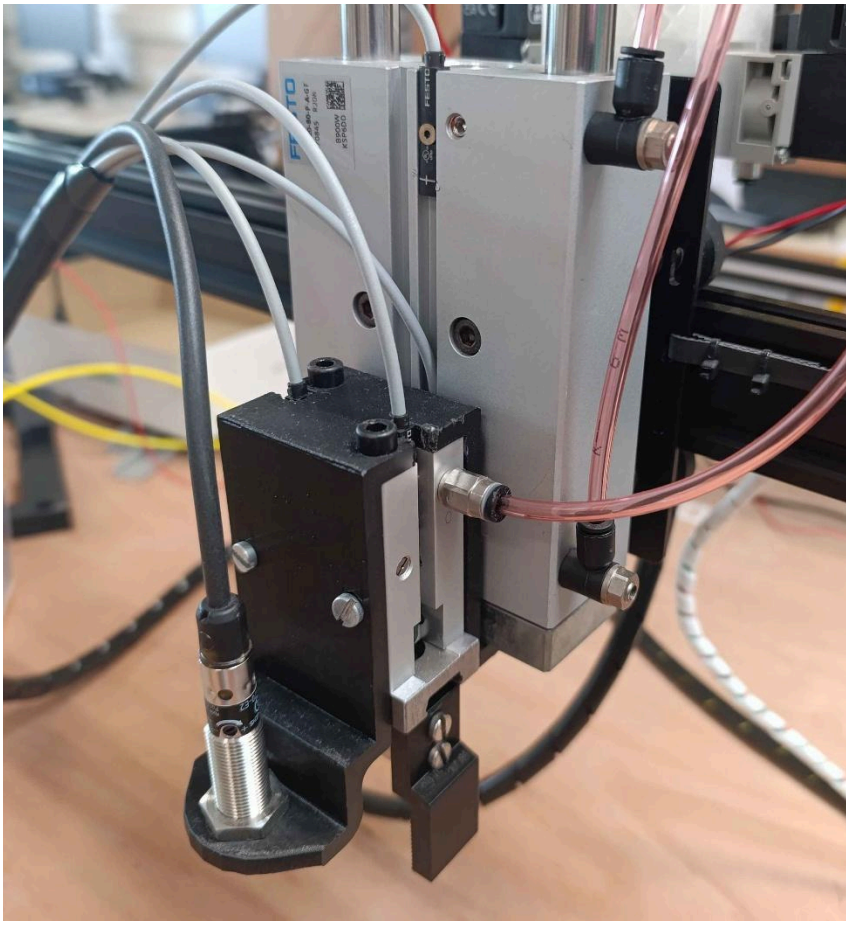
PROJET ECO RESPONSABLE

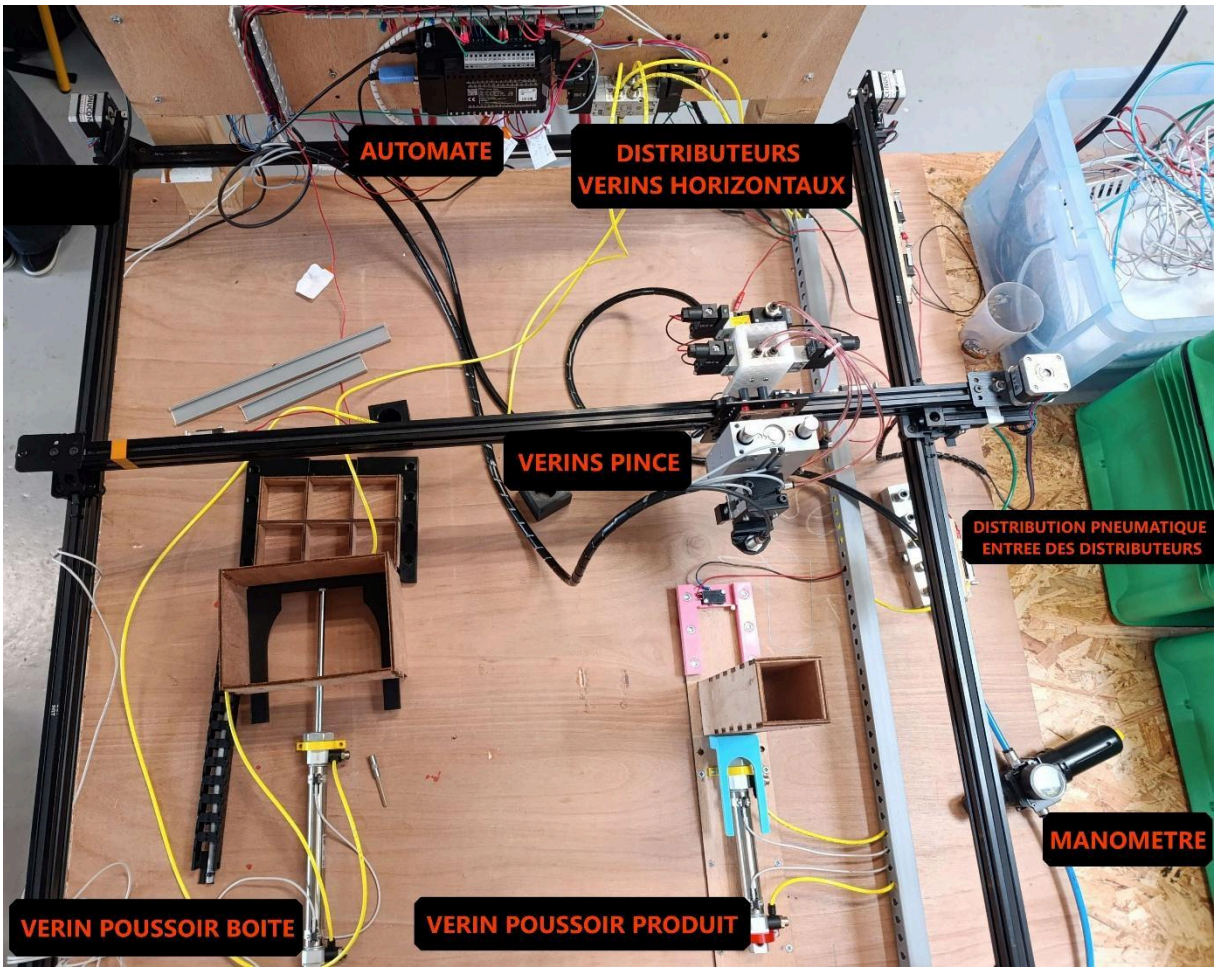
Robot Pick & Place

Documentation V1



Douae Choubri, Reda El Madani, Salma Letrach, Ayoub Bouquennouche, Souad Ait Bellauali, Achraf Bouchemama, Roua Boughanmi, Amaury Pinéda

Architecture globale du robot – Espace de travail

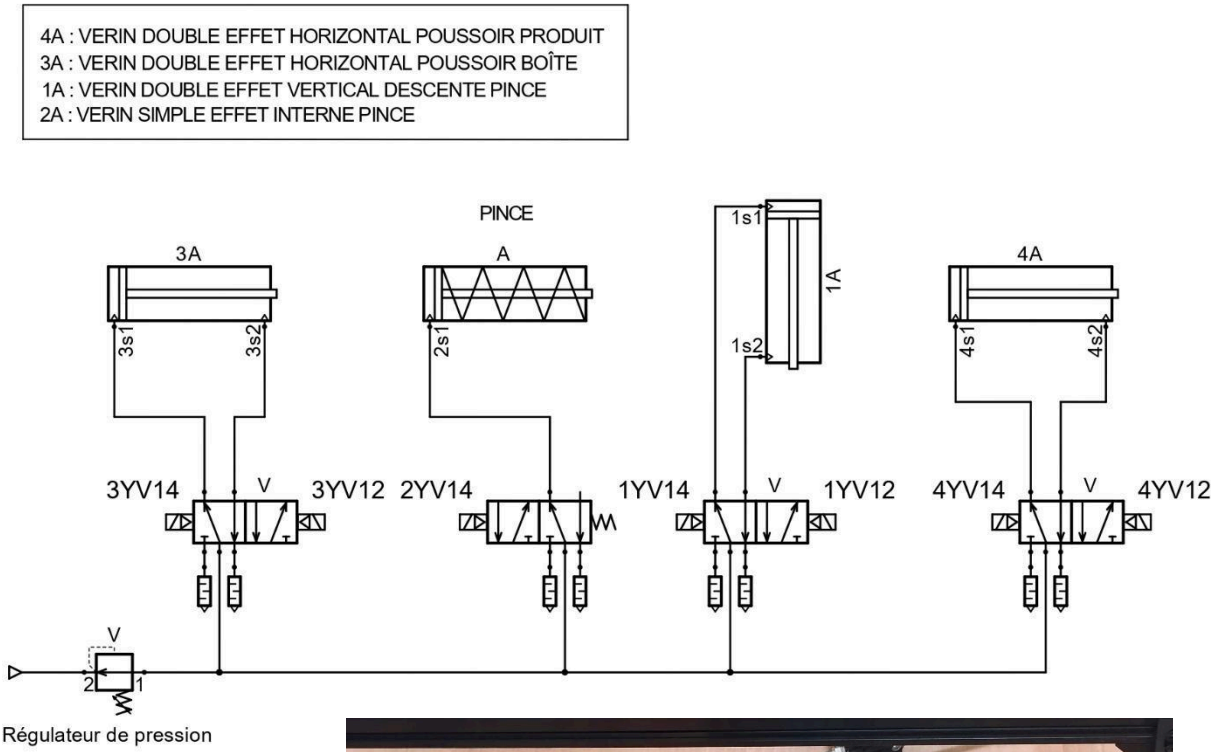


Douae Choubri, Reda El Madani, Salma Letrach, Ayoub Bouquennouche, Souad Ait Bellauali, Achraf Bouchemama, Roua Boughanmi, Amaury Pinéda

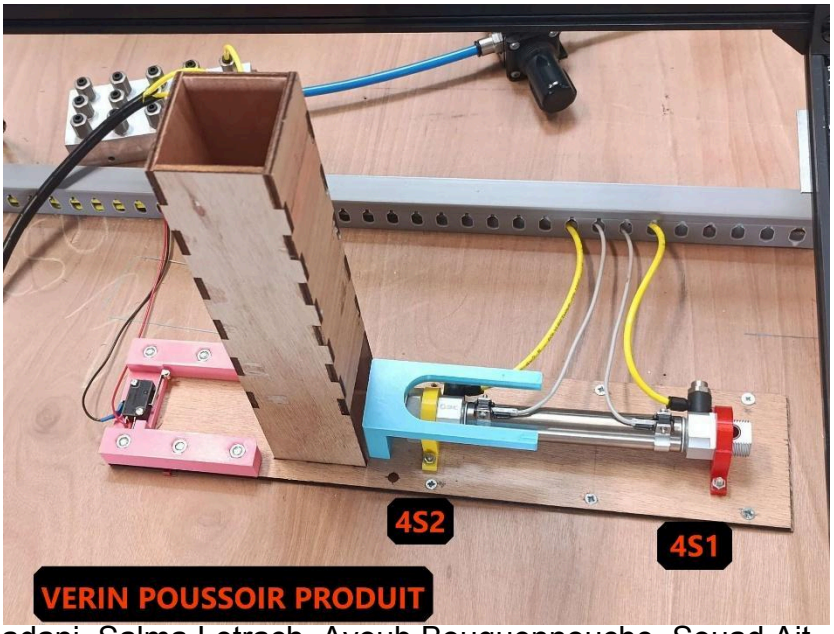
Partie pneumatique

Les éléments pneumatiques du système sont notamment des distributeurs électropneumatiques et des vérins servant à actionner le robot sur l'axe Z, la fermeture de la pince du robot ainsi que deux vérins horizontaux servant à la mise en position des éléments

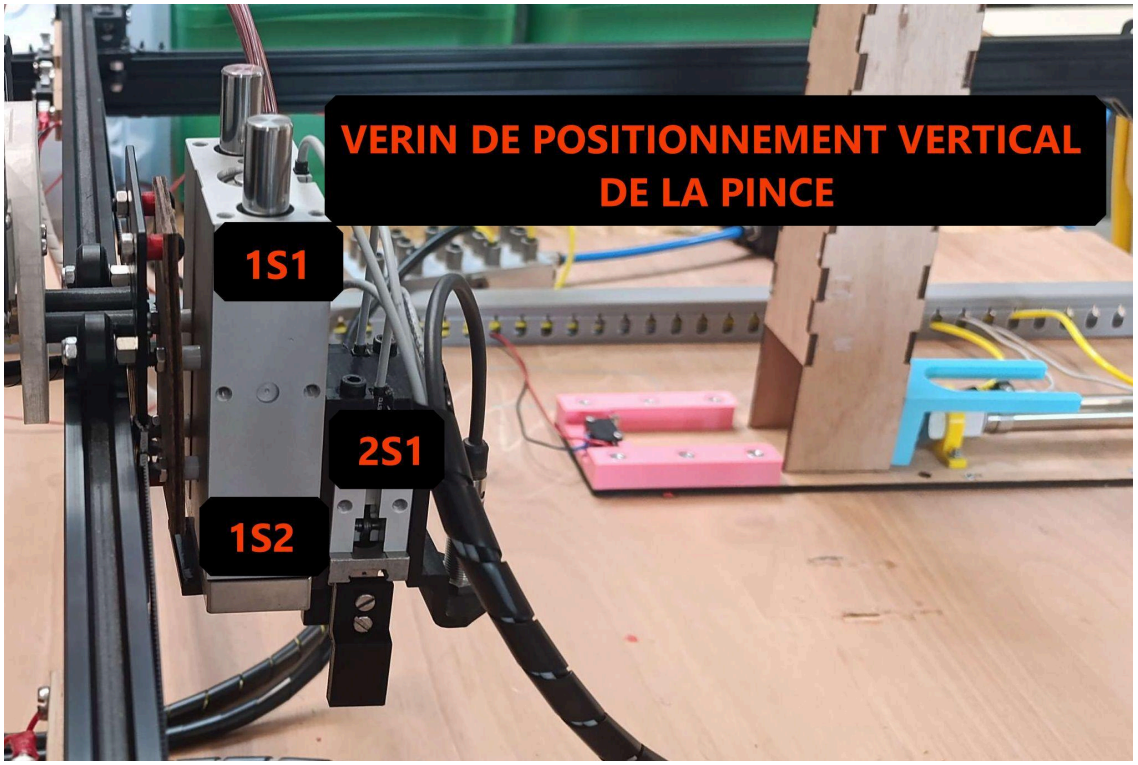
- Schéma

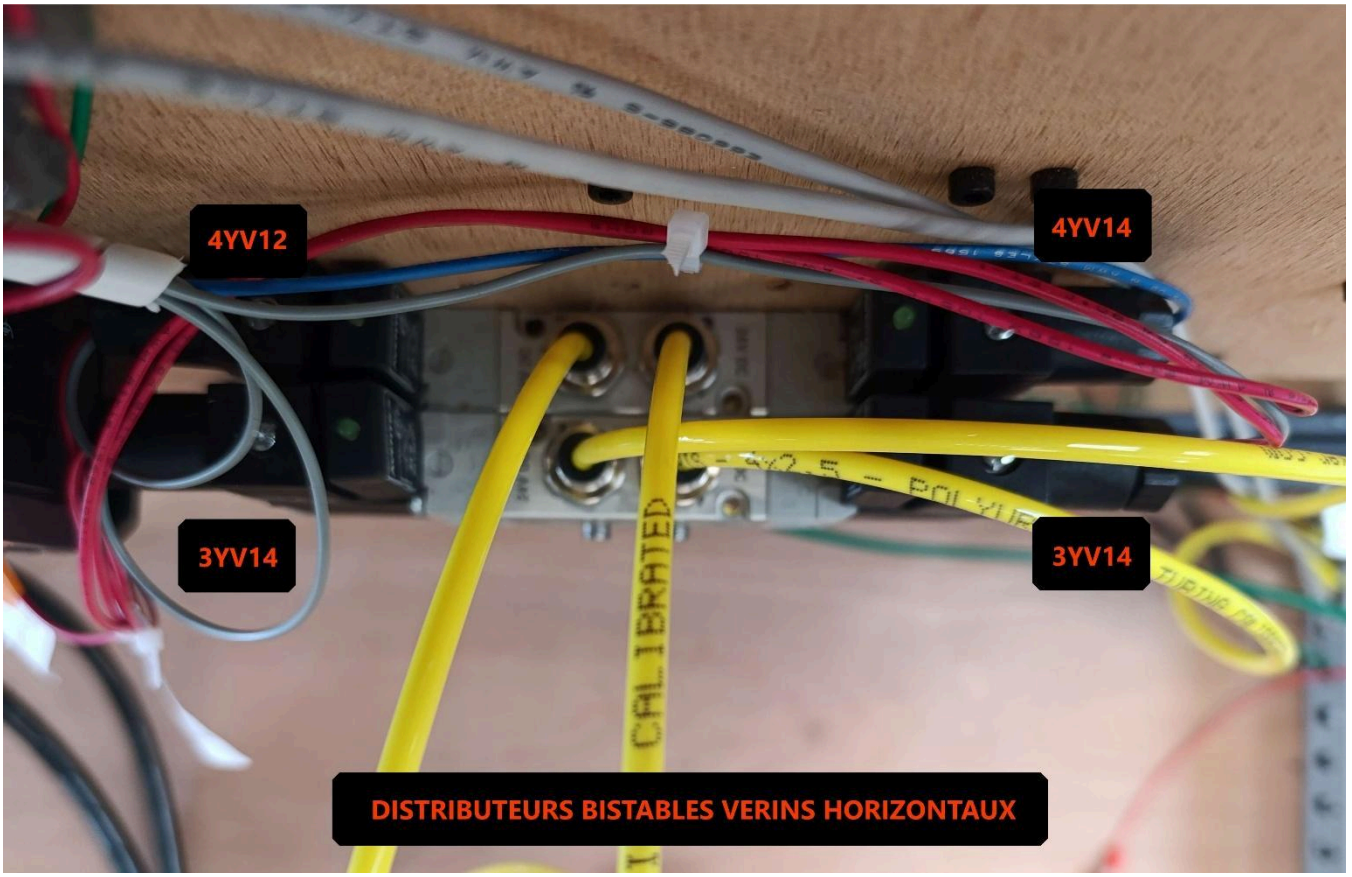


Régulateur de pression

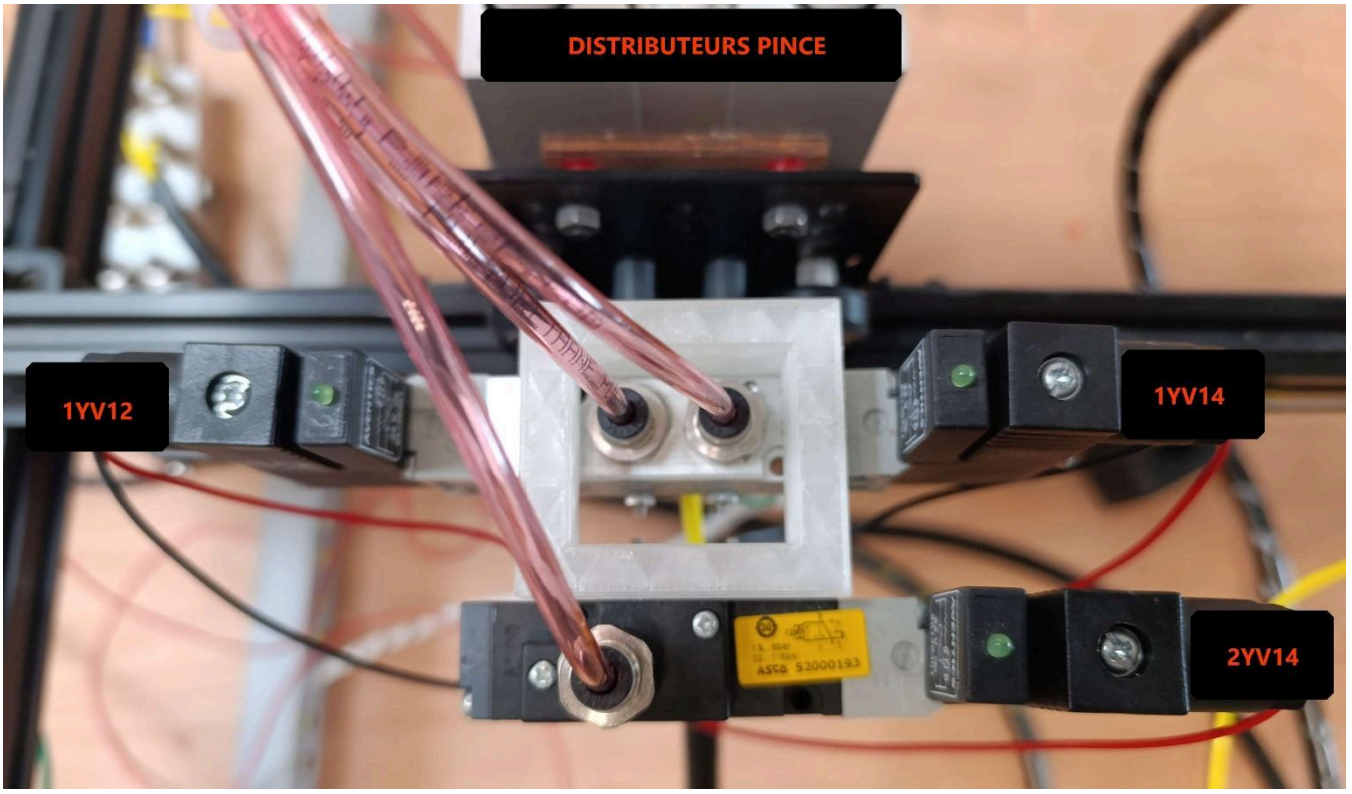


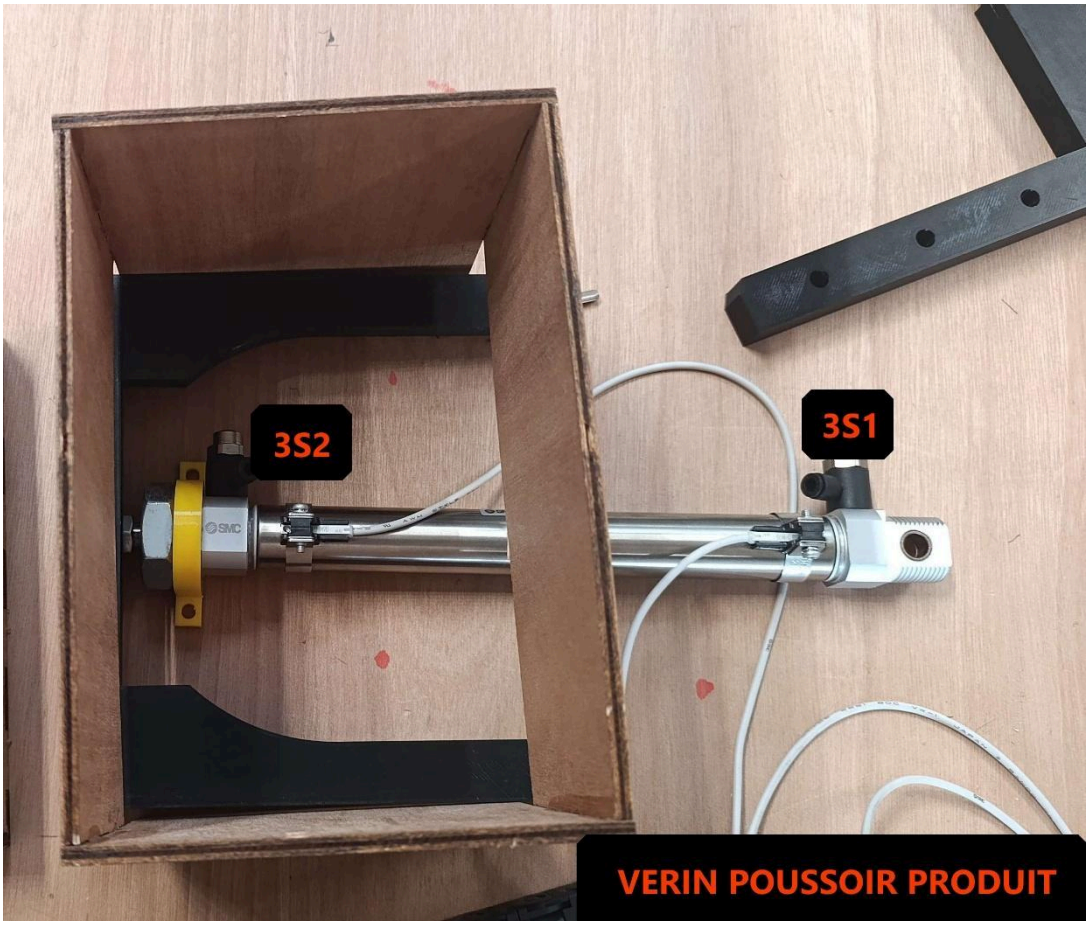
Douae Choubri, Reda El Madani, Salma Letrach, Ayoub Bouquennouche, Souad Ait Bellauali, Achraf Bouchemama, Roua Boughanmi, Amaury Pinéda





Douae Choubri, Reda El Madani, Salma Letrach, Ayoub Bouquennouche, Souad Ait Bellauali, Achraf Bouchemama, Roua Boughanmi, Amaury Pinéda



