Adrien BACHMEYER
Jade BARRET
Alexis CORBILLET
Khalid LAHBABI
Maurice MANNONI
Charley THOM
Groupe 2

Livrable IETI : Projet de Vision Industrielle

L'enjeu principal de ce projet est la mise en place d'un système de détection d'objets sur une chaîne de production, et du système électromécanique de tri de ces derniers.

Objectif

Trier des objets selon leur forme et leur couleur

Cahier des charges

- Créer une interface utilisateur claire et intuitive
- Créer un algorithme de reconnaissance de forme et couleur à l'aide d'un réseau de neurones
- Maîtriser le fonctionnement d'un moteur pas-à-pas et de servo-moteurs

Diagramme fonctionnel

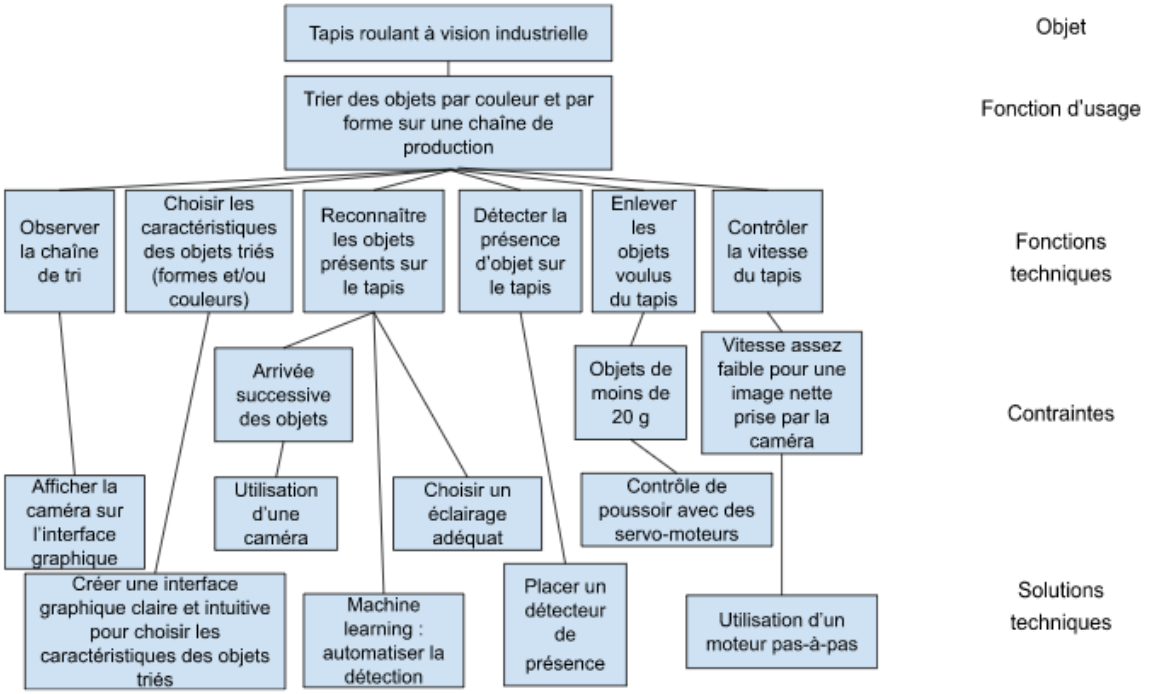
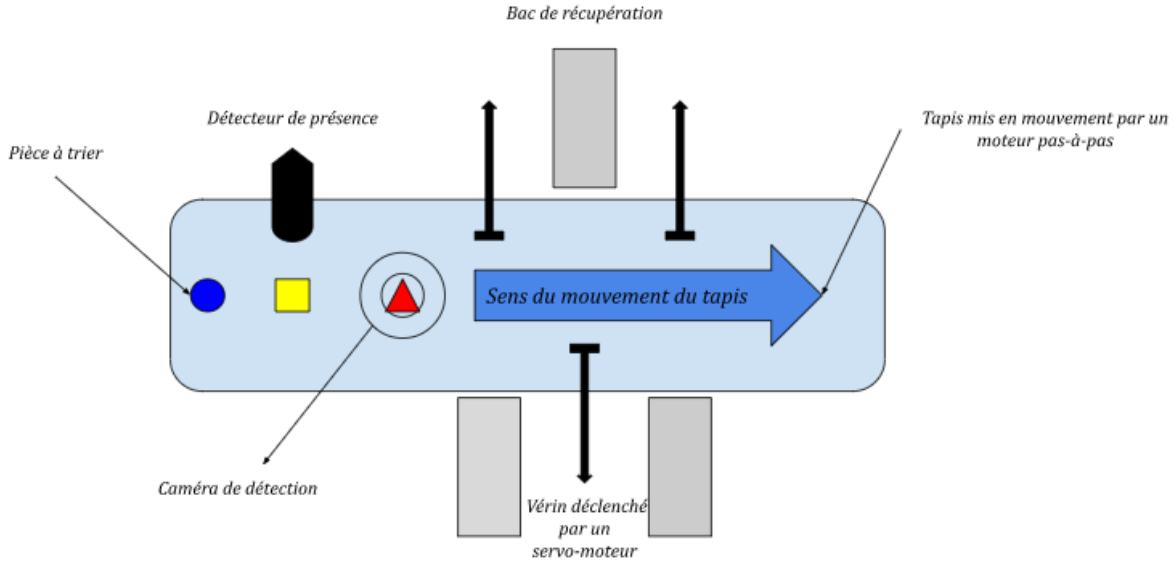
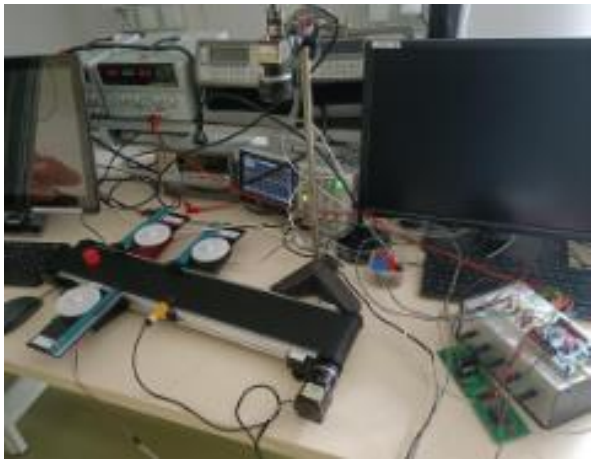


Schéma et photo du système





Notice d'utilisation

1. Mettre en marche le moteur du tapis.
2. Lancer le code et rentrer le numéro du port COM auquel la carte Nucléo est reliée.
3. Aller sur l'interface qui vient de s'ouvrir et choisir les options de tri puis valider en cliquant sur « Envoi des options ».
4. Mettre les éléments à trier sur le tapis.

Les principales fonctions qui vont être détailler dans la suite ont été réparties dans trois parties différentes à savoir :

I. La mise en œuvre de la motorisation du tapis et des vérins (Moteurs p.3),
II. La création d'une interface (Interface p.10),
III. La création d'un algorithme de détection des formes et couleurs (Détection p.13).

I. Moteurs

Notre objectif est la mise en place des moteurs permettant le bon fonctionnement du tapis, ainsi que des vérins employés pour pousser les pièces dans des bacs accolés au système. Ceci présuppose que le tapis doit fonctionner dès l'allumage du système, et ce à vitesse constante, et que les moteurs permettant le mouvement des vérins autorisent -sur commande extérieure au vérin lui-même- les mouvements allers et retours de ces derniers. La réponse technologique à ce besoin technique est un moteur à quatre temps pour le mouvement du tapis, et l'emploi de servomoteurs pour le contrôle des vérins latéraux. De plus, nous concevons le système de telle sorte que l'ordre fourni aux servomoteurs sur la position angulaire à adopter provienne de la partie de l'algorithme de contrôle du système sélectionnant les pièces à trier.
Il convient ainsi de mettre en place un système de contrôle des moteurs, dont l'information reçue de l'extérieur (le reste des agents du système) est l'ordre de déclenchement des servomoteurs.

