Polytech Montpellier

Aide Automatisme

Stage découverte de l'entreprise en MEA3

Rapport sur le stage effectué du 06/06/2022 au 04/07/2022

Tuteur de stage : Stéphane Ply

Etudiant : Sébastien Doyez, étudiant en 3 ème année Microélectronique et Automatique,

à Polytech Montpellier.

Remerciement :

Je tiens tout d'abord à remercier la société Aide Automatisme et plus particulièrement Stéphanie Ply, pour avoir cru en mon potentiel et pour m'avoir fait confiance.

Ce stage fut particulièrement pédagogique, j'ai eu la chance de pouvoir travailler en autonomie, ce qui m'a permis d'obtenir une véritable simulation du métier d'ingénieur! Mon tuteur m'a également transmis son expertise lors de nos nombreuses réunions.
Je tiens également à remercier l'équipe enseignante de Polytech Montpellier.
Enfin, je n'oublie pas non plus, mes parents qui m'ont soutenu, sans cesse, lors de l'élaboration de mon projet professionnel.

Du 6 Juin 2022 au 4 Juillet 2022, j'ai effectué un stage au sein de l'entreprise Aide Automatisme, une entreprise spécialisée dans la robotique et l'automatisme, travaillant pour de grands groupes tel que Bayer ou SICOS. Ce fut l'occasion pour moi d'approfondir mes connaissances en robotique et de me confronter à la réalité du monde du travail. Cela m'a également fait comprendre que la robotique est le futur de l'industrie et que de nouvelles technologies continuent de voir le jour.

L'objectif de ce stage était d'effectuer le préprojet d'un robot visseur de flacon de produits cosmétiques pour la SICOS, ainsi que de réaliser une interface HMI, pour permettre aux ouvriers travaillant sur cette ligne, d'utiliser manuellement le bras visseur sans avoir besoin de faire appel à leurs supérieurs voir d'autres entreprises.

Sommaire :

I) Présentation de l'entreprise :

Aide Automatisme est une startup qui a été fondée en 2016, par Stéphane Ply. Cette entreprise est spécialisée dans les automatismes industriels et les machines de conditionnement dans les secteurs pharmaceutiques, chimiques, agro-alimentaires ou encore métallurgiques du nord de la France.

Ils réalisent des consultations d'avant projets, c'est-à-dire les réalisations des plans, la création d'un cahier des charges permettant ainsi de mener à bien la réalisation d'un édifice, mais aussi des suivis de chantier (via des plannings, ils s'occupent de la gestion du matériel avec les fournisseurs et les sous-traitants), ainsi que le dépannage de robots pour certaines entreprises.

Afin de réaliser leurs projets, les employés d'Aide Automatisme utilisent les logiciels de DAO, CFAO tels que SolidWorks ou Autocad, mais aussi les automates provenant de chez Siemens, Schneider Electric ou encore Beckhoff. Dans ce stage, nous avons travaillé sur l'automatisation d'un robot visseur de flacons de parfums de l'entreprise SICOS, basé à Caudry, à une quarantaine de kilomètres de la start-up.

Le secteur de l'automatisme et de la robotique est un secteur en plein essor : les grandes entreprises veulent désormais sécuriser leurs usines via des robots, mais cela va aussi leurs permettre d'augmenter leurs rendements. Aide Automatisme participe alors au réglage de certaines machines, afin que l'entreprise en tire le meilleur profit.

II) Travail effectué lors du stage :

a) Mise en contexte de la mission :

L'entreprise SICOS réalisant des produits cosmétiques, a fait appel à Aide Automatisme afin d'améliorer la fonction d'une machine visseuse de bouchons. En effet, celle-ci étant composée d'une visseuse de 18 kg, casse régulièrement le moteur de montée descente. Notre mission lors de ce stage sera alors de reprogrammer les trois moteurs, afin de manipuler les valeurs des couples des moteurs afin d'assurer un meilleur rendement, et moins de casse moteurs. Pour cela, notre travail se divisera en plusieurs tâches : Dans un premier temps, nous allons passer de la configuration Profinet à Ethercat, puis nous déterminerons de bonnes valeurs de couple, de vitesse et d'accélération, qui nous permettraient un bon fonctionnement de la machine. Ensuite, nous allons créer les blocs fonctions qui vont nous servir à coder les cycles. Et enfin, nous allons programmer le cycle de la visseuse.

b) Les réglages des moteurs afin d'assurer la stabilité sur MacTalk :

L'objectif fixé lors de ce stage est d'automatiser un robot visseur de flacon de produit cosmétique de l'entreprise SICOS. En effet, ce robot n'a jamais réellement fonctionné de manière optimale : il nécessitait l'échange du moteur de montée-descente de la visseuse tous les mois, car celle-ci pesant 18 kg, cela demande un grand effort afin d'assurer les mouvements demandés.

Figure 1 : Robot visseur étudié lors du stage

Lors des premiers jours de ce stage, j'ai tout d'abord dû suivre une formation vidéo dans le but de m'approprier le langage informatique propre à Beckhoff, sur Twincat. En effet, chaque entreprise de robotique crée son propre langage informatique. J'ai tout de même retrouvé des similitudes avec ce que nous avions vu en cours, puisque l'environnement de travail est celui de Visual studio code, que nous avons utilisé notamment lors de notre projet de robotique de manipulation.

Par la suite, nous avons travaillé sur le logiciel MacTalk, qui va nous permettre de communiquer directement avec les moteurs du robot (celui-ci en est équipé de 3 : ceux de la montée- descente de la visseuse et de la pince sont des MAC200, et celui qui permet la rotation de la tête est un MAC 50). Il nous faut donc trouver les meilleures valeurs de couple : le LOAD et le TORQUE, ainsi que les valeurs de la vitesse et l'accélération, qui vont nous permettre d'éviter toute instabilité, mais aussi d'avoir un bon rendement.

Les trois moteurs étant paramétrés en Profinet (standard de communication ouvert pour l'automatisation industrielle), le cahier des charges nous obligeait à passer dans la configuration EtherCat. Nous devions alors changer le firmware des moteurs, ce qui est possible sur MacTalk, via l'onglet update. Puis, nous avons dû régler certains paramètres des moteurs : voici une impression d'écran des différents onglets à modifier du logiciel utilisé :