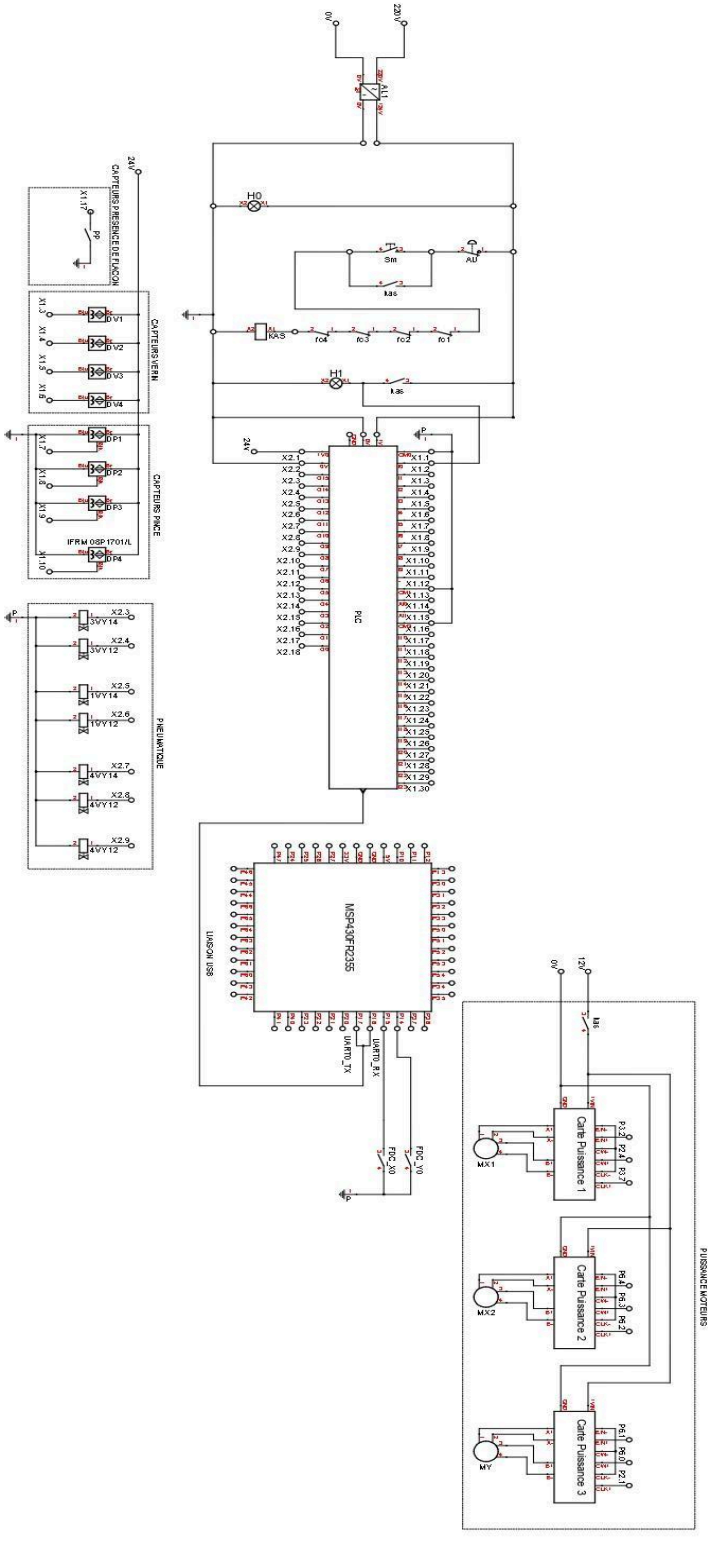


Douae Choubri, Reda El Madani, Salma Letrach, Ayoub Bouquennouche, Souad Ait Bellauali, Achraf Bouchemama, Roua Boughanmi, Amaury Pinéda

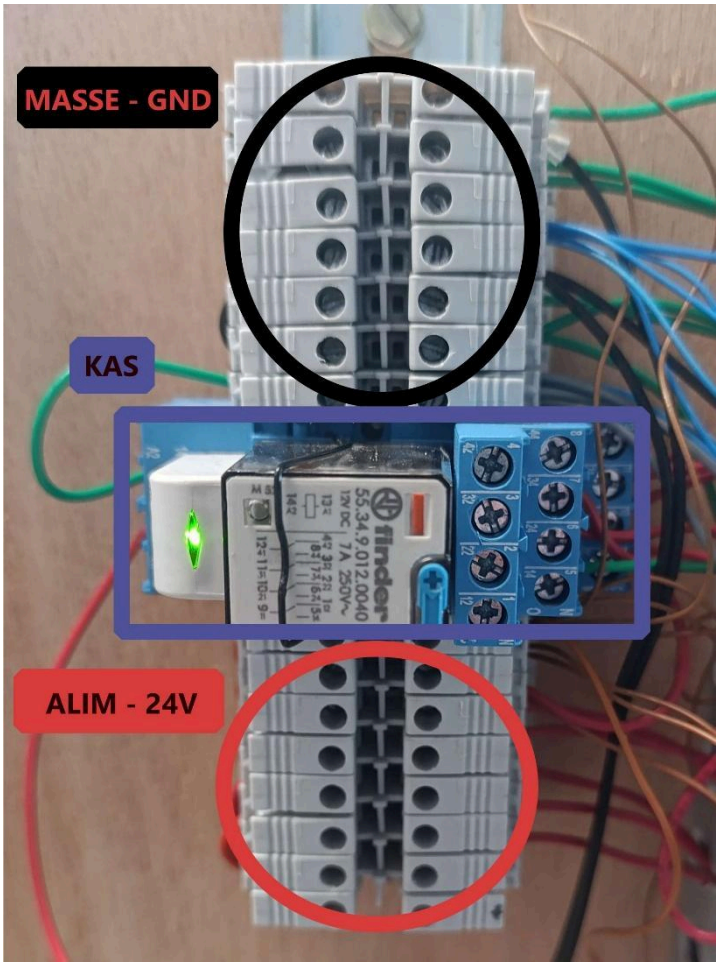
Partie électrique

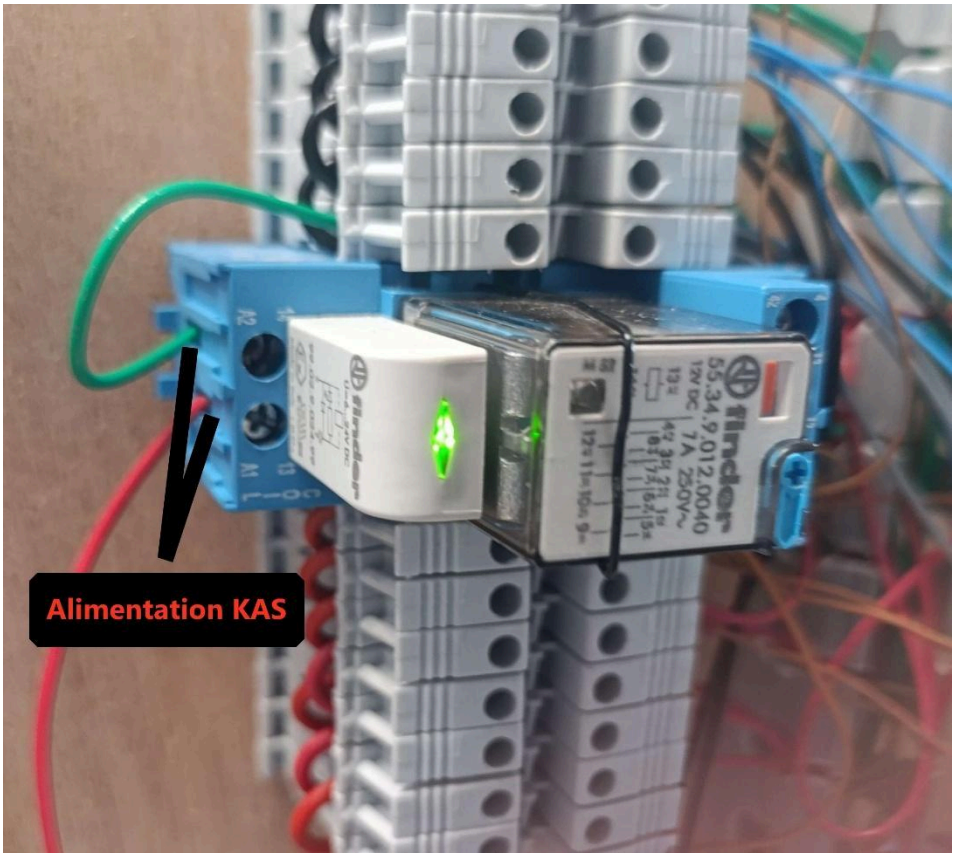
- Schéma (disponible dans les ressources)

La partie électrique du système concerne aussi bien la récupération des données capteurs, le câblage des sécurités et la mise sous tension du système que l'asservissement des moteurs. La coordination des différents éléments est assurée par l'automate (PLC sur le schéma)

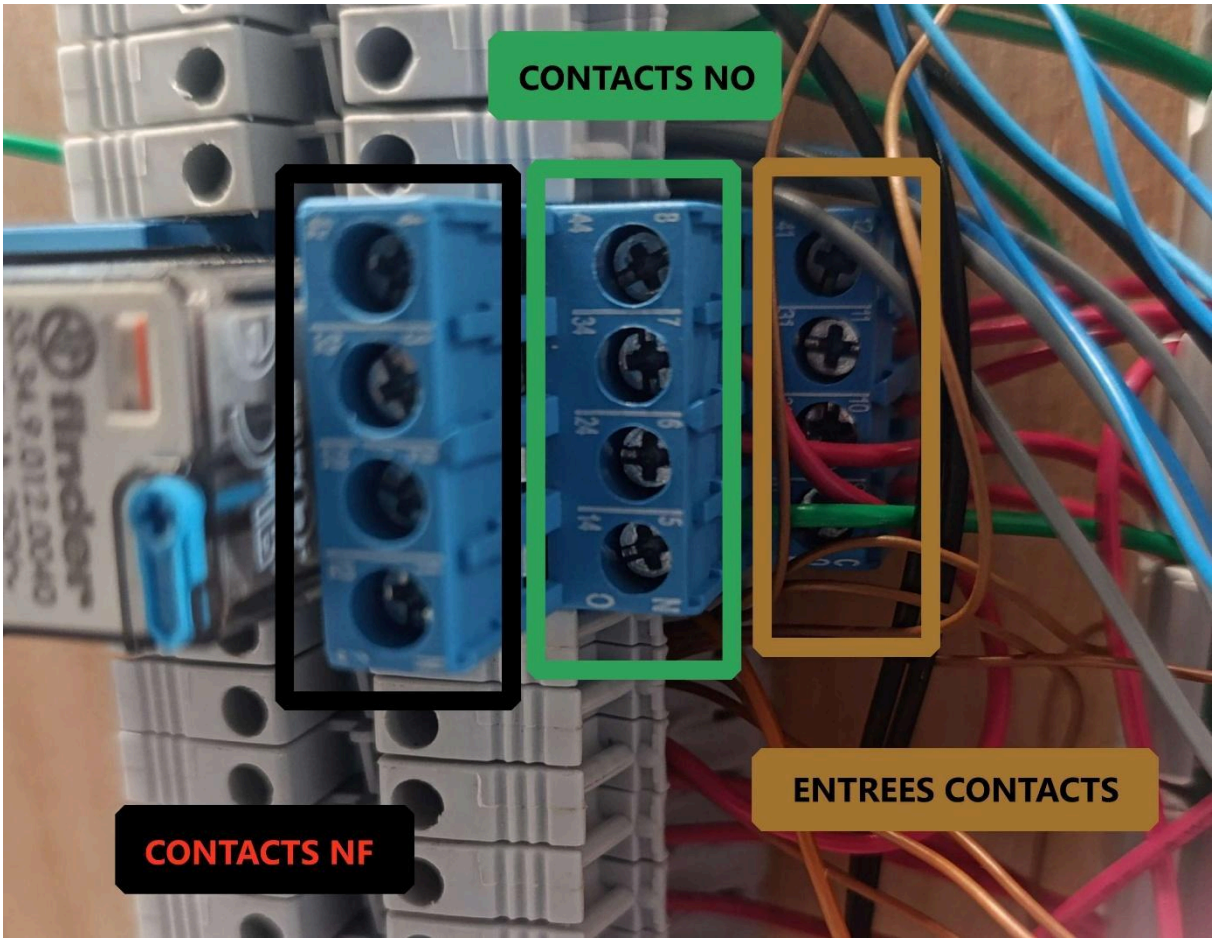


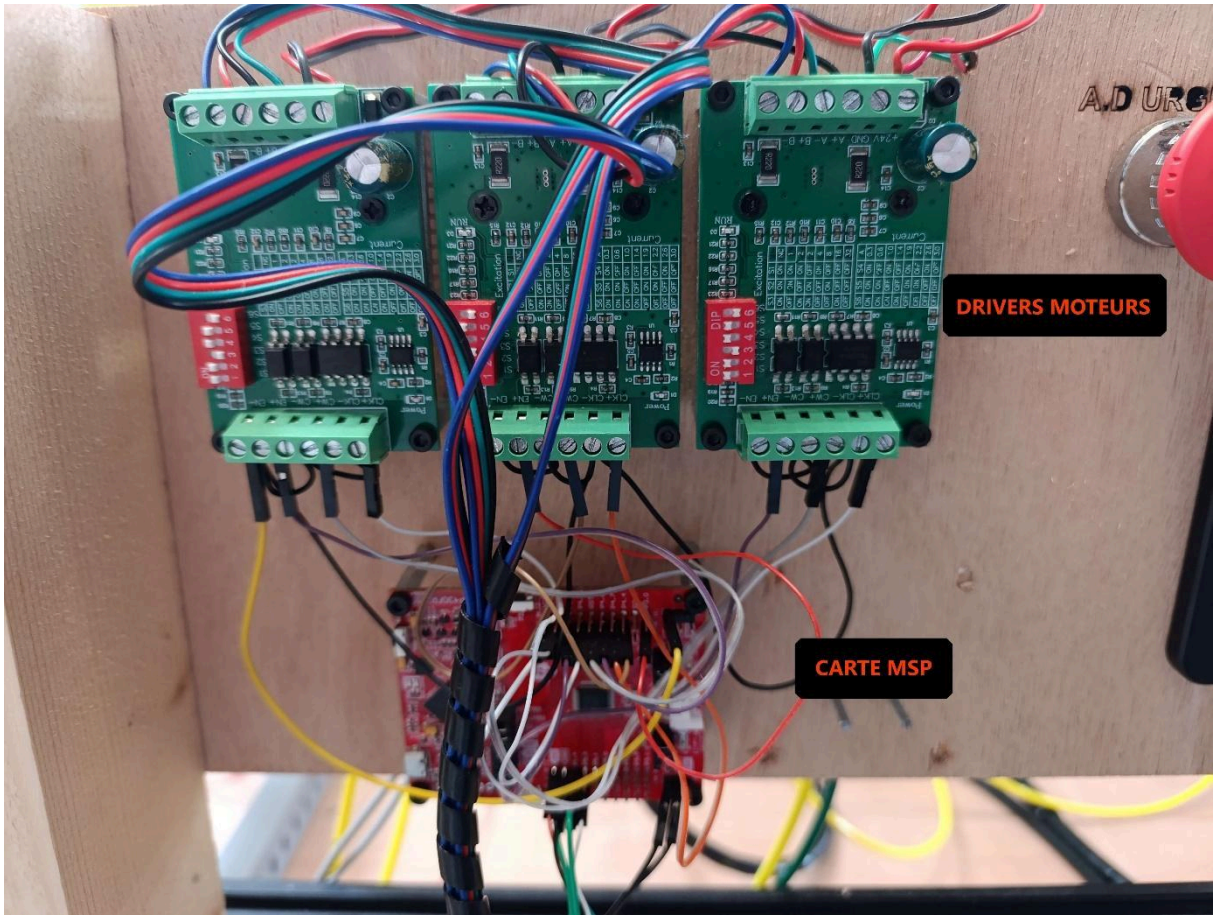
- Vue réelle





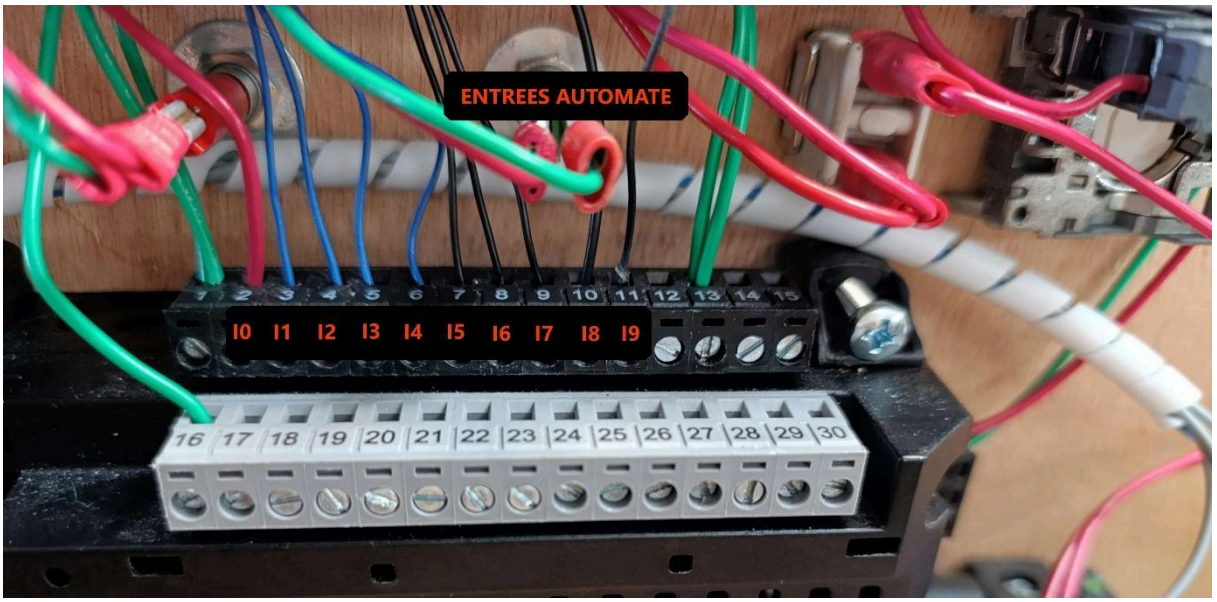
Douae Choubri, Reda El Madani, Salma Letrach, Ayoub Bouquennouche, Souad Ait Bellauali, Achraf Bouchemama, Roua Boughanmi, Amaury Pinéda





Douae Choubri, Reda El Madani, Salma Letrach, Ayoub Bouquennouche, Souad Ait Bellauali, Achraf Bouchemama, Roua Boughanmi, Amaury Pinéda

- Ports de l’automate :
  Correspondance entre les notations du schéma et les entrées-sorties réelles de l’automate (se référer au schéma électrique)



ENTREES		
Nom	Désignation	Numéro bornier
I0	Alimentation	X1.2
I1	Détecteur Vérin Boite sortir	X1.3
I2	Détecteur Vérin Boite entrer	X1.4
I3	Détecteur Vérin Pièce sortir	X1.5
I4	Détecteur Vérin Pièce entrer	X1.6
I5	Détecteur Pince haute	X1.7
I6	Détecteur Pince basse	X1.8
I7	Détecteur Pince 1	X1.9
I8	Détecteur Distance Pince	X1.10
I9	Détecteur Pince 2	X1.11

SORTIES			
Nom	Désignation	Numéro bornier	REF Bobine
O9	Ouvrir/Fermer Pince	X2.9	2VY14
O10	Fermer Vérin Pièce	X2.8	4VY12
O11	Ouvrir Vérin Pièce	X2.7	4VY14
O12	Descendre Pince	X2.6	1VY12
O13	Monter Pince	X2.5	1VY14
O14	Ouvrir Vérin Boite	X2.4	3VY12
O15	Fermer Vérin Boite	X2.3	3VY14

Douae Choubri, Reda El Madani, Salma Letrach, Ayoub Bouquennouche, Souad Ait Bellauali, Achraf Bouchemama, Roua Boughanmi, Amaury Pinéda

Partie informatique

- Analyse et traitement Code MSP ancien

Analyse et Améliorations du Code MSP430

Ayoub BOUQUENNOUCHE

Ce rapport présente une analyse détaillée du code `main.c` exécuté sur une carte MSP430. Le programme permet de piloter deux moteurs pas-à-pas (axes X et Y) à travers des commandes reçues via l'interface UART. Il gère aussi les capteurs de fin de course et le contrôle de la vitesse à l'aide d'un timer interne.

L'objectif de ce document est :

— d'expliquer le fonctionnement du code,

— d'identifier précisément la partie assurant la communication UART avec l'automate (PLC),

— de proposer les corrections et améliorations nécessaires pour fiabiliser le système.

1 Structure Générale du Programme

Le code s'organise en plusieurs blocs :

— **Initialisation** : horloges, GPIO, UART et timer.

— **Boucle principale** : vide, tout le fonctionnement repose sur les interruptions.

— **Gestion des moteurs** : générée par le timer TB0.

— **Communication UART** : réception, décodage et envoi de messages.

— **Capteurs de fin de course** : via interruptions GPIO.

2 Fonctionnement du Code

2.1 Initialisation du système

— Le **watchdog** est désactivé.

— Les **horloges** (ACLK, SMCLK, MCLK) sont configurées sur le DCO pour une fréquence stable de 1 MHz.

— Les **GPIO** sont configurées :

  — sorties pour les signaux moteurs (`STEP, DIR, EN`),

  — entrées avec résistances de tirage pour les capteurs de fin de course (`FCX, FCY`).

— Le module **UART** est configuré à 115200 bauds, 8 bits, sans parité, 1 bit de stop.

— Le **timer TB0** est initialisé en mode UP pour générer les impulsions moteur.

2.2 Structure de données principale

Le robot est décrit par une structure :

```
typedef struct {
    int x;    // Position X (mm)
    int y;    // Position Y (mm)
    int v;    // Vitesse (mm/s)
    char mode; // Mode de mouvement
} Robot;
```

Cette structure contient la position courante, la vitesse et le mode de déplacement.

3 Communication UART avec l'Automate

La communication entre le MSP430 et l'automate s'effectue via UART et repose sur trois composants clés :

1. **Réception** des commandes texte depuis le PLC.
2. **Interprétation** des commandes reçues.
3. **Envoi** d'accusés de réception et de messages d'état.

3.1 Variables principales liées à l'UART

— `TX_Buffer, RX_Buffer` : stockent les messages à envoyer et à recevoir.

— `tx_index, rx_index` : indices de progression dans les buffers.

— `command_flag` : indique si une commande complète est détectée.

3.2 Réception et traitement des commandes

La réception se fait dans l'interruption UART :

```
#pragma vector = EUSCI_A1_VECTOR
__interrupt void ISR_EUSCI_A1(void) {
    switch(__even_in_range(UCA1IV, USCI_UART_UCTXCPTIFG)) {
        case USCI_UART_UCRXIFG:
            char RX_received_char = UCA1RXBUF;
            ...
            switch(RX_received_char) {
                case MSG_START_CHAR: command_flag = 1; rx_index = 0; break;
                case MSG_END_CHAR:
                case '\n':
                case '\r':
                    if(command_flag){
                        RX_Buffer[rx_index] = '\0';
                        get_command(RX_Buffer);
                        command_flag = 0;
                    }
                    break;
```

```
                default:
                    RX_Buffer[rx_index++] = RX_received_char;
            }
    }
}
```

Chaque caractère reçu est ajouté au buffer jusqu'à détection du caractère de fin de message. Une fois complet, le message est transmis à la fonction `get_command()`.

3.3 Décodage et exécution

```
void get_command(const char *command) {
    switch(command[0]) {
        case 'P': set_position_command(command, &robot); break;
        case 'V': set_speed_command(command, &robot); break;
        case 'H': home_position_command(command, &robot); break;
        default: uart_put_string("ERROR - Invalid Command"); break;
    }
}
```

Les formats de commande sont :

- `P<x>-<y>` : position en millimètres.
- `V<speed>` : vitesse en mm/s.
- `H` : retour à la position d'origine.

3.4 Envoi de réponses vers l'automate

Les messages de retour sont envoyés via la fonction :

```
void uart_put_string(char *string) {
    strcpy(TX_Buffer, string);
    tx_length = strlen(TX_Buffer);
    EUSCI_A_UART_transmitData(EUSCI_A1_BASE, TX_Buffer[0]);
}
```

Les réponses possibles sont :

- `<P1>` : commande de position acceptée.
- `<S1>` : vitesse configurée avec succès.
- `<S0>` : erreur de vitesse.
- `ERROR - Invalid Command` : commande non reconnue.

4 Analyse de la Partie UART

Cette section résume les points clés de la communication avec le PLC :

- **Réception** : caractère par caractère, assemblage dans `RX_Buffer`.
- **Analyse** : détection des caractères de début et fin de message.
- **Décodage** : sélection de la commande selon son identifiant (P, V, H).
- **Exécution** : actions sur les moteurs et mise à jour des variables.
- **Réponse** : envoi d'un message d'accusé de réception.

5 Améliorations et Corrections Proposées

5.1 1. Fiabilité et Sécurité de la Communication UART

- **Validation du format de commande** : vérifier les paramètres reçus via `sscanf()` avant exécution.
- **Ajout d'un checksum** : pour détecter les erreurs de transmission.
- **Gestion du débordement du buffer RX** : envoyer un message d'erreur comme `<E_BUF>`.
- **Timeout de réception** : réinitialiser `command_flag` si aucun caractère reçu pendant un certain temps.

5.2 2. Amélioration du Dialogue avec l'Automate

- **Standardiser les messages** :
  - `<ACK_P>` : position reçue.
  - `<ACK_S>` : vitesse reçue.
  - `<DONE>` : mouvement terminé.
  - `<BUSY>` : mouvement en cours.
  - `<ERR>` : erreur générique.
- **Confirmer la fin du mouvement** : envoyer `<DONE>` depuis l'interruption du timer.
- **Refuser les commandes pendant un mouvement** : utiliser un flag `isMoving` pour bloquer les nouvelles requêtes.

5.3 3. Structure et Maintenabilité

- Factoriser le code UART A0/A1 en une fonction unique.
- Ajouter des journaux UART internes pour suivre l'activité.
- Documenter les fonctions et les messages échangés.