- Un moteur pas à pas est un moteur dont le principe de fonctionnement consiste à diviser une rotation en un certain nombre d'étapes (d'où le terme de « pas »). Pour ce faire, on dispose un aimant permanent sur un axe autour duquel on dispose des bobines régulièrement positionnées angulairement. Chacune de ces bobines devra pouvoir être contrôlée sur demande, a priori indépendamment des autres. Nous verrons par la suite que contrôler les bobines deux-à-deux (celles symétriques par rapport à l'axe de l'aimant) est nettement plus efficace et facile à mettre en place à l'aide de ponts en H.
  On rappelle que pour une unique bobine, un champ magnétique est émis dans le milieu extérieur lorsque du courant la traverse. De plus, le sens d'orientation des lignes de champ est défini par la règle de la main droite, et dépend donc uniquement du sens de circulation du courant.

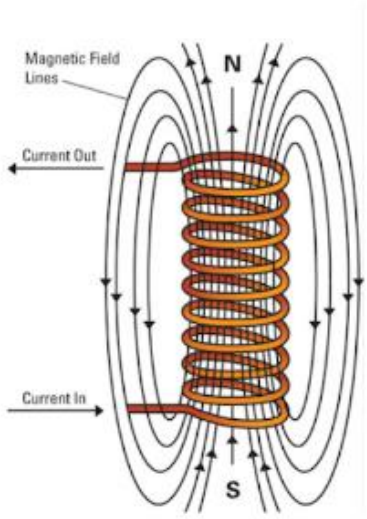


Figure 1 : Champ magnétique émis par un solénoïde fini parcouru par un courant (Source : PERJES)

La force ainsi exercée sur l'aimant peut être calculée d'après la loi de Laplace. Notons l'angle représentant l'orientation de la bobine par rapport à l'aimant. Si nous disposons au total de 2n bobines, à l'instant initial, la bobine k-ième fait un angle $\theta_k = \pi k/n$ par rapport à l'aimant. Si une bobine donnée (de longueur L) est parcourue par un courant I (d'où un champ magnétique de norme B), la loi de Laplace stipule donc que l'aimant subit une force induite $F_k = BIL \sin(_k)$. Afin de minimiser son énergie, l'aimant va avoir tendance à se positionner parallèlement à l'axe défini par les deux bobines, d'autant plus si la bobine opposée se voit appliquer un courant en sens contraire. Ainsi, l'aimant tourne sur son axe, et entraîne avec lui le système mécanique pour peu qu'ils soient solidaires. Suite à quoi on cesse l'alimentation du premier jeu de bobines, on active le suivant, et ainsi de suite. Notons qu'après un nombre de pas égal au nombre de bobines disposées, le système a effectué un tour complet.
Nous pourrions noter qu'il existe deux positions angulaires garantissant l'alignement des bobines activées et de l'aimant, mais une seule est stable car l'autre consiste en un maximum de potentiel. Du fait des faibles perturbations naturellement provoquées par le système, on peut très raisonnablement considérer que seul l'équilibre stable est atteint à chaque pas.
Nous utilisons ci-dessous un schéma dans le cas particulier de quatre bobines (c'est le cas du moteur à disposition).

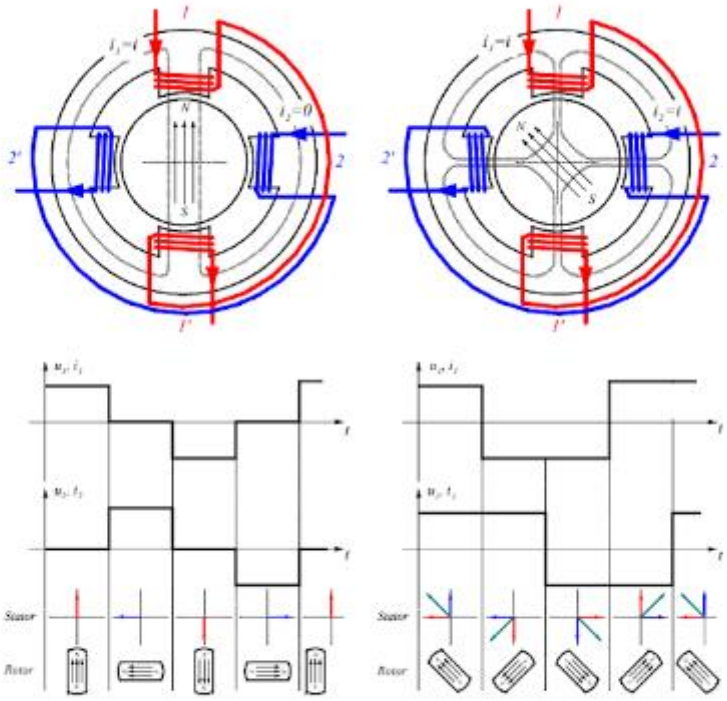


Figure 2 : Moteur pas-à-pas à aimant permanent et quatre bobines (Source : Systèmes électromécaniques, Haute Ecole Spécialiste de Suisse Occidentale, Chapitre 7, page 18)

Le schéma présenté précédemment expose le cas du fonctionnement à pas complet (à gauche, où chaque paire de bobines est alimentée successivement), et de celui avec un couple maximal (à droite, où les bobines sont toutes alimentées en même temps). Si le nombre de pas ne change pas entre les deux configurations, on opte cependant pour le

premier choix technologique, car bien que le champ magnétique total soit moins puissant, on gagne en précision sur l'orientation angulaire car chaque pas est orienté à l'aide d'une unique source d'alimentation en courant.
On opte ainsi pour un ordre d'alimentation régit de la façon suivante, en utilisant les conventions du schéma précédent :

N° arbitraire de pas	Bobine 1	Bobine 1'	Bobine 2	Bobine 2'
1	+	-	Ø	Ø
2	Ø	Ø	+	-
3	-	+	Ø	Ø
4	Ø	Ø	-	+

Il reste maintenant à expliquer comment nous procédons en pratique à l'alimentation successive des bobines. Nous utilisons à cet effet la carte de puissance L298, constituant un étage de puissance, lui-même reposant sur une paire de ponts en H (chacun étant alloué respectivement à une paire de bobines opposées par rapport à l'aimant). Revenons alors succinctement sur le principe de fonctionnement du pont en H.

Un pont en H est un système employé en électronique afin de contrôler le courant imposé à un dipôle. L'objectif de ce système est de pouvoir, selon une commande extérieure, imposer un courant dans un sens ou dans l'autre au dipôle, ou bien d'imposer un courant nul. La structure générale de ce système est rappelée ci-dessous, où l'objet contrôlé est un dipôle donné.