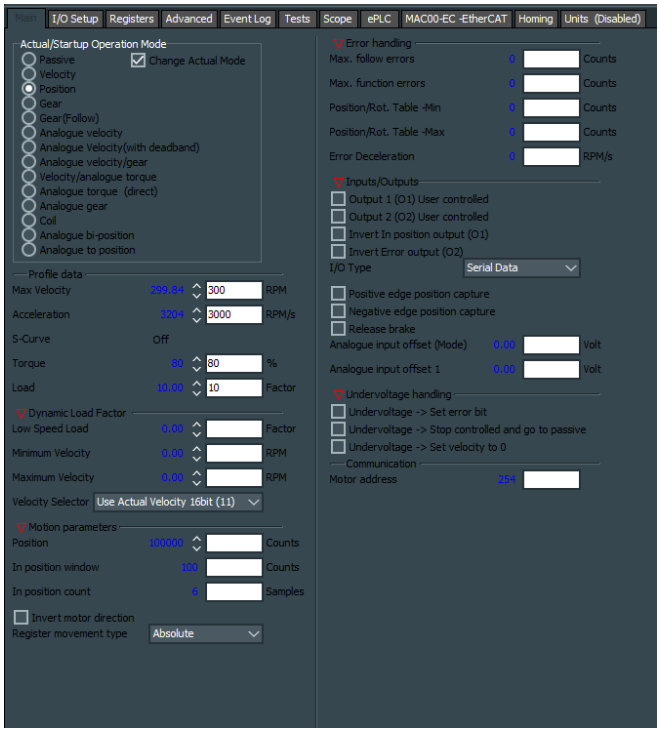


Figure 2 : impression d'écran des paramètres modifiable de MacTalk

Toute la partie « MAC00 EC EtherCat » est non modifiable (« read only »), nous avons utilisé la partie « Test » qui nous a permis de réaliser les réglages effectués ci-dessous :

Tout d'abord réglons le moteur 1 (montée-descente de la pince) :

Grâce au faible poids de la pince, nous n'avons pas eu de difficulté à trouver les valeurs suivantes : celle-ci garantissent donc une bonne vitesse et aucun signe d'instabilité :

Max velocity	-300 RPM
Acceleration	3000 RPM
Torgue	80 %
Load	10

Cependant, on peut entendre des « bruits » lors de la montée de la pince, mais cela est dû au fait que le plastique « frotte » sur son guide…

Réglage du moteur 2 (montée descente visseuse) :

Comme nous l'avions vu, la visseuse est assez lourde, ce qui implique le fait qu'il va falloir donner au moteur un bon couple afin de pouvoir assurer sa tâche. Néanmoins, il y a une instabilité : le moteur vibre énormément… Tout d'abord nous avons essayé les valeurs suivantes :

Max Velocity	100 RPM
Acceleration	1000 RPM/s
Torque	40 %
Load	5

Les instabilités sont encore présentes… nous allons donc doubler le Torque.

Max velocity	200 RPM
Acceleration	2000 RPM/s
Torque	80 %
Load	10

Désormais il n'y a plus de vibration mais la vitesse est lente, ceci va donc nous empêcher de satisfaire la condition de rendement du cahier des charges.

Voici les paramètres nous assurant une bonne vitesse et aucun signe d'instabilité :

Max velocity	500 RPM
Acceleration	5000 RPM/s
Torque	40 %
Load	8

Nous avons également remarqué que le Torque semble jouer un rôle sur les vibrations de la courroie….

Réglage du moteur 3 (rotation de la visseuse) :

Après un réglage assez rapide, nous avons alors déterminé les paramètres nous donnant une bonne vitesse de rotation et aucun signe d'instabilité :

Max velocity	1500 RPM
Acceleration	10000 RPM/s
Torque	100 %
Load	20

Une fois ce travail effectué sur MacTalk, il nous faut alors sauvegarder les paramètres dans la mémoire flash des moteurs.

c) La vue synoptique des actionneurs du robot :

Ensuite, le cahier des charges demande un diaporama de vue synoptique de certaines parties du robot : nous devions faire apparaitre les parties actionneurs : pour se faire, nous avons utilisé le fichier Solidwork fournit avec le robot, et la « datasheet » du bras visseur. Le tuteur du stage nous avait fourni la liste des actionneurs : il ne nous reste plus qu'à les identifier. Cette partie va nous servir à mieux visualiser le robot, et à être plus efficace lorsque nous allons coder les blocs de fonctionnement sur Twincat.

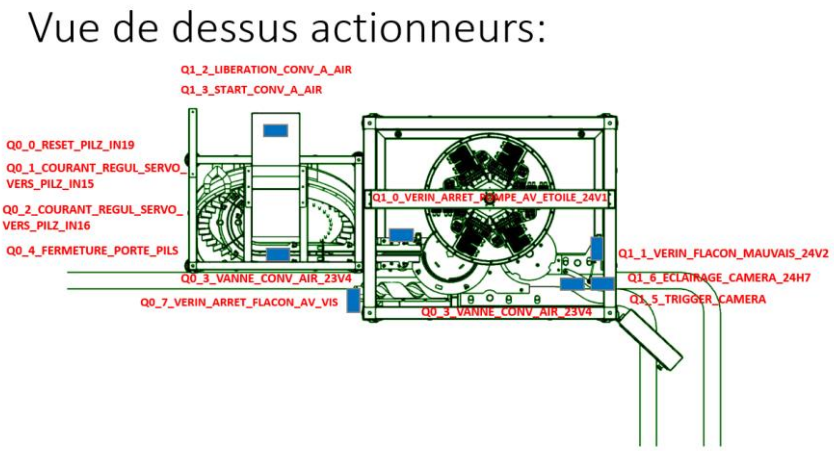


Figure 3 : impression d'écran de la vue synoptique du robot

d) Code pour les fonctionnalités simples du robot

Nous allons alors utiliser le langage de programmation propre à Beckhoff, sur Twincat. L'interface graphique ne m'était pas inconnue car il s'agit de Visual Studio. Ici nous allons coder les fonctions de déverrouillage / verrouillage des moteurs, les commandes en position, en vitesse ainsi que les commandes de mouvements.

Sur Twincat, il existe de nombreux dossiers qui permettent de mieux ranger les parties de code déjà écrit :

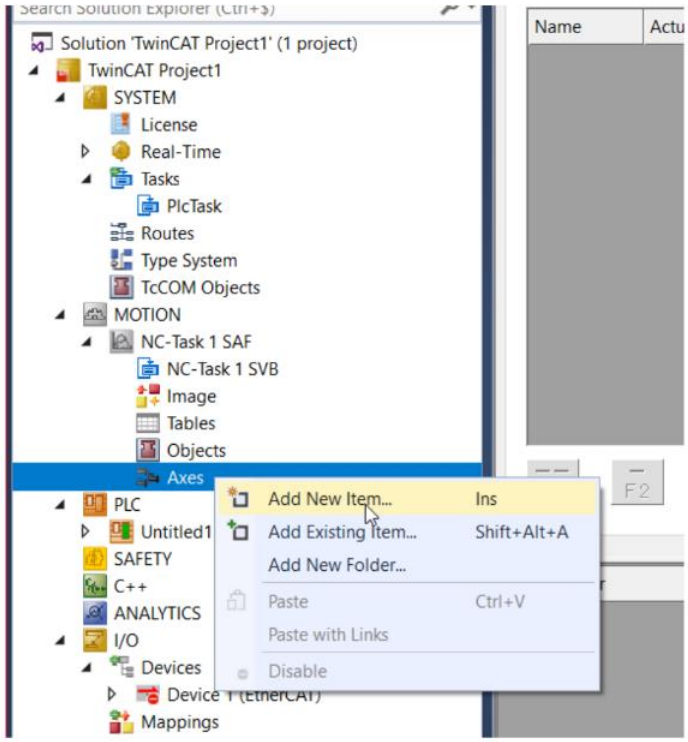


Figure 4 : impression d'écran de l'organisation de Twincat

Tout d'abord la partie MOTION : c'est ici qu'il va nous falloir définir nos axes :

_ Axe1 de la montée/ descente de la pince

_ Axe 2 de la visseuse

_ Axe 3 de la montée/ descente de la visseuse

Par la suite, nous ajouterons un axe virtuel : celui du maître qui aura pour but de gouverner les moteurs. Ensuite, nous devons indiquer des vitesses et des accélérations maximales pour la partie PLC et la partie manuel : il nous faut faire attention de ne prendre que de faibles valeurs afin d'éviter l'endommager le robot lors de nos phases de test. Une fois cette partie paramétrée, nous allons pouvoir coder les HMI des mouvements simples des robots. Twincat a une particularité : on peut

coder en langage structuré (code) ou grâce à des blocs, ce qui présente l'avantage d'être une manière de coder visuelle, différente de tout ce que l'on a déjà vu à l'école.