5.4 4. Contrôle Moteur et Synchronisation

- Ajouter un profil d'accélération/décélération (vitesse progressive).
- Synchroniser les axes X et Y pour les trajectoires droites.
- Contrôler la largeur d'impulsion STEP selon les spécifications matérielles.

- Ports, coté IHM

Correspondance entre les entrées/sorties et les écrans de l'IHM

I. Liste des entrées/sorties et leurs fonctions

A. Liste des entrées

Entrée	*Fonction d'entrée*
I0	Alimentation active
I1	Détecteur vérin Boîte sortir
I2	Détecteur vérin Boîte entrer
I3	Détecteur vérin Pièce sortir
I4	Détecteur vérin Pièce entrer
I5	Détecteur vérin haute
I6	Détecteur vérin basse
I7	Détecteur pince 1
I8	Détecteur distance pince (capteur)
I9	Détecteur pince 2
I10	Prise pièce

B. Liste des sorties

Sortie	*Signal de sortie*	
O0	gripper Piston Opened Out	**Non utilisés**
O1	gripper Piston Closed Out	
O2	gripper Piston Up Out	
O3	gripper Piston Down Out	
O9	Prendre/Lâcher Pièce	
O10	Fermer Verin Pièce	
O11	Ouvrir Verin Pièce	
O12	Descendre Pièce	
O13	Monter Pièce	
O14	Ouvrir Verin Boîte	
O15	Fermer Verin Boîte	

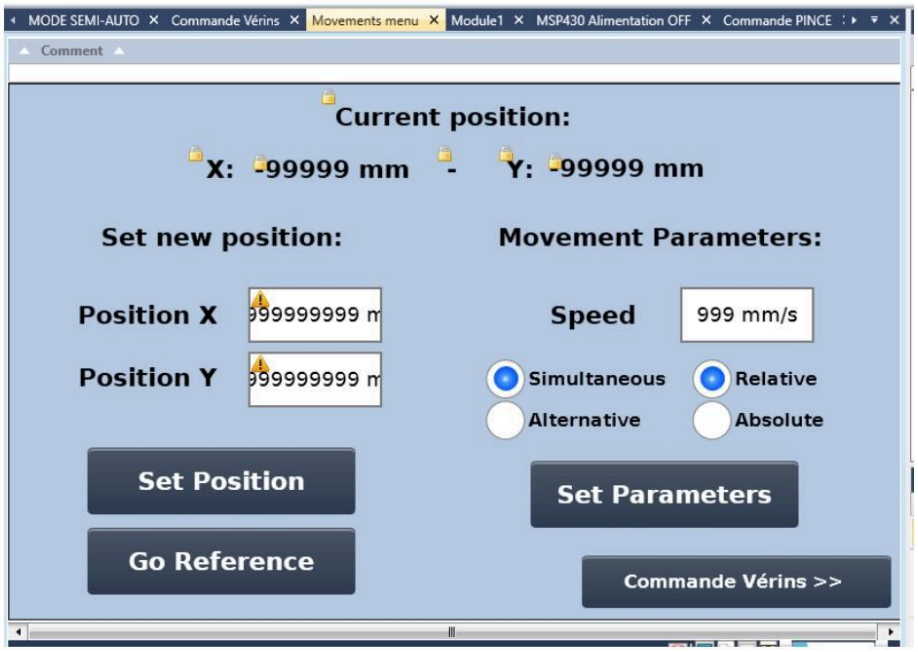
N.B : Statuts en format HEX

Valeur Hex	*Signification*
0x00	OK
0x08	Short circuit
0x10	Supply voltage low or missing
0xFFFF	On Board IO not initialized

II. Côté IHM

A. **Movements Menu :** Page d'accueil / Commande Moteurs

L'écran **« Movements Menu »** correspond à la page d'accueil de la commande des moteurs. Il permet de définir la nouvelle position des axes **X** et **Y**, ainsi que les paramètres de mouvement (vitesse, mode de déplacement, etc.).



-
-

Élément IHM	*Tag / Variable associée*	*Type d'action*	*Détails / Trigger et Liaison Ladder*	*Fonction associée (MSP430)*
Bouton "***Set Position***"	Touch Enable -> Enable Bit	Press : Set Bit Release : Reset Bit	Status Set Position	MSP430 Set Position
Bouton "***Set Parameters***"	Touch Enable -> Enable Bit	Press : Set Bit Release : Reset Bit	Status Set Parameters	MSP430 Set Parameters
Bouton "***Go Reference***"	Touch Enable -> Enable Bit	Press : Set Bit Release : Reset Bit	Status Go Reference	MSP430 Go to Reference

Pour le bouton "***Commande Vérins>>***", il déclenche '***Load Screen***' c'est à dire qu'il redirige vers l'écran de la page "***Commande vérins***" qui sera présenté dans la section suivante.

Il y a aussi 2 boutons radio : Simultaneous/Alternative(où 0 représente mode simultanée et 1 alternatif) et Relative/Absolute (où 0 représente mode relatif et 1 absolu).

B. Commande Vérins

L'écran « Commande des vérins » permet de piloter les vérins situés du côté *boîte* et du côté *pièce*. Chaque bouton de commande agit sur une sortie numérique du MSP430, tandis que les états des vérins sont surveillés via les entrées numériques correspondantes.

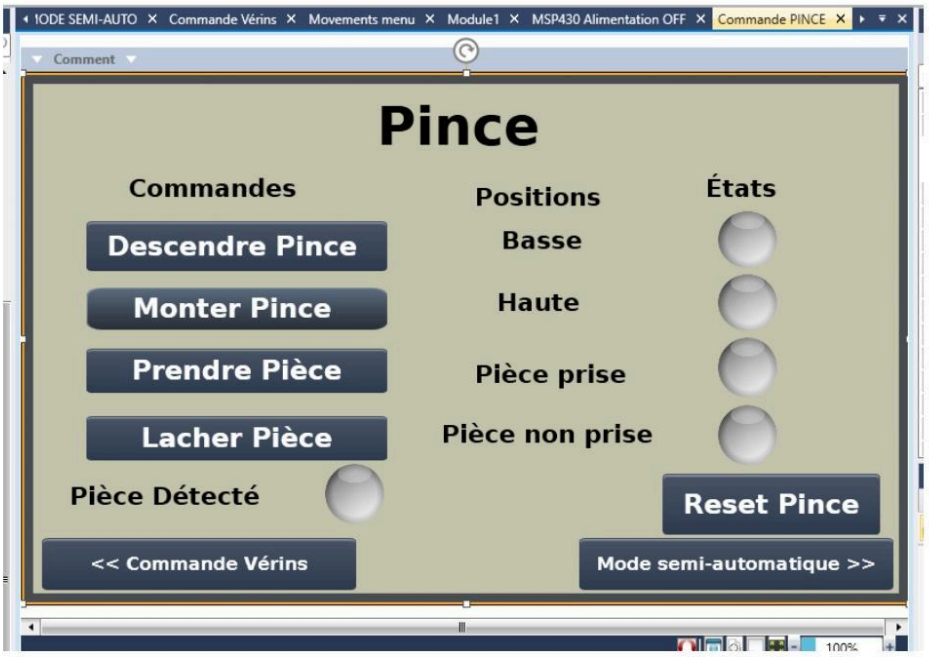
Élément IHM	*Action*	*Signal associé*	*Sortie associée*
Sortir Vérin (côté Boîte)	Reset Bit	Boîte Out	Digital Output 14 (O_14)
	Set Bit	Boîte In	Digital Output 15 (O_15)
Entrer Vérin (côté Boîte)	Toggle Bit	Boîte In	Digital Output 15 (O_15)
	Set Bit	Boîte Out	Digital Output 14 (O_14)
RAZ Distributeurs	Reset Bit	Boîte In	Digital Output 15 (O_15)
	Reset Bit	Boîte Out	Digital Output 14 (O_14)
	Reset Bit	Pièce In	Digital Output 11 (O_11)
	Reset Bit	Pièce Out	Digital Output 10 (O_10)
Sortir Vérin (côté Pièce)	Reset Bit	Pièce Out	Digital Output 10 (O_10)
	Set Bit	Pièce In	Digital Output 11 (O_11)
Entrer Vérin (côté Pièce)	Reset Bit	Pièce In	Digital Output 11 (O_11)
	Set Bit	Pièce Out	Digital Output 10 (O_10)

Élément IHM	*Entrée associée*
État 1 côté Boîte	Digital Input 1 (I_1)
État 2 côté Boîte	Digital Input 2 (I_2)
État 1 côté Pièce	Digital Input 3 (I_3)
État 2 côté Pièce	Digital Input 4 (I_4)

Pour le bouton "***Commande Moteurs***>>", il déclenche '***Load Screen***' c'est à dire qu'il redirige vers l'écran de la page "***Current position***" présenté précédemment. Même chose pour le bouton "***Commande Pince***>>", il déclenche '***Load Screen***' c'est à dire qu'il redirige vers l'écran de la page "***Pince***" qui sera présenté dans la section suivante.

C. Commande Pince

L'écran « Commande Pince » permet de contrôler les mouvements verticaux de la pince, ainsi que la prise et la libération des pièces. Il offre également un retour d'état indiquant la position de la pince (haute ou basse) et la présence ou non d'une pièce détectée.



Élément IHM	*Action*	*Signal associé*	*Sortie associée*
Descendre Pince	Set Bit	Pince Up	Digital Output 12 (O_12)
	Reset Bit	Pince Down	Digital Output 13 (O_13)
Monter Pince	Set Bit	Pince Down	Digital Output 13 (O_13)
	Reset Bit	Pince Up	Digital Output 12 (O_12)
Prendre Pièce	Reset Bit	Grab	Digital Output 9 (O_9)
Lâcher Pièce	Set Bit	Grab	Digital Output 9 (O_9)
Reset Pince	Reset Bit	Pince Down	Digital Output 13 (O_13)
	Reset Bit	Pince Up	Digital Output 12 (O_12)
	Reset Bit	Grab	Digital Output 9 (O_9)

Documentation du code – Liaison automate et code MSP430

1. Introduction

Cette documentation présente le fonctionnement de la communication entre l'automate **Unitronics** et la carte **MSP430**, ainsi que la structure du code côté microcontrôleur.

L'objectif est d'assurer une correspondance claire entre les **messages transmis par UniLogic (automate)** et les **fonctions exécutées sur la carte MSP430**, notamment pour le contrôle des moteurs, la mise en position initiale et la gestion des vitesses.

2. Structure générale et communication

Les différentes commandes envoyées depuis l'automate sont reçues par le MSP430 via le port **UART**. Chaque message est interprété selon son identifiant initial (ex. *P*, *V*, *H*), permettant de déterminer la fonction à exécuter.

Exemple de structure générale d'un message :

```
<Px-y>     // Commande de position
<Vx-y-z>   // Commande de vitesse + mode
<H>        // Home Position
```

Ces messages sont reçus dans une interruption UART, stockés dans un buffer puis analysés dans la fonction `get_command()` qui oriente vers la fonction correspondante :

```c
void get_command(const char *command) {
    switch(command[0]){
        case 'P': set_position_command(command, &robot); break;
        case 'V': set_speed_command(command, &robot); break;
        case 'H': home_position_command(command, &robot); break;
        default: uart_put_string("ERROR - Invalid Command"); break;
    }
}
```

Douae Choubri, Reda El Madani, Salma Letrach, Ayoub Bouquennouche, Souad Ait Bellauali, Achraf Bouchemama, Roua Boughanmi, Amaury Pinéda

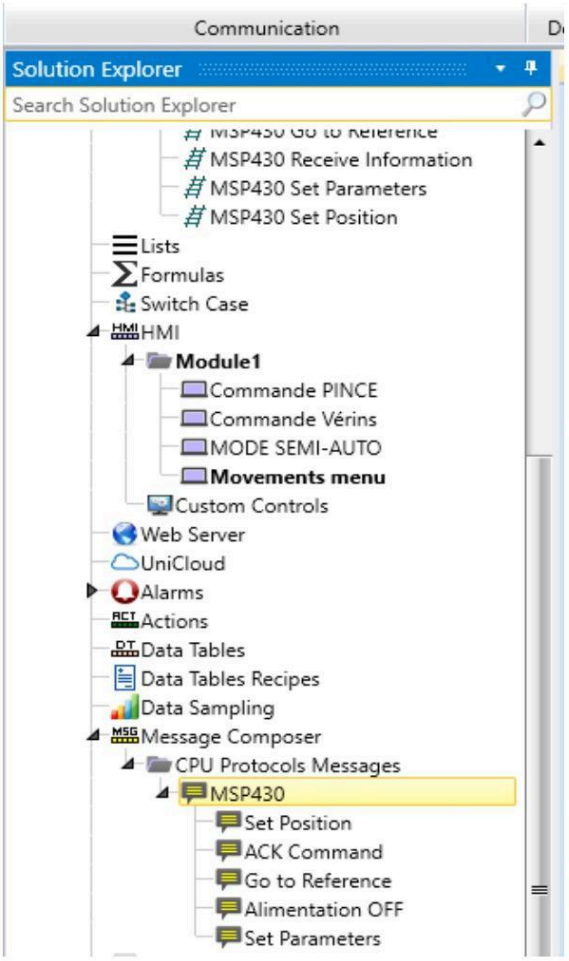


Figure 1 – Organisation des messages dans le Solution Explorer

Cette figure montre la structure du projet UniLogic.

Dans l'arborescence du Solution Explorer, les messages de communication sont regroupés dans ***Message Composer → CPU Protocols Messages → MSP430***.

Chaque message correspond à une fonction du code MSP430 (Set Position, ACK Command, Go to Reference, Set Parameters,..). Cette hiérarchie facilite la communication entre l'automate et le microcontrôleur.

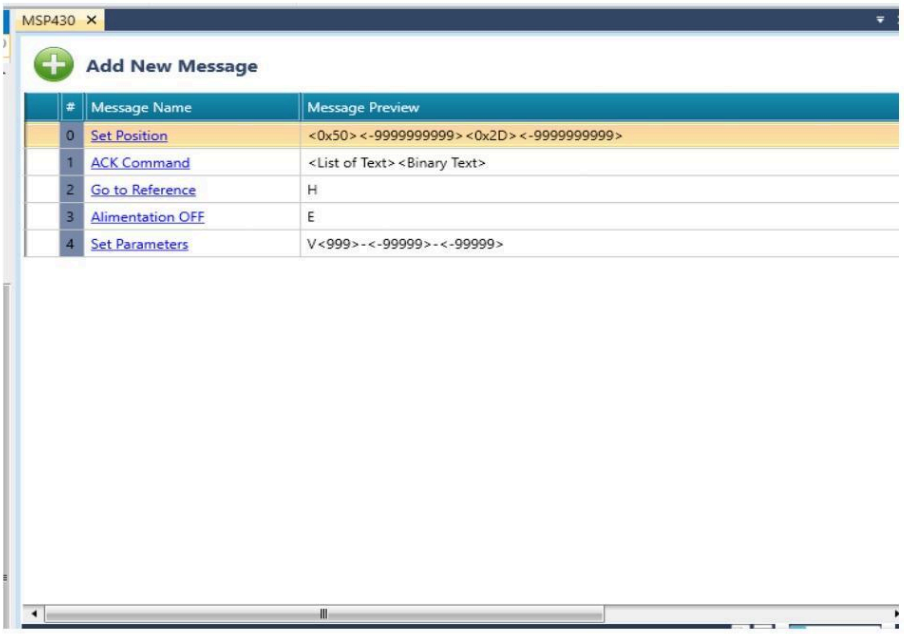


Figure 2 – Liste des messages MSP430 dans UniLogic

Ici, chaque message est défini avec son format d'envoi et sa prévisualisation.
Par exemple, Set Position utilise la structure `<Px-y>`, Set Parameters correspond à `V<999>-<9999>-<9999>`, etc.
Ces formats sont interprétés côté MSP430 dans la fonction `get_command()`, permettant l'exécution de la commande correspondante.

3. Commandes principales et code associé

3.1 Set Position

Cette commande positionne les moteurs X et Y à une coordonnée absolue ou relative.
Le message ***'P%d-%d'*** contient les valeurs X et Y.

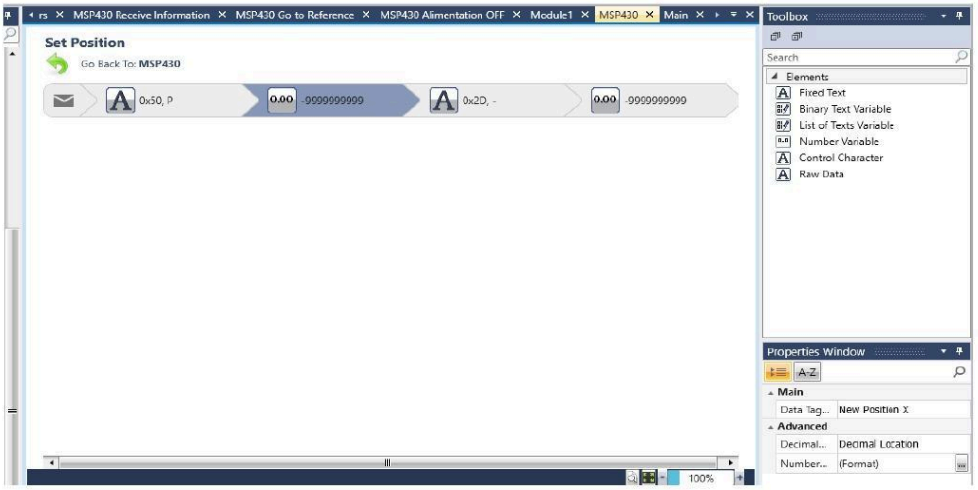


Figure 3 – Liaison de la commande Set Position dans UniLogic

Code associé :

```
void set_position_command(const char *command, Robot *robot) {
    // Lecture de la commande reçue et extraction des coordonnées X et Y
    sscanf(command, "P%d-%d", &(robot->x), &(robot->y));

    // Conversion du déplacement de mm en pas moteur
    int steps_x = mmToStep(robot->x);
    int steps_y = mmToStep(robot->y);

    // Détermination du sens de rotation selon le signe du déplacement
    if (steps_x > 0) {
        GPIO_setOutputLowOnPin(DIR_MX1_PORT, DIR_MX1_PIN);   // Mouvement en
avant (axe X)
        GPIO_setOutputLowOnPin(DIR_MX2_PORT, DIR_MX2_PIN);   // Mouvement en
avant (axe X)
    } else {
        GPIO_setOutputHighOnPin(DIR_MX1_PORT, DIR_MX1_PIN);  // Mouvement en
arrière (axe X)
        GPIO_setOutputHighOnPin(DIR_MX2_PORT, DIR_MX2_PIN);  // Mouvement en
arrière (axe X)
    }

    if (steps_y > 0) {
        GPIO_setOutputHighOnPin(DIR_MY_PORT, DIR_MY_PIN);    // Mouvement en
avant (axe Y)
    } else {
        GPIO_setOutputLowOnPin(DIR_MY_PORT, DIR_MY_PIN);     // Mouvement en
```

```
arrière (axe Y)
    }

    abs_steps_x = steps_x >=0 ? steps_x : -steps_x;
    abs_steps_y = steps_y >=0 ? steps_y : -steps_y;

    // Activation des moteurs et lancement du timer
    enableMotors(1, 1);
    timer_on();

    // Envoi d'un message d'accusé de réception (ACK) à l'automate
    uart_put_string("<P1>"); // ACK de confirmation
}
```

3.2 Home Position

Cette commande permet de renvoyer les moteurs en position initiale (définie par les capteurs de fin de course).

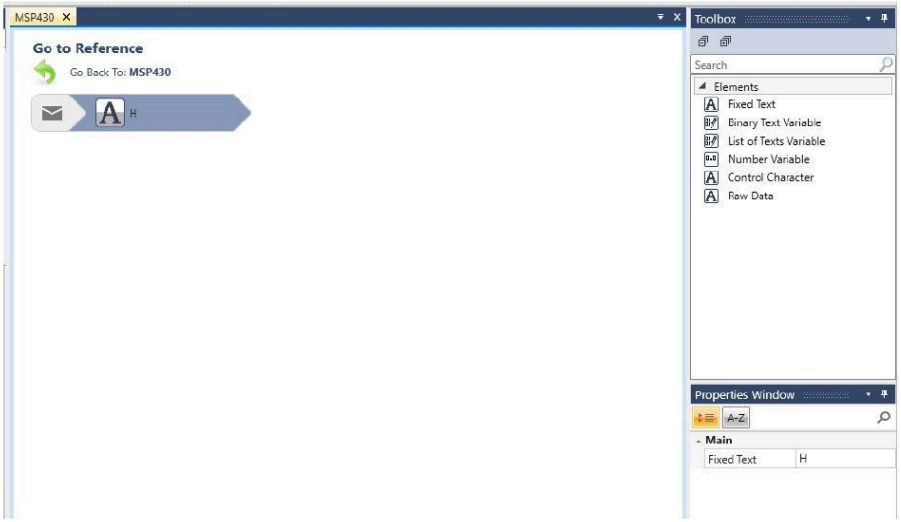


Figure 4 – Commande Home Position et retour capteurs

Code associé :

```
void home_position_command(const char *command, Robot *robot){
    HomePositionXY(1,1);
    abs_steps_x = 1000; abs_steps_y = 1000;
    enableMotors(1,1);
    timer_on();
}
```

La fonction `HomePositionXY()` déclenche le mouvement en arrière jusqu'à détection des capteurs :

```
void HomePositionXY(int flagX, int flagY) {
    if(flagX){
      GPIO_setOutputHighOnPin(DIR_MX1_PORT, DIR_MX1_PIN); //en arrière
      GPIO_setOutputHighOnPin(DIR_MX2_PORT, DIR_MX2_PIN); //en arrière
      enableMotorsX(1);
    }
    if(flagY){
      GPIO_setOutputLowOnPin(DIR_MY_PORT, DIR_MY_PIN); //en arrière
      enableMotorsY(1);
    }
}
```

N.B : cette fonction est à modifier selon la demande du professeur. Le calcul des positions sera effectué côté automate : le code MSP430 recevra directement les valeurs réelles de x et y (enregistrées à chaque déplacement). Ces valeurs permettront de revenir à la position d'origine réelle.

Par exemple, si l'axe x s'est déplacé successivement de 50, puis 70, puis 5 mm, la valeur totale reçue sera 125. Le retour à la source consistera donc à reculer de 125 pas (ou à convertir directement la valeur reçue en pas si elle est déjà exprimée en mm).

3.3 Set Speed Command

Cette commande gère la vitesse des moteurs ainsi que deux modes :
- **Simultaneous / Alternative**
- **Relative / Absolute**

Le message ***'V%d-%d-%d'*** contient respectivement :
- la vitesse,
- le mode simultané/alternatif,
- le mode relatif/absolu.

Figure 5 – Commande Set Speed et structure du message

N.B. prob 1 :

Avant, il n'y avait **aucun appel à cette fonction** dans le Ladder, ce qui empêchait son déclenchement.

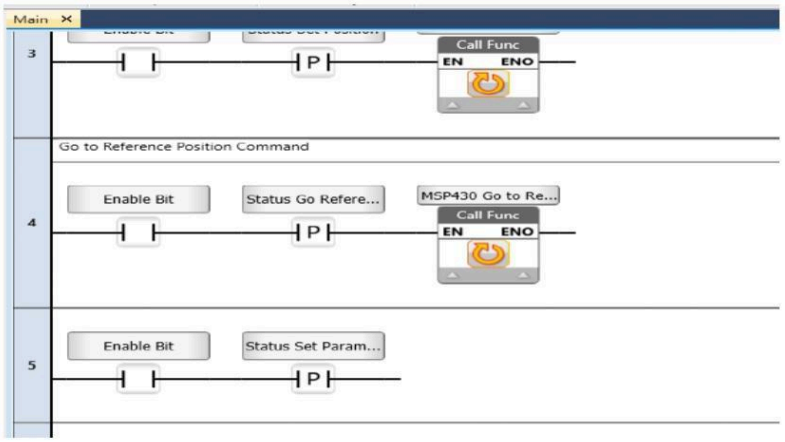


Figure 6 – Absence d'appel à la fonction Set Parameters

Après rectification, l'appel à la fonction a été ajouté dans le Ladder (*MSP430 Set Param*), ce qui permet désormais l'exécution correcte de la commande.

