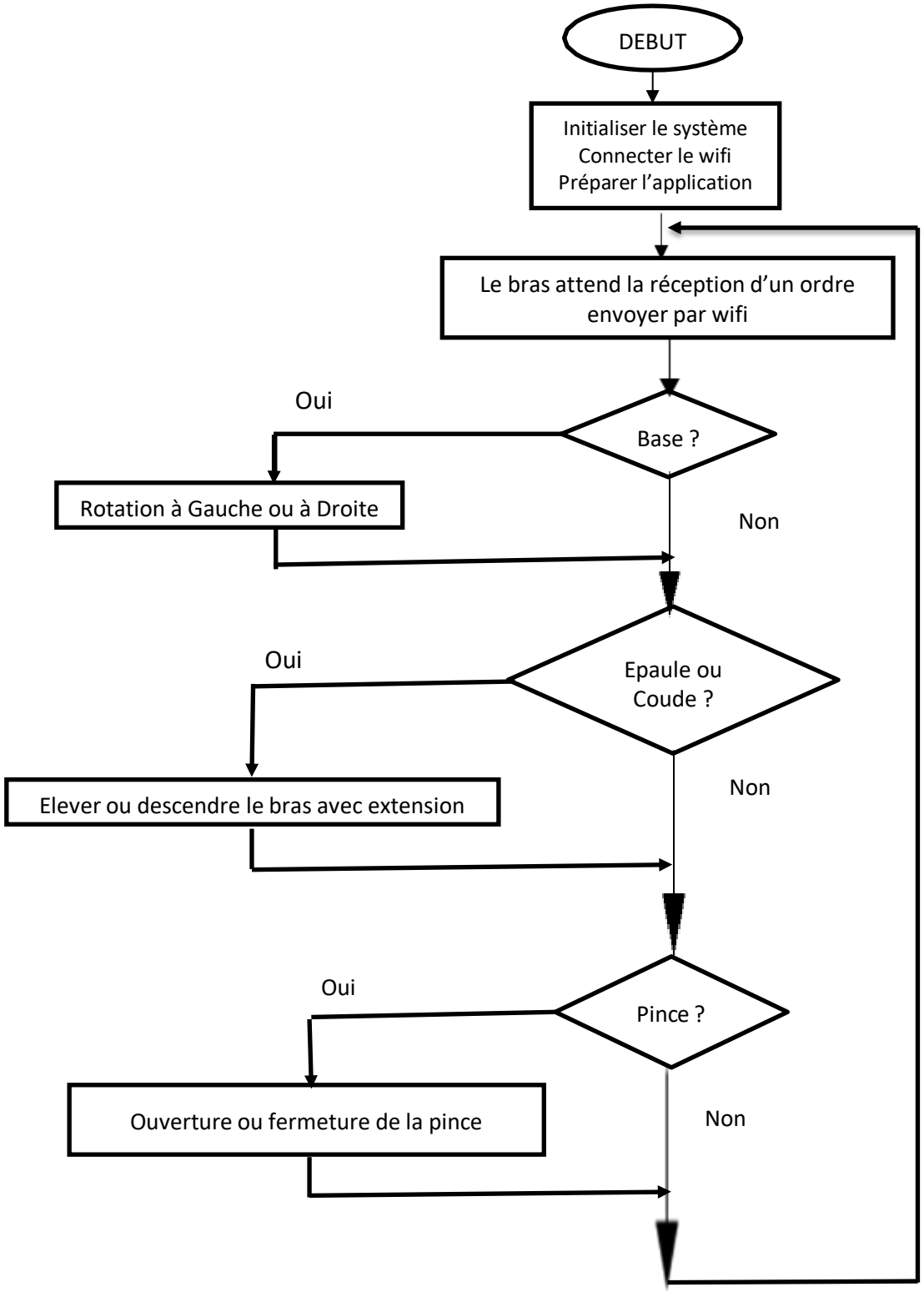


Figure III. 9 plant de projet

III.4.3 Organigramme du programme principal



III.5 : Partie programmation

Le code fourni est une implémentation d'un gestionnaire d'événements WebSocket dans une esquisse Arduino basée sur ESP32. Il est utilisé pour gérer les événements WebSocket liés à une application de contrôle de bras de robot.

III.5.1. Les librairies incluses dans le code

Ce code inclut plusieurs bibliothèques pour accomplir un projet Arduino basé sur ESP32.

