



POLYTECH SORBONNE
SPÉCIALITÉ : ELECTRONIQUE ET INFORMATIQUE
- SYSTÈMES EMBARQUÉS (EISE)

RAPPORT DE STAGE DE 4ÈME ANNÉE

**Stage Technique, Assistant
Ingénieur, Weeroc**



Dates:
3 Juin 2019 - 2 Août 2019

Auteur:
Raphaël BOUILHOL

Année Universitaire 2018-2019

Abstract

Weeroc is a fabless company that designs **ASICs** (*Application Specific Integrated Circuit*) specialized in photo-detectors read-out for the fields of Scientific Instrumentation, Aerospace and Medical Imagery. Each ASIC, once received, has to go through an automated series of tests to certify its proper functioning.

To do so, the test operator has to place the ASIC on a test-board, launch the test which lasts about 5 minutes and write the results on an Excel file.

Because running the tests disrupt the operator in doing his work, and the quantities are becoming too large for by-hand work (several hundreds), I have been tasked to modify a **CNC** machine in order to fully automate the tests.

The system had to be able to read the chip serial number, pick-up the ASIC and transport it to the test-board, apply pressure on the chip, run the test, and sort the ASIC depending on the test results.

Because the test software was written in **C#** using Visual Studio, it was decided to also use **C#** to program the system.

The CNC machine operates from commands in the **G-Code** language, sent from a computer to a modified **Arduino** board that controls the motors, allowing an arm to move along 3 axis.

To move the robot, we simply needed to send the cartesian position through a G-Code command like G00XxxxYyyyZzzz, from the **C#** software to the CNC.

Next for the serial number we used a camera wired to another Arduino board to take pictures. Those pictures are then processed and given to the OCR algorithm **Tesseract**, which give us the number.

To move the ASIC, we used an Air Pump, connected with a suction cup through a tube. The pump powering was controlled also by an Arduino board, via a NPN Transistor. Finally, we used a linear actuator screwed to the back of the X axis rail to apply pressure on the ASIC. By reading its current consumption and putting a threshold on it, we were able to stop its progress once it had apply the wanted pressure.

All of this constitute our complete system, which by the end of the internship was able to fully perform test cycles.

Remerciements

Je souhaiterais remercier l'intégralité de l'équipe de Weeroc pour l'excellente opportunité de stage qu'ils m'ont offert, pour leur générosité, leur bonne humeur... En particulier, je voudrais remercier:

- **Julien, Salleh et Florent** pour leur vision, leur disponibilité et leur aide précieuse, ce fut un plaisir d'apprendre à vos côtés;
- "Docteur" **JB** sans qui l'open space aurait été beaucoup trop calme;
- **Sabrina**, dans les mots que je partage à 100% d'un précédent stagiaire bien connu, "C'est une vraie crème cette femme";
- **Samir, Valentin, Hugo et Sébastien** pour leur compagnie de qualité supérieure.

Ils ont su créer un environnement de travail extrêmement sympathique et motivant, et ce fut un véritable plaisir d'avoir pu faire partie de leur équipe pendant une courte période.

Je remercie également **Alexis R**, sans qui je n'aurais pas obtenu cette opportunité, ainsi que Victor V, Dimitri K, Valentin R, Rania G, Marie-Claire H , tous les autres membres de la spécialité EISE avec lesquels j'ai eu l'occasion de discuter, et Adam V, pour leur aide, leur soutien et leur humour incomparable.

Sigles, Abréviations, et Notions abordées

ASIC : *Application Specific Integrated Circuit*, circuit intégré regroupant sur la même puce un grand nombre de fonctionnalités, dédié à une application spécifique.

CNC : *Computer Numerical Control*, machine-outil à commande numérique.

OCR : *Optical Character Recognition*, algorithme permettant de reconnaître du texte perceptible sur une image.

Arduino : Marque de cartes électroniques permettant l'exploitation simple d'un microcontrôleur. Une carte Arduino permet, grâce à son langage propre de programmation et son environnement de développement, de contrôler divers composants et modules électroniques.

microcontrôleur : Circuit intégré regroupant tous les éléments essentiels d'un ordinateur (processeur, mémoire, interface d'entrée/sortie ...).

Logiciel Open-Source : Logiciel dont le code source est accessible gratuitement sur internet.

GUI : *Graphic User Interface*, interface graphique permettant l'interaction homme-machine.

Transistor : Composant électronique à 3 électrodes actives permettant de contrôler le courant ou la tension sur l'électrode de sortie grâce à une électrode d'entrée.

Les transistors utilisés pour le stage sont des transistors de type bipolaire NPN.

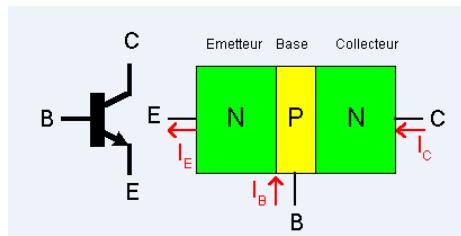


Figure 1: Schéma d'un Transistor NPN

En appliquant une tension sur l'électrode *base*, on peut laisser passer plus ou moins de courant entre le collecteur et l'émetteur. On va ici utiliser les transistors dans leurs états saturés (bloquant ou passant), le Transistor devient alors l'équivalent d'un interrupteur, que nous allons pouvoir contrôler grâce à la carte Arduino.

Thread : En programmation informatique, un thread désigne un processus inclus dans un autre et qui va permettre d'exécuter une portion de code en simultané avec le reste du programme, avec lequel il va partager la mémoire.

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 5 |
| 2 | Principes du fonctionnement Originel | 7 |
| 2.1 | Le Matériel | 7 |
| 2.1.1 | Kit CNC | 7 |
| 2.1.2 | Caméra | 9 |
| 2.1.3 | Autre | 9 |
| 2.2 | Introduction au G-Code | 11 |
| 3 | Modifications du Système | 13 |
| 3.1 | Software | 14 |
| 3.1.1 | La "Machine à État" | 14 |
| 3.1.2 | La génération des commandes | 17 |
| 3.1.3 | La communication avec les cartes Arduinos | 20 |
| 3.1.4 | OCR | 21 |
| 3.2 | Hardware | 23 |
| 3.2.1 | Montage électronique | 23 |
| 3.2.2 | Mécanique | 27 |
| 4 | Intégration du Système | 29 |
| 4.1 | Interface | 29 |
| 4.2 | Intégration du Test | 30 |
| 4.3 | Logs et Détection d'erreurs | 31 |
| 5 | Conclusion | 33 |
| 6 | Table des figures | 34 |
| 7 | Références | 35 |

1 Introduction

Weeroc est une start-up de 5 employés, créée en Février 2012 par Julien Fleury. C'est une spin-off (Start-up issue d'un laboratoire) du laboratoire de recherche *OMEGA, l'IN2P3/CNRS*.

Son domaine d'activité se situe dans la création d'**ASIC**. En particulier, elle est spécialisée dans la création d'ASIC responsable de la lecture de photorécepteurs, répondant à des contraintes de basse consommation, de faible bruit et de résistance à la radioactivité. Ces puces ont pour principaux secteurs d'application l'aérospatiale, l'imagerie médicale et l'instrumentation scientifique, avec au moment du stage, des contrats portant sur des ASICs destinés à être installés dans des télescopes, des satellites ou bien encore des outils servant au démantèlement de centrales nucléaires.

Dans ce contexte, mon rôle était d'automatiser les tests d'ASICs. En effet, chaque ASIC issu de la production doit être soumis à une batterie de tests pour vérifier sa validité selon l'évaluation de plusieurs paramètres (environ 10% de la production n'atteind pas les objectifs de performance). Pour ce faire, la personne réalisant les tests doit placer l'ASIC sur une plateforme de test et atteindre la fin de la réalisation des tests (soit environ 5 minutes), avant de pouvoir regarder les résultats sur un logiciel créé par l'entreprise et de lancer un nouveau test. Or, le volume à traiter commence à devenir trop grand pour un travail "à la main" (plusieurs centaines par mois), et le testage à la main requiert une fragmentation du travail, ce qui empêche l'opérateur de s'occuper efficacement d'une autre tâche pendant la réalisation des tests.

Le stage consistait donc en l'automatisation de ces tests, en modifiant un kit **CNC** (*Computer Numerical Control*), dans ce cas précis une fraiseuse de table à commande numérique (Kit CNC Robuste Carbide3D Shapeoko XXL).



Figure 2: *Kit CNC Robuste Carbide3D Shapeoko XXL de Carbide 3D, élément central du projet*

Il s'agissait alors de prendre en main le contrôle de la machine, et de remplacer la fraiseuse par un système de ventouse et de pompe à air permettant d'attraper l'ASIC, et de le déposer à l'emplacement voulu, ici la plateforme de test. Une fois le test réalisé, le logiciel de test est capable de déduire si l'ASIC est correspondant aux attentes de performance ou non. En utilisant cette information, le système doit être capable de trier les ASICs bon des ASICs défectueux.