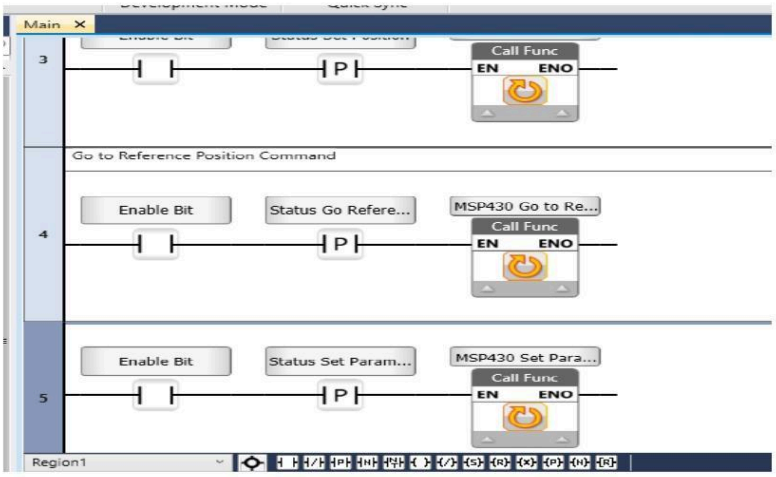


Figure 7 – Présence d'appel à la fonction Set Parameters

N.B. prob 2 :

De plus, au niveau du code, seule la **vitesse** était initialement récupérée, sans les modes de fonctionnement. Le code original était :

```
void set_speed_command(const char *command, Robot *robot){
    int commanded_speed;
    sscanf(command, "V%d", &commanded_speed);

    if(commanded_speed<= MAX_SPEED && commanded_speed >= MIN_SPEED){
        robot->v = commanded_speed;
        timer_off();
        TB0CCR0 = (MM_BY_TOUR*TIMER_FREQ)/(2*STEP_BY_TOUR*robot->v);
        uart_put_string("<S1>"); // ACK message from MSP430 board to PLC
    } else {
        uart_put_string("<S0>"); // ACK message from MSP430 board to PLC
    }
}
```

Après rectification (mais avec améliorations encore possibles) :

```
void set_speed_command(const char *command, Robot *robot) {
    int commanded_speed = 0;
    int mode_sim_alt = 0;
    int mode_rel_abs = 0;

    sscanf(command, "V%d-%d-%d", &commanded_speed, &mode_sim_alt,
&mode_rel_abs);

    if (commanded_speed <= MAX_SPEED && commanded_speed >= MIN_SPEED) {
        robot->v = commanded_speed;
        // → bit 1 = simultaneous/alternative, bit 0 = relative/absolute
        robot->mode = (mode_sim_alt << 1) | mode_rel_abs;

        // Stop and reconfigure timer
        timer_off();
        TB0CCR0 = (MM_BY_TOUR * TIMER_FREQ) / (2 * STEP_BY_TOUR * robot->v);

        /* À ajouter : gestion du comportement selon le mode :
        if (mode_sim_alt == 0) {
            // Simultaneous mode → X et Y bougent ensemble
        } else {
            // Alternative mode → X puis Y
        }

        if (mode_rel_abs == 0) {
            // Relative mode → déplacement par rapport à la position
actuelle
        } else {
```

```
            // Absolute mode → déplacement vers la position donnée
        }*/

        uart_put_string("<S1>"); // ACK OK
    } else {
        uart_put_string("<S0>"); // Erreur : vitesse hors limites
    }
}
```

3.4 ISR Fin de course (PORT1_VECTOR)

Cette interruption permet d'arrêter automatiquement les moteurs lorsqu'un capteur de fin de course est activé.

Code associé :

```
#pragma vector = PORT1_VECTOR
__interrupt void ISR_CAPTEURS_FDC(void){
    if(P1IFG & FCX_PIN){ enableMotorsX(0); robot.x = 0; }
    if(P1IFG & FCY_PIN){ enableMotorsY(0); robot.y = 0; }
    P1IFG &= ~(FCX_PIN | FCY_PIN);
}
```

4. Problèmes rencontrés et améliorations

1. **Problème de synchronisation initiale** : certaines commandes (*Set Parameters*) ne se déclenchaient pas car la fonction n'était pas encore appelée dans le code principal.
   Correction : ajout de l'appel à la fonction correspondante et gestion des modes via `sscanf()`.

2. **Limitation du calcul de position pour retour à l'origine** : la version initiale ne tenait pas compte de l'enregistrement cumulatif des positions X/Y.
   Correction prévue : récupérer directement la valeur totale X/Y de l'automate et effectuer le retour en pas selon cette valeur.

3. **Validation de la communication UART** : ajout de messages de confirmation *ACK* (`<S1>`, `<P1>`) pour vérifier la réception correcte des ordres.

5. Conclusion

Cette documentation prouve la mise en place d'une communication fonctionnelle entre l'automate **Unitronics** et la carte **MSP430**.
Les prochaines évolutions concerneront les issues liées au retour à l'origine ainsi que la gestion dynamique des positions absolues/relatives et l'intégration de modes de déplacement simultanés/alternatifs plus avancés.

Prise en main du logiciel UniLogic – Configuration matérielle, création d'alias et liaison Ladder/IHM

I. Objectif du document

Ce document est un guide pratique pour apprendre à configurer, programmer et tester un automate **Unitronics** à l'aide du logiciel **UniLogic**.

L'objectif est d'expliquer les étapes essentielles pour : - configurer la partie **matérielle et I/O** ; - créer et nommer les **alias** pour les entrées/sorties ; - relier les signaux matériels au programme **Ladder** ; - associer les actions dans l'**IHM** ; - et vérifier la cohérence entre les différents éléments du projet.

II. Présentation générale d'UniLogic

UniLogic est l'environnement de développement utilisé pour programmer les automates **Unitronics**. Il intègre à la fois :

- ➔ la **configuration matérielle (I/O, modules, alimentation)** ;
- ➔ la **programmation Ladder** ;
- ➔ la conception de l'**IHM (Interface Homme-Machine)** ;
- ➔ et la gestion des **communications et diagnostics**.

L'interface est composée de trois zones principales :

1. **Solution Explorer (à gauche)** : navigation entre les éléments du projet (Ladder, IHM, communications…).
2. **Espace de travail (au centre)** : affichage du Ladder, des écrans IHM ou des modules.
3. **Onglet I/O et propriétés (en bas)** : configuration des entrées/sorties, paramètres et alias.

Douae Choubri, Reda El Madani, Salma Letrach, Ayoub Bouquennouche, Souad Ait Bellauali, Achraf Bouchemama, Roua Boughanmi, Amaury Pinéda

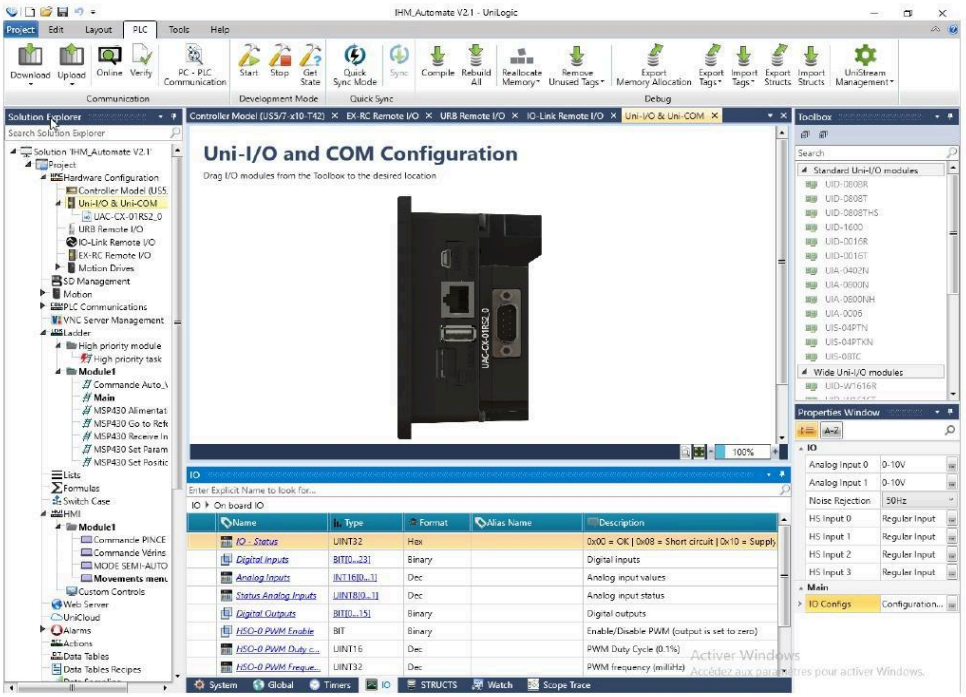


Figure 1 – Interface générale du logiciel UniLogic

III. Configuration matérielle

A. Définition du matériel

Dans un nouveau projet UniLogic :

1. Sélectionner le modèle de contrôleur (ex. **US5/7-x10-T42**).
2. Ajouter les modules **Uni-I/O** et **Uni-COM** nécessaires à la configuration.
3. Relier les modules modules d'extension au contrôleur via le bus **UniCAN**.

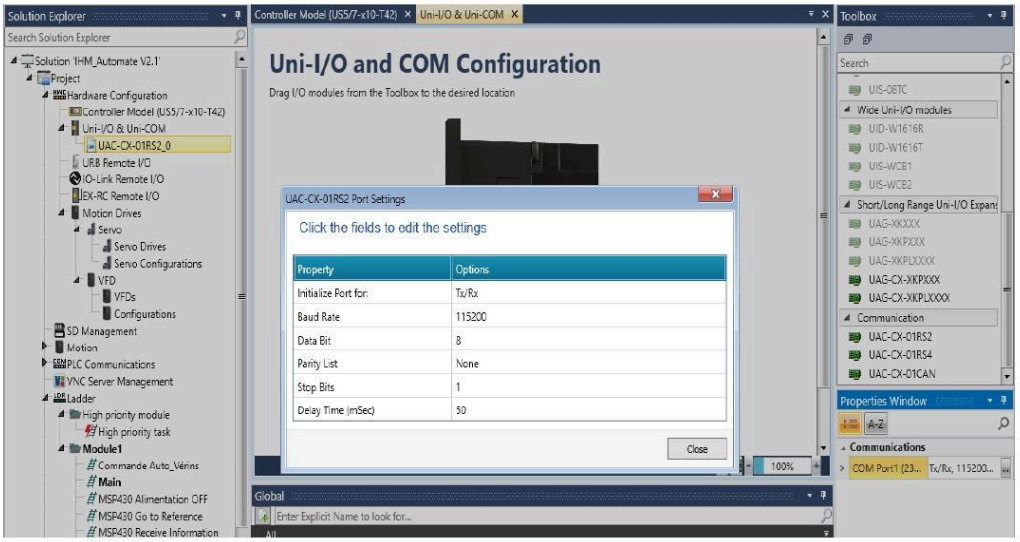


Figure 2 – Configuration matérielle de l'automate et des modules I/O

Cette étape permet d'identifier les modules physiques connectés et leurs capacités d'entrées/sorties numériques et analogiques.

IV. Création et gestion des entrées/sorties

A. Accès à la table I/O

Les entrées et sorties sont configurées dans la fenêtre **I/O and COM Configuration**, accessible depuis la barre latérale gauche ou l'onglet inférieur **I/O**.

Chaque ligne correspond à une entrée ou sortie physique (ex. *Digital Output 0, 1, 2…*).

B. Création d'un alias

Pour faciliter la lecture du programme, chaque E/S doit être renommée par un **Alias Name** explicite, ce qui permet d'identifier facilement chaque signal dans le Ladder et l'IHM.

Procédure pour créer un alias :

1. Ouvrir l'onglet **I/O**.

2. Cliquer sur la cellule de la colonne **Alias Name** correspondant à la sortie souhaitée (ex : *Digital Output 0*).
3. Saisir un nom logique clair, comme :

   - `OUVRIR_VERIN_PIECE`
   - `FERMER_VERIN_PIECE`
   - `DESCENDRE_PINCE`

4. Valider la saisie avec **Entrée**.

L'alias est automatiquement ajouté au projet et pourra être réutilisé dans le Ladder ou dans l'IHM.

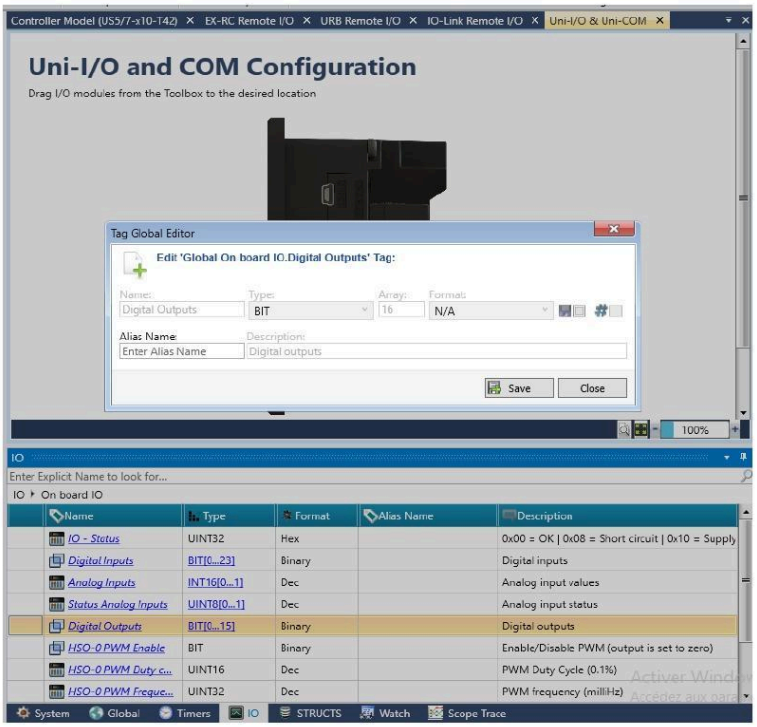


Figure 3 – Création et affectation d'un alias à une sortie

V. Liaison Ladder et IHM

A. Création des tags logiques

Les **tags** sont des variables logiques servant à faire le lien entre le matériel, le Ladder et l'IHM.
Pour associer un tag à un élément IHM (par exemple un bouton) :

- ➔ Sélectionner l'élément concerné dans l'écran de conception IHM.
- ➔ Ouvrir l'onglet **Actions** puis cliquer sur les **trois points** à droite de *Collection*.
- ➔ Dans la fenêtre **Element Actions**, choisir **Add New Action** et sélectionner le tag à relier.

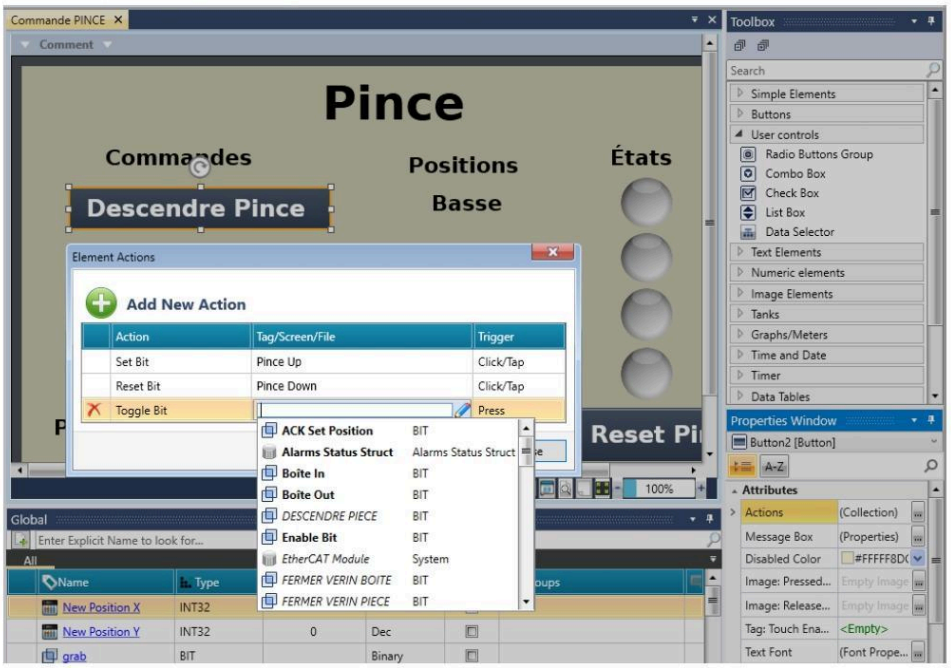


Figure 4 – Affectation des tags aux éléments IHM

B. Liaison avec les actions IHM

Chaque élément IHM (bouton, voyant, indicateur) peut être relié à un tag via la fenêtre **Element Actions**.

Exemple : le bouton *Descendre Pince* est relié aux tags `Pince_Down et Pince_Up` qui commandent respectivement les sorties **O_13** et **O_12**.

Étapes :

1. Sélectionner le bouton dans l'écran IHM ;
2. Ouvrir **Element Actions** ;
3. Associer l'action *Set Bit* ou *Reset Bit* au tag concerné.

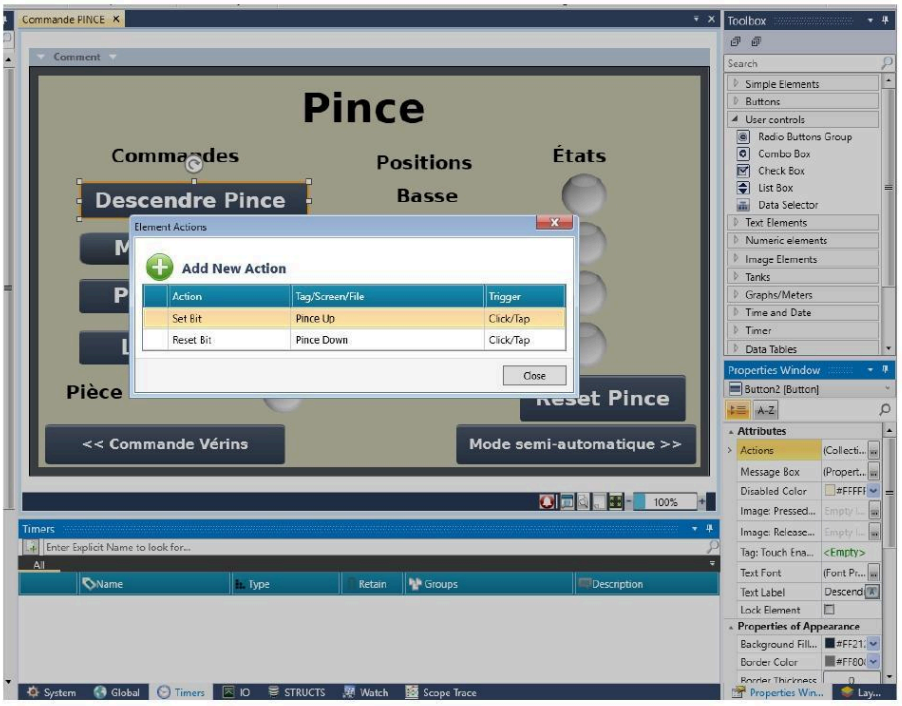


Figure 5 – Association d'un bouton IHM à un tag logique

C. Vérification dans le Ladder

Dans le **Ladder**, les actions de la pince ou des vérins utilisent directement les tags créés.
Exemple :
- `Pince_Up` → Descendre la pièce ;
- `Pince_Down` → Monter la pièce.

Le lien entre IHM et Ladder peut être vérifié via un clic droit sur le tag → **Find Tag Results**.

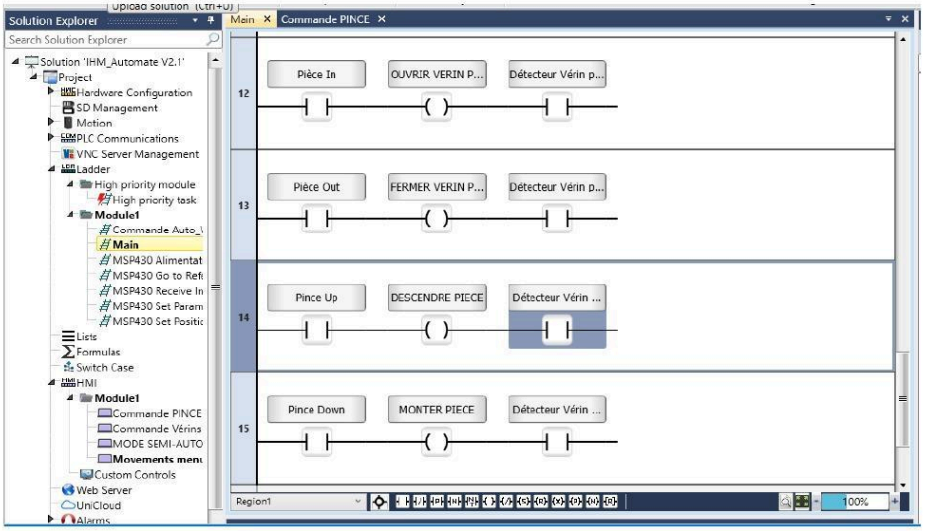


Figure 6 – Correspondance entre Ladder et IHM

Le lien entre Ladder et sortie physique peut être vérifié via un clic droit sur le tag → **Go to Definition**.

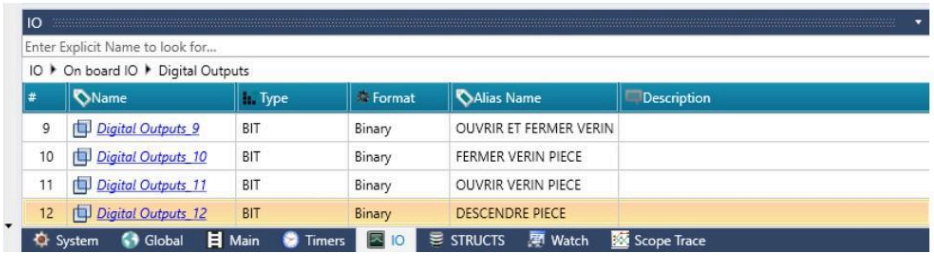


Figure 7 – Correspondance entre Ladder et sorties physiques

Cette cohérence entre le Ladder, l'IHM et la configuration matérielle valide le repérage correct des sorties.

VI. Vérification des entrées

Les capteurs de position ou de détection sont configurés comme **entrées digitales** dans la table **Digital Inputs**.
Chaque capteur reçoit un alias permettant de l'identifier facilement dans le Ladder.

Exemple :

- ➔ `DETECTEUR_VERIN_BASSE` : Entrée **I_6**
- ➔ `DETECTEUR_VERIN_HAUTE` : Entrée **I_5**

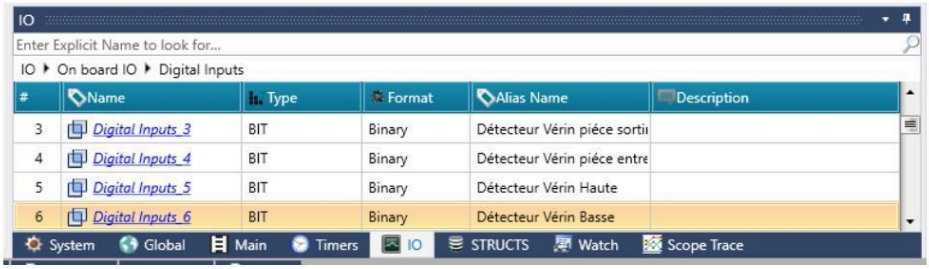


Figure 8 – Table des entrées digitales avec alias

Le test du bon fonctionnement peut être effectué en simulation ou en mode *Online* : les voyants de l'IHM changent d'état selon le retour des capteurs.

VII. Conclusion

Ce guide a présenté les principales étapes pour utiliser UniLogic, depuis la configuration matérielle jusqu'à la vérification des entrées/sorties.

Grâce à ces manipulations, l'utilisateur est en mesure de concevoir, tester et valider une application complète sur automate Unitronics.

VIII. Annexe – Raccourcis et astuces UniLogic

Action	***Raccourci / Méthode***
Rechercher une variable ou un tag	Clic droit → *Find Tag Results*
Identifier Sortie/Entrée associée à un tag	Clic droit → *Go To Definition*
Ajouter un module I/O	Clic droit sur *Hardware Configuration* → *Add Module*
Renommer un tag	F2 sur le nom dans la colonne Alias Name
Tester un programme	Menu *Build* → *Download & Run (F5)*
Activer la simulation	*Online Test* → *Monitor Tags*

Ces raccourcis permettent de gagner du temps lors de la configuration et du débogage des projets.

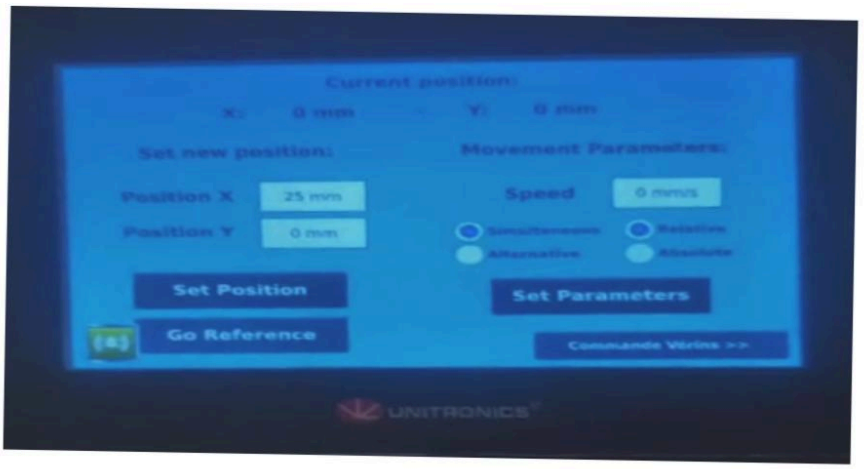
- Analyse et traitement du nouveau Code MSP modularisé

Modularisation du code MSP430 et correction de l'IHM

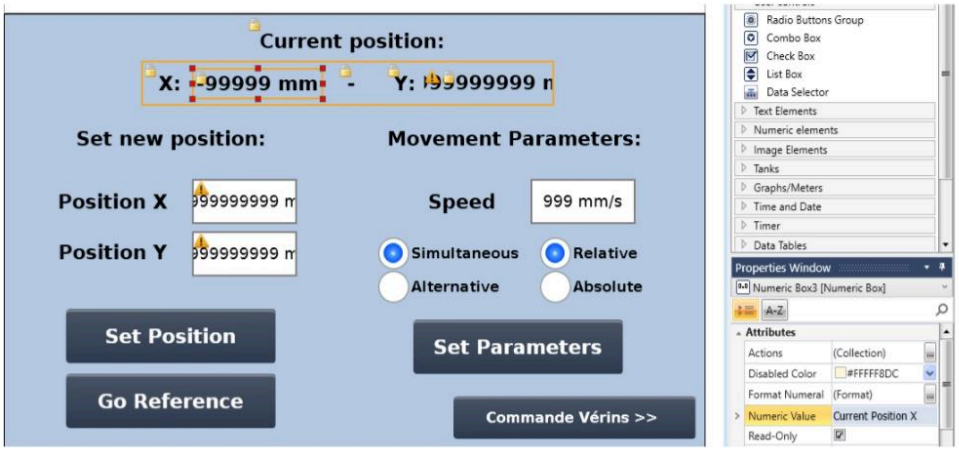
CHOUBRI Douae

1. Correction de l'IHM

Auparavant, dans la section *Current position*, les valeurs X et Y ne se mettaient pas à jour, même après avoir modifié les positions via *Set Position*.