1. «<**Arduino. h**>»: Il s'agit de la bibliothèque Arduino standard, qui fournit des fonctions et des définitions essentielles pour le développement d'Arduino.
2. «<**WiFi. h**>» : Cette bibliothèque vous permet de connecter votre ESP32 à un réseau sans fil, comme le Wi-Fi, et d'interagir avec les fonctions liées au réseau.
3. «<**AsyncTCP. h**>» : Cette bibliothèque prend en charge la communication TCP asynchrone, ce qui vous permet de créer une communication réseau non bloquante dans votre projet ESP32.
4. <**ESPAsyncWebServer. h**>» : Cette bibliothèque est utilisée pour configurer et exécuter un serveur Web asynchrone sur votre ESP32. Elle vous permet de créer des interfaces utilisateur Web et de gérer les requêtes HTTP.
5. <**ESP32Servo.h**>» : Cette bibliothèque est utilisée pour contrôler les servomoteurs avec un ESP32. Les servomoteurs sont couramment utilisés dans la robotique et d'autres projets pour contrôler le mouvement de divers composants.
6. <**iostream**> et <**sstream**> : Ce sont des en-têtes de bibliothèque standard C++. Ils ne sont généralement pas utilisés dans les projets Arduino, car l'IDE Arduino est basé sur C/C++ et n'utilise souvent pas la bibliothèque standard C++. Ces en-têtes peuvent être inclus pour la compatibilité du code avec d'autres plateformes.

III.5.2. Explication brève du code développé

1. ***Structure servos :***
   - Cette structure définit un ensemble de propriétés liées à un servomoteur. Il comprend un objet « *Servo* », un « *int* » représentant la broche du servo, une « chaîne » pour le nom du servo et un « *int* » pour la position initiale.
2. ***servoPins std::vector<ServoPins>***
   - Il s'agit d'un vecteur (tableau dynamique) de structures « *ServoPins* ». Il initialise un tableau de broches servo avec leurs propriétés associées.
3. ***Structure « RecordedStep »***
   - Cette structure définit un ensemble de propriétés liées à l'enregistrement et à la lecture des étapes. Il comprend un *int* pour l'index *servo*, un *int* pour la valeur et un *int* pour le délai entre les étapes.
4. ***"std::vector<RecordedStep> recordedSteps***
   - Il s'agit d'un autre vecteur de structures « *RecordedStep* », vraisemblablement utilisées pour l'enregistrement et la lecture de séquences de mouvements de servo.

5. ***Divers indicateurs booléens***

   - « *recordSteps* » et « *playRecordedSteps* » sont des drapeaux booléens qui déterminent si les mouvements des servomoteurs doivent être enregistrés ou lus.

6. ***«unsigned long TimeInMilli»***

   - Cette variable enregistre l'heure précédente en millisecondes et est souvent utilisée pour les calculs de synchronisation et d'intervalle.

7. ***Configuration WiFi***

   - Il définit le SSID et le mot de passe d'un réseau Wi-Fi auquel l'appareil se connectera.

8. ***AsyncWebServer et AsyncWebSocket :***

   - Ce code initialise un *AsyncWebServer* et un *AsyncWebSocket* avec des routes et des terminaux spécifiques, ce qui suggère que le code peut faire partie d'une application de serveur Web pour contrôler un bras de robot sur le réseau.

9. ***Le code HTML***

   - Le contenu HTML est stocké dans la variable *'htmlHomePage'* et il est envoyé aux clients lorsqu'ils accèdent à l'URL racine du serveur Web.

   - Ce code fourni une page HTML pour contrôler le bras du robot à l'aide d'une interface Web. Il inclut des éléments HTML, CSS et JavaScript, et il est destiné à être servi par un serveur Web basé sur ESP32 en utilisant la bibliothèque *'ESPAsyncWebServer'*.

   - L'interface comprend des curseurs et des boutons pour ajuster les positions et les actions du bras. La connexion *WebSocket* permet une communication en temps réel entre la page Web et l'ESP32, permettant aux utilisateurs d'interagir avec le bras du robot et de le contrôler à distance.

<u>les principaux composants et fonctionnalités de cette page HTML</u>

1. *« **Structure HTML** »*

   - La structure HTML contient divers éléments, y compris des en-têtes, des tableaux et des curseurs, qui sont utilisés pour contrôler différents aspects du bras du robot.

2. *« **Style CSS** »*

   - La section " <**style**> définit les styles CSS de la page, y compris l'apparence des boutons, la conception du curseur et l'alignement du texte.

3. *« **Curseurs** »*

   - Il existe des curseurs pour contrôler différentes parties du bras du robot, telles que la pince, le coude, l'épaule et la base. Ces curseurs permettent aux utilisateurs de définir la position ou l'angle souhaité pour chaque membre.

4. *« **Communication WebSocket** »*

   - La section JavaScript met en place une connexion WebSocket à un serveur fonctionnant sur l'ESP32 (*'webSocketRobotArmInputUrl'*) pour échanger des données avec le bras du robot.

- La fonction *'initRobotArmInputWebSocket'* initialise la connexion WebSocket et gère des événements tels que *'onopen'*, *'onclose'* et *'onmessage'*.

- La fonction *'sendButtonInput'* envoie les données d'entrée du bouton au serveur via WebSocket.

5. *« **Gestion des boutons** »*

   - Les fonctions JavaScript *'onclickButton'* et *'enableDisableButtonsSliders'* gèrent les clics sur les boutons et activent / désactivent les boutons et les curseurs en fonction de certaines conditions.