En plus de cela, le système devait comporter une caméra capable de lire le numéro de série gravé sur la puce. Il y avait donc un traitement **OCR** (*Optical Character Recognition*) à réaliser.

Les réalisations du stage seront abordées en 3 parties, avec tout d'abord une partie détaillant les principes de fonctionnement du système originel, une seconde abordant les modifications effectuées au système pour correspondre aux besoins de l'entreprise et enfin une dernière partie portant sur l'intégration de l'automatisation des tests sur la plateforme de tests déjà existante.

2 Principes du fonctionnement Originel

Le système utilisé comme base au projet était donc un kit CNC. En particulier, il s'agit d'une table muni d'un "bras" disposé sur des rails et pouvant se déplacer sur 3 axes : X, Y et Z.

Originellement, un dessin est effectué, le plus souvent grâce à un logiciel de **CAO** (*Conception Assisté par Ordinateur*). Ce dessin, mis sous forme vectorielle va être interprété en une série de mouvements par un logiciel de **FAO** (*Conception Assisté par Ordinateur*), dont le but va être d'écrire le fichier contenant le programme qui va être lu par la machine pour réaliser le dessin.

Dans notre cas précis, le programme va être écrit dans le langage de programmation de commande numérique **G-CODE**.

On va, dans cette partie, décrire le matériel utilisé et son fonctionnement, puis introduire le language G-CODE.

2.1 Le Matériel

2.1.1 Kit CNC

Le kit CNC est constitué principalement de 4 moteurs pas-à-pas NEMA23 et des interrupteurs de butée, permettant le mouvement du "bras" sur des rails, et ce selon les 3 axes X, Y et Z. Les moteurs sont contrôlés à l'aide d'une carte Arduino modifiée. L'élément essentiel de cet ensemble se trouvait installé dans la carte Arduino, il s'agit du micrologiciel open-source **GRBL**. C'est un interpréteur open-source de G-Code qui va transposer les commandes en une série de mouvement moteurs.

Le G-Code créé va être envoyé grâce à une communication série à la carte Arduino munie de GRBL, suivant un protocole commande par commande. Une fois la commande effectuée correctement, la carte va renvoyer un message "OK" qui, une fois reçu par l'ordinateur, va déclencher l'envoi de la commande suivante. Ce protocole est utilisé par la plupart des GUI, eux aussi open-source, créés pour interfaçer GRBL.

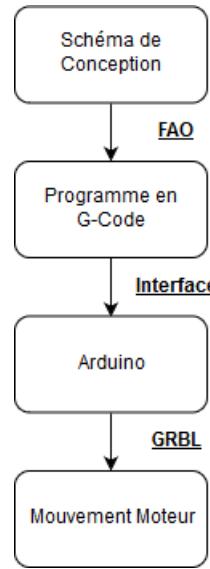


Figure 3: Fonctionnement hiérarchique de la CNC

Ce système permet donc de piloter un bras muni d'une fraiseuse et disposé sur des rails. Le Kit est en plus fourni avec 2 logiciels propriétaires, **Carbide Create** et **Carbide Motion**, qui permettent d'assurer le côté CAO, FAO et interface avec l'Arduino. Carbide Motion permet entre autres de piloter librement le bras, ce qui constitue une aide non négligeable pour se familiariser à son pilotage et à la génération du G-Code approprié, grâce aux logs.

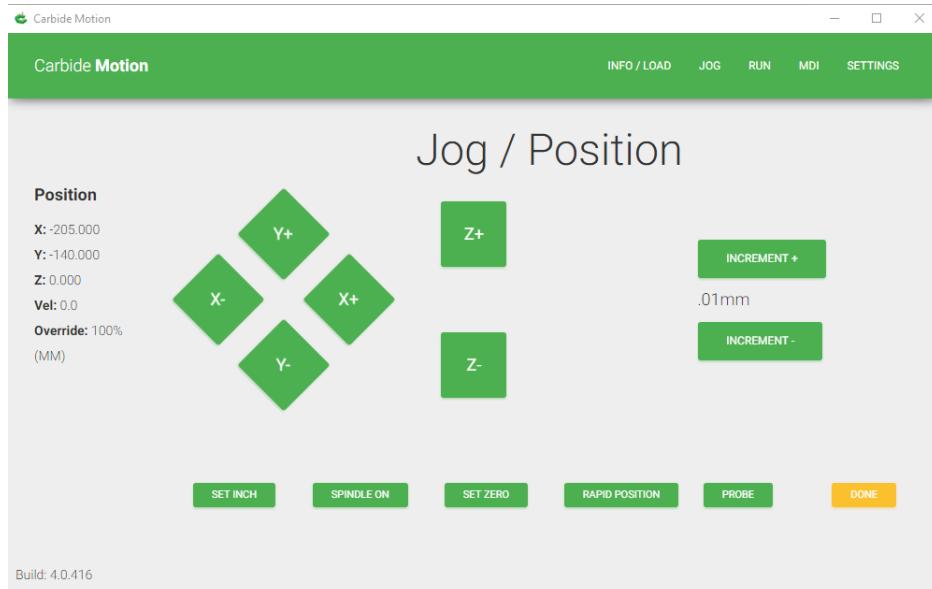


Figure 4: Interface du Logiciel Carbide Motion

2.1.2 Caméra

On dispose également d'une caméra, connectée à une deuxième carte Arduino qui va servir à contrôler les autres parties du système. Disposée elle aussi sur le bras, elle a pour but de prendre une photo à chaque cycle du traitement de l'ASIC, auquel on va appliquer un algorithme d'OCR (*Optical Character Recognition*) permettant d'identifier le numéro de série de l'ASIC. La même opération sera effectuée à différents points du test, pour s'assurer que le même ASIC est en traitement.

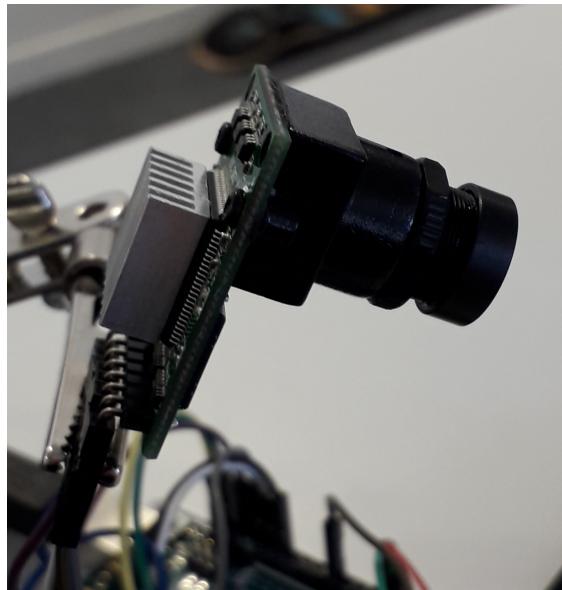


Figure 5: *Caméra ArduCAM Mini avec OV5642 5MP*

2.1.3 Autre

Enfin, nous avons besoin de solutions pour réaliser 2 fonctions : la préhension de l'ASIC, et l'appuie sur l'ASIC quand il se trouve sur la plateforme de test.

En effet, ils sont de type BGA, la connexion se fait donc par des billes sous la puce. Hors pour faire sortir les billes de connexions de la plateforme de test, il faut appliquer une pression d'environ 100N. Nous avons donc besoin d'un moyen d'appliquer cette pression sur l'ASIC pour assurer la connexion.



Figure 6: *ASIC QFP (à gauche, avec des pattes) et BGA (à droite, avec des billes)*

Pour remplir la première fonction, on décide de se servir d'une pompe à air, connectée à une ventouse par l'intermédiaire d'un tuyau.



Figure 7: *Pompe à air D2028B*

Son fonctionnement doit être simple, activer la pompe pour prendre l'ASIC, et la désactiver pour le relâcher. La manière la plus simple d'obtenir ce fonctionnement est de contrôler l'alimentation de la pompe à l'aide d'un Transistor NPN, lui-même contrôlé par une pin d'un microcontrôleur.

Pour ce qui est de l'application de la pression, on décide de se servir d'un vérin. Son fonctionnement est lui plus complexe car il doit être capable de s'arrêter lorsqu'il applique une certaine pression.

Cela peut-être réalisé de 2 manières différentes :

- Comme on va considérer que le vérin et la plateforme de test seront toujours à la même

position, on peut simplement faire en sorte que le vérin s'arrête lorsqu'il atteind un certain point de son expansion, que l'on peut connaître en choisissant un vérin avec feedback.