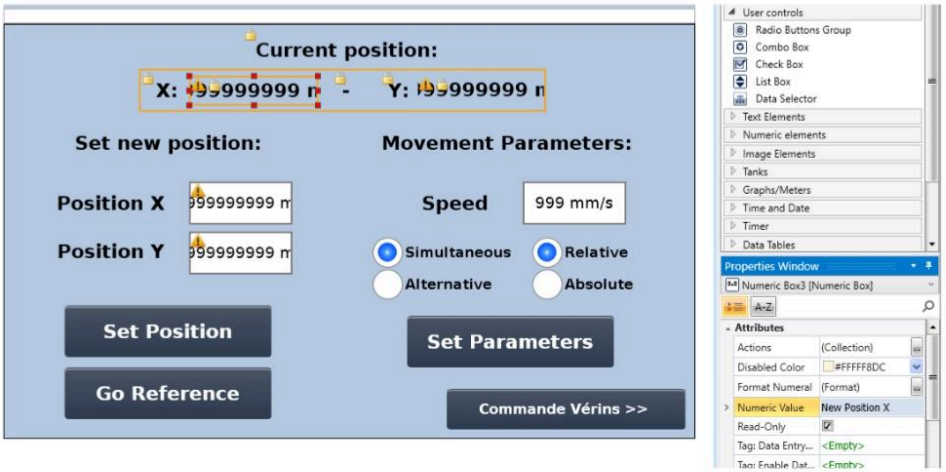


Le problème venait du fait que les variables utilisées ***(Numeric Value)*** pour afficher la position actuelle, **Current Position X** et **Current Position Y**, n'étaient définies nulle part dans le programme.

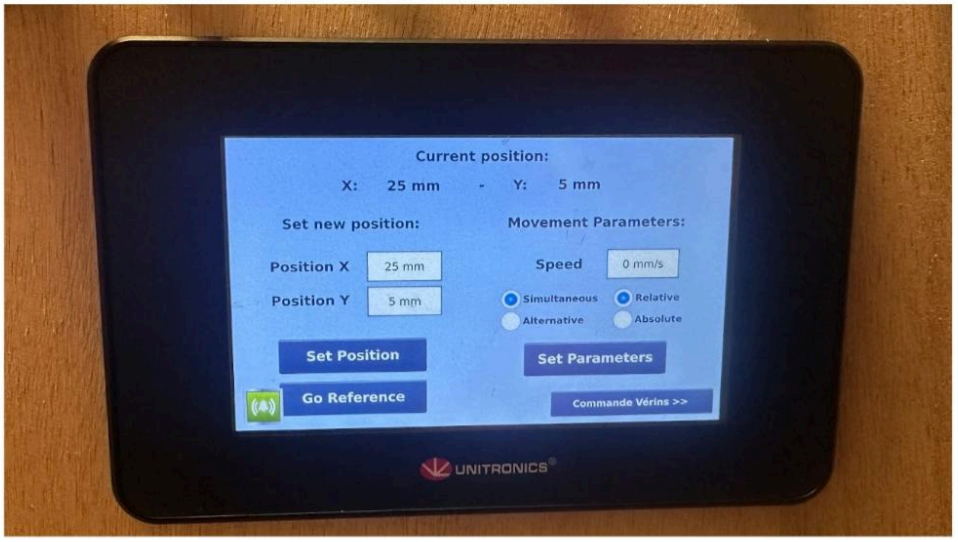


Douae Choubri, Reda El Madani, Salma Letrach, Ayoub Bouquennouche, Souad Ait Bellauali, Achraf Bouchemama, Roua Boughanmi, Amaury Pinéda

Pour corriger cela, les valeurs affichées dans la position actuelle ont été reliées aux variables **New Position X** et **New Position Y**, qui elles sont correctement définies dans les champs de saisie *Position X* et *Position Y* de la section *Set New Position*.



Grâce à cette correction, les valeurs X et Y se mettent désormais à jour correctement et varient simultanément dès qu'un changement est effectué dans l'IHM.



Douae Choubri, Reda El Madani, Salma Letrach, Ayoub Bouquennouche, Souad Ait Bellauali, Achraf Bouchemama, Roua Boughanmi, Amaury Pinéda

2. Modularisation du code MSP430

Avant modularisation, l'intégralité du programme était concentrée dans un seul fichier `main.c`. Ce fichier regroupait le contrôle moteurs, le décodage des commandes, la communication UART, les interruptions, les conversions mécaniques, la configuration GPIO, etc.
Le fichier atteignait **près de 498 pages**, ce qui rendait le code difficile à lire, maintenir ou déboguer.

Pour résoudre ce problème, une **réorganisation complète du projet** a été réalisée. Le programme est désormais séparé en plusieurs modules, chacun chargé d'une fonctionnalité précise.
Grâce à cette modularisation, le fichier `main.c` a été considérablement allégé : il ne fait plus que **67 pages**.

2.1 Nouvelle architecture du projet

La nouvelle structure du code repose sur des modules spécialisés :

➔ `motor.c / motor.h` — Gestion matérielle des moteurs

Ce module contient :

- l'initialisation du timer TB0 (`motor_timer_init()`)
- le contrôle du timer (`timer_on()`, `timer_off()`)
- la configuration des GPIO moteurs (`motor_gpio_init()`)
- les fonctions d'activation/désactivation (`enableMotors`, `enableMotorsX`, `enableMotorsY`)
- le homing utilisant les capteurs de fin de course (`HomePositionXY`)
- les conversions mécaniques mm→ pas (`mmToStep`, `stepToMm`)

Il regroupe ainsi tout le contrôle bas-niveau des moteurs pas-à-pas X et Y.

➔ `command.c / command.h` — Interprétation des commandes

Ce module traite les commandes envoyées par l'IHM/PLC sous forme de chaînes UART :

- Commande de ***position*** : **P**xx-yy → Convertit mm → pas, choisit la direction, lance le mouvement, envoie <P1>.
- Commande de ***vitesse*** : **V**vitesse-simAlt-relAbs → Met à jour la vitesse, reconfigure le timer, envoie <S1>.
- Commande ***Home*** : **H** → Lance le retour automatique des axes à leur position d'origine.
- Sélecteur de commande : get_command() → Dirige vers la fonction adaptée (P / V / H)

Ce fichier regroupe tout le niveau logique du robot.

➔ `uart.c / uart.h` — Communication UART

Ce module gère :

- la configuration complète des UART A0 et A1,
- les buffers de réception/émission (RX_Buffer, TX_Buffer),
- l'envoi non bloquant via interruptions (`uart_put_string()`),
- la réception caractère par caractère et la reconstruction automatique d'une commande.

Il prend en charge toute la communication série entre MSP430 et l'IHM/PLC.

➔ `isr.c / isr.h` — Routines d'interruption

Ce module centralise toutes les ISR :

- ***ISR Timer TB0 (moteurs)***
  - génère les pas moteurs (toggle STEP),
  - décrémente les pas restants,
  - arrête automatiquement le timer en fin de mouvement.
- ***ISR UART A0/A1***
  - réception caractère par caractère,
  - détection du début (<) et fin (> ou CR/LF) de commande,
  - appel direct à `get_command`().

- ***ISR fin de course (Port 1)***
    - arrête immédiatement l'axe concerné,
    - remet la position à zéro (Home position).

  Ces ISR permettent un fonctionnement asynchrone, fluide et totalement non bloquant.

2.2 Un fichier `main.c` plus simple et clair

Le fichier `main.c` ne contient plus que les actions essentielles :

- ➢ Initialisation des horloges
- ➢ Initialisation des GPIO moteurs
- ➢ Initialisation des capteurs de fin de course
- ➢ Initialisation UART
- ➢ Activation des moteurs
- ➢ Configuration des interruptions
- ➢ Boucle infinie vide (tout fonctionne par interruptions)

Exemple extrait du main :

```
motor_gpio_init();
motor_timer_init();
uart_init();
GPIO_enableInterrupt(FCX_PORT, FCX_PIN);
GPIO_enableInterrupt(FCY_PORT, FCY_PIN);
__enable_interrupt();
while(1){
    // Programme entièrement piloté par ISR
}
```

Le rôle du main est désormais de **configurer l'environnement**, puis de laisser les modules travailler en arrière-plan.

2.3 Résultats de la modularisation

Grâce à cette séparation :

- ★ `main.c` passe de 498 pages → 67 pages
- ★ Chaque fonctionnalité possède son propre module

- ★ Le code est lisible, structuré et évolutif
- ★ Les interruptions gèrent tout en temps réel
- ★ La communication UART est propre et non bloquante
- ★ Le mouvement moteur est sécurisé et centralisé
- ★ Cette modularisation constitue une amélioration majeure en termes de **clarité**, de **maintenance**, de **fiabilité**, et d'**extensibilité** du code.

Partie mécanique

- Chaine porte câbles (en cours de développement)

L'idée est d'implémenter une chaîne porte câbles afin de libérer l'espace de travail de tout élément structurel (câbles encombrant l'espace de travail du robot notamment)

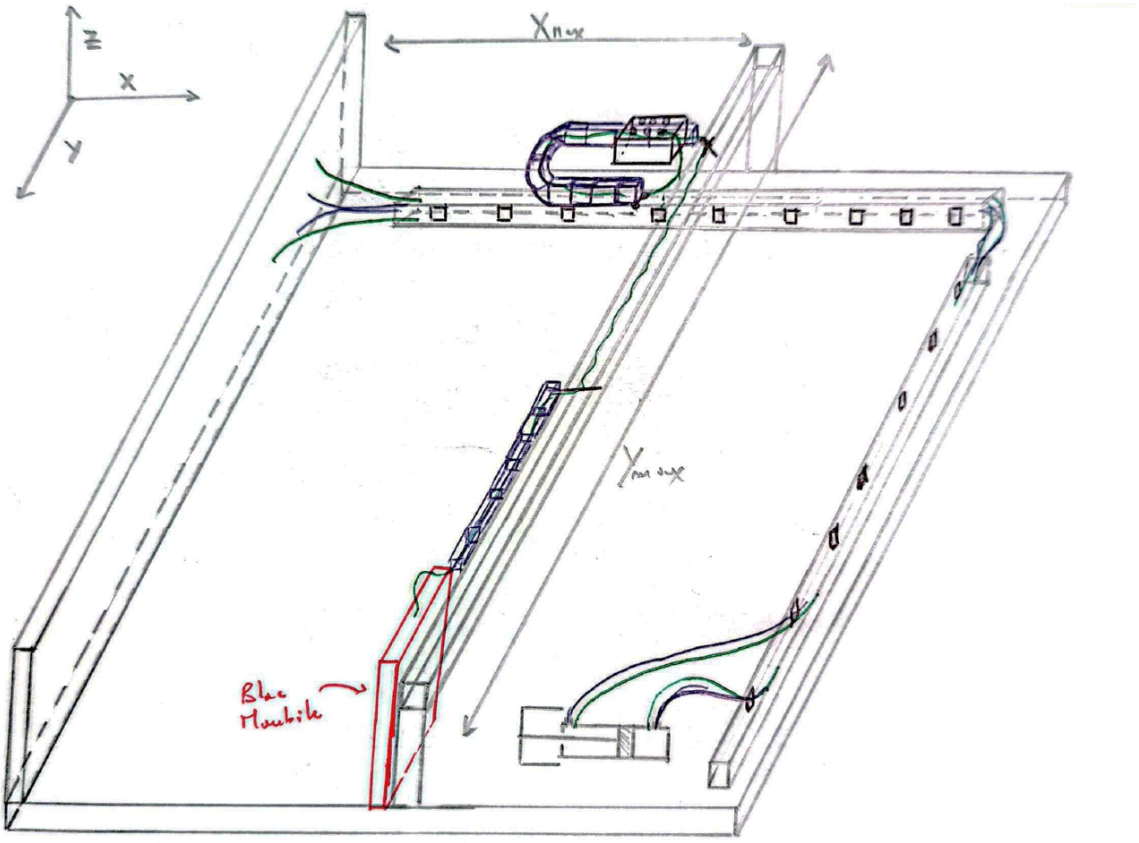
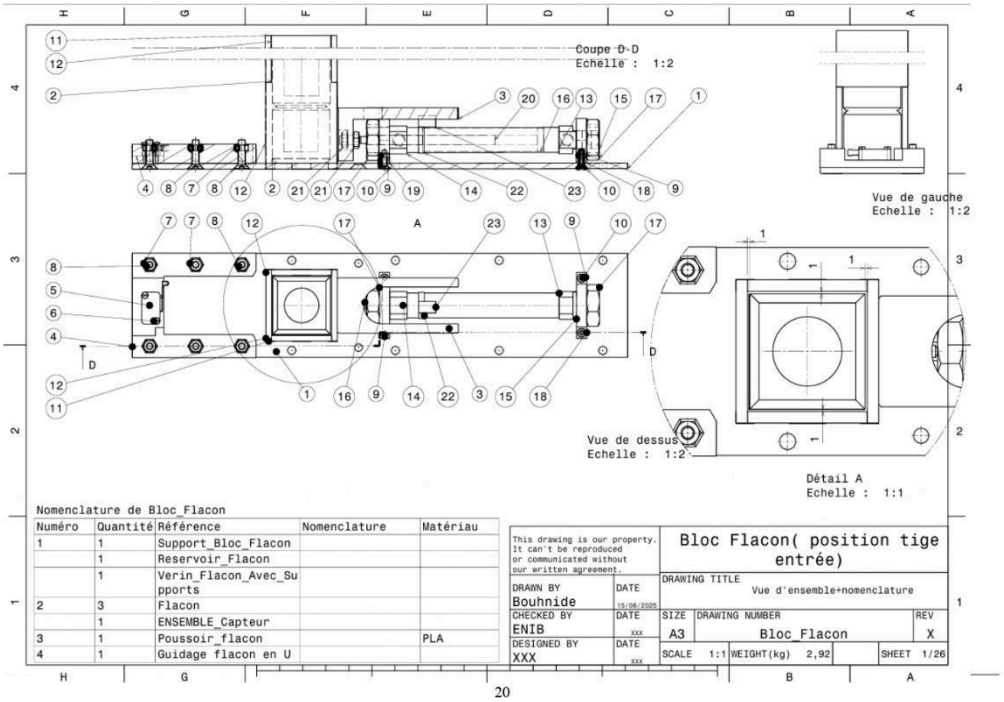


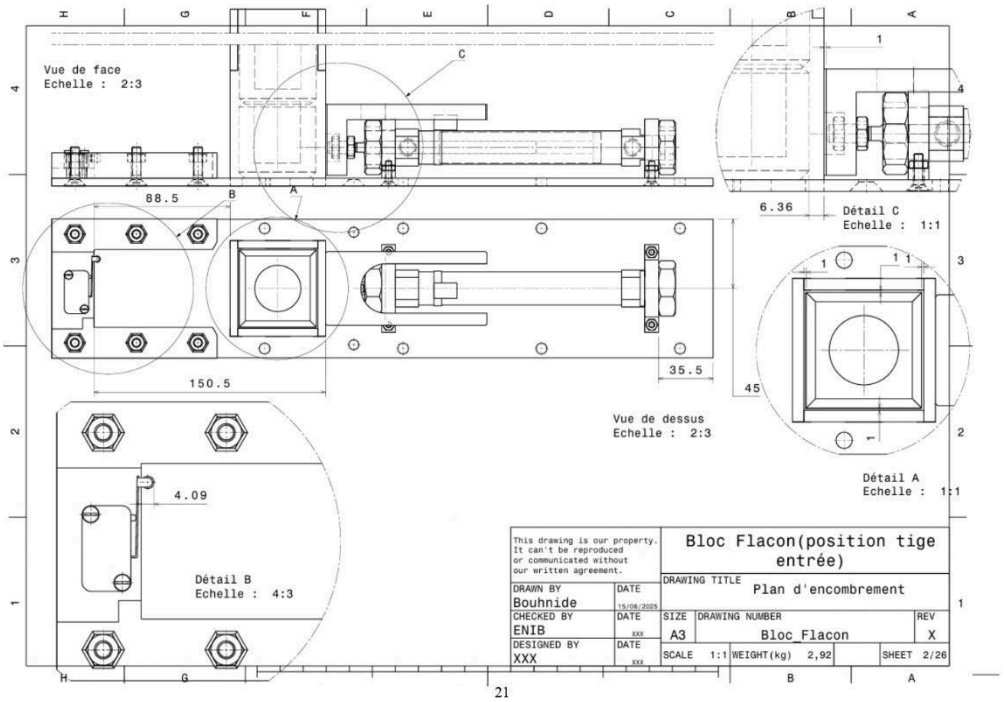
Schéma conceptuel ci-dessous :

Douae Choubri, Reda El Madani, Salma Letrach, Ayoub Bouquennouche, Souad Ait Bellauali, Achraf Bouchemama, Roua Boughanmi, Amaury Pinéda

- Mises en plan et nomenclature

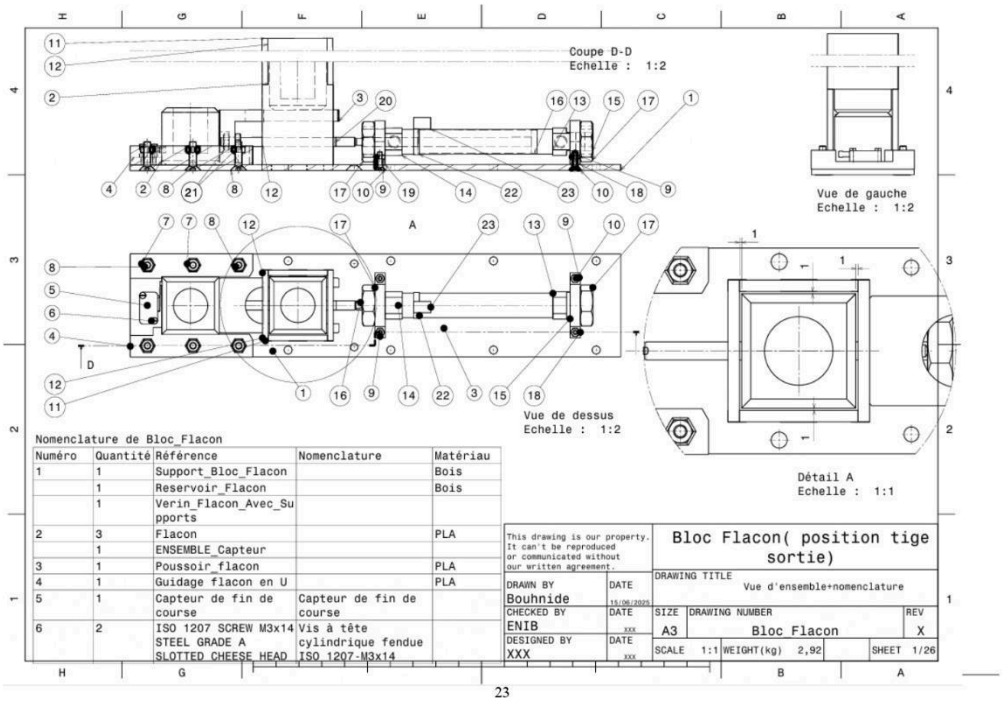


20



21

Douae Choubri, Reda El Madani, Salma Letrach, Ayoub Bouquennouche, Souad Ait Bellauali, Achraf Bouchemama, Roua Boughanmi, Amaury Pinéda



Nomenclature de Bloc_Flacon

Numéro	Quantité	Référence	Nomenclature	Matériau
1	1	Support_Bloc_Flacon		Bois
	1	Reservoir_Flacon		Bois
	1	Verin_Flacon_Avec_Supports		
2	3	Flacon		PLA
	1	ENSEMBLE_Capteur		
3	1	Poussoir_flacon		PLA
4	1	Guidage flacon en U		PLA
5	1	Capteur de fin de course	Capteur de fin de course	
6	2	ISO 1207 SCREW M3x14 STEEL GRADE A SLOTTED CHEESE HEAD	Vis à tête cylindrique fendue ISO 1207-M3x14	

This drawing is our property. It can't be reproduced or communicated without our written agreement.

Bloc Flacon(position tige sortie)

DRAWN BY Bouhnide	DATE 15/06/2025	DRAWING TITLE Vue d'ensemble+nomenclature		
CHECKED BY ENIB	DATE XXX	SIZE A3	DRAWING NUMBER Bloc Flacon	REV X
DESIGNED BY XXX	DATE XXX	SCALE 1:1	WEIGHT(kg) 2,92	SHEET 1/26

23

Numéro	Quantité	Référence	Nomenclature	Matériau
4	1	Guidage flacon en U		PLA
5	1	Capteur de fin de course	Capteur de fin de course	
6	2	ISO 1207 SCREW M3x14 STEEL GRADE A SLOTTED CHEESE HEAD	Vis à tête cylindrique fendue ISO 1207-M3x14	
7	6	ISO 4032 NUT M6 STEEL GRADE A WASHER FACED HEXAGON HEAD	Ecrou hexagonal M6	
8	6	ISO 10642 SCREW M6x25 STEEL HEXAGON SOCKET COUNTERSUNK HEAD	Vis TF M6x25 8.8	
9	4	ISO 4032 NUT M4 STEEL GRADE A HEXAGON	Ecrou hexagonal M4	
10	4	ISO 10642 SCREW M4x16 STEEL HEXAGON SOCKET COUNTERSUNK HEAD	Vis TF M4x16 8.8	
11	2	Reservoir_Flacon_Face		Bois
12	2	Reservoir_Flacon_Cote		Bois
13	1	CD85N20-100-B_TUBE1_20	CD85N20-100-B_TUBE1_20	
14	1	CD85N20-100-B_END_COVER_20E1	CD85N20-100-B_END_COVER_20E1	
15	1	CD85N20-100-B_END_COVER_20N	CD85N20-100-B_END_COVER_20N	
16	1	CD85N20-100-B_ROD1_20	CD85N20-100-B_ROD1_20	
17	2	CD85N20-100-B_NT-SN_NUT_C85SN25	CD85N20-100-B_NT-SN_NUT_C85SN25	
18	1	Support_Verin_Flacon_AR_2		
19	1	support verin avec cercle		PLA
20	1	Part1	Tige de vérin	

Numéro	Quantité	Référence	Nomenclature	Matériau
21	2	CD85N20-100-B_NT-SN_NUT_M8	CD85N20-100-B_NT-SN_NUT_M8	
2	3	Flacon		PLA
22	1	bague_fixation_capteur		
23	1	Part23		
3	1	Poussoir_flacon		PLA

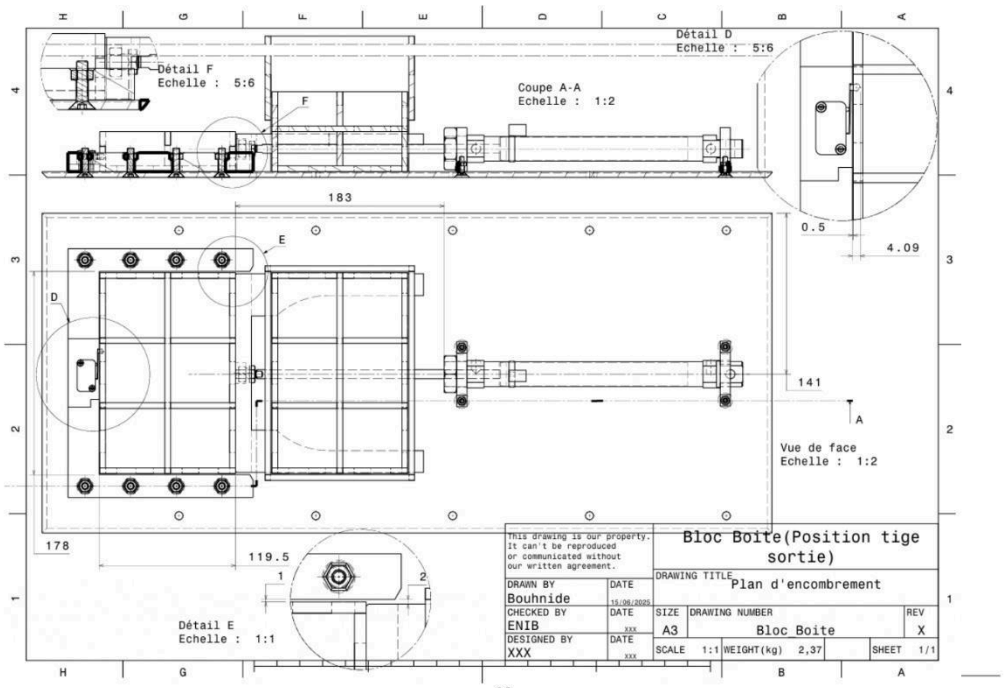
This drawing is our property. It can't be reproduced or communicated without our written agreement.