   - Les boutons ont deux états, « ON » et « OFF », et changent de couleur (vert ou rouge) en fonction de leur état.

6. *« **Auditeurs de l'événement** »*

   - L'événement *'window.onload'* initie la connexion WebSocket lors du chargement de la page.

   - Il existe un écouteur d'événements pour *'touchend'* afin d'éviter le comportement par défaut lorsque vous touchez des éléments de la page.

7. *« **Gestion des requêtes du serveur** »*

   - La fonction *'handleRoot'* sert la page HTML lors de l'accès à l'URL racine.

   - La fonction *'handleNotFound'* est utilisée pour répondre avec une erreur 404 si le fichier demandé n'est pas trouvé. Cette page HTML est conçue pour contrôler les mouvements d'un bras de robot via une interface Web.

8. *« **Signature de fonction** »*

   - *'onRobotArmInputWebSocketEvent'* est une fonction de rappel qui est appelée lorsque des événements WebSocket se produisent. Elle prend plusieurs paramètres, y compris le serveur WebSocket, le client, le type d'événement, l'argument, les données et la longueur des données.

9. *« **Gestion des événements** »*

   La fonction utilise une instruction *'switch'* pour gérer différents types d'événements WebSocket :

   - 'WS_EVT_CONNECT' : Ce cas est déclenché lorsqu'un client se connecte au WebSocket. Il enregistre les informations sur le client connecté et envoie l'état actuel du bras du robot au client.

- 'WS_EVT_DISCONNECT' : Ce cas est déclenché lorsqu'un client se déconnecte du WebSocket. Il enregistre les informations sur le client déconnecté.

   - 'WS_EVT_DATA' : Ce boîtier gère les données entrantes du WebSocket. Il analyse les données pour extraire la clé et la valeur, qui doivent être dans un format séparé par des virgules. Selon la clé reçue, il effectue différentes actions :

     « Enregistrer » et « Jouer » : Ces touches sont utilisées pour contrôler l'enregistrement et la lecture des mouvements du bras du robot.

     « Base », « Épaule », « Coude » et « Préhenseur » : Ces touches sont utilisées pour contrôler les positions d'asservissement des différents composants du bras du robot.

Les valeurs associées à ces touches sont analysées et utilisées en conséquence pour contrôler les actions du bras du robot.

10. *« **Autres types d'événements** »*

- Le code inclut également des cas pour la gestion des événements 'WS_EVT_PONG' et 'WS_EVT_ERROR', bien qu'ils ne contiennent pas d'actions spécifiques pour le moment.

Ce code est essentiel pour gérer la communication avec les clients connectés au serveur WebSocket. Il permet aux clients d'envoyer des commandes pour contrôler le bras du robot, telles que l'enregistrement, la lecture et le réglage des positions servo. Il fournit également des commentaires sur l'état de la connexion du client.

L'implémentation suppose que la communication WebSocket implique l'échange de paires clé-valeur dans un format texte, où la clé spécifie l'action à effectuer et la valeur fournit les informations nécessaires pour cette action.

Le code développé contient deux fonctions qui font partie de l'esquisse Arduino pour contrôler un bras de robot via une interface Web et la communication WebSocket. Examinons chacune de ces fonctions :

1. Fonction *'sendCurrentRobotArmState'* :

- Cette fonction est responsable de l'envoi de l'état actuel du bras du robot aux clients connectés au serveur WebSocket.
- Il itére à travers le vecteur *'servoPins'*, qui contient des informations sur chaque servo, et envoie le nom du servo et sa position actuelle à tous les clients connectés en utilisant *'wsRobotArmInput.textAll()*'.
- De plus, il envoie l'état des boutons « Enregistrer » et « Lecture », indiquant si l'enregistrement ou la lecture sont actuellement actifs ou non.

2. Fonction *'writeServoValues'* :

- Cette fonction est chargée d'écrire des valeurs dans les servomoteurs et, éventuellement, d'enregistrer ces valeurs si le mode d'enregistrement est actif.
- Elle prend deux paramètres : *'servoIndex'*, qui indique l'index du servo dans le vecteur *'servoPins'*, et *'value'*, qui représente la nouvelle position ou l'angle à définir pour le servo.
- À l'intérieur de la fonction, si le mode d'enregistrement (*'recordSteps'*) est actif, elle crée une structure *'RecordedStep'* pour enregistrer la position du servo et le temps écoulé depuis l'étape précédente. Ceci est utile pour une lecture ultérieure.
- Si le vecteur *'recordedSteps'* est vide (indiquant l'étape d'enregistrement de la position initiale), elle enregistre la position initiale de tous les servos.
- Elle enregistre ensuite la position actuelle du servo spécifié et calcule le temps écoulé depuis l'étape précédente.

- Après avoir enregistré l'étape, elle met à jour la variable *'previousTimeInMilli'* avec l'heure actuelle.
- Enfin, elle définit la nouvelle position du servo en utilisant *'servoPins[servoIndex].servo.write(value)*'.