- Plus le vérin va pousser avec une grande force, plus la consommation en courant va augmenter. On peut donc mesurer le courant consommé par le vérin, et le faire s'arrêter lorsque sa consommation dépasse un seuil.

On choisit au final de combiner les deux puisqu'on va se baser sur la consommation, car cela nous permet de s'assurer que le vérin ne cassera rien en cas de problème, mais on va également se servir du feedback pour mettre une limite à l'expansion.

Dans tous les cas, nous avons également besoin d'un pont en H pour le contrôler (une alimentation en +12V le fait s'étendre, -12V le fait se rétracter, un pont en H permet d'obtenir ces deux tensions en contrôlant simplement avec un microcontrôleur).



Figure 8: *Verin Actuonix L16-P, pouvant appliquer une force de 200N*

2.2 Introduction au G-Code

Le G-Code est un langage permettant la programmation de commande numérique. Une instruction va représenter un mouvement, un changement de paramètre ou un changement au niveau du bras (changement de tête de fraiseuse par exemple). Le G-Code permet de réaliser beaucoup de types d'action différentes, mais nous n'avons besoin en réalité pour réaliser le projet que de mouvements basiques.

Ceux-ci sont de la forme **G00XxxYyyZzz**.

Le G00 indique la vitesse de déplacement (ici la plus grande possible), et le reste indique la position à atteindre, en coordonnées absolues (il y a possibilité de passer en coordonnées relatives à un point, mais cela compliquerait les choses sans grand gain).

Cette commande sera alors interprétée par le logiciel GRBL, qui agira sur les actionneurs grâce aux informations des capteurs, pour atteindre la position voulue.

Mise à part les mouvements simples, les commandes "\$h", permettant de réaliser une

recalibration, et ”?” permettant d’interroger la machine sur son état et sa position, nous seront utiles.

3 Modifications du Système

Le système devait avoir pour base de fonctionnement cet organigramme :

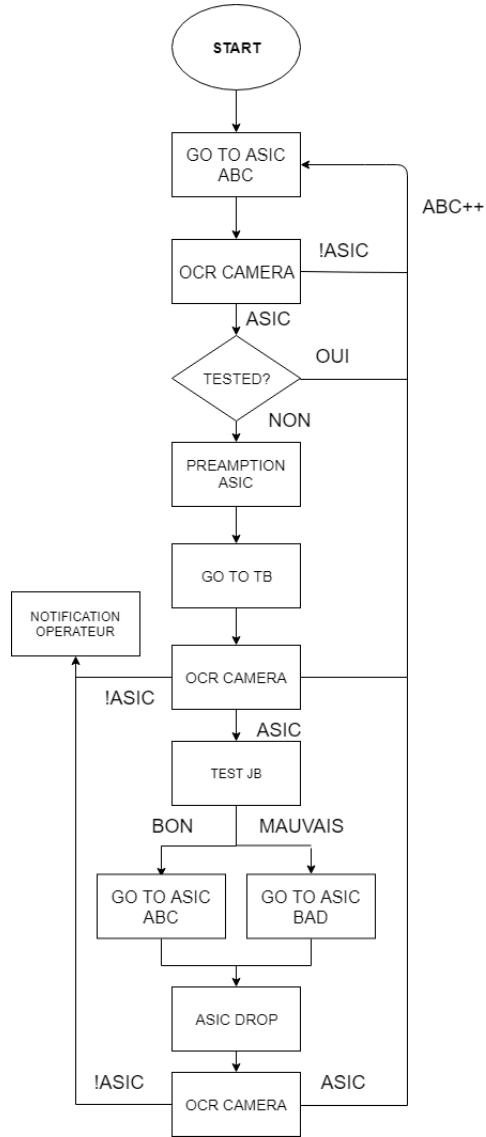


Figure 9: Organigramme de fonctionnement du système

Pour atteindre cet objectif, nous avons effectué un grand nombre de modifications, que nous allons ici décrire, en passant d'abord par un aspect logiciel, puis matériel.

3.1 Software

3.1.1 La "Machine à État"

Le bras a un fonctionnement très cyclique. Le "squelette" est toujours construit autour de Se Déplacer - Descendre - Remonter, avec certaines actions en plus selon le processus en cours. Il est donc intéressant de se baser sur un fonctionnement à la manière d'une machine à état pour générer le G-Code.

À chaque état sera associé une fonction qui produira le G-Code associé, selon le processus en cours : Récupération de l'ASIC à tester (**Retrieving**), Déplacement vers la plateforme de test et réalisation du test (**Testing**), et Triage de l'ASIC (**Sorting**), le processus en cours changeant à la fin de la fonction associée à l'état de Remontée. On peut donc arriver à ces deux diagrammes :

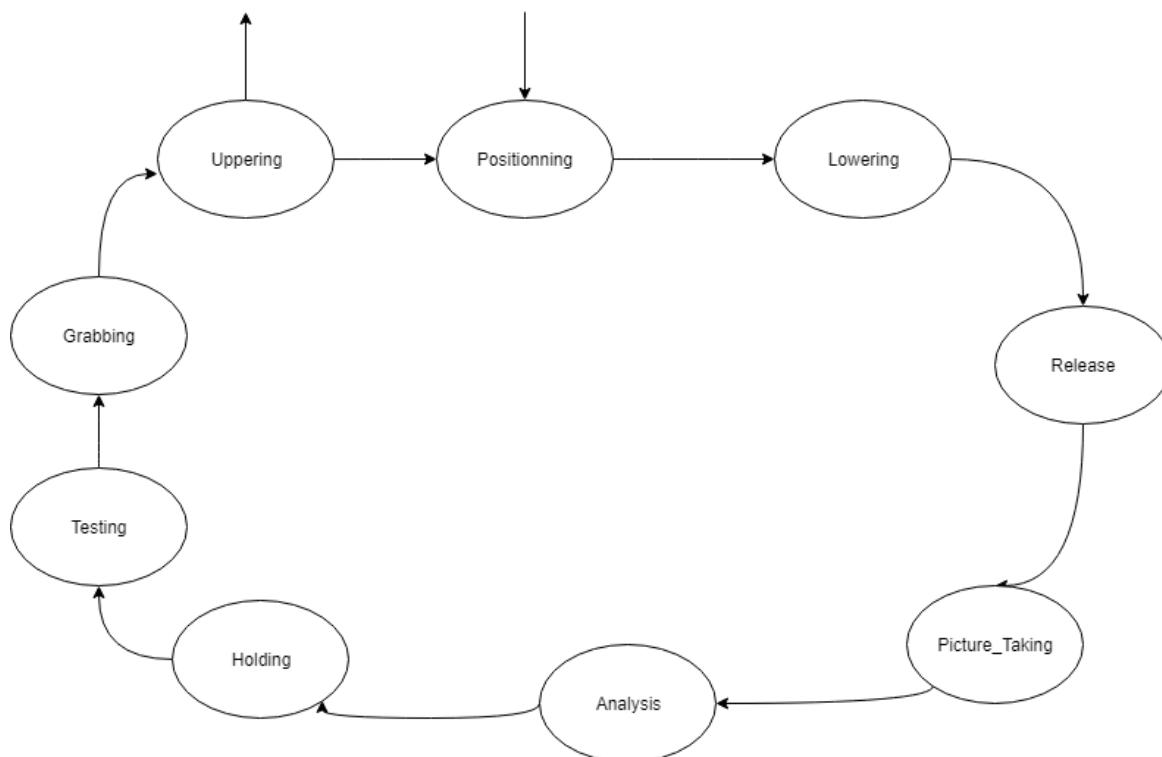


Figure 10: *Diagramme des changement d'états*

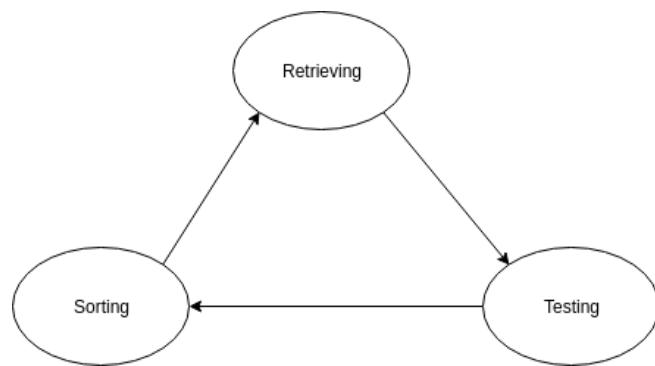


Figure 11: *Diagramme des changements de processus, s'effectuant à la fin de l'état Up-
pering*