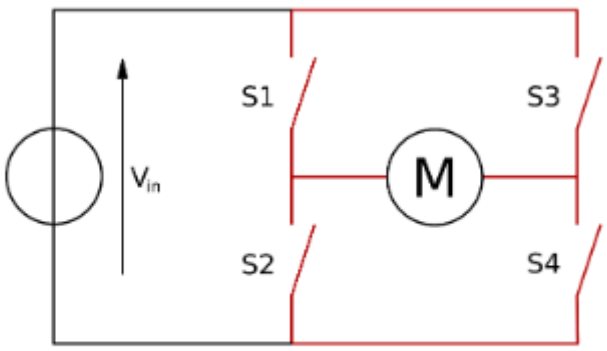


Figure 3 : Schéma d'un circuit avec un pont en H (Source : Wikipédia)

On observe que lorsque tous les commutateurs sont ouverts, la tension est nulle aux bornes du dipôle, que le courant appliqué est strictement positif (respectivement strictement négatif) à travers le dipôle lorsque seuls les commutateurs S1 et S4 sont fermés

(respectivement S2 et S3), et que l'on court-circuite le dipôle lorsque seuls S1 et S3 ou S2 et S4 sont fermés. De fait, dans notre application au contrôle du courant traversant des solénoïdes, on dispose de l'ensemble des configurations nécessaires pour faire fonctionner le système. Notons enfin que, disposant de seulement deux ponts en H, on a pallié ce manque de deux ponts en H en associant les bobines en paires opposées par rapport à l'axe du rotor, ce qui nous est rendu possible en pratique en constatant qu'à chaque instant, le courant les traversant est soit nul pour toutes les deux, soit de signes opposés, ce qui est précisément la situation escomptée.
Disposant de l'ensemble des sous-systèmes principaux utilisés pour le contrôle du moteur pas-à-pas, il reste naturellement à en expliquer le contrôle via la carte de puissance L298 et celle de contrôle (L297). Ces dernières seront reliées l'une à l'autre via des connexions mâles/femelles, et nous nous intéresserons exclusivement aux broches du circuit d'information de la carte L297, et à celles de puissance de la carte L298.

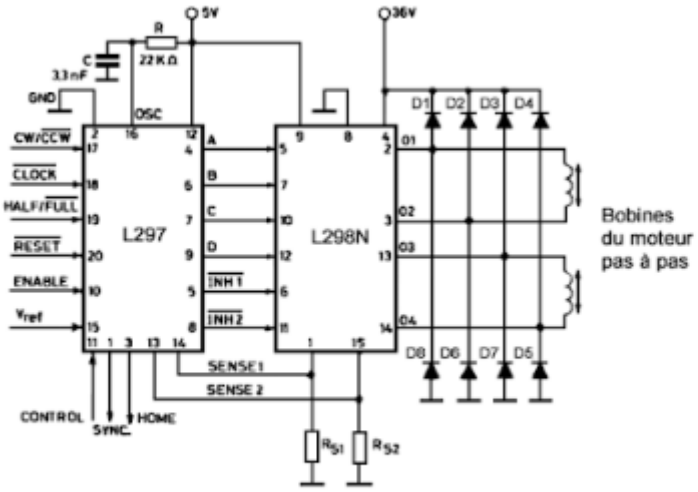


Figure 4 : Commande de moteur pas-à-pas avec les circuits L297 et L298 (Source : Positron-libre)

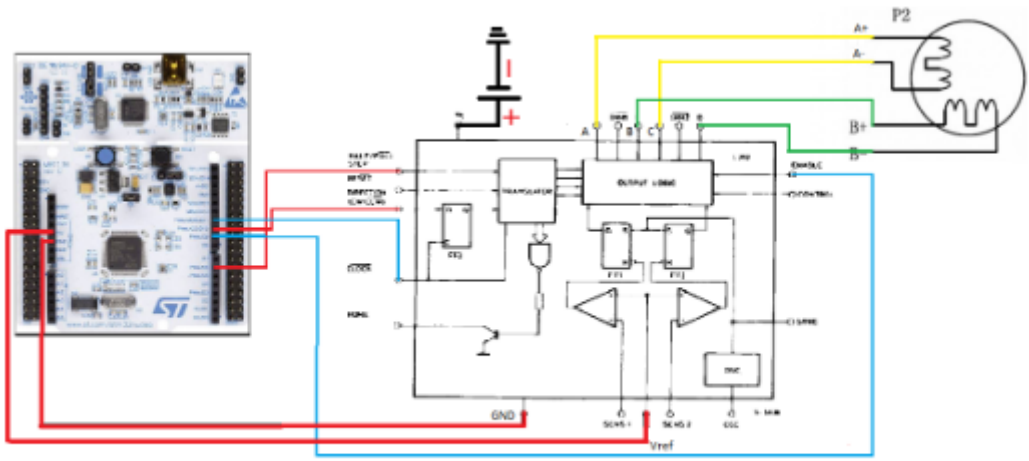


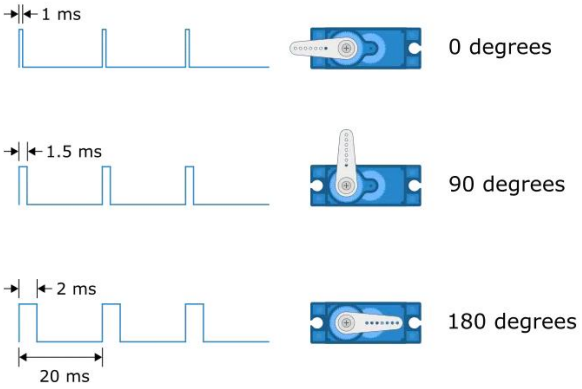
Figure 5: schéma électrique du moteur et assemblé avec la carte de contrôle et la nucléo.

Concernant l'alimentation, nous noterons que la carte de puissance est alimentée par la carte Nucléo, sur laquelle nous reviendrons par la suite, à l'aide de sa broche de 5V, alimentant exclusivement le circuit d'information. De plus, une seconde source d'alimentation externe permet de fournir une tension de 15V (tension nominale du moteur à piloter pour assurer la rotation, sans détériorer les composants), dont a besoin le moteur pour l'alimentation du circuit de puissance, via l'alimentation contrôlée des bobines à l'aide du pont en H.
La carte employée dispose de plusieurs broches, permettant le contrôle optimal du système. En effet, on dispose de la broche *ENABLE*, permettant de démarrer le moteur, et dont l'entrée sera tout naturellement maintenue au niveau logique 1 tout au long de l'utilisation du programme.
On dispose aussi de la broche *CW*, permettant de déterminer le sens de la séquence demandée, et donc le sens dans lequel va évoluer le champ tournant, c'est-à-dire le sens dans lequel tourne le moteur.
Quant à l'entrée CLOCK, elle permet de déterminer l'horloge du système en faisant évoluer les configurations du séquenceur à chaque pas. Cette broche sera reliée à une broche de type *PWM* de la carte nucléo afin de synchroniser les horloges. Les impulsions ainsi communiquées ont la particularité de fournir un signal créneau, qui sera utile dans notre cas pour établir un rythme de communication uniforme.
Finalement, la broche *HALF* permet de séquencer l'activation des bobines en pas ou en demi-pas, de telle sorte que le changement de direction du champ magnétique résultant soit plus ou moins progressif. Remarquons que ceci permettrait une plus grande précision sur la position du rotor, mais n'ayant pas usage d'un tel degré de précision, nous ne l'utiliserons pas ici.