Ces deux fonctions jouent un rôle crucial dans la mise à jour de l'état du bras du robot et la gestion de l'enregistrement des pas. La fonction *'sendCurrentRobotArmState'* fournit un retour en temps réel aux clients connectés, tandis que la fonction *'writeServoValues'* gère à la fois les mises à jour de la position des servomoteurs et l'enregistrement des mouvements si nécessaires.

La configuration assurée par la fonction « *setup()* » comprend les fonctions suivantes :

1) ***'setUpPinModes'***, est responsable de la configuration des modes de broches et des positions initiales des servomoteurs définis dans le vecteur *'servoPins'*. Voici une ventilation de ce que fait cette fonction :

   Elle utilise une boucle *'for'* pour itérer à travers les éléments du vecteur *'servoPins'*. La boucle itére de l'index 0 à la taille du vecteur (*'servoPins.size()*'). Pour chaque élément du vecteur *'servoPins'*, elle effectue les actions suivantes :

   - Appelle la méthode *'attach'* sur l'objet *'Servo'* (*'servoPins[i].servo'*) associé à cette broche du servo. Cette méthode est utilisée pour initialiser et attacher le servo à une broche spécifique, ce qui le rend prêt pour le contrôle.
   - Définit la position initiale du servo à l'aide de la méthode *'write'*. La position initiale est spécifiée par *'servoPins[i].initialPosition'*. Cette action positionne le servo au point de départ souhaité.

   En résumé, la fonction *'setUpPinModes'* est responsable de la préparation des servomoteurs pour le fonctionnement. Elle attache chaque servo à sa broche désignée et définit sa position initiale. Cette fonction est généralement appelée une fois au début du programme pour initialiser les servos avant que des commandes de mouvement ou de contrôle ne leur soient émises.

2) ***'Serial.begin(115200)***' : Cette ligne initialise la communication série avec un débit en bauds de 115200. Ceci est souvent utilisé pour le débogage et la surveillance de la sortie du programme. Les informations sont généralement envoyées à un moniteur série, ce qui vous permet d'observer le comportement du programme.
3) ***'WiFi.softAP(ssid, mot de passe)***' : Ce code configure un point d'accès logiciel (AP) avec le SSID (nom du réseau) et le mot de passe fournis. Ceci est généralement utilisé lorsqu'on souhaite que l'appareil (par exemple, un ESP8266 ou ESP32) agisse comme un point d'accès Wi-Fi, permettant à d'autres appareils de s'y connecter.

4) ***'Serial.print(« AP IP address : ( )***' et ***'Serial.println(« 192.168.4.1 ( )***' : Ces lignes impriment des informations sur l'adresse IP du point d'accès au moniteur série. Dans ce cas, cela indique que l'adresse IP de l'AP est 192.168.4.1.
5) Les lignes suivantes définissent des itinéraires et des gestionnaires pour un serveur HTTP à l'aide de l'objet *'server'*. Il spécifie deux gestionnaires :

- *'**server.on**(« / », HTTP_GET, handleRoot)*' : Lorsqu'une requête GET est envoyée à l'URL racine (« / »), elle appelle la fonction *'handleRoot'* pour traiter la requête.
- *'**server.onNotFound**(handleNotFound)*' : Si une URL demandée ne correspond à aucune des routes définies, elle appellera la fonction *'handleNotFound'* pour traiter la requête.

6) *'**wsRobotArmInput.onEvent**(onRobotArmInputWebSocketEvent)'* : Cette ligne configure un gestionnaire d'événements WebSocket. Elle associe la fonction *'onRobotArmInputWebSocketEvent'* aux événements WebSocket reçus via la connexion WebSocket *'wsRobotArmInput'*.
7) *'**server.addHandler**(&wsRobotArmInput)*' : Cette ligne ajoute la connexion WebSocket *'wsRobotArmInput'* en tant que gestionnaire pour le serveur HTTP. Cela permet la communication WebSocket avec des requêtes HTTP standard.
8) *'**server.begin()***' : Cela démarre le serveur HTTP, le rendant prêt à écouter les requêtes HTTP entrantes et les connexions WebSocket.
9) *'**Serial.println**(« serveur HTTP démarré »)*' : Cette ligne imprime un message sur le moniteur série pour indiquer que le serveur HTTP a démarré avec succès.

En résumé, la fonction de configuration est responsable de l'initialisation et de la configuration de divers composants, tels que les servos, la communication série, le point d'accès Wi-Fi, le serveur HTTP et la connexion WebSocket, afin de préparer l'appareil au fonctionnement.