<https://www.overleaf.com/project/5cf627a55ebb6676e26dc2b5>

Ce qui va donner, en code C# :

```

if (exec_done && !is_paused && testForm.IsFinished) //Attend la fin de l'exécution des commandes si on en a déclenché une
{
    currentState = nextState;

    switch (currentState)
    {
        case States.Positionning:
            nextState = States.Lowering;
            Positionning(asic, asic_sorting);
            break;

        case States.Lowering:
            nextState = States.Releasing;
            Lowering(asic, asic_sorting);
            break;

        case States.Releasing:
            nextState = States.Picture_Taking;
            Releasing(asic);
            break;

        case States.Picture_Taking:
            nextState = States.Analysing;
            Picture_Taking(asic, asic_sorting);
            break;

        case States.Analysing:
            nextState = States.Holding;
            Analysing(ref asic, ref asic_sorting);
            break;

        case States.Holding:
            nextState = States.Testing;
            Holding();
            break;

        case States.Testing:
            nextState = States.Grabbing;
            Testing(ref asic);
            break;

        case States.Grabbing:
            nextState = States.Upper;
            Grabbing(asic, asic_sorting);
            break;

        case States.Upper:
            nextState = States.Positionning;
            Uppering(asic);
            break;
    }
}

```

Figure 12: *Code de la fonction State Machine*

La machine à état est une fonction, permettant l'exécution d'un état. Cette fonction est contrôlé par un Thread :

```

public void MainFlow()
{
    while (true)
    {
        while (started) //Débute quand on appuie sur le bouton de lancement
        {
            for (n_boucle = 0; n_boucle < n_plateau; n_boucle++)
                for (i_boucle = 0; i_boucle < n_ligne; i_boucle++) //Parcours du plateau
                    for (j_boucle = 0; j_boucle < n_colonne; j_boucle++)
                    {
                        while (!cycle_over) //Tant que le cycle de test n'est pas fini
                        {
                            if (!is_paused)
                                this.Invoke(new EventHandler(State_Machine)); //Rentre dans la machine à état
                            Thread.Sleep(10);
                        }
                        cycle_over = false;
                        bad_read = false;
                        nextState = States.Positionning;
                        inProcess = Processus.Retrieving;
                        if (!no_asic) //Si le cycle précédent s'est terminé car aucun ASIC n'était dans la case, passe directement au suivant
                        {
                            gcode_cmd.Add("G00X-10Y-10Z-10"); //Rapproche le bras de sa position de homing, pour éviter de perdre du temps
                            gcode_cmd.Add(Homing());
                        }
                        testForm.Date = DateTime.Now.ToString("hh_mm_ss");
                    }
            started = false;
        }
        Thread.Sleep(10);
    }
}

```

Figure 13: *Code du Thread contrôlant la Machine à État*

Lorsque l'utilisateur appuie sur le bouton de lancement du test, un embriquement de boucles *for* va être lancé, permettant d'effectuer le test sur tous les ASICs de tous les plateaux renseignés.

Le corps de la boucle est constitué d'une boucle *while* qui va relancer une itération de la machine à état, et ce tant que la variable *cycle over* n'est pas mise à true. Elle est mise à cette valeur à la fin de l'état *Uppering* du process *Sorting*, donc lorsque l'ASIC a été disposé dans l'emplacement approprié selon le résultat du test.

Quand c'est le cas, le programme sort de la boucle et on repasse la variable à *false*, puis on ajoute une commande **Homing**, permettant à la machine de se recalibrer en allant vers ses interrupteurs de fin de courses. Cela permet d'éviter qu'un problème lors d'un déplacement du bras, entraînant un faussage du calibrage, se propage au reste des tests. On passe enfin à l'ASIC suivant.

Il est intéressant de noter que passer par un Thread pour réaliser cette fonction a été nécessaire, car avoir une boucle while comme cela dans le programme principal empêchait la communication avec les cartes Arduinos de se réaliser correctement.

3.1.2 La génération des commandes

Chaque fonction de la machine à état va donc générer le programme servant à accomplir l'action voulue. Il est en particulier composé de mouvements, codés en G-Code et transmis à la carte Arduino de la CNC, et d'actions sur les composants externes (la pompe à air pour la préhension de l'ASIC, le vérin pour faire pression pendant le test, et la caméra) qui seront transmis à la carte Arduino correspondante via une commande simple ("CAM" pour déclencher la prise de photo, "PUMP 1" pour déclencher la pompe à air

...) qui vont être interprétée par la carte Arduino pour agir sur les différents composants et réaliser l'action.

Le but du code réalisé par chaque fonction est le suivant :

- **Positionning** : Positionnement au-dessus de l'ASIC à traiter, puis au-dessus de la carte de test, puis au-dessus de l'emplacement où on va disposer l'ASIC après le test;
- **Lowering** : Descente du bras pour se rapprocher de l'emplacement de l'ASIC;
- **Release** : Positionnement de la pompe au contact de l'ASIC et relâchage de la pression, lors des phases de tests et de tri;
- **Picture Taking** : Décalage du bras pour placer la caméra au-dessus de l'ASIC, et prise de la photo, auquel on applique l'algorithme d'OCR;
- **Analysis** : Affectation du numéro de série à l'ASIC en phase *Retrieving*, et comparaison dans les autres phases;
- **Holding** : Appuie sur l'ASIC avec le verin en phase *Testing*;
- **Testing** : Lancement du test et attente du résultat, uniquement en phase *Testing*;
- **Grabbing** : Récupération de l'ASIC par la pompe, uniquement dans les phases de *Retrieving* et de *Testing*;
- **Uppering** : Interrogation de la pompe dans les phases où elle est censée aspirer l'ASIC, élévation du bras de quelques centimètres et passage à la phase suivante.

Il a également été pertinent de créer une classe "ASIC" correspondant à une case du plateau, pour clarifier et rendre le code plus intuitif. Chaque instance de la classe contient en attribut les coordonnées de la case, si elle contient un asic, et un numéro de série. Elle contient aussi des méthodes pour générer les commandes de déplacement associées à la case.

```
#region Méthodes
6 references
public String getPositionningCmd()
{
    return "G00X" + _x.ToString() + "Y" + _y.ToString();
}

4 references
public String getLoweringCmd() //Descend pour se rapprocher de l'ASIC
{
    float temp = _z + 10;
    return "G00Z" + temp.ToString();
}

4 references
public String getTouchCmd() //Descend assez bas pour rentrer en contact avec l'ASIC
{
    return "G00Z" + _z.ToString();
}

3 references
public String getUpperingCmd()
{
    float temp = (_z + 40 > 0 ? 0 : _z + 40); //Lève de 4cm, met à 0 si le résultat est supérieur à 0
    return "G00Z" + temp;
}
#endregion
```

Figure 14: *Méthodes de la classe ASIC*

Ainsi, le code d'une fonction associée à un état va avoir cette forme, ici pour l'état Positionning :

```
void Positionning(ASIC asic, ASIC[,] plateau_bon, ASIC[,] plateau_mauvais)
{
    lock (gcode_cmd)
    {
        switch (inProcess)
        {
            case Processus.Retrieving: //Quand on se positionne pour aller chercher l'ASIC à tester
                gcode_cmd.Add(asic.getPositionningCmd());
                break;

            case Processus.Testing: //Quand on se positionne au dessus de la test board
                gcode_cmd.Add(test_board.getPositionningCmd());
                break;

            case Processus.Sorting: //Quand on se positionne pour trier l'ASIC
                if (asic.isGood)
                {
                    gcode_cmd.Add(getFirstFreeAsic(plateau_bon).getPositionningCmd());
                }
                else
                {
                    gcode_cmd.Add(getFirstFreeAsic(plateau_bon).getPositionningCmd());
                }
                break;
        }
    }
}
```

Figure 15: *Code d'un état*

3.1.3 La communication avec les cartes Arduinos

C'est la partie la plus critique du système. Une communication Serial robuste devait être mise en place entre le code C# et les cartes Arduinos, pour assurer la réception aussi bien que l'envoi.

C'est en particulier la réception qui a posé le plus de problèmes. La solution la plus simple était de passer par un Thread qui s'exécuterait en arrière-plan qui aurait pour but de gérer la réception des données. Pour la réception, 3 cas principaux se distinguaient: La réception de l'image prise par la caméra, la réception du string de position renvoyé par la CNC, et toutes les autres réceptions.

Le premier cas nécessite de disposer les données dans un buffer spécial, qui sera utilisé pour reconstituer l'image, le deuxième nécessite d'extraire les informations voulues de la String, qui va par exemple être "*Idle — MPos:-5.000,-5.000,-5.000 — Bf:14,127 — FS:0,0*". Ici, on va récupérer la position du bras en coordonnées cartésiennes (*Idle — MPos:-5.000,-5.000,-5.000 — Bf:14,127 — FS:0,0*) et le statut du bras (*Idle — MPos:-5.000,-5.000,-5.000 — Bf:14,127 — FS:0,0*). On veut en particulier pouvoir savoir quand le bras est immobile.