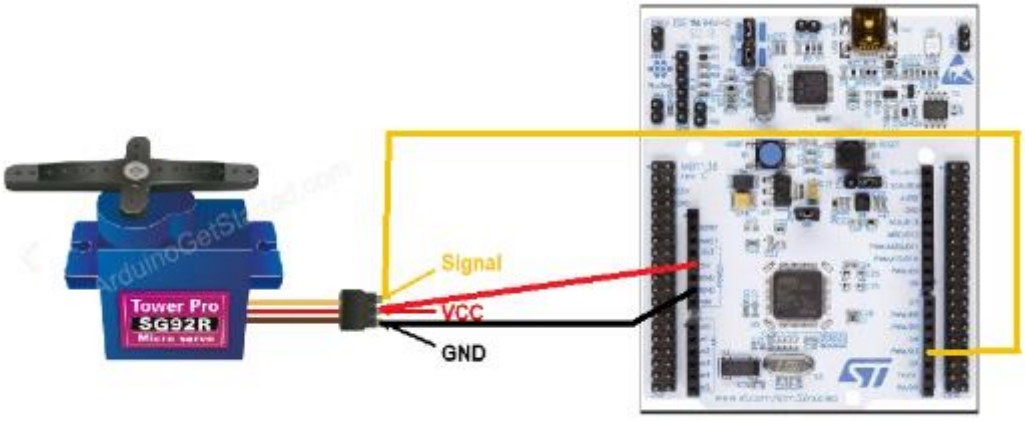
Ayant largement expliqué le principe de fonctionnement et de contrôle des moteurs pas-à-pas (en omettant pour l'instant les programmes informatiques employés), on complète notre étude du fonctionnement théorique via le paragraphe suivant, portant sur les servomoteurs. Il ne reste plus qu'à détailler les programmes codés, afin de comprendre pleinement la façon dont on peut exploiter ces moteurs.

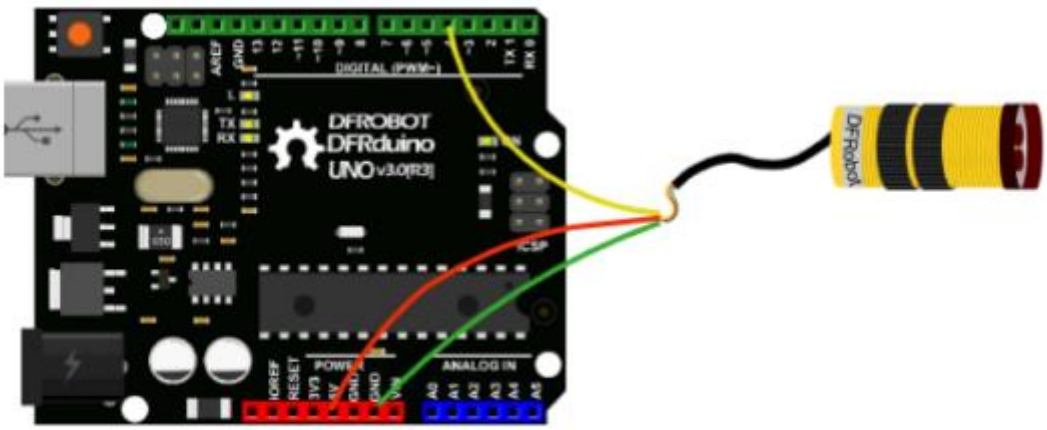
- Afin de pouvoir pousser les objets hors du tapis, notre système dispose de plusieurs bras qui sont motorisés à l'aide de servomoteurs. Ces derniers ont pour but de maintenir une position (angulaire en l'occurrence), suivant la commande analogique leur étant imposée par l'utilisateur. De fait, ils ne peuvent fonctionner que dans une plage de valeurs limitée, et doivent impérativement recouvrer une position antérieure si l'on souhaite les faire fonctionner sur commande. Concrètement, on utilisera ces servomoteurs en les déployant selon leur élongation maximale pour faire tomber une pièce, puis on les fera se rétracter entièrement avant l'arrivée d'une seconde pièce, en vue d'une prochaine utilisation. Les précautions mises en place pour éviter ce chevauchement de commandes seront explicitées lorsque nous détaillerons le programme informatique produit. Ces composants électroniques sont constitués d'un réducteur de vitesse, ainsi que d'un moteur à courant continu, sur

lequel nous ne reviendrons pas, lui-même alimenté par une alimentation de 5V via la carte Nucléo, et sera commandé par une broche de type *PWM*.



La commande de position de ce type de moteurs est donnée par la valeur moyenne du signal et par l'écart en ms entre les créneaux. Ainsi, une période de 1000 ms correspond à la position angulaire initiale (nulle), alors qu'une période de 2500 ms correspond approximativement à la position angulaire pour laquelle le bras du vérin est totalement déployé, comme imagé ci-contre.





```
#include "mbed.h"
#define T_CLK 3


UnbufferedSerial      my_pc(USBTX, USBRX);

char data_piston;

void ISR_my_pc_reception(void);

DigitalOut half (D6);
DigitalOut CW (D7);
PwmOut  CLK(D11);
DigitalOut Enable (D4);


PwmOut myservo1(D3);
PwmOut myservo2(D9);
PwmOut myservo3(D5);

DigitalIn capt(D2);
```

Explication du code

Dans un premier temps nous devons Déclarer toutes les sorties, il est important de déclarer les sorties du servo moteur et de la clock comme étant des PMW car ces deux fonctions nécessitent l'existence d'un signal créneau.

Nous utilisons aussi la fonction unbufferedSerial qui permettra de recevoir des données depuis un programme de traitement sous python.

La commande `void ISR_my_pc_reception void` permet d'initier la communication entre la carte et le programme du PC.

Les trois servo-moteurs sont aussi déclarés ils serviront au tri des pièces.

```
int main(){
  half = 0;
  CW = 1;
  CLK.period_ms(T_CLK);
  CLK.write(0.5);
  Enable =1 ;
  {
    myservo1.period_ms(20);
    myservo2.period_ms(20);
    myservo3.period_ms(20);
    my_pc.baud(115200);
    my_pc.attach(&ISR_my_pc_reception, UnbufferedSerial::RxIrq);
    while (true){}
}

}
```

Dans notre cas l'utilisation de l'option half cité plus haut n'est pas intéressante, c'est pour cela que le niveau logique de cette broche sera maintenu à 0. Nous avons choisi de faire tourner le moteur dans le sens antihoraire donc la broche CW sera elle aussi à 0. Finalement un rapport temps haut sur temps bas de 0.5 est donné pour la clock.

Enable est mise à 1 elle permet de faire tourner le moteur pas à pas.

Les commandes `myservo.period_ms(20)` déterminent une période de référence à partir de laquelle les servo moteurs seront pilotés.

Enfin l'utilisation de my _pc.attach initialise l'attente d'une variable qui proviendra du PC.

```
void ISR_my_pc_reception(void){
    my_pc.read(&data_piston, 1);      // get the received byte



    if(data_piston == '1'){ // echo of the byte received
        if (capt==1){
            wait(1);
        myservo1.pulsewidth_us(2500);
            wait_us (1500000);
        myservo1.pulsewidth_us(1000);
    }
```

Enfin ce dernier module permet de lire la consigne envoyée par le PC avec la commande : my_pc.read.

En fonction de ce que reçoit la carte, depuis l'ordinateur soit 1.2 ou 3, le bras du servo moteur auquel correspond le numéro est déployé.

Pour ce faire c'est la fonction pulsewidth qui permet de changer la taille de la période.

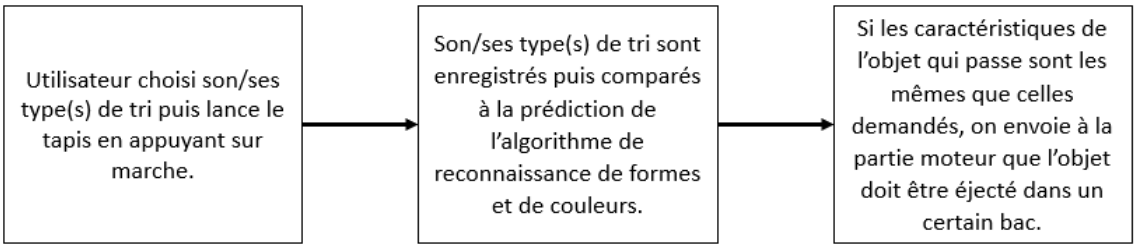
II. Interface

Dans cette partie nous allons décrire le travail d'une partie de l'équipe consistant à créer une interface graphique pour contrôler le système et à écrire le code la mettant en relation avec les différents actionneurs. Pour cela nous pouvons distinguer différentes étapes dans l'avancement de ce travail qui nous serviront de plan.