Le reste du code fournit une fonction appelée *'playRecordedRobotArmSteps'*. Cette fonction permet de contrôler un bras de robot ou un mécanisme similaire en jouant une séquence de pas enregistrés. Dans la suite, nous expliquons ce que fait chaque partie de ce code :

1) Il vérifie s'il y a des étapes enregistrées dans le vecteur *'recordedSteps'*. S'il n'y a pas d'étapes enregistrées (c'est-à-dire que le vecteur est vide), la fonction rend la main immédiatement.
2) Il initialise les servos du bras du robot à leurs positions initiales en interpolant itérativement entre les positions servo actuelles et les positions initiales pour les 4 premières étapes. L'objectif est de déplacer en douceur les servos vers leurs positions initiales. Ce mouvement se poursuit tant que la variable *'playRecordedSteps'* est vraie.
3) Après avoir atteint les positions initiales des servos, la fonction introduit un délai de 2 secondes avant de commencer les étapes réellement enregistrées. Ce délai laisse le temps aux servos de se stabiliser dans leurs positions initiales.
4) La fonction itère ensuite sur les étapes enregistrées restantes à partir de l'index 4 (en supposant que ces étapes représentent les actions réelles que le bras du robot doit effectuer). Pour chaque étape, il effectue les opérations suivantes :
   - Attend une durée spécifiée par *'recordedStep.delayInStep'*.
   - Définit le servo à *'recordedStep.servoIndex'* à la position spécifiée par *'recordedStep.value'*.
   - Envoie un message via *'wsRobotArmInput'* avec des informations sur le nom du servo et la nouvelle position du servo.

Notons que le code repose sur certaines variables et objets externes ou globaux, tels que *'recordedSteps'*, *'playRecordedSteps'*, *'servoPins'*, *'servoName'* et *'wsRobotArmInput'*. Pour bien comprendre et utiliser ce code, il faut examiner comment ces variables sont définies et gérées dans le contexte plus large de notre programme. En outre, le code suppose qu'il n'y a aucun problème avec les données fournies et que les bibliothèques servo et toutes les autres dépendances sont correctement configurées.

III.6. Conclusion

Dans ce chapitre, nous avons expliqué comment nous avons utilisé une imprimante 3D pour imprimer les différents composants du bras robot dans une première étape, puis nous avons montrés les branchements réalisés entre la carte ESP32 et le bras robot, enfin nous avons essayé de clarifier le rôle de chaque fonction du code rédigé, du côté client (la page web développée pour le téléphone) et du coté serveur (hébergé sur l'ESP32 auquel est relié le bras robot). Le code dans sa totalité permet de contrôler à distance, à l'ide d'un téléphone, un bras robot à quatre degrés de libertés.

Conclusion générale

Le présent projet a exploré le domaine passionnant de la commande à distance d'un robot, avec un accent particulier sur l'utilisation de l'ESP32 comme plateforme de contrôle et l'intégration d'une interface HTML pour permettre une interaction intuitive. Notre Travail a montré les avancées significatives réalisées dans ce domaine et a mis en lumière les opportunités prometteuses qu'offre cette combinaison technologique.

L'un des principaux résultats de notre travail est la conception et la réalisation réussies d'un système de commande à distance d'un bras fonctionnel. Ce système, basé sur l'ESP32, offre une flexibilité remarquable, permettant à un opérateur humain de prendre le contrôle d'un robot à distance, en temps réel, via une interface HTML conviviale. Les essais pratiques et les évaluations ont démontré la robustesse de notre système dans divers scénarios d'application.

La commande par wifi d'un bras robot programmé par ESP32 offre des solutions innovantes pour des situations où l'intervention humaine directe est limitée ou risquée, Ouvrant la voie vers des applications dans divers domaines, du secteur médical à la recherche scientifique.

Cependant, il est essentiel de reconnaître que ce domaine présente encore des défis à relever. Les considérations de sécurité, la latence de communication et l'ergonomie de l'interface utilisateur sont autant des domaines qui nécessitent une attention continue et des améliorations constantes. Les futurs projets auront l'opportunité d'explorer ces défis et de contribuer à l'évolution de cette technologie.

En conclusion, ce projet a montré le potentiel immense de la commande à distance d'un robot programmé par ESP32 contrôlé via une interface HTML. Nous espérons que ce travail servira de fondement pour les promos futures et inspirera des projets concrets visant à exploiter pleinement cette technologie. Alors que la robotique continue de s'étendre dans des domaines toujours plus vastes, le contrôle à distance des robots offre une passerelle vers un avenir où les machines et les humains peuvent collaborer de manière plus étroite et efficace pour relever des défis complexes et divers.