Les moteurs, lorsqu'ils sont à l'arrêt, sont mis en sécurité : Nous allons donc ajouter les librairies de Tc2, permettant de déverrouiller le moteur :

Nous allons tout d'abord travailler sur la partie montée- descente de la visseuse :

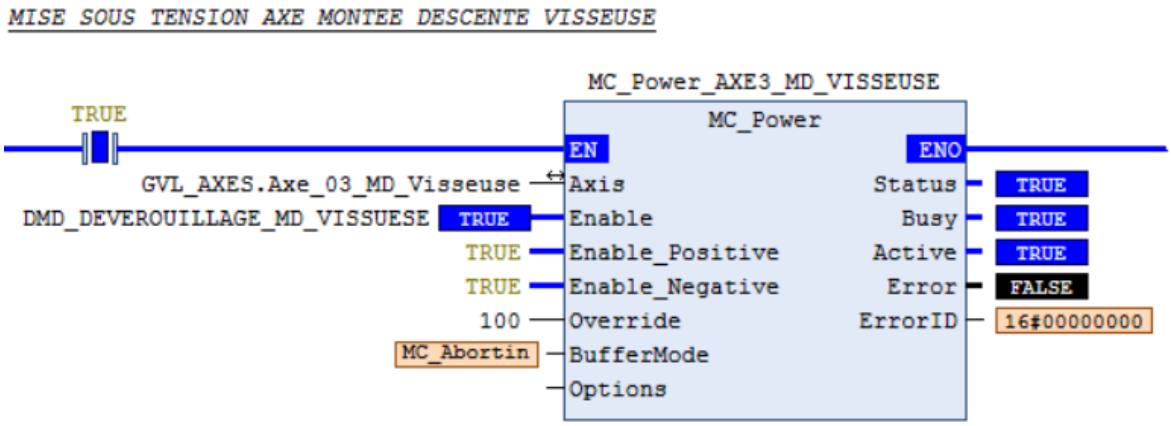
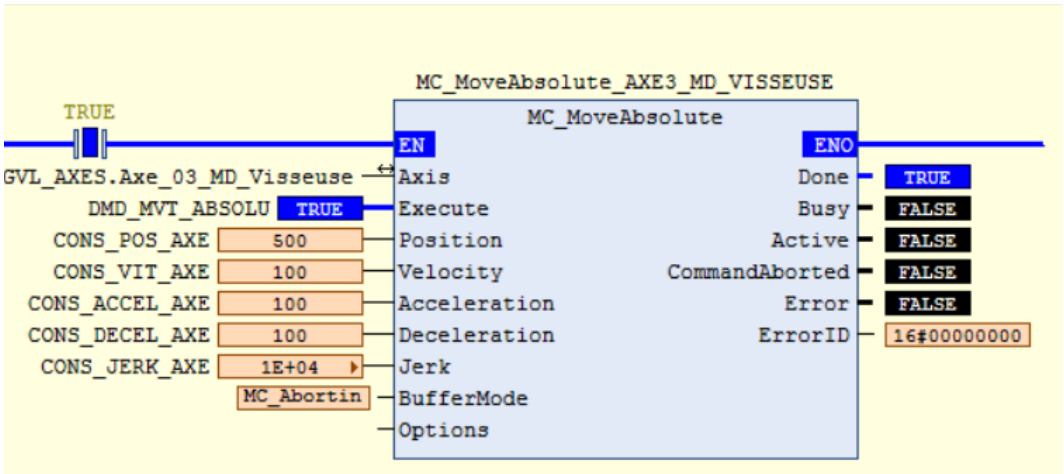


Figure 5 : Bloc permettant la mise sous tension du moteur

Le Bloc nous demande de préciser avec quel moteur nous désirions travailler : il nous faut donc créer une liste de variables globales que nous allons lier à nos axes. Ensuite, il nous faut créer un booléen qui active ou désactive le moteur (DMD_DEVEROUILLAGE_MD_VISSEUSE) : lorsqu'il est « TRUE », le moteur est mis sous tension, sinon il est verrouillé. Sur la partie de gauche, il s'agit des outputs : celle-ci nous indique si l'action a été effectuée, s'il y a eu ou non une erreur et quelle est son identifiant. Nous avons alors effectué ce bloc pour les deux autres axes.

Ensuite réalisons les commandes en vitesse et position de nos moteurs, pour ce faire, nous avons utilisé les blocs MC_MoveAbsolute et MC_MoveVelocity. Ces deux blocs sont commandés par un booléen, de la même manière que le bloc précédemment. Ces deux blocs vont alors commander les moteurs par un échelon en vitesse ou par une position à atteindre…



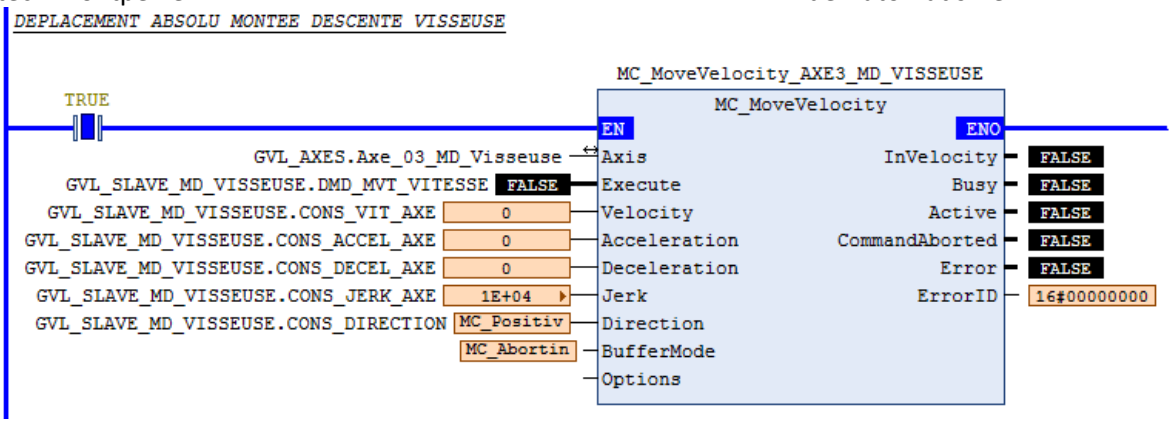


Figure 6 et 7 : blocs de commande en vitesse et en position.