1. Fonctionnement global et chaîne de prise de décisions.
2. Interface graphique QWidget.
3. Code reliant l'interface graphique au reste du programme.
4. Communication avec la partie mécanique du projet.

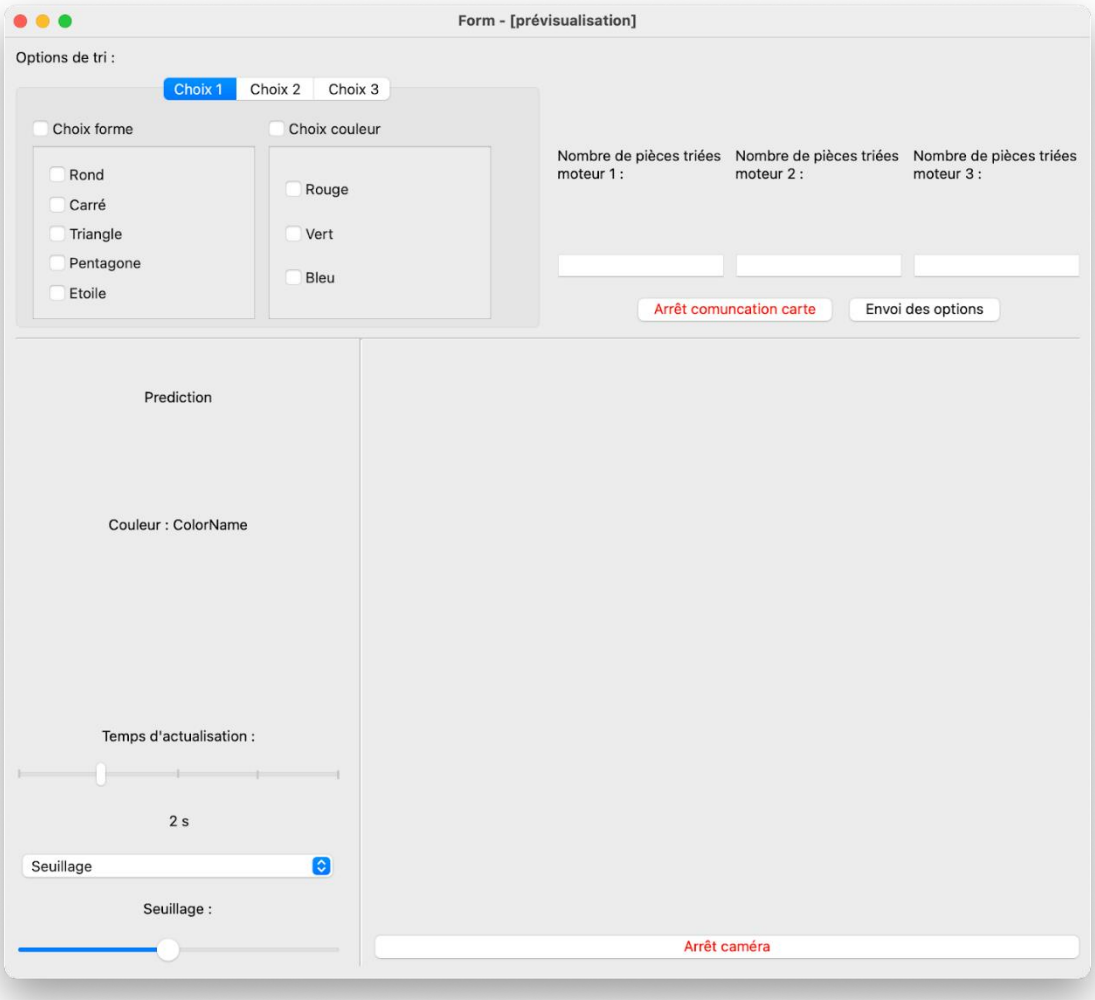
1. Fonctionnement global et chaîne de prise de décisions.

Pour créer une interface avec un sens, il nous a d'abord fallu imaginer le squelette du déroulement des actions internes. Ainsi nous nous sommes d'abord fixés de trier sur trois couleurs possibles et trois formes possibles, et ce sous le format une seule forme et/ou une seule couleur. Cependant, nous avions trois boîtes avec trois vérins/servomoteur donc nous avons donné trois choix à notre utilisateur, une fois encore seulement le premier étant obligatoire. La chaîne de prise de décision initiale était donc la suivante :



2. Interface graphique QWidget

Pour la création de l'interface nous avons utilisé le logiciel Qt car il est intuitif et permet d'intégrer l'interface au code plus facilement. Nous avons ensuite fait évoluer plusieurs générations d'interfaces pour obtenir la suivante :



3. Code reliant l'interface graphique au reste du programme.

Pour enregistrer et comparer les caractéristiques demandées et détectées nous les avons mises sous la forme de matrices : [[bleu : 1 ou 0, rouge : 1 ou 0, jaune : 1 ou 0],[rond : 1 ou 0, carré : 1 ou 0, triangle : 1 ou 0, pentagone : 1 ou 0, étoile : 1 ou 0]]. Le code d'enregistrement est le suivant :

```
def marcheCl(self):
    self.Worker2.tab1=[[0,0,0],[0,0,0,0,0]]
    if self.forme1.isChecked():
        if self.rond1.isChecked():
                self.Worker2.tab1[1][0]=1
        if self.carre1.isChecked():
                self.Worker2.tab1[1][1]=1
        if self.triangle1.isChecked():
                self.Worker2.tab1[1][2]=1
        if self.pentagone1.isChecked():
                self.Worker2.tab1[1][3]=1
        if self.etoile1.isChecked():
                self.Worker2.tab1[1][4]=1
    if self.couleur1.isChecked():
        if self.rouge1.isChecked():
                self.Worker2.tab1[0][0]=1
        if self.vert1.isChecked():
                self.Worker2.tab1[0][1]=1
        if self.bleu1.isChecked():
                self.Worker2.tab1[0][2]=1

    self.Worker2.tab2=[[0,0,0],[0,0,0,0,0]]
    if self.forme2.isChecked():
```

4. Communication avec la partie mécanique du projet.

La communication avec la partie mécanique du projet a été un vrai challenge car celle-ci est codée et C embarqué tandis que tout le reste du programme est codé en python. Ainsi, nous avons dû utiliser une liaison série pour transmettre les informations. Les lignes de code permettant d'envoyer des informations depuis le code python sont les suivantes :

```
self.serNuc.write(bytes(str(1), 'ascii'))#commande moteur1

while self.serNuc.inWaiting() == 0:
    pass
data_rec = self.serNuc.read(4)  # bytes
print(str(data_rec))
```

Voilà comment nous les avons intégrés :

```
def detection(self):

    print(self.tab1)
    print(self.tab)




    if self.tab1[0]!=[0,0,0] or self.tab1[1]!=[0,0,0,0,0]:
        if self.tab1[1]==[0,0,0,0,0]:
            if self.tab1[0][self.tab[0]-1]==1:#/!\-1?
                self.serNuc.write(bytes(str(1), 'ascii'))#commande moteur1

                while self.serNuc.inWaiting() == 0:
                    pass
                data_rec = self.serNuc.read(4)  # bytes
                print(str(data_rec))
                self.cp1 += 1#compteur de piece
                #self.nbpiece1.setText(str(self.cp1))
        if self.tab1[0]==[0,0,0]:
            if self.tab1[1][self.tab[1]]==1:#/!\-1?
                self.serNuc.write(bytes(str(1), 'ascii'))#commande moteur1

                while self.serNuc.inWaiting() == 0:
                    pass
                data_rec = self.serNuc.read(4)  # bytes
                print(str(data_rec))
                self.cp1 += 1#compteur de piece
                #self.nbpiece1.setText(str(self.cp1))
        else:
            if self.tab1[0][self.tab[0]-1]==1:#-1?
                if self.tab1[1][self.tab[1]]==1:
                    self.serNuc.write(bytes(str(1), 'ascii'))#commande moteur1
                    while self.serNuc.inWaiting() == 0:
                        pass
                    data_rec = self.serNuc.read(4)  # bytes
                    print(str(data_rec))
                    self.cp1 += 1#compteur de piece
                    #self.nbpiece1.setText(str(self.cp1))
```

III. Détection

Pour la reconnaissance de forme, nous avons traité le problème avec une intelligence artificielle utilisant un réseau de neurones, un perceptron multicouche (MLP Multi-Layer Perceptron). Il est donc nécessaire de découper le travail en deux parties, la première est consacré à la conception et l'entrainement du modèle, la seconde consiste à la mise en place et l'intégration du modèle d'intelligence artificielle.