Dans le troisième cas, on recueille simplement les données dans un buffer, auquel on appliquera des tests pour reconnaître son contenu. En particulier, on testera la présence des "ok" venant de la machine CNC indiquant que la commande s'est bien exécuté, et des messages similaires venant de l'Arduino contrôlant la Pompe à air et la caméra.

Dans ce même Thread, on peut également placer le code permettant de reconstituer l'image, l'afficher et la sauvegarder. Ce code a été pris du code-source en C# du logiciel Arducam, fait pour fonctionner avec la caméra dont on dispose. On modifie simplement le code pour faire que l'image soit reconstituée en *.jpg*, et toujours sauvegardée.

Pour la transmission des données, et donc des commandes, on passe par l'intermédiaire d'un troisième et dernier Thread.

Pour ce qui est de la CNC, on va utiliser un simple protocole d'envoi/réponse.

On va envoyer une commande, et attendre que la machine renvoie un "ok" pour envoyer la commande suivante. C'est le protocole à la fois le plus lent et le plus simple, mais comme le test dure 5 minutes, la vitesse n'est pas une considération importante du projet. Tout cela passera par l'intermédiaire d'un booléen que l'on pourra mettre à *True* pour lancer l'exécution et quand un "ok" est reçu, et à *False* à la fin de l'envoi d'une commande à la machine.

Pour l'envoi des commandes, on commence par sécuriser le buffer de commande avec une instruction *lock* par sécurité. Ensuite, on va effectuer un traitement différent selon la commande. Pour les mouvements et la prise de photo, on passe avant par une phase ayant pour but de s'assurer que le bras est immobile :

```

*****TEMPORISATION JUSQU'A CE QUE LE BRAS SOIT IMMOBILE (IDLE)*****
try
{
    is_pos_query = true;
    serialPort1.WriteLine("?");
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
Thread.Sleep(50);
while (!is_idle) //Tant que la tête n'est pas immobile
{
    try
    {
        is_pos_query = true;
        serialPort1.WriteLine("?");
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    Thread.Sleep(50);
}
*****

```

Figure 16: *Code de la temporisation du bras*

On envoie ensuite la commande à la carte Arduino appropriée. Une fois qu'on arrive à la fin du buffer de commandes, on vide le buffer et on indique par un booléen que l'exécution s'est terminée, ce qui va relancer l'avancement de la machine à état.

3.1.4 OCR

Une partie essentielle du projet pour l'entreprise était la capacité du système à pouvoir prendre une photo de tous les ASICs en traitement, et de pouvoir reconnaître leur numéro de série, qui est gravé dessus. D'une part, la reconnaissance de ce numéro est primordiale pour avoir une automatisation complète des tests (l'opérateur de test ne doit pas avoir à devoir entrer le numéro de série de l'ASIC en cours de test), mais d'autre part elle est utile pour s'assurer à différents points du test que l'on est toujours en possession du même ASIC ou pour voir si un ASIC est déjà présent ou non dans la plateforme de test.

Le système devait donc posséder un algorithme d'OCR robuste. Notre choix s'est tourné vers **Tesseract**, une API d'OCR open-source très utilisée, et disponible en C#.

Cependant, nous nous sommes très vite rendu compte que lui donner une image sans traitement ne suffisait pas. En effet, Tesseract est très performant pour lire du texte noir sur fond blanc, mais beaucoup moins pour d'autres utilisations. Nous avons donc dû imaginer un traitement d'image permettant de se rapprocher de ces conditions. Nous avons donc dû faire une série de tests en modifiant les paramètres de contraste, de luminosité, de couleurs et de dimensions afin de trouver la séquence permettant d'obtenir les meilleurs résultats.



Figure 17: *Traitemet d'image (base - contraste augmenté - couleurs inversés)*

Ces tests se sont montrés le plus concluant lorsqu'on augmente le contraste de 200%, suivi d'une inversion des couleurs. Le logo de Weeroc gênant la reconnaissance, il est également nécessaire de rogner l'image pour n'avoir que le numéro de série.

Nous avons donc intégré ce traitement d'image au code C#. La caméra prend une photo avec la résolution la plus haute disponible (2592x1944). On applique ensuite un rognage pour ne garder que le numéro de série (si la régularité et le parallélisme des plateaux avec la machine sont respectés, le rognage peut rester le même pour toutes les photos). On fait ensuite une inversion des couleurs et on passe en nuances de gris.

Ceci constitue le pré-traitement que l'on fait sur l'image.

Ensuite, pour s'assurer de la robustesse de l'algorithme, on fait une boucle augmentant le contraste progressivement jusqu'à atteindre une limite, et dans chaque itération de la boucle, nous faisons en sortes de passer chaque pixel n'étant pas assez foncé en blanc. Comme l'augmentation du contraste fait s'assombrir le numéro de série, il va progressivement ressortir en noir sur blanc, permettant à Tesseract de bien fonctionner.



Figure 18: *Traitemen t d'image effectif (Croppé - Inversion des Couleurs - Augmentation progressive du contraste et passage en noir et blanc)*

3.2 Hardware

3.2.1 Montage électronique

Le circuit électrique réalisé se compose en 2 parties, une pour la pompe à air, et l'autre pour le verin :

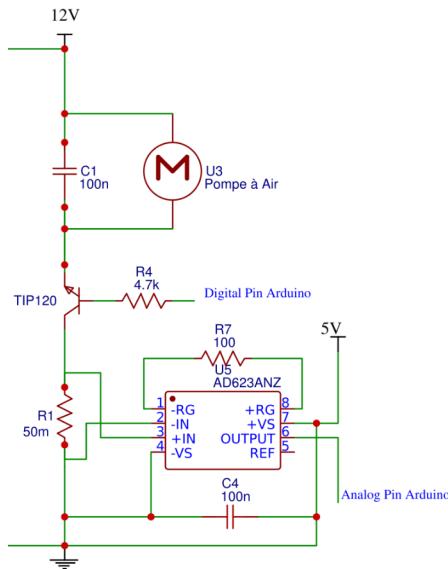


Figure 19: Schéma électrique du montage associé à la pompe à air

Le montage de la pompe est assez direct. il peut se décomposer en 2 parties, avec la pompe et le Transistor pour contrôler son alimentation, et tout le montage à sa suite, servant à avoir un retour sur la consommation de la pompe.

Pour ce faire, on va mesurer la différence de tension aux bornes d'une résistance dites de "Shunt", de faible résistance ($50m\Omega$ ici), qui relie la charge à la masse. La différence mesurée tourne autour de la dizaine de mV. Or, un problème se pose: le tout va aller sur une pin Analogique de la carte Arduino, qui va nous donner la consommation à travers un nombre allant de 0 à 1023 (car l'Arduino possède un convertisseur Analogique-Numérique sur 10 bits, faisant la correspondance 0-5V vers 0-1023). Le pas est donc de $4,88mV$, et si la différence de tension est autour de $10mV$, la valeur donnée par l'Arduino sera juste entre 1 et 3 et ne sera pas traitable, car les variations ne peuvent pas être détectées avec fiabilité. Pour résoudre ce problème, on utilise un amplificateur d'instrumentation (ici l'**AD623AN**) qui va permettre d'amplifier cette différence, pour rendre les données traitables. Ici, on utilise une résistance de 100Ω sur l'ampli pour avoir un gain d'amplification de 1000 . La différence est maintenant supérieure à $1V$, et les variations sont détectables par l'Arduino.

La mesure de la consommation nous sert à être capable de savoir si la ventouse est en possession d'un ASIC ou non. En effet, la consommation augmente de manière significative lorsque la pompe n'aspire pas dans le vide. Seulement, un problème s'est révélé lors des tests. Lorsqu'on coupe la pompe, l'ASIC n'est pas relâché car l'air a du mal à s'échapper, il a donc fallu faire un petit trou dans le tuyau pour que l'air puisse s'échapper. Si cette solution marche pour arriver à nos fins vis-à-vis de l'ASIC, cela fait aussi que la consommation ne monte plus lorsque la ventouse tient un ASIC. Il a donc fallu trouver un moyen détourné pour déterminer si l'ASIC est pris.

Dans un premier temps, les retours sur la consommation sont assez variables, mais en faisant la moyenne sur 20 mesures, on obtient des résultats stables.