Bloc Flacon

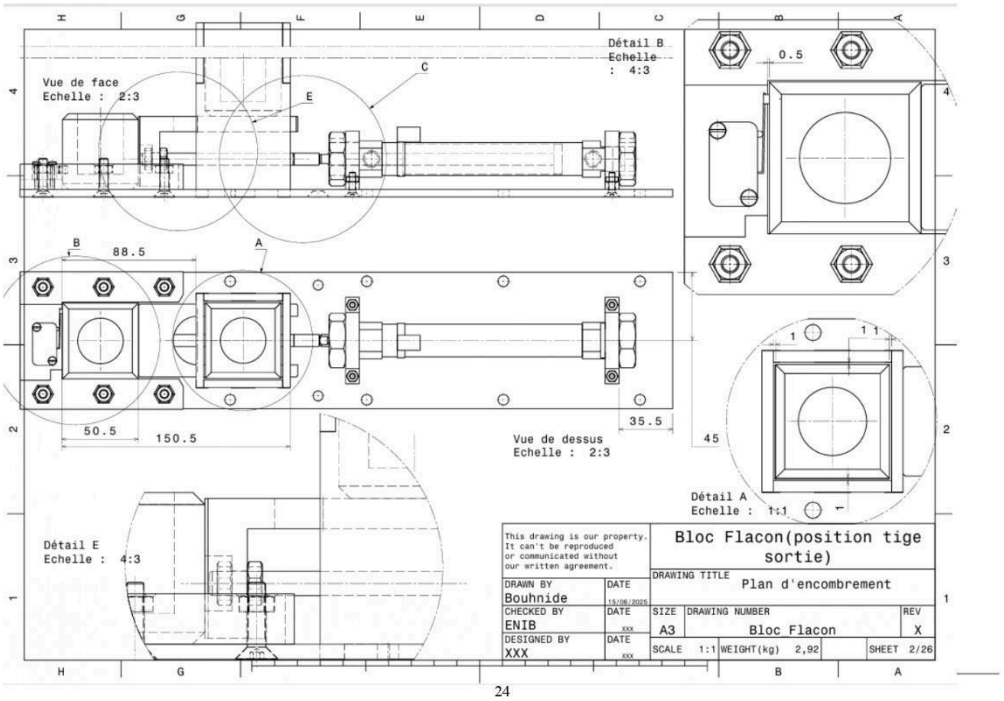
DRAWN BY Bouhnide	DATE 15/06/2025	DRAWING TITLE Nomenclature Bloc Flacon		
CHECKED BY ENIB	DATE XXX	SIZE A3	DRAWING NUMBER XXX	REV X
DESIGNED BY XXX	DATE XXX	SCALE 1:1	WEIGHT(kg) XXX	SHEET 3/26

22

Douae Choubri, Reda El Madani, Salma Letrach, Ayoub Bouquennouche, Souad Ait Bellauali, Achraf Bouchemama, Roua Boughanmi, Amaury Pinéda

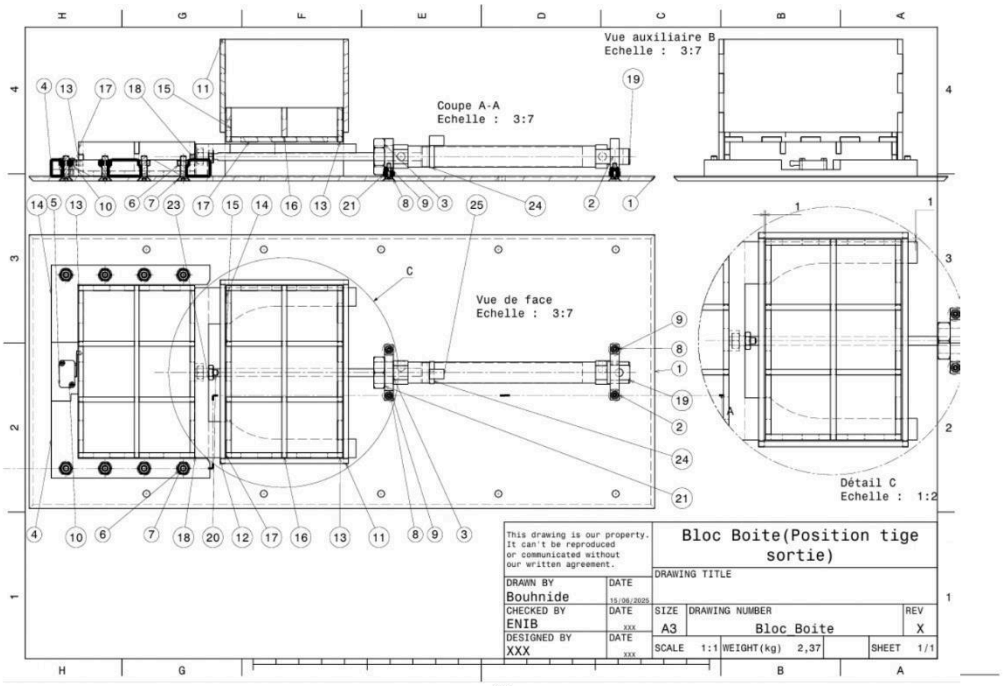


25



24

Douae Choubri, Reda El Madani, Salma Letrach, Ayoub Bouquennouche, Souad Ait Bellauali, Achraf Bouchemama, Roua Boughanmi, Amaury Pinéda



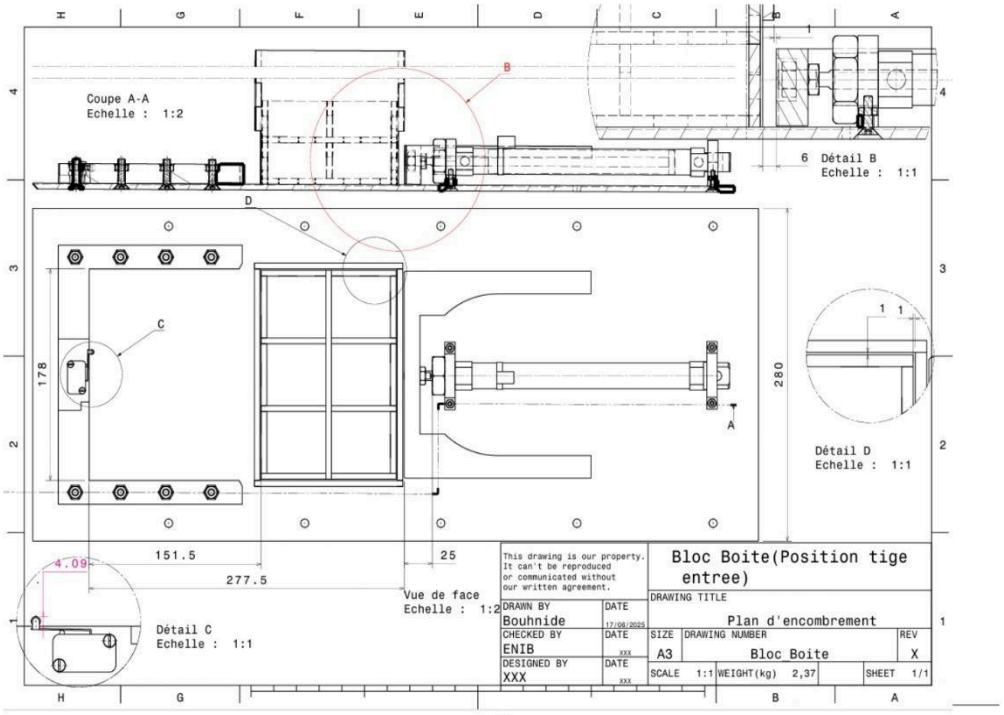
26

Numéro	Quantité	Référence	Nomenclature	Matériau
	1	Reservoir_Boite		Bois
	3	Boite		Bois
	1	Product9		
1	1	Support bloc boite		Bois
4	1	Guidage boite en U		PLA
5	1	Capteur de fin de course	Capteur de fin de course	
7	8	ISO 4032 NUT M6 STEEL GRADE A WASHER FACED HEXAGON HEAD	Ecrou hexagonal M6	
7	8	ISO 10642 SCREW M6x25 STEEL HEXAGON SOCKET COUNTERSUNK HEAD	Vis TF M6x25 8.8	
8	4	ISO 10642 SCREW M4x16 STEEL HEXAGON SOCKET COUNTERSUNK HEAD	Vis TF M4x16 8.8	
9	4	ISO 4032 NUT M4 STEEL GRADE A HEXAGON	Ecrou hexagonal M4	
6	2	ISO 1207 SCREW M3x14 STEEL GRADE A SLOTTED CHEESE HEAD	Vis à tête cylindrique fendue ISO 1207-M3x14	
11	2	Reservoir_Boite_Face	Boite	Bois
12	2	Reservoir_Boite_Cote		
13	6	Boite_Paroi_Longueur		
14	3	Boite_Fond		
15	6	Boite_Intercalaire_Largeur		
16	3	Boite_Intercalaire_Longueur		
17	6	Boite_Paroi_Largeur		
18	1	Ensemble poussoir boite		PLA
20	1	CD85N20-160-B_ROD1_20	CD85N20-160-B_ROD1_20	
21	1	02763033_03		
22	1	02763033_04		
23	1	CD85N20-160-B_NT-SN_NUT_M8	CD85N20-160-B_NT-SN_NUT_M8	

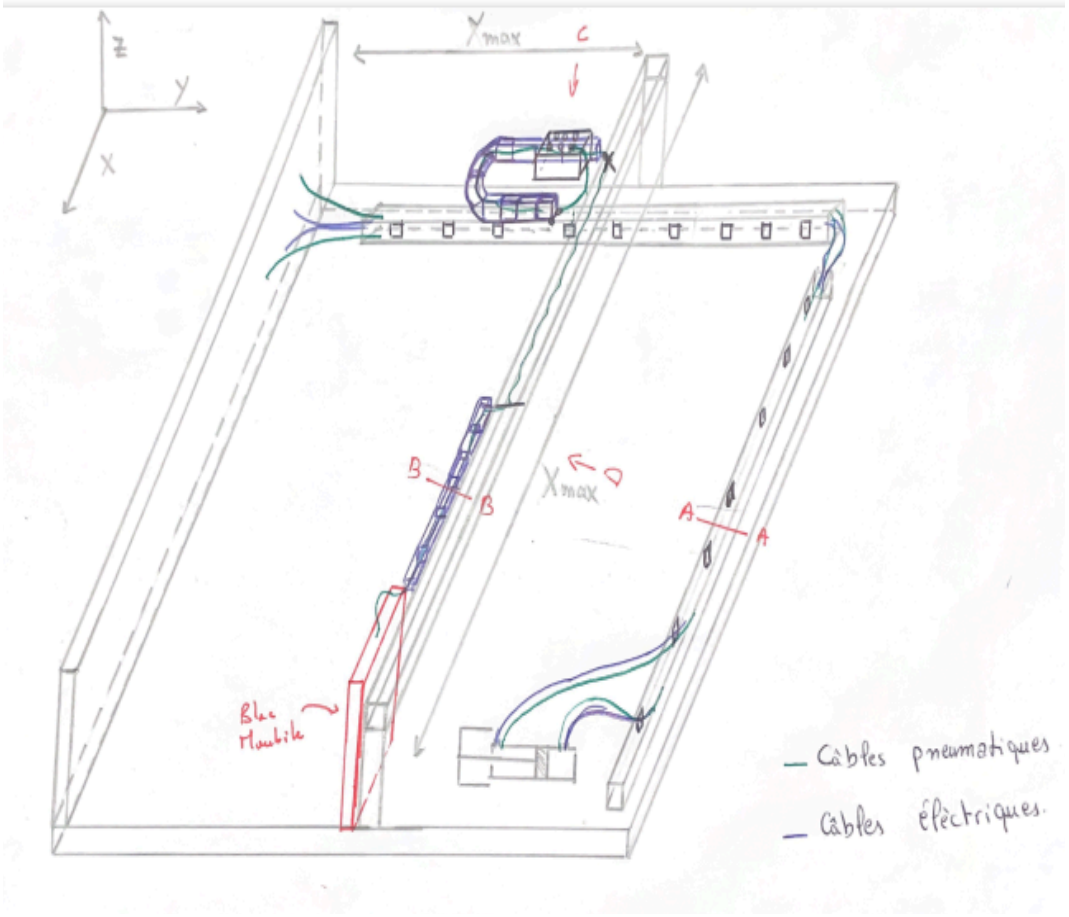
This drawing is our property. It can't be reproduced or communicated without our written agreement.
Bloc Boite(Position tige sortie)
DRAWING TITLE
DRAWN BY Bouhnide
DATE 15/06/2025
CHECKED BY ENIB
DESIGNED BY XXX
SIZE A3
DRAWING NUMBER Bloc Boite
REV X
SCALE 1:1
WEIGHT(kg) 2,37
SHEET 1/1

27

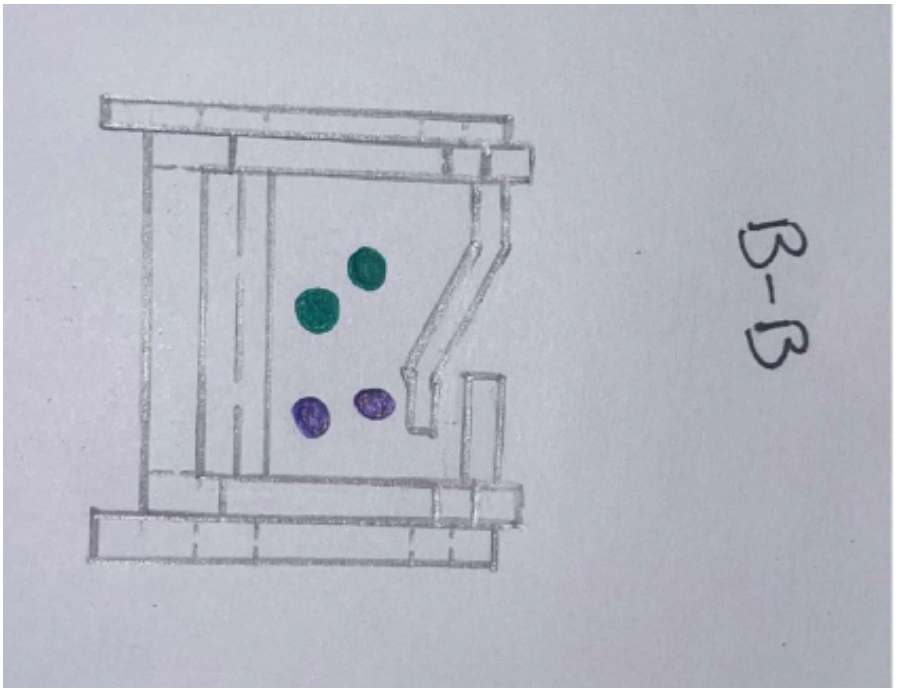
Douae Choubri, Reda El Madani, Salma Letrach, Ayoub Bouquennouche, Souad Ait Bellauali, Achraf Bouchemama, Roua Boughanmi, Amaury Pinéda

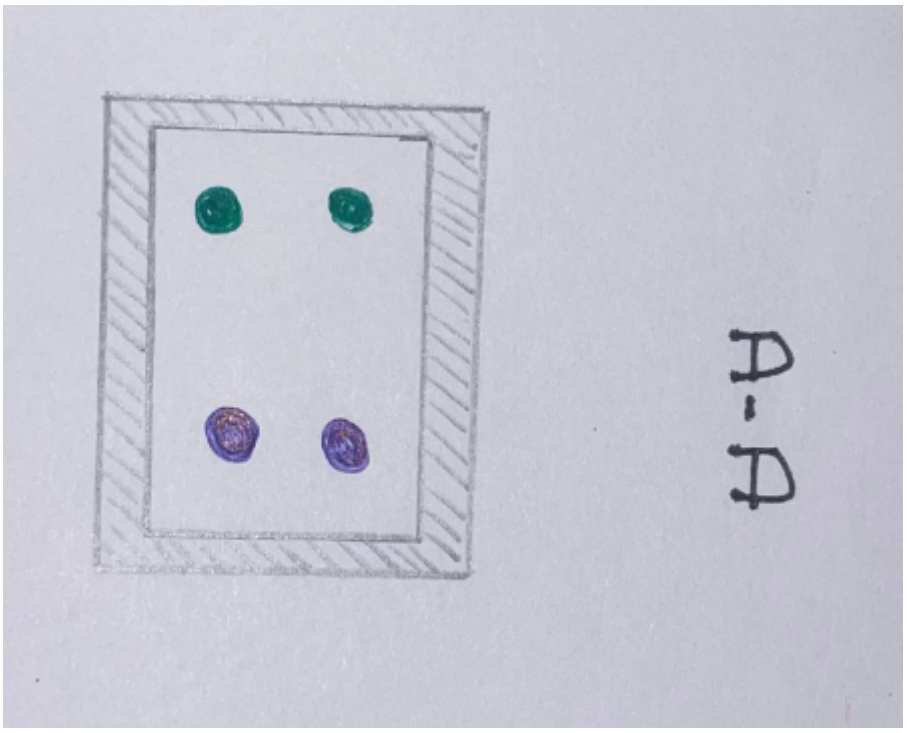


- Étapes de mises en place du maillon

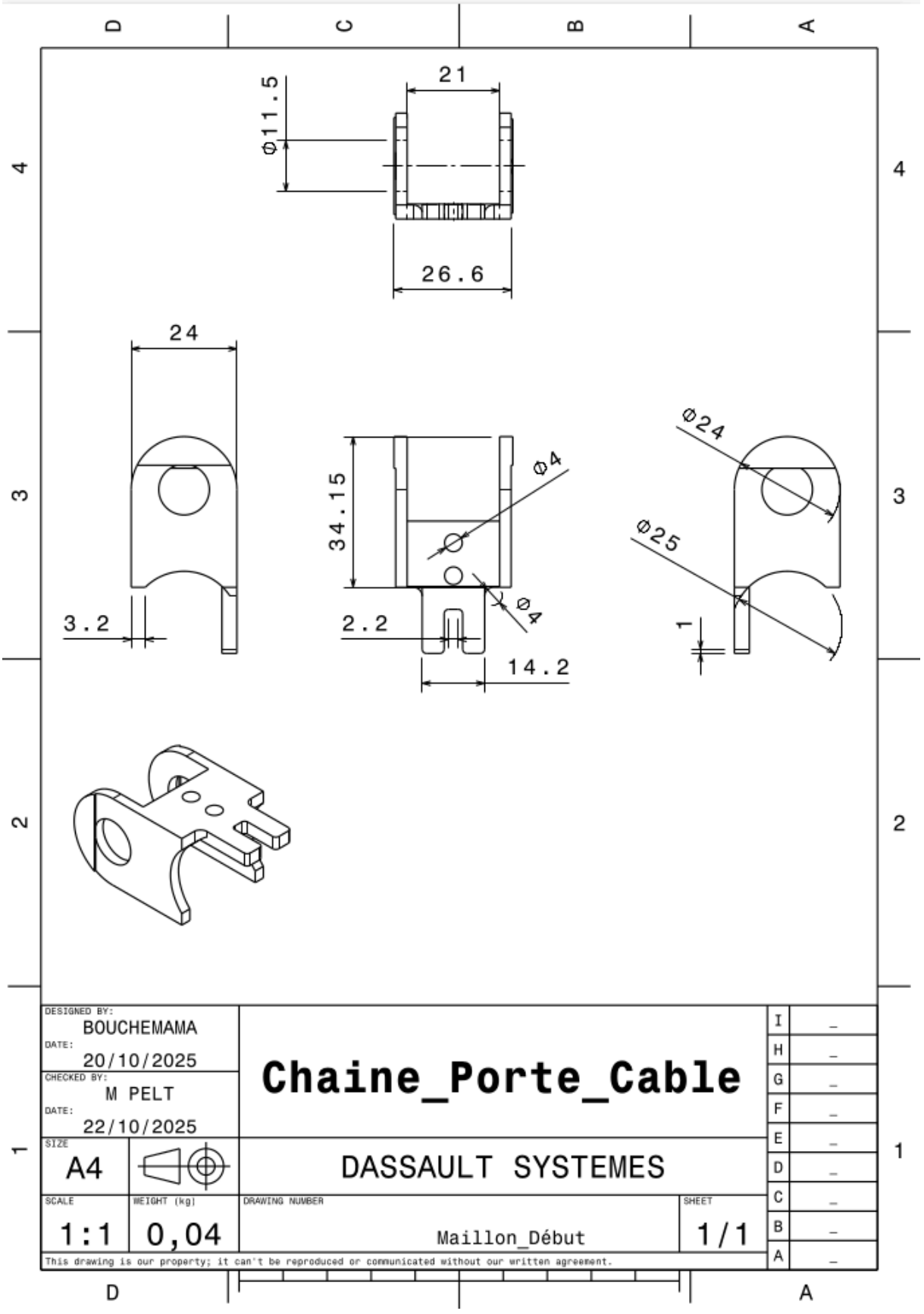


Douae Choubri, Reda El Madani, Salma Letrach, Ayoub Bouquennouche, Souad Ait Bellauali, Achraf Bouchemama, Roua Boughanmi, Amaury Pinéda

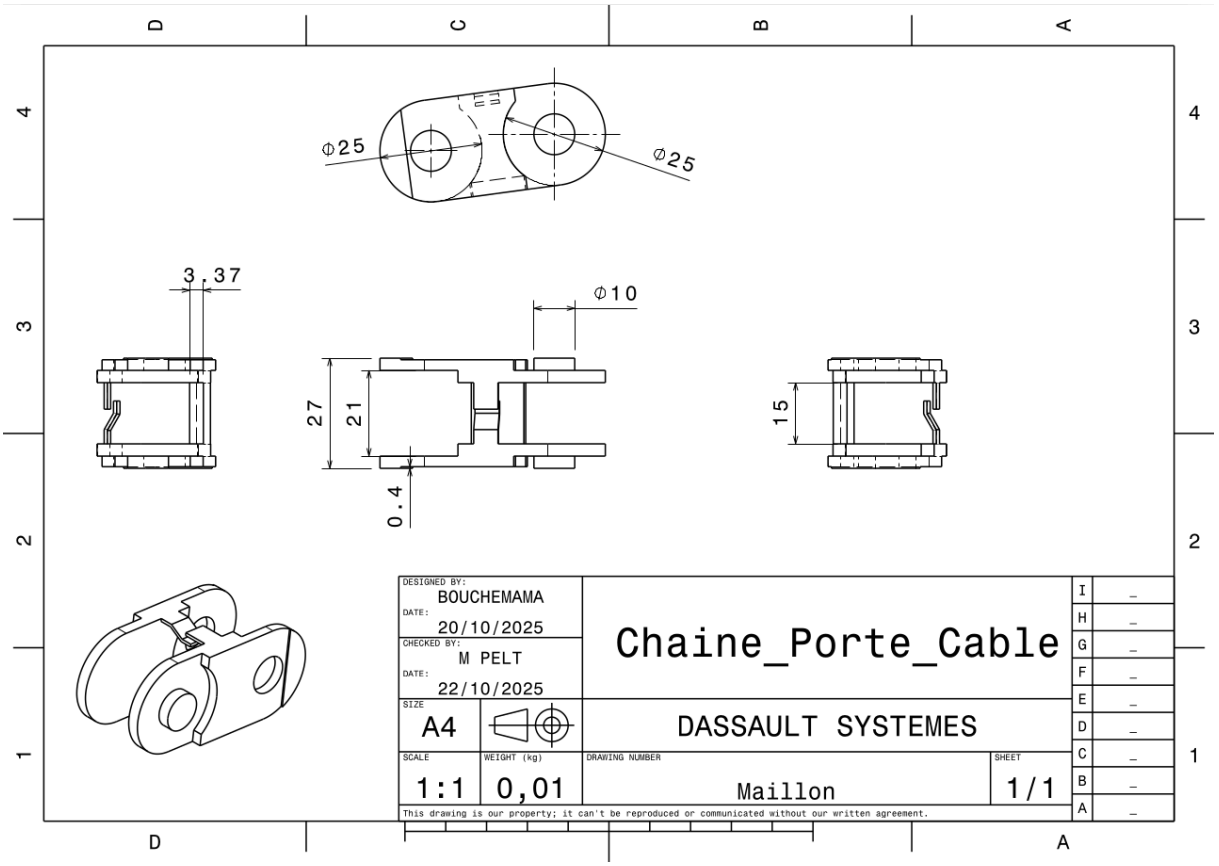




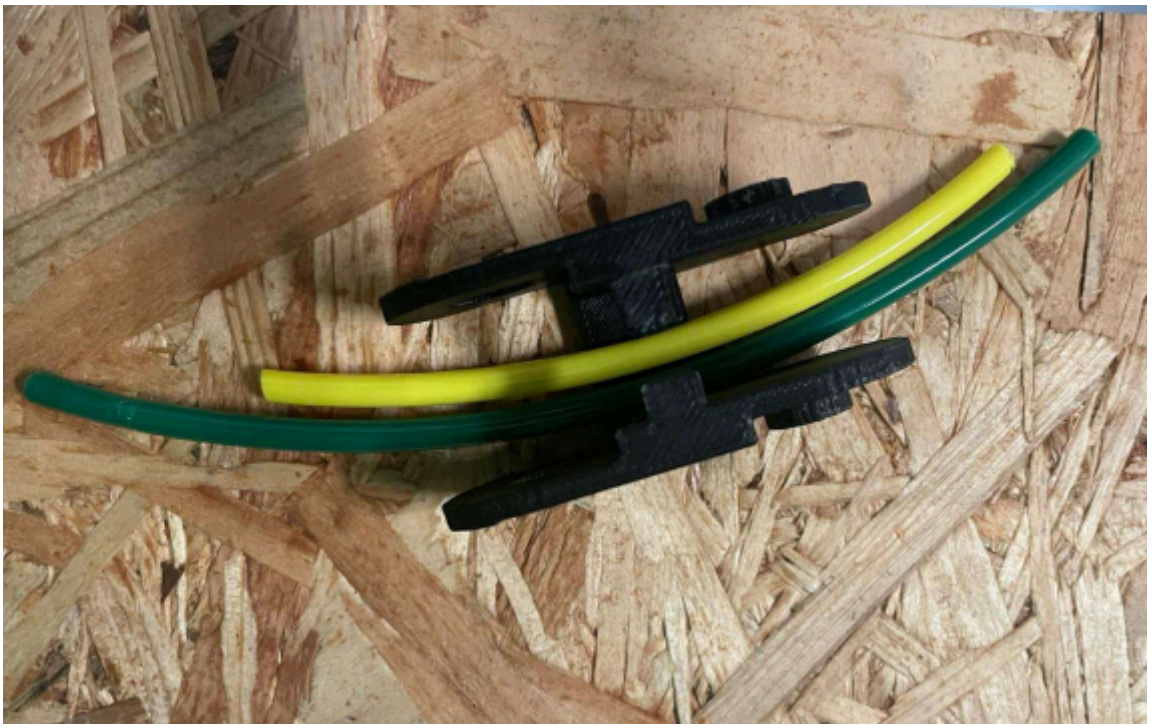
Douae Choubri, Reda El Madani, Salma Letrach, Ayoub Bouquennouche, Souad Ait Bellauali, Achraf Bouchemama, Roua Boughanmi, Amaury Pinéda