Nous avons créé également des consignes (nommé par « CONS_… »), que nous devons alors déterminer pendant le runtime de la machine (lorsque le code est en exécution). Pour ce qui est du MC_MoveAbsolute, le moteur tourne jusqu'à ce que la position de la visseuse soit atteinte. Maintenant nous allons réaliser les commandes MC_Jog qui vont nous permettre de réaliser une commande « manuelle » de montée-descente ou encore de vissage- dévissage.

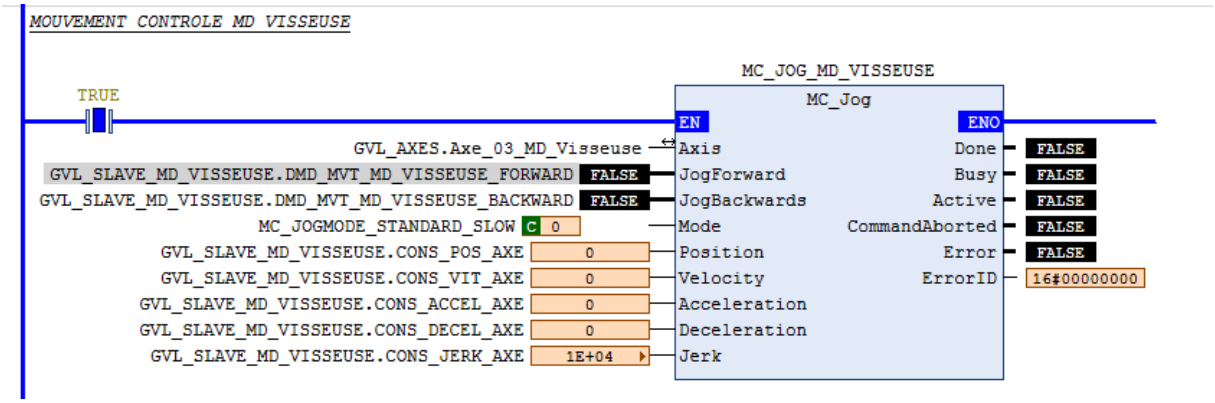


Figure 8 : bloc MC_Jog

Pour la figure 8, j'ai choisi la commande de vissage- dévissage, celle-ci est donc relier à un axe via la variable globale GVL_Axe.Axe2_Visseuse. Il y a ici deux booléens : « DMD_VISSE » et « DMD_MVT_DEVISSE » qui, lorsqu'ils passeront à TRUE, feront tourner le moteur dans un sens ou dans l'autre. Lors de la partie HMI, il faudra faire attention à n'activer qu'un seul à la fois, pour ce faire nous rédigeons quelques lignes de code (Structured Text). Nous retrouvons ici les consignes de positions, vitesse etc. qui ont été définies précédemment.

Reste alors à faire la prise d'origine : le robot a besoin de définir une origine afin de bien coordonner ses mouvements : pour ce faire, des capteurs sont placés en haut de l'axe de montée de la visseuse. Lorsque la visseuse touche le capteur, la prise d'origine est alors faite : Twincat met alors la position à 0.

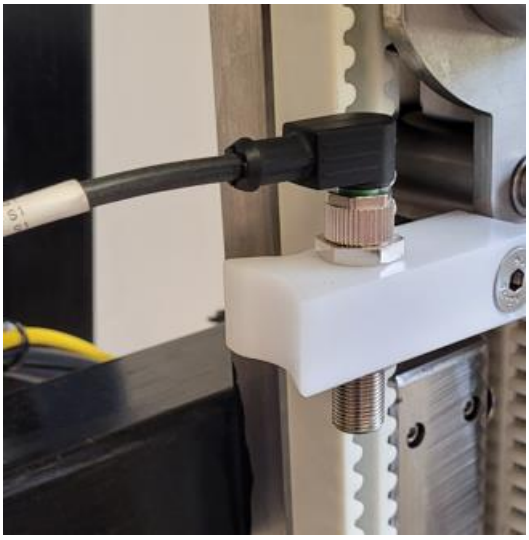


Figure 9 : Capteur pour l'Homing de la visseuse

e) Réalisation d'une HMI pour les mouvements simples des moteurs

Maintenant nous allons créer la partie HMI (Interaction Homme Machine) : le but est de simplifier l'utilisation du robot pour lors de nos manœuvres avec le robot.

Nous avons alors créé 3 pages HMI pour les 3 moteurs ainsi qu'une page d'accueil, où nous disposerons d'un bouton d'arrêt d'urgence.

Tout d'abord, voici la page d'accueil :

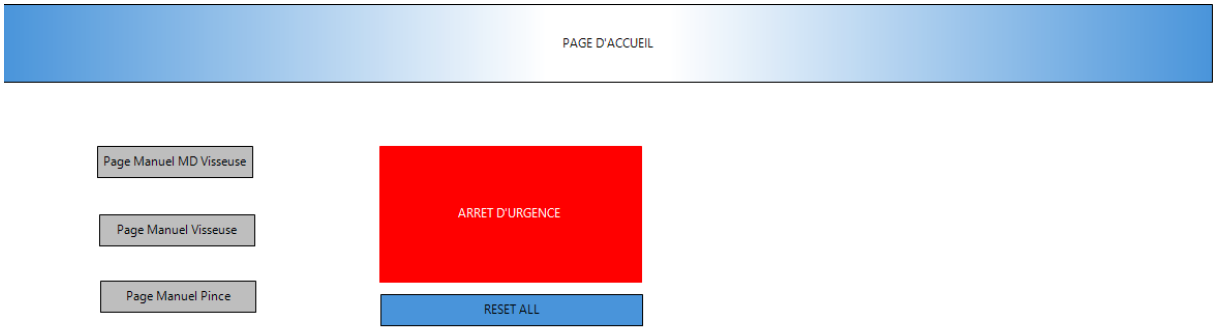


Figure 10_a : Page d'accueil pour HMI simple

Cette page va donc nous servir à aller d'une page de configuration à une autre le plus simplement possible. Lorsque le moteur va à une vitesse trop importante et qu'il dépasse une limite paramétrée, Twincat met le moteur en « défaut », nous avons dû effectuer des reset moteurs régulièrement donc pour nous faire gagner du temps, nous avons mis au point un bouton dans la HMI relié à la fonction bloc MC_Reset. De même, nous avons créé un arrêt d'urgence, qui arrête tous les moteurs sans les mettre en « défaut », ce qui permet donc d'éviter toute casse matériel.