Ensuite, en étudiant ces retours, nous avons remarqué que même si les moyennes restaient les mêmes avec ou sans préhension de l'ASIC, les variations au sein des mesures sont plus grande quand la ventouse est en possession de l'ASIC. En récupérant l'écart maximum entre 2 mesures sur les 20 mesures, on peut remarquer une différence notable entre quand la ventouse tient l'ASIC et quand elle ne le tient pas, on peut donc établir un seuil au-dessus duquel on va considérer que la pompe est en possession de l'ASIC.

Ce qui nous donne comme code Arduino :

```
void loop() {
    shunt = analogRead(CURRENT); //On lit la différence en tension
    Serial.print(",");
    Serial.println(shunt);
    if (i < 20) { //Prend l'écart max entre 2 mesures sur 20 mesures
        if (shunt > 230 and shunt < 320){
            mini = (shunt < mini ? shunt : mini);
            maxi = (shunt > maxi ? shunt : maxi);
            i++;
        }
    }else{
        Serial.print("ECART MAX : ");
        Serial.println(maxi-mini);
        if (maxi-mini > THRESHOLD ASIC){
            //VENTOUSE EN POSSESSION DE L'ASIC
        }else{
            //VENTOUSE DANS LE VIDE
        }
        somme = 0;
        mini = 500; maxi = 0;
        i = 0;
    }
    delay(100);
}
```

Figure 20: *Code Arduino associé au test de la prise d'ASIC de la pompe à air*

Ensuite, concernant le verin :

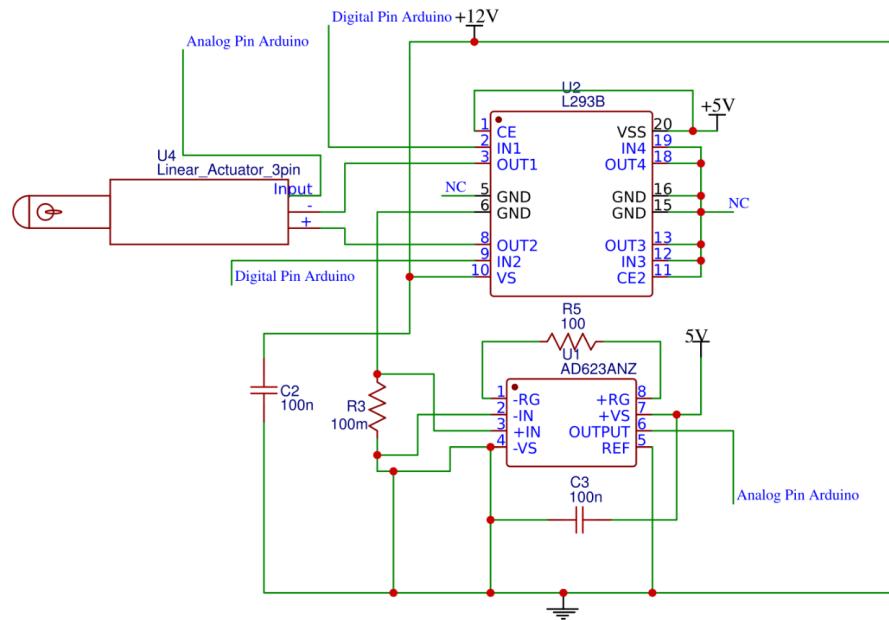


Figure 21: Schéma électronique du montage associé au verin

De la même manière que le montage de la pompe, on peut séparer celui du verin en 2 parties. On a d'abord le verin avec son pont en H (ici un **L293B**), qui va permettre de déterminer le sens de son mouvement grâce aux deux entrées *In1* et *In2* que l'on va activer ou non avec la carte Arduino.

Cette partie est suivie d'un montage analogue à celui de la pompe pour la mesure du courant.

Ici, nous avons remarqué expérimentalement que la valeur moyenne observée changeait à chaque fois que nous relancions le programme, on doit donc programmer dynamiquement le seuil à dépasser (on ne peut pas coder une valeur seuil précise, car elle est susceptible de changer).

On fait donc une moyenne des 20 premières mesures, quand le verin pousse dans le vide, auquel on va ajouter une certaines valeur mesurée expérimentalement pour créer le seuil. Si le seuil est dépassé, cela veut dire qu'on pousse suffisamment sur l'ASIC, auquel cas on coupe l'alimentation du verin qui va garder son extension.

On ajoute simplement les seuils d'extension et de rétractation à ne pas dépasser. Au final, on a le code Arduino suivant:

```

void loop() {
    feedback = analogRead(FB);
    shunt = analogRead(CURRENT);

    if (digitalRead(INPUT1) == 1){ //Si le verin s'étend
        if (i < 20) { //Fait la moyenne des 20 premières mesures en courant,
            somme += shunt; //quand le verin pousse dans le vide
            i++;
        }else{
            if (shunt > (somme / 20) + THRESHOLD_PUSH){
                ShutOff(); //Seuil attend, arrêt du verin
            }
        }
    }

    if (feedback > THRESHOLD_EXTEND){
        setDirectionBW();
    }else if (feedback < THRESHOLD_RETRACT){
        setDirectionFW();
        i = 0;
        somme = 0;
    }
    delay(200);
}

```

Figure 22: *Code Arduino associé au fonctionnement du verin*

3.2.2 Mécanique

Dans le projet, l'aspect mécanique du système est à prendre en compte. En effet, il faut créer des emplacements stables pour disposer les plateaux. Pour que l'automatisation fonctionne correctement, ces emplacements ont besoin d'être d'une grande régularité, et le plus possible parallèle à la machine. En effet, on attribue la position d'une case d'un plateau relativement à la première, grâce à des boucles *for* imbriquées, on peut donc difficilement ajuster si les plateaux ne sont pas parallèles à la machine.

De manière à ce que l'OCR ait un fonctionnement optimal, la plaque doit être peinte en noir mat, afin de limiter la réflexion de la lumière et de faire ressortir le numéro de série gravé le plus possible.

Cependant, le principal point de réflexion était de trouver une solution pour faire tenir la caméra et la pompe sur le bras, avec le plus de stabilité possible. En effet, le tout devait tenir sur l'axe Z, qui est plat.

Nous avons pour cela décidé de créer une pièce 3D sur SolidWorks, et de la faire imprimer par le service en ligne Sculpteo.

Il faut également un emplacement où placer les cartes Arduino et la pompe. Cet ensemble doit suivre le bras, le meilleur emplacement est donc sur le moteur de l'axe X.

Pour réaliser un premier prototype, nous avons utilisé une boîte en carton, dans lequel nous avons disposé les cartes Arduino avec du scotch double face, et attaché la pompe à air avec des serres-câble. Le carton même est attaché à l'axe avec des serres-câble.

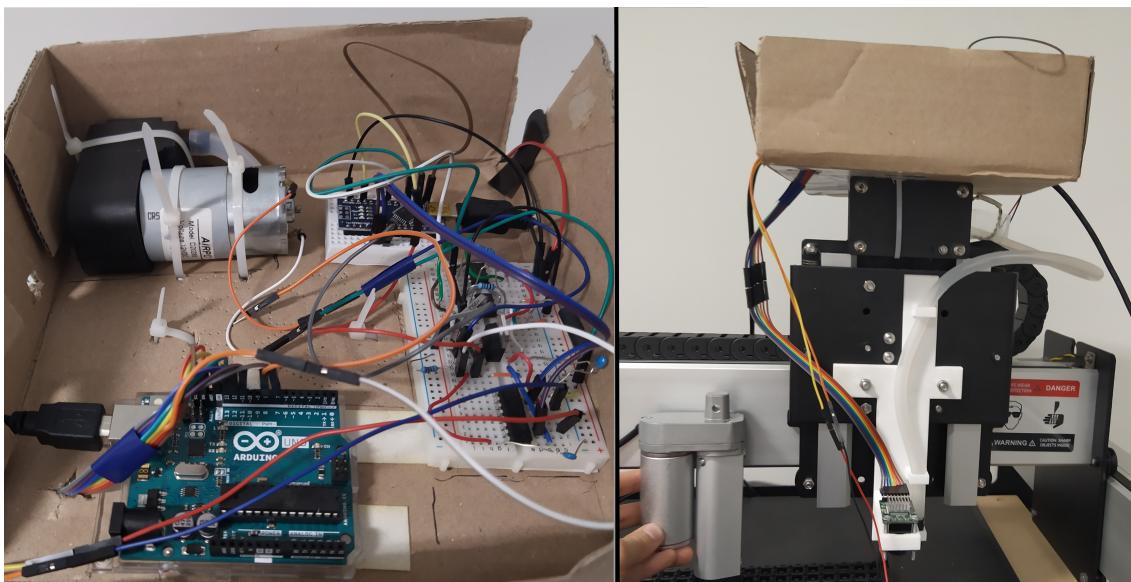


Figure 23: *Système réalisé - Fin Juillet 2019*

A terme, un socle imprimé en 3D et un circuit imprimé seront réalisés pour améliorer la robustesse et rendre le système durable dans le temps.