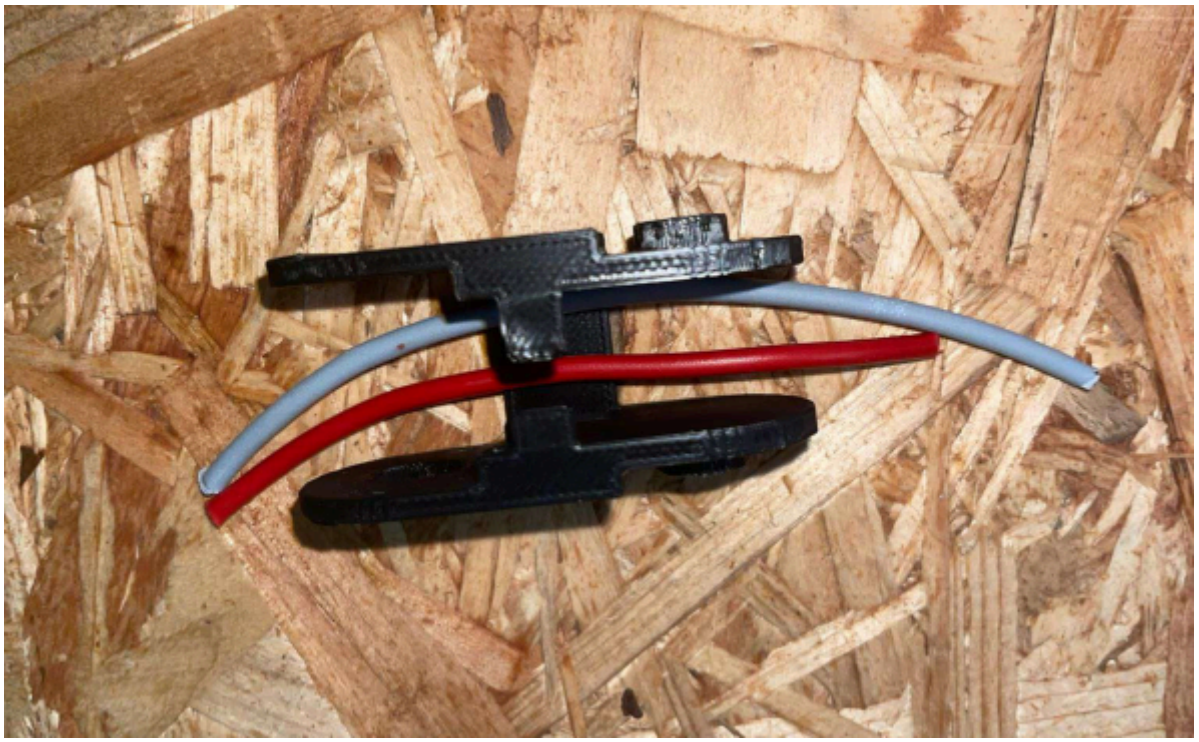
Douae Choubri, Reda El Madani, Salma Letrach, Ayoub Bouquennouche, Souad Ait Bellauali, Achraf Bouchemama, Roua Boughanmi, Amaury Pinéda



- Résultat visuel du maillon



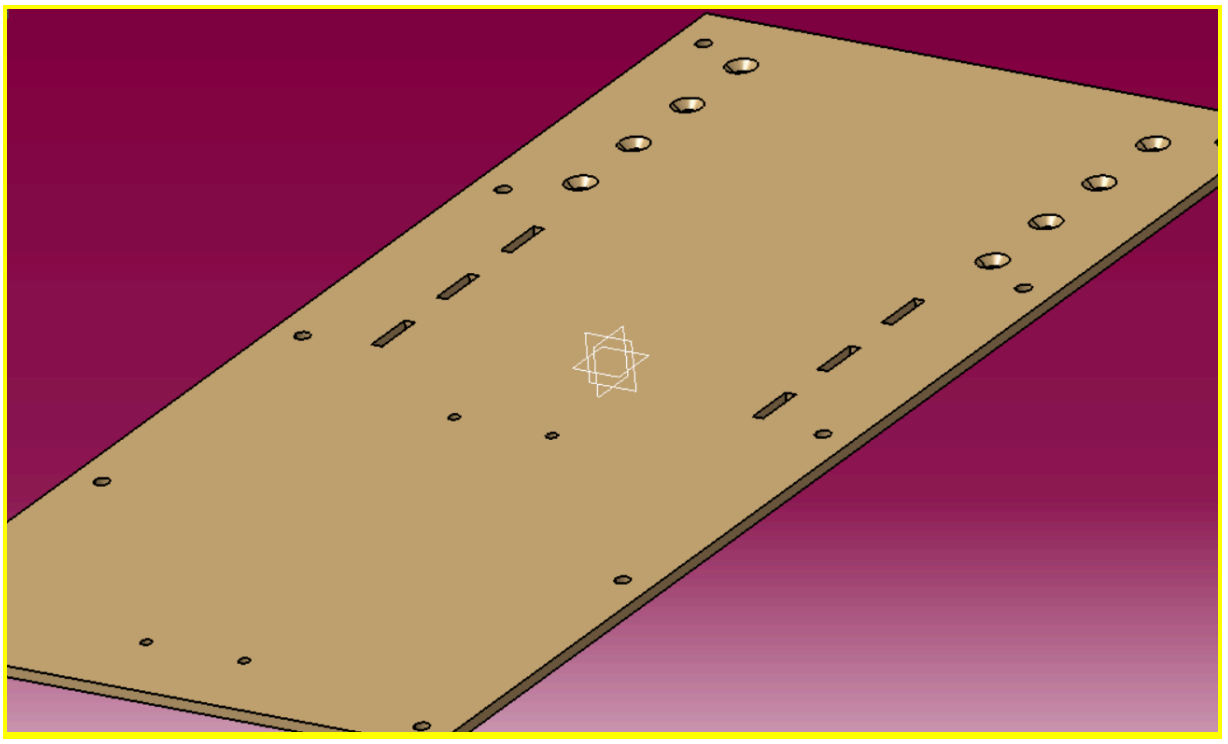
Douae Choubri, Reda El Madani, Salma Letrach, Ayoub Bouquennouche, Souad Ait Bellauali, Achraf Bouchemama, Roua Boughanmi, Amaury Pinéda



Douae Choubri, Reda El Madani, Salma Letrach, Ayoub Bouquennouche, Souad Ait Bellauali, Achraf Bouchemama, Roua Boughanmi, Amaury Pinéda

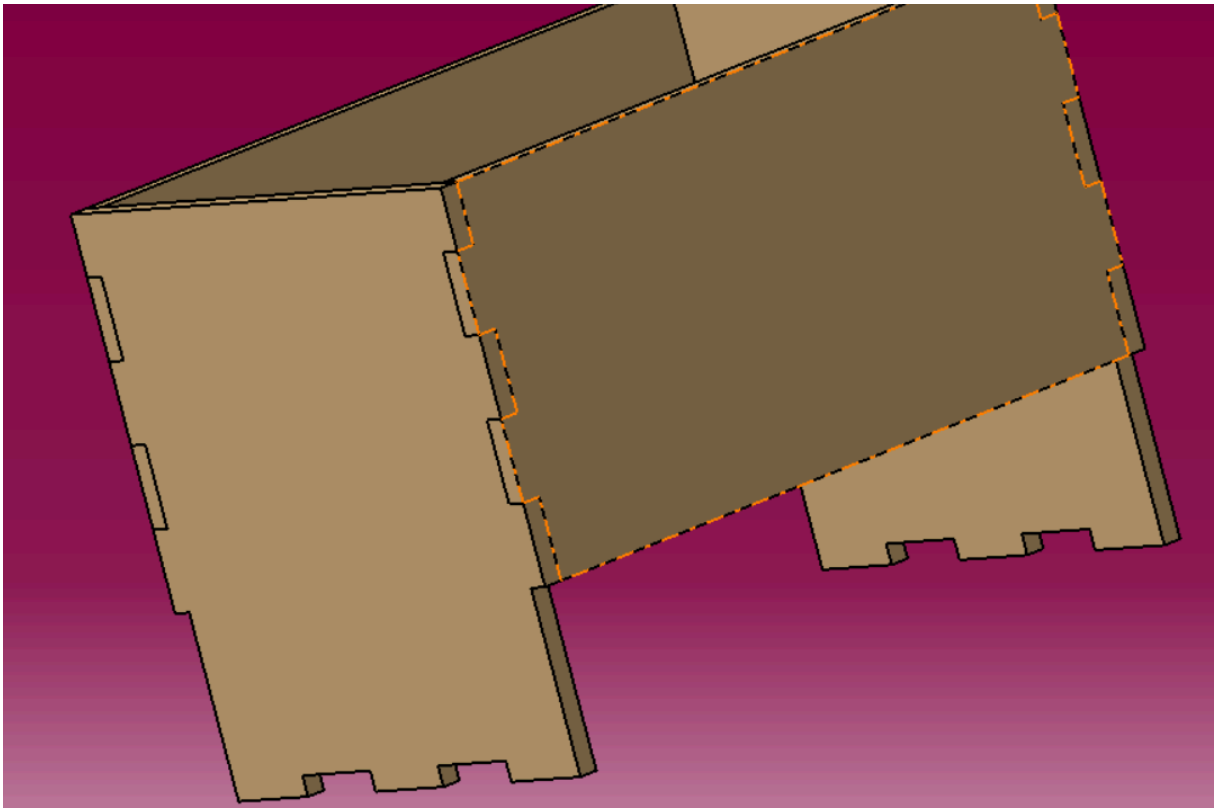
- Support vérin boîte

Afin de fiabiliser la fixation du réservoir sur la plaque support, cette dernière a d'abord été modifiée par l'intégration de trois encoches et par une mise à jour de la géométrie globale, afin d'améliorer le positionnement des éléments et de faciliter l'assemblage. En parallèle, les entraxes des perçages ont été redéfinis et intégrés directement dans l'assemblage global, garantissant un montage cohérent et sans ajustements manuels.



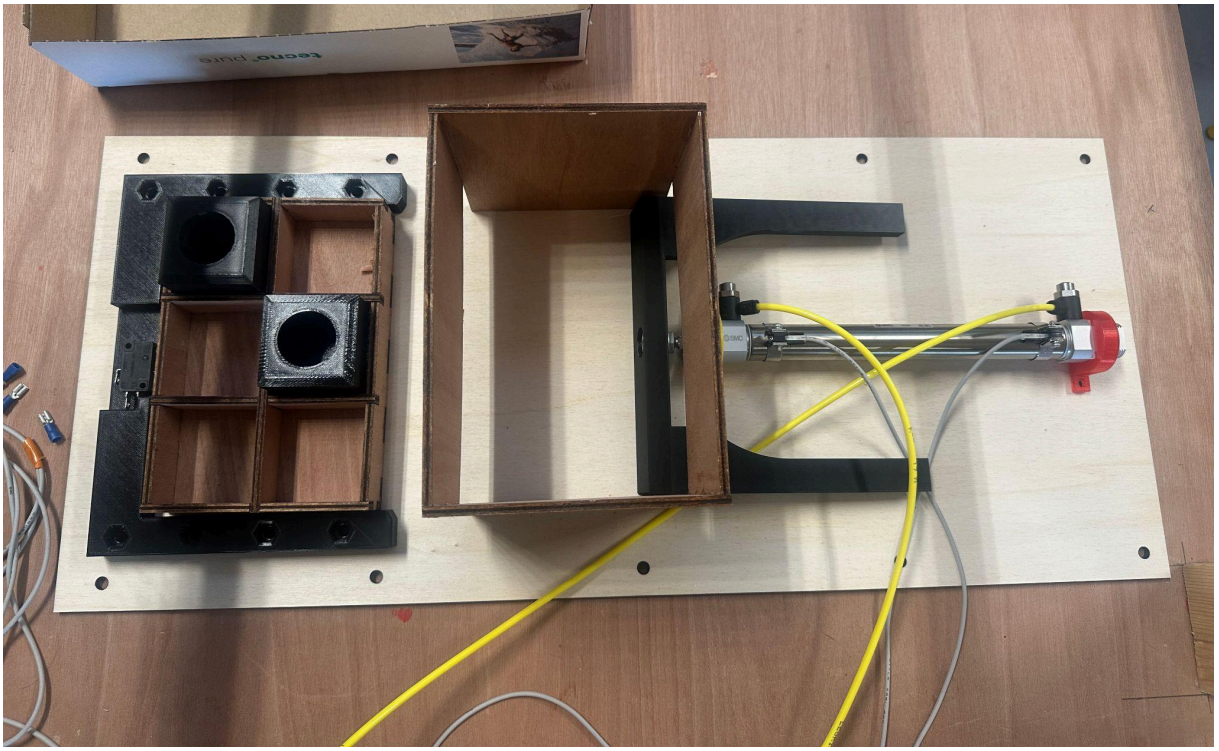
Ensuite, le réservoir a été modifié pour permettre une **fixation correcte sur la plaque**, sans collage. L'idée est d'obtenir un assemblage :

- **plus rigide** (le collage peut créer des jeux ou se décoller avec les efforts répétés),
- **plus fiable dans le temps** (meilleure tenue face aux vibrations et manipulations),



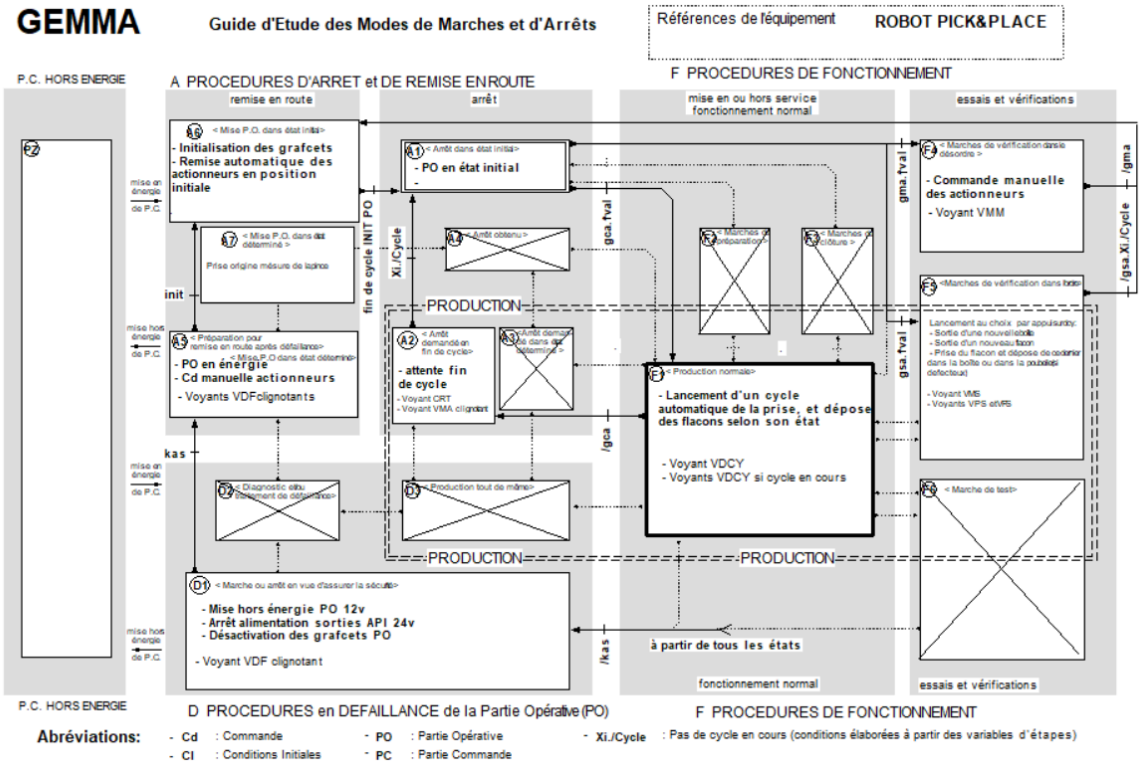
Enfin, l'ensemble modifié (plaque + réservoir) a été **mis à jour dans l'assemblage global**, et les vis ont été adaptées aux nouveaux trous pour correspondre au montage réel prévu.

- Résultat visuel du support vérin boîte



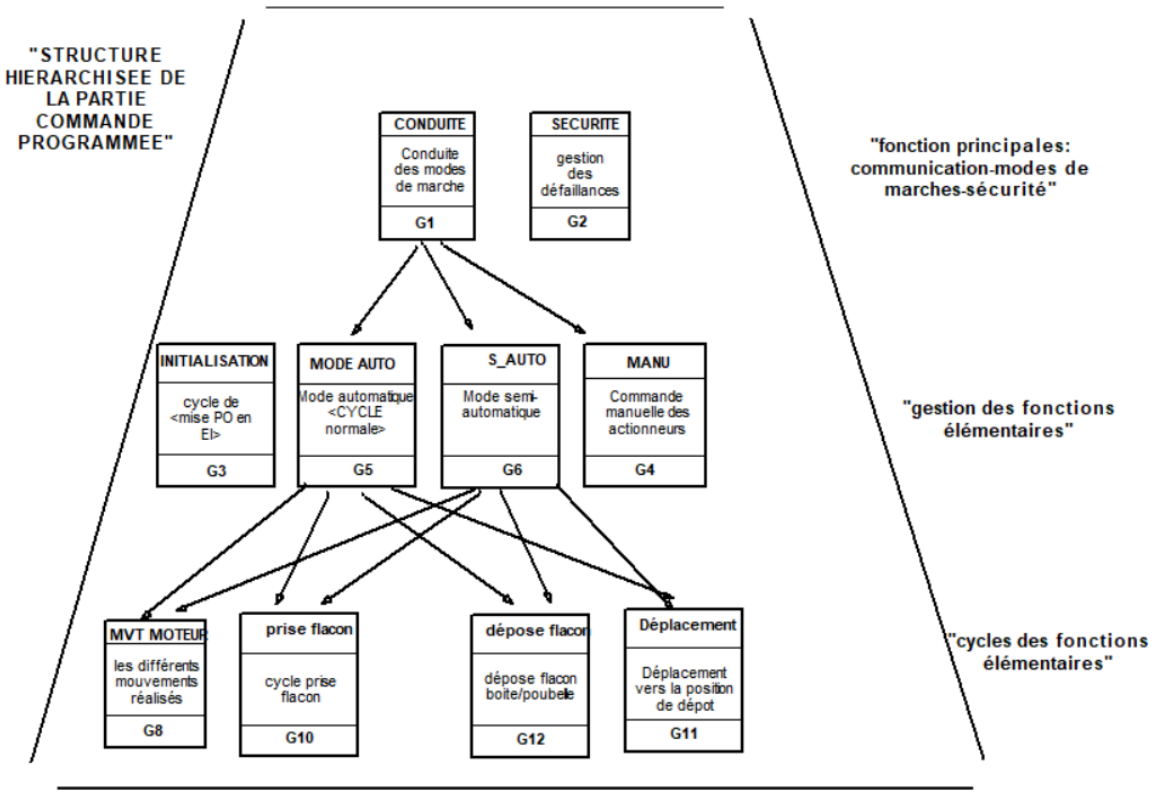
Partie électronique

- Nouvelle version de Gemma proposée



Douae Choubri, Reda El Madani, Salma Letrach, Ayoub Bouquennouche, Souad Ait Bellauali, Achraf Bouchemama, Roua Boughanmi, Amaury Pinéda

- Structure hiérarchisée Gemma proposée



- Nouvelle version des Grafcets proposée

Variables utilisées :

AUTO : Choix du mode Auto

S-AUTO : Choix du mode Semi auto

MANU : Choix du mode manuel

dep : Choix sous-cycle Dépose flacon

pri : Choix sous-cycle Prise flacon

dcy : Départ cycle auto

acq : Acquittement de la prise de la boîte pleine

- Capteurs course des vérins

1s1 : Pince rentrée

1s2 : Pince sortie

2s1 : Pince ouverte

2s2 : Pince fermée

3s1 : Dépileur boîte rentré

3s2 : Dépileur boîte sorti

4s1 : Dépileur flacon rentré

4s2 : Dépileur flacon sorti

- Capteurs

pomX : Capteur d'origine Axe X

pomY : Capteur d'origine Axe Y

def : Défaut flacon

- Moteur

kmxa : Moteur X sens A

kmxb : Moteur X sens B

kmya : Moteurs Y sens A

kmyb : Moteurs Y sens B

- Positions

POS_DES : Position envoyé depuis la carte pour la prochaine position du flacon dans la boîte

POS_DEF : Position de la boîte à défauts

POS_CAP : Position pour faire la mesure sur le capteur

POS_INIT : Position pour que la pince soit au-dessus du flacon

- Compteurs

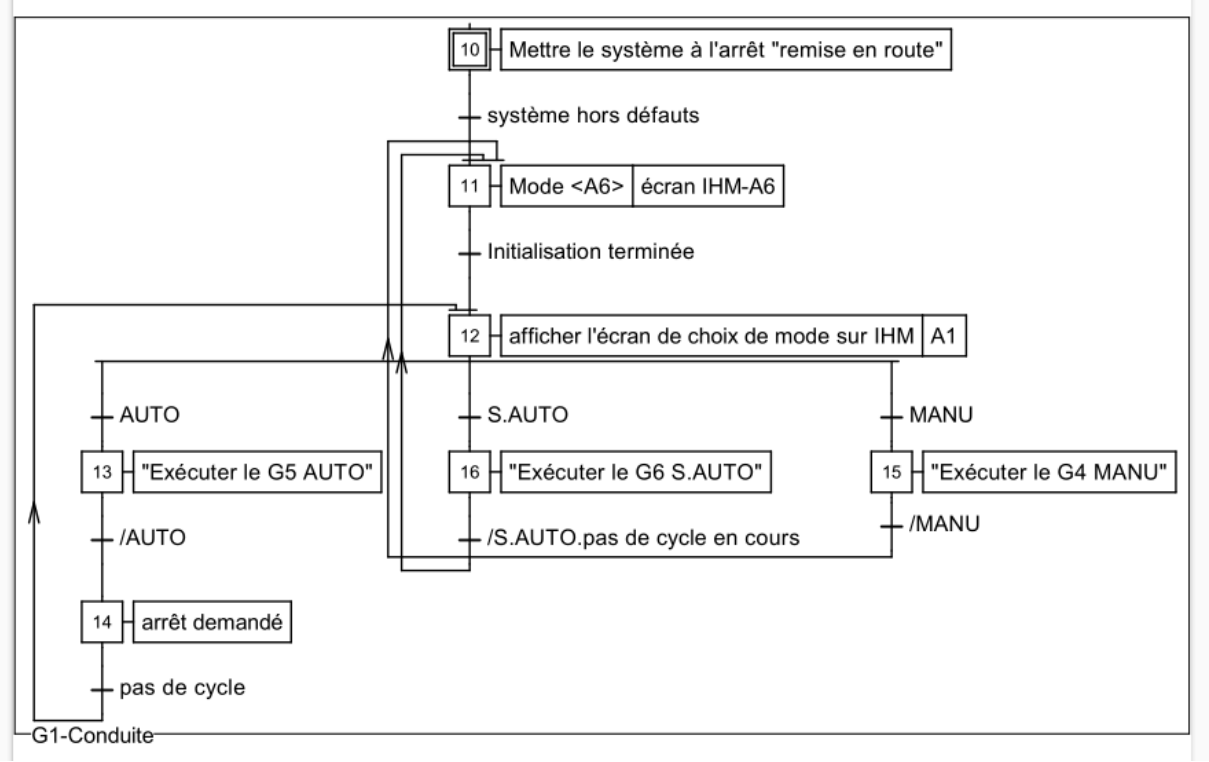
C1 : Compteur de flacons présents dans la boîte