Voici l'interface crée pour la visseuse :

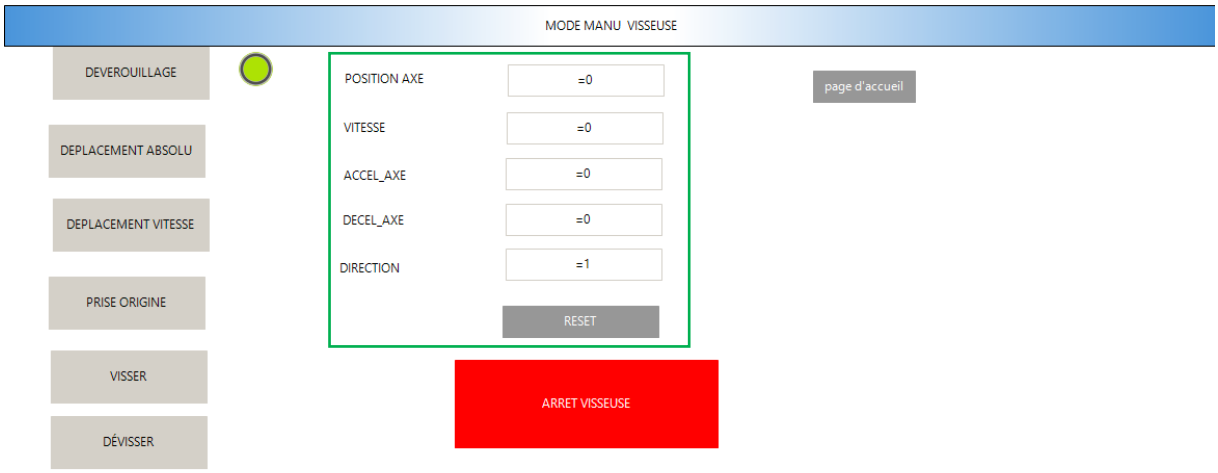


Figure 10_b : Page HMI de la visseuse

Tout d'abord le bouton « DEVEROUILLAGE » est relié avec le booléen « DMD_DEVEROUILLAGE_VISSEUSE », lorsque l'utilisateur clique sur ce bouton, cela réalise un « toggle » (autrement dit un OUEX entre la variable utilisée et 1), ainsi le moteur se déverrouille, ce qui est essentielle afin de réaliser des mouvements. Nous avons également couplé ce bouton avec un voyant qui passe au vert si le moteur est opérationnel. Pour la partie encadrée en vert, l'utilisateur pourra écrire les consignes en vitesse, en position, etc.… Celles-ci sont reliées directement avec les variables du code. De même, lorsque les boutons déplacement vitesse et déplacement position sont activés, cela actionne respectivement les blocs MC_MoveVelocity et MC_MoveAbsolute.

Maintenant pour le vissage et le dévissage, cela est relié au MC_Jog, cependant ici il est impossible d'utiliser un « toggle » car il faut qu'une seule des variables DMD_VISSE et DMD_MVT_DEVISSE soit un TRUE. Nous avons alors réalisé un code ST afin que lorsque l'utilisateur maintient le bouton (click down) sur VISSER, la variable du vissage soit à TRUE et l'autre à FALSE, et les repasse à FALSE si on est en situation « click up ».

f) Couplage des moteurs

Nous devons également créer une commande de couplage : celle-ci a tout d'abord était fait manuellement grâce au bloc MC_Gearin et MC_GearOut qui vont nous permettre de coupler de coupler un moteur avec un autre, (la visseuse pourra alors descendre et visser puis monter et dévisser).

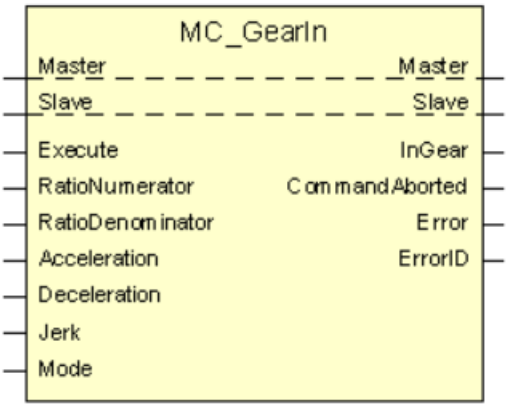


Figure 11 : allure du bloc Gear in

Comme nous pouvons le voir, nous avons besoin d'un maitre, qui sera ici l'axe MD_VISSEUSE et d'un esclave : VISSEUSE. De plus, il nous faut un ratio, lorsque nous avions couplé MD_VISSEUSE à un axe virtuelle, comme nous avons des réducteurs 1/10, il nous a fallu écrire RatioNumerator = 1 et RatioDenomiator = 10, cependant ici nous mettrons un rapport de 1/1. Nous retrouvons alors les consignes d'accélération, décélération et e jerk (dérivée triple de la position, il nous sert à visualiser la brutalité de la mise en mouvement) celles-ci sont définies par l'utilisateur via l'HMI.

g) Profil de Cam :

Nous allons utiliser par la suite MC_Camin et MC_Camout, qui vont nous permettre de déterminer les mouvements à réaliser avec certains axes suivant la position du maitre. Nous retrouvons comme d'habitude un booléen d'exécution, les valeurs d'offset du maitre et de l'esclave que nous avons alors définies à 0, et la graduation (Scaling) qui est défini par un autre bloc MC_CamScalling où l'on rentre l'axe du maître et de l'esclave, ce qui nous permet de calculer directement le scalling, sans passer par des calculs mathématiques compliqués. Une fois ce bloc effectué, il ne nous reste plus qu'à créer la table qui sera reliée à ces blocs.

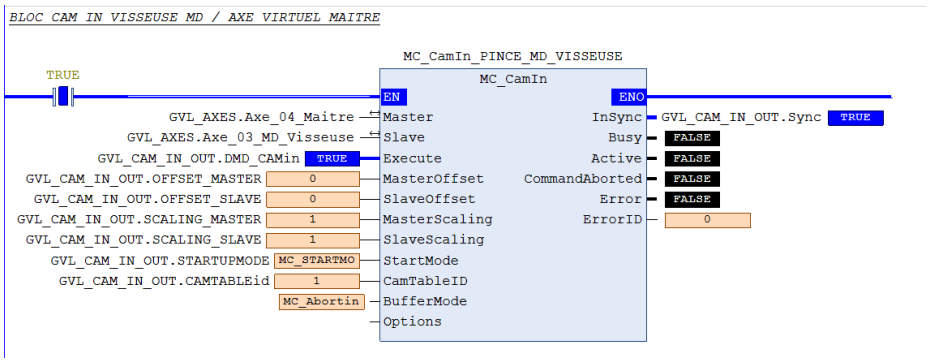


Figure 11 : allure du bloc MC_CamIn

Afin d'assurer le bon fonctionnement de l'axe montée-descente de la visseuse, nous avons besoin de créer un profil de CAM, c'est-à-dire un graphique qui représentera les positions d'un esclave (ici ce sera l'axe « MD_Visseuse ») en fonction des positions d'un maître.