Références

[1] BELFAR Anes : Elaboration d'un Langage de Programmation pour le Robot Edtf ED-7220C : Université de Mohamed El-Bachir El-Ibrahimi - Bordj Bou Arreridj :2022

[2] https ://www.larousse.fr/dictionnaires/francais/robot/69647

[3] Liyla BELADI : Conception et réalisation d'UN bras manipulateur à trois degrés de liberté : UNIVERSITE MOULOUD MAMMERI DE TIZI-OUZOU :29/09/2015

[4] par OUMOUSSA Yacine : Commande à distance d'un robot mobile : UNIVERSITEMOULOUD MAMMERI DE TIZI-OUZOU : le 15/09/2016

[5] CHEMOUNE Mourad : Commande en position d'un bras manipulateur à trois degrés de liberté avec une carte Arduino : UNIVERSITE MOULOUD MAMMERI DE TIZI-OUZOU : 12/ 07/ 2018

[6] ACHOUR HALIM : CONCEPTION ET REALISATION D'UN BRAS MANIPULATEUR à 3 DEGEREE LIBERTE : université SAAD DAHLAB de Blida :2016_2017

[7] SOLTANI Amir : Étude du mouvement optimal d'un bras manipulateur à 3 degrés de liberté en exécutant des taches imposées : Université Djilali BOUNAAMA Khemis Miliana : 2020/2021

[8] Mohamed LARIBI : Commande gestuelle d'un bras manipulateur à 4 degrés de liberté : UNIVERSITE MOULOUD MAMMERI DE TIZI-OUZOU : 01 / 10 / 2015

[9] DJEBIRI Mohamed : Réalisation et commande d'un bras robotique à quatre degrés de liberté : UNIVERSITE AKLI MOHAND OULHADJE-BOUIRA : 2018/2019

[10] Djebarni Alaeddine : Etude et conception d'un bras de robot : Université Mohamed El Bachir El Ibrahimi Bordj Bou Arréridj : 2020/2021

[11] Sadoudi Abdelhamid : Conception et réalisation d'un bras Robot manipulateur piloté par application Android : UNIVERSITE YAHIA FARES DE MEDEA : 2020

[12] HAMDACHE Zine Eddine : Réalisation et commande d'un bras robotique à quatre degrés de liberté : UNIVERSITE AKLI MOHAND OULHADJE-BOUIRA : 2018/2019

[13] ABED OUSSAMA : Etude et Réalisation d'un système de reconnaissance faciale basé sur une carte ESP32-cam et la librairie OpenCV pour le langage Python : Université de Mohamed El-Bachir El-Ibrahimi - Bordj Bou Arreridj : 26/06/2022.

[14] Rui Santos and Sara Santos, "Learn ESP32 with Arduino IDE.pdf," VERSION

[15] Esp32_datasheet.

[16] https://randomnerdtutorials.com/esp32-useful-wi-fi-functions-arduino/

[17] Amir Djebiri : Etude et réalisation d'un compteur d'énergie intelligent : UNIVERSITE KASDI MERBAH OUARGLA : 2019/2020.