Conception du modèle :

Commençons par un petit calcul d'ordre de grandeur : le réseau de neurones qui est utilisé ici et, dont le fonctionnement sera détallé par la suite, contient 256 neurones dans sa première couche, et, chacun de ces neurones est connecté à chaque pixel de l'image d'entré. Ainsi puisque dans ce modèle le nombre de paramètre à entrainer par neurone est proportionnel au nombre d'entré, pour une image provenant de la caméra c'est-à-dire en haute définition de $1280 \times 720\ pixels$ alors le nombre de paramètres entraîner pour un réseau de 256 neurones est : $w_{tot} \sim 1280 \times 720 \times 256 \approx 2 \cdot 10^8$. Un autre bon ordre de grandeur quant au nombre d'image pour entrainer l'algorithme dans notre cas, est de au moins avoir le même ordre de grandeur de nombre d'images que de nombre de paramètres. Nous atteignons ainsi rapidement des quantités en pratiques irréalisables.

C'est pour cette raison que nous avons choisi de prétraiter les images en leur appliquant un certain nombre d'opération afin de simplifier au maximum la tâche pour le réseau de neurones. En effet si un humain peut clairement distinguer différentes formes sans pour autant vérifier qu'un carré contient 4 coins, un réseau de neurones lui, doit extraire d'une image les caractéristiques importantes (features) de l'objet d'intérêt afin de pouvoir le distinguer avec précision.

Voici le schéma du traitement appliqué à une image provenant de la caméra :

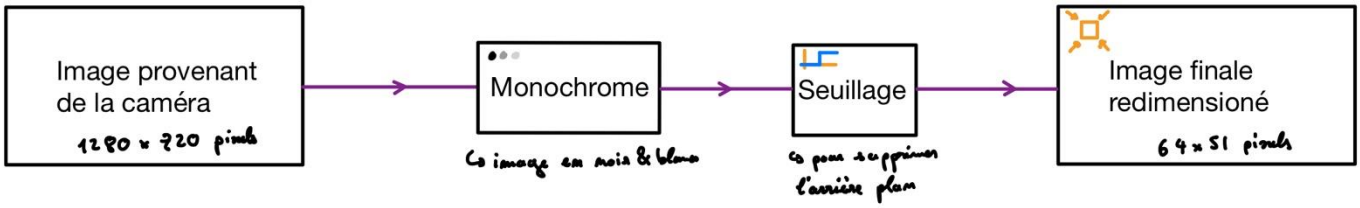


Figure 1 : schéma de transformation de l'image.

Finalement l'image finale est donc composée de $3264\ pixels$, est en nuances de gris, et comporte tous ses pixels de l'arrière-plan avec une valeur de pixel de 0 sur 255.
L'image peut donc être transformé en vecteur numpy :

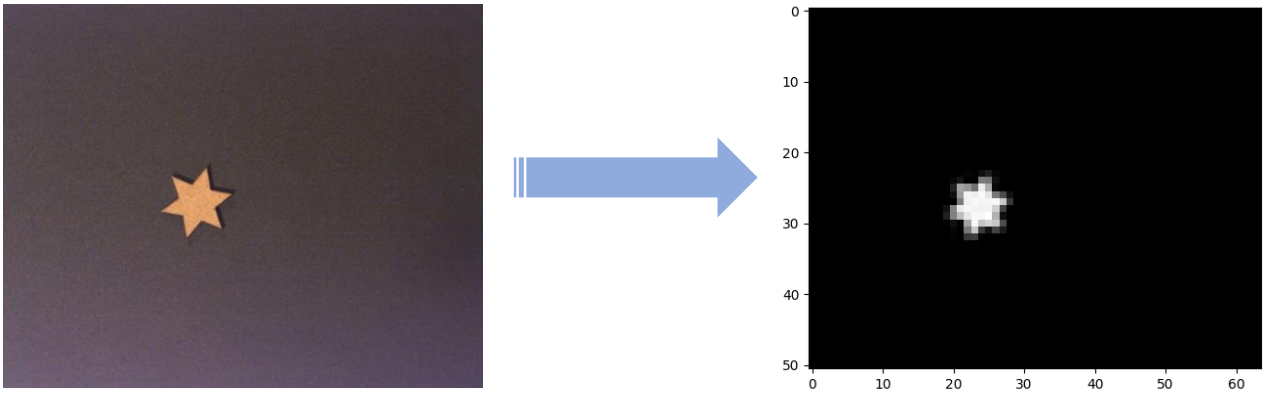


Figure 2 : Image originale

Figure 3 : Image finale

Ainsi le nombre de paramètres du réseau de neurones est de l'ordre de $w_{tot} \sim 256 \times 3264 \approx 8 \cdot 10^5$. Dans notre cas nous allons donc utiliser une base de données de $91\,000$ images.

Il est clair que prendre un tel nombre d'image peut être relativement long, nous avons donc opté pour une génération artificielle de cette base de données. Pour ce faire nous utilisons donc une banque d'image réellement prise avec la caméra puis traité, ici 10 images réelles par formes soit 50 au total (carré, rond, triangle, pentagone, étoile).

Pour créer des images supplémentaires pour l'entrainement de l'algorithme, chaque image originale permet d'engendrer $1\,750$ nouvelles images. Afin de générer ces images, l'image originale effectue une rotation d'angle aléatoire, puis une translation elle aussi aléatoire. L'intérêt ici est de faire en sorte de générer d'autres situation ou l'objet est placé sur le tapis de façon différente. En conséquence pour chaque forme un fichier .csv sera créé avec la bibliothèque Panda pour stocker les 17500 images associés. L'avantage est donc par la suite de titrer (labelliser) les images de chaque forme, mais aussi d'avoir des variables python rapides et adapté à la manipulation de fichiers de grande taille afin de ne pas utiliser toute la mémoire RAM de l'ordinateur. De surcroit, 3500 images « vides » (tableaux numpy remplis de 0) sont ajouté dans le but de pouvoir aussi reconnaitre lorsqu'aucun objet n'est présent à la vue de la caméra.
Par la suite des bases de données de test de 500 images chacune serons également ajoutés dans le code de l'entrainement de l'algorithme. Cette base de test permet d'éviter une situation d'overfitting.