Nous avons créé 4 axes, sachant qu'il y a 3 moteurs, il y a donc un axe qui va nous servir à créer un axe virtuel : ce sera l'axe maître. Comme la machine se présente sous la forme d'un carrousel possédant 6 visseuses, nous avons choisi de configurer l'axe maître en degrés (modulo 360°). Le profil de Cam aura donc l'allure suivante :

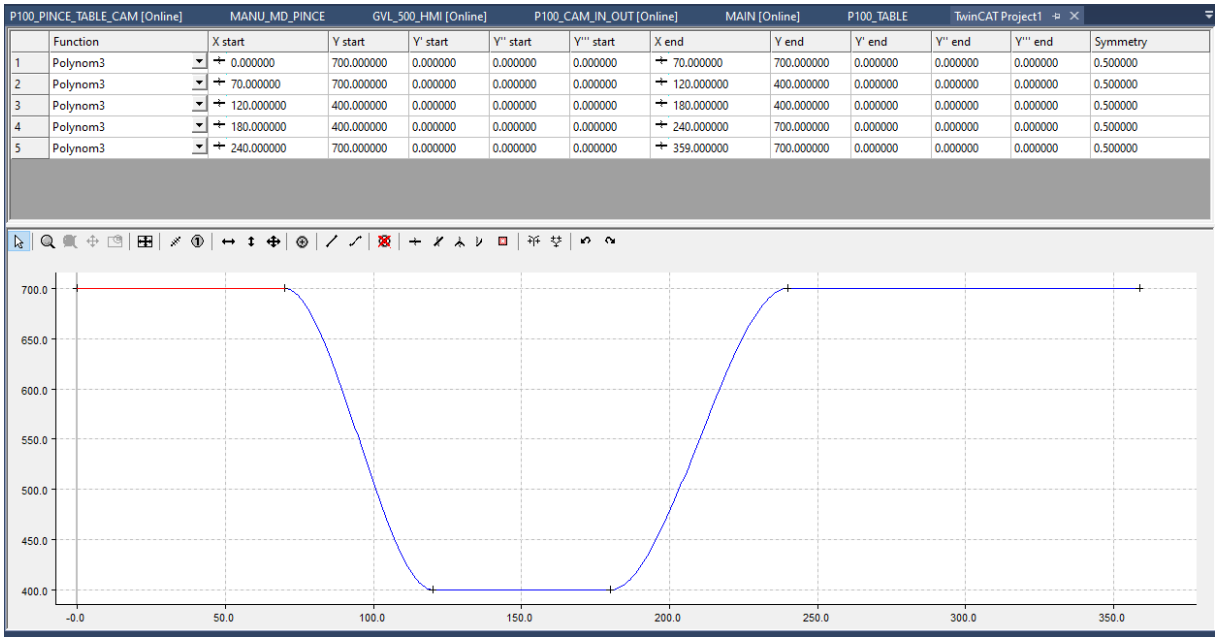


Figure 12: allure de la CAM

Pour ce faire, nous avons besoin de créer une table avec un identifiant id =1, celui-ci va nous permettre de la relier au code de création de CAM. Ensuite, nous allons déterminer quelques points, Twincat reliera alors ces points grâce à un dispositif d'interpolation, nous avons ici choisi d'utiliser des polynômes d'ordre 3, car ils vont nous permettre de réaliser des courbes de positions facile à suivre, et nous pourrons augmenter l'ordre du polynôme si elle est trop brutale pour les moteurs. En annexe, vous retrouverez le code de ce profil de CAM.

h) Programmation du cycle de la visseuse :

Maintenant que nous avons notre profil de CAM, et nos fonctions de base codé pour les axes de montée descente et de rotation de la visseuse, nous pouvons programmer le cycle d'une partie de la machine. Le robot est composé d'un carrousel tournant avec 6 visseuses afin d'assurer le meilleur rendement possible. Lors de la mise sous tension, les visseuses doivent faire un homing (une prise d'origine), puis doivent se mettre à la position du début de CAM et attendre un changement au niveau de la courbe de CAM (créé précédemment). Puis elles doivent suivre l'évolution en position imposée par le graphique, lorsqu'elles sont arrivées au minimum de la position, la visseuse doit tourner.

Pour ce faire, nous allons créer un nouveau POU qui sera appelé dans la fonction main. Nous avons choisi de coder cela sous la forme d'un Switch case (voir annexe pour voir le code en entier).

Tout d'abord, nous avons configuré une variable globale qui effectuera des actions suivant les valeurs qu'elle prend : lorsqu'elle vaut 0 : on crée la CAM ; à 10, on réalise le homing…

Vu que nous avons déjà détaillé la création de la CAM, nous nous intéresserons plus particulièrement dans cette partie aux étapes suivantes. Le homing sera réalisé qu'une seule fois, la machine sera « bloqué » dans une boucle infini dans les étapes 30 à 50… Il faut donc créer pour l'étape 20, un déplacement qui nous place à la position de départ de l'esclave de la CAM : nous utiliserons pour se faire un MC_MoveAbsolute, qui va nous placer à la position 700. Puis nous allons réaliser le CAM in (nous allons coupler les axes « Maître » et « MD_visseuse ». Pour ce faire, nous déverrouillons les deux axes, puis nous réalisons le couplage.

```
// déverrouillage de MD visseuse :
    GVL_SLAVE_MD_VISSEUSE.DMD_DEVEROUILLAGE_MD_VISSEUSE := TRUE;
// couplage du maitre et  de la MD visseuse:
    GVL_CAM_IN_OUT.DMD_CAMin := TRUE;
// définition d'un couple max pour la visseuse :
//étape suivante...
    IF GVL_CAM_IN_OUT.Sync (* AND GVL_CAM_IN_OUT.Sync_visseuse*)  THEN
        Etape := 40;
    END_IF
```

Figure 13 : copie d'écran d'une partie du code

Maintenant, on doit passer à l'étape 40 : un déplacement en vitesse de l'axe virtuelle : la position du maître varie, or l'axe « MD_visseuse » étant relié par un profil de CAM au maître, la visseuse va suivre la courbe que nous avons vue dans la partie g. Maintenant la visseuse doit se mettre en marche lorsque la position minimum est atteinte : nous allons donc utiliser un autre POU, que nous appellerons dans la fonction MAIN. Celui-ci sera également un Switch case comprenant 2 cas : un cas où la visseuse est à l'arrêt et un second, où la position est comprise entre 120 et 180, et où le robot visse. Dans cette étape, nous utiliserons alors la fonction Beckhoff « MC_MoveRelative », qui va nous permettre de faire un certain nombre de tours de la visseuse à chaque fois que cette fonction est appelée.

Maintenant que nous avons programmé notre cycle, ajoutons des booléens de « sécurité » : Nous avons donc ajouté une étape 45 qui va nous servir à arrêter les moteurs : ici nous allons utiliser la fonction MC_Stop. On va donc créer la variable «GVL_000_MODE_DE_MARCHE.MARCHE_CYCLE », lorsqu'elle passe à FALSE, cela signifie qu'il faut sortir du cycle. Cependant il faut que le maître soit en fin de cycle afin d'éviter que le cycle reprenne si la variable est remise à TRUE lors de l'étape 0 : on rajoute donc une condition supplémentaire à notre boucle : il faut que l'angle actuel du maître soit supérieur à 359° ou inférieur à 1°. Une fois la condition validée, il faut également découpler les moteurs, on utilisera alors la fonction « Cam_out ».

i) Les compétences acquises :

Au cours de mon stage, j'ai découvert un nouveau langage de programmation robotique qui est celui propre à l'entreprise Beckhoff, même si celui si avait la même interface graphique que Visual Studio, l'organisation est différente : nous avons différentes catégories comme la « motion » qui permet de travailler directement sur les axes ou encore la « PLC » où on peut écrire du code dans le dossier « POU ». J'ai également travaillé sur un exemple concret d'instabilité en robotique, ce qui m'a fait comprendre l'intérêt d'être le plus loin possible du point critique et d'avoir des marges de phase et de gain importantes.