Annexes

```
#include <Arduino.h>
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>

#include <ESP32Servo.h>
#include <iostream>
#include <sstream>

struct ServoPins
{
  Servo servo;
  int servoPin;
  String servoName;
  int initialPosition;
};
std::vector<ServoPins> servoPins =
{
  { Servo(), 27 , "Base", 90},
  { Servo(), 33 , "Shoulder", 90},
  { Servo(), 32 , "Elbow",90},
  { Servo(), 26 , "Gripper", 90},
};

struct RecordedStep
{
  int servoIndex;
  int value;
  int delayInStep;
};
std::vector<RecordedStep> recordedSteps;

bool recordSteps = false;
bool playRecordedSteps = false;

unsigned long previousTimeInMilli = millis();

const char* ssid     = "RobotArm";
const char* password = "12345678";

AsyncWebServer server(80);
AsyncWebSocket wsRobotArmInput("/RobotArmInput");

const char* htmlHomePage PROGMEM = R"HTMLHOMEPAGE(
<!DOCTYPE html>
<html>
  <head>
```

```
  <meta name="viewport" content="width=device-width, initial-scale=1, maximum-
scale=1, user-scalable=no">
    <style>
    input[type=button]
    {
      background-color:red;color:white;border-
radius:30px;width:100%;height:40px;font-size:20px;text-align:center;
    }

    .noselect {
      -webkit-touch-callout: none; /* iOS Safari */
        -webkit-user-select: none; /* Safari */
         -khtml-user-select: none; /* Konqueror HTML */
           -moz-user-select: none; /* Firefox */
            -ms-user-select: none; /* Internet Explorer/Edge */
                user-select: none; /* Non-prefixed version, currently
                                      supported by Chrome and Opera */
    }
    .slidecontainer {
      width: 100%;
    }
    .slider {
      -webkit-appearance: none;
      width: 100%;
      height: 20px;
      border-radius: 5px;
      background: #d3d3d3;
      outline: none;
      opacity: 0.7;
      -webkit-transition: .2s;
      transition: opacity .2s;
    }
    .slider:hover {
      opacity: 1;
    }

    .slider::-webkit-slider-thumb {
      -webkit-appearance: none;
      appearance: none;
      width: 40px;
      height: 40px;
      border-radius: 50%;
      background: red;
      cursor: pointer;
    }
    .slider::-moz-range-thumb {
      width: 40px;
      height: 40px;
      border-radius: 50%;
```

```
      background: red;
      cursor: pointer;
    }
    </style>

  </head>
  <body class="noselect" align="center" style="background-color:white">

    <h1 style="color: teal;text-align:center;">master 2 ESE</h1>
    <h2 style="color: teal;text-align:center;">Controle du bras du robot</h2>

    <table id="mainTable" style="width:400px;margin:auto;table-layout:fixed"
CELLSPACING=10>
      <tr/><tr/>
      <tr/><tr/>
      <tr>
        <td style="text-align:left;font-size:25px"><b>pince:</b></td>
        <td colspan=2>
         <div class="slidecontainer">
            <input type="range" min="0" max="180" value="90" class="slider"
id="Gripper" oninput='sendButtonInput("Gripper",value)'>
          </div>
        </td>
      </tr>
      <tr/><tr/>
      <tr>
        <td style="text-align:left;font-size:25px"><b>coude:</b></td>
        <td colspan=2>
         <div class="slidecontainer">
            <input type="range" min="50" max="130" value="90" class="slider"
id="Elbow" oninput='sendButtonInput("Elbow",value)'>
          </div>
        </td>
      </tr>
      <tr/><tr/>
      <tr>
        <td style="text-align:left;font-size:25px"><b>epaule:</b></td>
        <td colspan=2>
         <div class="slidecontainer">
            <input type="range" min="50" max="130" value="90" class="slider"
id="Shoulder" oninput='sendButtonInput("Shoulder",value)'>
          </div>
        </td>
      </tr>
      <tr/><tr/>
      <tr>
        <td style="text-align:left;font-size:25px"><b>Base:</b></td>
        <td colspan=2>
         <div class="slidecontainer">
```

```
            <input type="range" min="10" max="180" value="90" class="slider"
id="Base" oninput='sendButtonInput("Base",value)'>
          </div>
        </td>
      </tr>
      <tr/><tr/>
      <tr>
    </table>

    <script>
      var webSocketRobotArmInputUrl = "ws:\/\/" + window.location.hostname +
"/RobotArmInput";
      var websocketRobotArmInput;

      function initRobotArmInputWebSocket()
      {
        websocketRobotArmInput = new WebSocket(webSocketRobotArmInputUrl);
        websocketRobotArmInput.onopen    = function(event){};
        websocketRobotArmInput.onclose   =
function(event){setTimeout(initRobotArmInputWebSocket, 2000);};
        websocketRobotArmInput.onmessage    = function(event)
        {
          var keyValue = event.data.split(",");
          var button = document.getElementById(keyValue[0]);
          button.value = keyValue[1];
          if (button.id == "Record" || button.id == "Play")
          {
            button.style.backgroundColor = (button.value == "ON" ? "green" :
"red");
            enableDisableButtonsSliders(button);
          }
        };
      }

      function sendButtonInput(key, value)
      {
        var data = key + "," + value;
        websocketRobotArmInput.send(data);
      }

      function onclickButton(button)
      {
        button.value = (button.value == "ON") ? "OFF" : "ON" ;
        button.style.backgroundColor = (button.value == "ON" ? "green" :
"red");
        var value = (button.value == "ON") ? 1 : 0 ;
        sendButtonInput(button.id, value);
        enableDisableButtonsSliders(button);
      }
```

```
      function enableDisableButtonsSliders(button)
      {
        if(button.id == "Play")
        {
          var disabled = "auto";
          if (button.value == "ON")
          {
            disabled = "none";
          }
          document.getElementById("Gripper").style.pointerEvents = disabled;
          document.getElementById("Elbow").style.pointerEvents =
disabled;
          document.getElementById("Shoulder").style.pointerEvents =