Débute désormais la partie concernant le modèle d'intelligence artificielle, dans tout ce projet l'architecture du MLP (Multi-Layer Perceptron) sera réalisé à l'aide de la bibliothèque scikit-learn, et plus précisément de la bibliothèque MLPClassifier[1].
Voici donc un résumé du fonctionnement d'un réseau de neurones multicouche[2] :

[1] La documentation en ligne est disponible à l'adresse suivante : https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier
[2] Une explication plus détaillé est disponible sur le site de sklearn : https://scikit-learn.org/stable/modules/neural_networks_supervised.html

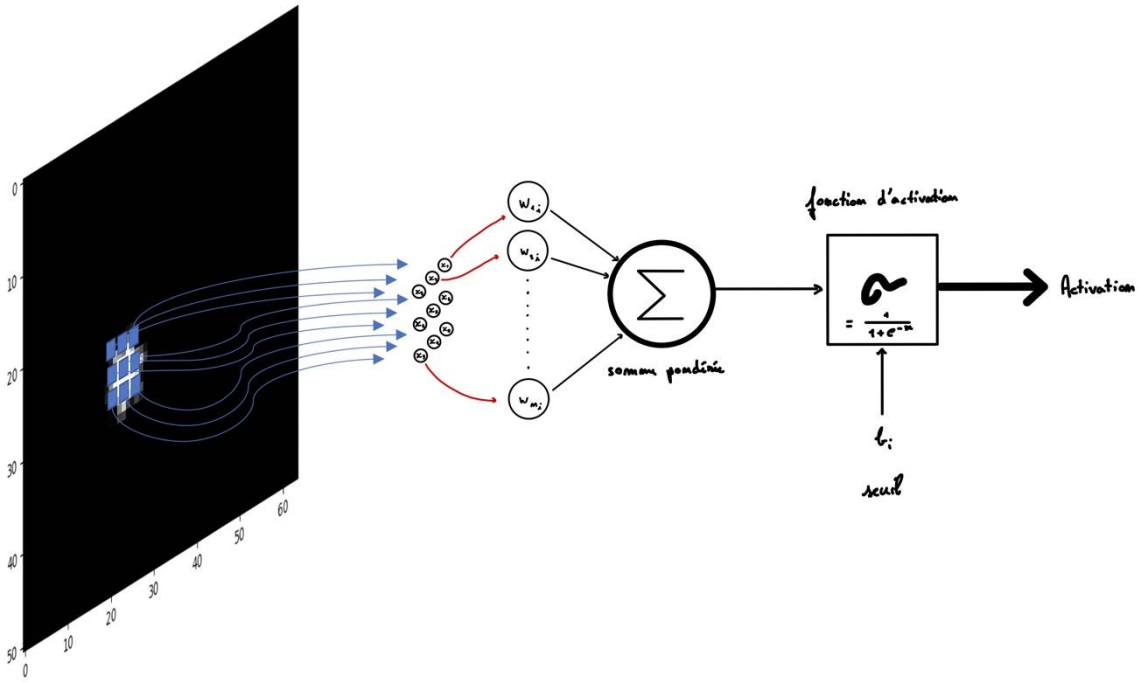


Figure 4 : schéma d'un perceptron

- Entrées :
  Chaque perceptron reçoit un ensemble de valeurs d'entrée. Chaque valeur d'entrée est associée à un poids qui représente l'importance relative de cette entrée par rapport à la sortie du perceptron.

- Somme pondérée :
  Le perceptron n°i effectue d'abord une somme pondérée des valeurs d'entrée en multipliant chaque valeur d'entrée par son poids correspondant, puis en additionnant les produits pondérés. Ceci peut être représenté par la formule suivante :

$$\sum_{j=1}^{n} w_{i,j} \times x_j$$

- Fonction d'activation : Une fois que la somme pondérée est calculée, il est ajouté un biais : b_i la somme devient donc :

$$X_i = b_i + \sum_{j=1}^{n} w_{i,j} \times x_j$$

  Elle est ensuite passée à une fonction d'activation, ici la fonction sigmoïde, pour donner une sortie Y_i :

$$Y_i = \sigma(X_i) = \frac{1}{1 + e^{-X_i}}$$

La fonction d'activation introduit une non-linéarité dans le modèle et permet au perceptron d'apprendre des relations complexes entre les entrées et la sortie.
La sortie du perceptron est donc le résultat de la fonction d'activation appliquée à la somme pondérée. Cette sortie peut être utilisée comme entrée pour le perceptron suivant dans le réseau, ou comme sortie finale si le perceptron se trouve dans la dernière couche du MLP.

Finalement différents perceptrons sont associés pour compléter le modèle, il est important de noter que le calcul des fonctions précédentes sont par la suite décrites sous la forme de multiplications matricielles, cela permet de simplifier, de compacter et de les généraliser les équations.

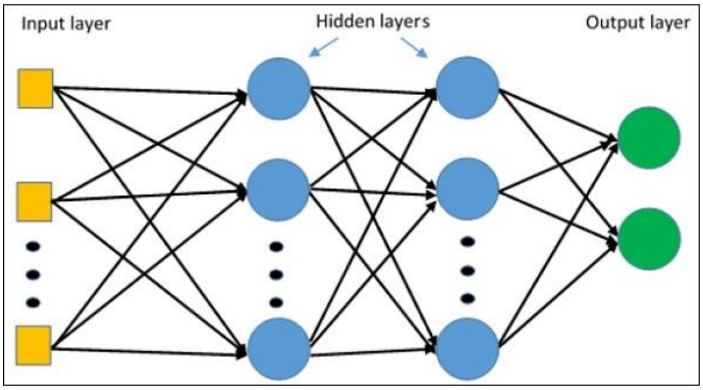


Figure 5 : schéma du MLP (source : www.tutorialspoint.com)

Lors de la conception de ce réseau de neurones, la tailles des différentes couches sont les suivantes :

- Input layer : $64 \times 51 = 3264$ (taille de l'image traité)
- 1ère couche cachée (hidden layer) : 256
- 2nd couche cachée : 128
- Output layer : 6 (une sortie pour chaque forme plus une pour une image vide)

La méthode que nous avons utilisée pour entrainer le MLP est la méthode de descente du gradient. Son objectif est d'ajuster les poids des connexions entre les perceptrons du réseau de manière à minimiser la fonction de perte, qui mesure l'écart entre les sorties réelles du réseau et les sorties attendues.
Une façon simplifié[3] de représenter l'actualisation de paramètres est la formule suivante :

$$w \leftarrow w - \alpha \frac{\partial R(w)}{\partial w}$$

Avec $R(w)$ la fonction de perte (ici l'entropie croisé). Cette formule fait apparaitre le paramètre critique de l'entrainement du MLP, α le learning rate.
Le learning rate contrôle l'amplitude des ajustements effectués sur les poids du réseau à chaque étape de la mise à jour des poids lors de la descente du gradient. Il détermine la vitesse à laquelle le modèle converge vers une solution optimale.
Un learning rate trop élevé peut conduire à des oscillations ou à des divergences, où les ajustements des poids sont si importants qu'ils dépassent le minimum global de la fonction de perte. D'un autre côté, un learning rate trop faible peut ralentir la convergence du modèle, nécessitant plus d'itérations pour atteindre une solution optimale.

Dans ce projet il a donc été nécessaire de d'ajuster cet hyperparamètre à tâtons pour essayer de ne pas tomber dans aucun de ces problèmes.