J'ai également travaillé sur les interfaces HMI, qui ont pour but de simplifier au mieux l'usage de la machine, sans avoir à regarder le code.

J'ai également compris l'intérêt de bien commenter le code : il faut qu'il soit facile pour une personne n'ayant pas travaillé sur le projet, de comprendre le code, et de pouvoir aider l'individu qui a programmé s'il n'arrive pas à résoudre un problème…

Conclusion : Ce stage m'a apporté une vision concrète du métier d'ingénieur, j'ai travaillé sur un nouveau langage informatique : celui de Beckhoff, sur l'aspect automatisation d'une machine, qui a pourtant été construite par une autre entreprise. J'ai créé des fonctionnalités HMI simples, afin d'activer et d'exécuter certaines tâches. Ceci m'a également fait prendre en conscience l'impact et la dangerosité que peut avoir une instabilité. Un autre point positif de ce stage : j'ai été en autonomie une grande partie du temps, ce qui m'a permis d'avoir une réelle simulation du travail d'ingénieur.

Le secteur de l'automatisme et de la robotique est en plein essor : les grandes entreprises souhaitent augmenter la sécurité des ouvriers, diminuer la pénibilité et également avoir un meilleur rendement. De nouvelles technologies voient le jour, comme les XTS, qui semble être le futur des actionneurs de robots : elles reposent sur la force magnétique et permettent de mettre en mouvement certaines parties tout en limitant le frottement de ces pièces. A la fin de mon stage, j'ai eu la chance d'assister à une formation Twincat sur ces nouvelles technologies, qui m'ont ouvert les yeux vers de nouvelles possibilités.

Annexe :