disabled;
          document.getElementById("Base").style.pointerEvents = disabled;
          document.getElementById("Record").style.pointerEvents = disabled;
        }
        if(button.id == "Record")
        {
          var disabled = "auto";
          if (button.value == "ON")
          {
            disabled = "none";
          }
          document.getElementById("Play").style.pointerEvents = disabled;
        }
      }

      window.onload = initRobotArmInputWebSocket;
      document.getElementById("mainTable").addEventListener("touchend",
function(event){
        event.preventDefault()
      });
    </script>
  </body>
</html>
)HTMLHOMEPAGE";

void handleRoot(AsyncWebServerRequest *request)
{
  request->send_P(200, "text/html", htmlHomePage);
}

void handleNotFound(AsyncWebServerRequest *request)
{
        request->send(404, "text/plain", "File Not Found");
}
```

```
void onRobotArmInputWebSocketEvent(AsyncWebSocket *server,
                      AsyncWebSocketClient *client,
                      AwsEventType type,
                      void *arg,
                      uint8_t *data,
                      size_t len)
{
  switch (type)
  {
    case WS_EVT_CONNECT:
      Serial.printf("WebSocket client #%u connected from %s\n", client->id(),
client->remoteIP().toString().c_str());
      sendCurrentRobotArmState();
      break;
    case WS_EVT_DISCONNECT:
      Serial.printf("WebSocket client #%u disconnected\n", client->id());
      break;
    case WS_EVT_DATA:
      AwsFrameInfo *info;
      info = (AwsFrameInfo*)arg;
      if (info->final && info->index == 0 && info->len == len && info->opcode
== WS_TEXT)
      {
        std::string myData = "";
        myData.assign((char *)data, len);
        std::istringstream ss(myData);
        std::string key, value;
        std::getline(ss, key, ',');
        std::getline(ss, value, ',');
        Serial.printf("Key [%s] Value[%s]\n", key.c_str(), value.c_str());
        int valueInt = atoi(value.c_str());

        if (key == "Record")
        {
          recordSteps = valueInt;
          if (recordSteps)
          {
            recordedSteps.clear();
            previousTimeInMilli = millis();
          }
        }
        else if (key == "Play")
        {
          playRecordedSteps = valueInt;
        }
        else if (key == "Base")
        {
          writeServoValues(0, valueInt);
        }
```

```
        else if (key == "Shoulder")
        {
          writeServoValues(1, valueInt);
        }
        else if (key == "Elbow")
        {
          writeServoValues(2, valueInt);
        }
        else if (key == "Gripper")
        {
          writeServoValues(3, valueInt);
        }

      }
      break;
    case WS_EVT_PONG:
    case WS_EVT_ERROR:
      break;
    default:
      break;
  }
}

void sendCurrentRobotArmState()
{
  for (int i = 0; i < servoPins.size(); i++)
  {
    wsRobotArmInput.textAll(servoPins[i].servoName + "," +
servoPins[i].servo.read());
  }
  wsRobotArmInput.textAll(String("Record,") + (recordSteps ? "ON" : "OFF"));
  wsRobotArmInput.textAll(String("Play,") + (playRecordedSteps ? "ON" :
"OFF"));
}

void writeServoValues(int servoIndex, int value)
{
  if (recordSteps)
  {
    RecordedStep recordedStep;
    if (recordedSteps.size() == 0) // We will first record initial position of
all servos.
    {
      for (int i = 0; i < servoPins.size(); i++)
      {
        recordedStep.servoIndex  =  i;
        recordedStep.value = servoPins[i].servo.read();
        recordedStep.delayInStep = 0;
        recordedSteps.push_back(recordedStep);
```

```
      }
    }
    unsigned long currentTime = millis();
    recordedStep.servoIndex = servoIndex;
    recordedStep.value = value;
    recordedStep.delayInStep = currentTime - previousTimeInMilli;
    recordedSteps.push_back(recordedStep);
    previousTimeInMilli = currentTime;
  }
  servoPins[servoIndex].servo.write(value);
}

void playRecordedRobotArmSteps()
{
  if (recordedSteps.size() == 0)
  {
    return;
  }
  //This is to move servo to initial position slowly. First 4 steps are
initial position
  for (int i = 0; i < 4 && playRecordedSteps; i++)
  {
    RecordedStep &recordedStep = recordedSteps[i];
    int currentServoPosition =
servoPins[recordedStep.servoIndex].servo.read();
    while (currentServoPosition != recordedStep.value && playRecordedSteps)
    {
      currentServoPosition = (currentServoPosition > recordedStep.value ?
currentServoPosition - 1 : currentServoPosition + 1);
      servoPins[recordedStep.servoIndex].servo.write(currentServoPosition);
      wsRobotArmInput.textAll(servoPins[recordedStep.servoIndex].servoName +
"," + currentServoPosition);
      delay(50);
    }
  }
  delay(2000); // Delay before starting the actual steps.

  for (int i = 4; i < recordedSteps.size() && playRecordedSteps ; i++)
  {
    RecordedStep &recordedStep = recordedSteps[i];
    delay(recordedStep.delayInStep);
    servoPins[recordedStep.servoIndex].servo.write(recordedStep.value);
    wsRobotArmInput.textAll(servoPins[recordedStep.servoIndex].servoName + ","
+ recordedStep.value);
  }
}

void setUpPinModes()
{
```

```
  for (int i = 0; i < servoPins.size(); i++)
  {
    servoPins[i].servo.attach(servoPins[i].servoPin);
    servoPins[i].servo.write(servoPins[i].initialPosition);
  }
}


void setup(void)
{
  setUpPinModes();
  Serial.begin(115200);

  WiFi.softAP(ssid, password);
  Serial.print("AP IP address: ");
  Serial.println("192.168.4.1");

  server.on("/", HTTP_GET, handleRoot);
  server.onNotFound(handleNotFound);

  wsRobotArmInput.onEvent(onRobotArmInputWebSocketEvent);
  server.addHandler(&wsRobotArmInput);

  server.begin();
  Serial.println("HTTP server started");

}

void loop()
{
  wsRobotArmInput.cleanupClients();
  if (playRecordedSteps)
  {
    playRecordedRobotArmSteps();
  }
}
```