[3] Pour plus de détail consulter le site de sklearn : https://scikit-learn.org/stable/modules/sgd.html#id5

Finalement, voici l'ensemble des hyperparamètres utilisés dans le MLP :

```
self.__mlp2=MLPClassifier(hidden_layer_sizes=([256,128]), activation='logistic', alpha=1e-4, solver='sgd', tol=5e-3, random_state=None,
    verbose=False,max_iter=1,warm_start=False,learning_rate_init=0.005)
```

Voici donc le résultat final du meilleur entrainement obtenu[4] :

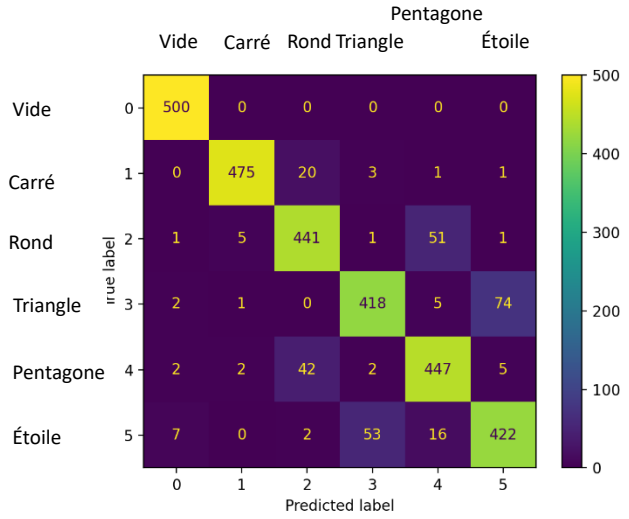


Figure 7 : Matrice de confusion (images de test)

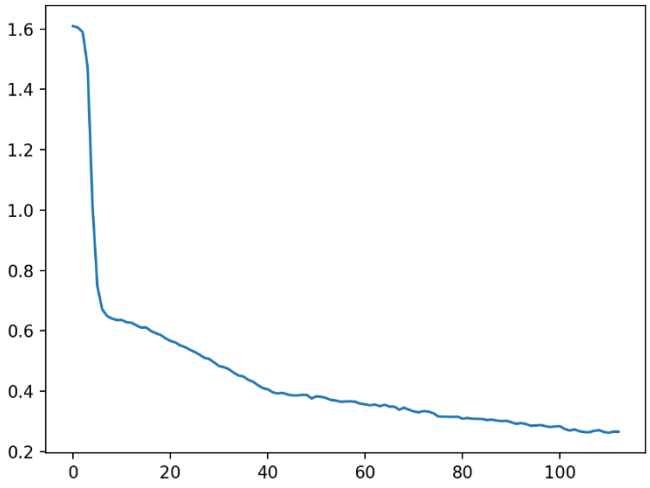


Figure 6 : Training loss

$$\textit{Training set score: } 0.900495 \quad \| \| \quad \textit{Test set score: } 0.901000$$

Pour conclure cette partie, on peut noter que les résultats obtenus après l'entrainement du réseau de neurone donnent des résultats précis et fiables sur les échantillons d'entrainement et de test.

Reconnaissance de couleurs

La reconnaissance de couleurs elle est basé sur une méthode plus simple, dans l'image provenant de la caméra, le triplet de nuance RGB le plus clair de l'image est comparé à un ensemble de 132 autres triplets de couleurs de références provenant des couleurs disponibles en CSS (disponible sur python avec la bibliothèque webcolors). La comparaison est effectuée au sens des moindres carré, en conséquence la couleur la plus proche de la base de référence permet de donner un nom à la couleur effectivement observé.

[4] $\sim 1h$ de calcul (~ 100 itérations) avec un MacBook Pro : Possesseur : 2,3 GHz Intel Core i9 8 cœurs, Carte graphique : Intel UHD Graphics 630 1536 Mo, RAM :16 Go 2667 MHz DDR4

Intégration :

Les algorithme de reconnaissance de forme et de couleurs sont regrouper en une classe python qui permet de donner une prédiction avec une image de la caméra toutes les τ secondes (ce taux de rafraichissement est choisi par l'utilisateur).

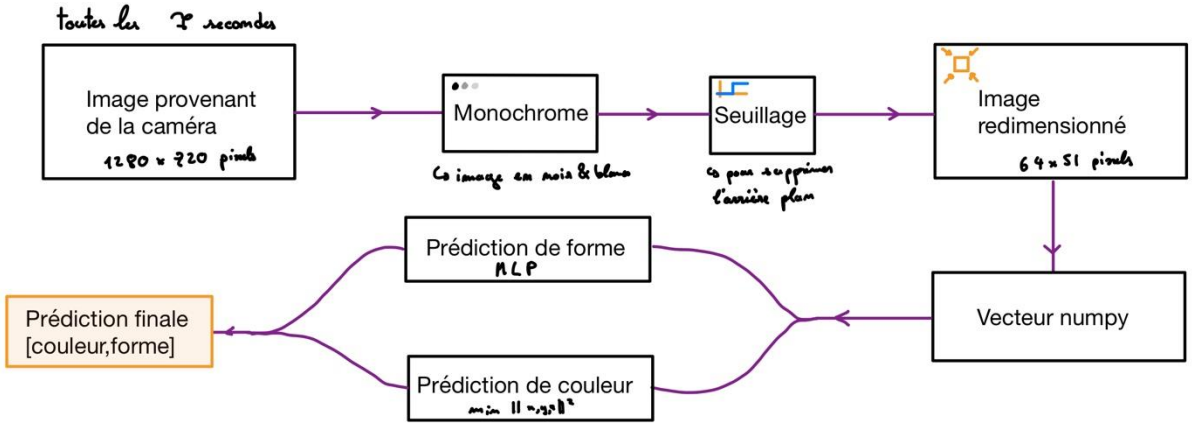


Figure 8 : traitement en continu des images de la caméra

Pour encore améliorer la précision des estimations, et pour profiter du fait que lorsqu'un objet défile devant la caméra, ce dernier est visible sur plusieurs images. L'ensembles des images dans lesquelles l'objet est visible sont donc traité par l'algorithme, et la prédiction qui est le plus souvent revenue sera envoyé à la carte nucléo pour le tri.

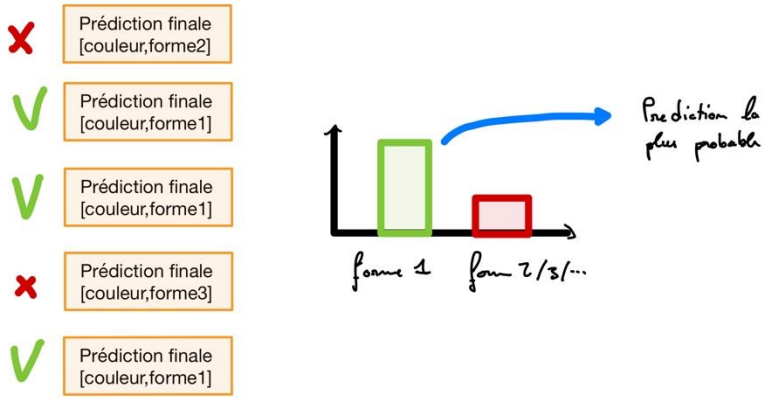


Figure 9 : processus de décision amélioré

IV. Conclusion

Possibilités d'amélioration

- Compteur d'éléments qui sont repérés
- Dissocier plus de couleurs et de formes
- Interface qui commande la mise en route du tapis
- Plus de précision en commandant totalement les moteurs pas-à-pas
- Entrer le port de communication dans l'interface et qu'elle permette même d'aller le chercher

Planning

- Séance 1 : Description du sujet et création du cahier des charges.
- Séance 2 : Répartition du travail et prise en main des différents outils.
- Séance 3 et 4 : Avancement de chaque partie indépendamment.
- Séance 5 et 6 : Travail de liaison entre les parties.
- Séance 7 : Mise en fonction du système.

Répartition des tâches

Pôle Moteur	**Pôle Interface**	**Pôle Détection**
Adrien Charley Khalid	Jade Maurice	Alexis

Compétences acquises

- ✓ Contrôler et moduler le fonctionnement de moteurs pas-à-pas et de servomoteurs.
- ✓ Utiliser QWidget pour créer une interface.
- ✓ Etablir une connexion série entre un programme Python et une carte Nucléo.
- ✓ Mettre en place un système de réseau neuronal.