Annexe 1 : code d'un cycle (Maître / MD visseuse) :

```
PROGRAM P100_TABLE
VAR
    // déclaration de la CAM
    CAM1: MC_CAM_REF;
    // création d'une table:
    Nb_ligne_table              :UDINT  := 6;
    Table_points_1              : ARRAY[1..6] OF Table;
    PointsCam1                  : ARRAY [1..6] OF MC_MotionFunctionPoint;

    //paramètre de la table:
    CPT_POINT                   : INT;
    MC_CamTableSelect1          : MC_CamTableSelect;
    CamID                       : UDINT := 1;
    Ecriture_Cam                : BOOL;
    // déclaration des varaibles utilisées dans le CASE
    Etape                       : INT;
    Option                      : STRING(INT#4);
    // déclaration des fonctions blocs utilisées
    EncoderIndex                : ST_SetPositionOptions;
    MC_SetPosition_maitre       : MC_SetPosition;
    MC_SetPosition_MD_visseuse  : MC_SetPosition;

    stop                        : BOOL := FALSE;
    Angle_actuel_maitre: LREAL;
    Vitesse_actuelle_maitre: LREAL;
    Pos_actuelle_MD_Visseuse: LREAL;
END VAR

Angle_actuel_maitre:=GVL_AXES.Axe_04_Maitre.NcToPlc.ModuloActPos;
Vitesse_actuelle_maitre:=GVL_AXES.Axe_04_Maitre.NcToPlc.ActVelo;
Pos_actuelle_MD_Visseuse:=GVL_AXES.Axe_03_MD_Visseuse.NcToPlc.ActPos;


CASE Etape OF
    // Etape 0 ************************
        // remplissage de la CAM1:
    0:
        // RAZ DES COMMANDES
        GVL_SLAVE_MD_VISSEUSE.DMD_MVT_ABSOLU := FALSE;
        GVL_SLAVE_MD_VISSEUSE.DMD_DEVEROUILLAGE_MD_VISSEUSE := FALSE;
        GVL_SLAVE_MD_VISSEUSE.DMD_PRISE_ORIGINE :=FALSE;
        GVL_MAITRE.DMD_MVT_VITESSE_MAITRE :=FALSE;
        GVL_VISSEUSE.DMD_POWER_VISSEUSE := FALSE;
        GVL_CAM_IN_OUT.DMD_CAM_OUT := TRUE;
        GVL_MAITRE.DMD_MAITRE_POWER := FALSE;
        GVL_CAM_IN_OUT.DMD_CAMin := FALSE;
        GVL_CAM_IN_OUT.DMD_CAMin_visseuse := FALSE;
        GVL_MAITRE.DMD_STOP_MAITRE :=FALSE;
        GVL_PARAM_CYCLE.bool_maitre_verif := TRUE;
        /// RAZ du switch de la visseuse :
        GVL_VISSEUSE.etape_visseuse := 0;

        // Préparation des consignes de la cam
        CAM1.pArray := ADR(PointsCam1);
        CAM1.ArraySize := SIZEOF (PointsCam1);
        CAM1.TableType := MC_TABLETYPE_MOTIONFUNCTION;
        CAM1.NoOfRows := Nb_ligne_table ;
        CAM1.NoOfColumns:= 1;

        // première série de points dans la table
        // valeur à renseigner
        Table_points_1[1].POS_Y:=700;
        Table_points_1[2].POS_X:=70;
```

```
    Table_points_1[2].POS_Y:=700;
    Table_points_1[3].POS_X:=120;
    Table_points_1[3].POS_Y:=400;
    Table_points_1[4].POS_X:=180;
    Table_points_1[4].POS_Y:=400;
    Table_points_1[5].POS_X:=240;
    Table_points_1[5].POS_Y:=700;
    Table_points_1[6].POS_X:=359;
    Table_points_1[6].POS_Y:=Table_points_1[1].POS_Y;

    // ON REMPLIT LE TABLEAU PointsCam1
    FOR CPT_POINT:=1 TO 6 DO
        PointsCam1[CPT_POINT].PointIndex := INT_TO_UDINT (CPT_POINT); // index donné avec l'indice du for
        PointsCam1[CPT_POINT].FunctionType := MOTIONFUNCTYPE_POLYNOM3; // choix du polynome d'interpolation ordre 3
        PointsCam1[CPT_POINT].PointType :=  MOTIONPOINTTYPE_REST;
        PointsCam1[CPT_POINT].RelIndexNextPoint := 0;
        // remplissage de la CAM avec la table
        PointsCam1[CPT_POINT].MasterPos :=Table_points_1[CPT_POINT].POS_X;
        PointsCam1[CPT_POINT].SlavePos := Table_points_1[CPT_POINT].POS_Y;
    END_FOR

    //Creation - Ecriture de la table 1
    // écriture des paramètre de MC_CamTableSelect
    IF Ecriture_Cam THEN
        MC_CamTableSelect1.Execute := TRUE;
        MC_CamTableSelect1.Periodic := TRUE;
        MC_CamTableSelect1.MasterAbsolute := FALSE;
        MC_CamTableSelect1.SlaveAbsolute := TRUE;
        MC_CamTableSelect1.CamTableID := CamID;
    ELSE
        MC_CamTableSelect1.Execute := FALSE;
    END_IF
    // reste à ajouter l'axe du maitre, de l'esclave et la CamTable
    MC_CamTableSelect1(Master:=GVL_AXES.Axe_04_Maitre, Slave:=GVL_AXES.Axe_03_MD_Visseuse, CamTable:=CAM1);
        GVL_MAITRE.DMD_MVT_VITESSE_MAITRE :=FALSE;
    // si MC_CamTableSelect1.busy et pas erreur etape 10
    IF  GVL_000_MODE_DE_MARCHE.MARCHE_CYCLE AND
        (Pos_actuelle_MD_Visseuse>Table_points_1[1].POS_Y+1
        OR Pos_actuelle_MD_Visseuse<Table_points_1[1].POS_Y-1)  THEN
        Ecriture_Cam:=FALSE;
        Etape := 10;
     END_IF
     IF  GVL_000_MODE_DE_MARCHE.MARCHE_CYCLE AND
        (Pos_actuelle_MD_Visseuse<=Table_points_1[1].POS_Y+1
        AND Pos_actuelle_MD_Visseuse>=Table_points_1[1].POS_Y-1)  THEN
        Ecriture_Cam:=FALSE;
        Etape := 30;
     END_IF
// Etape 10 ************************
    // prise de l'origine de MD visseuse
10:
    // Prise d'origine de l'axe
    GVL_SLAVE_MD_VISSEUSE.DMD_DEVEROUILLAGE_MD_VISSEUSE := TRUE;
    GVL_SLAVE_MD_VISSEUSE.DMD_PRISE_ORIGINE :=TRUE;
    GVL_SLAVE_MD_VISSEUSE.CONS_POS_HOMING:=780;
    // Homing du maitre (MC_set position) à la valeur de début de la cam (0°)
    //déverrouillage du maitre pour le homing
    GVL_MAITRE.DMD_MAITRE_POWER := TRUE;
    MC_SetPosition_maitre(Axis := GVL_AXES.Axe_04_Maitre , Execute := TRUE,  Position := 0, Mode := TRUE, Options := EncoderIndex);
    // étape suivante...
    IF GVL_SLAVE_MD_VISSEUSE.MC_HOME_DONE AND MC_SetPosition_maitre.Done THEN
        // revérouillage pour le maitre
        MC_SetPosition_maitre(Axis := GVL_AXES.Axe_04_Maitre , Execute := FALSE,  Position := 0, Mode := TRUE, Options := EncoderIndex);
        GVL_SLAVE_MD_VISSEUSE.DMD_PRISE_ORIGINE :=FALSE;
        Etape := 20;
    END_IF;
// Etape 20 ************************
20:
    // Deplacement absolu de l'esclave a la position de depart de la came

    GVL_SLAVE_MD_VISSEUSE.CONS_POS_AXE:= Table_points_1[1].POS_Y;
    GVL_SLAVE_MD_VISSEUSE.CONS_VIT_AXE:=10;
    GVL_SLAVE_MD_VISSEUSE.DMD_MVT_ABSOLU := TRUE;

    // étape suivante...
    IF GVL_SLAVE_MD_VISSEUSE.MC_moveabs_DONE  THEN
        GVL_SLAVE_MD_VISSEUSE.DMD_MVT_ABSOLU := FALSE;
        Etape := 30;
```

```
        END_IF
    // Etape 30 *************************
    30:
    // Étape de Couplage de l'eclave (CAM IN)
        // remet CAM out à false pour éviter de découpler
        GVL_CAM_IN_OUT.DMD_CAM_OUT :=FALSE;
    // dévérouillage du maitre
        GVL_MAITRE.DMD_MAITRE_POWER :=TRUE;

    // déverrouillage de MD visseuse :
        GVL_SLAVE_MD_VISSEUSE.DMD_DEVEROUILLAGE_MD_VISSEUSE := TRUE;
    // couplage du maitre et  de la MD visseuse:
        GVL_CAM_IN_OUT.DMD_CAMin := TRUE;
    // définition d'un couple max pour la visseuse :
    //étape suivante...
        IF GVL_CAM_IN_OUT.Sync (* AND GVL_CAM_IN_OUT.Sync_visseuse*)  THEN
            Etape := 40;
        END_IF

    // Etape 40 *************************
    40:
    //  test si on est dans le tout premier cycle ou pas:
    IF GVL_PARAM_CYCLE.bool_maitre_verif THEN
        // ici, on est dans le premier cycle:
        // Consigne de déplacement vitesse du maitre
        IF GVL_MAITRE.CONS_VIT_AXE_MAITRE=0 THEN
        GVL_MAITRE.CONS_VIT_AXE_MAITRE := 10;
        END_IF;
        GVL_MAITRE.CONS_ACCEL_AXE_MAITRE := 100;
        GVL_MAITRE.CONS_DEC_MAITRE := 100;
        GVL_MAITRE.CONS_JERK_AXE_MAITRE := 1000;

        IF GVL_MAITRE.MAITRE_READY (*AND GVL_VISSEUSE.VISSEUSE_READY *) AND GVL_SLAVE_MD_VISSEUSE.MD_visseuse_ready   THEN
            GVL_MAITRE.DMD_MVT_VITESSE_MAITRE :=TRUE;
        END_IF

    END_IF


        GVL_MAITRE.DMD_READ_POS_MAITRE := TRUE;


        GVL_PARAM_CYCLE.bool_maitre_verif := FALSE;
        IF GVL_000_MODE_DE_MARCHE.MARCHE_CYCLE=FALSE
            AND (Angle_actuel_maitre>359 OR Angle_actuel_maitre<1)  THEN

            Etape := 45;
        END_IF
        // Etape 45 *************************
    45:
        // Arret vitesse du maitre
        GVL_MAITRE.DMD_STOP_MAITRE:=TRUE;
            GVL_MAITRE.CONS_DEC_MAITRE :=100;
        GVL_MAITRE.CONS_JERK_MAITRE := 1000;
        IF Vitesse_actuelle_maitre<1 AND P100_AXE_4_MAITRE.MC_Stop_MASTER.Done THEN
                GVL_MAITRE.DMD_STOP_MAITRE:=FALSE;
                Etape := 50;
        END_IF



    // Etape 50 *************************
    50:
    // DESYNCHRO DE LA MD VISSEUSE
        GVL_CAM_IN_OUT.DMD_CAM_OUT:=TRUE;
        GVL_CAM_IN_OUT.DMD_CAMin:=FALSE;
        GVL_MAITRE.DMD_STOP_MAITRE := TRUE;

        IF P100_CAM_IN_OUT.CAMOUT_MAITRE_MD_VISSEUSE.Done THEN
            GVL_CAM_IN_OUT.DMD_CAM_OUT:=FALSE;
            Etape := 0;
        END_IF

        ///Etape := 60;
END_CASE
```