4 Intégration du Système

//

Pour une automatisation du test, le système doit être capable de réaliser toutes les actions effectuées par l'opérateur. On doit donc avoir une automatisation du lancement du logiciel de test, mais également récupérer une capture d'écran du résultat et renseigner un fichier Excel avec le numéro de série et le résultat du test.

Il est par ailleurs essentiel d'avoir une interface graphique utilisateur.

4.1 Interface

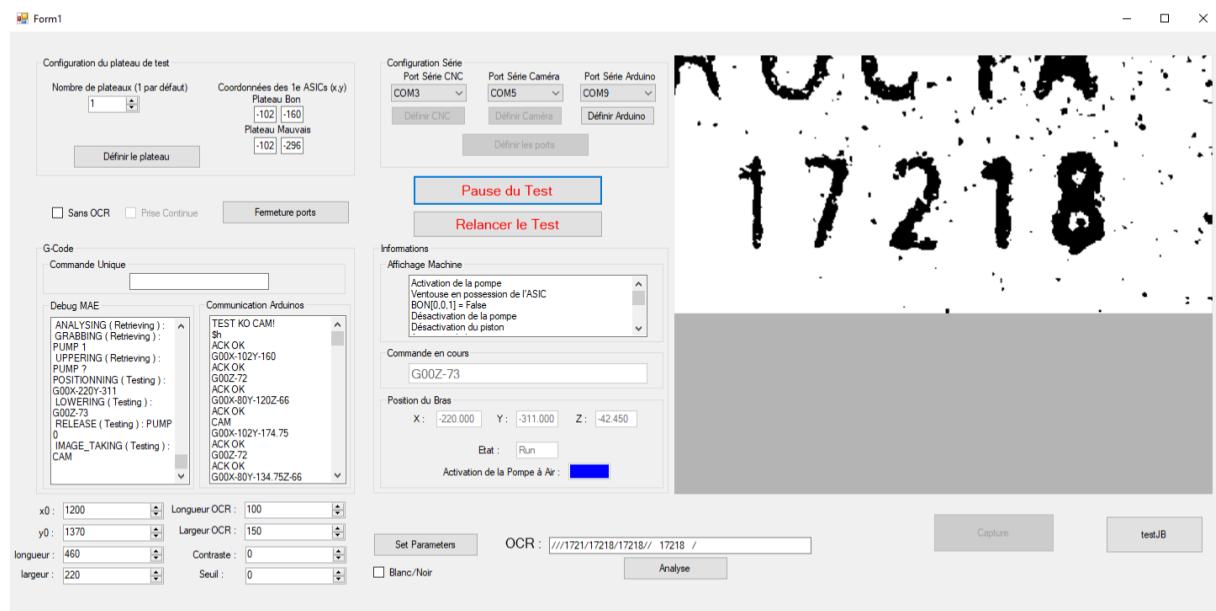


Figure 24: Interface Graphique Réalisée

Pour permettre une interaction homme-machine, une interface a été réalisé. Elle permet en particulier de pouvoir faire des tests de caméra, avec le bouton "Capture", les Check-Box "Sans OCR" qui permet de prendre une photo sans réaliser de traitement OCR, et "Prise Continue" qui permet de prendre des images en continu pour avoir un flux vidéo avec une cadence assez faible (environ 3 images par seconde).

On peut également tester le traitement OCR, en prenant une capture avec l'option "Sans OCR", puis d'appuyer sur le bouton "Analyse" avec les paramètres renseignés en bas à

gauche (x_0 et y_0 l'origine du rognage, longueur et largeur sa taille, longueur et largeur OCR la taille de l'image qui sera donnée à l'algorithme OCR, car il a plus de facilité à lire des images d'une taille assez condensée, Contraste pour l'augmentation du contraste après inversion des couleurs, et enfin "Seuil" pour le seuil du passage en noir et blanc si l'option est activée).

Elle permet surtout de pouvoir monitorer le fonctionnement des tests, avec les états réalisés par la MAE à gauche, les logs de communication entre l'ordinateur et les Arduinos à droite, la commande G-Code en cours d'exécution et la position du bras.

4.2 Intégration du Test

Le test de l'ASIC est automatique, réalisé par ordinateur par l'intermédiaire d'une carte de test et d'un logiciel créées par l'entreprise. Comme évoqué dans l'introduction, le logiciel de test a été réalisé en C# sous Visual Studio.

Pour l'intégrer au système, la solution la plus simple est d'instancier la classe de test, et de faire apparaître et disparaître l'interface au moment opportun avec les fonctions *Show()* et *Hide()* du C#.

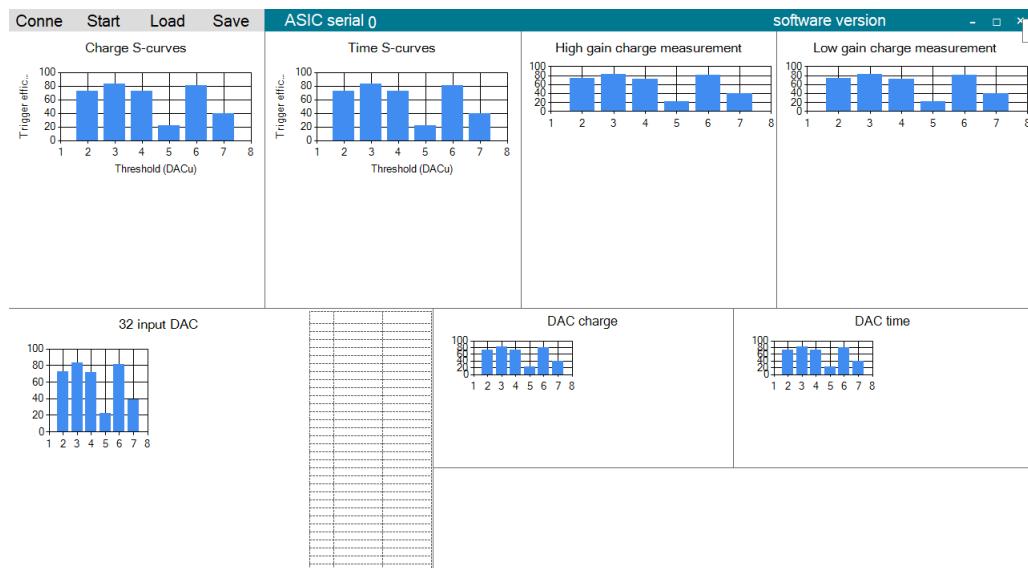


Figure 25: *Interface de test*

Pour lancer le test automatiquement, nous devons simplement faire appel à un équivalent de la fonction associée au bouton "Start" de l'interface.

En plus de cela, nous avons besoin d'ajouter un booléen changeant de valeur au lancement du test et lorsque celui-ci se termine, puis de l'utiliser pour interrompre toute action le temps que le test soit fait, et le programme de test est intégré.

En plus de s'occuper de placer les puces sur la carte de test, l'opérateur de test avait

aussi pour mission de renseigner un Excel avec les résultats du test et de prendre une capture d'écran de l'interface avec les tests effectués.

Là encore pour la capture, on peut faire appel à une fonction analogue à celle associé au bouton "Save" de l'interface.

Ensuite pour le remplissage du fichier Excel, nous devons cette fois passer par l'intermédiaire de la bibliothèque **Microsoft.Office.Interop.Excel**. On crée simplement un nouveau workBook et workSheet, que l'on remplit avec les résultats du test. En particulier, le rendu du test est fait sur 5 booléens, et le test est passé si les 5 sont à *true*. On veut donc renseigner la valeur de ces 5 booléens sur le fichier Excel. On crée donc une fonction que l'on appellera à chaque fin de test, et une structure contenant toutes les informations.

```
var chip1 = new object[1, 7]
{
    { date+"_"+textBox_asicNumber.Text, "YES", TestDC_OK,TestInDac_OK,TestThDac_OK,TestScurves_OK,TestHit_OK },
};

AjouterChip(chip1);
isGood = true;
```

Figure 26: Structure "chip"

```
2 références
private void AjouterChip(object[,] chip)
{
    int j;
    for (j = 0; j < 7; j++)
    {
        xlWorkSheet.Cells[i + 1, j + 1] = chip[0, j];
    }
    i = i + 1;
    xlWorkSheet.Cells[1, 8] = i;

    xlWorkBook.SaveAs(savepath);
}
```

Figure 27: Fonction permettant d'ajouter un ASIC au fichier Excel

4.3 Logs et Détection d'erreurs

Enfin, le système doit être capable de s'arrêter sans intervention extérieure si un problème apparaît. Nous devons donc trouver un moyen de détecter les erreurs, et de pouvoir donner la possibilité de retracer son origine. Les erreurs auront principalement des causes matériels, on peut donc implémenter des **Timeout** à chaque fois qu'une commande est envoyé vers une Arduino. Si le résultat voulu (un *ok* de l'Arduino, l'affichage de l'image capté, le résultat d'une interrogation pompe...) n'est pas reçu dans un temps imparti, on peut considérer qu'un problème a eu lieu et on arrête l'exécution.