C2 : Compteur de boîtes réalisées

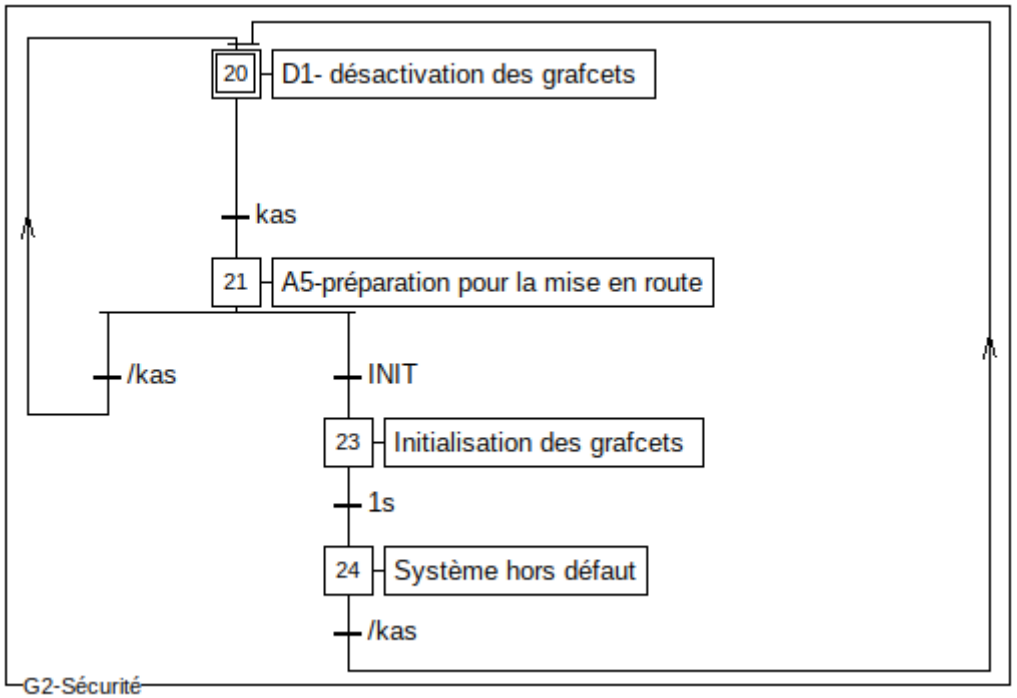
C3 : Compteur de flacons défectueux

POS : Compteur de position désirée du nouveau emplacement du flacon

1. Grafcet de conduite (G1)

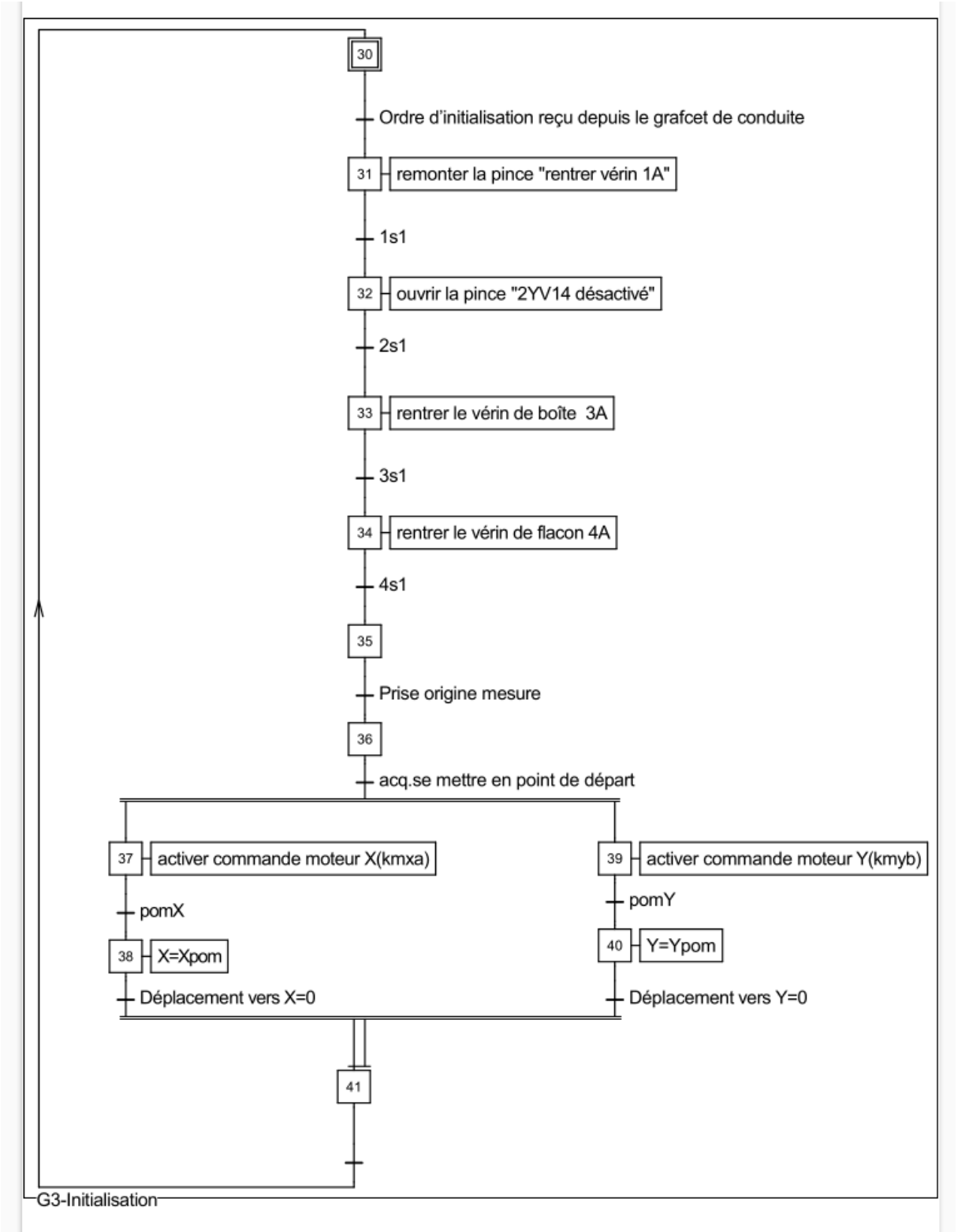


2. Grafcet de sécurité (G2)



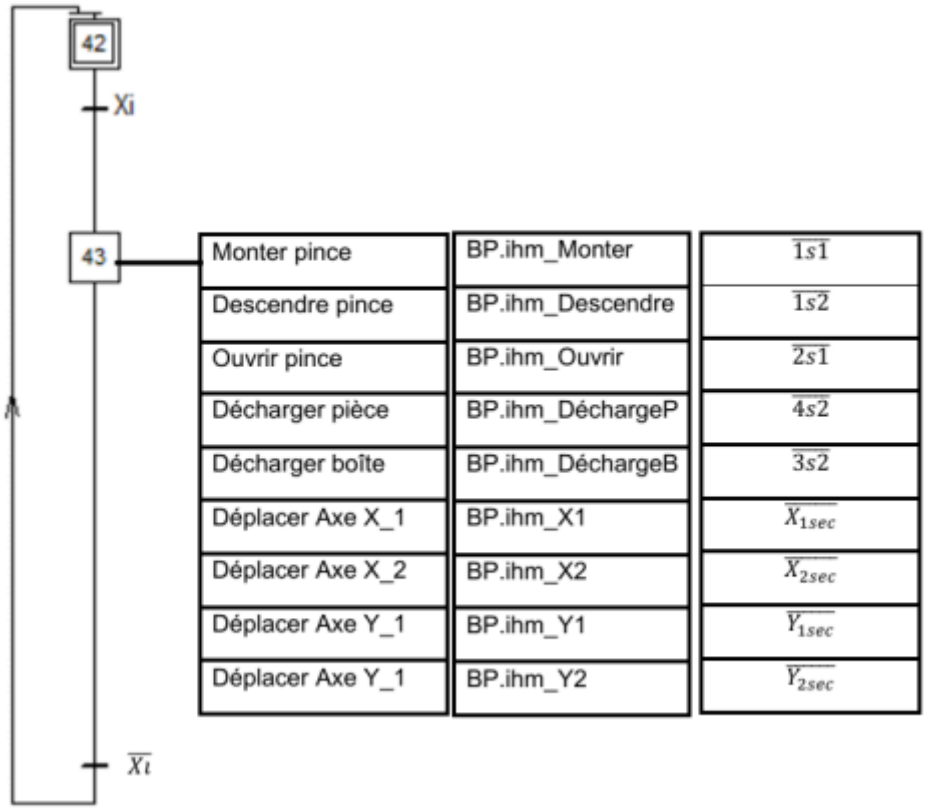
Douae Choubri, Reda El Madani, Salma Letrach, Ayoub Bouquennouche, Souad Ait Bellauali, Achraf Bouchemama, Roua Boughanmi, Amaury Pinéda

3. Grafcet d'initialisation (G3)

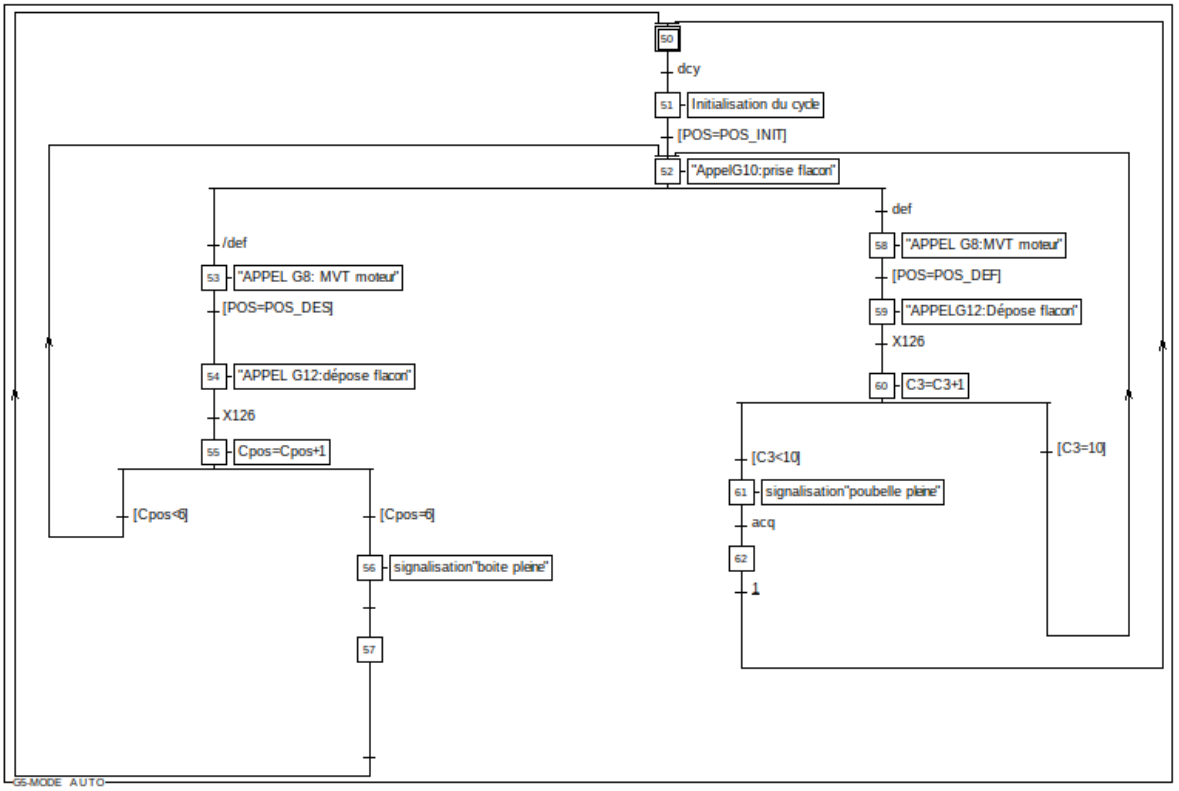


Douae Choubri, Reda El Madani, Salma Letrach, Ayoub Bouquennouche, Souad Ait Bellauali, Achraf Bouchemama, Roua Boughanmi, Amaury Pinéda

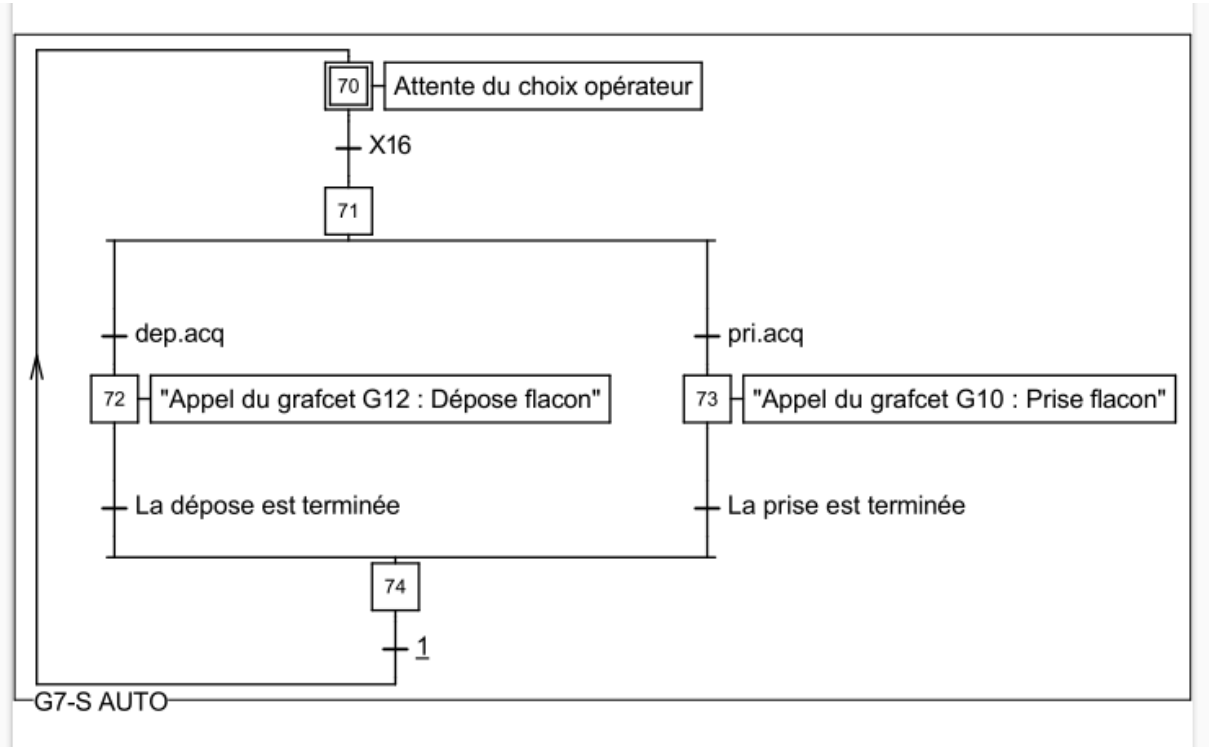
4. Grafcet Commande manuelle (G4)



5. Grafcet Commande auto (G5)

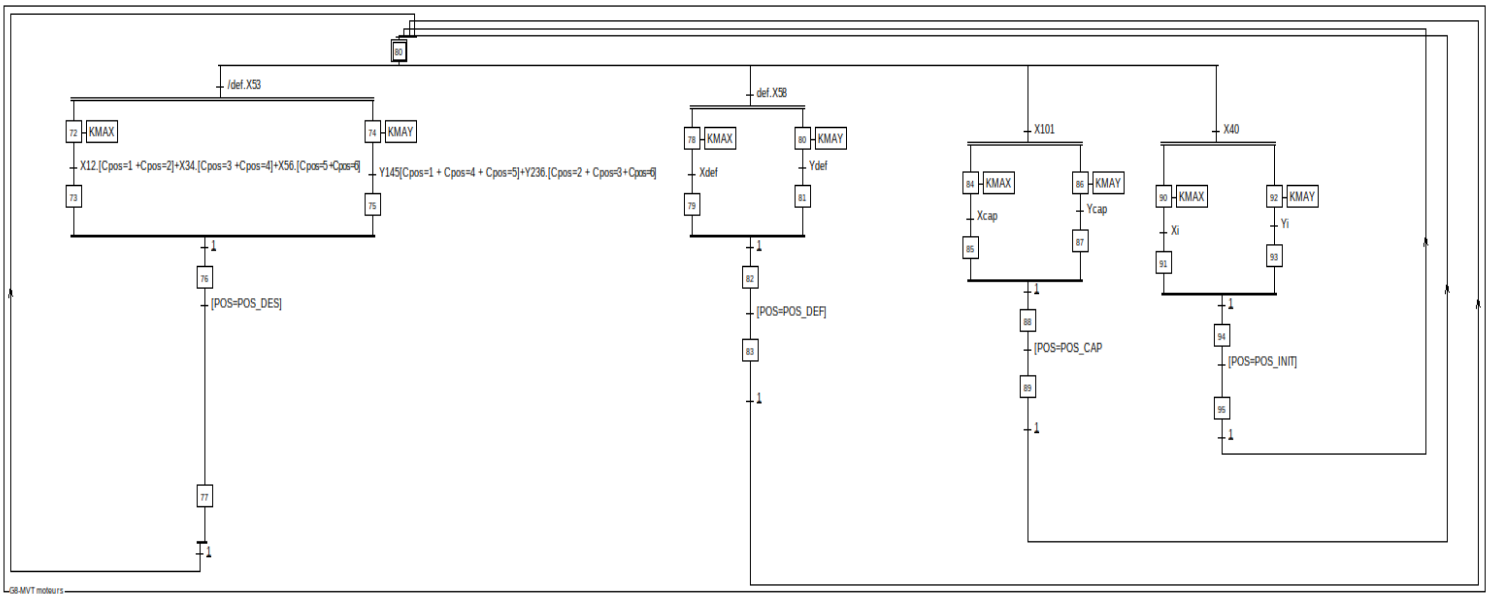


6. Grafcet Commande semi-auto (G7)



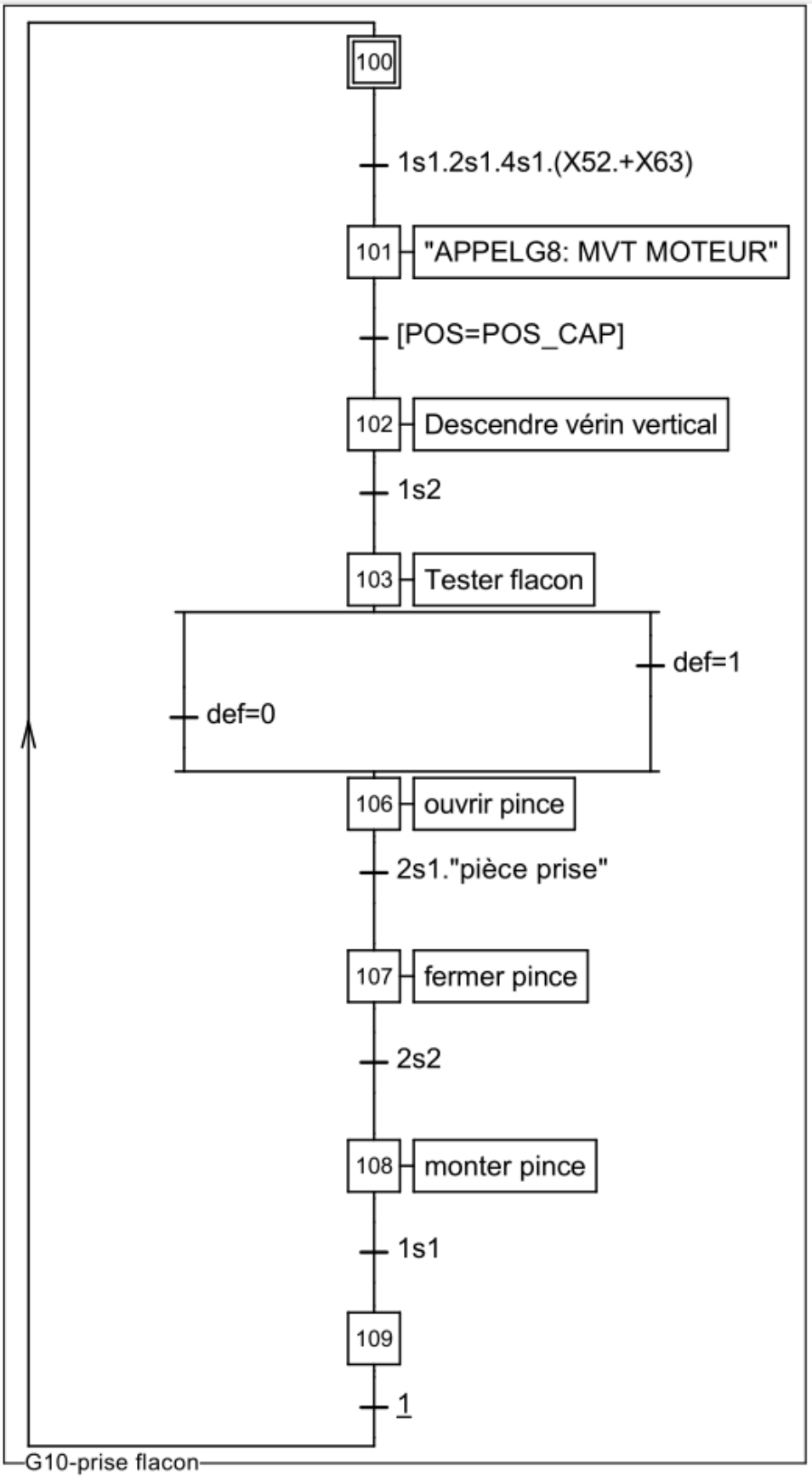
Douae Choubri, Reda El Madani, Salma Letrach, Ayoub Bouquennouche, Souad Ait Bellauali, Achraf Bouchemama, Roua Boughanmi, Amaury Pinéda

7. Grafcet mouvement moteur (G8)



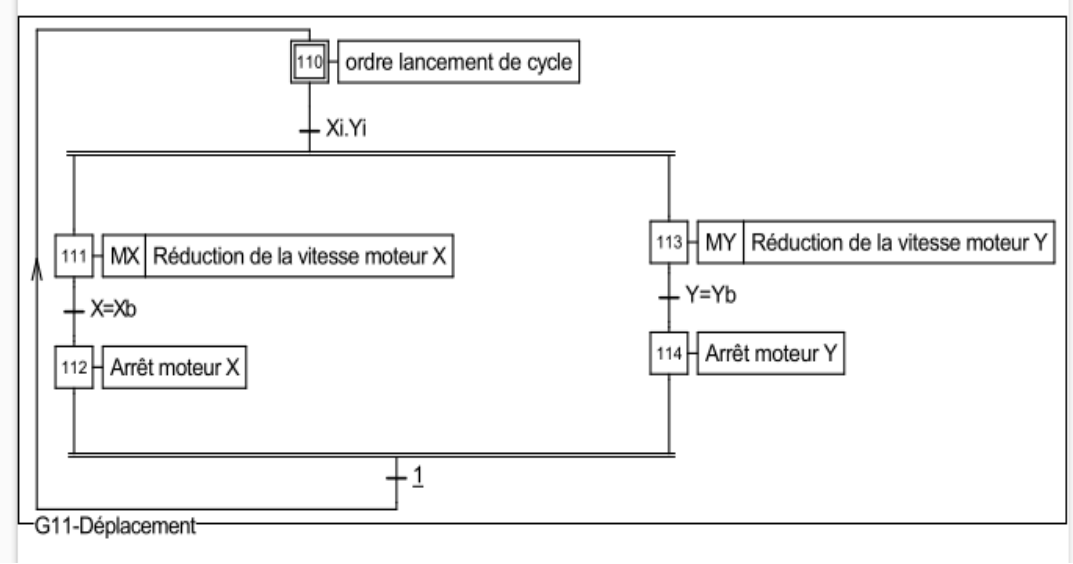
Douae Choubri, Reda El Madani, Salma Letrach, Ayoub Bouquennouche, Souad Ait Bellauali, Achraf Bouchemama, Roua Boughanmi, Amaury Pinéda

8. Grafcet de prise du flacon (G10)

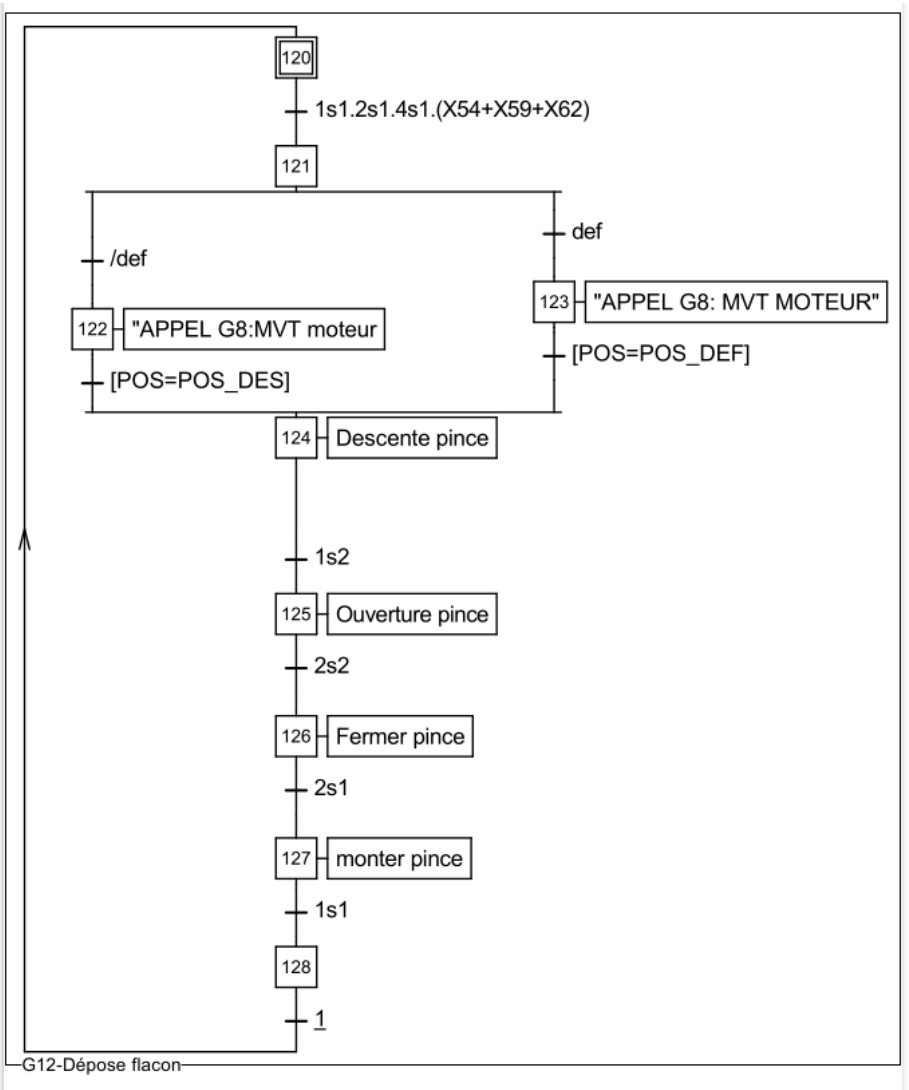


Douae Choubri, Reda El Madani, Salma Letrach, Ayoub Bouquennouche, Souad Ait Bellauali, Achraf Bouchemama, Roua Boughanmi, Amaury Pinéda

9. Grafcet de déplacement (G11)



10. Grafcet de dépose flacon (G12)



Douae Choubri, Reda El Madani, Salma Letrach, Ayoub Bouquennouche, Souad Ait Bellauali, Achraf Bouchemama, Roua Boughanmi, Amaury Pinéda