Pour arriver à cet objectif, on utilise la fonction **millis** pour connaître le temps d'exécution d'une tâche, en faisant la différence du temps de début d'exécution et le temps courant. Si la limite est dépassée, on envoie un message à l'ordinateur.

```

else if (inputString.startsWith("PUMP ?"))
{
    //Déduis si la pompe tient l'ASIC
    //Relève la valeur min et max sur 20 mesures
    previousMil = millis(); //Prends le temps de ré

    while (i < 20){

        currentMil = millis();
        if(currentMil - previousMil > TIMEOUT_PUMP){
            digitalWrite(PUMP_TRANSISTOR, LOW);
            Serial.println("TIMEOUT POMPE\n");
            is_to = true;
            break;
        }
    }
}

```

Figure 28: *Code du Timeout de la requête Pompe (Arduino)*

En C#, le fonctionnement est similaire, la seule différence étant que l'on peut directement utiliser un chronomètre avec l'objet **Stopwatch**. On le déclenche à l'envoie de la commande et on relève sa valeur régulièrement dans le Thread *CmdExecution*. Si la limite de temps est dépassée, on arrête tout.

```

if (sw.IsRunning && sw.ElapsedMilliseconds > timeout_time)
{
    MessageBox.Show("TIMEOUT" + gcode_cmd[n-1] + "ARRET DU PROGRAMME.");
    is_paused = true;
    sw.Reset();
    init_cnc();
    is_capturing = false;
    end_thread();
}

```

Figure 29: *Code du Timeout d'une commande (C#)*

Pour ce qui est des logs, on peut consigner quand un état principal de la MAE s'est déroulé correctement, le résultat des interrogations pompes, le numéro de série trouvé à chaque prise de photo, le résultat du test, les indices de la case du plateau en cours de traitement et les timeouts obtenus, tout cela avec l'horodatage.

Par ailleurs, il est préférable de relâcher le contrôle du fichier texte à intervalles très réguliers pour éviter de corrompre l'intégralité du fichier si quelque chose se passe mal et provoque un arrêt inopiné de l'exécution du programme.

5 Conclusion

Le but du stage était donc de réaliser un système permettant d'automatiser des tests de puces, à partir d'une machine CNC. En particulier, il devait être capable de déplacer les puces, de détecter le numéro de série, de faire une pression sur la puce avant de lancer l'interface de test, puis de classer l'ASIC en fonction du résultat du test (bon ou mauvais). À la fin du stage, le système était en effet capable de déplacer l'ASIC, de reconnaître le numéro de série dans une majorité des cas, de lancer l'interface de test et de trier l'ASIC en fonction du résultat. Le système parvient par ailleurs à fonctionner selon l'organigramme montré précédemment.

Le plus gros challenge non résolution était l'aspect mécanique de la fixation du verin pour faire pression sur l'ASIC. La solution proposée de fixer le verin à l'arrière de l'axe X n'a pas encore pu être implémenté ou testé.

Le code Arduino associé au verin, lui permettant de s'arrêter quand une certaines pression est appliquée et de provoquer un timeout le cas échéant, à lui en revanche été fait et testé.

Des solutions pour permettre au système de détecter certains problèmes et de s'arrêter complètement si besoin ont également été mises en place, par l'intermédiaire de timeout si la commande ne s'est pas exécuté après un certains temps.

De plus, le système est capable de recommencer une action s'il a détecté qu'elle ne s'est pas déroulé correctement, comme un 2e passage par l'OCR avec des paramètres plus souples si la première tentative a été infructueuse. Si les deux tentatives ne donnent rien, l'ASIC est placé avec les mauvais, qui sont de toute façon destinés à être testé une deuxième fois.

Concernant les améliorations pouvant être apportées au système, elles concernent surtout l'OCR. En effet, même si de nombreuses pistes ont été explorées pour le rendre plus robuste et que l'algorithme de traitement d'image a été considérablement amélioré au cours du stage, il n'arrive toujours pas à détecter avec une grande fiabilité le numéro de série. Comme il s'agit d'une composante essentielle du système, la fiabilité doit être améliorée. Ensuite, une fois l'aspect mécanique de la pose du verin résolu, le système doit passer par une longue période de test pour mettre à l'épreuve et améliorer la fiabilité du projet dans son intégralité.

On peut également améliorer la robustesse de l'objet, en concevant un PCB (Printed Circuit Board, circuit imprimé), qui nous permettrait de nous passer d'une labdec et de la grande majorité des fils. De la même manière, une pièce 3D peut-être créer pour servir de socle au-dessus du moteur de l'axe Z. Il pourrait être conçu en corrélation avec le PCB pour lui permettre d'avoir un emplacement dédié.

6 Table des figures

List of Figures

| | | |
|----|---|----|
| 1 | <i>Schéma d'un Transistor NPN</i> | 3 |
| 2 | <i>Kit CNC Robuste Carbide3D Shapeoko XXL de Carbide 3D, élément central du projet</i> | 5 |
| 3 | <i>Fonctionnement hiérarchique de la CNC</i> | 8 |
| 4 | <i>Interface du Logiciel Carbide Motion</i> | 8 |
| 5 | <i>Caméra ArduCAM Mini avec OV5642 5MP</i> | 9 |
| 6 | <i>ASIC QFP (à gauche, avec des pattes) et BGA (à droite, avec des billes)</i> | 10 |
| 7 | <i>Pompe à air D2028B</i> | 10 |
| 8 | <i>Verin Actuonix L16-P, pouvant appliquer une force de 200N</i> | 11 |
| 9 | <i>Organigramme de fonctionnement du système</i> | 13 |
| 10 | <i>Diagramme des changement d'états</i> | 14 |
| 11 | <i>Diagramme des changements de processus, s'effectuant à la fin de l'état Uppering</i> | 15 |
| 12 | <i>Code de la fonction State Machine</i> | 16 |
| 13 | <i>Code du Thread contrôlant la Machine à État</i> | 17 |
| 14 | <i>Méthodes de la classe ASIC</i> | 19 |
| 15 | <i>Code d'un état</i> | 19 |
| 16 | <i>Code de la temporisation du bras</i> | 21 |
| 17 | <i>Traitemet d'image (base - contraste augmenté - couleurs inversés)</i> | 22 |
| 18 | <i>Traitemet d'image effectif (Croppé - Inversion des Couleurs - Augmentation progressive du contraste et passage en noir et blanc)</i> | 23 |
| 19 | <i>Schéma électronique du montage associé à la pompe à air</i> | 24 |
| 20 | <i>Code Arduino associé au test de la prise d'ASIC de la pompe à air</i> | 25 |
| 21 | <i>Schéma électronique du montage associé au verin</i> | 26 |
| 22 | <i>Code Arduino associé au fonctionnement du verin</i> | 27 |
| 23 | <i>Système réalisé - Fin Juillet 2019</i> | 28 |
| 24 | <i>Interface Graphique Réalisée</i> | 29 |
| 25 | <i>Interface de test</i> | 30 |
| 26 | <i>Structure "chip"</i> | 31 |
| 27 | <i>Fonction permettant d'ajouter un ASIC au fichier Excel</i> | 31 |
| 28 | <i>Code du Timeout de la requête Pompe (Arduino)</i> | 32 |
| 29 | <i>Code du Timeout d'une commande (C#)</i> | 32 |

7 Références

Kit CNC Robuste Carbide3D Shapeoko XXL <https://www.robotshop.com/eu/fr/kit-cnc-robuste.html>

CNC - Wikipedia https://fr.wikipedia.org/wiki/Machine-outil_%C3%A0_commande_num%C3%A9rique

FAO - Wikipedia https://fr.wikipedia.org/wiki/Fabrication_assist%C3%A9e_par_ordinateur

Arduino - Wikipedia <https://fr.wikipedia.org/wiki/Arduino> Micro-Controleur - Wikipedia <https://fr.wikipedia.org/wiki/Microcontr%C3%B4leur>

G-Code - Wikipedia https://fr.wikipedia.org/wiki/Programmation_de_commande_num%C3%A9rique

GRBL - Github <https://github.com/grbl/grbl/wiki>

Arducam <https://github.com/ArduCAM>

Arducam - Code-Source C# <https://github.com/ArduCAM/Arduino/issues/39>